Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Кафедра прикладной математики и информатики

Отчет по лабораторной работе №1 на тему

"RSA"

по дисциплине

"Дискретная математика"

Выполнил студент гр. 5030102/20201 Фатахов Т.М.

Санкт-Петербург

2024

1 Задание

Цель лабораторной работы — реализовать алгоритм шифрования с открытым ключом RSA, который включает следующие этапы:

- Генерацию ключей — открытого (пара чисел (n, e)) и закрытого (число d);
- Шифрование сообщения с использованием открытого ключа;
- Расшифрование сообщения с использованием закрытого ключа.

Реализация должна использовать случайные простые числа длиной не менее 1024 бит. Ключи сохраняются в отдельных файлах, поддерживается работа с большими числами. В дополнение к требованиям реализованы методы генерации простых чисел, возведения в степень по модулю, теста Миллера-Рабина для проверки простоты, а также алгоритмы Евклида и расширенного Евклида для нахождения обратного по модулю.

2 Язык программирования

Реализация выполнена на языке программирования Python (версия 3.10). Для работы с большими числами использовалась библиотека gmpy2 (версия 2.2.1).

3 Описание алгоритмов

3.1 Генерация ключей RSA

Генерация ключей состоит из следующих шагов:

1. Выбираются два случайных простых числа p и q.
2. Вычисляется $n = p \times q$, первая часть открытого ключа.
3. Вычисляется $\varphi(n) = (p - 1)(q - 1)$.
4. Определяется открытая экспонента e, которая является стандартной константой, равной 65537.
5. Вычисляется закрытая экспонента d как обратное к e по модулю $\varphi(n)$ с использованием расширенного алгоритма Евклида.

3.2 Шифрование

Для шифрования сообщения M (в данном случае, строки, преобразованной в большое число):

$$C = M^e \mod n$$

Результат C сохраняется в файл.

3.3 Расшифрование

Процесс расшифрования использует частное значение d:

$$M = C^d \mod n$$

Затем результат переводится обратно в строку.

3.4 Алгоритм быстрого возведения в степень по модулю (power_mod)

Алгоритм быстрого возведения в степень используется для эффективного вычисления выражения $a^b \mod m$. Он основан на разложении степени на биты, что позволяет выполнять операции умножения только тогда, когда это необходимо. Ниже представлен псевдокод алгоритма:

function PowerMod(a, b, m)
    $result \leftarrow 1$
    $a \leftarrow a \mod m$
    while $b > 0$ **do**
        if b **mod** $2 = 1$ **then**
            $result \leftarrow (result \cdot a) \mod m$
        end if
        $a \leftarrow (a \cdot a) \mod m$
        $b \leftarrow b \div 2$
    end while
    return $result$
end function

Описание алгоритма:

- Инициализируется переменная `result` значением 1, которая будет хранить результат.
- На каждом шаге проверяется, является ли текущий бит экспоненты нечётным (проверка `b mod 2 = 1`). Если это так, результат умножается на текущее значение `a` и берётся по модулю m.
- Основание `a` возводится в квадрат и берётся по модулю m на каждом шаге.
- Показатель `b` сдвигается вправо (делится на 2) для перехода к следующему биту.
- Процесс повторяется, пока показатель `b` не станет равен 0.

3.5 Алгоритм проверки простоты Миллера-Рабина (miller_rabin_test)

Алгоритм Миллера-Рабина используется для проверки, является ли число n вероятно простым. Он основан на случайном выборе баз a и проверке, нарушают ли они условия простоты. Ниже представлен псевдокод алгоритма:

function MillerRabinTest(n, k)

```
    if n ≤ 1 or n = 4 then
        return False
    end if
    if n ≤ 3 then
        return True
    end if
    d ← n − 1
    r ← 0
    while d mod 2 = 0 do
        d ← d/2
        r ← r + 1
    end while
    for i = 1 to k do
        Выбрать случайное a в диапазоне [2, n − 2]
        x ← PowerMod(a, d, n)
        if x = 1 or x = n − 1 then
            continue
        end if
        for j = 1 to r − 1 do
            x ← PowerMod(x, 2, n)
            if x = n − 1 then
                break
            end if
        end for
        if x ≠ n − 1 then
            return False
        end if
    end for
    return True
end function
```

Описание алгоритма:

- Находится представление числа $n-1$ в виде $2^r \cdot d$, где d нечётное.
- Алгоритм выполняет k раундов, каждый из которых проверяет, не является ли число составным:
  1. Случайное основание a выбирается в диапазоне $[2, n-2]$.
  2. Вычисляется $x = a^d \mod n$.
  3. Если $x = 1$ или $x = n-1$, переход к следующему раунду.
  4. В противном случае, значение x возводится в квадрат $r-1$ раз, проверяя, становится ли оно равным $n-1$. Если ни одно из возведений не даёт $n-1$, число n считается составным.

