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

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#.

1 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.

1.1 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 N_1 и N_2 блоки, длинной 32 бита каждый.

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

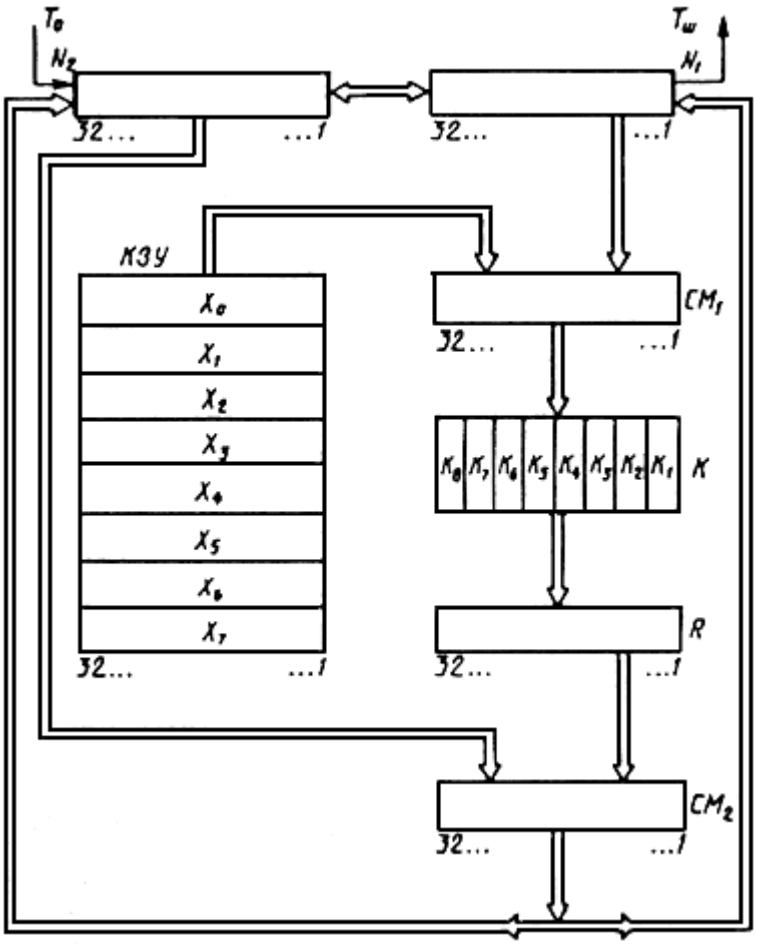

Рисунок 1 – Схема алгоритма ГОСТ 28147-89 в режиме простой замены

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

В первом цикле начальное заполнение накопителя N_1 суммируется по модулю 2^{32} в сумматоре $СМ_1$ с заполнением накопителя X_0, при этом заполнение накопителя N_1 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 $СМ_2$ с 32-разрядным заполнением накопителя N_2. Полученный в $СМ_2$ результат записывается в N_1, при этом старое заполнение N_1 переписывается в N_2. Первый цикл заканчивается.

Последующие циклы осуществляются аналогично, при этом во 2-м цикле из КЗУ считывается заполнение X_1 в 3-м цикле из КЗУ считывается заполнение X_2 и т.д.,

в 8-м цикле из КЗУ считывается заполнение X_7. 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: X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7. В последних восьми циклах с 25-го по 32-й порядок считывания заполнений КЗУ обратный: X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0.

В 32 цикле результат из сумматора $СМ_2$ вводится в накопитель N_2, а в накопителе N_1 сохраняется старое заполнение. Полученные после 32-го цикла зашифрования заполнения накопителей N_1 и N_2 являются блоком зашифрованных данных, соответствующим блоку открытых данных.

Уравнения зашифрования в режиме простой замены имеют вид:

$$\begin{cases} a(j) = (a(j-1) \boxplus X(j-1)\ (mod 8))KR \oplus b(j-1) \\ b(j) = a(j-1) \end{cases}, \text{при } j \in [1, 2, 3 \dots 24] \quad (1)$$

$$\begin{cases} a(j) = \big(a(j-1) \boxplus X(32-j)\big)KR \oplus b(j-1) \\ b(j) = a(j-1) \end{cases}, \text{при } j \in [25, 26, 27 \dots 31] \quad (2)$$

$$\begin{cases} a(32) = a(31) \\ b(32) = \big(a(32) \boxplus X(0)\big)KR \oplus b(31) \end{cases}, \text{при } j = 32 \quad (3)$$

Где $a(0) = (a_{32}(0), a_{31}(0), \dots, a_1(0))$ - начальное заполнение N_1 перед первым циклом зашифрования;

$b(0) = (b_{32}(0), b_{31}(0), \dots, b_1(0))$ - начальное заполнение N_2 перед первым циклом зашифрования;

$a(j) = (a_{32}(j), a_{31}(j), \dots, a_1(j))$ - заполнение N_1 после j-го цикла зашифрования;

$b(j) = (b_{32}(j), b_{31}(j), \dots, b_1(j))$ - заполнение N_2 после j-го цикла зашифрования;

$\oplus$ - поразрядное суммирование 32-разрядных векторов по модулю 2;

$\boxplus$ - суммирование 32-разрядных векторов по модулю 2^{32};

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 X_0, X_1, … , X_7 считываются из КЗУ в циклах расшифрования в следующем порядке: X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0.

