

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

Дисциплина: «Разработка программных систем»

Отчёт по лабораторной работе № 2

**Задание:** создать интерфейс ICipher, который определяет методы поддержки шифрования строк. В качестве алгоритмов шифрования используются ROT13 и AES.

**Выполнил студент группы №485:**

Зобнин Илья Михайлович

**Проверили:**

Иван Григорьевич Корниенко

Алексей Константинович Федин

Санкт-Петербург

2020

# Постановка задачи

Создать интерфейс ICipher, который определяет методы поддержки шифрования строк. В интерфейсе объявить два метода Encode() и Decode(), которые используются для шифрования и дешифрования строк, соответственно. Реализовать 2 класса реализующих данный интерфейс.

В качестве алгоритмов используются ROT13 и AES.

# Особые ситуации

* Шифрование ROT13 поддерживает только символы латиницы. Остальные символы останутся неизменными.
* Если при указании пути к файлу для считывания текста пользователь указал несуществующий файл или файл с запрещённым именем, программа попросит ввести путь заново.
* Если пользователь попытается сохранить результат работы программы в уже существующий файл с данными, программа оповестит пользователя и предложит перезаписать файл либо же указать другой файл.

# Математические методы и алгоритмы решения задач

Согласно постановке задачи для составления программы будут использоваться два алгоритма шифрования — ROT13 и AES.

Для шифрования текста необходимо сначала преобразовать его в последовательность байт. Для этой цели рассматривались такие кодировки как: ASCII, UTF16 и UTF8. ASCII вмещает в себя лишь 128 символов, что при кодировании, например, символов кириллицы вызывает переполнение, и алгоритмы шифрования могут повести себя непредсказуемо. UTF16 и UTF8 кодируют символы согласно таблице Юникода, однако, UTF16 один символ преобразует минимум в два байта. При кодировании символов латиницы второй байт всегда равен 0 и не используется, поэтому UTF16 в этом случае лишь в два раза увеличивает размер массива байтов и время на шифрование. UTF8 лишён этой проблемы, а также из-за Юникода проблемы переполнения. Поэтому для ROT13 и для AES была использована кодировка UTF8.

## **ROT13**

ROT13 – алгоритм шифрования, при котором символы сдвигаются на 13 позиций, образуя тем самым новую строку. ROT13 является обратным себе методом шифрования, поэтому блок-схема его алгоритма (Рисунок 1) применяется и для дешифровки. Также этот алгоритм предназначен только для букв латиницы, остальные символы будут проигнорированы.

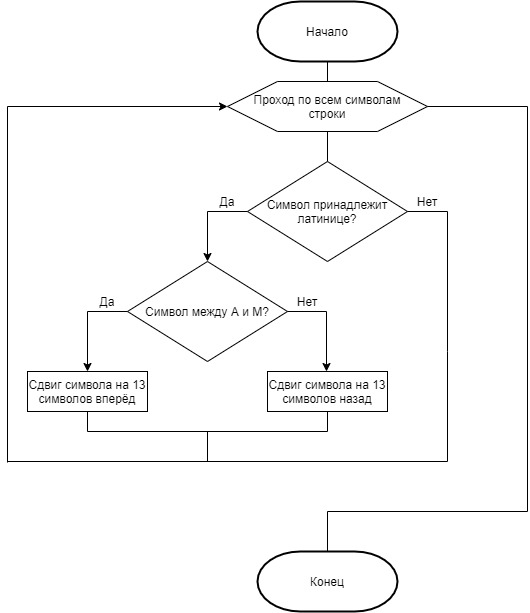


Рисунок 1 – Блок-схема алгоритма шифрования ROT13

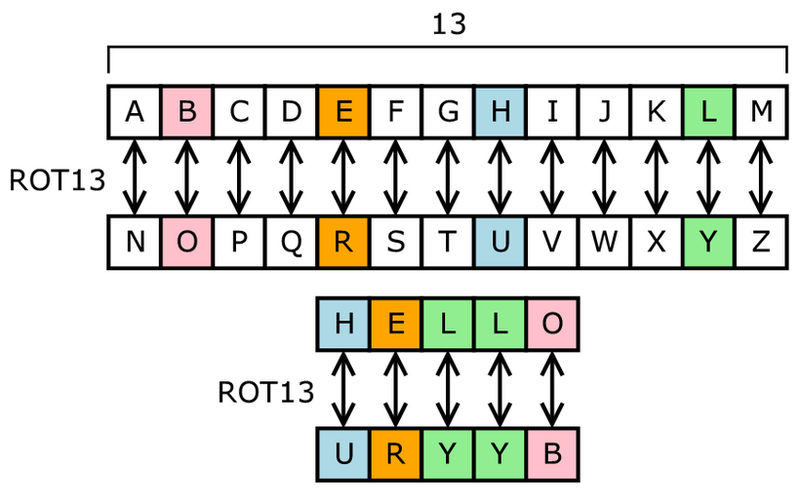


Рисунок 2 – Схема сдвигов символов ROT13 с примером

## **AES**

### **Шифрование**

Из разновидностей реализации AES был выбран алгоритм с длиной ключа в 128 бит (16 байт), а также режимом шифрования ECB (Electronic Codebook), режим простой замены.

Суть этого режима заключается в разбиении текста на блоки байтов (в случае AES каждый блок имеет длину в 16 байт), их шифрование по одному и соединение в одну строку.

Чтобы разбить текст на блоки по 16 байт, его длина должна быть кратна 16. Если кратности нет, то его последовательность байт дополняется нулями, что называется «паддингом» (дополнением).

Хотя AES допускает длину ключа не только в 16 байт, но ещё и в 24 и 32, длина блока шифрования всегда равна 16 байтам. С увеличением длины ключа увеличивается количество раундов шифрования. То есть то, сколько раз будет шифроваться блок текста. Для 16 байт это 10 раундов, для 24 – 12, для 32 – 14.

Также в методе шифрования AES обычные операции сложения и умножения заменены на таковые в полях Галуа. Там операция сложения представляет собой исключающее ИЛИ, а умножение производится по следующему алгоритму:

1. Пусть A и B, где A=8710, B=13110.
2. Множители должны быть представлены в виде полиномов. То есть, например, число A. В двоичной системе это 101 0111. Полиномиальное представление: x6+x4+x2+x+1. Тогда B=1000 00112 или x7+x+1 как полином.
3. Далее полиномы перемножаются (x6+x4+x2+x+1)\*(x7+x+1)=x13+x11+x9++x8+x7+x7+x5+x3+x2+x+x6+x4+x2+x+1. Суммирование коэффициентов при одинаковых степенях производится по правилам сложения в поле, т.е. по модулю 2, например: 1\*x7+1\*x7=(1⊕1)\*x7=0\*x7. Получим C= =(x6+x4+x2+x+1)\*(x7+x+1)=x13+x11+x9+x8+x6+x5+x4+x3+1.
4. Вычислить остаток от деления на неприводимый полином (полином, делящийся только на себя и на единицу поля). Для алгоритма AES это: m=x8+x4+x3+x+1. Таким образом, деля C на m, получим остаток 403 – это и есть произведение 87\*131 в поле Галуа.

Однако впоследствии нам не понадобится перемножать такие большие числа. Чаще всего понадобится умножать n\*m, где n ∈ [0;255], а m = 2, 3, 9, 11, 13, 14. Эти таблицы представлены на Рисунках 3, 4, 5, 6, 7, 8 соответственно.



