Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

ОТЧЁТ

по лабораторной работе №1

По дисциплине «Методы защиты информации»

По теме «Симметричная криптография. Двойной и тройной DES. ГОСТ 28147-89»

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик В.В.

Минск 2021

## **1. Введение**

В современном мире остро стоит вопрос о безопасности. Обеспечение безопасности является важным аспектом деятельности любой компании. Для обеспечения безопасности используется множество различных средств, как аппаратных, так и программных. Одним из таких средств является шифрование данных.

Шифрование — обратимое преобразование информации в целях сокрытия от [неавторизованных](https://ru.wikipedia.org/wiki/Авторизация) лиц, с предоставлением, в это же время, [авторизованным](https://ru.wikipedia.org/wiki/Авторизация) пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения [конфиденциальности](https://ru.wikipedia.org/wiki/Конфиденциальность) передаваемой информации.

Одним из самых распространенных алгоритмов для шифрования является DES и его модификации. DES — алгоритм для [симметричного](https://ru.wikipedia.org/wiki/Симметричный_шифр) [шифрования](https://ru.wikipedia.org/wiki/Шифрование), разработанный фирмой [IBM](https://ru.wikipedia.org/wiki/IBM) и утвержденный правительством [США](https://ru.wikipedia.org/wiki/США) в 1977 году как официальный стандарт. Размер блока для DES равен 64 [битам](https://ru.wikipedia.org/wiki/Бит). В основе алгоритма лежит [сеть Фейстеля](https://ru.wikipedia.org/wiki/Сеть_Фейстеля) с 16 циклами ([раундами](https://ru.wikipedia.org/wiki/Раунд_(в_криптографии))) и [ключом](https://ru.wikipedia.org/wiki/Ключ_(криптография)), имеющим длину 56 [бит](https://ru.wikipedia.org/wiki/Бит). Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

В ходе данной лабораторной работы необходимо изучить теоретическую часть об алгоритме шифрования DES, который необходимо использовать для шифрования данных, и на ее основании создать приложение, реализующее алгоритм DES.

1. **Теоретическая часть**

## **2.1. Алгоритм шифрования DES**

DES (англ. Data Encryption Standard) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований. Прямым развитием DES в настоящее время является алгоритм Triple DES (3DES). В 3DES шифрование/расшифровка выполняются путём троекратного выполнения алгоритма DES.

***Схема шифрования с помощью алгоритма DES:***

* ***Начальная перестановка***

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М- это произвольные 64 бита, то X = IP (M)-переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

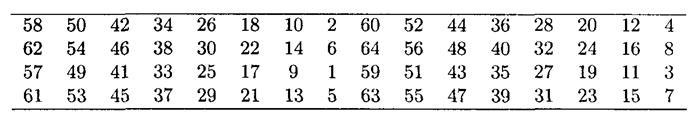


Рисунок 1 - DES. Начальная перестановка

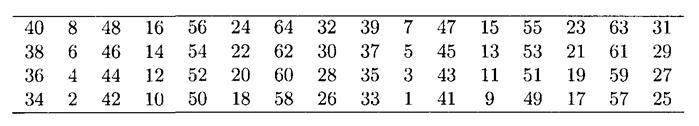


Рисунок 2 - DES. Заключительная перестановка

* ***Последовательность преобразований отдельного раунда***

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.

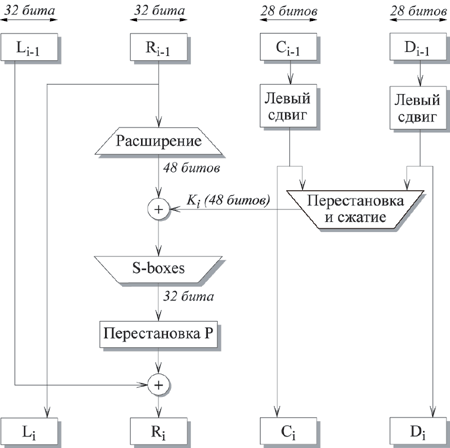


Рисунок 3 - i-ый раунд DES

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32-битные значения, обозначенные *L* и *R*. Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1 F(Ri-1, Ki)

Где обозначает операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri*является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно.

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Ri расширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения

. . . efgh ijkl mnop . . .

то в результате расширения получается сообщение

. . . defghi hijklm lmnopq . . .