- Если n прошло все k раундов, оно считается вероятно простым.

3.6 Преобразование строки в число и наоборот

В данной реализации для шифрования текстового сообщения строка сначала преобразуется в целое число, так как криптографические операции выполняются над числовыми данными. Далее представлены алгоритмы преобразования строки в число и обратно.

3.6.1 Преобразование строки в целое число (string_to_integer)

Алгоритм считывает каждый символ строки, преобразует его в ASCII-код и последовательно формирует целое число путём побитового сдвига:

function $\textsc{StringToInteger}(message)$
  $num \leftarrow 0$
  for each $char$ **in** $message$ **do**
    $num \leftarrow (num \ll 8) + \mathrm{ord}(char)$
  end for
  return num
end function

Описание:

- Инициализируется переменная `num`, в которую записывается результат.
- Каждый символ строки конвертируется в целое число с использованием функции `ord`, возвращающей ASCII-код символа.
- Полученное число добавляется к `num` через сдвиг на 8 битов (один байт), позволяя конструировать уникальное представление всей строки.

3.6.2 Преобразование целого числа в строку (integer_to_string)

Алгоритм извлекает каждый байт числа, декодируя его обратно в символы строки:

function $\textsc{IntegerToString}(num)$
  `result` $\leftarrow$ пустой список
  while $num > 0$ **do**
    $char \leftarrow \mathrm{chr}(num \& 0xFF)$
    `result.append`$(char)$
    $num \leftarrow num \gg 8$
  end while
  return `"".join(result[::-1])`
end function

Описание:

- На каждом шаге используется операция `num & 0xFF` для извлечения младших 8 бит (1 байт) числа, что даёт ASCII-код символа.
- Полученный код конвертируется в символ с помощью функции `chr` и добавляется в список `result`.
- После извлечения символа число сдвигается вправо на 8 бит (удаляя младший байт).
- Символы в списке `result` инвертируются (так как были добавлены в обратном порядке) и объединяются в строку, которая возвращается как результат.

Эти два алгоритма обеспечивают обратимое преобразование строки в числовое представление, что позволяет использовать текстовые сообщения в криптографических операциях.

3.7 Обоснование выбора констант в тесте Миллера-Рабина и алгоритме RSA

- **Константа $k = 25$ в тесте Миллера-Рабина**

  Константа $k = 25$ определяет количество раундов вероятностного теста простоты Миллера-Рабина, который используется для проверки числа на простоту. Число раундов k задаёт вероятность ошибки, которая становится пренебрежимо малой при $k = 25$ (менее 2^{-50}). Это значение является балансом между точностью и скоростью проверки, что делает его подходящим для большинства криптографических задач.

- **Константа $e = 65537$ (открытая экспонента в RSA)**

  Значение открытой экспоненты $e = 65537$ выбрано как стандартное для RSA по нескольким причинам:

  – Число 65537 является простым, что важно для обеспечения надёжности вычислений.

  – Значение $65537 = 2^{16}+1$ позволяет эффективно выполнять операции возведения в степень, поскольку данное значение требует меньшего количества умножений.

  – Использование этого значения снижает вероятность ошибок при шифровании и обеспечивает устойчивость алгоритма к ряду криптографических атак.

3.8 Ограничения на входные данные

В данной реализации алгоритма шифрования и расшифрования RSA накладываются следующие ограничения на входные данные, необходимые для корректного выполнения программы:

- **Шифруемое сообщение**: Входное сообщение должно состоять из символов в кодировке UTF-8, так как символы преобразуются в целое число через их ASCII-коды, а для хранения текстовой информации и выполнения математических операций используется 8-битное представление каждого символа. Длина сообщения ограничена числом `n` (модулем), получаемым при умножении простых чисел `p` и `q`, и не должна превышать битовую длину `n`.

4 Демонстрация работы на примере

Рассмотрим генерацию ключей и шифрование небольшого сообщения на тестовых данных. Пусть $p = 3$, $q = 11$, тогда:

- $n = 3 \times 11 = 33$;
- $\varphi(n) = (3-1)(11-1) = 20$;
- $e = 7$, выбранное как взаимно простое с $\varphi(n)$.