Уравнения расшифрования имеют вид:

$$\begin{cases} a(32-j) = \big(a(32-j+1) \boxplus X(j-1)\big) KR \oplus b(32-j+1) \\ b(32-j) = a(32-j+1) \end{cases}, \text{при } j \in [1, 2, 3 \dots 8] \quad (4)$$

$$\begin{cases} a(32-j) = (a(32-j+1) \boxplus X(32-j)(mod 8)) KR \oplus b(32-j+1) \\ b(32-j) = a(32-j+1) \end{cases}, \text{при } j \in [9, 10, 11 \dots 31] \quad (5)$$

$$\begin{cases} a(0) = a(1) \\ b(32) = \big(a(1) \boxplus X(0)\big) KR \oplus b(1) \end{cases}, \text{при } j = 32 \quad (6)$$

Где $a(0) = (a_{32}(0), a_{31}(0), \dots, a_1(0))$ - начальное заполнение N_1 перед первым циклом зашифрования;

$b(0) = (b_{32}(0), b_{31}(0), \dots, b_1(0))$ - начальное заполнение N_2 перед первым циклом зашифрования;

$a(j) = (a_{32}(j), a_{31}(j), \dots, a_1(j))$ - заполнение N_1 после j-го цикла зашифрования;

$b(j) = (b_{32}(j), b_{31}(j), \ldots , b_1(j))$ - заполнение N_2 после j-го цикла зашифрования;

$\oplus$ - поразрядное суммирование 32-разрядных векторов по модулю 2;

$\boxplus$ - суммирование 32-разрядных векторов по модулю 2^{32};

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

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

1.2 Режим гаммирования

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

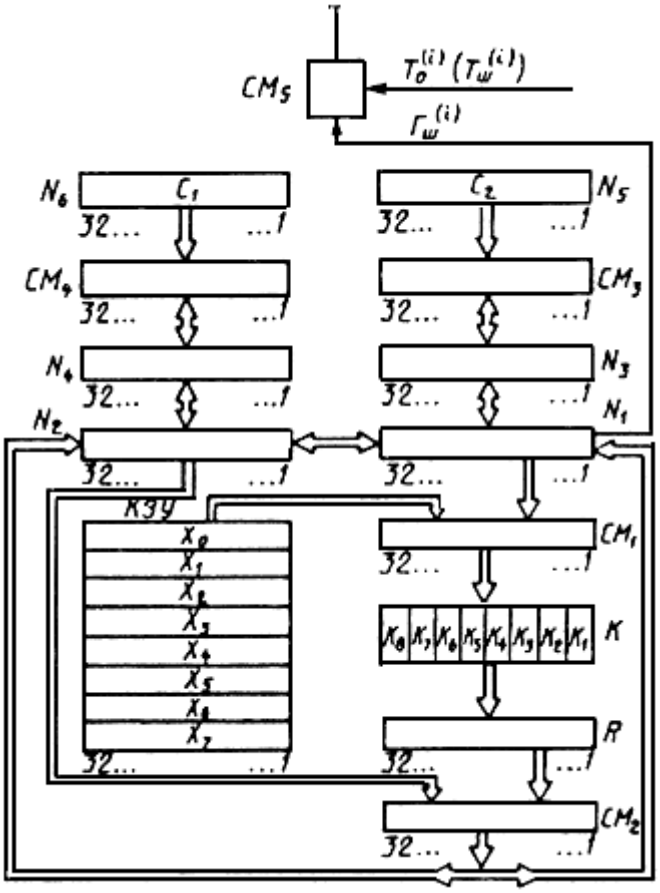

Рисунок 2 – Схема алгоритма ГОСТ 28147-89 в режиме гаммирования

Открытые данные, разбитые на 64-разрядные блоки $T_0^{(1)}$, $T_0^{(2)}$ ... , $T_0^{(M-1)}$, $T_0^{(M)}$, зашифровываются в режиме гаммирования путем поразрядного суммирования по модулю 2 в сумматоре $СМ_5$ с гаммой шифра $Г_Ш$, которая вырабатывается блоками по 64 бита $Г_Ш = (Г_Ш^{(1)}, Г_Ш^{(2)}, \ldots , Г_Ш^{(M-1)}, Г_Ш^{(M)})$, где M - определяется объемом шифруемых данных.