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми *S-boxes,* каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

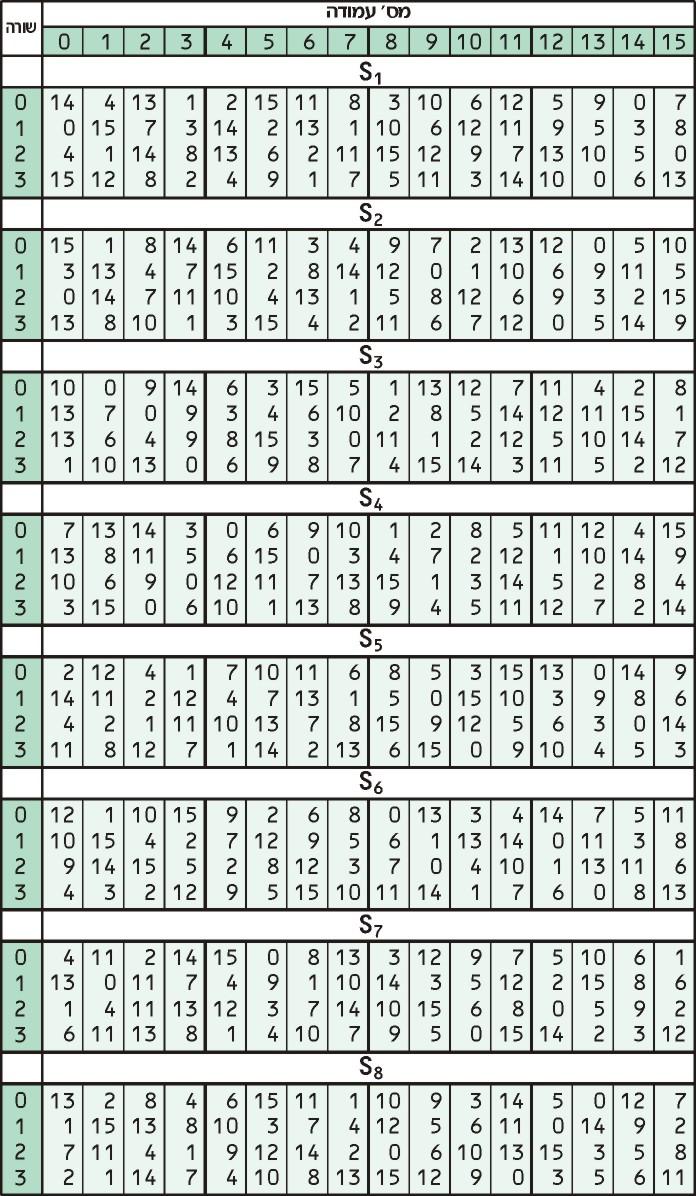


Рисунок 4 - S-boxes

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

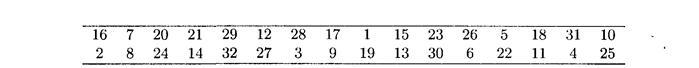


Рисунок 5 - Перестановка с помощью Р-блоков

* ***Создание подключей***

Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма (если используется 64-битный ключ, то, как видно из рис. 6 убираются биты 64, 56, 48, 40, 32, 16, 8), вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1).