Рисунок 3 - Таблица умножения на 2 в поле Галуа



Рисунок 4 - Таблица умножения на 3 в поле Галуа



Рисунок 5 - Таблица умножения на 9 в поле Галуа

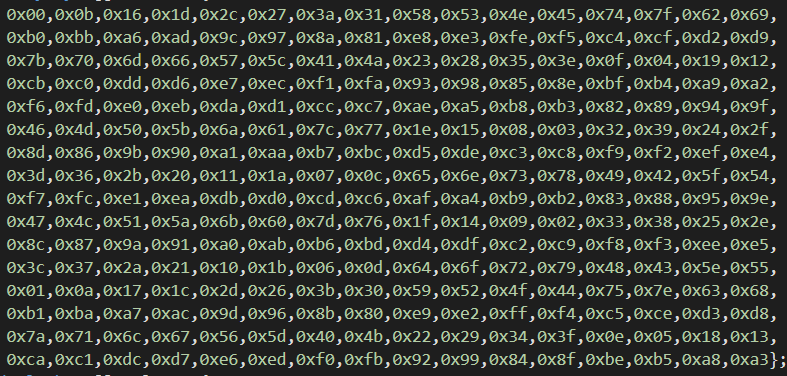


Рисунок 6 - Таблица умножения на 11 в поле Галуа

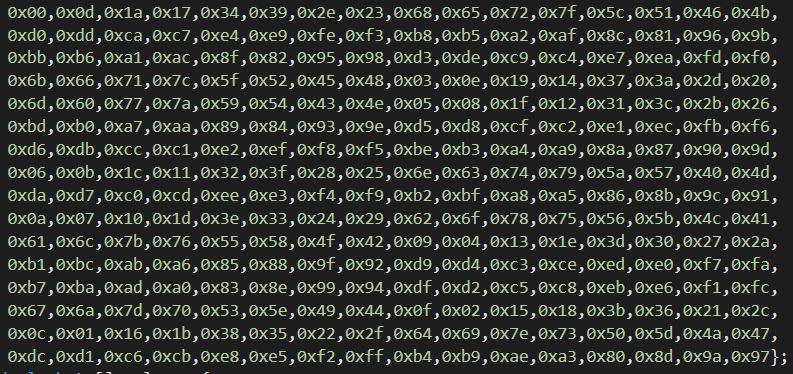


Рисунок 7 - Таблица умножения на 13 в поле Галуа

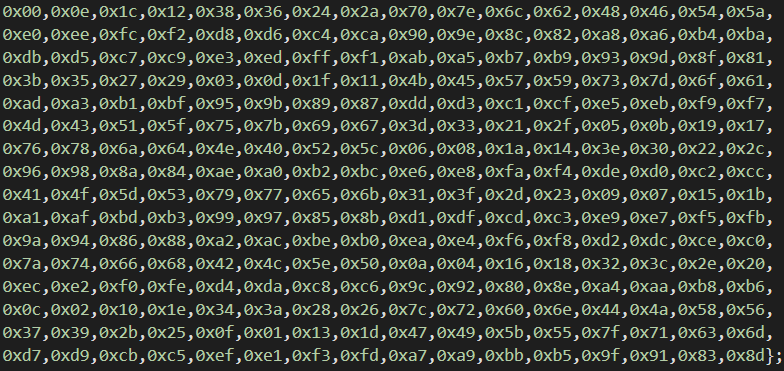


Рисунок 8 - Таблица умножения на 14 в поле Галуа

Итак, вначале берётся ключ, который ввёл пользователь, если надо, дополняется, путём кодировки преобразуется в последовательность из 16 байт и расширяется. Размер, до которого расширяется ключ, выбирается по следующей логике: вначале блок из 16 байтов текста преобразуется методом AddRoundKey, в который передаётся именно та 16-байтовая последовательность изначального ключа (пока что 16 байт), после идут 9 раундов, для AddRoundKey каждого из которых требуются другие, сгенерированные 16 байт (16+9\*16=160) и последний, 10 раунд требует ещё 16 байт (итого 160+16=176). 176 – размер расширенного ключа (ExpandedKey).

Блок-схема алгоритма расширения ключа (KeyExpansion) представлена на Рисунке 9.

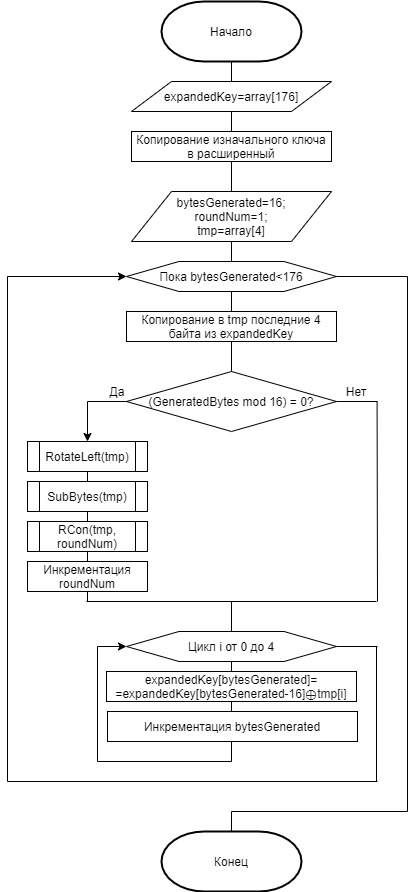


Рисунок 9 – Блок-схема алгоритма KeyExpansion

Функция RotateLeft представляет собой сдвиг байтов влево на один. Пример такого сдвига изображён на Рисунке 10.



Рисунок 10 – Пример работы функции RotateLeft

Функция SubBytes заменяет байты согласно таблице SBox в 16-ричной системе счисления, показанной на Рисунке 11. То есть array[i]=SBox[array[i]]. Предположим, что array[i]=50=3216. Тогда по таблице SBox array[i]=2316=3510.

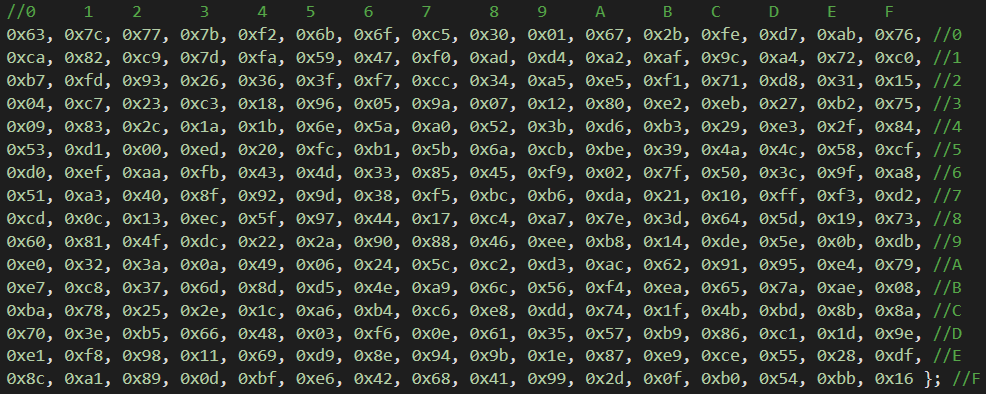


Рисунок 11 – Таблица замен SBox

Функция RCon делает следующее: array[i]=array[i] ⊕ Rcon[roundNum], где Rcon – таблица подстановок, а roundNum – номер раунда. Rcon – таблица степеней двойки в поле Галуа, представленная на Рисунке 12.



Рисунок 12 – Таблица подстановок Rcon

После расширения ключа идёт основная функция Encode, блок-схема которой представлена на Рисунке 13.

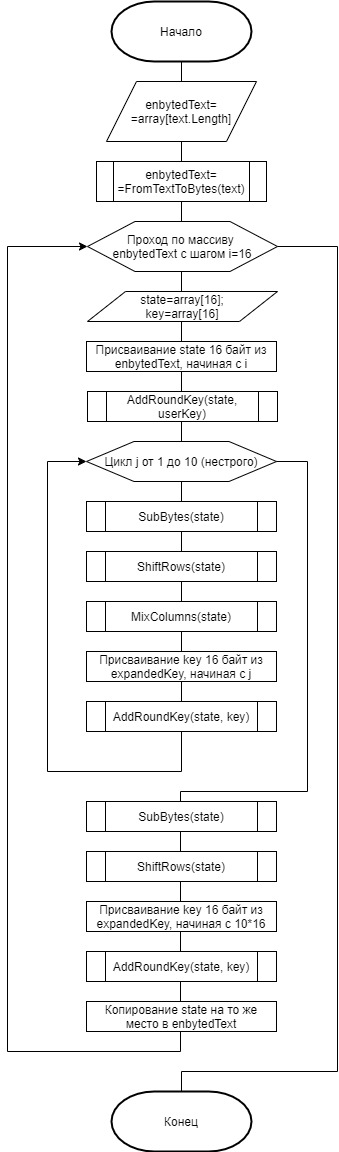


Рисунок 13 – Блок-схема алгоритма шифрования AES128

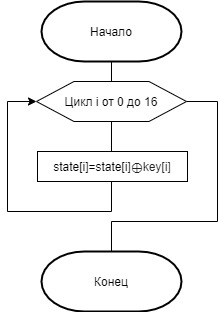
Блок-схема алгоритма AddRoundKey изображена на Рисунке 14. 

Рисунок 14 – Блок-схема метода AddRoundKey

Метод SubBytes был уже упомянут выше при описании алгоритма функции KeyExpansion.

В преобразовании ShiftRows байты в последних трех строках state циклически смещаются влево на различное число байт. Строка 1 (нумерация строк с нуля) смещается на один байт, строка 2 – на два байта, строка 3 – на три байта. На Рисунке 15 проиллюстрировано применение преобразования ShiftRows к state.

