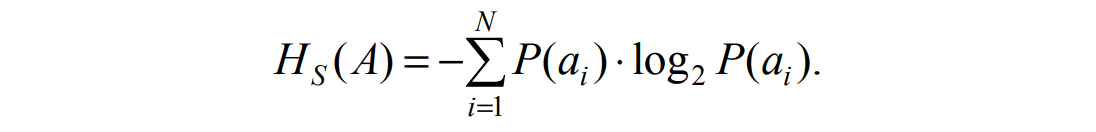
Яшный Никита 4 ПОИТ

Лабораторная работа 1

# Энтропия алфавита

Информационной характеристикой алфавита (источника сообщений на основе этого алфавита) является энтропия. Энтропию алфавита *А =* {*ai*} по К. Шеннону рассчитывают по следующей формуле:

(1.1)

С физической точки зрения энтропия алфавита показывает, какое количество информации приходится в среднем на один символ алфавита.

Для вычисления энтропии в начале необходимо подсчитать количество появлений каждого символа в исходной строке. Для этого используется функция, представленная на рисунке 1.1.

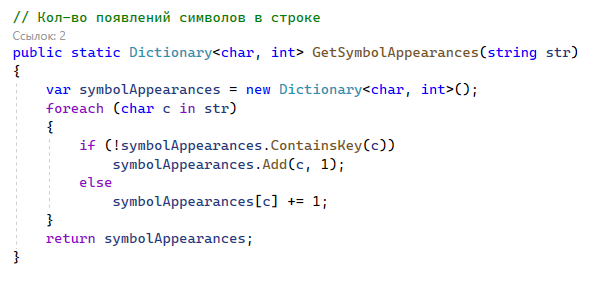


Рисунок 1.1 – Функция подсчёта кол-ва появлений символов

Далее, для вычисления энтропии, необходимо перейти от количества появлений символов к вероятности их появления. Для этого необходимо кол-во появлений каждого символа разделить на кол-во символов в строке. Далее эту вероятность *P*(*ai*)можно подставлять в формулу (1.1). Функция, вычисляющая энтропию Шеннона, представлена на рисунке 1.2.

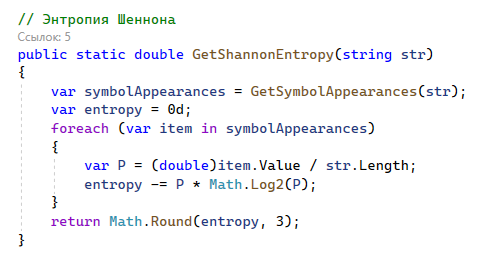


Рисунок 1.2 – Функция вычисления энтропии Шеннона

# Кириллица

В качестве кириллического алфавита был выбран белорусский. Электронный текстовый документ на основе алфавита содержится в файле belarusian.txt, представленном на рисунке 1.3.

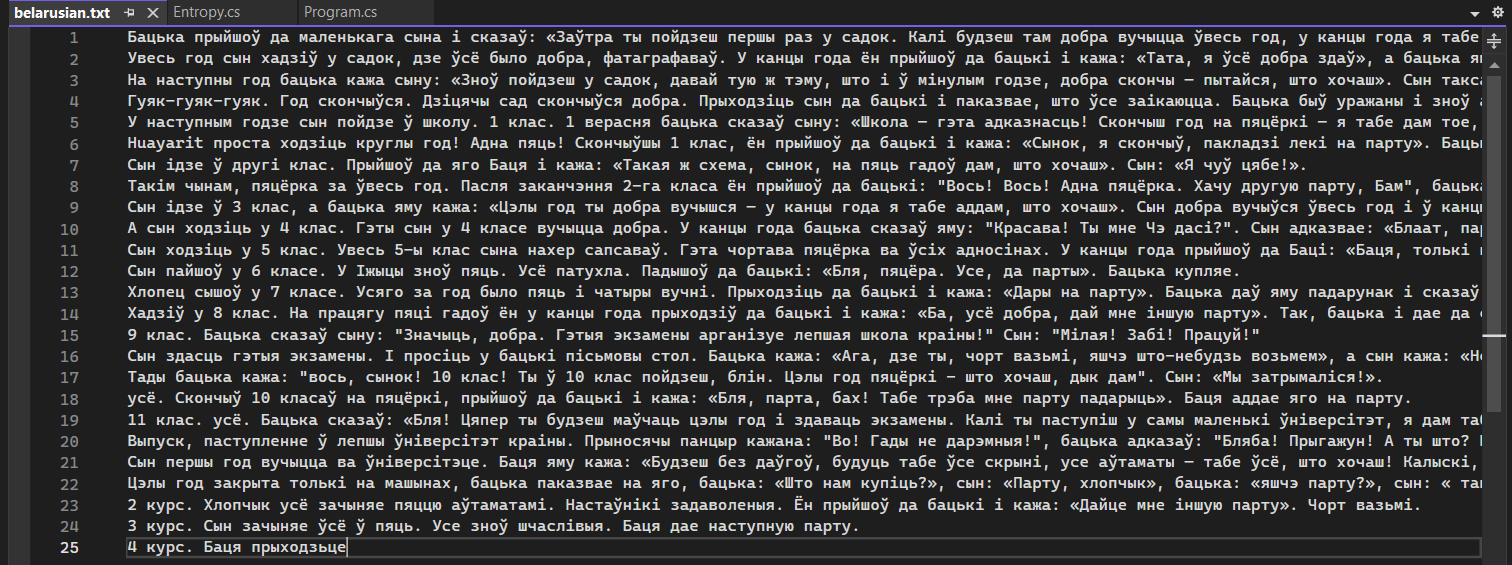


Рисунок 1.3 – Текст на белорусском алфавите

# Латиница

В качестве алфавита на латинице был выбран польский алфавит. Электронный документ, в котором содержится текст на исходном польском алфавите для подсчета энтропии алфавита, содержится в файле polish.txt, представленном на рисунке 1.4.

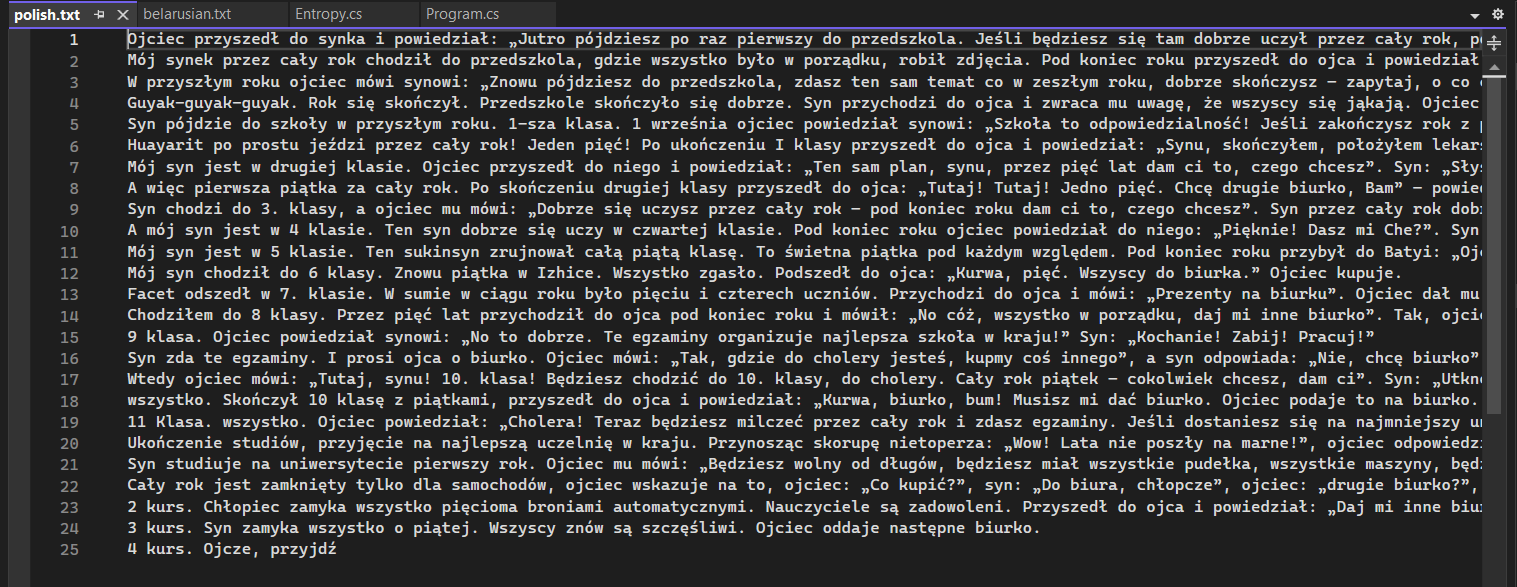


Рисунок 1.4 – Текст на польском алфавите

# Вычисление энтропии алфавита

Графики, отражающие частоты появления символов в тексте на белорусском и польском алфавитах, представлены на рисунке 1.5.

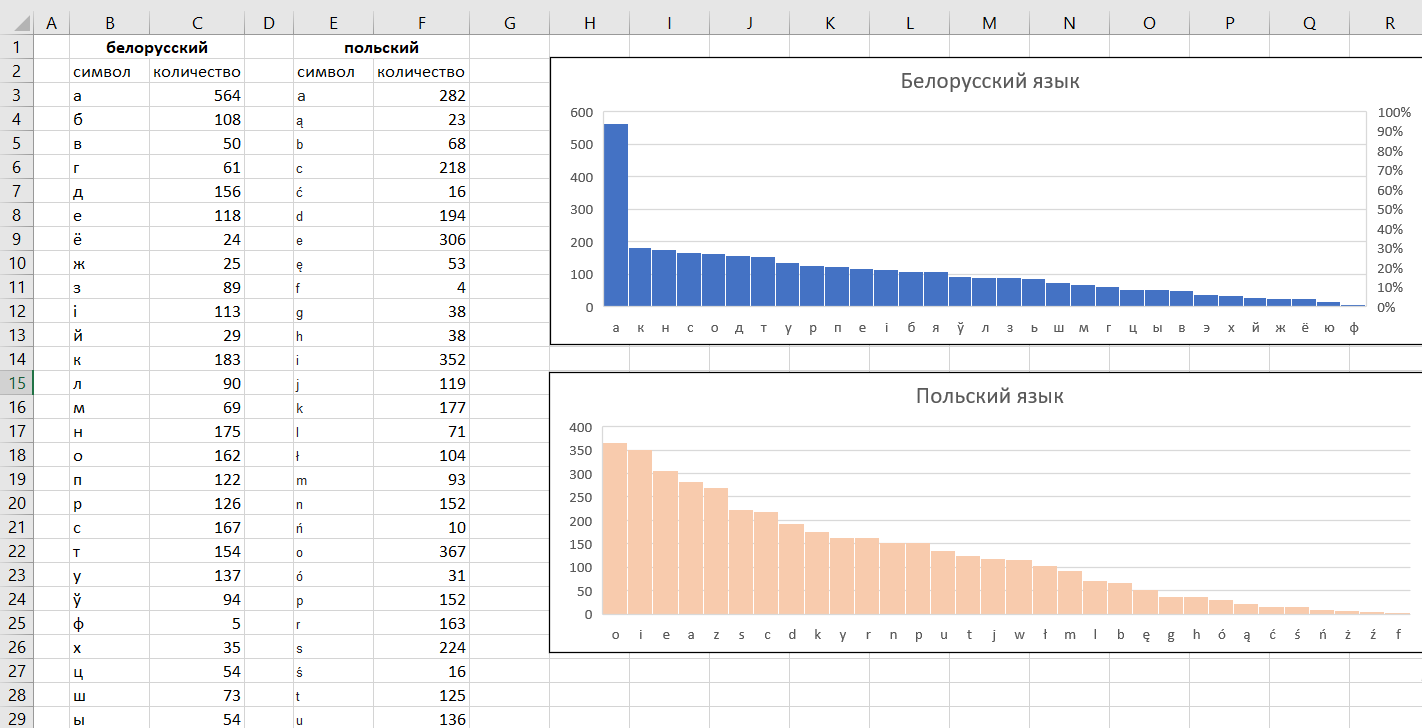


Рисунок 1.5 – Частоты появления символов в алфавитах

Из этих данных можно получить вероятности появления каждого символа *P*(*ai*)*.* При подстановке их в формулу (1.1) получаем следующие значения энтропии для белорусского, польского и бинарного алфавитов, продемонстрированные на рисунке 1.6:

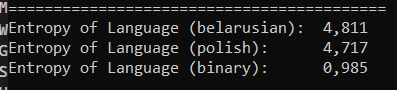


Рисунок 1.6 – Вывод функции подсчёта энтропии

Энтропия белорусского и польского алфавитов равны соответственно 4,811 бит и 4,717 бит. Энтропия бинарного алфавита, которая всегда примерно равна 1 биту, в данном случае равняется 0,985 бит.

# Количество информации

При известной энтропии алфавита количество информации вычисляется по следующей формуле:

(1.2)

То есть, необходимо умножить полученную энтропию на количество символов в сообщении. Программный код вычисления количества информации сообщения представлен на рисунке 2.1.

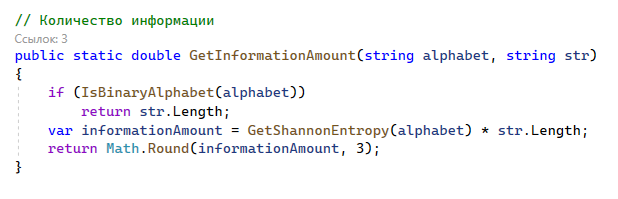


Рисунок 2.1 – Функция количества информации

Если алфавит является бинарным, то за количество информации принимается длина строки, так как энтропия равняется 1. В противном случае, энтропия выбранного алфавита умножается на количество символов в сообщении, количество информации в котором мы желаем найти.

Вывод функции представлен на рисунке 2.2.

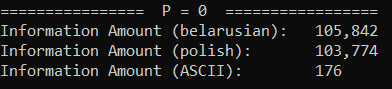


Рисунок 2.2 – Вывод количества информации

Как видно из данного рисунка, количество информации в сообщении, состоящем из ФИО в кодировке ASCII, равняется 176 битам, то есть равняется количеству символов в исходном сообщении, записанном в бинарном виде.

# Эффективная энтропия

Если сообщение может передаться ошибочно, то для вычисления количества информации используется формула эффективной энтропии:

(1.3)

где *H*(*Y | X*) – условная энтропия:

(1.4)

Программный код функции для вычисления эффективной энтропии представлен на рисунке 3.1.

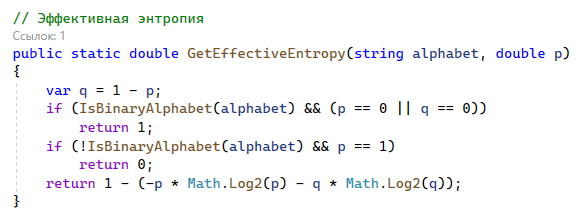


Рисунок 3.1 – Функция эффективной энтропии

В бинарном алфавите эффективная энтропия и, соответственно, количество информации, будет равняться длине строки в случае, если вероятность ошибки равняется 0 или 1. В небинарных же алфавитах, при вероятности ошибки, равной 1, эффективная энтропия и количество информации будет равняться 0.

В таком случае, функция вычисления количества информации станет выглядеть следующим образом:

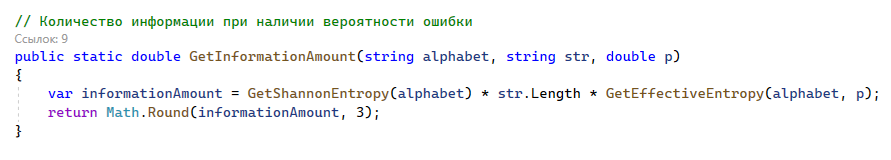


Рисунок 3.2 – Функция кол-ва информации при наличии ошибок

Вывод функции вычисления количества информации при вероятности ошибки, отличной от нулевой, представлен на рисунке 3.3.