В КЗУ вводятся 256 бит ключа. В накопители N_1, N_2 вводится 64-разрядная двоичная последовательность (синхропосылка) $S = (S_1, S_2, \ldots , S_{64})$, являющаяся исходным заполнением этих накопителей для последующей выработки М блоков гаммы шифра. Синхропосылка вводится в N_1 и N_2 так, что значение S_1 вводится в 1-й разряд N_1, значение S_2 вводится во 2-й разряд N_1 и т.д., значение S_{32} вводится в 32-й разряд N_1; значение S_{33} вводится в 1-й разряд N_2, значение S_{34} вводится во 2-й разряд N_2 и т.д., значение S_{64} вводится в 32-й разряд N_2.

Исходное заполнение накопителей N_1 и N_2 (синхропосылка S) зашифровывается в режиме простой замены. Результат зашифрования $A(S) = (Y_0, Z_0)$ переписывается в 32-разрядные накопители N_3 и N_4.

Заполнение накопителя N_4 суммируется по модулю $(2^{32} - 1)$ в сумматоре $СМ_4$ с 32-разрядной константой C_1 из накопителя N_6, результат записывается в N_4. Заполнение накопителя N_3 суммируется по модулю 2^{32} в сумматоре $СМ_3$ в 32-разрядной константой C_2 из накопителя N_5, результат записывается в N_3.

Заполнение N_3 переписывается в N_1, а заполнение N_4 переписывается в N_2, при этом заполнение N_3, N_4 сохраняется.

Заполнение N_1 и N_2 зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение N_1, N_2, образует первый 64-разрядный блок гаммы шифра $Г_Ш^{(1)}$, который суммируется поразрядно по модулю 2 в сумматоре $СМ_5$ 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 N_1 и N_2 и осуществляется процесс

выработки М блоков гаммы шифра. Блоки зашифрованных данных суммируются поразрядно по модулю 2 в сумматоре $СМ_5$ с блоками гаммы шифра, в результате получаются блоки открытых данных при этом последний блок расшифрованных данных может содержать меньше 64 разрядов.

1.3 Режим гаммирования с обратной связью

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

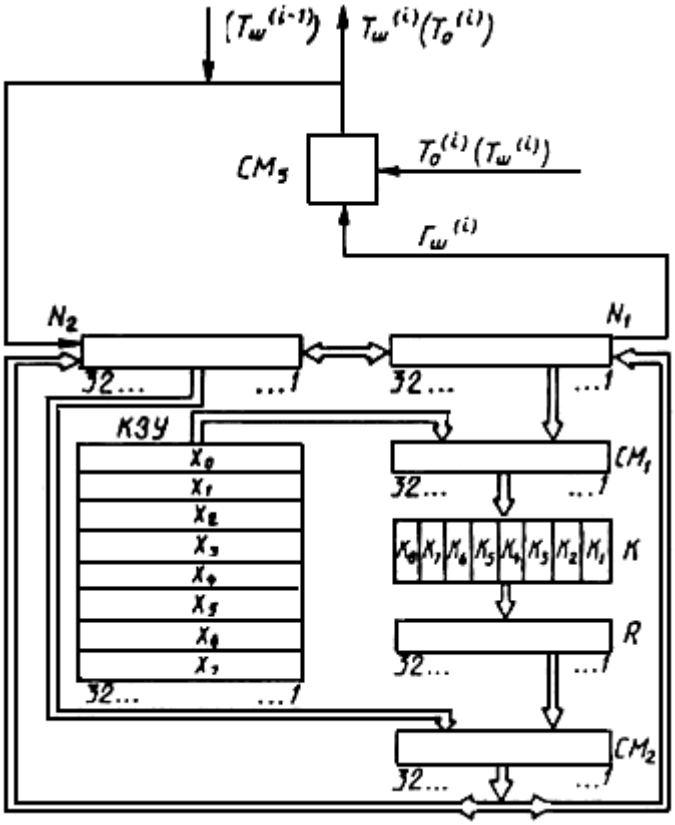

Рисунок 3 – Схема алгоритма ГОСТ 28147-89 в режиме гаммирования с обратной связью

Открытые данные, разбитые на 64-разрядные блоки $T_0^{(1)}$, ..., $T_0^{(M)}$, зашифровываются в режиме гаммирования с обратной связью путем поразрядного суммирования по модулю 2 в сумматоре $СМ_5$ с гаммой шифра $Г_Ш$, которая вырабатывается блоками по 64 бита, т.е. $Г_Ш = (Г_Ш^{(1)}, Г_Ш^{(2)}, \ldots, Г_Ш^{(M)})$, где определяется объемом открытых данных, $Г_Ш^{(i)}$ - i-й 64-разрядный блок, $i \in [1, \ldots, M]$. Число двоичных разрядов в блоке $T_0^{(M)}$ может быть меньше 64.