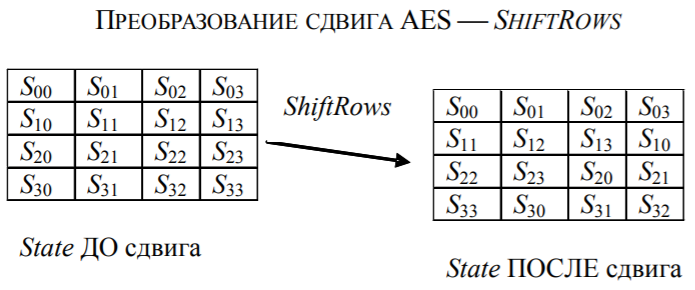


Рисунок 15 – Преобразование ShiftRows

Метод MixColumns помножает каждый столбец state на матрицу, изображённую на Рисунке 16.

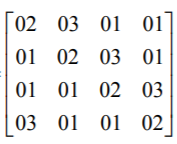


Рисунок 16 – Матрица умножения в методе MixColumns

В результате такого умножения байты столбца c {S0c, S1c, S2c, S3c} заменяются, соответственно, на байты:

S'0c = (2•S0c) ⊕ (3•S1c) ⊕ S2c ⊕ S3c;

S'1c = S0c ⊕ (2•S1c) ⊕ (3•S2c) ⊕ S3c;

S'2c = S0c ⊕ S1c ⊕ (2•S2c) ⊕ (3•S3c);

S'3c = (3•S0c) ⊕ S1c ⊕ S2c ⊕ (2•S3c).

Однако умножение происходит в полях Галуа, и так как здесь среди коэффициентов наблюдаются только 2 и 3, то можно воспользоваться таблицами на Рисунках 3 и 4.

### **Дешифрование**

Дешифрование, так же, как и шифрование, начинается с KeyExpansion. Этот метод не меняется. Однако реализация других методов слегка меняется, а их последовательность инвертируется. Блок-схема функции Decode изображена на Рисунке 17.

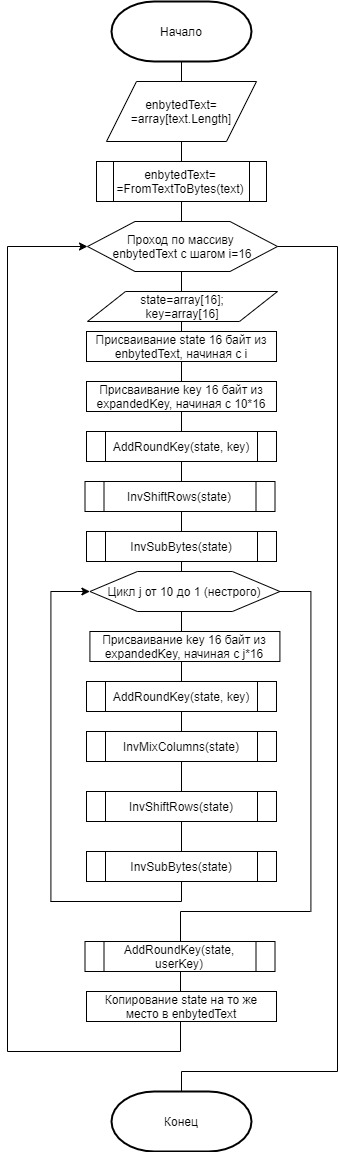


Рисунок 17 – Блок-схема алгоритма дешифрования AES128

Метод AddRoundKey остался неизменным и его алгоритм работы можно посмотреть в части Encode.

Преобразование InvShiftRows обратно преобразованию ShiftRows. Байты последних трех рядов массива state циклически сдвигаются вправо. Строка 1 (нумерация с нуля) смещается на 1 байт, строка 2 – на 2 байта, строка 3 – на 3 байта. На Рисунке 18 проиллюстрировано применение преобразования ShiftRows к state.

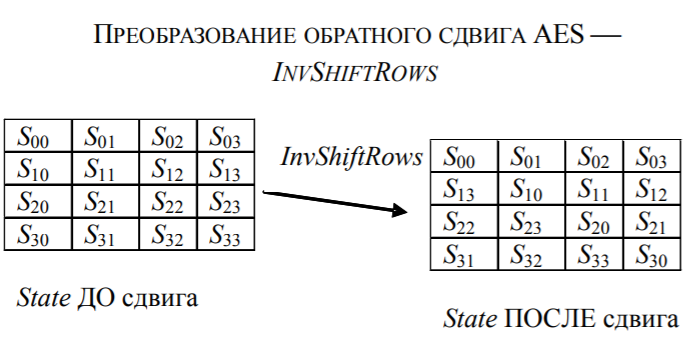


Рисунок 18 – Преобразование InvShiftRows

Преобразование InvSubBytes выполняет обратную замену байт с помощью обратной таблицы замен InvSbox, изображённой на Рисунке 19.

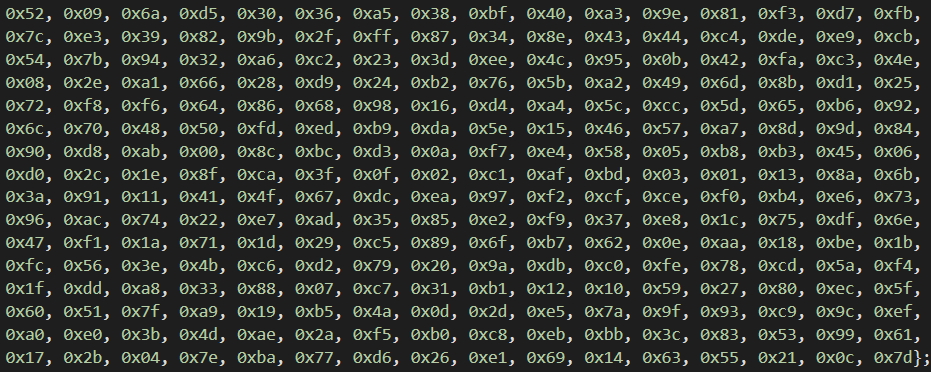


Рисунок 19 – Обратная таблица замен InvSBox

InvMixColumns преобразует столбцы state следующим образом:

S'0c = (14•S0c) ⊕ (11•S1c) ⊕ (13•S2c) ⊕ (9•S3c);

S'1c = (9•S0c) ⊕ (14•S1c) ⊕ (11•S2c) ⊕ (13•S3c);

S'2c = (13•S0c) ⊕ (9•S1c) ⊕(14•S2c) ⊕ (11•S3c);

S'3c = (11•S0c) ⊕ (13•S1c) ⊕ (9•S2c) ⊕ (14•S3c).

Здесь для умножения можно использовать Рисунки 5, 6, 7, 8.

# Форматы представления данных

Программа использует следующие переменные:

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| inputStr | string | Вводимая пользователем строка |
| resultStr | string | Результат шифрования/дешифрования |
| codeMethod | ICipher | Метод шифрования |
| bytesToCipher | byte[] | Последовательность байтов из текста |
| utf8Encoding | UTF8Encoding | Кодирование текста в последовательность байт |
| path | string | Путь к файлу для загрузки текста |

# Структура программы

Программа разделена на 12 классов, из которых один является основным и отвечает за запуск программы, один является интерфейсом, в котором объявляются методы Encode() и Decode(), а шесть оставшихся содержат в себе функции, необходимые для работы программы.

Класс Program.cs:

Таблица 2 – Функции, составляющие Класс Program.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Main | Приветствие пользователя и цикличность программы |

Класс ColorPrint.cs:

Таблица 3 – Функции, составляющие класс ColorPrint.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| PrintWithColor | Вывод цветного текста в консоль |
| ErrorPring | Вывод текста ошибки и предложения в консоль |

Класс Gets.cs:

Таблица 4 – Функции, составляющие класс Gets.cs

|  |  |
| --- | --- |
| MyParse | Преобразование текста в целое число с выводом сообщений об ошибках |
| GetInt | Получение целого числа из консоли |
| GetMenuItem | Получение пункта меню из консоли |
| AskYesNo | Получение «y» или «n» из консоли |
| ExtensionCheck | Проверка корректности расширения при вводе пользователем пути к файлу |
| GetFilePathForRead | Получение пути к файлу для чтения массива |
| GetFilePathForWrite | Получение пути к файлу для сохранения данных |
| GetCipher | Получение метода шифрования, основываясь на выбранном пункте меню |

Класс Menu.cs:

Таблица 5 – Функции, составляющие класс Menu.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| SetWayToInputStrOrExit | Выбор пользователем способа ввода текста или выход из программы |
| SetEncodeOrDecode | Выбор пользователя шифровать или дешифровать введённую строку |
| SetEncodeDecodeMethod | Выбор пользователем метода шифрования/дешифрования |
| SetWhatToSaveInFile | Выбор пользователем что сохранить в файл |