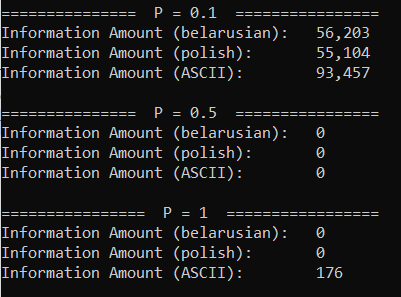


Рисунок 3.3 – Вывод кол-ва информации при *p>0*

**Вывод:** При вероятности ошибки *p=*1 количество информации в бинарном алфавите равняется количеству информации при *p=*0. Это связано с тем, что все биты исходного сообщения инвертируются, то есть заменяются на обратные, поэтому из такого сообщения можно получить исходную информацию. В небинарном же алфавите при *p=*1 определить исходное сообщение не представляется возможным ввиду того, что мощность алфавита больше 2, и каждый символ заменяется на некоторый неизвестный символ из алфавита, поэтому количество информации в таком случае равняется 0.

Ответы на вопросы

**1. Что такое алфавит источника сообщения?**

Алфавит источника сообщения – это конечная или бесконечная совокупность символов и знаков, формирующих сообщение.

**2. Что такое мощность алфавита источника сообщения?**

Мощность алфавита – это количество символов, составляющих алфавит.

**3. Какова мощность алфавита белорусского языка?**

32

**4. Какова мощность алфавита русского языка?**

33

**5. Какова мощность алфавита «компьютерного» языка?**

2

**6. Что такое энтропия алфавита?**

Энтропия алфавита – это информационная характеристика алфавита. Энтропия характеризует количество информации, приходящееся в среднем на один символ алфавита.

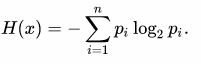
**7. Что такое энтропия сообщения?**

количество информации, приходящееся в среднем на один символ сообщения.

**8. От чего зависит энтропия алфавита?**

Зависит только от распределения вероятностей, а не от алфавита

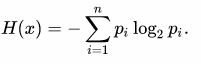
**9. Записать формулу для вычисления энтропии.**



**10. Что нужно знать для вычисления энтропии алфавита?**

Сумма всех элементов и вероятность появления каждого элемента

**11. Как рассчитываются энтропия Шеннона и энтропия Хартли? В чем принципиальное различие между этими характеристиками? Дайте толкование физического смысла энтропии.**

 (шенона)

где pi – вероятность i-ого события.

*, (хартли)*

где N – мощность алфавита.

Энтропия по Хартли рассчитывается с учетом того, что алфавит состоит из символов с равными вероятностями появления. Для расчета энтропии по Шеннону необходимо вычислить отдельно вероятность появления каждого символа алфавита.

**12. Пояснить назначение знака «минус» в формулах (2.1) и (2.4).**

**13. Что такое избыточность алфавита и избыточность сообщений, сформированных в компьютерных системах? Принцип действия каких систем основан на существовании данной избыточности?**

Избыточность – термин, означающий превышение количества информации, используемой для передачи или хранения сообщения, над его информационной энтропией. Возникает, когда энтропия Шеннона и энтропия Хартли не равны. Неполно используется алфавит.

**14. Расположить в порядке возрастания энтропии известные Вам алфавиты.**

**15. Вычислить энтропию алфавита белорусского (русского) языка.**

4,811

**16. Вычислить энтропию Шеннона бинарного алфавита, если вероятность появления в произвольном документе на основе этого алфавита одного из символов составляет 0,25, другого – 0,75; либо 0 и 1,0; либо 0,5 и 0,5.**

H = - (0,25 \* log2(0,25) + 0,75 \* log2(0,75))

H ≈ 0,811 бит.

H = - (0 \* log2(0) + 1,0 \* log2(1,0))

H = 0.

H = - (0,5 \* log2(0,5) + 0,5 \* log2(0,5))

H ≈ 1 бит.

**17. Чему равна энтропия алфавита по Хартли, если мощность этого алфавита равна: а) 1 символ; б) 2 символа; в) 8 символов?**

H = log2(1) = 0.

H = log2(2) = 1.

H = log2(8) = 3.

Лабораторная работа 2

**Ответы на вопросы**

Base64 — это стандартное кодирование, которое используется для преобразования двоичных данных в текстовый формат. Оно широко применяется в компьютерных системах для передачи и хранения данных, которые не могут быть непосредственно представлены в виде текста. Алгоритм base64 кодирует каждые три байта данных (24 бита) в четыре символа ASCII. Таким образом, получается текстовое представление данных, состоящее из символов алфавита (A-Z, a-z), цифр (0-9) и специальных символов (+, /).

Для проверки, была ли определенная строка символов закодирована в base64, вы можете использовать следующий подход:

Проверьте, что длина строки кратна 4.

Проверьте, что все символы строки являются допустимыми символами base64 (A-Z, a-z, 0-9, +, /) или символом заполнения "=".

Если оба условия выполняются, это может указывать на то, что строка была закодирована в base64.

Base64 не предоставляет непосредственных средств для проверки подлинности вводимых данных в форму пароля и логина. Он служит для преобразования данных в текстовый формат и не обеспечивает шифрование или защиту паролей. Для проверки подлинности вводимых данных вам следует рассмотреть использование хэш-функций, шифрования и других методов безопасности.

Энтропия алфавита в форматах данных описывает степень случайности или неопределенности символов в алфавите. Чем выше энтропия, тем больше информации содержится в каждом символе.

Алфавиты в различных форматах данных могут иметь разную энтропию. Например, в двоичном формате (алфавит {0, 1}) энтропия будет максимальной, так как каждый символ может принимать два возможных значения, и каждый символ содержит один бит информации.

В текстовых форматах, таких как ASCII или Unicode, энтропия будет зависеть от количества символов в алфавите и частоты их использования. Некоторые символы могут иметь более высокую энтропию, если они редко встречаются, в то время как другие символы могут иметь более низкую энтропию из-за их частого использования.

Результат операции a XOR b XOR b будет равен a. Операция XOR (исключающее ИЛИ) применяется к каждому биту двух операндов. Если биты совпадают, результат будет 0, а если биты различны, результат будет 1. Когда вы применяете XOR дважды к одному и тому же значению, вы получаете исходное значение.

Операция XOR может использоваться для множества целей, включая шифрование данных, проверку целостности и обеспечение безопасности.

Предоставленные строки после их конвертации в base64 будут выглядеть следующим образом:

efd8b295a633908a3c0828b2 -> ZWZkOGIyOTVhNjMzOTA4YTNjMDgyOGIy

faea8766 -> ZmFlYTg3NjY=

4d72cde3aaa0 -> TWdyY2RlM2FhYTA=

Чтобы найти значение b в операции a XOR b, необходимо выполнить операцию обратного XOR с использованием известных значений a и результата.

Для нахождения значения b в данной операции a XOR b XOR b, необходимо выполнить следующие шаги:

Выполните операцию XOR между a и результатом операции: a XOR (a XOR b XOR b).

Упростите выражение: a XOR a XOR b XOR b.

Примените свойство a XOR a = 0 и b XOR b = 0: 0 XOR 0 XOR b.

Поскольку XOR двух нулей всегда дает 0, выражение упрощается до: 0 XOR b.

Любое значение XOR с 0 дает исходное значение, поэтому результат равен b.

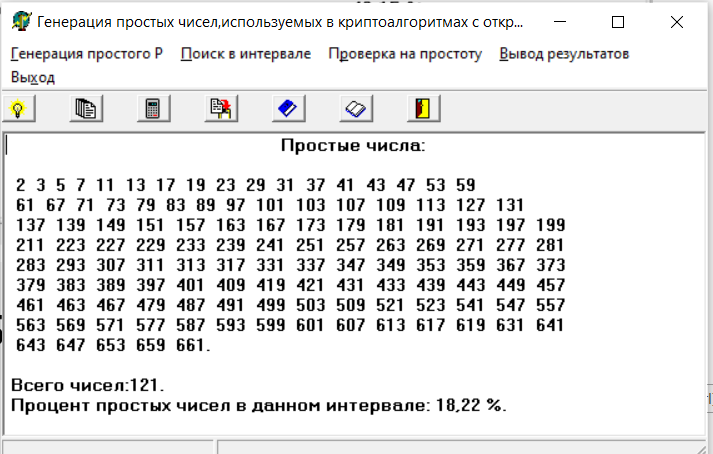
Таким образом, значение b в данной операции a XOR b XOR b будет равно 0.

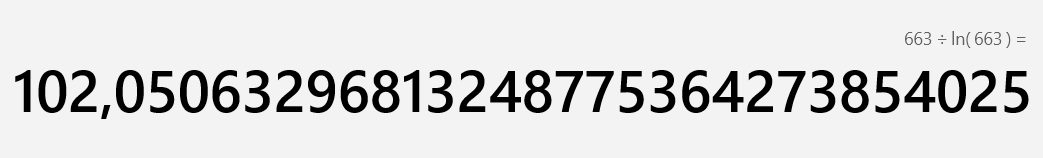
Лабораторная работа 3

**Вариант 14**

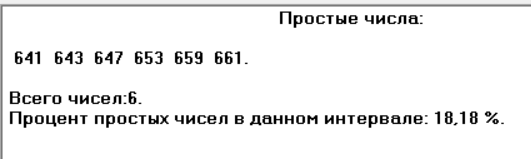
**Первая часть**

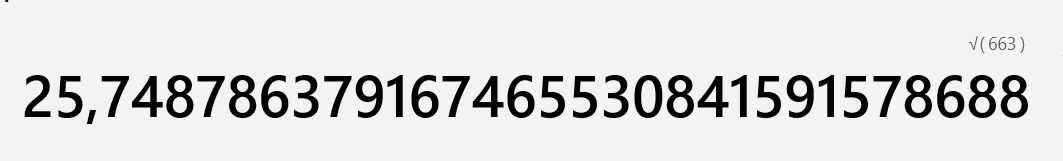
**Задание 1**





**Задание 2**



Проверим вручную через решето эратосфена  


Берём число 26. Простые числа из диапазона 2 – 26: 2, 3, 5, 7, 11, 13, 17, 19, 23.

Все четные делятся на 2, они не подходят.

633 делится на 3

635 делится на 5

637 делится на 7

639 делится на 3

641 простое

643 простое

645 делится на 5

647 простое

649 делится на 11

651 делится на 7

653 простое

655 делится на 5

657 делится на 3

659 простое

661 простое

663 делится на 3

Таким образом простыми являются 641, 643, 647, 653, 659, 661

**Задание 3**  
Число 632 в канонической форме: 2^3 \* 79

Число 633 в канонической форме: 3 \* 211

Число 634 в канонической форме: 2 \* 317

Число 635 в канонической форме: 5 \* 127

Число 636 в канонической форме: 2^2 \* 3 \* 53

Число 637 в канонической форме: 7^2 \* 13

Число 638 в канонической форме: 2 \* 11 \* 29

Число 639 в канонической форме: 3^2 \* 71

Число 640 в канонической форме: 2^7 \* 5

Число 641 в канонической форме: 641

Число 642 в канонической форме: 2 \* 3 \* 107

Число 643 в канонической форме: 643

Число 644 в канонической форме: 2^2 \* 7 \* 23

Число 645 в канонической форме: 3 \* 5 \* 43

Число 646 в канонической форме: 2 \* 17 \* 19

Число 647 в канонической форме: 647

Число 648 в канонической форме: 2^3 \* 3^4

Число 649 в канонической форме: 11 \* 59

Число 650 в канонической форме: 2 \* 5^2 \* 13

Число 651 в канонической форме: 3 \* 7 \* 31

Число 652 в канонической форме: 2^2 \* 163

Число 653 в канонической форме: 653

Число 654 в канонической форме: 2 \* 3 \* 109

Число 655 в канонической форме: 5 \* 131

Число 656 в канонической форме: 2^4 \* 41

Число 657 в канонической форме: 3^2 \* 73

Число 658 в канонической форме: 2 \* 7 \* 47

Число 659 в канонической форме: 659

Число 660 в канонической форме: 2^2 \* 3 \* 5 \* 11

Число 661 в канонической форме: 661

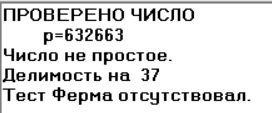
Число 662 в канонической форме: 2 \* 331

Число 663 в канонической форме: 3 \* 13 \* 17

Число 632 в канонической форме: 2^3 \* 79

Число 663 в канонической форме: 3 \* 13 \* 17

**Задание 4**



**Задание 5**

По алгоритму Евклида

a = 663, b = 632

663 = 632 \* 1 + 31

632 = 31 \* 20 + 12

31 = 12 \* 2 + 7

12 = 7 \* 1 + 5

7 = 5 \* 1 + 2

5 = 2 \* 2 + 1

2 = 1 \* 2 + 0  
последний ненулевой = 1. Ответ 1

**Вторая часть**

# Вычисление наибольшего общего делителя

Наибольшее целое число, которое делит без остатка числа *a* и *b*, называется наибольшим общим делителем этих чисел – НОД (*a, b*).

Наибольший общий делитель можно найти с помощью алгоритма Евклида, который заключается в том, чтобы от наибольшего числа из двух отнимать наименьшее, пока одно из них не станет равно нулю, а после вернуть наибольшее из двух число.

Для вычисления наибольшего общего делителя двух чисел реализована следующая функция, представленная на рисунке 1.1:

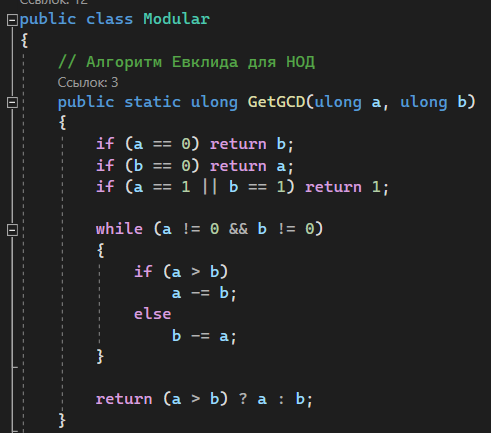


Рисунок 1.1 – Функция нахождения НОД двух чисел

Чтобы найти НОД нескольких чисел (например, *a, b, c*), достаточно найти НОД двух чисел (например, НОД (*a, b*) = *d*), потом НОД полученного (НОД (*a, b*)) и следующего числа (НОД (*c, d*)), и так далее.

Следовательно, для нахождения НОД трёх чисел достаточно в качестве одного из параметров указать вызов функции нахождения НОД. Таким образом получен НОД трёх чисел. Вывод данной функции, а также функции нахождения НОД (*m, n*), где *m* = 632, *n* = 663, представлен на рисунке 1.2.

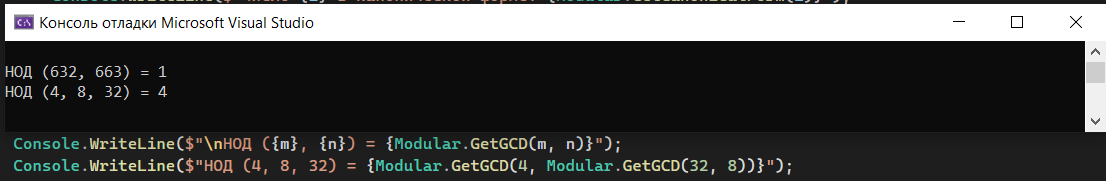


Рисунок 1.2 – Вывод НОД двух и трех чисел

# Поиск простых чисел

Для нахождения всех простых чисел не больше заданного числа *n* используется алгоритм, называемый «решетом Эратосфена». В соответствии с этим алгоритмом нужно выполнить следующие шаги:

1. выписать подряд все целые числа от двух (либо от *m*) до *n* (2, …, *n*). Пусть некоторая переменная (например, *s*) изначально равна 2, то есть первому простому числу;
2. удалить из списка числа от 2*s* до *n*, считая шагами по *s* (это будут числа, кратные *s*: 2*s*, 3*s*, 4*s*, …);
3. найти первое из оставшихся чисел в списке, большее чем *s*, и присвоить значению переменной *s* это число;
4. повторять шаги 2 и 3, пока возможно.

