МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**Факультет информационных технологий и робототехники (ФИТР)**

**Отчёт по лабораторной работе №2**

По дисциплине: «Основы защиты информации»

На тему: «АЛГОРИТМ ШИФРОВАНИЯ ДАННЫХ DES»

Выполнил: ст. г. 10701118 Васильков В.С.

Приняла: Белова С.В.

Минск 2020

**Цель работы**

Познакомиться с основами симметричного шифрования. Изучить

алгоритм шифрования DES на примере упрощенной версии S-DES.

**Результаты выполнения**

**Вариант 8**

Символ: f = 01100110

Ключ: 412

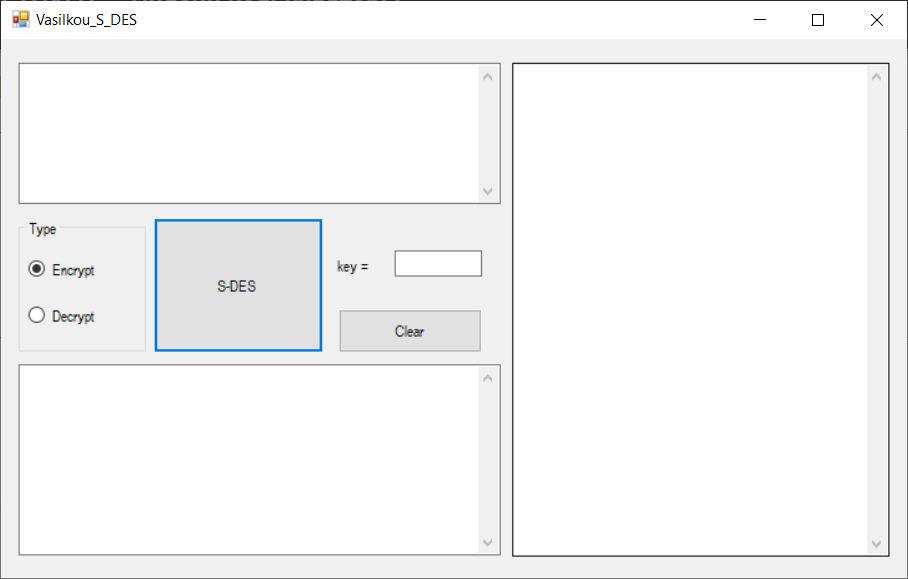


Рисунок 1 – Вид приложения

Для начала работы, необходимо поместить двоичное представление символа в верхнее текстовое поле и ввести ключ в отдельное поле. Шифрование происходит по нажатию на центральную кнопку «S-DES».