Класс ResultOuput.cs:

Таблица 6 – Функции, составляющие класс ResultOuput.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| OutputResult | Вывод в консоль изначальной строки и результата, вызов функции сохранения в файл |
| OutputStr | Запись строки в файл |
| OutputOrigAndChangedStr | Запись изначальной строки и результата |

Класс StrInput.cs:

Таблица 7 – Функции, составляющие класс StrInput.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| ManualInput | Ввод текста пользователем через консоль |
| FileInput | Ввод текста из файла |
| GetStr | Вызов функции ввода текста на основе выбранного пункта меню |

Класс ICipher.cs:

Таблица 8 – Функции, составляющие класс ICipher.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Encode | Функция шифрования |
| Decode | Функция дешифрования |

Класс ROT13.cs:

Таблица 9 – Функции, составляющие класс ROT13.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Encode | Функция шифрования |
| Decode | Функция дешифрования |

Класс AES128.cs:

Таблица 10 – Функции, составляющие класс AES128.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| FromStringToBytes | Преобразование строки в последовательность байт |
| FromBytesToString | Преобразование последовательности байт в строку |

Класс KeyMethods.cs:

Таблица 11 – Функции, составляющие класс KeyMethods.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| SetKey | Запрос ключа у пользователя, преобразование ключа в последовательность байт |
| KeyExpansion | Расширение ключа до 176 байт |
| KeyExpansionCore | Вспомогательная функция для метода KeyExpansion |

Класс EncodeMethods.cs:

Таблица 12 – Функции, составляющие класс EncodeMethods.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Encode | Функция шифрования |
| AddRoundKey | Прибавление к блоку байт ключа |
| SubBytes | Замена байт |
| ShiftRows | Сдвиг строк влево |
| MixColumns | Перемешивание столбца |

Класс DecodeMethods.cs:

Таблица 13 – Функции, составляющие класс DecodeMethods.cs

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Decode | Функция дешифрования |
| InvMixColumns | Обратна MixColumns |
| InvShiftRows | Сдвиг строк вправо |
| InvSubBytes | Обратна SubBytes |

# Описание хода выполнения лабораторной работы

* При работе программы с файлами нужно было добавить проверки на валидность имени файла, а также на то, создан ли файл или нет при сохранении.
* При получении пользовательского ввода необходимо было добавить проверку, на его корректность.
* При выполнении лабораторной работы были созданы тесты для проверки правильности работы алгоритмов.

# Результат работы программы

В результате работы программа выводит местоположение искомого слова или же сообщение об отсутствии данного слова в исходном тексте.

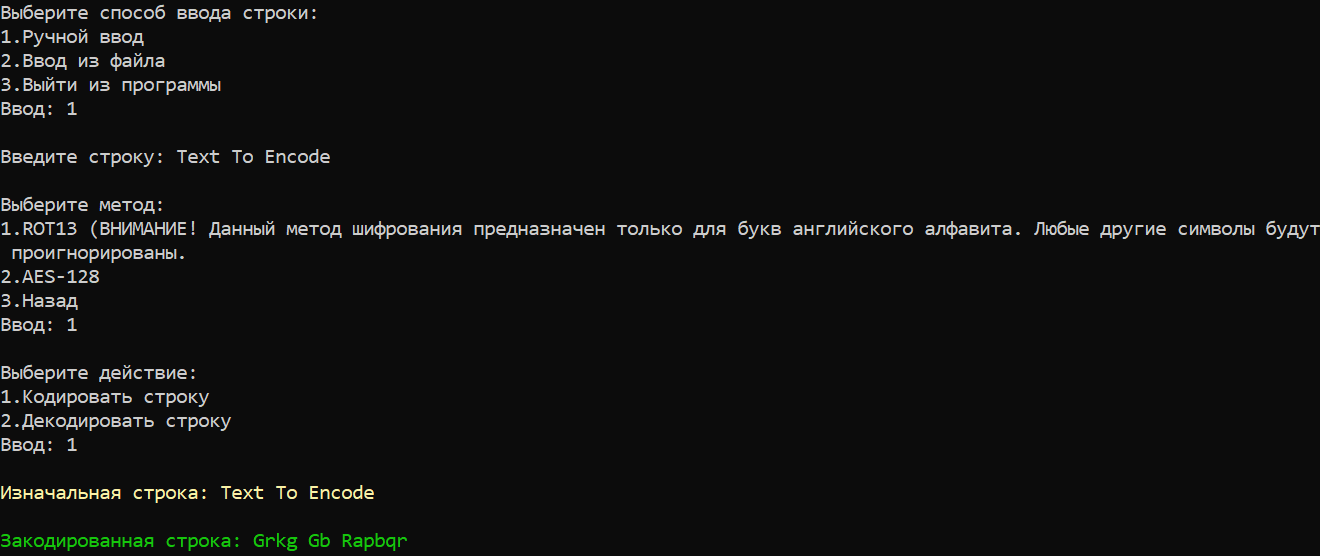


Рисунок 20 – Успешная работа программы, текст зашифрован (ROT13).

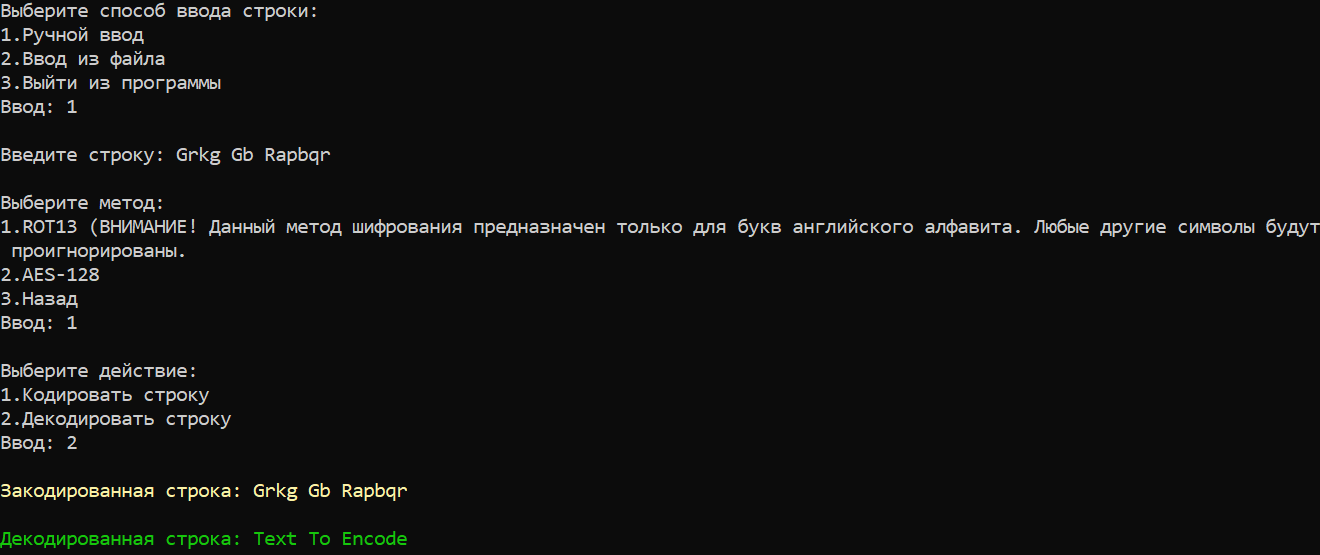


Рисунок 21 – Успешная работа программы, текст дешифрован (ROT13).