Так же, как и в ГОСТ 28147-89, В КЗУ вводятся 256 бит ключа. Синхропосылка $S = (S_1, S_2, \ldots, S_{64})$ из 64 бит вводится в N_1 и N_2.

Исходное заполнение N_1 и N_2 зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение N_1 и N_2 образует первый 64-разрядный блок гаммы шифра $Г_{Ш}^{(1)} = A(S)$, который суммируется поразрядно по модулю 2 в сумматоре с первым 64-разрядным блоком открытых данных $T_0^{(1)} = (t_1^{(1)}, t_2^{(1)}, \ldots, t_{64}^{(1)})$. В результате получается 64-разрядный блок зашифрованных данных $Т_Ш^{(1)}$. Блок зашифрованных данных $T_{Ш}^{(1)}$ одновременно является также исходным состоянием N_1 и N_2 для выработки второго блока гаммы шифра $Г_{Ш}^{(2)}$ и по обратной связи записывается в указанные накопители.

При расшифровании криптосхема имеет тот же вид, что и при зашифровании. Исходное заполнение N_1 и N_2 (синхропосылка S) зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение N_1 и N_2 образует первый блок гаммы шифра $Г_{Ш}^{(1)} = A(S)$, который суммируется поразрядно по модулю 2 в сумматоре $СМ_5$ с блоком зашифрованных данных $T_{Ш}^{(1)}$. В результате получается первый блок открытых данных $T_0^{(1)}$.

Блок зашифрованных данных $T_{Ш}^{(1)}$ является исходным заполнением N_1 и N_2 для выработки второго блока гаммы шифра $Г_{Ш}^{(2)}$. Блок $T_{Ш}^{(1)}$ записывается в N_1 и N_2 так же, как и при шифровании в данном режиме. Полученное заполнение N_1 и N_2 зашифровывается в режиме простой замены, полученный в результате блок $Г_{Ш}^{(2)}$ суммируется поразрядно по модулю 2 в сумматоре $СМ_5$ со вторым блоком зашифрованных данных $T_{Ш}^{(2)}$. В результате получается блок открытых данных $T_0^{(2)}$.

1.4 Режим выработки имитовставки

Для обеспечения имитозашиты открытых данных, состоящих из M 64-разрядных блоков $T_0^{(1)}, \ldots, T_0^{(M)}$, $M \geq 2$, вырабатывается дополнительный блок из l бит (имитовставка $И_l$). Процесс выработки имитовставки единообразен для всех режимов шифрования.

Первый блок открытых данных $T_0^{(1)} = (t_1^{(1)}, t_2^{(1)}, \dots, t_{64}^{(1)}) = (a_1^{(1)}(0), a_2^{(1)}(0), \dots, a_{32}^{(1)}(0), b_1^{(1)}(0), b_2^{(1)}(0), \dots, b_{32}^{(1)}(0))$ записывается в накопители N_1 и N_2, при этом значение $t_1^{(1)} = a_1^{(1)}(0)$ вводится в 1-й разряд N_1, значение $t_2^{(1)} = a_2^{(1)}(0)$ вводится во 2й разряд N_1 и т.д.

Заполнение N_1 и N_2 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 $T_0^{(1)}$, $T_0^{(2)}$, ... $T_0^{(M)}$ в соответствующие блоки зашифрованных данных $T_Ш^{(1)}$, $T_Ш^{(2)}$, ... $T_Ш^{(M)}$.

Полученное после 16 циклов работы заполнение N_1 и N_2, имеющее вид $(a_1^{(1)}(16), a_2^{(1)}(16), \dots, a_{32}^{(1)}(16), b_1^{(1)}(16), b_2^{(1)}(16), \dots, b_{32}^{(1)}(16))$, суммируется в $СМ_5$ по модулю 2 со вторым блоком $T_0^{(2)} = (t_1^{(2)}, t_2^{(2)}, \dots, t_{64}^{(2)})$. Результат суммирования заносится в N_1 и N_2 и подвергается преобразованию, соответствующему первым 16 циклам алгоритма зашифрования в режиме простой замены.

Полученное заполнение N_1 и N_2 суммируется в $СМ_5$ по модулю 2 с третьим блоком $T_0^{(3)}$ и т.д. Имитовставка $И_l$ передается по каналу связи или в память ЭВМ в конце зашифрованных данных, т.е. $T_Ш^{(1)}$, $T_Ш^{(2)}$, ... $T_Ш^{(M)}$, $И_l$.

