**Министерство науки и высшего образования Российской федерации**

**ФГБОУ ВО «Магнитогорский государственный технический университет им. Носова»**

Кафедра информатики и информационной безопасности

**КУРСОВАЯ РАБОТА**

по дисциплине «Криптографические методы защиты информации»

на тему: «Разработка программного обеспечения для шифрования и дешифрования текста на основе шифра ГОСТ 28147-89»

Исполнитель:   Шпак В.А. студент 4 курса, группа АИБ-17

Руководитель: Михайлова У.В. доцент каф. ИиИБ, к.т.н.

Работа допущена к защите "\_\_\_\_" \_\_\_\_\_\_\_ 2021г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работа защищена "\_\_\_\_" \_\_\_\_\_\_\_ 2021г. с оценкой \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Магнитогорск, 2021

**Министерство науки и высшего образования Российской федерации**

**ФГБОУ ВО «Магнитогорский государственный технический университет им. Носова»**

Кафедра информатики и информационной безопасности

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Тема: «Разработка программного обеспечения для шифрования и дешифрования текста на основе шифра ГОСТ 28147-89»

Студенту Шпак Виталию Алексеевичу

Исходные данные: Разработать Windows приложение, осуществляющее шифрование и расшифрование файлов и текстовых строк, вводимых пользователем. Использовать метод шифрования – ГОСТ 28147-89 в режиме простой замены.

Срок сдачи: «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021г

Руководитель: Михайлова У.В. /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

Задание получил: Шпак В.А. /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

Магнитогорск, 2021

Содержание

[Введение 4](#_Toc71543712)

[1 Техническое описание алгоритма ГОСТ 28147-89 5](#_Toc71543713)

[1.1 Режим простой замены 5](#_Toc71543714)

[1.2 Режим гаммирования 9](#_Toc71543715)

[1.3 Режим гаммирования с обратной связью 11](#_Toc71543716)

[1.4 Режим выработки имитовставки 12](#_Toc71543717)

[2 Реализация приложения для шифрования информации 14](#_Toc71543718)

[2.1 Общая структура приложения с описанием семейств классов 14](#_Toc71543719)

[2.2 Метод Main 17](#_Toc71543720)

[2.3 Реализация алгоритмов шифрования и расшифрования 17](#_Toc71543721)

[2.4 Семейство классов ввода-вывода 21](#_Toc71543722)

[2.5 Семейство разбора пользовательского ввода 26](#_Toc71543723)

[Заключение 31](#_Toc71543724)

[Список использованных источников 32](#_Toc71543725)

[Приложение А 33](#_Toc71543726)

# Введение

Шифрование – это математический процесс преобразования сообщения в вид, нечитаемый для всех, кроме того человека или устройства, у которого имеется ключ для «расшифровки» этого сообщения обратно в читаемый вид.

Шифрование является лучшей имеющейся в наличии технологией, для защиты данных от злоумышленников, правительств, поставщиков услуг. На текущий момент оно достигло такого уровня развития, что при корректном использовании его практически невозможно взломать.

Однако при шифрованной передаче данных будет защищено только содержимое общения, метаданные зашифрованы не будут. Например, при общении в сети, один пользователь может зашифровать общение с другим пользователем, но это шифрование не сможет скрыть

* Факт общения пользователей;
* Факт использования шифрования при общении;
* Другие виды информации, относящейся к вашему общению (местоположение, время и продолжительность общения).

В данной курсовой работе нужно спроектировать объектно-ориентированную структуру разрабатываемого приложения, разработать функционал классов и методы их взаимодействия. Следующий этап – это реализация спроектированного распределенного приложения для операционной системы Windows на языке программирования C#.

## Техническое описание алгоритма ГОСТ 28147-89

ГОСТ 28147-89 — симметричный блочный алгоритм шифрования с 256-битным ключом, оперирует блоками данных по 64 бита.

Этот алгоритм установлен в качестве единого стандарта криптографического преобразования для систем обработки информации в сетях электронных вычислительных машин, отдельных вычислительных комплексах и ЭВМ, который определяет правила шифрования данных и выработки имитовставки.

Алгоритм криптографического преобразования предназначен для аппаратной или программной реализации, удовлетворяет криптографическим требованиям и по своим возможностям не накладывает ограничений на степень секретности защищаемой информации.

В криптосхеме предусмотрены четыре вида работы:

* зашифрование (расшифрование) данных в режиме простой замены;
* зашифрование (расшифрование) данных в режиме гаммирования;
* зашифрование (расшифрование) данных в режиме гаммирования с обратной связью;
* режим выработки имитовставки.

### Режим простой замены

Криптосхема, реализующая алгоритм зашифрования в режиме простой замены, должна иметь вид, указанный на рис. 1.

Открытые данные, подлежащие зашифрованию, разбиваются на блоки по 64 бита длинной. Каждый блок делится пополам, формируя N1 и N2 блоки, длинной 32 бита каждый.

В ключевое запоминающее устройство (КЗУ) вводятся 256 бит ключа. Содержимое ключа разбивается на восемь 32-разрядных накопителя.

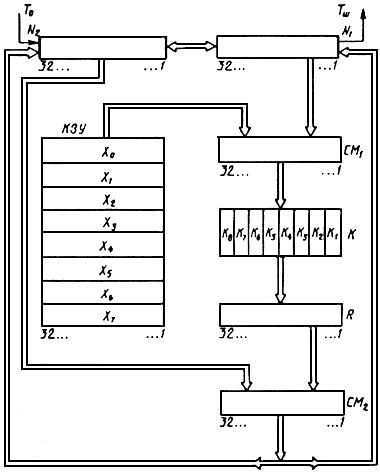


Рисунок 1 – Схема алгоритма ГОСТ 28147-89 в режиме простой замены

Алгоритм зашифрования 64-разрядного блока открытых данных в режиме простой замены состоит из 32 циклов.

В первом цикле начальное заполнение накопителя N1 суммируется по модулю 232 в сумматоре CM1 с заполнением накопителя X0, при этом заполнение накопителя N1 сохраняется.

Результат суммирования преобразуется в блоке подстановки K и полученный вектор поступает на вход регистра R, где циклически сдвигается на одиннадцать шагов в сторону старших разрядов. Результат сдвига суммируется поразрядно по модулю 2 в сумматоре CM2 с 32-разрядным заполнением накопителя N2. Полученный в CM2 результат записывается в N1, при этом старое заполнение N1 переписывается в N2. Первый цикл заканчивается.

Последующие циклы осуществляются аналогично, при этом во 2-м цикле из КЗУ считывается заполнение X1 в 3-м цикле из КЗУ считывается заполнение X2 и т.д., в 8-м цикле из КЗУ считывается заполнение X7. В циклах с 9-го по 16-й, а также в циклах с 17-го по 24-й заполнения из КЗУ считываются в том же порядке: X0, X1, X2, X3, X4, X5, X6, X7. В последних восьми циклах с 25-го по 32-й порядок считывания заполнений КЗУ обратный: X7, X6, X5, X4, X3, X2, X1, X0.

В 32 цикле результат из сумматора CM2 вводится в накопитель N2, а в накопителе N1 сохраняется старое заполнение. Полученные после 32-го цикла зашифрования заполнения накопителей N1 и N2 являются блоком зашифрованных данных, соответствующим блоку открытых данных.

Уравнения зашифрования в режиме простой замены имеют вид:

(1)

(2)

(3)

Где *a(0)* = *(a32(0), a31(0), … , a1(0))* - начальное заполнение N1 перед первым циклом зашифрования;

*b(0)* = *(b32(0), b31(0), … , b1(0))* - начальное заполнение N2 перед первым циклом зашифрования;

*a(j) = (a32(j), a31(j), … , a1(j))* - заполнение N1 после j-го цикла зашифрования;

*b(j) = (b32(j), b31(j), … , b1(j))* - заполнение N2 после j-го цикла зашифрования;

- поразрядное суммирование 32-разрядных векторов по модулю 2;

- суммирование 32-разрядных векторов по модулю 232;

R - операция циклического сдвига на одиннадцать шагов в сторону старших разрядов.

Криптосхема, реализующая алгоритм расшифрования в режиме простой замены, имеет тот же вид, что и при зашифровании. В КЗУ вводятся 256 бит того же ключа, на котором осуществлялось зашифрование. Зашифрованные данные, подлежащие расшифрованию, разбиты на блоки по 64 бита в каждом.