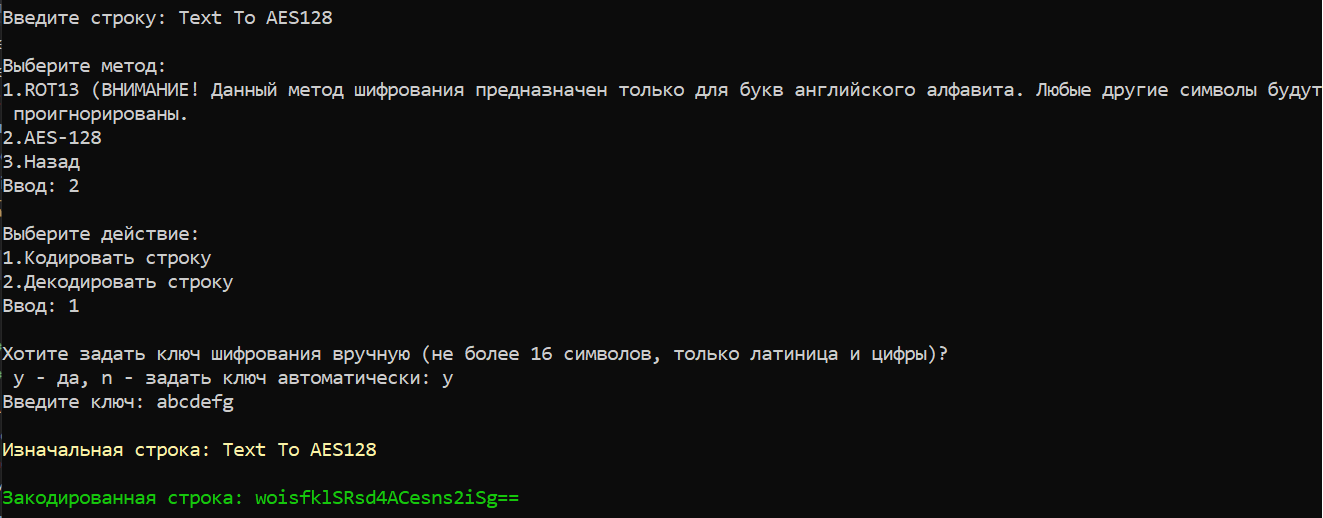


Рисунок 22 – Успешная работа программы, текст зашифрован (AES128).

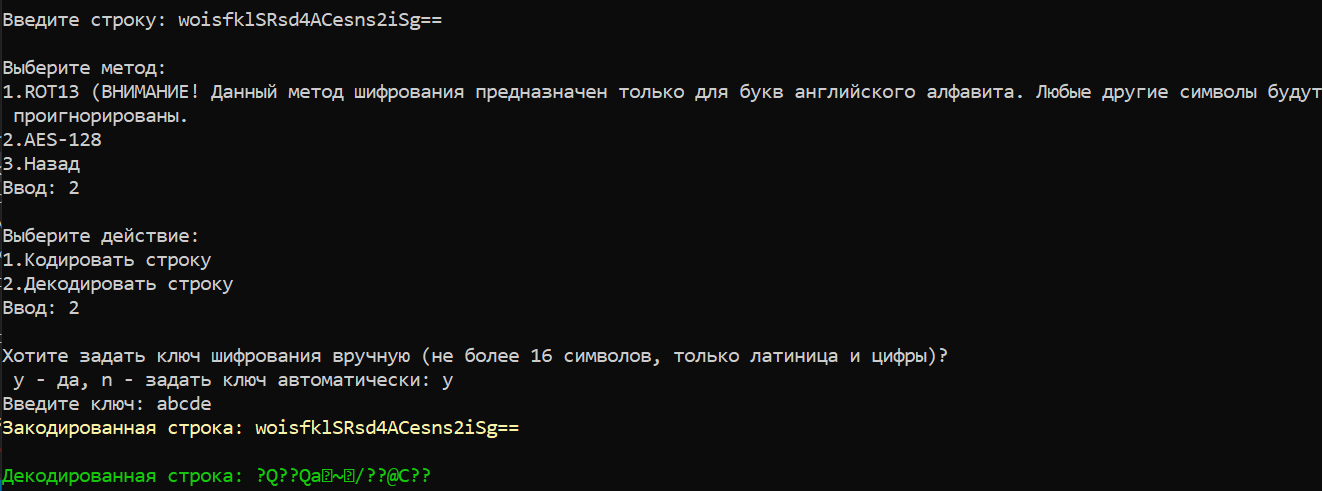


Рисунок 23 – Неверное дешифрование из-за неправильного ключа (AES128).

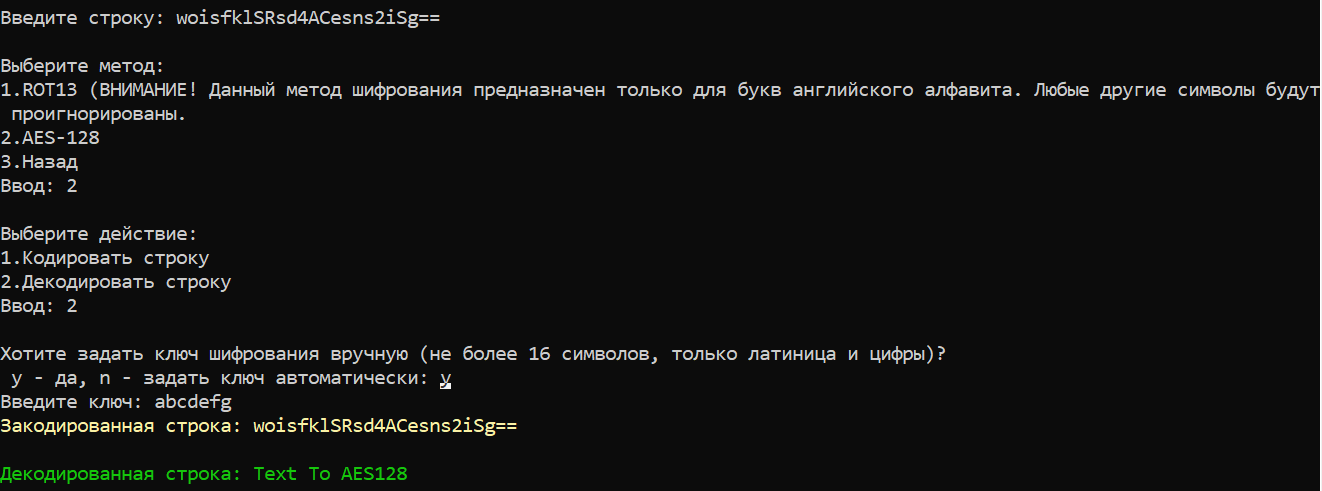


Рисунок 24 – Успешное дешифрование (AES128)

# Текст программы

**[--- Начало программы ---]**

**// Program.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

namespace Lab2

{

public static class Program

{

static string NL = Environment.NewLine;

static void Main()

{

Console.WriteLine("Здравствуйте. Эта программа кодирует или декодирует введённую строку " +

"по выбранному методу." + NL +

"На выбор предоставляются ROT13 (только символы английского алфавита) или AES-128." + NL +

"Программа студента 485 группы Зобнина Ильи." + NL);

string inputStr;

string resultStr;

while (true)

{

Menu menu = new Menu();

if (menu.WayToInputStrChoice == MenuItems.WayToInputStrOrExit.exitProgram)

break;

inputStr = StrInput.GetStr(menu.WayToInputStrChoice);

menu.SetEncodeDecodeMethod();

menu.SetEncodeOrDecode();

ICipher codeMethod = Gets.GetCipher(menu.ROT13AES128Choice);

if (menu.EncodeOrDecodeChoice == MenuItems.EncodeOrDecodeMenu.encode)

{

resultStr = codeMethod.Encode(inputStr);

ResultOutput.OutputResult(inputStr, resultStr, "Изначальная строка: ", "Закодированная строка: ", menu);

}

else

{

resultStr = codeMethod.Decode(inputStr);

ResultOutput.OutputResult(inputStr, resultStr, "Закодированная строка: ", "Декодированная строка: ", menu);

}

}

}

}

}

**// ColorPrint.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

namespace Lab2

{

static class ColorPrint

{

public static void PrintWithColor(ConsoleColor color, string message)

{

Console.ForegroundColor = color;

Console.Write(message);

Console.ResetColor();

}

public static void ErrorPrint(string errorMessage, string askMessage)

{

PrintWithColor(ConsoleColor.Red, errorMessage);

PrintWithColor(ConsoleColor.Yellow, askMessage);

}

}

}

**// Gets.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

using System.IO;

namespace Lab2

{

static class Gets

{

public static bool MyParse(string stringValue, string askMessage, out int value)

{

try

{

value = int.Parse(stringValue);

return true;

}

catch (ArgumentNullException)

{

ColorPrint.ErrorPrint("Число не было введено! ", askMessage);

}

catch (FormatException)

{

ColorPrint.ErrorPrint("Неверный формат! ", askMessage);

}

catch (OverflowException)

{

ColorPrint.ErrorPrint("Слишком большое или маленькое число! ", askMessage);

}

value = 0;

return false;

}

public static int GetInt(string askMessage)

{

int value;

string stringValue;

bool parseSuccess;

do

{

stringValue = Console.ReadLine();

parseSuccess = MyParse(stringValue, askMessage, out value);

} while (!parseSuccess);

return value;

}

public static EnumType GetMenuItem<EnumType>()

{

int value = GetInt("Введите пункт меню ещё раз: ");

while (!Enum.IsDefined(typeof(EnumType), value))