Поступившие зашифрованные данные $T_Ш^{(1)}$, $T_Ш^{(2)}$, ... $T_Ш^{(M)}$ расшифровываются, из полученных блоков открытых данных $T_0^{(1)}$, $T_0^{(2)}$, ... , $T_0^{(M)}$ 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 $T_0^{(1)}$, $T_0^{(2)}$, ... , $T_0^{(M)}$ считают ложными.

Выработка имитовставки $И_l$ $(И_l')$ 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.

2 Реализация приложения для шифрования информации

2.1 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)

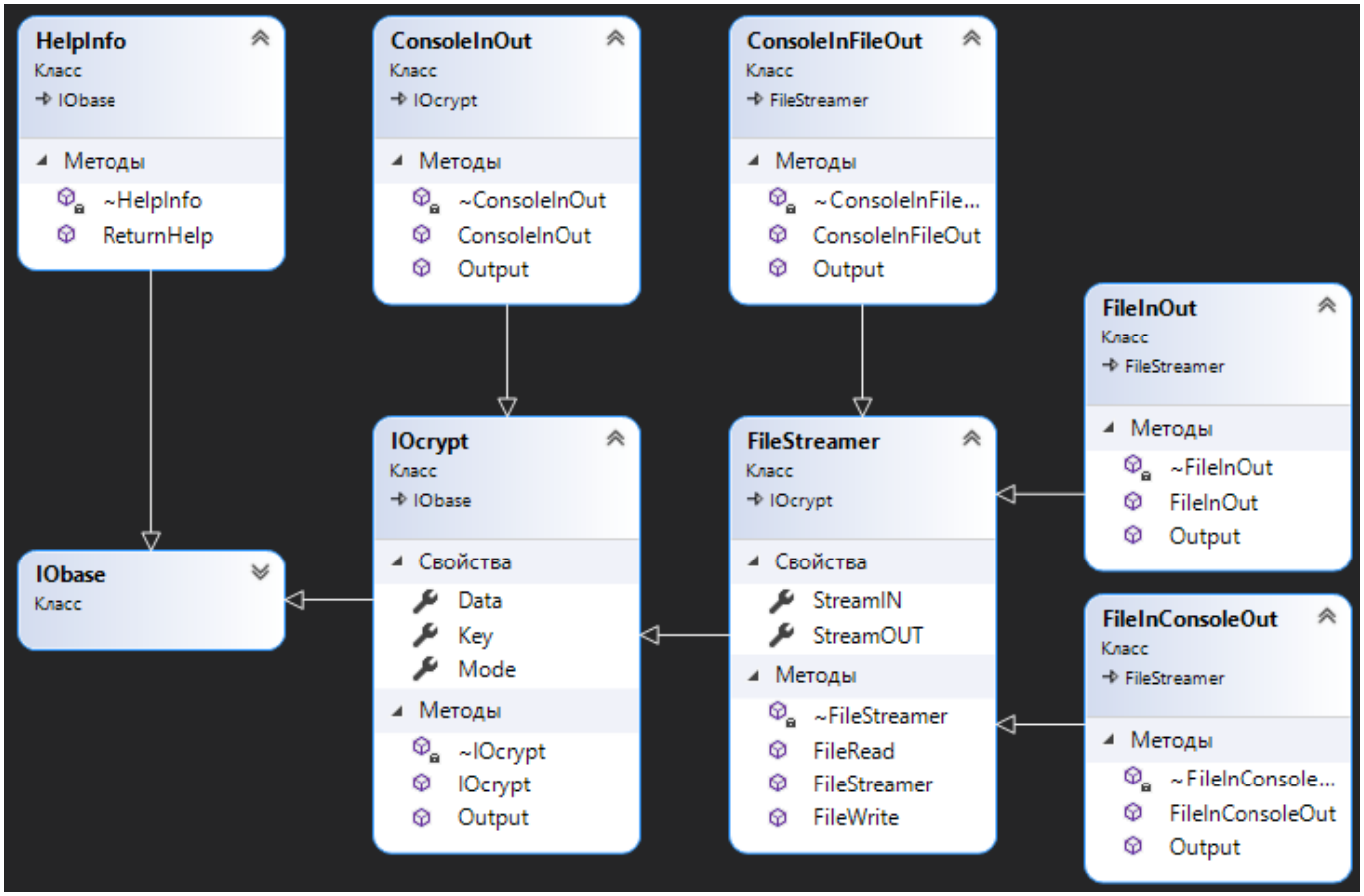

Рисунок 4 – Схема семейства ввода-вывода

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

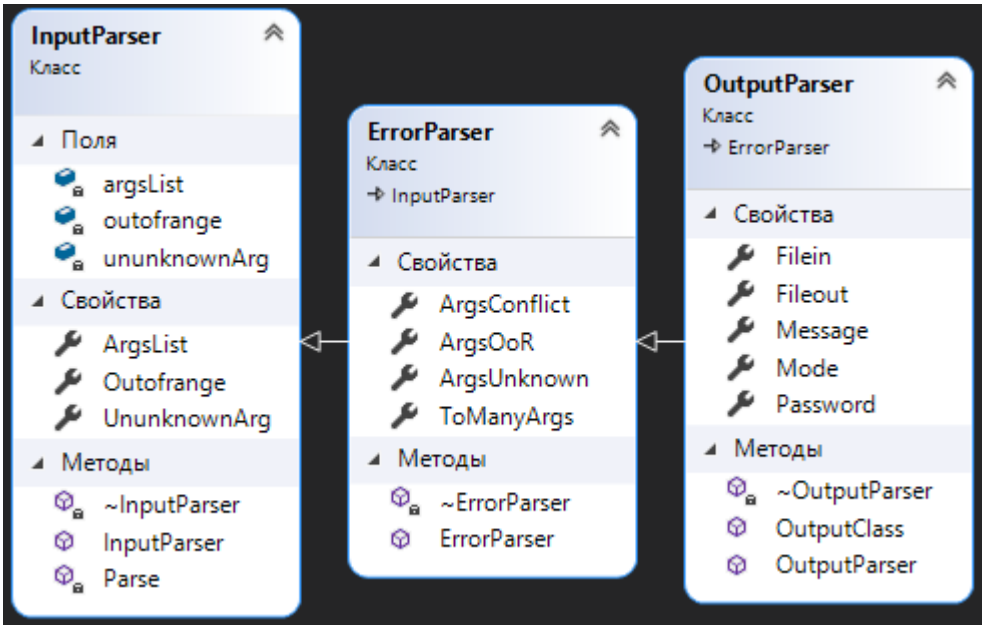

Рисунок 5 – Схема семейства классов разбора пользовательского ввода