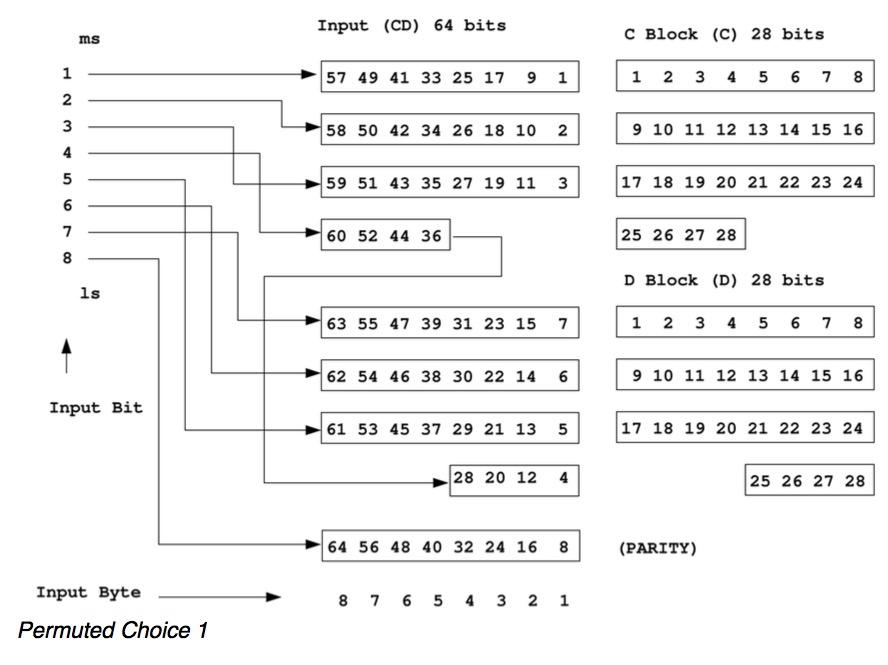


Рисунок 6 - Схема Permuted Choice

Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *цикла*.

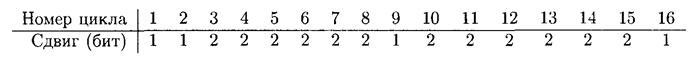


Рисунок 7 - Сдвиг ключа в зависимости от номера цикла

Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

Процесс ***дешифрования*** аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

## **2.2. Двойной DES**

Наиболее логичным способом противодействия полному перебору ключа DES выглядит многократное шифрование данных алгоритмом DES с различными ключами. Следующий алгоритм получил название Double DES (двойной DES):

где: — половины двойного ключа алгоритма Double DES, каждая из которых представляет собой обычный 56-битный ключ DES; Е — функция шифрования блока данных обычным алгоритмом DES. Если бы при двойном шифровании DES выполнялось следующее свойство:  для любых значений, то двойное шифрование не приводило бы к усилению против полного перебора ключа — всегда нашелся бы такой ключ *k*, *однократное* шифрование которым было бы эквивалентно двукратному шифрованию на ключах, а для нахождения ключа к достаточно было бы перебрать 255 ключей. Double DES действительно удваивает эффективный размер ключа— до 112 битов, а при современном развитии вычислительной техники полный перебор 112-битного ключа невозможен.

## **2.3. Тройной DES**

Triple DES (3DES) — симметричный [блочный шифр](https://ru.wikipedia.org/wiki/Блочный_шифр), созданный [Уитфилдом Диффи](https://ru.wikipedia.org/wiki/Диффи,_Уитфилд), [Мартином Хеллманом](https://ru.wikipedia.org/wiki/Хеллман,_Мартин) и Уолтом Тачманном в [1978 году](https://ru.wikipedia.org/wiki/1978_год) на основе алгоритма [DES](https://ru.wikipedia.org/wiki/DES) с целью устранения главного недостатка последнего — малой длины ключа (56 бит), который может быть взломан методом [полного перебора](https://ru.wikipedia.org/wiki/Полный_перебор) ключа. Скорость работы 3DES в 3 раза ниже, чем у DES, но криптостойкость намного выше — время, требуемое для криптоанализа 3DES, может быть в миллиард раз больше, чем время, нужное для вскрытия DES. 3DES используется чаще, чем DES, который легко взламывается при помощи сегодняшних технологий (в [1998 году](https://ru.wikipedia.org/wiki/1998_год) организация [Electronic Frontier Foundation](https://ru.wikipedia.org/wiki/Electronic_Frontier_Foundation), используя специальный компьютер *DES Cracker*, вскрыла DES за 3 дня). 3DES является простым способом устранения недостатков DES. Алгоритм 3DES построен на основе DES, поэтому для его реализации возможно использовать программы, созданные для DES. Официальное название алгоритма, используемое в стандартах - TDEA или Triple DEA (англ. Triple Data Encryption Algorithm). Однако, термин "3DES" используется более широко поставщиками, пользователями и разработчиками криптосистем.