{

ColorPrint.ErrorPrint("Такого пункта меню нет! ", "Введите пункт меню ещё раз: ");

value = GetInt("Введите пункт меню ещё раз: ");

}

return (EnumType)Enum.ToObject(typeof(EnumType), value);

}

public static bool AskYesNo()

{

char yn;

while (true)

{

yn = Console.ReadKey().KeyChar;

Console.WriteLine();

if (yn != 'y' && yn != 'n')

{

ColorPrint.ErrorPrint("Неверный формат! ", "Введите y/n: ");

}

else

break;

}

return true ? yn == 'y' : false;

}

static bool ExtensionCheck(string filePath)

{

const int minPathLength = 4;

const string newValueAsk = "Введите путь ещё раз: ";

if (filePath.Length > minPathLength)

{

if (filePath.EndsWith(".txt"))

{

return true;

}

else

{

ColorPrint.ErrorPrint("Неверное расширение! ", newValueAsk);

return false;

}

}

else

{

ColorPrint.ErrorPrint("Слишком короткий путь к файлу! ", newValueAsk);

return false;

}

}

public static string GetFilePathForRead()

{

string path;

const string newValueAsk = "Введите путь ещё раз: ";

Console.Write("Введите путь к файлу для чтения текста (допускаются только .txt файлы): ");

while (true)

{

path = Console.ReadLine();

if (ExtensionCheck(path))

{

try

{

using (StreamReader streamReader = new StreamReader(path)) { }

break;

}

catch (FileNotFoundException)

{

ColorPrint.ErrorPrint("Файла по указанному пути не существует! ", newValueAsk);

}

catch (UnauthorizedAccessException)

{

ColorPrint.ErrorPrint("Доступ к файлу запрещён! ", newValueAsk);

}

catch (System.NotSupportedException)

{

ColorPrint.ErrorPrint("Запрещённое имя файла! ", newValueAsk);

}

}

}

return path;

}

public static string GetFilePathForWrite()

{

string path;

Console.Write("Введите путь для сохранения результата в файл (допускаются только .txt файлы): ");

while (true)

{

path = Console.ReadLine();

if (ExtensionCheck(path))

{

try

{

if (File.Exists(path))

{

ColorPrint.ErrorPrint("Файл уже существует. ", "y - перезаписать, n - указать другой путь: ");

if (AskYesNo())

{

using (StreamWriter streamWriter = new StreamWriter(path)) { }

break;

}

else

{

Console.Write("Введите путь к файлу: ");

continue;

}

}

else

{

using (StreamWriter streamWriter = new StreamWriter(path)) { }

return path;

}

}

catch

{

ColorPrint.ErrorPrint("Доступ к файлу запрещён или было использовано запрещённое имя! ", "Введите путь ещё раз: ");

}

}

}

return path;

}

public static ICipher GetCipher(MenuItems.ROT13AES128Menu ROT13AES128Choice)

{

switch (ROT13AES128Choice)

{

case MenuItems.ROT13AES128Menu.ROT13:

return new ROT13();

case MenuItems.ROT13AES128Menu.AES128:

return new AES128();

default:

return null;

}

}

}

}

**// Menu.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

namespace Lab2

{

class MenuItems

{

public enum WayToInputStrOrExit

{

manualInput = 1,

fileInput,

exitProgram

}

public enum EncodeOrDecodeMenu

{

encode = 1,

decode,

}

public enum ROT13AES128Menu

{

ROT13 = 1,

AES128,

back

}

public enum WhatToSaveInFile

{

originalStr = 1,

changedStr,

originalAndChangedStr,

saveNothing

}

}

class Menu

{

string NL = Environment.NewLine;

public MenuItems.WayToInputStrOrExit WayToInputStrChoice { get; private set; }

public MenuItems.EncodeOrDecodeMenu EncodeOrDecodeChoice { get; private set; }

public MenuItems.ROT13AES128Menu ROT13AES128Choice { get; private set; }

public MenuItems.WhatToSaveInFile WhatToSaveInFileChoice { get; private set; }

public void SetWayToInputStrOrExit()

{

Console.Write("Выберите способ ввода строки:" + NL +

"1.Ручной ввод" + NL +

"2.Ввод из файла" + NL +

"3.Выйти из программы" + NL +

"Ввод: ");

WayToInputStrChoice = Gets.GetMenuItem<MenuItems.WayToInputStrOrExit>();

Console.WriteLine();

}

public void SetEncodeOrDecode()

{

Console.Write("Выберите действие:" + NL +

"1.Кодировать строку" + NL +

"2.Декодировать строку" + NL +

"Ввод: ");

EncodeOrDecodeChoice = Gets.GetMenuItem<MenuItems.EncodeOrDecodeMenu>();

Console.WriteLine();

}

public void SetEncodeDecodeMethod()

{

while (true)

{

Console.Write("Выберите метод:" + NL +

"1.ROT13 (ВНИМАНИЕ! Данный метод шифрования предназначен только для букв английского алфавита. " +

"Любые другие символы будут проигнорированы." + NL +

"2.AES-128" + NL +

"3.Назад" + NL +

"Ввод: ");

ROT13AES128Choice = Gets.GetMenuItem<MenuItems.ROT13AES128Menu>();

Console.WriteLine();

if (ROT13AES128Choice == MenuItems.ROT13AES128Menu.back)

SetEncodeOrDecode();

else

break;

}

}

public void SetWhatToSaveInFile()

{

Console.Write("Что сохранить в файл:" + NL +

"1.Исходную строку" + NL +

"2.Преобразованную строку" + NL +

"3.Исходную и преобразованную строки" + NL +

"4.Ничего не сохранять" + NL +

"Ввод: ");

WhatToSaveInFileChoice = Gets.GetMenuItem<MenuItems.WhatToSaveInFile>();

Console.WriteLine();

}

public Menu()

{

SetWayToInputStrOrExit();

}

}

}

**// ResultOutput.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

using System.IO;

namespace Lab2

{

static class ResultOutput

{

static string NL = Environment.NewLine;

public static void OutputResult(string origStr, string resultStr, string firstMessage, string secondMessage, Menu menu)

{

ColorPrint.PrintWithColor(ConsoleColor.Yellow, firstMessage + origStr + NL + NL);

ColorPrint.PrintWithColor(ConsoleColor.Green, secondMessage + resultStr + NL + NL);

menu.SetWhatToSaveInFile();

switch (menu.WhatToSaveInFileChoice)

{

case MenuItems.WhatToSaveInFile.originalStr:

OutputStr(origStr);

ColorPrint.PrintWithColor(ConsoleColor.Green, "Успешно сохранено!" + NL + NL);

break;

case MenuItems.WhatToSaveInFile.changedStr:

OutputStr(resultStr);

ColorPrint.PrintWithColor(ConsoleColor.Green, "Успешно сохранено!" + NL + NL);

break;

case MenuItems.WhatToSaveInFile.originalAndChangedStr:

OutputOrigAndChangedStr(origStr, resultStr);

ColorPrint.PrintWithColor(ConsoleColor.Green, "Успешно сохранено!" + NL + NL);

break;

default:

break;

}

}

static void OutputStr(string str)

{

using (var streamWriter = new StreamWriter(Gets.GetFilePathForWrite()))

streamWriter.Write(str);

}

static void OutputOrigAndChangedStr(string origStr, string changedStr)

{

OutputStr(origStr + NL + changedStr);

}

}

}

**// StrInput.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

using System.IO;

namespace Lab2

{

static class StrInput

{

static string NL = Environment.NewLine;

static string ManualInput()

{

Console.Write("Введите строку: ");

string inputStr;

while (true)

{

inputStr = Console.ReadLine();

if (inputStr.Length != 0)

{

Console.WriteLine();

return inputStr;

}

else

ColorPrint.ErrorPrint("Вы не ввели строку! ", "Введите строку ещё раз: ");

}

}

static string FileInput()

{

while (true)

{

using (StreamReader streamReader = new StreamReader(Gets.GetFilePathForRead()))

{

string inputStr = streamReader.ReadToEnd();

if (inputStr.Length != 0)

{

ColorPrint.PrintWithColor(ConsoleColor.Green, "Строка введена: ");

Console.WriteLine(inputStr + NL);

return inputStr;

}

else

ColorPrint.PrintWithColor(ConsoleColor.Red, "Файл пуст или строку считать не удалось!" + NL);

}

}

}

public static string GetStr(MenuItems.WayToInputStrOrExit wayToInputStrChoice)

{

switch (wayToInputStrChoice)

{

case MenuItems.WayToInputStrOrExit.manualInput:

return ManualInput();

case MenuItems.WayToInputStrOrExit.fileInput:

return FileInput();

default:

return null;