Расшифрование осуществляется по тому же алгоритму, что и зашифрование открытых данных, с тем изменением, что заполнения накопителей X0, X1, … , X7 считываются из КЗУ в циклах расшифрования в следующем порядке: X0, X1, X2, X3, X4, X5, X6, X7, X7, X6, X5, X4, X3, X2, X1, X0, X7, X6, X5, X4, X3, X2, X1, X0, X7, X6, X5, X4, X3, X2, X1, X0.

Уравнения расшифрования имеют вид:

(4)

(5)

(6)

Где *a(0)* = *(a32(0), a31(0), … , a1(0))* - начальное заполнение N1 перед первым циклом зашифрования;

*b(0)* = *(b32(0), b31(0), … , b1(0))* - начальное заполнение N2 перед первым циклом зашифрования;

*a(j) = (a32(j), a31(j), … , a1(j))* - заполнение N1 после j-го цикла зашифрования;

*b(j) = (b32(j), b31(j), … , b1(j))* - заполнение N2 после j-го цикла зашифрования;

- поразрядное суммирование 32-разрядных векторов по модулю 2;

- суммирование 32-разрядных векторов по модулю 232;

R - операция циклического сдвига на одиннадцать шагов в сторону старших разрядов.

Полученные после 32 циклов работы заполнения накопителей и составляют блок открытых данных.

### Режим гаммирования

Криптосхема, реализующая алгоритм зашифрования в режиме гаммирования, имеет вид, указанный на рис. 2.

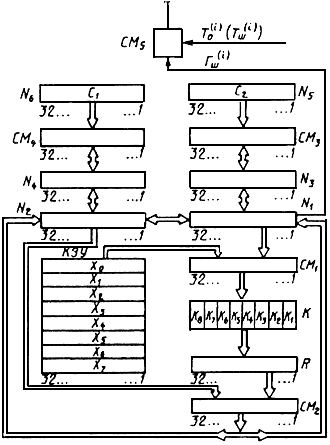


Рисунок 2 – Схема алгоритма ГОСТ 28147-89 в режиме гаммирования

Открытые данные, разбитые на 64-разрядные блоки *T0(1), T0(2) ... , T0(M-1), T0(M)*, зашифровываются в режиме гаммирования путем поразрядного суммирования по модулю 2 в сумматоре CM5 с гаммой шифра *ГШ*, которая вырабатывается блоками по 64 бита *ГШ = (ГШ(1), ГШ(2), … , ГШ(М-1), ГШ(М))*, где *М* - определяется объемом шифруемых данных.

В КЗУ вводятся 256 бит ключа. В накопители N1, N2 вводится 64-разрядная двоичная последовательность (синхропосылка) *S = (S1, S2, … , S64)*, являющаяся исходным заполнением этих накопителей для последующей выработки M блоков гаммы шифра. Синхропосылка вводится в N1 и N2 так, что значение *S1* вводится в 1-й разряд N1, значение *S2* вводится во 2-й разряд N1 и т.д., значение *S32* вводится в 32-й разряд N1; значение *S33* вводится в 1-й разряд N2, значение *S34* вводится во 2-й разряд N2 и т.д., значение *S64* вводится в 32-й разряд N2.

Исходное заполнение накопителей N1 и N2 (синхропосылка *S*) зашифровывается в режиме простой замены. Результат зашифрования *A(S) = (Y0, Z0)* переписывается в 32-разрядные накопители N3 и N4.

Заполнение накопителя N4 суммируется по модулю (232 - 1) в сумматоре CM4 с 32-разрядной константой C1 из накопителя N6, результат записывается в N4. Заполнение накопителя N3 суммируется по модулю 232 в сумматоре CM3 в 32-разрядной константой C2 из накопителя N5, результат записывается в N3.

Заполнение N3 переписывается в N1, а заполнение N4 переписывается в N2, при этом заполнение N3, N4 сохраняется.

Заполнение N1 и N2 зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение N1, N2, образует первый 64-разрядный блок гаммы шифра *ГШ(1)*, который суммируется поразрядно по модулю 2 в сумматоре CM5 с первым 64-разрядным блоком открытых данных. В результате суммирования получается 64-разрядный блок зашифрованных данных.

При расшифровании криптосхема имеет тот же вид, что и при зашифровании. В КЗУ вводятся 256 бит ключа, с помощью которого осуществлялось зашифрование данных. Синхропосылка S вводится в накопители N1 и N2 и осуществляется процесс выработки M блоков гаммы шифра. Блоки зашифрованных данных суммируются поразрядно по модулю 2 в сумматоре CM5 с блоками гаммы шифра, в результате получаются блоки открытых данных при этом последний блок расшифрованных данных может содержать меньше 64 разрядов.

### Режим гаммирования с обратной связью

Криптосхема, реализующая алгоритм зашифрования в режиме гаммирования с обратной связью, имеет вид, указанный на рис. 3.

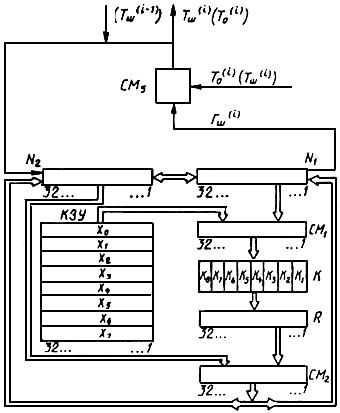


Рисунок 3 – Схема алгоритма ГОСТ 28147-89 в режиме гаммирования с обратной связью

Открытые данные, разбитые на 64-разрядные блоки T0(1), ..., T0(M), зашифровываются в режиме гаммирования с обратной связью путем поразрядного суммирования по модулю 2 в сумматоре CM5 с гаммой шифра ГШ, которая вырабатывается блоками по 64 бита, т.е. *ГШ = (ГШ(1), ГШ(2), ... , ГШ(M))*, где определяется объемом открытых данных, *ГШ(i)* - i-й 64-разрядный блок, *i [1, …, M].* Число двоичных разрядов в блоке *T0(M)* может быть меньше 64.

Так же, как и в ГОСТ 28147-89, В КЗУ вводятся 256 бит ключа. Синхропосылка *S = (S1, S2, … , S64)* из 64 бит вводится в N1 и N2.

Исходное заполнение N1 и N2 зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение N1 и N2 образует первый 64-разрядный блок гаммы шифра *ГШ(1) = A(S),* который суммируется поразрядно по модулю 2 в сумматоре с первым 64-разрядным блоком открытых данных *T0(1) = (t1(1), t2(1), … , t64(1)).* В результате получается 64-разрядный блок зашифрованных данных TШ(1). Блок зашифрованных данных *TШ(1)* одновременно является также исходным состоянием N1 и N2 для выработки второго блока гаммы шифра *ГШ(2)* и по обратной связи записывается в указанные накопители.

При расшифровании криптосхема имеет тот же вид, что и при зашифровании. Исходное заполнение N1 и N2 (синхропосылка *S*) зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение N1 и N2 образует первый блок гаммы шифра *ГШ(1) = A(S),* который суммируется поразрядно по модулю 2 в сумматоре CM5 с блоком зашифрованных данных *TШ(1).* В результате получается первый блок открытых данных *T0(1).*

Блок зашифрованных данных *TШ(1)* является исходным заполнением N1 и N2 для выработки второго блока гаммы шифра *ГШ(2).* Блок *TШ(1)* записывается в N1 и N2 так же, как и при шифровании в данном режиме. Полученное заполнение N1 и N2 зашифровывается в режиме простой замены, полученный в результате блок *ГШ(2)* суммируется поразрядно по модулю 2 в сумматоре CM5 со вторым блоком зашифрованных данных *TШ(2).* В результате получается блок открытых данных *T0(2).*

### Режим выработки имитовставки

Для обеспечения имитозашиты открытых данных, состоящих из *M* 64-разрядных блоков *T0(1), ..., T0(M), M ≥ 2*, вырабатывается дополнительный блок из l бит (имитовставка Иl). Процесс выработки имитовставки единообразен для всех режимов шифрования.