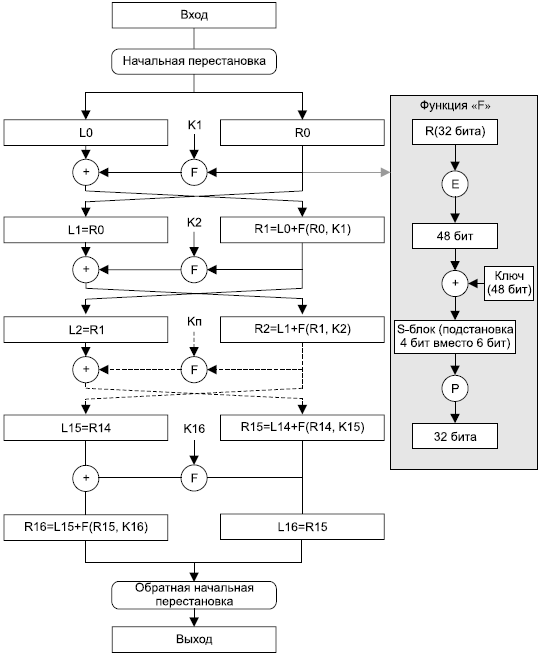
Существует 2 основных варианта шифрования алгоритмом 3DES: 3-key Triple DES и 2-key Triple DES. Как видно из названия, принципиальное различие этих методов - количество ключей (три и два соответственно). В свою очередь, каждый из этих алгоритмов имеет по 2 разных типа: EEE (encryption-encryption-encryption) и EDE (encryption-decryption-encryption) шифрование. Начнем с трех ключей:

* DES-EEE3: Исходный текст шифруется три раза, используя разные ключи.
* DES-EDE3: Исходный текст шифруется, затем дешифруется (уже другим ключом), затем снова шифруется (третьим ключом). Наглядное представление можно увидеть ниже.
* DES-EEE2: Исходный текст шифруется три раза, однако ключи на первом и последнем шаге одинаковые.
* DES-EDE2: Исходный текст шифруется, затем дешифруется (другим ключом), затем снова шифруется (ключом, используемым при первом шифровании).

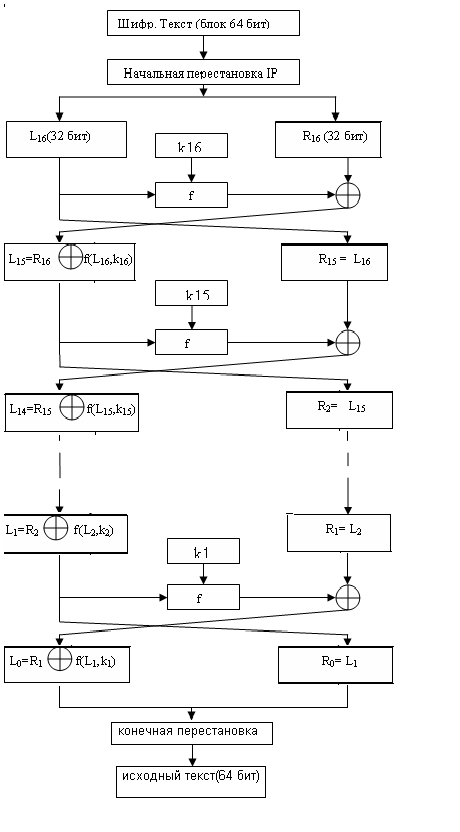
На практике, самый используемый тип 3DES шифрования - DES-EDE3.

## **3. Схема алгоритма**

Общая схема алгоритма *кодирования* для DES.



Общая схема алгоритма *декодирования* для DES.



## **4. Пример работы программы**

Рассмотрим пример программы. В исходном файле содержится текст :

“A traditional English game played on grass. Players use long wooden hammers called mallets to hit wooden balls along the grass and through arches made of curved pieces of metal.”

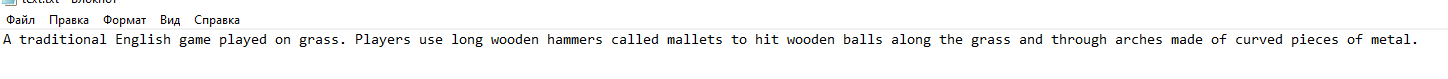


Рисунок 1. Исходный файл

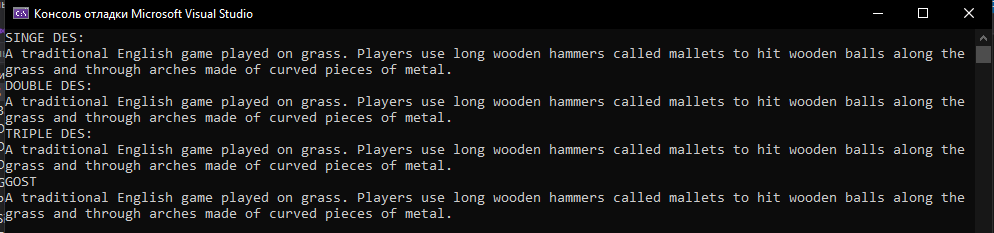


Рисунок 2. Результат шифрования при помощи алгоритмов шифрования

## **5. Текст программы**

## **Program.cs**

using System;

using System.Text;

using System.Collections.Generic;

using System.Linq;

using System.IO;

namespace DES

{

class Program

{

static void Main(string[] args)