Для реализации данного алгоритма созданы следующие функции, представленные на рисунке 2.1:

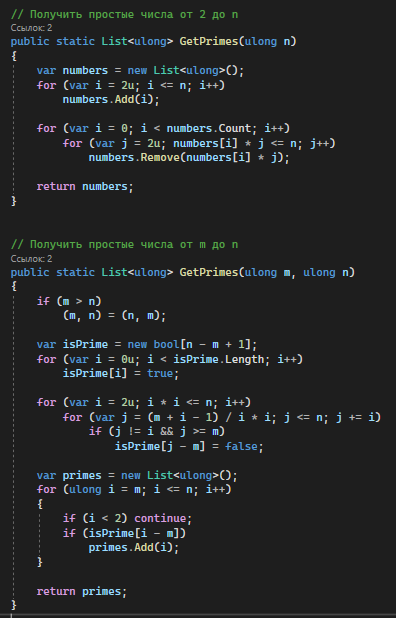


Рисунок 2.1 – Функция поиска простых чисел

Вывод данных функций для поиска простых чисел в интервалах [2, *n*] и [*m*, *n*], а также количество простых чисел в этих диапазонах и значение *n* / ln(*n*) представлены на рисунке 2.2.

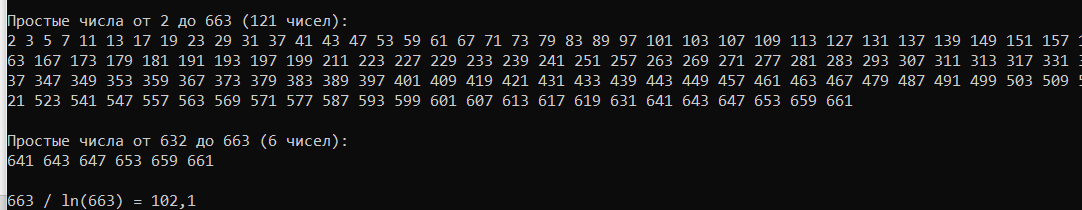


Рисунок 2.2 – Вывод функции поиска простых чисел

# Каноническая форма записи числа

Любое число *n* можно представить в следующем виде, называемое канонической формой записи числа:

 (1.1)

где *p1*, *p2*,…, *pn* – разные простые множители числа, *a1*, *a2*, …, *an* – степени данных простых множителей. Также данные множители можно записывать подряд, по возрастанию, без использования степеней.

Для реализации представления числа в канонической форме реализована следующая функция, представленная на рисунке 3.1.

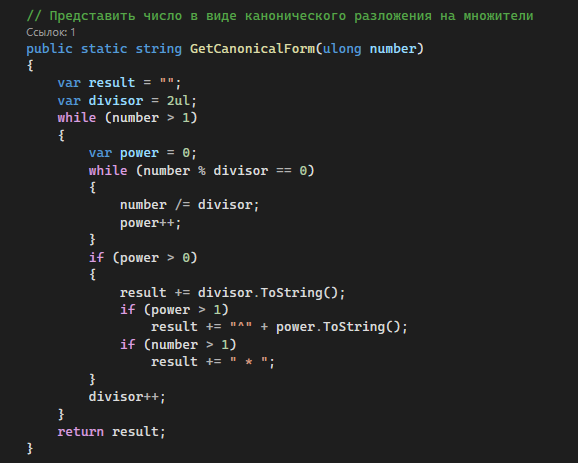


Рисунок 3.1 – Функция представления числа в канонической форме

Данная функция принимает в качестве параметра некоторое число *n* и последовательно делит его на числа от 2 до *√n*. Эти числа записываются в список, который возвращает данная функция. По данному списку в дальнейшем можно циклически пройти и вывести все числа из канонического разложения. Вывод данной функции представлен на рисунке 3.2.

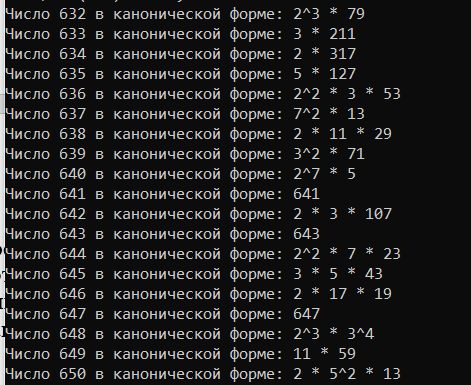


Рисунок 3.2 – Вывод канонической формы числа

# Проверка числа на простоту

Для проверки числа на простоту можно использовать следующий алгоритм: в цикле чисел от 2 до *√n* проверить, делится ли исходное число на одно из них. Если исходное число не делится ни на одно в заданном диапазоне, то оно является простым. Реализация функции проверки числа на простоту представлена на рисунке 4.1.

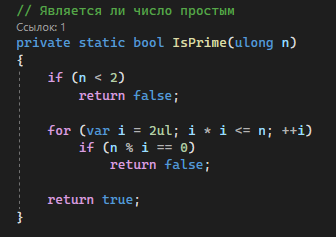


Рисунок 4.1 – Функция проверки числа на простоту

Для выполнения задания проверки на простоту числа, состоящего из конкатенации цифр, из которых состоят числа *m* и *n*, реализована следующая функция, представленная на рисунке 4.2.

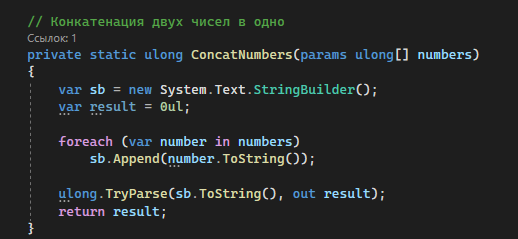


Рисунок 4.2 – Функция конкатенации двух чисел

Далее, необходимо вызвать метод проверки числа на простоту, передав в параметры возвращаемое значение функции конкатенации чисел. Вывод данной функции для чисел *m* = 632 и *n* = 663 представлен на рисунке 4.3.



Рисунок 4.3 – Вывод функции проверки на простоту конкатенации чисел

**Вывод:** в данной лабораторной работе были приобретены практические навыки выполнения операций с числами для решения задач в области криптографии, и разработано приложение для автоматизации этих операций.

Лабораторная работа 4

1. **Описание приложения**

Приложение написано на языке программирования C# и позволяет зашифровать и расшифровать текстовый документ на основе польского языка, используя:

* шифр Виженера с ключевым словом – yashny;
* шифр Порты.

1. **Методика выполнения поставленных задач**

При шифровании исходного текста шифром Виженера каждый символ входного сообщения последовательно замещается на символ, находящийся на (x+k(mod N)) позиции в польском алфавите, где x – позиция исходного символа, k – позиция текущего символа ключевого слова в алфавите, N – мощность алфавита. Реализация функции шифрования приведена на рисунке 2.1.

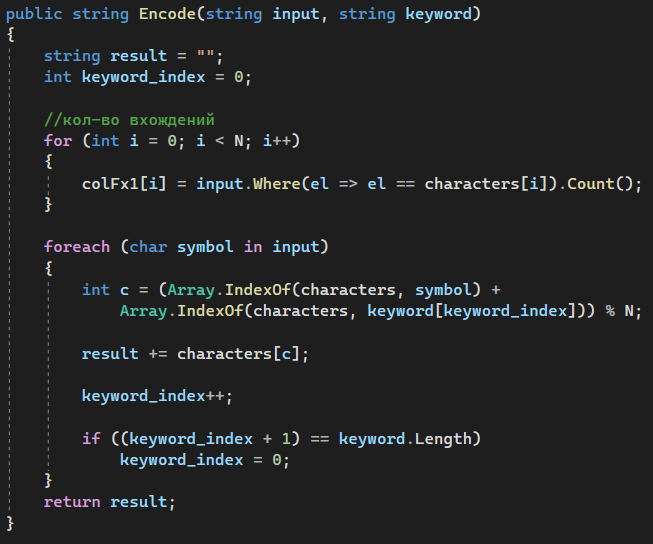


Рис. 2.1 – Реализация шифра Виженера

Расшифрование выполняется путем вычисления новой позиций в алфавите каждого символа после расшифровки по формуле (y–k(modN)), где y – позиция в алфавите текущего зашифрованного символа. Результат шифрования Виженера а также исходный текст представлены на рисунке 2.2.

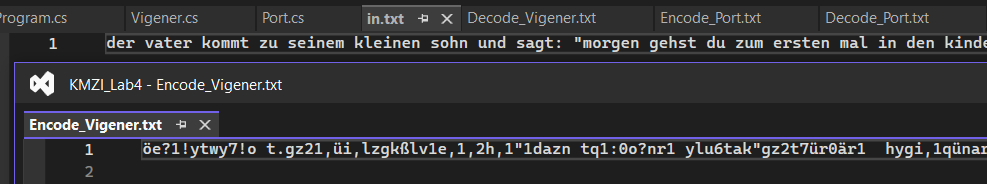


Рисунок 2.2 – Результат шифра Виженера

Для шифрования исходного текста с помощью шифра Порты, используется таблица N x N, где N – количество символов алфавита. Комбинаций пар символов представляются числами. Реализация функции шифрования отображена на рисунке 2.3.

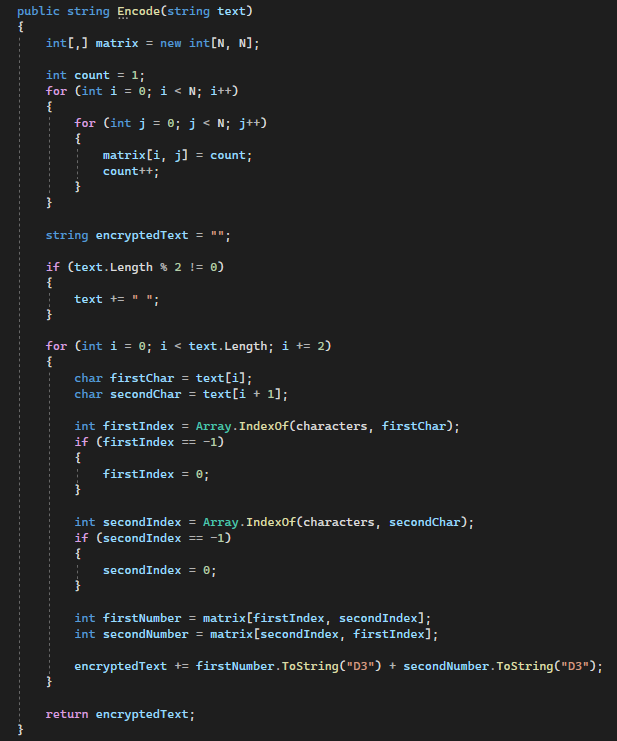


Рис. 2.3 – Реализация шифра Порты

Также приложение оценивает скорость выполнения шифрования/дешифрования с помощью встроенной C#-структуры DateTime.Now.Ticks. Результаты вычисления скоростей представлены на рисунке 2.4. Можно заметить, что оба шифра выполняются с примерно одинаковой скоростью.

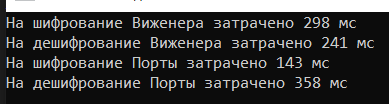


Рис. 2.4 – Скорость выполнения шифров.

Гистограммы частот появления символов в исходном и зашифрованном текстах, представлены на рисунке 2.5.

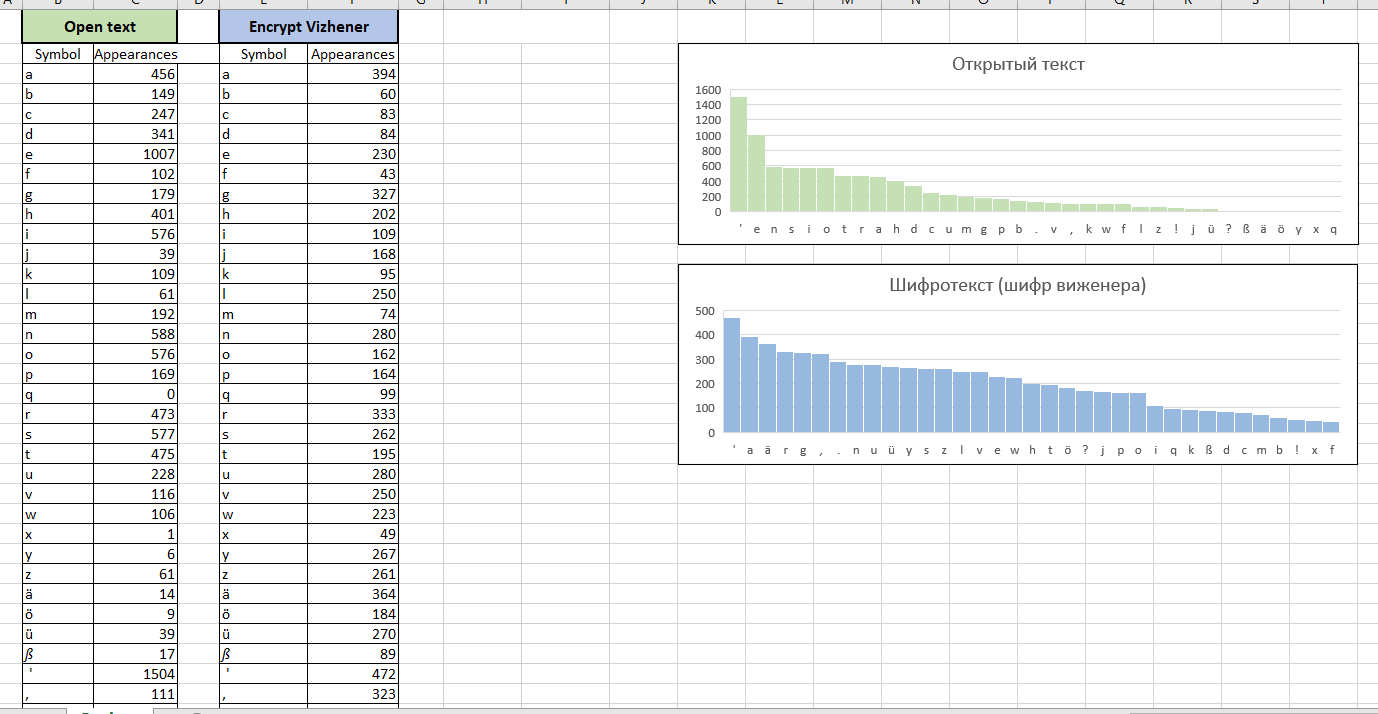


Рис. 2.5 – Гистограммы частот

Можно четко проследить, что распределение символов после кодирования шифром виженера стало намного равномернее.

**Вывод**

В ходе лабораторной работы были приобретены навыки разработки и использования приложений для реализации подстановочных шифров, а также оценены скорость зашифрования/расшифрования реализованных способов шифров.

Лабораторная работа 5

# Шифр маршрутной перестановки

Сущность перестановочного шифрования состоит в том, что исходный текст (*М*) и зашифрованный текст (*С*) основаны на использовании одного и того же алфавита, а тайной или ключевой информацией является алгоритм перестановки.

Одним из перестановочных шифров является шифр маршрутной перестановки. Основой современных шифров рассматриваемого типа является геометрическая фигура, обычно прямоугольник или прямоугольная матрица. В ячейки этой фигуры по определенному маршруту (слева направо, сверху вниз или каким-либо иным образом) записывается открытый текст. Для получения шифрограммы нужно записать символы этого сообщения в иной последовательности, т. е. по иному маршруту.

# Зашифрование

В маршрутной перестановке ключом является маршрут перестановки, а также количество строк и столбцов, произведение которых должно быть больше или равно длине сообщения. Количество строк и столбцов равно 66, что позволяет записывать исходное текстовое сообщение длиной 4322 символа. Выбранный маршрут – «змейкой».

В данном маршруте нечетные столбцы записываются сверху вниз, чётные – снизу вверх. Сам маршрут направляется слева направо. Функция, реализующая зашифрование текста шифром маршрутной перестановки представлена на рисунке 1.1.