Первый блок открытых данных *T0(1) = (t1(1), t2(1), … , t64(1)) = (a1(1)(0), a2(1)(0), … , a32(1)(0), b1(1)(0), b2(1)(0), … , b32(1)(0))* записывается в накопители N1 и N2, при этом значение *t1(1) = a1(1)(0)* вводится в 1-й разряд N1, значение *t2(1) = a2(1)(0)* вводится во 2й разряд N1 и т.д.

Заполнение N1 и N2 подвергается преобразованию, соответствующему первым 16 циклам алгоритма зашифрования в режиме простой замены. В КЗУ при этом находится тот же ключ, которым зашифровываются блоки открытых данных *T0(1), T0(2), ... T0(M)* в соответствующие блоки зашифрованных данных *TШ(1), TШ(2), ... TШ(M).*

Полученное после 16 циклов работы заполнение N1 и N2, имеющее вид *(a1(1)(16), a2(1)(16), … , a32(1)(16), b1(1)(16), b2(1)(16), … , b32(1)(16)),* суммируется в CM5 по модулю 2 со вторым блоком *T0(2) = (t1(2), t2(2), … , t64(2)).* Результат суммирования заносится в N1 и N2 и подвергается преобразованию, соответствующему первым 16 циклам алгоритма зашифрования в режиме простой замены.

Полученное заполнение N1 и N2 суммируется в CM5 по модулю 2 с третьим блоком *T0(3)* и т.д. Имитовставка *Иl* передается по каналу связи или в память ЭВМ в конце зашифрованных данных, т.е. *TШ(1), TШ(2), ... TШ(M), Иl.*

Поступившие зашифрованные данные *TШ(1), TШ(2), ... TШ(M)* расшифровываются, из полученных блоков открытых данных *T0(1), T0(2), ... , T0(M)* аналогично предыдущему пункту вырабатывается имитовставка , которая затем сравнивается с имитовставкой , полученной вместе с зашифрованными данными из канала связи или из памяти ЭВМ. В случае несовпадения имитовставок полученные блоки открытых данных *T0(1), T0(2), ... , T0(M)* считают ложными.

Выработка имитовставки *Иl (Иl')* может производиться или перед зашифрованием (после расшифрования) всего сообщения, или параллельно с зашифрованием (расшифрованием) по блокам. Первые блоки открытых данных, которые участвуют в выработке имитовставки, могут содержать служебную информацию (адресную часть, отметку времени, синхропосылку и др.) и не зашифровываться.

## Реализация приложения для шифрования информации

### Общая структура приложения с описанием семейств классов

Реализованное в ходе выполнения данной курсовой работы приложение (полный листинг программы в приложении А) было спроектировано с использованием основных концепций ООП (инкапсуляция, наследование, полиморфизм). В соответствии с этими принципами, приложение содержит два семейства классов:

* Семейство классов ввода-вывода шифруемой и зашифрованной информации (рис. 4)
* Семейство разбора пользовательского ввода (рис. 5)

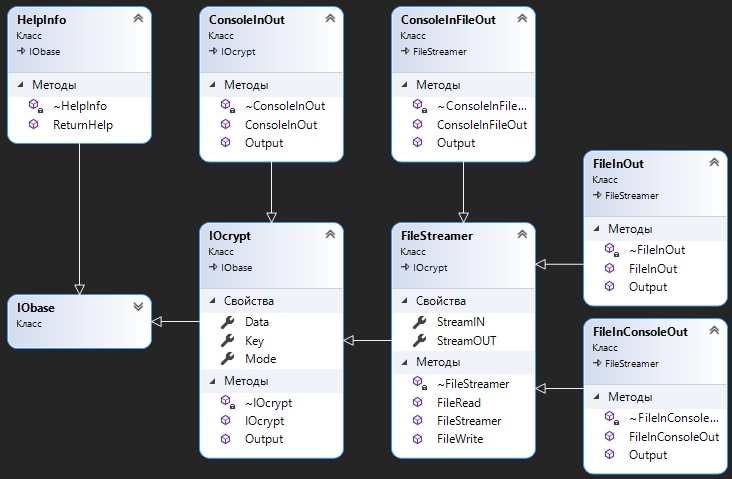


Рисунок 4 – Схема семейства ввода-вывода

В семействе классов ввода-вывода шифруемой и зашифрованной информации программист непосредственно взаимодействует с пятью дочерними классами, которые реализуют различные сценарии взаимодействия пользователя с приложением:

* Запрос справки о приложении
* Шифрование/расшифровка вводимой пользователем в консоль строки с выводом результата в консоль
* Шифрование/расшифровка вводимой пользователем в консоль строки с выводом результата в файл
* Шифрование/расшифровка файла с выводом результата в консоль
* Шифрование/расшифровка файла с выводом результата в файл

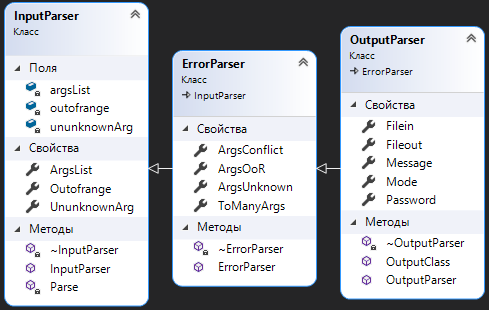


Рисунок 5 – Схема семейства классов разбора пользовательского ввода

Семейство разбора пользовательского ввода существует для анализа введенных пользователем аргументов, выявления ошибок (отсутствие или противоречие аргументов, излишнее их использование, или применение неизвестных аргументов и т.д.) и создание соответствующего объекта из семейства классов ввода-вывода шифруемой и зашифрованной информации для выполнения необходимого пользователю сценария работы приложения.

Так же в данном приложении есть классы, не выделенные в отдельные семейства либо по причине отсутствия необходимости расширения функционала (класс алгоритма шифрования ГОСТ 28147-89 на рис. 6), либо по причине того, что что он реализует вспомогательные функции (класс хранения аргументов команд, их параметры и количество упоминаний параметра в вводимой пользователем команде, рис. 7).

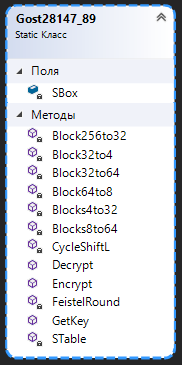


Рисунок 6 – Схема криптографического класса

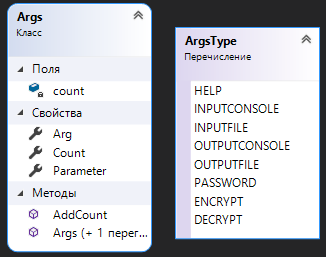


Рисунок 7 – Схема вспомогательных объектов программы

Распределение классов по семействам преследовало три цели: объединение классов по целям, разделение классов по функциональному назначению, упрощение и уменьшение кода в основном теле программы.

### Метод Main

Программа начинает выполнение из метода Main (рис. 8), который принимает строковый массив из аргументов, введенных пользователем в консоль.

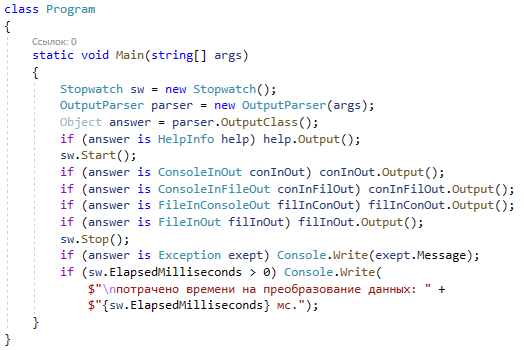


Рисунок 8 – Скриншот метода Main в классе Program

Далее, программа создает объект класса Stopwatch для замера времени шифрования или расшифрования данных. После класс OutputParser анализирует пользовательский ввод и возвращает ответ на запрос пользователя. После выявления, к какому из сценариев принадлежит ответ, программа выводит результат в консоль или в файл и оповещает пользователя о затраченном времени на шифрование.

### Реализация алгоритмов шифрования и расшифрования

ГОСТ 28147-89, в режиме простой замены, разбивает исходное сообщение на блоки по 64 бита длинной. Каждый блок делится пополам, формируя N1 и N2 блоки, длинной 32 бита каждый.