}

}

}

}

**// ROT13.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System.Text;

namespace Lab2

{

public class ROT13 : ICipher

{

const byte symbolA = 65;

const byte symbolM = 77;

const byte symbolN = 78;

const byte symbolZ = 90;

const byte symbola = 97;

const byte symbolm = 109;

const byte symboln = 110;

const byte symbolz = 122;

const byte shift = 13;

public string Encode(string textToEncode)

{

//Преобразование строки в последовательность байт

var utf8Encoding = new UTF8Encoding();

byte[] encodedBytes = utf8Encoding.GetBytes(textToEncode);

//Шифрование

for (int i = 0; i < encodedBytes.Length; ++i)

{

//сдвиг букв верхнего регистра

if (encodedBytes[i] >= symbolA && encodedBytes[i] <= symbolM)

encodedBytes[i] += shift;

else if (encodedBytes[i] >= symbolN && encodedBytes[i] <= symbolZ)

encodedBytes[i] -= shift;

//сдвиг букв нижнего регистра

if (encodedBytes[i] >= symbola && encodedBytes[i] <= symbolm)

encodedBytes[i] += shift;

else if (encodedBytes[i] >= symboln && encodedBytes[i] <= symbolz)

encodedBytes[i] -= shift;

}

return utf8Encoding.GetString(encodedBytes);

}

public string Decode(string textToDecode)

{

return Encode(textToDecode); //так как функция ROT13 является обратной для самой себя, то её можно

//использовать и для дешифровки

}

}

}

**// AES128.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System.Linq;

using System.Text;

namespace Lab2

{

public partial class AES128 : ICipher

{

const int blockSize = 128 / 8; // величина блока шифрования

const int keySize = blockSize; // величина ключа, равна блоку шифрования

const int numberOfRounds = 9; // количество раундов шифрования за исключением последнего, 10

public AES128()

{

SetKey();

expandedKey = KeyExpansion();

}

// преобразование строки в последовательность байт

byte[] FromStringToBytes(string text)

{

var utf8Encoding = new UTF8Encoding();

byte[] textBytes = utf8Encoding.GetBytes(text);

// паддинг байтов

while (textBytes.Length % 16 != 0)

textBytes = textBytes.Append<byte>(0).ToArray();

return textBytes;

}

// преобразование последовательности байт обратно в строку

string FromBytesToString(byte[] bytes)

{

var uft8Encoding = new UTF8Encoding();

// анпаддинг байтов

var unpaddedBytes = bytes.Where(i => i != 0).ToArray();

return uft8Encoding.GetString(unpaddedBytes);

}

}

}

**// KeyMethods.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

//#define UnitTest

using System;

using System.Text.RegularExpressions;

namespace Lab2

{

public partial class AES128 : ICipher

{

readonly string NL = Environment.NewLine;

readonly byte[] defaultKey = new byte[keySize] {

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }; // стандартный ключ, если пользователь не захочет ввести собственный

byte[] key; // изначальный ключ шифрования

byte[] expandedKey; // ключ шифрования на 10 раундов (расширенный ключ)

readonly byte[] RCon = new byte[numberOfRounds + 2] { 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 };

// запрос ключа у пользователя или присвоение ключа по умолчанию, создание расширенного ключа

void SetKey()

{

#if (!UnitTest)

Console.Write("Хотите задать ключ шифрования вручную (не более 16 символов, только латиница и цифры)?" +

$"{NL} y - да, n - задать ключ автоматически: ");

if (Gets.AskYesNo())

{

Console.Write("Введите ключ: ");

while (true)

{

string userKey = Console.ReadLine();

if (userKey.IsRegular(keySize))

{

key = FromStringToBytes(userKey);

break;

}

}

}

else

key = defaultKey;

#else

key = defaultKey;

#endif

}

// методы для расширения ключа

void KeyExpansionCore(byte[] keyBytes, int roundNumber)

{

// сдвиг влево

byte tmp = keyBytes[0];

keyBytes[0] = keyBytes[1];

keyBytes[1] = keyBytes[2];

keyBytes[2] = keyBytes[3];

keyBytes[3] = tmp;

SubBytes(keyBytes);

keyBytes[0] ^= RCon[roundNumber];

}

// расширение ключа

byte[] KeyExpansion()

{

// первые 16 байт - начальный ключ

byte[] expandedKey = new byte[(numberOfRounds + 2) \* keySize];

for (int i = 0; i < keySize; ++i)

expandedKey[i] = key[i];

// далее идёт расширения до ключа, нужного для метода AddRoundKey

int bytesGenerated = keySize;

int roundIteration = 1;

byte[] tmp = new byte[4];

while (bytesGenerated < (numberOfRounds + 2) \* keySize)

{

for (int i = 0; i < 4; ++i)

tmp[i] = expandedKey[i + bytesGenerated - 4];

if (bytesGenerated % 16 == 0)

KeyExpansionCore(tmp, roundIteration++);

for (int i = 0; i < 4; ++i)

{

expandedKey[bytesGenerated] = (byte)(expandedKey[bytesGenerated - 16] ^ tmp[i]);

++bytesGenerated;

}

}

return expandedKey;

}

}

static class StringExpansion

{

public static bool IsRegular(this string key, int keySize)

{

Regex regex = new Regex(@"^[a-zA-z0-9]\*$");

if (key.Length == 0)

{

ColorPrint.ErrorPrint("Ключ не был введён! ", "Введите ключ: ");

return false;

}

if (key.Length >= keySize)

{

ColorPrint.ErrorPrint("Введён ключ слишком большой длины! ", "Введите ключ длиной не более 16 символов: ");

return false;

}

if (!regex.IsMatch(key))

{

ColorPrint.ErrorPrint("В ключе содержатся запрещённые символы! ", "Введите ключ, в котором содержатся только латиница и цифры: ");

return false;

}

return true;

}

}

}

**// EncodeMethods.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

using System.Linq;

namespace Lab2

{

public partial class AES128 : ICipher

{

// метод шифрования

readonly byte[] sBox = new byte[256] {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0

0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1

0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3

0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4

0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5

0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6

0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7

0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8

0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9

0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A

0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B

0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C

0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D

0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E

0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 }; //F // таблица для замен байтов метода SubBytes

readonly byte[] mul2 = {

0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,

0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,

0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,

0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,

0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,

0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,

0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,

0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,

0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,

0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,

0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,

0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,

0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,

0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,

0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,

0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5 }; // таблица умножения любого числа на 2 в поле Галуа

readonly byte[] mul3 = {

0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,

0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,

0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,

0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,

0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,

0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,

0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,

0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,

0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,

0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,

0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,

0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,

0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,

0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,

0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,

0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a

}; // таблица умножения любого числа на 3 в поле Галуа

public string Encode(string textToEncode)

{

string encodedText;

byte[] bytesToCipher = FromStringToBytes(textToEncode);

// шифрование

for (int i = 0; i < bytesToCipher.Length; i += blockSize)

{

byte[] state = new byte[blockSize];

Array.Copy(bytesToCipher, i, state, 0, blockSize);

AddRoundKey(state, key);

// первые 9 раундов

for (int j = 0; j < numberOfRounds; ++j)

{

SubBytes(state);

ShiftRows(ref state);

MixColumns(ref state);

AddRoundKey(state, expandedKey.Skip(blockSize \* (j + 1)).ToArray());

}

// последний 10 раунд

SubBytes(state);

ShiftRows(ref state);

AddRoundKey(state, expandedKey.Skip(blockSize \* 10).ToArray());

Array.Copy(state, 0, bytesToCipher, i, blockSize);

}

Console.WriteLine();

encodedText = Convert.ToBase64String(bytesToCipher);

return encodedText;

}

void AddRoundKey(byte[] state, byte[] roundKey)

{

for (int i = 0; i < state.Length; ++i)

state[i] ^= roundKey[i];

}

void SubBytes(byte[] state)

{

for (int i = 0; i < state.Length; ++i)

state[i] = sBox[state[i]];

}

void ShiftRows(ref byte[] state)

{

byte[] tmp = new byte[state.Length];

tmp[0] = state[0];

tmp[1] = state[5];

tmp[2] = state[10];

tmp[3] = state[15];

tmp[4] = state[4];

tmp[5] = state[9];

tmp[6] = state[14];

tmp[7] = state[3];

tmp[8] = state[8];

tmp[9] = state[13];

tmp[10] = state[2];

tmp[11] = state[7];

tmp[12] = state[12];

tmp[13] = state[1];

tmp[14] = state[6];

tmp[15] = state[11];

state = tmp;

}

void MixColumns(ref byte[] state)