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).

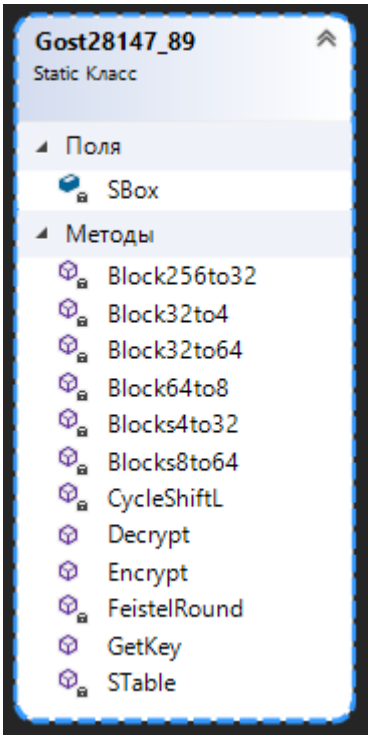

Рисунок 6 – Схема криптографического класса

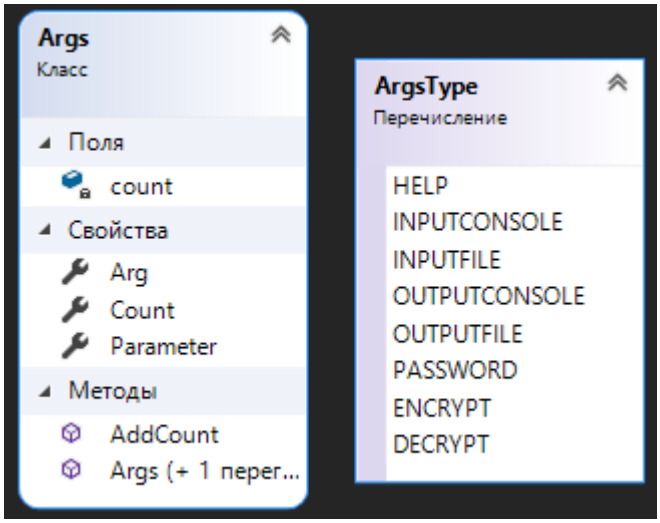

Рисунок 7 – Схема вспомогательных объектов программы

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

2.2 Метод Main

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

```
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        Stopwatch sw = new Stopwatch();
        OutputParser parser = new OutputParser(args);
        Object answer = parser.OutputClass();
        if (answer is HelpInfo help) help.Output();
        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: " +
            $"{sw.ElapsedMilliseconds} мс.");
    }
}
```

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.