Последующие операции с полученными данными производится на основе сети Фейстеля. В одном раунде сети Фейстеля между блоком N1 и 32-битной частью ключа шифрования производится операция исключающей дизъюнкции.

Далее результат этой операции разбивается на восемь 4-битовых подпоследовательностей, каждая из которых поступает на вход своего узла таблицы замен (в порядке возрастания старшинства битов), называемого S-блоком. Общее количество S-блоков стандарта — восемь, то есть столько же, сколько и подпоследовательностей. Каждый S-блок представляет собой перестановку чисел от 0 до 15 (конкретный вид S-блоков в стандарте не определен).

Выходы всех восьми S-блоков объединяются в 32-битное слово, затем всё слово циклически сдвигается влево на 11 битов. Полученный результат N1 блока складывается по модулю 232 с N2 блоком. Последняя операция раунда сети Фейстеля – замена блоков N1 и N2 местами.

Количество раундов сети Фейстеля в данном алгоритме равно 32, но последовательность 32-битных ключей по мере выполнения алгоритма меняется. При шифровании последовательность ключей выглядит следующим образом: k1, k2, k3, k4, k5, k6, k7, k8, k1, k2, k3, k4, k5, k6, k7, k8, k1, k2, k3, k4, k5, k6, k7, k8, k8, k7, k6, k5, k4, k3, k2, k1. При дешифровке, используется та же последовательность ключей, но в обратном порядке. На рис. 9 показано как данная особенность алгоритма ГОСТ 28147-89 реализована в классе CryptoClass.

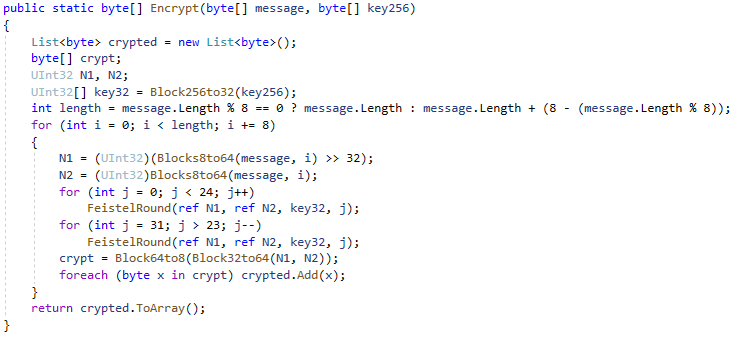


Рисунок 9 – Метод шифрования сообщения

Смена порядка использования ключей по ходу выполнения раундов сети Фейстеля обусловлено увеличением криптостойкости данного шифра. Так же на рисунке 10 видно, что расшифровка сообщения данным алгоритмом ничем, кроме порядка использования ключей шифрования не отличается.

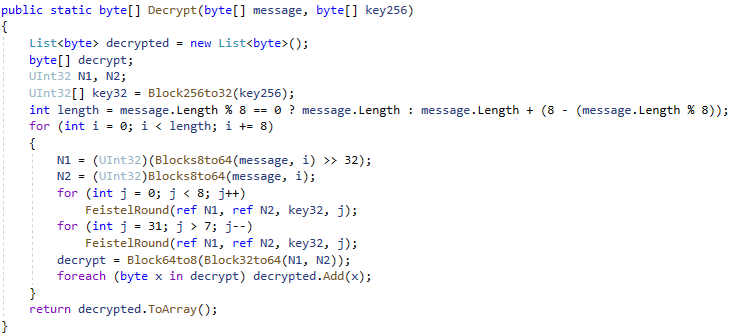


Рисунок 10 – Метод дешифровки сообщения

Так как в одном раунде сети Фейстеля происходит немного математических операций с данными, то и метод раунда Фейстеля небольшой (рис. 11).

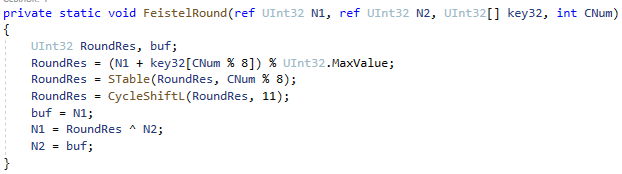


Рисунок 11 – Метод раунда сети Фейстеля

В данном методе раунда сети Фейстеля используется метод замены по S блоку (рис. 12). В методе замены по S блоку производятся операции по разбиению одного большого блока данных на массив меньших блоков и обратно. Пример реализации таких операций над блоками данных представлены на рисунках 13 и 14.

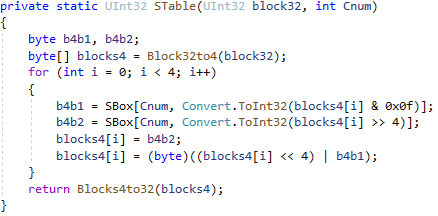


Рисунок 12 – Метод замены содержимого N блока по S блоку

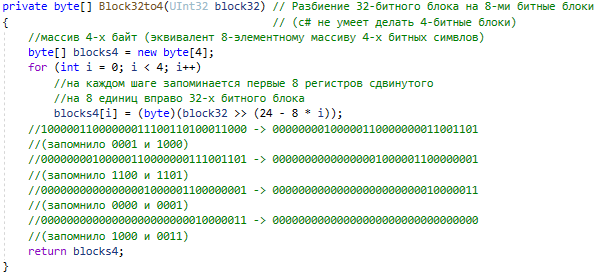


Рисунок 13 – Метод разбиения блока данных на массив меньших блоков

Как видно из примеров, при работе с данными в byte формате, очень удобно пользоваться побитовым сдвигом в связке с булевыми операциями.

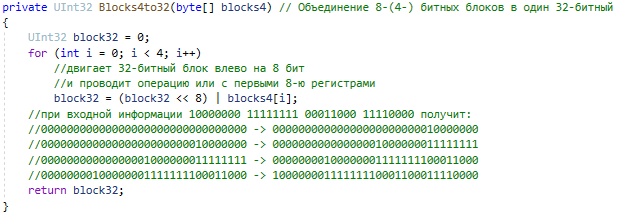


Рисунок 14 – Метод объединения массива блоков в один блок

Последний используемый метод в раунде сети Фейстеля – циклический сдвиг влево. Его реализацию можно увидеть на рисунке 151.

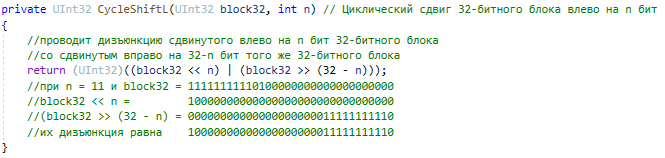


Рисунок 15 – Метод циклического сдвига влево на n бит

Последний неописанный метод класса CryptoClass – это метод генерации ключа шифрования (рис. 16).

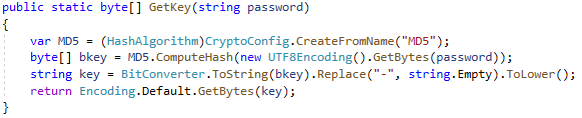


Рисунок 16 – Метод генерации ключа шифрования

Ключ для шифрования сообщений алгоритмом ГОСТ 28147-89 требует фиксированную длину в 256 бит (32 байт), что соразмерно с 32-мя печатными символами, если бы их вводил пользователь. Но такой способ ввода-вывода и хранения ключа неудобен для пользователя, поэтому можно использовать результат работы хеш функции парольной фразы произвольной длинны в строковом представлении.

### Семейство классов ввода-вывода

Родительский класс для всего семейства классов ввода-вывода IObase (рис. 17) реализует только виртуальный метод Output(), который будет переопределяться использоваться в дочерних классах. От IObase наследуются 2 класса: HelpInfo (рис. 18) и IOcrypt (рис. 19).

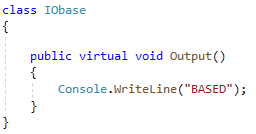


Рисунок 17 – Базовый класс IObase



Рисунок 18 – Класс HelpInfo, наследуемый от IObase



Рисунок 19 – Класс IOcrypt, наследуемый от IObase

Класс HelpInfo нужен только для вывода в консоль справочной информации о приложении со списком всех аргументов, поэтому в нем только переопределяется родительский метод Output().