Найдем d$7^{-1} \mod 20$ с использованием расширенного алгоритма Евклида.

1. Сначала находим НОД для 7 и 20 с использованием алгоритма Евклида:

$$20 = 2 \cdot 7 + 6$$

$$7 = 1 \cdot 6 + 1$$

$$6 = 6 \cdot 1 + 0$$

Последний ненулевой остаток — это 1, значит, $(7, 20) = 1$. Таким образом, обратный элемент существует.

2. Теперь выражаем 1 как линейную комбинацию 7 и 20:

$$1 = 7 - 1 \cdot (20 - 2 \cdot 7)$$

Упростим выражение:

$$1 = 7 - 1 \cdot 20 + 2 \cdot 7$$

$$1 = 3 \cdot 7 - 1 \cdot 20$$

Получаем:

$$1 = 3 \cdot 7 + (-1) \cdot 20$$

Таким образом, $d = 3$ — это обратный элемент для 7 по модулю 20.

Пусть сообщение $M = 2$. Тогда:

- Шифротекст $C = M^e \mod n = 2^7 \mod 33$. Для вычисления $C = M^e \mod n$, где $M = 2$, $e = 7$, и $n = 33$, используем метод быстрого возведения в степень по модулю.

  Мы хотим найти:

  $$C = 2^7 \mod 33$$

  1. Представим $e = 7$ в двоичном виде:

  $$7_{(10)} = 111_{(2)}$$

2. Распишем 2^7 как последовательность возведений в квадрат и умножений:

- Находим $2^1 \mod 33$:

$$2^1 = 2 \Rightarrow 2 \mod 33 = 2$$

- Находим $2^2 \mod 33$:

$$2^2 = 4 \Rightarrow 4 \mod 33 = 4$$

- Находим $2^4 \mod 33$:

$$2^4 = 16 \Rightarrow 16 \mod 33 = 16$$

- Находим 2^7 как $2^4 \cdot 2^2 \cdot 2^1 \mod 33$:

$$2^7 = 2^4 \cdot 2^2 \cdot 2 = 16 \cdot 4 \cdot 2 = 128 \mod 33$$

3. Применим модуль 33 к результату:

$$128 \mod 33 = 29$$

Таким образом, значение C будет:

$$C = 2^7 \mod 33 = 29$$

- Расшифрованное сообщение $M = C^d \mod n = 29^3 \mod 33$. Для вычисления $M = C^d \mod n$, где $C = 29$, $d = 3$, и $n = 33$, снова используем метод быстрого возведения в степень по модулю. В этом случае нам нужно найти:

$$M = 29^3 \mod 33$$

1. Распишем 29^3 как последовательность возведений в квадрат и умножений.

- Сначала находим $29^1 \mod 33$:

$$29^1 = 29 \Rightarrow 29 \mod 33 = 29$$

- Далее находим $29^2 \mod 33$:

$$29^2 = 841$$

Применим модуль:

$$841 \mod 33 = 16$$

Таким образом, $29^2 \equiv 16 \mod 33$.

- Теперь находим $29^3 \mod 33$ как произведение $29^2 \cdot 29 \mod 33$:

$$29^3 = 29^2 \cdot 29 = 16 \cdot 29 = 464$$

Применим модуль:

$$464 \mod 33 = 2$$

$$M = 29^3 \mod 33 = 2$$

Исходя из расчетов видно, что результат совпадает с исходным символом.

5 Область применения и возможные ошибки

RSA шифрование с открытым ключом широко применяется для обеспечения конфиденциальности передачи данных. Алгоритм надёжен, если выполняются требования к длине ключей (от 2048 бит) и соблюдаются принципы безопасности ключей. Ошибки могут возникнуть в следующих случаях:

- Несоблюдение длины ключа приводит к уязвимости RSA к атакам;
- Неправильная работа с файлами ввода/вывода может привести к некорректной загрузке ключей и данных;
- Использование малых простых чисел, как в данном примере, уязвимо для подбора значений ключа.

6 Формат входных и выходных данных

- **Входные данные**: Файлы с сообщением и ключами.
- **Выходные данные**: Файлы, содержащие зашифрованное и расшифрованное сообщение.
- Открытый ключ хранится в текстовом файле в виде чисел e и n в отдельной строке.
- Закрытый ключ хранится в текстовом файле в виде чисел d и n в отдельной строке.