Рисунок 1.1 – Функция зашифрования шифром маршрутной перестановки

Для зашифрования используется следующий текстовый документ на немецком языке, представленный на рисунке 1.2.

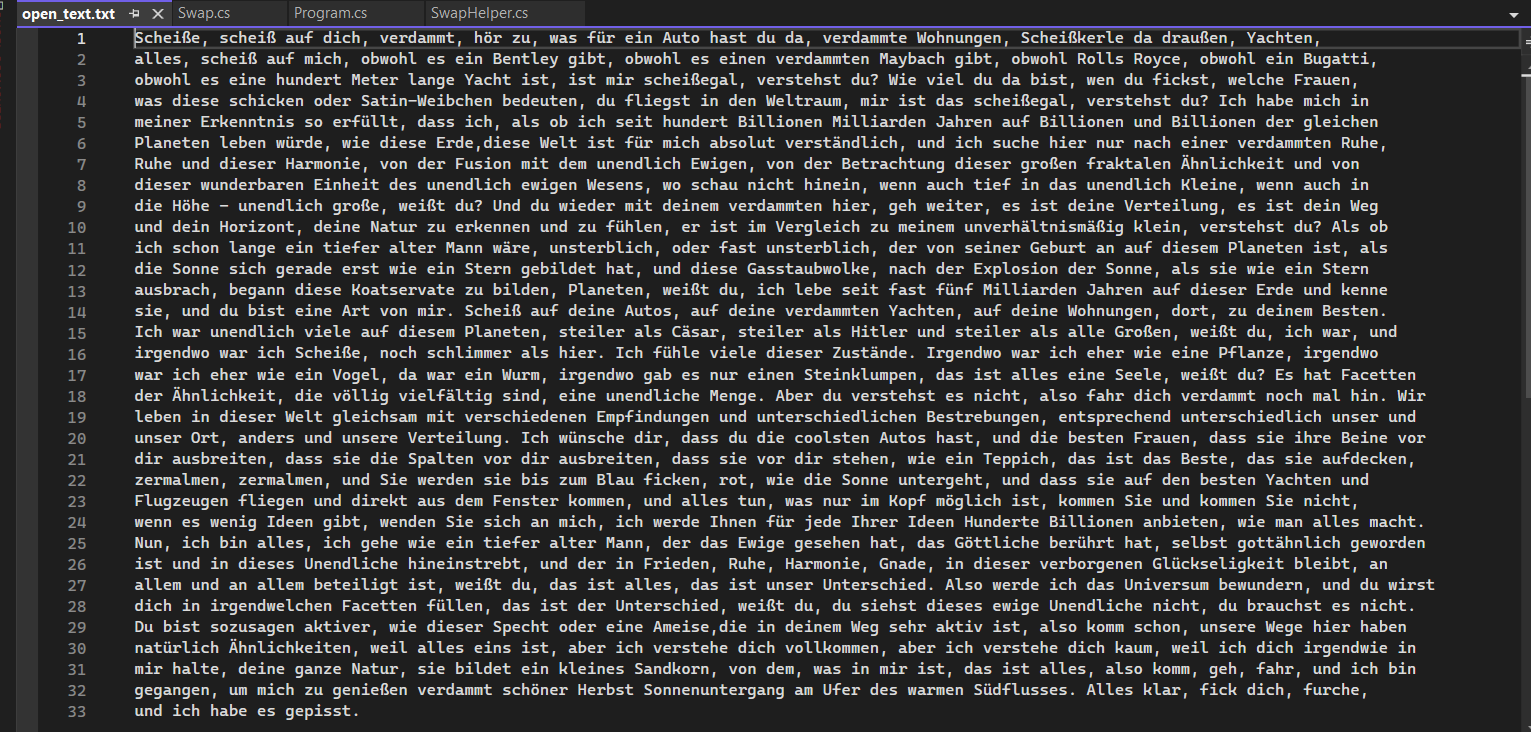


Рисунок 1.2 – Исходный текстовый документ

Результат зашифрования маршрутной перестановкой записывается в файл, содержимое которого представлено на рисунке 1.3.

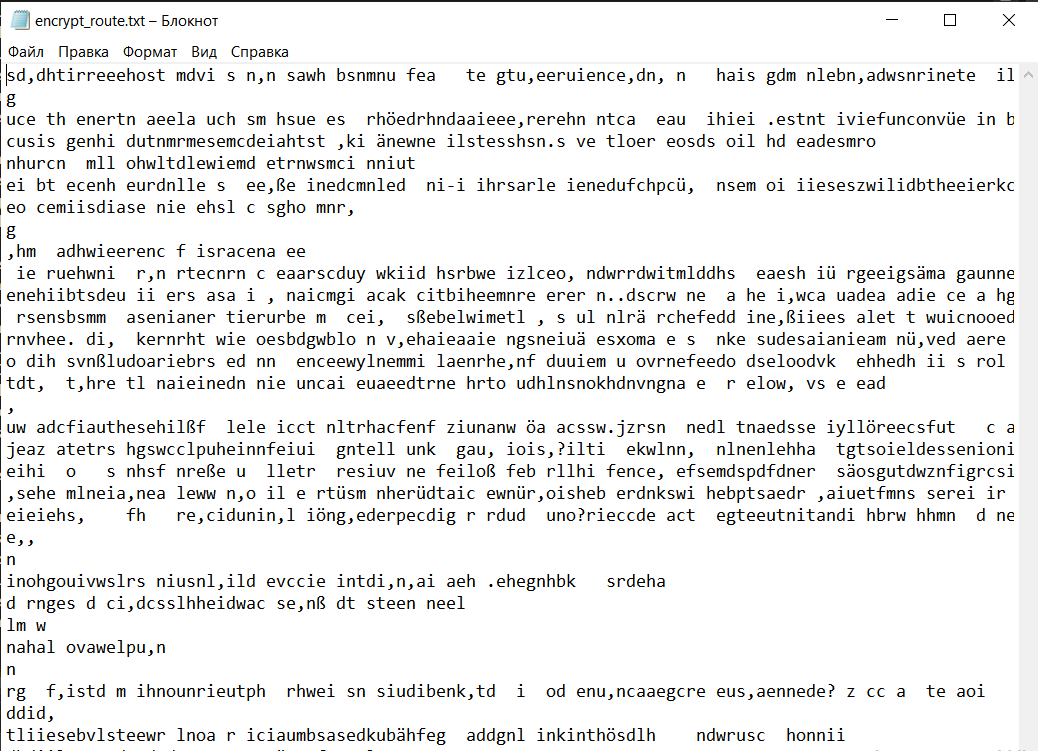


Рисунок 1.3 – Текст, зашифрованный маршрутным шифром

# Расшифрование

Для расшифрования необходимо знать маршрут и количество строк и столбцов. Далее необходимо выполнить тот же маршрут, но в обратном порядке, то есть справа налево, нечетные столбцы снизу вверх, а четные – сверху вниз. Функция, реализующая расшифрование шифра маршрутной перестановки, представлена на рисунке 1.4.



Рисунок 1.4 – Функция расшифрования шифром маршрутной перестановки

В результате действия функции был получен текст, представленный на рисунке 1.5, идентичный открытому тексту, что свидетельствует о том, что алгоритм зашифрования и расшифрования работает корректно.

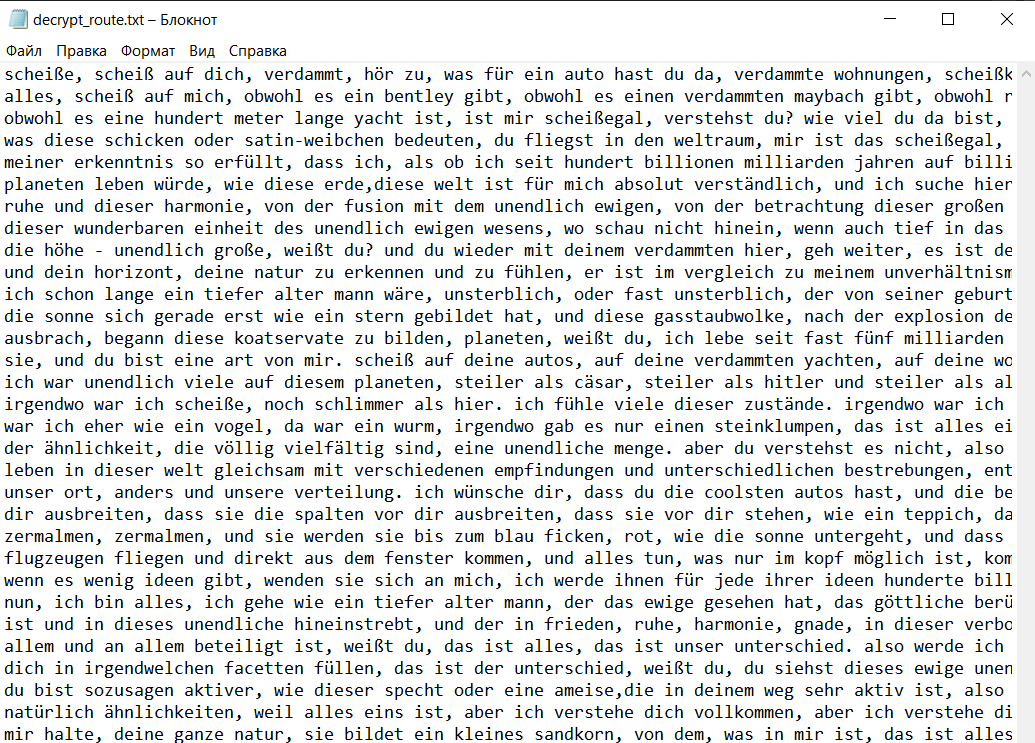


Рисунок 1.5 – Текст, расшифрованный маршрутным шифром

# Шифр множественной перестановки

Для начала, рассмотрим шифр вертикальной перестановки. К особенностям вертикального шифра можно отнести следующие:

* количество столбцов в таблице определяется длиной ключа;
* маршрут вписывания: слева направо, сверху вниз;
* шифрограмма выписывается по столбцам в соответствии с их нумерацией (ключом).

Ключ задается в виде текста. Лексикографическое местоположение символов в ключевом выражении определяет порядок считывания столбцов.

Шифр множественной перестановки является усовершенствованным вариантом шифра вертикальной перестановки. В нём производится не только перестановка по строкам, но и по столбцам. То есть, производится минимум двукратная перестановка символов исходного текста.

# Зашифрование

Для зашифрования необходима следующая информация: два ключевых слова, а также количество строк и столбцов в таблице, в которую будет вписано исходное сообщение, которое также должно быть больше или равно длине исходного сообщения. В случае, когда ключевые слова слишком короткие, они циклически повторяются. Далее записываются индексы строк и столбцов в соответствии с порядком символов в алфавите, после выполняется перестановка строк и далее столбцов. Функция, выполняющая зашифрование шифром множественной подстановки представлена на рисунке 2.1.

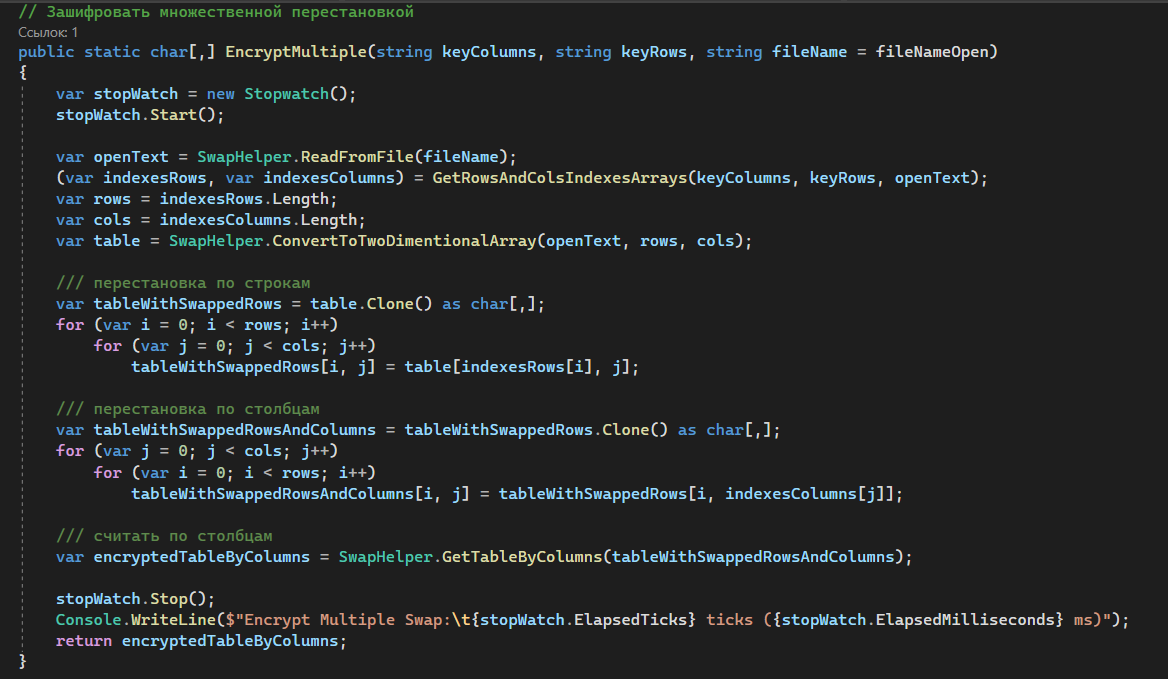


Рисунок 2.1 – Функция зашифрования шифром множественной подстановки

Результатом выполнения функции является зашифрованный текст, представленный на рисунке 2.2.



Рисунок 2.2 – Текст, зашифрованный множественным шифром

# Расшифрование

Для расшифрования текста, зашифрованного шифром множественной подстановки, необходимо выполнить те же действия в обратном порядке. Необходимо знать ключевые слова и сам зашифрованный текст. Ключевые слова по аналогичному алгоритму повторяются циклически. Функция, реализующая расшифрование текста, представлена на рисунке 2.3.

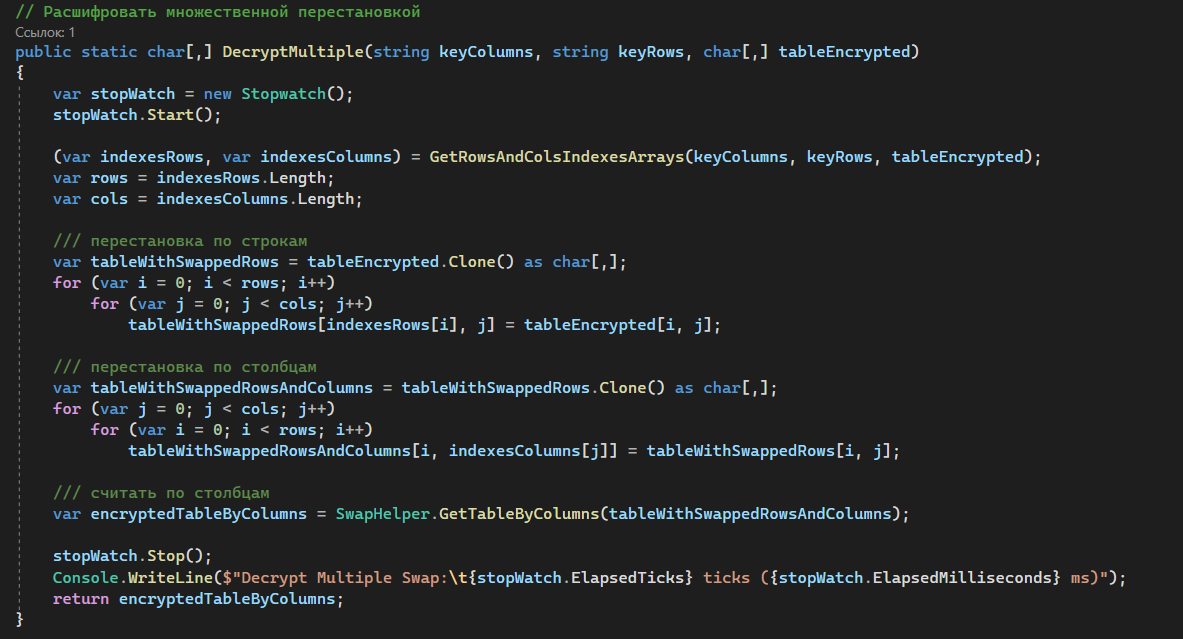


Рисунок 2.3 – Функция расшифрования шифра множественной подстановки

# Гистограммы частот появления символов

Для определения криптостойкости построим гистограммы частот появления символов в открытом тексте и текстах, зашифрованных моноалфавитным подстановочным шифром и таблицей Трисемуса.