Класс IOcrypt определяет 3 новых свойства:

* массив байт Key для хранения используемого для текущей операции ввода-вывода ключа шифрования,
* массив байт Data для хранения шифруемой или дешифруемой в текущей операции ввода-вывода информации,
* Enum флаг-значение Mode, в котором хранится название проделываемой с информацией операцией (шифрование или расшифрование).

В этом классе при инициализации объекта отрабатывает конструктор, генерирующий ключ шифрования по парольной фразе, вводимой пользователем.

IOcrypt является родительским классом для классов ConsoleInOut (рис. 20) и FileStreamer (рис. 21).

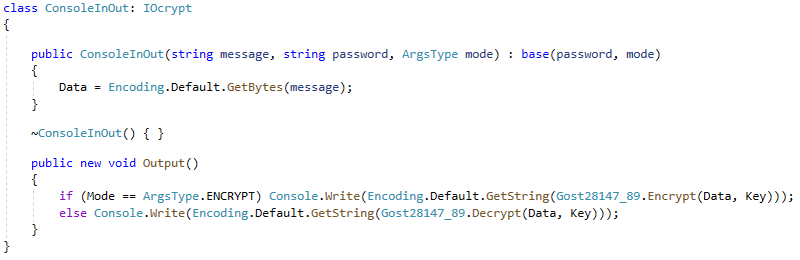


Рисунок 20 – Класс ConsoleInOut, наследуемый от IOcrypt

ConsoleInOut в конструкторе конвертирует шифруемое/дешифруемое сообщение из строкового формата в массив байт и заполняет им свойство родительского класса Data.

Так же этот класс переопределяет метод Output() и, в зависимости от того, что хранится в свойстве Mode родительского класса: ArgsType.Encrypt или ArgsType.Decrypt, соответственно шифрует или расшифровывает сообщение.

Класс FileStreamer определяет 2 новых свойства: StreamIN – поток для чтения из файла информации, StreamOUT – для записи информации в файл. В конструкторе эти свойства определяются, а в деструкторе они освобождают оперативную память и закрывают свои потоки.

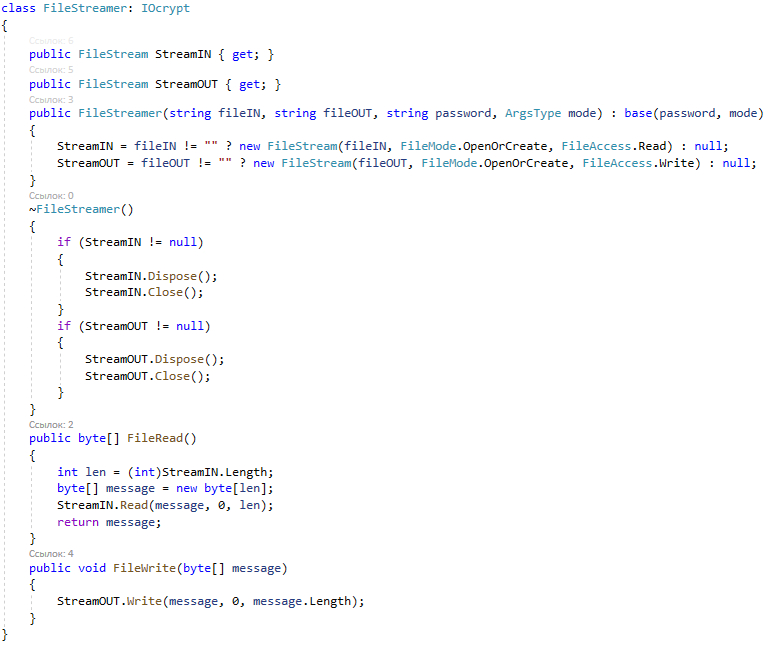


Рисунок 21 –Класс FileStreamer, наследуемый от IOcrypt

Так же соответственно новым свойствам, данный класс определяет методы чтения массива байт из файла и записи массива байт в файл.

На основе описанного класса наследуются классы, реализующие 3 сценария взаимодействия пользователя с программой:

* Шифрование/расшифровка вводимой пользователем в консоль строки с выводом результата в файл – ConsoleInFileOut (рисунок 22)
* Шифрование/расшифровка файла с выводом результата в файл – FileInOut (рисунок 23)
* Шифрование/расшифровка файла с выводом результата в консоль – FileInConsoleOut (рисунок 24)

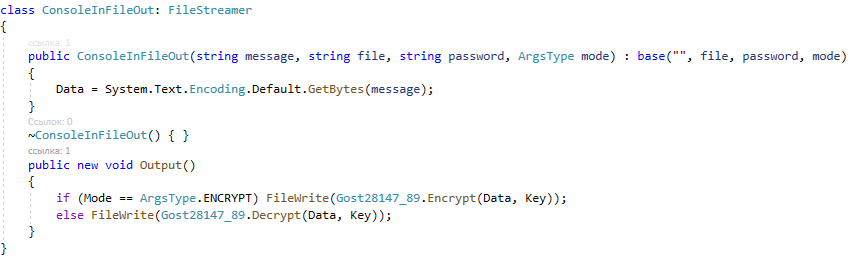


Рисунок 22 – Класс ConsoleInFileOut, наследуемый от FileStreamer



Рисунок 23 – Класс FileInOut, наследуемый от FileStreamer

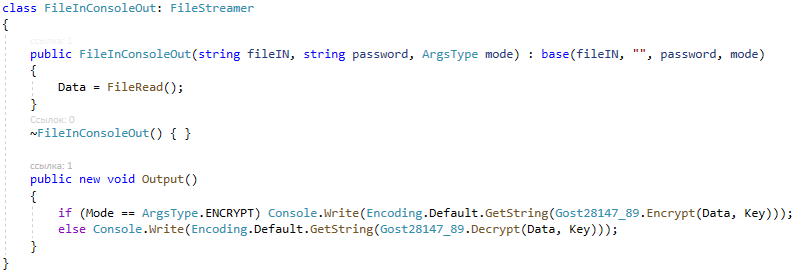


Рисунок 24 – Класс FileInConsoleOut, наследуемый от FileStreamer

Три вышеописанных класса имеют похожую структуру на класс ConsoleInOut: конструктор заполняет родительское свойство Data, а переопределенный метод Output() шифрует или расшифровывает содержимое свойства Data с выводом результата в соответствии со сценариями взаимодействия пользователя с программой.

### Семейство разбора пользовательского ввода

Так как в данной курсовой работе реализуется консольное приложение для шифрования и расшифровки информации, необходимо разработать механизм, позволяющий анализировать структуру введенных пользователем команд, выявлять в них ошибки или противоречия, и, если таких нет, реагировать на команду пользователя корректным способом.

Для решения этих задач было реализовано семейство классов для разбора пользовательского ввода. Родительский класс для этого семейства – InputParser (рисунок 25). В таблице 1 указаны соответствия между строками команд, вводимыми пользователем и Enum перечислением, с которым будет работать программа.

Таблица 1 – Обрабатываемые программой аргументы

|  |  |  |  |
| --- | --- | --- | --- |
| Аргумент | Альтернатива | Тип аргумента | Значение аргумента |
| -h | --help | ArgsType.HELP | Вывести подсказку |
| -e |  | ArgsType.ENCRYPT | Зашифровать информацию |
| -d |  | ArgsType.DECRYPT | Расшифровать информацию |
| -t | --text | ArgsType.INPUTCONSOLE | Взять информацию из консоли |
| -in |  | ArgsType.INPUTFILE | Взять информацию из файла |
| -out |  | ArgsType.OUTPUTFILE | Вернуть информацию в файл |
| -p | --password | ArgsType.PASSWORD | Ввести пароль |
|  |  | ArgsType.OUTPUTCONSOLE | Вернуть информацию в консоль |



Рисунок 25 – Класс InputParser

При инициализации этот класс объявляет новую коллекцию Dictionary, в которую добавляются все найденные в команде, введенной пользователем, аргументы и параметры аргументов. Так же, из-за отсутствия явного аргумента для вывода обработанной информации в консоль, после заполнения коллекция проверяется на наличие аргумента записи в файл. Если такого аргумента нет в списке, то в коллекцию автоматически добавляется аргумент вывода информации в консоль.