2.3 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 N_1 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 N_1 блока складывается по модулю 2^{32} с N_2 блоком. Последняя операция раунда сети Фейстеля – замена блоков N_1 и N_2 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: k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_8, k_7, k_6, k_5, k_4, k_3, k_2, k_1. 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.

```
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();
}
```

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.

```
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();
}
```

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).

```
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;
}
```

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.

```
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);
}
```

Рисунок 12 – Метод замены содержимого N блока по S блоку

```
private byte[] Block32to4(UInt32 block32) // Разбиение 32-битного блока на 8-ми битные блоки
{                                         // (c# не умеет делать 4-битные блоки)
    //массив 4-х байт (эквивалент 8-элементному массиву 4-х битных символов)
    byte[] blocks4 = new byte[4];
    for (int i = 0; i < 4; i++)
        //на каждом шаге запоминается первые 8 регистров сдвинутого
        //на 8 единиц вправо 32-х битного блока
        blocks4[i] = (byte)(block32 >> (24 - 8 * i));
    //10000011000000011100110100011000 -> 00000000100000110000000011001101
    //(запомнило 0001 и 1000)
    //00000000100000110000000111001101 -> 00000000000000001000001100000001
    //(запомнило 1100 и 1101)
    //00000000000000001000001100000001 -> 00000000000000000000000010000011
    //(запомнило 0000 и 0001)
    //00000000000000000000000010000011 -> 00000000000000000000000000000000
    //(запомнило 1000 и 0011)
    return blocks4;
}
```

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.

```
private UInt32 Blocks4to32(byte[] blocks4) // Объединение 8-(4-) битных блоков в один 32-битный
{
    UInt32 block32 = 0;
    for (int i = 0; i < 4; i++)
        //двигает 32-битный блок влево на 8 бит
        //и проводит операцию или с первыми 8-ю регистрами
        block32 = (block32 << 8) | blocks4[i];
    //при входной информации 10000000 11111111 00011000 11110000 получит:
    //00000000000000000000000000000000 -> 00000000000000000000000010000000
    //00000000000000000000000010000000 -> 00000000000000001000000011111111
    //00000000000000001000000011111111 -> 00000000100000001111111100011000
    //00000000100000001111111100011000 -> 10000000111111110001100011110000
    return block32;
}
```

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.

```
private UInt32 CycleShiftL(UInt32 block32, int n) // Циклический сдвиг 32-битного блока влево на n бит
{
    //проводит дизъюнкцию сдвинутого влево на n бит 32-битного блока
    //со сдвинутым вправо на 32-n бит того же 32-битного блока
    return (UInt32)((block32 << n) | (block32 >> (32 - n)));
    //при n = 11 и block32 = 11111111110100000000000000000000
    //block32 << n =         10000000000000000000000000000000
    //(block32 >> (32 - n) = 00000000000000000000011111111110
    //их дизъюнкция равна    10000000000000000000011111111110
}
```

Рисунок 15 – Метод циклического сдвига влево на n бит

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

```
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);
}
```

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.

2.4 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).

```
class IObase
{

    public virtual void Output()
    {
        Console.WriteLine("BASED");
    }
}
```

Рисунок 17 – Базовый класс IObase

```
class HelpInfo: IObase
{
    Ссылок: 0
    ~HelpInfo() { }
    Ссылок: 10
    public override void Output()
    {
        string help =
            "\n\t\t    Gost28147-89\n\n" +
            " -e \t\t\t\tпараметр шифрования\n" +
            " -d \t\t\t\tпараметр дефрования\n" +
            " -h (--help)\t\t\tсправка\n" +
            " -t (--text)\t\t\tшифруемый текст\n"+
            " -p (--password)\t\tпароль\n" +
            " -in *имя входного файла*\n"+
            " -out *имя выходного файла*\n";
        Console.Write(help);
    }
}
```

Рисунок 18 – Класс HelpInfo, наследуемый от IObase

```
class IOcrypt: IObase
{
    Ссылок: 12
    public byte[] Data { get; set; }
    Ссылок: 9
    public byte[] Key { get; }
    Ссылок: 5
    public ArgsType Mode { get; }
    Ссылок: 2
    public IOcrypt(string password, ArgsType mode)
    {
        Key = Gost28147_89.GetKey(password);
        Mode = mode;
    }
    Ссылок: 0
    ~IOcrypt() { }
}
```

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).

```
class ConsoleInOut: IOcrypt
{
    public ConsoleInOut(string message, string password, ArgsType mode) : base(password, mode)
    {
        Data = Encoding.Default.GetBytes(message);
    }

    ~ConsoleInOut() { }

    public new 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)));
    }
}
```

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.

```
class FileStreamer: IOcrypt
{
    Ссылок: 6
    public FileStream StreamIN { get; }
    Ссылок: 5
    public FileStream StreamOUT { get; }
    Ссылок: 3
    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;
    }
    Ссылок: 0
    ~FileStreamer()
    {
        if (StreamIN != null)
        {
            StreamIN.Dispose();
            StreamIN.Close();
        }
        if (StreamOUT != null)
        {
            StreamOUT.Dispose();
            StreamOUT.Close();
        }
    }
    Ссылок: 2
    public byte[] FileRead()
    {
        int len = (int)StreamIN.Length;
        byte[] message = new byte[len];
        StreamIN.Read(message, 0, len);
        return message;
    }
    Ссылок: 4
    public void FileWrite(byte[] message)
    {
        StreamOUT.Write(message, 0, message.Length);
    }
}
```