{

TaskDES("text.txt");

TaskGOST("text.txt");

}

public static void TaskDES(string filename)

{

string text = Read(filename);

DESDerivedKey desKey = new DESDerivedKey(Utils.RandomKey());

DESDerivedKey desKey1 = new DESDerivedKey(Utils.RandomKey());

DESDerivedKey desKey2 = new DESDerivedKey(Utils.RandomKey());

string binText = Utils.TextToBin(text);

string cipher\_data1 = SingeDES.Encrypt(binText, desKey);

string decryptData1 = SingeDES.Decrypt(cipher\_data1, desKey);

Console.WriteLine($"SINGE DES:\n{Utils.BinToText(decryptData1)}");

string cipher\_data2 = DoubleDES.Encrypt(binText, desKey, desKey1);

string decryptData2 = DoubleDES.Decrypt(cipher\_data2, desKey, desKey1);

Console.WriteLine($"DOUBLE DES:\n{Utils.BinToText(decryptData2)}");

string cipher\_data3 = TripleDES.Encrypt(binText, desKey, desKey1, desKey2);

string decryptData3 = TripleDES.Decrypt(cipher\_data3, desKey, desKey1, desKey2);

Console.WriteLine($"TRIPLE DES:\n{Utils.BinToText(decryptData3)}");

}

public static void TaskGOST(string filename)

{

List<byte> full\_encode = new List<byte>();

List<byte> full\_decode = new List<byte>();

string text = Read(filename);

byte[] testBytes = Encoding.UTF8.GetBytes(text);

uint[] key = { 12, 25, 11, 86, 0, 0, 1, 1 };

uint[] rkey = Utils.GenerateKey();

byte[] encode;

byte[] decode;

int add = 8 - testBytes.Length % 8;

var textBytes = testBytes.ToList();

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

{

textBytes.Add(0);

}

GOST gost = new GOST(rkey);

for (int i = 0; i < textBytes.Count / 8; i++)

{

var val = textBytes.Skip(i \* 8).Take(8).ToArray();

encode = gost.Encrypt(val);

full\_encode.AddRange(encode);

}

for (int i = 0; i < textBytes.Count / 8; i++)

{

var val = full\_encode.Skip(i \* 8).Take(8).ToArray();

decode = gost.Decrypt(val);

full\_decode.AddRange(decode);

}

string encodedText = Encoding.UTF8.GetString(full\_decode.ToArray());

Console.WriteLine("GOST\n" + Encoding.UTF8.GetString(full\_decode.ToArray()));

}

public static void Write(string data, string filename)

{

using (FileStream fs = new FileStream(filename, FileMode.OpenOrCreate))

{

byte[] array = Encoding.UTF8.GetBytes(data);

fs.Write(array, 0, array.Length);

}

}

public static string Read(string filename)

{

using (FileStream fs = File.OpenRead(filename))

{

byte[] array = new byte[fs.Length];

fs.Read(array, 0, array.Length);

return Encoding.UTF8.GetString(array);

}

}

}

}

## **DESCrypto.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public class DESCrypto

{

private static string Xor(string a, string b)

{

StringBuilder ans = new StringBuilder();

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

{

if (a[i] == b[i])

{

ans.Append("0");

}

else

{

ans.Append("1");

}

}

return ans.ToString();

}

public static string Encrypt(string text, DESDerivedKey key)

{

//StringBuilder message = new StringBuilder(Utils.HexToBin(text));

StringBuilder message = new StringBuilder((text));

if (message.Length % 64 != 0)

{

int add = 64 - (message.Length % 64);

message.Append('0', Math.Max(0, add));

}

StringBuilder result = new StringBuilder();

for (int q = 0; q < message.Length / 64; q++)

{

string pt = message.ToString().Substring(q \* 64, 64);

int[] initial\_perm = {58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7};

pt = Utils.Permute(pt, initial\_perm, 64);

string left = pt.Substring(0, 32);

string right = pt.Substring(32);