{

byte[] tmp = new byte[state.Length];

tmp[0] = (byte)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);

tmp[1] = (byte)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);

tmp[2] = (byte)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);

tmp[3] = (byte)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

tmp[4] = (byte)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);

tmp[5] = (byte)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);

tmp[6] = (byte)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);

tmp[7] = (byte)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

tmp[8] = (byte)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);

tmp[9] = (byte)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);

tmp[10] = (byte)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);

tmp[11] = (byte)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

tmp[12] = (byte)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);

tmp[13] = (byte)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);

tmp[14] = (byte)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);

tmp[15] = (byte)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

state = tmp;

}

}

}

**// DecodeMethods.cs**

**// Лабораторная работа №2.**

**// Студент группы 485, Зобнин Илья Михайлович. 2020 год**

using System;

using System.Linq;

namespace Lab2{

public partial class AES128 : ICipher

{

readonly byte[] mul9 = {0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65,0x7e,0x77,

0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5,0xee,0xe7,

0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e,0x45,0x4c,

0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce,0xd5,0xdc,

0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13,0x08,0x01,

0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83,0x98,0x91,

0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28,0x33,0x3a,

0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8,0xa3,0xaa,

0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89,0x92,0x9b,

0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19,0x02,0x0b,

0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2,0xa9,0xa0,

0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22,0x39,0x30,

0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff,0xe4,0xed,

0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f,0x74,0x7d,

0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4,0xdf,0xd6,

0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54,0x4f,0x46}; // таблица умножения любого числа на 9 в поле Галуа

readonly byte[] mul11 = {

0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f,0x62,0x69,

0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf,0xd2,0xd9,

0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f,0x04,0x19,0x12,

0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4,0xa9,0xa2,

0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89,0x94,0x9f,

0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39,0x24,0x2f,

0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2,0xef,0xe4,

0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42,0x5f,0x54,

0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88,0x95,0x9e,

0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38,0x25,0x2e,

0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3,0xee,0xe5,

0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43,0x5e,0x55,

0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e,0x63,0x68,

0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce,0xd3,0xd8,

0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05,0x18,0x13,

0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5,0xa8,0xa3}; // таблица умножения любого числа на 11 в поле Галуа

readonly byte[] mul13 = {

0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51,0x46,0x4b,

0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81,0x96,0x9b,

0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea,0xfd,0xf0,

0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a,0x2d,0x20,

0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c,0x2b,0x26,

0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec,0xfb,0xf6,

0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87,0x90,0x9d,

0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57,0x40,0x4d,

0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b,0x9c,0x91,

0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b,0x4c,0x41,

0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30,0x27,0x2a,

0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0,0xf7,0xfa,

0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6,0xf1,0xfc,

0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36,0x21,0x2c,

0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d,0x4a,0x47,

0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d,0x9a,0x97}; // таблица умножения любого числа на 13 в поле Галуа

readonly byte[] mul14 = {

0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46,0x54,0x5a,

0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6,0xb4,0xba,

0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d,0x8f,0x81,

0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d,0x6f,0x61,

0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb,0xf9,0xf7,

0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b,0x19,0x17,

0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30,0x22,0x2c,

0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0,0xc2,0xcc,

0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07,0x15,0x1b,

0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7,0xf5,0xfb,

0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc,0xce,0xc0,

0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c,0x2e,0x20,

0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa,0xb8,0xb6,

0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a,0x58,0x56,

0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71,0x63,0x6d,

0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f,0x91,0x83,0x8d}; // таблица умножения любого числа на 14 в поле Галуа

readonly byte[] invSBox = {

0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,

0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,

0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,

0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,

0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,

0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,

0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,

0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,

0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,

0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,

0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,

0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,

0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,

0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,

0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,

0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d}; // таблица, обратная к sBox для метода InvSubBytes

public string Decode(string textInBase64ToDecode)

{

string decodedText;

byte[] bytesToDecipher = Convert.FromBase64String(textInBase64ToDecode);

// дешифровка

for (int i = 0; i < bytesToDecipher.Length; i += blockSize)

{

byte[] state = new byte[blockSize];

Array.Copy(bytesToDecipher, i, state, 0, blockSize);

AddRoundKey(state, expandedKey.Skip(blockSize \* 10).ToArray());

InvShiftRows(ref state);

InvSubBytes(state);

// первые 9 раундов

for (int j = numberOfRounds - 1; j >= 0; --j)

{

AddRoundKey(state, expandedKey.Skip(blockSize \* (j + 1)).ToArray());

InvMixColumns(ref state);

InvShiftRows(ref state);

InvSubBytes(state);

}

// последний 10 раунд

AddRoundKey(state, key);

Array.Copy(state, 0, bytesToDecipher, i, blockSize);

}

decodedText = FromBytesToString(bytesToDecipher);

return decodedText;

}

void InvMixColumns(ref byte[] state)

{

// Sc0 = (14\*Sc0)^(11\*Sc1)^(13\*Sc2)^(9\*Sc3)

// Sc1 = (9\*Sc0)^(14\*Sc1)^(11\*Sc2)^(13\*Sc3)

// Sc2 = (13\*Sc0)^(9\*Sc1)^(14\*Sc2)^(11\*Sc3)

// Sc3 = (11\*Sc0)^(13\*Sc1)^(9\*Sc2)^(14\*Sc3)

byte[] tmp = new byte[state.Length];

tmp[0] = (byte)(mul14[state[0]] ^ mul11[state[1]] ^ mul13[state[2]] ^ mul9[state[3]]);

tmp[1] = (byte)(mul9[state[0]] ^ mul14[state[1]] ^ mul11[state[2]] ^ mul13[state[3]]);

tmp[2] = (byte)(mul13[state[0]] ^ mul9[state[1]] ^ mul14[state[2]] ^ mul11[state[3]]);

tmp[3] = (byte)(mul11[state[0]] ^ mul13[state[1]] ^ mul9[state[2]] ^ mul14[state[3]]);

tmp[4] = (byte)(mul14[state[4]] ^ mul11[state[5]] ^ mul13[state[6]] ^ mul9[state[7]]);

tmp[5] = (byte)(mul9[state[4]] ^ mul14[state[5]] ^ mul11[state[6]] ^ mul13[state[7]]);

tmp[6] = (byte)(mul13[state[4]] ^ mul9[state[5]] ^ mul14[state[6]] ^ mul11[state[7]]);

tmp[7] = (byte)(mul11[state[4]] ^ mul13[state[5]] ^ mul9[state[6]] ^ mul14[state[7]]);

tmp[8] = (byte)(mul14[state[8]] ^ mul11[state[9]] ^ mul13[state[10]] ^ mul9[state[11]]);

tmp[9] = (byte)(mul9[state[8]] ^ mul14[state[9]] ^ mul11[state[10]] ^ mul13[state[11]]);

tmp[10] = (byte)(mul13[state[8]] ^ mul9[state[9]] ^ mul14[state[10]] ^ mul11[state[11]]);

tmp[11] = (byte)(mul11[state[8]] ^ mul13[state[9]] ^ mul9[state[10]] ^ mul14[state[11]]);

tmp[12] = (byte)(mul14[state[12]] ^ mul11[state[13]] ^ mul13[state[14]] ^ mul9[state[15]]);

tmp[13] = (byte)(mul9[state[12]] ^ mul14[state[13]] ^ mul11[state[14]] ^ mul13[state[15]]);

tmp[14] = (byte)(mul13[state[12]] ^ mul9[state[13]] ^ mul14[state[14]] ^ mul11[state[15]]);

tmp[15] = (byte)(mul11[state[12]] ^ mul13[state[13]] ^ mul9[state[14]] ^ mul14[state[15]]);

state = tmp;

}

void InvShiftRows(ref byte[] state)

{

// 0 4 8 12 0 4 8 12 0 сдвигов вправо

// 1 5 9 13 --> 13 1 5 9 1 сдвиг вправо

// 2 6 10 14 10 14 2 6 2 сдвига вправо

// 3 7 11 15 7 11 15 3 3 сдвига вправо

byte[] tmp = new byte[state.Length];

tmp[0] = state[0];

tmp[1] = state[13];

tmp[2] = state[10];

tmp[3] = state[7];

tmp[4] = state[4];

tmp[5] = state[1];

tmp[6] = state[14];

tmp[7] = state[11];

tmp[8] = state[8];

tmp[9] = state[5];

tmp[10] = state[2];

tmp[11] = state[15];

tmp[12] = state[12];

tmp[13] = state[9];

tmp[14] = state[6];

tmp[15] = state[3];

state = tmp;

}

void InvSubBytes(byte[] state)

{

for (int i = 0; i < state.Length; ++i)

state[i] = invSBox[state[i]];

}

}

}

**[--- Конец программы ---]**