От класса InputParser наследуется класс ErrorParser (рис. 26).

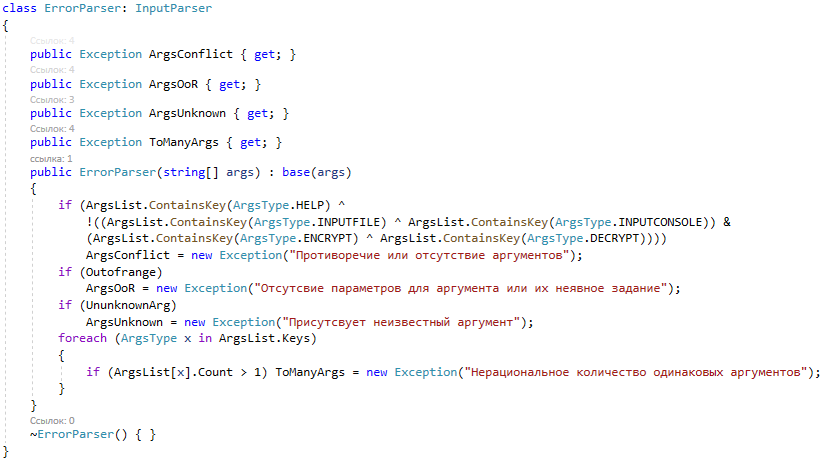


Рисунок 26 – Класс ErrorParser, наследуемый от InputParser

ErrorParser оперирует свойствами типа Exception. Всего есть 4 типа исключений, обрабатываемых в данном классе:

* ArgsConflict – ошибка, возникающая в случае использования пользователем в команде противоречащих друг-другу аргументов, например “-e -d -in file.txt”, или в случае отсутствия необходимых аргументов, например “-e -p password”.
* ArgsOoR (Out of Range) – ошибка, возникающая в случае, когда аргументы, требующие параметра не находят последний, например в строке “-e -t qwerty -p”, аргумент “-p” требует нахождения после себя параметра с парольной фразой, но так как его нет, это вызовет ошибку.
* ArgsUnknown – ошибка, возникающая в случае когда пользователь вводит либо неизвестный аргумент, либо параметр без аргумента.
* ToManyArgs – ошибка, возникающая в случае если пользователь вводит один и тот же аргумент несколько раз в пределах одной команды, например “-e -t qwerty -p password -e”

Поиск повторно использованных или недостающих аргументов производится благодаря классу Args (рис. 27).

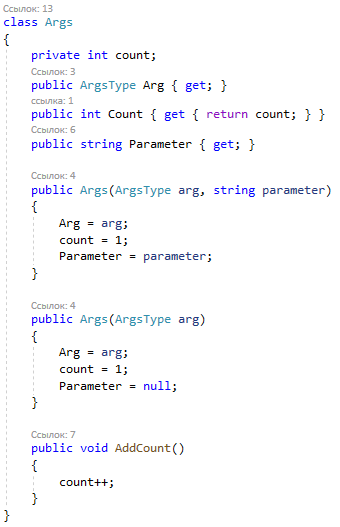


Рисунок 27 – Класс Args

Класс Args содержит в себе свойство для хранения типа аргумента, свойство для параметра аргумента и счетчик. В классе ErrorParser проверяется этот счетчик, и, если он больше 1, то возвращается ошибка ToManyArgs. Так же благодаря этому счетчику можно выявить ошибку типа ArgsConflict (отсутствие обязательных аргументов).

От класса ErrorParser наследуется класс OutputParser (рисунок 28).



Рисунок 28 – Класс OutputParser, наследуемый от InputParser

Класс OutputParser, пользуясь результатами своих родительских классов, формирует ответ на запрос пользователя. В случае, если входная строка пользователя не содержит ошибок, метод OutputClass возвращает объект из семейства классов ввода-вывода, в соответствие со сценарием пользователя. Если пользователь допустил ошибку при написании команды, метод возвращает свойство класса ErrorParser.

# Заключение

В ходе выполнения данной курсовой работы был исследован алгоритм шифрования информации ГОСТ 28147-89 в режимах простой замены, гаммирования, гаммирования с обратной связью выработки имитовставки, а также разработано с использованием языка программирования C# консольное приложение, организующее шифрование и расшифрование как текстовой информации, так и файлов, находящихся на устройствах хранения информации.

Разработанное приложение позволяет шифровать и расшифровывать информацию по парольной фразе. В случае отсутствия парольной фразы, используется стандартный ключ. В качестве алгоритма шифрования используется ГОСТ 28147-89 в режиме простой замены.

Так же разработанное приложение исключает возможные ошибки при программно-пользовательском взаимодействии, что было протестировано в данной курсовой работе.

# Список использованных источников

1. ГОСТ 28147 - 89. СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ. ЗАЩИТА КРИПТОГРАФИЧЕСКАЯ [Текст]. – Введен с 1990-07-01 по 2019-06-01. - М.: ИПК Издательство стандартов, 1996 год официальное издание

# Приложение А

(Обязательное)

**class** Program

{

**static** **void** Main(**string**[] args)

{

Stopwatch sw = **new** Stopwatch();

OutputParser parser = **new** OutputParser(args);

Object answer = parser.OutputClass();

**if** (answer **is** HelpInfo help) help.ReturnHelp();

sw.Start();

**if** (answer **is** ConsoleInOut conInOut) conInOut.Output();

**if** (answer **is** ConsoleInFileOut conInFilOut) conInFilOut.Output();

**if** (answer **is** FileInConsoleOut filInConOut) filInConOut.Output();

**if** (answer **is** FileInOut filInOut) filInOut.Output();

sw.Stop();

**if** (answer **is** Exception exept) Console.Write(exept.Message);

**if** (sw.ElapsedMilliseconds > 0) Console.Write(

$"\nпотрачено времени на преобразование данных: {0} мс.", sw.ElapsedMilliseconds);

}

}

**static** **class** Gost28147\_89

{

**private** **static** **readonly** **byte**[,] SBox = **new** **byte**[8, 16]

{

{ 0xA, 0x9, 0xD, 0x6, 0xE, 0xB, 0x4, 0x5, 0xF, 0x1, 0x3, 0xC, 0x7, 0x0, 0x8, 0x2 },

{ 0x8, 0x0, 0xC, 0x4, 0x9, 0x6, 0x7, 0xB, 0x2, 0x3, 0x1, 0xF, 0x5, 0xE, 0xA, 0xD },

{ 0xF, 0x6, 0x5, 0x8, 0xE, 0xB, 0xA, 0x4, 0xC, 0x0, 0x3, 0x7, 0x2, 0x9, 0x1, 0xD },

{ 0x3, 0x8, 0xD, 0x9, 0x6, 0xB, 0xF, 0x0, 0x2, 0x5, 0xC, 0xA, 0x4, 0xE, 0x1, 0x7 },

{ 0xF, 0x8, 0xE, 0x9, 0x7, 0x2, 0x0, 0xD, 0xC, 0x6, 0x1, 0x5, 0xB, 0x4, 0x3, 0xA },

{ 0x2, 0x8, 0x9, 0x7, 0x5, 0xF, 0x0, 0xB, 0xC, 0x1, 0xD, 0xE, 0xA, 0x3, 0x6, 0x4 },

{ 0x3, 0x8, 0xB, 0x5, 0x6, 0x4, 0xE, 0xA, 0x2, 0xC, 0x1, 0x7, 0x9, 0xF, 0xD, 0x0 },

{ 0x1, 0x2, 0x3, 0xE, 0x6, 0xD, 0xB, 0x8, 0xF, 0xA, 0xC, 0x5, 0x7, 0x9, 0x0, 0x4 }

};

**public** **static** **byte**[] Encrypt(**byte**[] message, **byte**[] key256)

{

List<**byte**> crypted = **new** List<**byte**>();

**byte**[] crypt;

UInt32 N1, N2;

UInt32[] key32 = Block256to32(key256);

**int** length = message.Length % 8 == 0 ? message.Length : message.Length + (8 - (message.Length % 8));

**for** (**int** i = 0; i < length; i += 8)

{

N1 = (UInt32)(Blocks8to64(message, i) >> 32);

N2 = (UInt32)Blocks8to64(message, i);