Для вычисления количества появлений символов в тексте используется функция, представленной на рисунке 3.1.

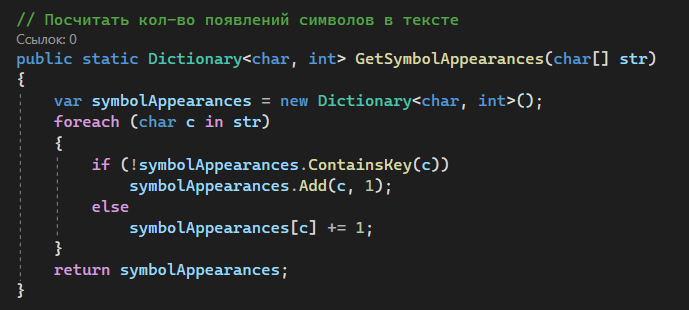


Рисунок 3.1 – Функция подсчёта количества символов

На основании количества появлений символов в тексте при известном общем количестве символов можно построить гистограммы частот появления символов в исходном тексте и шифротекстах, изображённые на рисунке 3.2.



Рисунок 3.2 – Гистограммы частот появления символов в шифротекстах

Как видно из рисунка, все три гистограммы идентичны между собой. Это объясняется тем, что в подстановочных шифрах каждому символу алфавита всегда соответствует только один символ того же алфавита, записанного в другом порядке.

Таким образом, подстановочные шифры являются уязвимыми к частотному криптоанализу, аналогично с перестановочными шифрами.

# Время выполнения зашифрования и расшифрования

Для оценки времени, затраченного на выполнение операций зашифрования и расшифрования, используется объект Stopwatch, выводящий время выполнения операций в процессорном времени и в миллисекундах.

Вывод функций оценки времени представлен на рисунке 4.1.

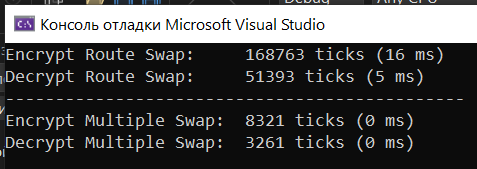


Рисунок 4.1 – Вывод функции оценки времени выполнения

# Вывод

Были исследованы два вида шифров - маршрутная перестановка и множественная перестановка. С точки зрения криптоанализа, оба шифра не являются криптостойкими, так как подвержены атаке частотного анализа.

При сравнении времени выполнения операций зашифрования и расшифрования обоих шифров было выяснено, что шифрование множественной перестановкой происходит быстрее, чем шифрование маршрутной перестановкой, примерно на 40% для входных документов длиной не более 1 миллиона символов. Однако с увеличением количества входных символов скорость шифрования множественной перестановкой начинает снижаться.

Для текстов длиной более 5 миллионов символов маршрутная перестановка становится примерно в 2 раза быстрее множественной перестановки.

Таким образом, для небольших документов (менее 1 миллиона символов) целесообразно использовать множественную перестановку.

При сравнении с подстановочными шифрами, такими как моноалфавитный шифр и таблица Трисемуса, выяснилось, что последние работают медленнее перестановочных шифров, примерно в 2-3 раза, независимо от количества входных символов.

С учетом того, что криптостойкость подстановочных и перестановочных шифров примерно одинакова, предпочтительнее использовать перестановочные шифры из-за их более высокой производительности.

Лабораторная работа 6

# Устройство шифровальной машины «Энигма»

Машина «Энигма» – это электромеханическое устройство шифрования. Как и другие роторные машины, «Энигма» состоит из комбинации механических и электрических подсистем.

Механическая часть включает в себя клавиатуру, набор вращающихся дисков – роторов, которые расположены вдоль вала и прилегают к нему, и ступенчатого механизма, двигающего один или несколько роторов при каждом нажатии на клавишу.

Электрическая часть, в свою очередь, состоит из электрической схемы, соединяющей между собой клавиатуру, коммутационную панель, лампочки и роторы (для соединения роторов использовались скользящие контакты).

Общий принцип таков:

При каждом нажатии на клавишу самый правый ротор сдвигался на одну или несколько позиций, а остальные роторы сдвигались на одну или несколько позиций в случае, если ротор, расположенный справа, делал полный оборот.

Движение роторов приводило к различным криптографическим преобразованиям при каждом следующем нажатии на клавишу на клавиатуре, т. е. зашифрование/расшифрование сообщений основано на выполнении ряда замен (подстановок) одного символа другим.

Идея заключается в том, чтобы добиться этих подстановок электрическими связями. Механические части двигались и, замыкая контакты, образовывали меняющийся электрический контур.

При нажатии на клавишу клавиатуры контур замыкается, ток проходит через созданную электрическую цепь и в результате включает одну из набора лампочек, отображающую искомую букву шифротекста (или расшифрованного сообщения).

# Зашифрование машиной «Энигма»

Для реализации зашифрования необходимо, во-первых, хранить в памяти следующие данные:

* подстановочные алфавиты, соответствующие каждому ротору;
* текущую позицию каждого ротора;
* общее количество смещений каждого ротора;
* количество полных оборотов роторов;
* шаги смещения каждого ротора.

Для имплементации хранения этих данных созданы следующие поля класса, представленные на рисунке 1.1.



Рисунок 1.1 – Поля для хранения данных

Также через конструктор класса возможно задать стартовые позиции роторов (то есть количество символов, на сколько будет сдвинут каждый ротор). Позиции могут быть только в диапазоне от 0 до 25, так как в английском алфавите, используем для шифрования, 26 символов. При установке некорректных значений в конструктор класса будет возвращено исключение. Структура конструктора класса представлена на рисунке 1.2.

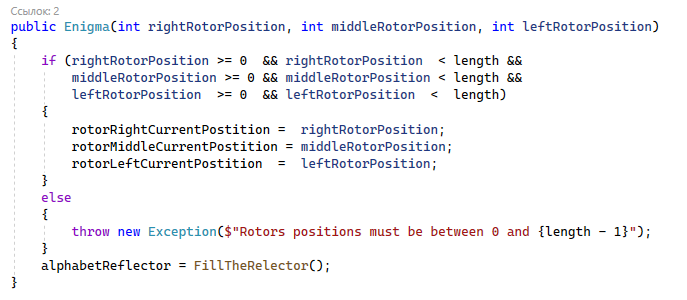


Рисунок 1.2 – Конструктор класса

Рассмотрим подробнее сам алгоритм зашифрования. Исходный открытый текст разбивается посимвольно и, если в английском алфавите присутствует такой символ, то производится зашифрование, иначе символ остается неизменным (например, знаки пунктуации не шифруются).

Далее символ шифруется правым ротором. Каждый ротор имеет собственную конфигурацию алфавита подстановки, которые были описаны ранее на рисунке 1.1. Для каждого ротора используется одинаковый алгоритм шифрования, предполагающий сдвиг на определенное количество позиций. Алгоритм зашифрования каждым ротором представлен на рисунке 1.3.

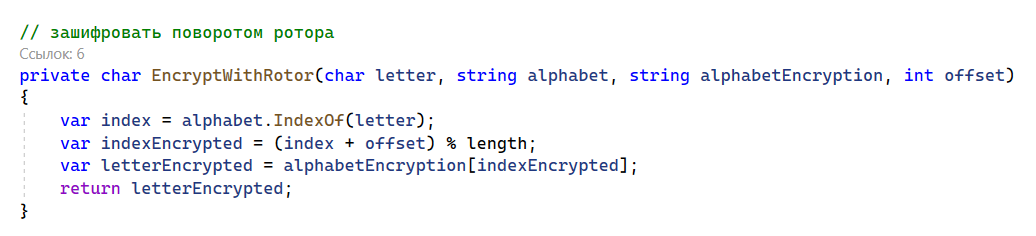


Рисунок 1.3 – Функция зашифрования ротором

Символ, который был получен при зашифровании правым ротором, передается в качестве входного параметра для среднего ротора. Аналогично, символ, зашифрованный средним ротором, шифруется после левым ротором. То есть, изначальный символ на данном этапе уже был зашифрован 3 раза.

Далее символ, полученный после зашифрования левым ротором, передается на рефлектор – устройство, которое также выполняет подстановку одного символа другим. Для реализации рефлектора используются пары символов, где каждый символ подставляется другим. Имплементация функции для зашифрования рефлектором представлена на рисунке 1.4.

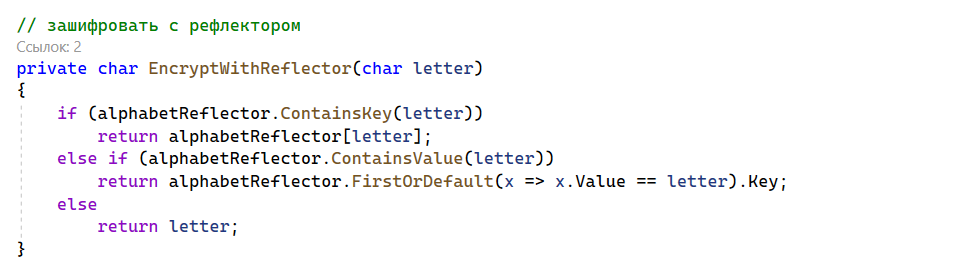


Рисунок 1.4 – Функция зашифрования рефлектором

После зашифрования рефлектором, символ исходного текста проходит те же самые роторы в обратном порядке. Выполняются еще 3 замены символа, и, учитывая замену в рефлекторе, символ будет заменён 7 раз.

Далее необходимо учесть повороты роторов. Правый ротор в конце шифрования каждого символа сдвигается на определённое количество символов, а средний и левый ротор будут сдвигаться только если расположенный правее ротор совершит полный оборот. Код для поворота роторов, а также код для функции зашифрования представлен на рисунке 1.5.



Рисунок 1.5 – Функция зашифрования машиной «Энигма»

# Расшифрование машиной «Энигма»

Для расшифрования необходимо установить роторы в такие же позиции, как и при зашифрования. Алгоритм аналогичен, кроме функции расшифрования ротором. Данная функция изображена на рисунке 1.6.

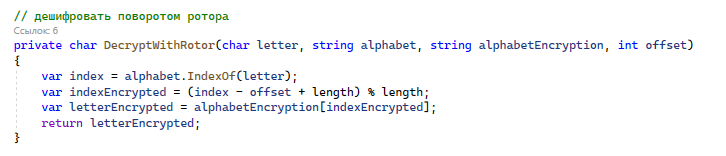


Рисунок 1.6 – Функция расшифрования с помощью ротора

# Вывод

Машина "Энигма" получила широкое распространение во время Второй мировой войны благодаря своей высокой криптостойкости. Она достигала этой стойкости за счет многократного шифрования каждого символа (обычно 7 раз в варианте с тремя роторами без коммутационной панели), при этом каждый символ преобразовывался в новый символ, что дополнительно повышало сложность криптоанализа.

При сравнении моего варианта реализации с реальной машиной "Энигма" можно сказать, что мой вариант немного менее криптостоек из-за отсутствия коммутационной панели, которая вносила от 0 до 2 замен каждого символа. Однако, даже без коммутационной панели шифровальная машина "Энигма" остается значительно более криптостойкой по сравнению с рассмотренными ранее шифрами.

Вывод можно сделать таким образом, что машина "Энигма" обладала высокой криптостойкостью благодаря своей сложной системе многократного шифрования символов. В сравнении с другими шифрами, включая мой вариант, "Энигма" оставалась намного более надежной и трудной для взлома.

Лабораторная работа 7

# Алгоритм DES

Алгоритм базируется на сети Фейстеля. Он принимает на вход блок данных размером 64 бита и преобразует его в выходной блок той же длины. Алгоритм использует подстановки и перестановки битов текста, которые были упомянуты ранее, чтобы достичь своей цели. Он состоит из 16 раундов, в каждом из которых один блок данных проходит через одни и те же операции.

В начале каждого раунда правая половина блока данных, обозначенная как Ri, расширяется до 48 бит с помощью таблицы, которая определяет перестановку и расширение на 16 бит. Затем происходит операция XOR между расширенным значением и 48-битным подключом Ki. Полученное значение подается на вход блока подстановки S, который состоит из восьми S-блоков. Каждый S-блок принимает 6-битные подблоки данных и заменяет их на 4-битные значения с помощью таблиц замен.

После операции с блоком подстановки S, полученные 32-битные значения проходят через операцию перестановки Р, которая не зависит от используемого ключа. Затем результат перестановки объединяется с помощью операции XOR с левой половиной исходного блока данных. Затем левая и правая половины меняются местами, и процесс повторяется для следующего раунда.

Этот алгоритм DES (Data Encryption Standard) является фундаментальным строительным блоком и обладает высокой криптостойкостью благодаря комбинации подстановок и перестановок, а также использованию ключей и таблиц замен. Он успешно применялся в криптографии, включая использование в машине "Энигма", и остается одним из важных алгоритмов шифрования.

# Зашифрование

Для начала, нам необходимо убедиться, что ключ, подаваемый на вход функции зашифрования, будет иметь размерность 64 бита. Для этого реализована функция, представленная на рисунке 1.1.

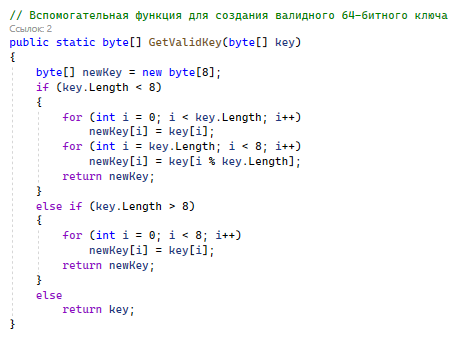


Рисунок 1.1 – Функция для создания 64-битного ключа

Для зашифрования необходимо убедиться, что блоки исходного текста будут иметь размерность 64 бита и при необходимости дополняться до данного размера. Для реализации этого используется значение PKCS7 в перечислении PaddingMode. Блоки будут дополняться байтами, равными длине дополнения. Например, если размер блока 8 байт, то для дополнения блока длиной 6 байт будет добавлено 2 байта со значением 0x02. Функция зашифрования представлена на рисунке 1.2.

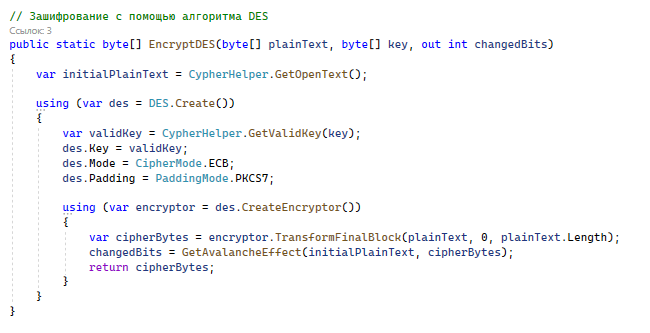


Рисунок 1.2 – Функция зашифрования алгоритмом DES

# Расшифрование

Для расшифрования необходимо повторить тот же самый алгоритм с обратным порядком ключей. Если использовались раундовые ключи с 1-го по 16-ый, то при расшифровании они будут применять в обратном порядке – с 16-го по 1-ый. Функция расшифрования представлена на рисунке 1.3.



Рисунок 1.3 – Функция расшифрования алгоритмом DES

# Лавинный эффект

Алгоритму DES присуще такое свойство, как «лавинный» эффект. Оно заключается в том, что изменение значения малого количества битов во входном тексте или в ключе ведет к «лавинному» изменению значений выходных битов шифротекста.