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)

```
class ConsoleInFileOut: FileStreamer
{
    ссылка: 1
    public ConsoleInFileOut(string message, string file, string password, ArgsType mode) : base("", file, password, mode)
    {
        Data = System.Text.Encoding.Default.GetBytes(message);
    }
    Ссылок: 0
    ~ConsoleInFileOut() { }
    ссылка: 1
    public new void Output()
    {
        if (Mode == ArgsType.ENCRYPT) FileWrite(Gost28147_89.Encrypt(Data, Key));
        else FileWrite(Gost28147_89.Decrypt(Data, Key));
    }
}
```

Рисунок 22 – Класс ConsoleInFileOut, наследуемый от FileStreamer

```
class FileInOut: FileStreamer
{
    ссылка: 1
    public FileInOut(string fileIN, string fileOUT, string password, ArgsType mode) : base(fileIN, fileOUT, password, mode)
    {
        Data = FileRead();
    }
    Ссылок: 0
    ~FileInOut() { }

    ссылка: 1
    public new void Output()
    {
        if (Mode == ArgsType.ENCRYPT) FileWrite(Gost28147_89.Encrypt(Data, Key));
        else FileWrite(Gost28147_89.Decrypt(Data, Key));
    }
}
```

Рисунок 23 – Класс FileInOut, наследуемый от FileStreamer

```
class FileInConsoleOut: FileStreamer
{
    ссылка: 1
    public FileInConsoleOut(string fileIN, string password, ArgsType mode) : base(fileIN, "", password, mode)
    {
        Data = FileRead();
    }
    Ссылок: 0
    ~FileInConsoleOut() { }

    ссылка: 1
    public new 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)));
    }
}
```

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.

2.5 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

```
class InputParser
{
    private bool outofrange;
    ссылка: 1
    public bool Outofrange { get { return outofrange; } }
    private bool ununknownArg;
    ссылка: 1
    public bool UnunknownArg { get { return ununknownArg; } }
    readonly private Dictionary<ArgsType, Args> argsList;
    Ссылок: 26
    public Dictionary<ArgsType, Args> ArgsList { get { return argsList; } }
    ссылка: 1
    public InputParser(string [] args)
    {
        argsList = new Dictionary<ArgsType, Args>();
        Parse(args);
        if (!argsList.ContainsKey(ArgsType.OUTPUTFILE)) argsList.Add(ArgsType.OUTPUTCONSOLE, new Args(ArgsType.OUTPUTCONSOLE));
    }
    Ссылок: 0
    ~InputParser(){ }

    ссылка: 1
    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")...
                else if (args[i].ToLower() == "-d")...
                else if (args[i].ToLower() == "-t" | args[i].ToLower() == "--text")...
                else if (args[i].ToLower() == "-in")...
                else if (args[i].ToLower() == "-out")...
                else if (args[i].ToLower() == "-p" | args[i].ToLower() == "--password")...
                else if (args[i][0] == '-') ununknownArg = true;
            }
            catch { outofrange = true; }
        }
    }
}
```

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).

```
class ErrorParser: InputParser
{
    Ссылок: 4
    public Exception ArgsConflict { get; }
    Ссылок: 4
    public Exception ArgsOoR { get; }
    Ссылок: 3
    public Exception ArgsUnknown { get; }
    Ссылок: 4
    public Exception ToManyArgs { get; }
    ссылка: 1
    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");
        }
    }
    Ссылок: 0
    ~ErrorParser() { }
}
```

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).

```
Ссылок: 13
class Args
{
    private int count;
    Ссылок: 3
    public ArgsType Arg { get; }
    ссылка: 1
    public int Count { get { return count; } }
    Ссылок: 6
    public string Parameter { get; }

    Ссылок: 4
    public Args(ArgsType arg, string parameter)
    {
        Arg = arg;
        count = 1;
        Parameter = parameter;
    }

    Ссылок: 4
    public Args(ArgsType arg)
    {
        Arg = arg;
        count = 1;
        Parameter = null;
    }

    Ссылок: 7
    public void AddCount()
    {
        count++;
    }
}
```

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).

```
class OutputParser : ErrorParser
{
    Ссылок: 5
    public string Password { get; }
    Ссылок: 3
    public string Message { get; }
    Ссылок: 3
    public string Filein { get; }
    Ссылок: 3
    public string Fileout { get; }
    Ссылок: 5
    public ArgsType Mode { get; }
    ссылка: 1
    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;
    }
    Ссылок: 0
    ~OutputParser() { }
    ссылка: 1
    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;
    }
}
```

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;
    }
}
```