// Expansion D-box Table

int[] exp\_d = {32, 1, 2, 3, 4, 5, 4, 5,

6, 7, 8, 9, 8, 9, 10, 11,

12, 13, 12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21, 20, 21,

22, 23, 24, 25, 24, 25, 26, 27,

28, 29, 28, 29, 30, 31, 32, 1};

// S-box Table

int[,,] s = {{{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},

{0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},

{4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},

{15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}},

{{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},

{3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},

{0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},

{13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}},

{{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},

{13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},

{13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},

{1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}},

{{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},

{13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},

{10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},

{3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}},

{{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},

{14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},

{4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},

{11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}},

{{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},

{10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},

{9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},

{4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}},

{{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},

{13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},

{1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},

{6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}},

{{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},

{1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},

{7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},

{2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}}};

// Straight Permutation Table

int[] per = { 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

for (int i = 0; i < 16; i++)

{

string right\_expanded = Utils.Permute(right, exp\_d, 48);

string x = Xor(key.rkb[i], right\_expanded);

StringBuilder op = new StringBuilder();

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

{

int row = 2 \* (x[j \* 6] - '0') + (x[j \* 6 + 5] - '0');

int col = 8 \* (x[j \* 6 + 1] - '0') + 4 \* (x[j \* 6 + 2] - '0') + 2 \* (x[j \* 6 + 3] - '0') + (x[j \* 6 + 4] - '0');

int val = s[j, row, col];

op.Append(val / 8);

val %= 8;

op.Append(val / 4);

val %= 4;

op.Append(val / 2);

val %= 2;

op.Append(val);

}

string op2 = Utils.Permute(op.ToString(), per, 32);

x = Xor(op2, left);

left = x;

if (i != 15)

{

string temp = left;

left = right;

right = temp;

}

}

string combine = left + right;

int[] final\_perm = {40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25};

//result.Append(Utils.BinToHex(Utils.Permute(combine, final\_perm, 64)));

result.Append((Utils.Permute(combine, final\_perm, 64)));

}

return result.ToString();

}

}

}

**DESDerivedKey.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public class DESDerivedKey

{

public List<string> rkb;

public DESDerivedKey(string key)

{

int[] keyp = { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4 };

key = Utils.Permute(key, keyp, 56); // key without parity

int[] shift\_table = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

int[] key\_comp = { 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32 };

string left = key.Substring(0, 28);

string right = key.Substring(28);

List<string> rkb = new List<string>();

for(int i = 0; i < 16; i++)

{

left = ShiftLeft(left, shift\_table[i]);

right = ShiftLeft(right, shift\_table[i]);

string combine = string.Concat(left, right);

string roundKey = Utils.Permute(combine, key\_comp, 48);

rkb.Add(roundKey);

}

this.rkb = rkb;

}

private string ShiftLeft(string ks,int shifts)

{

StringBuilder k = new StringBuilder(ks);

StringBuilder s = new StringBuilder();

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

{

for (int j = 1; j < 28; j++)

{

s.Append(k[j]);

}

s.Append(k[0]);

k = new StringBuilder(s.ToString());

//s.delete(0, s.length());

}

return k.ToString();

}

public void Reverse()

{

rkb.Reverse();

}

}

}

**DoubleDES.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public static class DoubleDES

{

public static string Encrypt(string data, DESDerivedKey key1, DESDerivedKey key2) =>

DESCrypto.Encrypt(DESCrypto.Encrypt(data, key1), key2);

public static string Decrypt(string data, DESDerivedKey key1, DESDerivedKey key2)

{

key1.Reverse();

key2.Reverse();

return DESCrypto.Encrypt(DESCrypto.Encrypt(data, key2), key1);

}

}

}

**SingeDES.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public static class SingeDES

{

public static string Encrypt(string data, DESDerivedKey key) =>

DESCrypto.Encrypt(data, key);

public static string Decrypt(string data, DESDerivedKey key)

{

key.Reverse();

return DESCrypto.Encrypt(data, key);

}

}

}

**TripleDES.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public static class TripleDES

{

public static string Encrypt(string data, DESDerivedKey key1, DESDerivedKey key2, DESDerivedKey key3) =>

DESCrypto.Encrypt(DESCrypto.Encrypt(DESCrypto.Encrypt(data, key1), key2),key3);

public static string Decrypt(string data, DESDerivedKey key1, DESDerivedKey key2, DESDerivedKey key3)

{

key1.Reverse();

key2.Reverse();

key3.Reverse();

return DESCrypto.Encrypt(DESCrypto.Encrypt(DESCrypto.Encrypt(data, key3), key2),key1);