Для оценки «лавинного» эффекта можно побитово сравнить исходный текст и шифротекст и посчитать количество изменённых битов. Функция, высчитывающая «лавинный» эффект, представлена на рисунке 1.4.

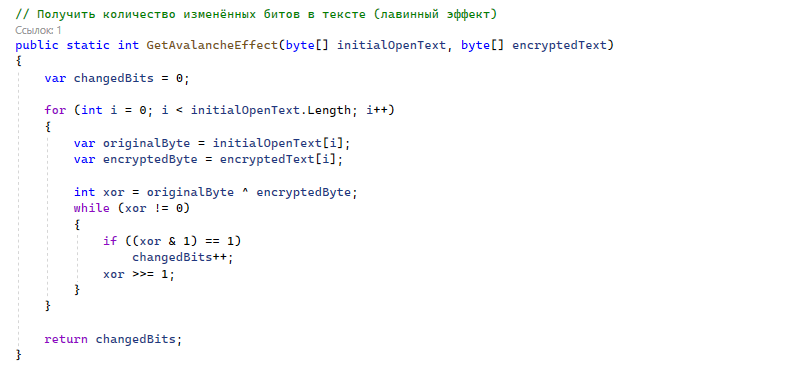


Рисунок 1.4 – Функция оценки «лавинного» эффекта

# Алгоритм DES-EEE2

Существует модификация DES, называемая 3DES или TripleDES. Смысл модификации в том, что алгоритм DES применяется три раза. Есть несколько разновидностей TripleDES:

* DES-EEE3: шифруется три раза с тремя разными ключами (операции шифрование-шифрование-шифрование);
* DES-EDE3: 3DES операции шифрование-расшифрование шифрование с тремя разными ключами;
* DES-EEE2 и DES-EDE2: как и предыдущие, однако на первом

и третьем шаге используется одинаковый ключ.

Рассмотрим подробнее алгоритм DES-EEE2.

Для реализации данного алгоритма используется функция, представленная на рисунке 2.1.

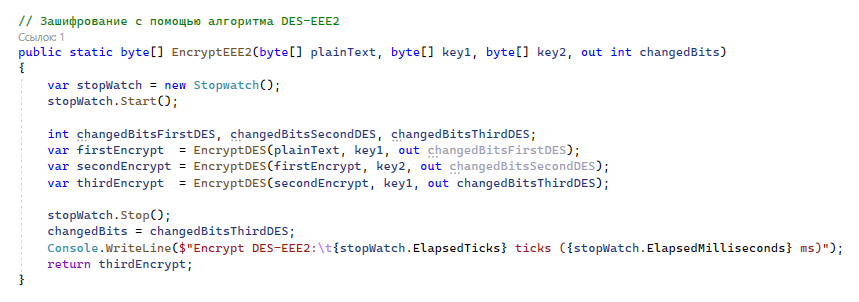


Рисунок 2.1 – Функция зашифрования алгоритмом DES-EEE2

Для расшифрования необходимо выполнить три операции расшифрования с одинаковыми ключами при первом и третьем расшифровании. Функция представлена на рисунке 2.2.



Рисунок 2.2 – Функция расшифрования алгоритмом DES-EEE2

# Вывод

В данный лабораторной работе были изучены и приобретены практические навыки разработки и использования приложений для реализации блочного шифра DES.

Лабораторная работа 8

# Потоковые шифры

Потоковые шифры представляют собой симметричные шифры, которые преобразуют каждый символ открытого текста в символ шифрованного текста. Шифрование основано на ключе и позиции символа в тексте.  
Основная задача потоковых шифров заключается в генерации последовательности данных, называемой гаммой, которая используется в качестве ключевого потока для шифрования сообщения.  
Главная цель потоковых шифров — создать эффект одноразового блокнота. В этом контексте ключ должен быть максимально похож на случайную числовую последовательность. Ключевые последовательности, такие как случайные последовательности (СП) или псевдослучайные последовательности (ПСП), генерируются специальными блоками систем потокового шифрования, называемыми генераторами.

# Генерация ПСП на основе алгоритма BBS

Начальное значение *x0* генератора вычисляется на основе соотношения



где *n* является произведением простых чисел *p* и *q*, однако в нашем случае эти простые числа должны быть сравнимы с числом 3 по модулю 4; число *x* должно быть взаимно простым с *n*; число *n* называют числом Блюма.

Выходом генератора на *t*-м шаге является младший бит числа *xt*:



Для реализации алгоритма генерации псевдослучайных последовательностей с помощью алгоритма BBS был создан одноименный класс, представленный на рисунке 1.1.

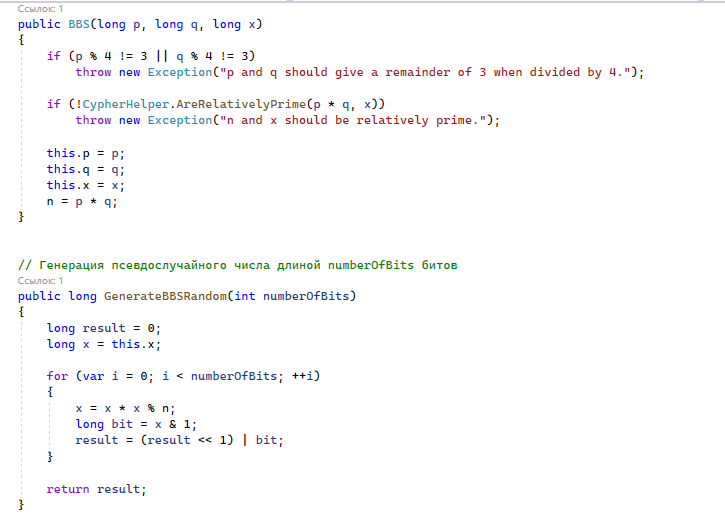


Рисунок 1.1 – Код генерации ПСП с помощью BBS

В соответствии с вариантом, на вход алгоритма подаются следующие числа: *p* = 56155, *q* = 87151, *x* = 512. Результат работы алгоритма, создающий 16-битное псевдослучайное число, представлен на рисунке 1.2.

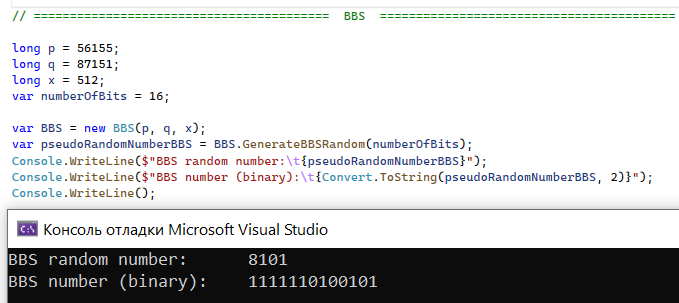


Рисунок 1.2 – Вывод алгоритма генерации ПСП

# Шифрование алгоритмом RC4

Для работы алгоритма необходимо выполнить следующие действия:

* инициализировать *S*-блок таблицы начальных замен;
* сгенерировать на основании *S*-блока псевдослучайные числа для ключей *K*;
* выполнить операцию сложения по модулю 2 каждого 8-битного ключа *K* с открытым текстом для получения шифротекста.

Для инициализации *S*-блока начальных замен реализована следующая функция, представленная на рисунке 1.3.

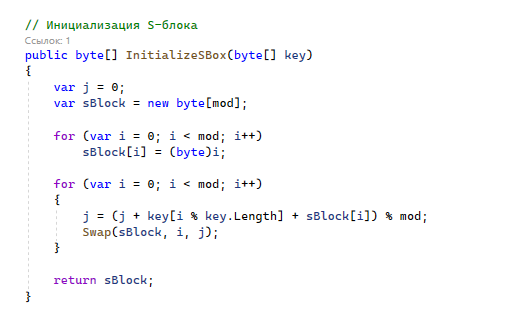


Рисунок 1.3 – Код функции инициализации S-блока

Далее по формуле необходимо создать псевдослучайную последовательность ключей *K*, что представлено на рисунке 1.4.

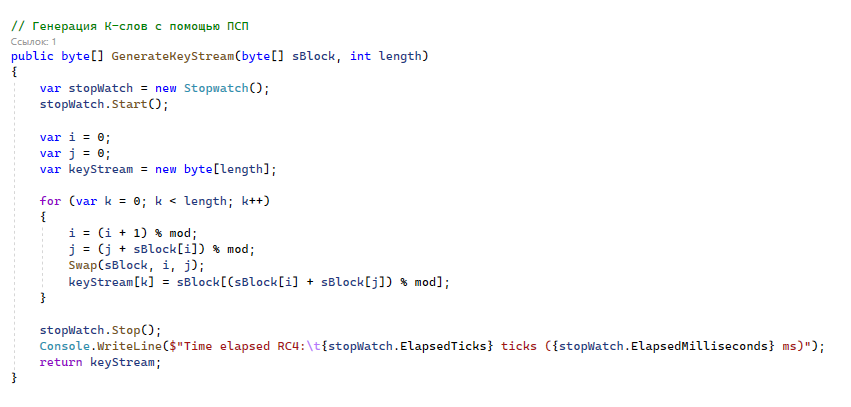


Рисунок 1.4 – Код функции генерации ПСП

Методы для зашифрования и расшифрования, а также вспомогательный метод для замены элементов местами в *S*-блоке представлены на рисунке 1.5.

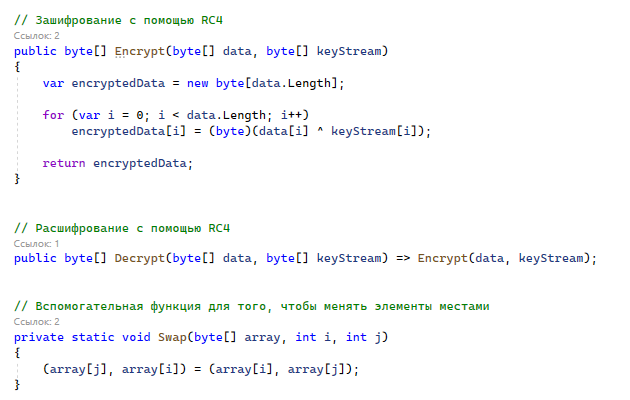


Рисунок 1.5 – Код функций зашифрования и расшифрования

Вывод функции зашифрования и расшифрования и затраченного времени представлен на рисунке 1.6.

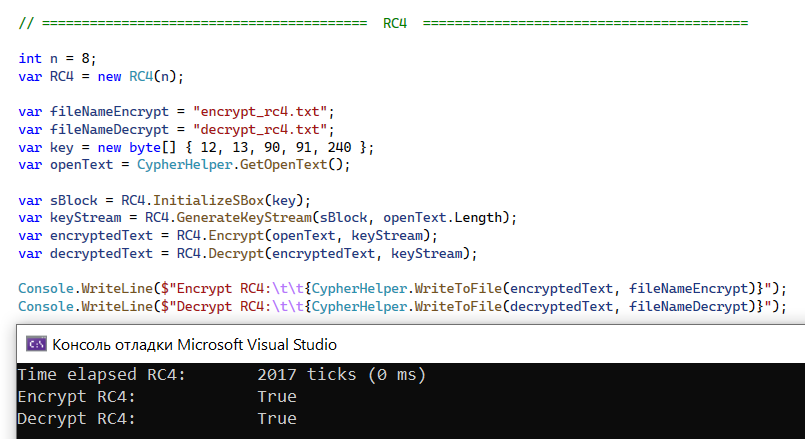


Рисунок 1.6 – Вывод функций зашифрования и расшифрования

# Время выполнения генерации ПСП

Для оценки времени выполнения операций генерации ПСП были созданы псевдослучайные последовательности разной длины. График представлен на рисунке 2.1.



Рисунок 2.1 – Время выполнения генерации ПСП

При анализе данного графика можно заметить, что генерация ПСП происходит гораздо быстрее, чем операции зашифрования и расшифрования в остальных алгоритмах, из чего можно сделать вывод, что алгоритм RC4 может быть довольно производительным для создания псевдослучайных последовательностей.

# Вывод

В данный лабораторной работе были изучены и приобретены практические навыки разработки и использования потоковых шифров.

Лабораторная работа 9

# Криптоалгоритм на основе задачи об укладке ранца

В задаче о ранце (рюкзаке) рюкзачный вектор S представляет собой упорядоченный набор из z различных натуральных чисел si, где z ≥ 3. Входом задачи о ранце является пара (S, S), где S - рюкзачный вектор, а S - натуральное число.  
Решением для данного входа (S, S) является подмножество элементов из S, сумма которых равна S.  
В наиболее известной формулировке задачи о ранце требуется определить, существует ли решение для данного входа (S, S). В криптографическом контексте используется вариант задачи, в котором уже известно, что решение существует, и требуется построить это решение. Оба варианта задачи являются NP-полными. Существуют также варианты задачи, которые не принадлежат даже классу NP.  
Основная идея метода шифрования на основе задачи о ранце заключается в том, что существуют две различные задачи укладки ранца: одна из них легко решается и имеет линейную сложность, а другая решается с трудом. Легкую задачу укладки ранца можно преобразовать в трудную.  
Трудная задача укладки ранца используется в качестве открытого ключа, который легко используется для шифрования сообщений, но трудно поддаётся расшифровке. Легкая задача укладки ранца, в свою очередь, выступает в роли закрытого ключа и предоставляет простой способ расшифровки сообщений.

# Генерация сверхвозрастающей последовательности

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

В качестве закрытого ключа ***d*** (легкого для укладки ранца) используется сверхвозрастающая последовательность, состоящая из *z* элементов: *d1*, *d2*, …, *dz*: ***d*** = {*di*}, *i* = 1, …, *z*.

Для генерации сверхвозрастающей последовательности реализована следующая функция, отображённая на рисунке 1.1.

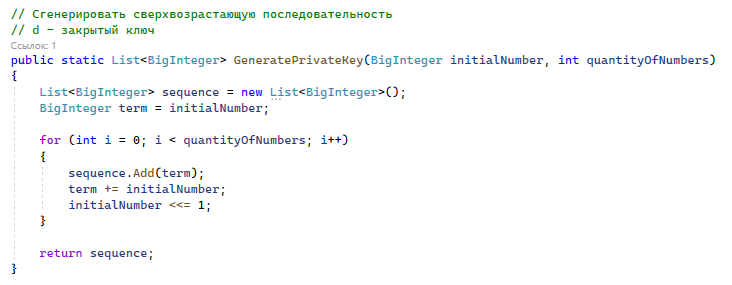


Рисунок 1.1 – Код генерации сверхвозрастающей последовательности

# Генерация нормальной последовательности

Открытый ключ ***e*** представляет собой нормальную (не сверхвозрастающую) последовательность. Он формируется на основе закрытого ключа и не позволяет легко решить задачу об укладке ранца.

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



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

Код функции для генерации нормальной последовательности представлен на рисунке 1.2.

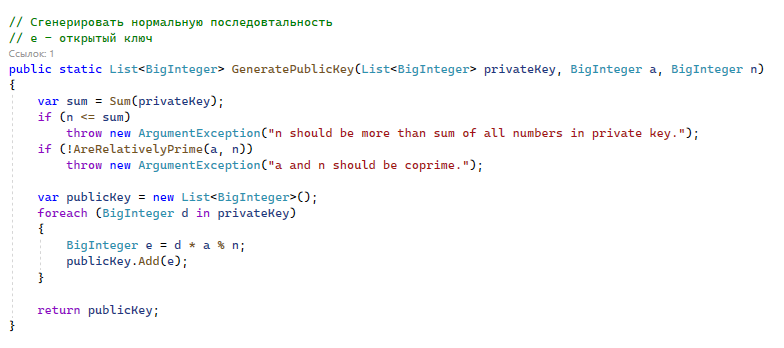


Рисунок 1.2 – Код функции генерации нормальной последовательности

# Зашифрование

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***.

Код функции для зашифрования представлен на рисунке 1.3.