**for** (**int** j = 0; j < 24; j++)

FeistelRound(**ref** N1, **ref** N2, key32, j);

**for** (**int** j = 31; j > 23; j--)

FeistelRound(**ref** N1, **ref** N2, key32, j);

crypt = Block64to8(Block32to64(N1, N2));

**foreach** (**byte** x **in** crypt) crypted.Add(x);

}

**return** crypted.ToArray();

}

**public** **static** **byte**[] Decrypt(**byte**[] message, **byte**[] key256)

{

List<**byte**> decrypted = **new** List<**byte**>();

**byte**[] decrypt;

UInt32 N1, N2;

UInt32[] key32 = Block256to32(key256);

**int** length = message.Length % 8 == 0 ? message.Length : message.Length + (8 - (message.Length % 8));

**for** (**int** i = 0; i < length; i += 8)

{

N1 = (UInt32)(Blocks8to64(message, i) >> 32);

N2 = (UInt32)Blocks8to64(message, i);

**for** (**int** j = 0; j < 8; j++)

FeistelRound(**ref** N1, **ref** N2, key32, j);

**for** (**int** j = 31; j > 7; j--)

FeistelRound(**ref** N1, **ref** N2, key32, j);

decrypt = Block64to8(Block32to64(N1, N2));

**foreach** (**byte** x **in** decrypt) decrypted.Add(x);

}

**return** decrypted.ToArray();

}

**public** **static** **byte**[] GetKey(**string** password)

{

**var** MD5 = (HashAlgorithm)CryptoConfig.CreateFromName("MD5");

**byte**[] bkey = MD5.ComputeHash(**new** UTF8Encoding().GetBytes(password));

**string** key = BitConverter.ToString(bkey).Replace("-", **string**.Empty).ToLower();

**return** Encoding.Default.GetBytes(key);

}

**private** **static** **void** FeistelRound(**ref** UInt32 N1, **ref** UInt32 N2, UInt32[] key32, **int** CNum)

{

UInt32 RoundRes, buf;

RoundRes = (N1 + key32[CNum % 8]) % UInt32.MaxValue;

RoundRes = STable(RoundRes, CNum % 8);

RoundRes = CycleShiftL(RoundRes, 11);

buf = N1;

N1 = RoundRes ^ N2;

N2 = buf;

}

**private** **static** UInt32 STable(UInt32 block32, **int** Cnum)

{

**byte** b4b1, b4b2;

**byte**[] blocks4 = Block32to4(block32);

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

{

b4b1 = SBox[Cnum, Convert.ToInt32(blocks4[i] & 0x0f)];

b4b2 = SBox[Cnum, Convert.ToInt32(blocks4[i] >> 4)];

blocks4[i] = b4b2;

blocks4[i] = (**byte**)((blocks4[i] << 4) | b4b1);

}

**return** Blocks4to32(blocks4);

}

**private** **static** UInt32[] Block256to32(**byte**[] key256)

{

UInt32[] blocs32 = **new** UInt32[8];

**for** (**int** i = 0; i < 8; i++)

{

**for** (**int** j = 0; j < 4; j++)

blocs32[i] = blocs32[i] << 8 | key256[4 \* i + j];

}

**return** blocs32;

}

**private** **static** UInt64 Blocks8to64(**byte**[] blocks8, **int** x)

{

UInt64 block64 = 0;

**for** (**int** i = x; i < x + 8; i++)

{

**if** (i < blocks8.Length)

block64 = block64 << 8 | blocks8[i];

**else**

block64 = block64 << 8 | 0;

}

**return** block64;

}

**private** **static** **byte**[] Block32to4(UInt32 block32)

{

**byte**[] blocks4 = **new** **byte**[4];

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

blocks4[i] = (**byte**)(block32 >> (24 - 8 \* i));

**return** blocks4;

}

**private** **static** UInt32 Blocks4to32(**byte**[] blocks4)

{

UInt32 block32 = 0;

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

block32 = (block32 << 8) | blocks4[i];

**return** block32;

}

**private** **static** UInt64 Block32to64(UInt32 N1, UInt32 N2)

{

UInt64 block64 = N2;

block64 = block64 << 32 | N1;

**return** block64;

}

**private** **static** **byte**[] Block64to8(UInt64 block64)

{

**byte**[] blocks8 = **new** **byte**[8];

**for** (**int** i = 0; i < 8; i++)

blocks8[i] = (**byte**)(block64 >> ((7 - i) \* 8));

**return** blocks8;

}

**private** **static** UInt32 CycleShiftL(UInt32 block32, **int** n)

{

**return** (UInt32)((block32 << n) | (block32 >> (32 - n)));

}

}

**public** **enum** ArgsType

{

HELP,

INPUTCONSOLE,

INPUTFILE,

OUTPUTCONSOLE,

OUTPUTFILE,

PASSWORD,

ENCRYPT,

DECRYPT

}

**class** Args

{

**private** **int** count;

**public** ArgsType Arg { **get**; }

**public** **int** Count { **get** { **return** count; } }

**public** **string** Parameter { **get**; }

**public** Args(ArgsType arg, **string** parameter)

{

Arg = arg;

count = 1;

Parameter = parameter;

}

**public** Args(ArgsType arg)

{

Arg = arg;

count = 1;

Parameter = **null**;

}

**public** **void** AddCount()

{

count++;

}

}

**class** IObase

{

**public** **virtual** **void** Output()

{

Console.WriteLine("BASED");

}

}

**class** IOcrypt: IObase

{

**public** **byte**[] Data { **get**; **set**; }

**public** **byte**[] Key { **get**; }

**public** ArgsType Mode { **get**; }

**public** IOcrypt(**string** password, ArgsType mode)

{

Key = Gost28147\_89.GetKey(password);

Mode = mode;

}

~IOcrypt() { }

}

**class** ConsoleInOut: IOcrypt

{

**public** ConsoleInOut(**string** message, **string** password, ArgsType mode) : **base**(password, mode)

{

Data = Encoding.Default.GetBytes(message);

}

**public** **override** **void** Output()

{

**if** (Mode == ArgsType.ENCRYPT)

Console.Write(Encoding.Default.GetString(Gost28147\_89.Encrypt(Data, Key)));

**else** Console.Write(Encoding.Default.GetString(Gost28147\_89.Decrypt(Data, Key)));

}

}

**class** FileStreamer: IOcrypt

{

**public** FileStream StreamIN { **get**; }

**public** FileStream StreamOUT { **get**; }

**public** FileStreamer(**string** fileIN, **string** fileOUT, **string** password, ArgsType mode):**base**(password, mode)

{

StreamIN = fileIN != "" ? **new** FileStream(fileIN, FileMode.OpenOrCreate, FileAccess.Read) : **null**;

StreamOUT = fileOUT != "" ? **new** FileStream(fileOUT, FileMode.OpenOrCreate, FileAccess.Write) : **null**;

}

~FileStreamer()

{

**if** (StreamIN != **null**)

{

StreamIN.Dispose();

StreamIN.Close();

}

**if** (StreamOUT != **null**)

{

StreamOUT.Dispose();

StreamOUT.Close();

}

}

**public** **byte**[] FileRead()

{

**int** len = (**int**)StreamIN.Length;

**byte**[] message = **new** **byte**[len];

StreamIN.Read(message, 0, len);

**return** message;

}

**public** **void** FileWrite(**byte**[] message)

{

StreamOUT.Write(message, 0, message.Length);

}

}

**class** ConsoleInFileOut: FileStreamer

{

**public** ConsoleInFileOut(**string** message, **string** file, **string** password, ArgsType mode) : **base**("", file, password, mode)

{

Data = System.Text.Encoding.Default.GetBytes(message);

}

~ConsoleInFileOut() { }

**public** **override** **void** Output()

{

**if** (Mode == ArgsType.ENCRYPT) FileWrite(Gost28147\_89.Encrypt(Data, Key));

**else** FileWrite(Gost28147\_89.Decrypt(Data, Key));

}

}

**class** FileInOut: FileStreamer

{

**public** FileInOut(**string** fileIN, **string** fileOUT, **string** password, ArgsType mode) : **base**(fileIN, fileOUT, password, mode)

{

Data = FileRead();