}

}

}

**GOST.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public class GOST

{

const int BLOCK\_SIZE\_BYTES = 8;

readonly byte[,] s =

{

{ 4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3 },

{ 14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9 },

{ 5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11 },

{ 7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3 },

{ 6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2 },

{ 4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14 },

{ 13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12 },

{ 1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12 }

};

private uint[] key = new uint[8];

public GOST(uint[] key)

{

this.key = key;

}

private uint Substitute(uint value)

{

byte index, sBlock;

uint result = 0;

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

{

index = (byte)(value >> (4 \* i) & 0x0f);

sBlock = s[i, index];

result |= (uint)sBlock << (4 \* i);

}

return result;

}

private uint F(uint block, uint subKey)

{

block = (block + subKey) % uint.MaxValue;

block = Substitute(block);

block = (block << 11) | (block >> 21);

return block;

}

public byte[] Encrypt(byte[] inputBlock)

{

uint left = BitConverter.ToUInt32(inputBlock, 0);

uint right = BitConverter.ToUInt32(inputBlock, 4);

byte[] result = new byte[BLOCK\_SIZE\_BYTES];

for (int i = 0; i < 24; i++)

{

uint fResult = left ^ F(right, key[i % 8]);

left = right;

right = fResult;

}

for (int i = 24, j = 7; i < 31; i++, j--)

{

uint fResult = left ^ F(right, key[j]);

left = right;

right = fResult;

}

left ^= F(right, key[0]);

Array.Copy(BitConverter.GetBytes(left), 0, result, 0, 4);

Array.Copy(BitConverter.GetBytes(right), 0, result, 4, 4);

return result;

}

public byte[] Decrypt(byte[] inputBlock)

{

uint left = BitConverter.ToUInt32(inputBlock, 0);

uint right = BitConverter.ToUInt32(inputBlock, 4);

byte[] result = new byte[BLOCK\_SIZE\_BYTES];

for (int i = 31, j = 0; i >= 24; i--, j++)

{

uint fResult = left ^ F(right, key[j]);

left = right;

right = fResult;

}

for (int i = 23; i > 0; i--)

{

uint fResult = left ^ F(right, key[i % 8]);

left = right;

right = fResult;

}

left ^= F(right, key[0]);

Array.Copy(BitConverter.GetBytes(left), 0, result, 0, 4);

Array.Copy(BitConverter.GetBytes(right), 0, result, 4, 4);

return result;

}

}

}

**Utils.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DES

{

public static class Utils

{

public static uint[] GenerateKey()

{

Random rand = new Random();

uint[] arr = new uint[8];

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

{

arr[i] = (uint)rand.Next(0, 255);

}

return arr;

}

public static string RandomKey()

{

Random rand = new Random();

StringBuilder key = new StringBuilder();

for (int i = 0; i < 64; i++)

{

key.Append((char)('0' + rand.Next(2)));

}

return key.ToString();

}

public static string Permute(string k, int[] arr, int n)

{

StringBuilder per = new StringBuilder();

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

{

per.Append(k[arr[i] - 1]);

}

return per.ToString();

}

public static string TextToBin(string s)

{

byte[] bytes = Encoding.UTF8.GetBytes(s);

StringBuilder binary = new StringBuilder();

foreach (byte b in bytes)

{

int val = b;

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

{

binary.Append((val & 128) == 0 ? 0 : 1);

val <<= 1;

}

}

return binary.ToString();

}

public static string BinToText(string s)

{

StringBuilder text = new StringBuilder();

for (int i = 0; i <= s.Length - 8; i += 8)

{

text.Append((char)Convert.ToByte(s.Substring(i, 8), 2));

}

return text.ToString();

}

}

}

## **6. Вывод**

DES был национальным стандартом США в 1977—1980 гг., но в настоящее время DES используется (с ключом длины 56 бит) только для устаревших систем, чаще всего используют его более криптоустойчивый вид (3DES, DESX). 3DES является простой эффективной заменой DES, и сейчас он рассмотрен как стандарт. В ближайшее время DES и Triple DES будут заменены алгоритмом AES (Advanced Encryption Standard — Расширенный Стандарт Шифрования).

ГОСТ 28147-89 является устаревшим с 2019 года, а на его замену был принят ГОСТ 34.12-2018, который описывает шифры “Кузнечик” и “Магма”