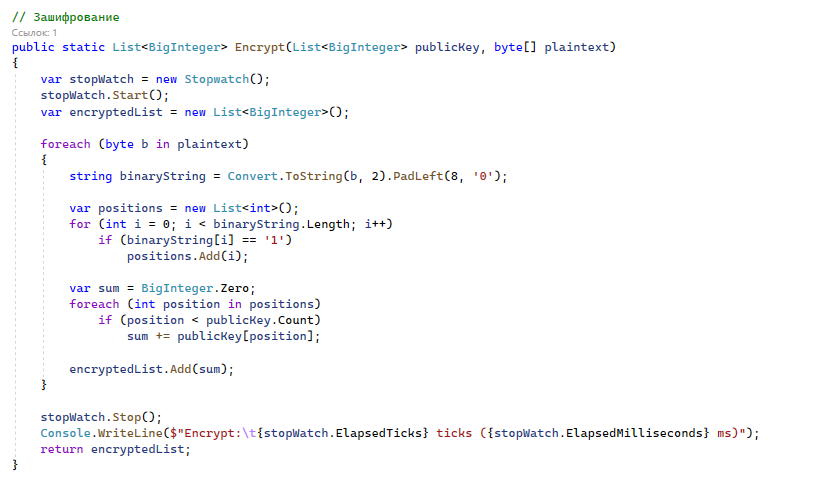


Рисунок 1.3 – Код функции зашифрования

# Расшифрование

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



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



Полученное на основании последней формулы для каждого блока число далее рассматривается как заданный вес ранца, который следует упаковать, используя сверхвозрастающую последовательность, то есть тайный ключ получателя.

Код функции расшифрования представлен на рисунке 1.4.



Рисунок 1.4 – Код функции расшифрования

# Время выполнения зашифрования и расшифрования

Для оценки времени выполнения операций зашифрования и расшифрования была проведена оценка с разными таблицами кодировки, и, соответственно, размером блоков шифрования, а также при подаче на вход алгоритма генерации ключей разных чисел z, которые отвечают за количество членов в ключевой последовательности.

Оценку времени операций зашифрования и расшифрования при использовании разных таблиц кодировки можно увидеть на рисунке 2.1. Разница между ними несущественная, однако, Base64 имеет тенденцию к немного более медленному зашифрованию и расшифрованию.



Рисунок 2.1 – Время зашифрования в ASCII и Base64

# Вывод

В данный лабораторной работе были изучены и приобретены практические навыки разработки и использования ассиметричных шифров.

Лабораторная работа 10

# Алгоритм RSA

Для генерации двух взаимосвязанных ключей - открытого и тайного, используются два больших случайных простых числа p и q. Чтобы обеспечить максимальную криптостойкость, рекомендуется выбирать p и q одинаковой длины. Затем произведение p и q, обозначенное как n, вычисляется и становится одной из компонент ключа.  
Далее случайным образом выбирается вторая компонента ключа, известная как открытый ключ или ключ шифрования, обозначенная как e. Она должна быть взаимно простым числом с (p - 1)(q - 1), которое является значением функции Эйлера φ(n).  
Наконец, расширенный алгоритм Евклида используется для вычисления третьей компоненты ключа, которая является тайным ключом или ключом расшифрования, обозначенным как d. Он должен удовлетворять условию:  
Таким образом, формируется ключ, состоящий из трех чисел, которые взаимосвязаны. Открытый ключ (e, n) и тайный ключ (d, n) образуют пару ключей. Отправитель использует открытый ключ для шифрования сообщения, а получатель расшифровывает зашифрованный текст с помощью своего тайного ключа.

# Зашифрование алгоритмом RSA

Если шифруется сообщение *М*, состоящее из r блоков: *m1*, *m2*, …, *mi*, …, *mr*, то шифртекст *С* будет состоять из такого же числа блоков, представляемых числами:



Для зашифрования используется следующий метод, представленный на рисунке 1.1.



Рисунок 1.1 – Функция зашифрования алгоритмом RSA

# Расшифрование алгоритмом RSA

Для расшифрования каждого зашифрованного блока производится вычисление вида:



Код функции для расшифрования представлен на рисунке 1.2.



Рисунок 1.2 – Функция расшифрования алгоритмом RSA

# Алгоритм Эль-Гамаля

Для генерации ключевой информации в алгоритме используются следующие шаги. Сначала выбирается простое число p. Затем выбирается число g, которое является первообразным корнем числа p. Это важно с точки зрения безопасности алгоритма. Далее выбирается число x, которое должно быть меньше числа p. И наконец, вычисляется последний компонент ключевой информации, обозначенный как y.  
Таким образом, формируется ключевая информация, состоящая из четырех чисел. Владелец этой ключевой информации может получать зашифрованные сообщения, созданные с использованием открытого ключа получателя, который состоит из чисел p, g и y. Для расшифровки шифртекста получатель использует свой тайный ключ, состоящий из чисел p, g и x. Как видно, в этом алгоритме тайным является только одно число - x.

# Зашифрование алгоритмом Эль-Гамаля

Для зашифрования алгоритмом Эль-Гамаля реализован код функции, представленной на рисунке 2.1.

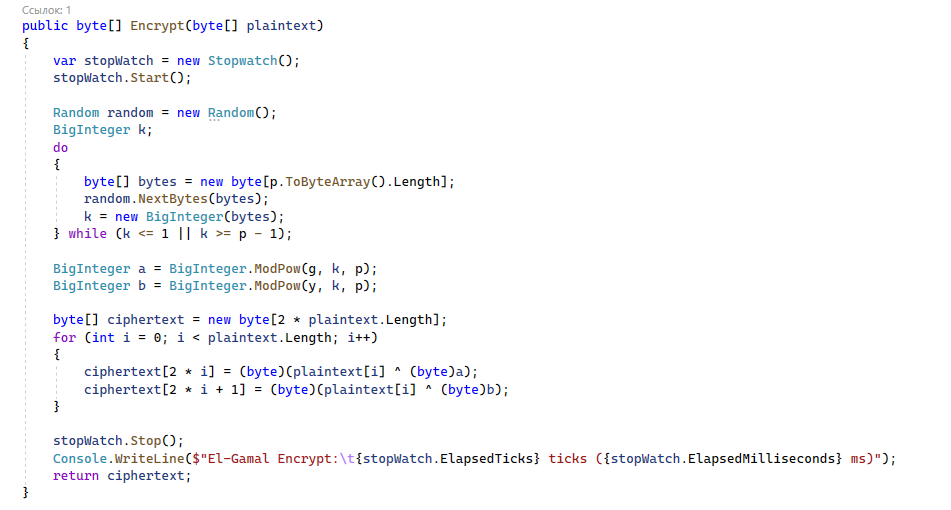


Рисунок 2.1 – Функция зашифрования алгоритмом Эль-Гамаля

# Расшифрование алгоритмом Эль-Гамаля

Для расшифрования реализована следующая функция, представленная на рисунке 2.2.



Рисунок 2.2 – Функция расшифрования алгоритмом Эль-Гамаля

# Сравнение алгоритмов RSA и Эль-Гамаля

На рисунке 4.1. представлен график сравнения скорости зашифрования и расшифрования в RSA и алгоритме Эль-Гамаля при примерно одинаковых ключах и при длине сообщения от 10 до 500 символов. Время измеряется в тиках процессора.

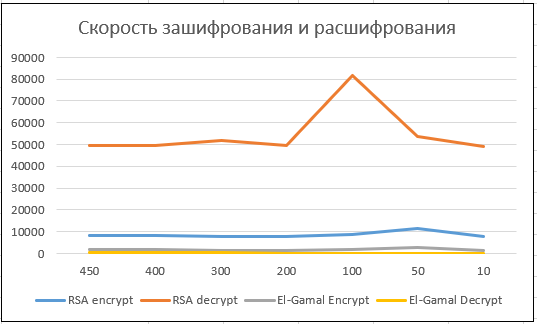


Рисунок 4.1 – График сравнения скорости алгоритмов

Как видно из графика, при малых значениях входных текстов (примерно 100 символов) скорость выполнения алгоритма RSA сильно падает, что связано с тем, что значение ключа остается большим. Алгоритм же Эль-Гамаля показывает примерно линейную зависимость.

# Вывод

В рамках данной лабораторной работы мы освоили и получили практические навыки разработки и использования приложений, реализующих асимметричные шифры RSA и Эль-Гамаля. Мы изучили принципы работы этих шифров и научились создавать программы, которые могут зашифровывать и расшифровывать данные, используя данные алгоритмы.

Лабораторная работа 11

# Алгоритм хеширования

Алгоритмы семейств MD и SHA, которые рассматриваются, включают следующие основные стадии обработки данных:

Расширение входного сообщения: Входное сообщение дополняется (расширяется) таким образом, чтобы его длина стала кратной 512 битам с остатком 448. Для этого к сообщению добавляется один бит "1", а затем добавляются нулевые биты, чтобы общая длина стала кратной 512. В последние 64 бита записывается двоичная длина исходного сообщения.

Разбивка расширенного сообщения на блоки: Расширенное сообщение разделяется на блоки длиной 512 битов. В каждом блоке может быть несколько подблоков длиной 32 или 64 бита, в зависимости от конкретного алгоритма.

Инициализация начальных констант: Задаются начальные значения для внутренних переменных и констант, которые будут использоваться в процессе обработки блоков данных.

Обработка сообщения поблочно: Каждый блок сообщения обрабатывается поэтапно с использованием циклических преобразований и операций, определенных конкретным алгоритмом.

Вывод результата: После обработки всех блоков данных получается итоговый хеш-значение или результат, который представляет собой фиксированную длину и может быть использован для проверки целостности данных или других криптографических целей.

Таким образом, данные алгоритмы работают в нескольких стадиях, включая расширение сообщения, разбиение на блоки, инициализацию констант, обработку блоков и вывод результата. Эти шаги позволяют обеспечить целостность и безопасность обработки данных при использовании этих алгоритмов.

Код функции для вычисления хеша на основе алгоритма SHA-1 представлен на рисунке 1.1.

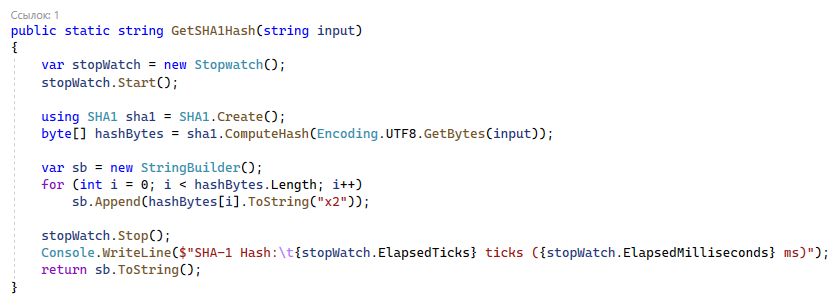


Рисунок 1.1 – Код функции вычисления хеша на основе алгоритма SHA-1

Аналогично, код функции для вычисления хеша на основе алгоритма MD5 представлен на рисунке 1.2.

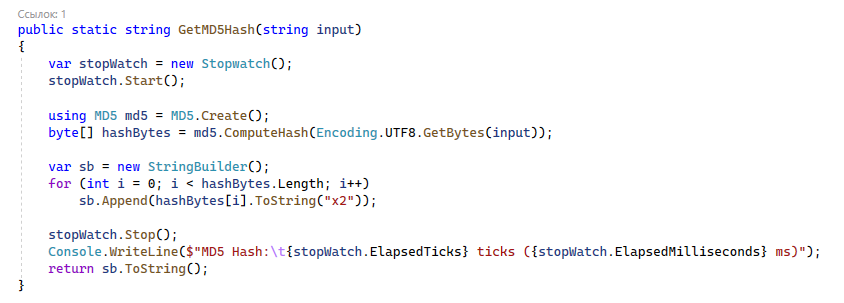


Рисунок 1.2 – Код функции вычисления хеша на основе алгоритма MD5

# Время вычисления хеша

Сравним время, с которым вычисляется хеш по алгоритмам SHA-1 и MD5. График сравнения представлен на рисунке 2.1.

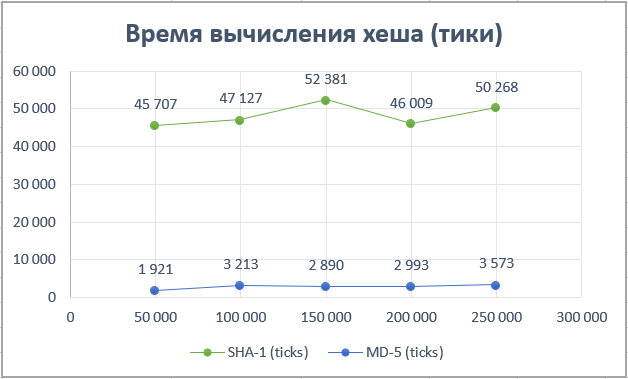


Рисунок 2.1 – Время вычисления хеша в тиках

Как видно из графика, не в зависимости от длины входного сообщения, время вычисления хеша всегда примерно одинаковое – примерно 45 тысяч тиков для SHA-1 и порядка 3 тысяч тиков для MD5, что соотносится с теоретическими предположениями, так как длина хеша в SHA-1 равна 160 битам, а в MD5 – 128 битам.

# Вывод

В данный лабораторной работе были изучены и приобретены практические навыки реализации и использования в криптографии алгоритмов хеширования SHA-1 и MD5.

Лабораторная работа 12

# Алгоритм RSA

При генерации ЭЦП (электронной цифровой подписи) с использованием алгоритма RSA можно рассмотреть две ситуации:

Сообщение Mo подписывается и передается в открытом (незашифрованном) виде: В этом случае подпись S вычисляется на основе соответствующих элементов тайного ключа отправителя, таких как dо (секретный экспонент) и nо (модуль). Передаваемое сообщение M' состоит из оригинального сообщения Mo и вычисленной подписи S. Приемная сторона (получатель) производит операцию расшифрования с использованием своего секретного ключа dп и модуля nп. Затем вычисляется хэш-значение от расшифрованного сообщения Мп. Если хэш-значение H(Mo) совпадает с полученным хэш-значением H(Mп), то подпись считается верифицированной.

Сообщение Mo подписывается и передается в зашифрованном виде: В этом случае перед основным процессом верификации подписи сообщение M' шифруется с использованием открытого ключа получателя (ep и np). Затем полученное зашифрованное сообщение передается получателю. На стороне получателя осуществляется расшифрование зашифрованного сообщения с использованием секретного ключа dп и модуля nп. После расшифрования, процесс верификации подписи продолжается так же, как и в первом случае, с вычислением хэш-значения и сравнением с хэш-значением оригинального сообщения Mo.

Таким образом, при генерации ЭЦП с использованием алгоритма RSA можно работать как с открытым, так и с зашифрованным сообщением, и в обоих случаях выполнять операции подписи, расшифрования и верификации подписи для обеспечения целостности и аутентичности передаваемых данных.

Код функции для генерации и верификации ЭЦП на основе алгоритма RSA представлен на рисунке 1.1.



Рисунок 1.1 – Код функции генерации и верификации ЭЦП на основе алгоритма RSA

# Алгоритм Эль-Гамаля

Ключевая информация отправителя: открытый ключ: *y*, *g* и *р*; тайный ключ: *х*. Чтобы подписать сообщение *Мо*, обладатель используемых для ЭЦП ключей должен выбрать случайное число *k*, взаимно простое с (*р* – 1). Затем вычисляются числа *а* и *b*, являющиеся цифровой подписью (*S* = {*a*, *b*}).

Для вычисления параметра *b* с помощью расширенного алгоритма Евклида решается следующее уравнение:



Получателю отправляется сообщение *М’* = *Мо*|| *S*. Для верификации подписи вычисляется хеш полученного сообщения *Н*(*Мп*) = *h*. Далее нужно убедиться, что выполняется равенство



Если равенство выполняется, подпись верифицируется.

Код функции для генерации и верификации ЭЦП на основе алгоритма Эль-Гамаля представлен на рисунке 2.1.