}

~FileInOut() { }

**public** **override** **void** Output()

{

**if** (Mode == ArgsType.ENCRYPT) FileWrite(Gost28147\_89.Encrypt(Data, Key));

**else** FileWrite(Gost28147\_89.Decrypt(Data, Key));

}

}

**class** FileInConsoleOut: FileStreamer

{

**public** FileInConsoleOut(**string** fileIN, **string** password, ArgsType mode):**base**(fileIN, "", password, mode)

{

Data = FileRead();

}

~FileInConsoleOut() { }

**public** **override** **void** Output()

{

**if** (Mode == ArgsType.ENCRYPT) Console.Write(

Encoding.Default.GetString(Gost28147\_89.Encrypt(Data, Key)));

**else** Console.Write(Encoding.Default.GetString(Gost28147\_89.Decrypt(Data, Key)));

}

}

**class** InputParser

{

**private** **bool** outofrange;

**public** **bool** Outofrange { **get** { **return** outofrange; } }

**private** **bool** ununknownArg;

**public** **bool** UnunknownArg { **get** { **return** ununknownArg; } }

**readonly** **private** Dictionary<ArgsType, Args> argsList;

**public** Dictionary<ArgsType, Args> ArgsList { **get** { **return** argsList; } }

**public** InputParser(**string** [] args)

{

argsList = **new** Dictionary<ArgsType, Args>();

Parse(args);

**if** (!argsList.ContainsKey(ArgsType.OUTPUTFILE))

argsList.Add(ArgsType.OUTPUTCONSOLE, **new** Args(ArgsType.OUTPUTCONSOLE));

}

~InputParser(){ }

**private** **void** Parse(**string**[] args)

{

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

{

**try**

{

**string** k = "";

**if** (args[i].ToLower() == "-h" | args[i].ToLower() == "--help")

**if** (!argsList.ContainsKey(ArgsType.HELP))

argsList.Add(ArgsType.HELP, **new** Args(ArgsType.HELP));

**else** argsList[ArgsType.HELP].AddCount();

**else** **if** (args[i].ToLower() == "-e")

**if** (!argsList.ContainsKey(ArgsType.ENCRYPT))

argsList.Add(ArgsType.ENCRYPT, **new** Args(ArgsType.ENCRYPT));

**else** argsList[ArgsType.ENCRYPT].AddCount();

**else** **if** (args[i].ToLower() == "-d")

**if** (!argsList.ContainsKey(ArgsType.DECRYPT))

argsList.Add(ArgsType.DECRYPT, **new** Args(ArgsType.DECRYPT));

**else** argsList[ArgsType.DECRYPT].AddCount();

**else** **if** (args[i].ToLower() == "-t" | args[i].ToLower() == "--text")

**if** (!argsList.ContainsKey(ArgsType.INPUTCONSOLE))

{

k = args[i + 1][0] == '\\' ? args[i + 1].Remove(0, 1) : args[i + 1];

argsList.Add(ArgsType.INPUTCONSOLE, **new** Args(ArgsType.INPUTCONSOLE, k));

}

**else** argsList[ArgsType.INPUTCONSOLE].AddCount();

**else** **if** (args[i].ToLower() == "-in")

**if** (!argsList.ContainsKey(ArgsType.INPUTFILE))

argsList.Add(ArgsType.INPUTFILE, **new** Args(ArgsType.INPUTFILE, args[i + 1]));

**else** argsList[ArgsType.INPUTFILE].AddCount();

**else** **if** (args[i].ToLower() == "-out")

**if** (!argsList.ContainsKey(ArgsType.OUTPUTFILE))

argsList.Add(ArgsType.OUTPUTFILE, **new** Args(ArgsType.OUTPUTFILE, args[i + 1]));

**else** argsList[ArgsType.OUTPUTFILE].AddCount();

**else** **if** (args[i].ToLower() == "-p" | args[i].ToLower() == "--password")

**if** (!argsList.ContainsKey(ArgsType.PASSWORD))

{

k = args[i + 1][0] == '\\' ? args[i + 1].Remove(0, 1) : args[i + 1];

argsList.Add(ArgsType.PASSWORD, **new** Args(ArgsType.PASSWORD, k));

}

**else** argsList[ArgsType.PASSWORD].AddCount();

**else** **if** (args[i][0] == '-') ununknownArg = **true**;

}

**catch** { outofrange = **true**; }

}

}

}

**class** ErrorParser: InputParser

{

**public** Exception ArgsConflict { **get**; }

**public** Exception ArgsOoR { **get**; }

**public** Exception ArgsUnknown { **get**; }

**public** Exception ToManyArgs { **get**; }

**public** ErrorParser(**string**[] args) : **base**(args)

{

**if** (ArgsList.ContainsKey(ArgsType.HELP) ^

!((ArgsList.ContainsKey(ArgsType.INPUTFILE) ^ ArgsList.ContainsKey(ArgsType.INPUTCONSOLE)) &

(ArgsList.ContainsKey(ArgsType.ENCRYPT) ^ ArgsList.ContainsKey(ArgsType.DECRYPT))))

ArgsConflict = **new** Exception("Противоречие или отсутствие аргументов");

**if** (Outofrange)

ArgsOoR = **new** Exception("Отсутсвие параметров для аргумента или их неявное задание");

**if** (UnunknownArg)

ArgsUnknown = **new** Exception("Присутсвует неизвестный аргумент");

**foreach** (ArgsType x **in** ArgsList.Keys)

{

**if** (ArgsList[x].Count > 1) ToManyArgs = **new** Exception(

"Нерациональное количество одинаковых аргументов");

}

}

~ErrorParser() { }

}

**class** OutputParser : ErrorParser

{

**public** **string** Password { **get**; }

**public** **string** Message { **get**; }

**public** **string** Filein { **get**; }

**public** **string** Fileout { **get**; }

**public** ArgsType Mode { **get**; }

**public** OutputParser(**string**[] args) : **base**(args)

{

Password = ArgsList.ContainsKey(ArgsType.PASSWORD) ? ArgsList[ArgsType.PASSWORD].Parameter : "";

Message = ArgsList.ContainsKey(ArgsType.INPUTCONSOLE)? ArgsList[ArgsType.INPUTCONSOLE].Parameter:"";

Filein = ArgsList.ContainsKey(ArgsType.INPUTFILE) ? ArgsList[ArgsType.INPUTFILE].Parameter : "";

Fileout = ArgsList.ContainsKey(ArgsType.OUTPUTFILE) ? ArgsList[ArgsType.OUTPUTFILE].Parameter : "";

Mode = ArgsList.ContainsKey(ArgsType.DECRYPT) ?

ArgsList[ArgsType.DECRYPT].Arg == ArgsType.DECRYPT ?

ArgsType.DECRYPT : ArgsType.ENCRYPT : ArgsType.ENCRYPT;

}

~OutputParser() { }

**public** Object OutputClass()

{

**if** (ArgsConflict == **null** & ArgsOoR == **null** & ToManyArgs == **null** & ArgsUnknown == **null**)

{

**if** (ArgsList.ContainsKey(ArgsType.HELP)) **return** **new** HelpInfo();

**else** **if** (ArgsList.ContainsKey(ArgsType.INPUTCONSOLE) &

ArgsList.ContainsKey(ArgsType.OUTPUTCONSOLE))

**return** **new** ConsoleInOut(Message, Password, Mode);

**else** **if** (ArgsList.ContainsKey(ArgsType.INPUTCONSOLE) & ArgsList.ContainsKey(ArgsType.OUTPUTFILE))

**return** **new** ConsoleInFileOut(Message, Fileout, Password, Mode);

**else** **if** (ArgsList.ContainsKey(ArgsType.INPUTFILE) & ArgsList.ContainsKey(ArgsType.OUTPUTCONSOLE))

**return** **new** FileInConsoleOut(Filein, Password, Mode);

**else** **if** (ArgsList.ContainsKey(ArgsType.INPUTFILE) & ArgsList.ContainsKey(ArgsType.OUTPUTFILE))

**return** **new** FileInOut(Filein, Fileout, Password, Mode);

}

**else** **if** (ArgsConflict != **null**) **return** ArgsConflict;

**else** **if** (ArgsOoR != **null**) **return** ArgsOoR;

**else** **if** (ToManyArgs != **null**) **return** ToManyArgs;

**return** ArgsUnknown;

}

}