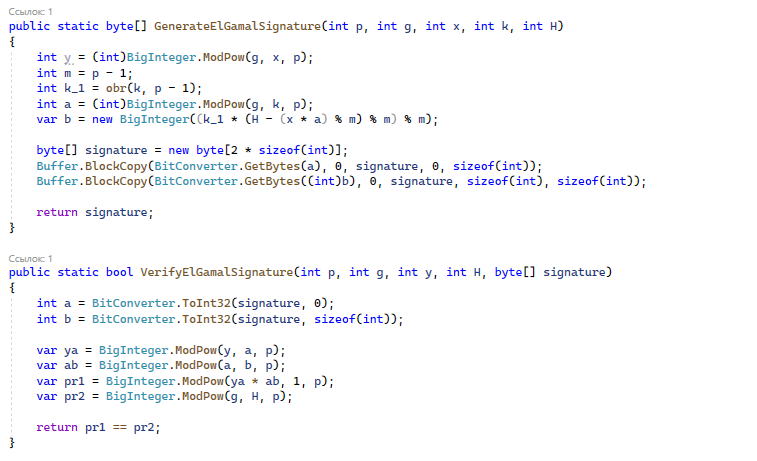


Рисунок 2.1 – Код функции генерации и верификации ЭЦП на основе алгоритма Эль-Гамаля

# Алгоритм Шнорра

Алгоритм Шнорра является вариантом алгоритма ЭЦП Эль-Гамаля.

Числа *p*, *g*, *q* являются открытыми и могут применяться группой пользователей. Выбирается число *х* < *q* (*х* является тайным ключом) и вычисляется последний элемент открытого ключа:



Секретный ключ имеет длину не менее 160 битов. Для подписи сообщения *Мо* выбирается случайное число *k* (1 < *k* < *q*) и вычисляется параметр *а*.



Получателю отправляются *М’* = *Мо* || *S*; *S* = {*h*, *b*}. Для проверки подписи получатель вычисляет



Затем он проверяет выполнение равенства: *h* = *Н*(*Mп* || *Х*). Подпись достоверна, если равенство выполняется.

Код функции для генерации и верификации ЭЦП на основе алгоритма Шнорра представлен на рисунке 3.1.

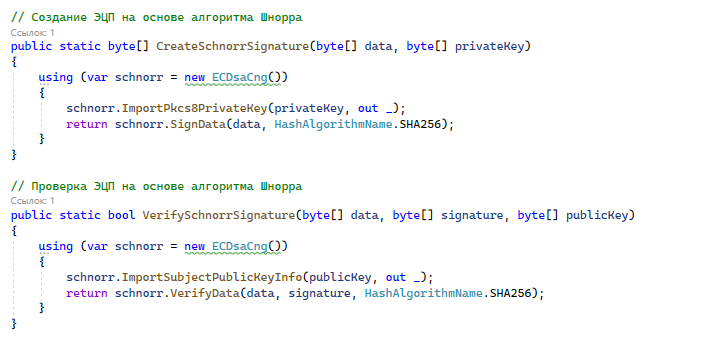


Рисунок 3.1 – Код функции генерации и верификации ЭЦП на основе алгоритма Шнорра

# Вывод

В данный лабораторной работе были изучены алгоритмы генерации и верификации электронной цифровой подписи на основе алгоритмов RSA, Эль-Гамаля и Шнорра.

Лабораторная работа 14

# Стеганографическая система

Стеганографическая система - это комплекс методов и средств, используемых для создания скрытого канала передачи или хранения информации. Основное отличие от криптографии заключается в том, что в стеганографии скрывается сам факт наличия передаваемого сообщения.

Основными компонентами стеганографической системы являются:

Контейнер (файл или документ), в котором размещается скрываемое сообщение. Контейнер служит в качестве скрытого канала передачи информации.

Тайное сообщение, которое нужно передать или сохранить, например, для подтверждения авторских прав.

Ключи или ключевая информация, контролирующие процесс сокрытия/извлечения сообщения, аналогично криптографическим ключам.

Стеганоконтейнер - контейнер с встроенным в него сообщением, который передается по открытому каналу связи.

Таким образом, стеганография позволяет скрыть не само сообщение, а факт его наличия в передаваемых данных.

# Метод наименее значимых битов

Большинство исследований в стеганографии сосредоточено на использовании изображений (и даже текста) в качестве контейнеров для скрытой передачи информации. Это связано с ограниченными способностями человеческого восприятия.

Одним из популярных методов является метод наименее значимых битов. Он основан на том, что младшие биты каждой составляющей цвета пикселя вносят незначительный вклад в итоговое изображение по сравнению со старшими битами. Для человеческого глаза замена одного или нескольких младших битов будет почти незаметна, поскольку человек способен различать лишь около 150 оттенков цвета.

Таким образом, стеганографические методы, основанные на изменении младших битов в изображениях, позволяют скрывать информацию без заметных искажений контейнера для человеческого восприятия. Это делает такие методы эффективными для создания скрытых каналов передачи данных.

В основе стеганографических преобразований лежит изображение, представленное на рисунке 1.1.



Рисунок 1.1 – Изначальное изображение для преобразований

Цветовая матрица наименее значащих битов для исходного изображения представлена на рисунке 1.2.



Рисунок 1.2 – Цветовая матрица НЗБ исходного изображения

Для реализации встраивания сообщения (по строкам) реализована следующая функция, представленная на рисунке 1.3.

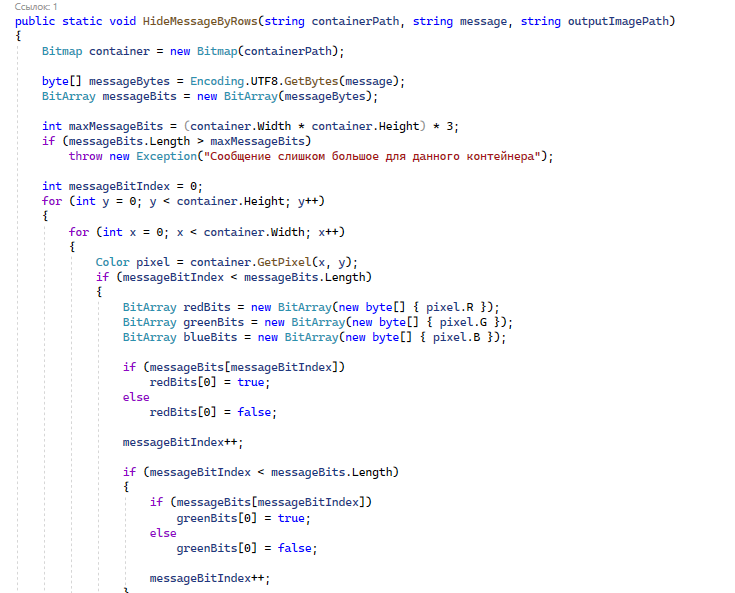


Рисунок 1.3 – Функция встраивания текста в контейнер

В результате получаем изображение, предсталвенное на рисунке 1.4.



Рисунок 1.4 – Изображение с встроенным текстом

Цветовая карта наименее значащих битов данного изображения представлена на рисунке 1.5.



Рисунок 1.5 – Цветовая матрица НЗБ с встроенным текстом

Из рисунка можно заметить, что на изображении не заметно никаких изменений. Однако, в левом верхнем углу цветовой матрицы можно заметить небольшое различие, в котором встроено наше сообщение.

Аналогичные результаты получаются для встраивания текста по столбцам, а не по строкам. Код данной функции представлен на рисунке 1.7.

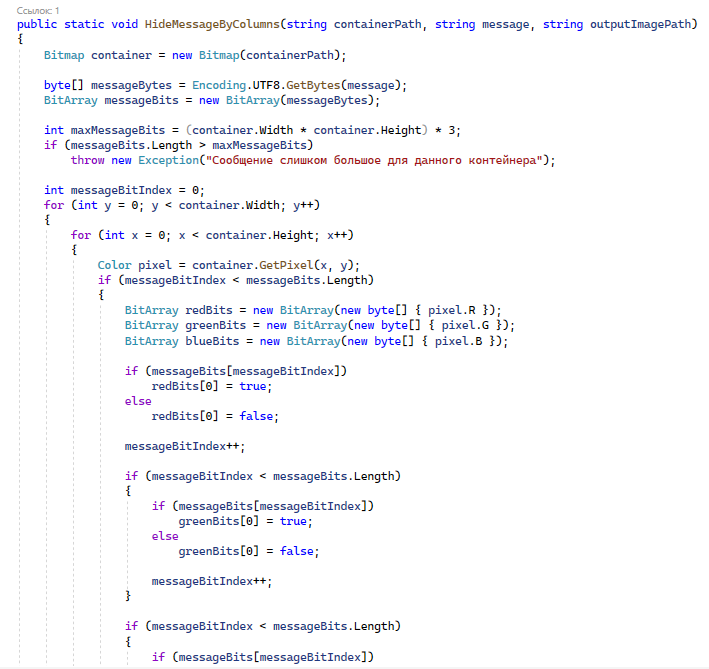


Рисунок 1.7 – Функция встраивания сообщения по столбцам

Аналогично, изменения возможно заметить только в цветовой матрице. Это является уязвимостью алгоритма, посколько по методу «Хи-квадрата» возможно выяснить, что в изображение стенографическим методом встроено некоторое сообщение. Для избежания этого необходимо использовать случайных разброс наименее значимых битов с информацией.

# Вывод

В данный лабораторной работе был изучен стеганографический метод встраивания и извлечения тайной информации с использованием электронного файла-контейнера на основе преобразования наименее значащих битов (НЗБ), а также приобретены практические навыки программной реализации данного метода.

Лабораторная работа 15

# Текстовая стеганография

Текстовая стеганография использует текстовые файлы в качестве контейнеров для скрытой передачи информации. Существует два основных подхода к текстовой стеганографии:

Синтаксические методы: они не затрагивают семантику текста, а вместо этого используют различные синтаксические изменения, такие как:

Изменение расстояния между строками

Изменение расстояния между словами

Изменение количества пробелов между словами

Изменения в шрифтах

Изменение интервала табуляции

Использование невидимых символов

Увеличение длины строк

Использование регистра букв

Использование специальных "нулевых" символов (Null Chipper)

Эти методы обеспечивают относительно низкую эффективность в плане объема скрываемой информации.

Лингвистические методы: они основаны на эквивалентной трансформации текста, сохраняющей его семантическое содержание. К ним относятся:

Метод синонимов

Метод переменной длины слова

Метод первой буквы

Мимикрия

Эти методы основаны на интерпретации текста как двоичного изображения и отличаются большей устойчивостью к различным искажениям.

Таким образом, текстовая стеганография предлагает широкий спектр методов, различающихся по сложности, эффективности и устойчивости к атакам.

# Программное средство

Необходимо реализовать метод изменения длины строки. Суть метода заключается в том, чтобы добавить в конце некоторых строк пробел, что не будет визуально заметно в большом тексте. Соотвественно, наличие двух пробелов шифруется единицей, а одного пробела – нулём.

Код функции, реализующей встраивание сообщения в документ-контейнер, представлен на рисунке 1.5.

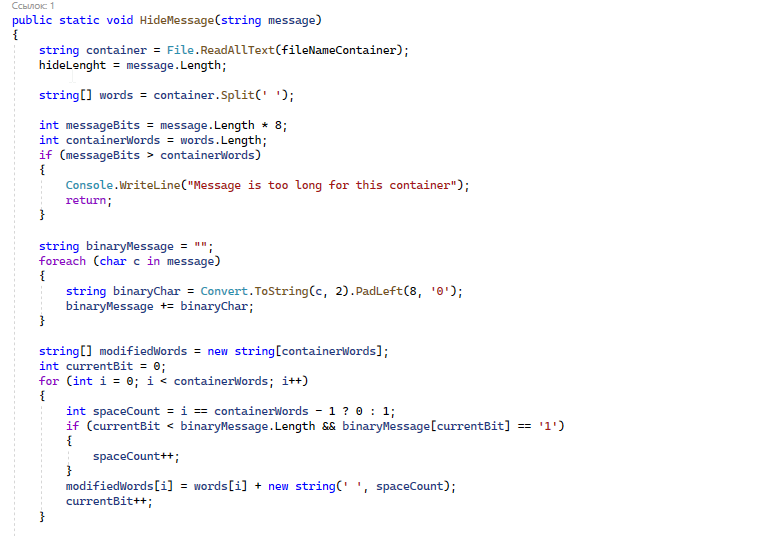


Рисунок 1.5 – Функция встраивания текста в контейнер

Вывод функции представлен на рисунке 1.6.

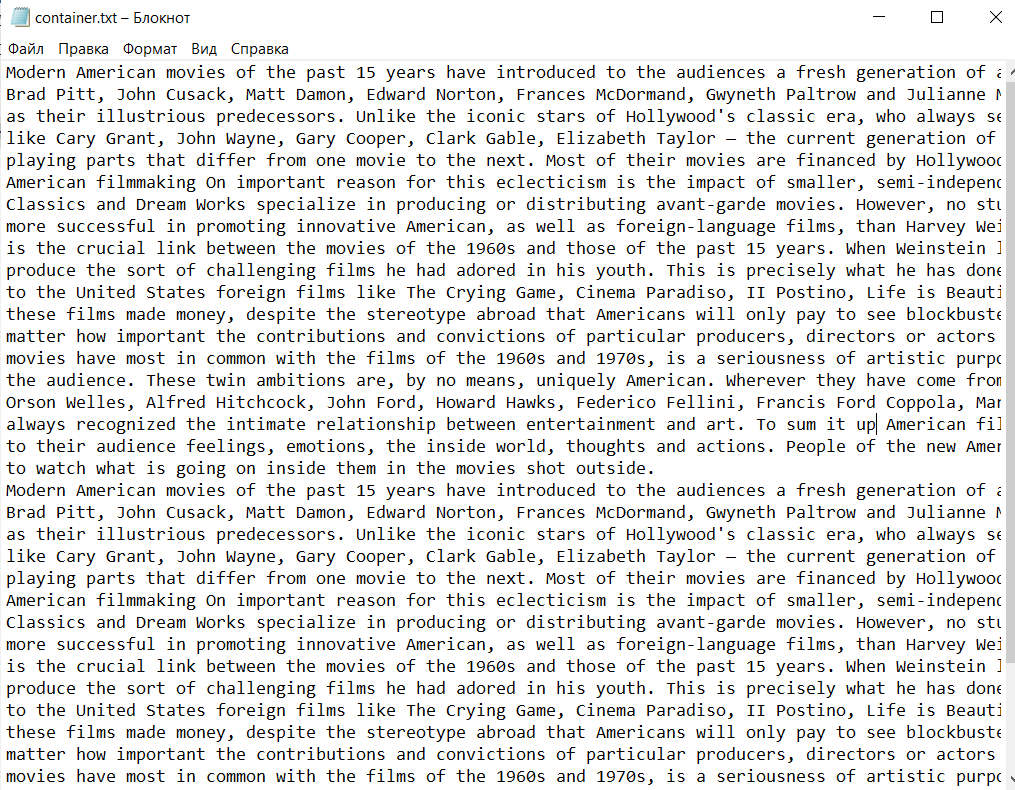


Рисунок 1.6 – Вывод функции встраивания текста в контейнер

Для извлечения текста из контейнера реализована следующая функция, представленная на рисунке 1.7.



Рисунок 1.7 – Функция извлечения текста из контейнера

Встраиваемое сообщение – «qw». Вывод функции извлечения текста из контейнера представлен на рисунке 1.8.



Рисунок 1.8 – Вывод функции извлечения текста из контейнера

Как видно из рисунка, извлеченный текст соответствует встроенному, что свидетельствует о корректном алгоритме встраивания и извлечения текста из текстового контейнера.

# Вывод

В рамках данной лабораторной работы были изучены различные методы текстовой стеганографии. Конкретно, были рассмотрены способы встраивания и извлечения скрытой информации с использованием текстовых файлов в качестве контейнеров. Кроме того, в ходе работы были приобретены практические навыки по программной реализации этих стеганографических методов.