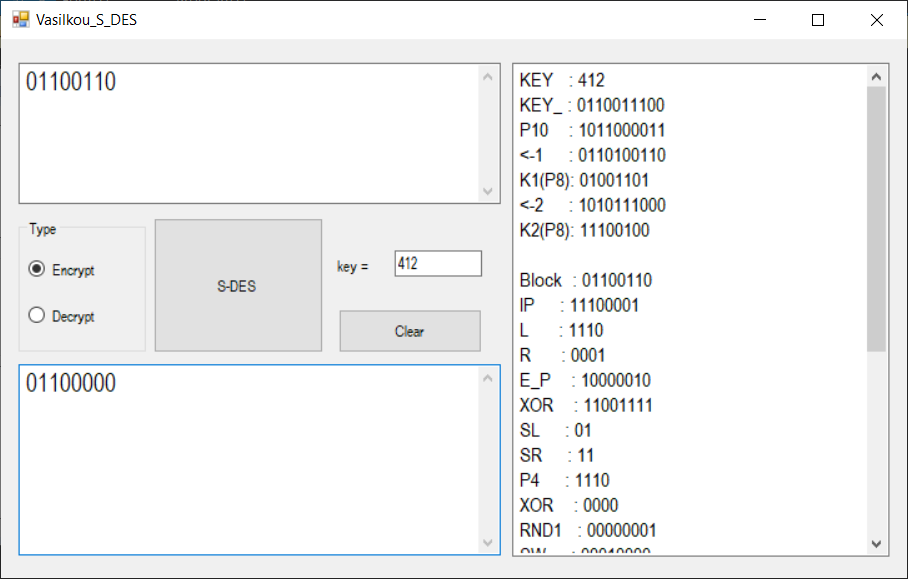


Рисунок 2 – Результат шифрования ч.1

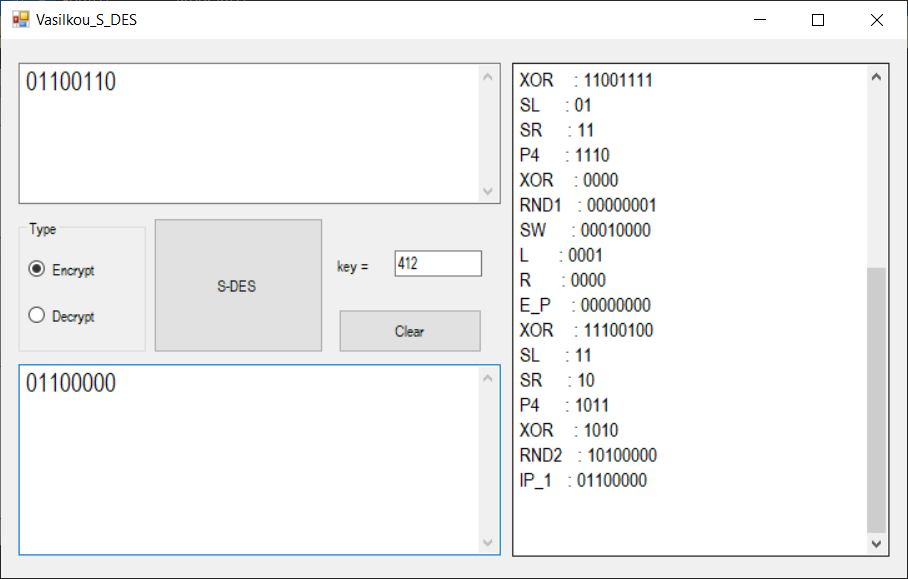


Рисунок 3 – Результат шифрования ч.2

**Вывод:** в ходе выполнения лабораторной работы были изучены основные алгоритмы симметричного шифрования DES, S-DES. Так же, было разработано приложение, реализующее алгоритм S-DES.

**Приложение А**

**Файл Form1.cs**

class S\_DES

{

internal static StringBuilder log;

internal static Int32[,] s\_l = new Int32[,]

{

{ 1, 0, 3, 2 },

{ 3, 2, 1, 0 },

{ 0, 2, 1, 3 },

{ 3, 1, 3, 1 },

};

internal static Int32[,] s\_r = new Int32[,]

{

{ 1, 1, 2, 3 },

{ 2, 0, 1, 3 },

{ 3, 0, 1, 0 },

{ 2, 1, 0, 3 },

};

// Преобразование данных и расчет ключей

public static Tuple<List<BitArray>, String> MakeS\_DES(String bitTextInStr, String keyText, Boolean type)

{

log = new StringBuilder();

log.Append(String.Format("{0, -5} : {1}", "KEY", keyText)).Append(Environment.NewLine);

BitArray keyBit = ToBitArray(Int32.Parse(keyText), 10);

log.Append(String.Format("{0, -5}: {1}", "KEY\_", ToString(keyBit))).Append(Environment.NewLine);

BitArray key = RefreshArray(keyBit, Permutations.P10);

log.Append(String.Format("{0, -5} : {1}", "P10", ToString(key))).Append(Environment.NewLine);

MoveLeft(key, 1);

log.Append(String.Format("{0, -5} : {1}", "<-1", ToString(key))).Append(Environment.NewLine);

BitArray k1 = RefreshArray(key, Permutations.P8);

log.Append(String.Format("{0, -5}: {1}", "K1(P8)", ToString(k1))).Append(Environment.NewLine);

MoveLeft(key, 2);

log.Append(String.Format("{0, -5} : {1}", "<-2", ToString(key))).Append(Environment.NewLine);

BitArray k2 = RefreshArray(key, Permutations.P8);

log.Append(String.Format("{0, -5}: {1}", "K2(P8)", ToString(k2))).Append(Environment.NewLine).Append(Environment.NewLine);

if (!type)

{

BitArray buf = new BitArray(k1);

k1 = new BitArray(k2);

k2 = new BitArray(buf);

}

List<BitArray> result = new List<BitArray>();

Int32 blockLength = 8;

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

{

String part = bitTextInStr.Substring(i, blockLength);

BitArray textBit = new BitArray(blockLength);

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

{

textBit.Set(j, part[j].Equals('1') ? true : false);

}

log.Append(String.Format("{0, -5} : {2}", "Block", part.ToString(), ToString(textBit))).Append(Environment.NewLine);

result.Add(MakeTransformation(textBit, k1, k2));

}

return Tuple.Create(result, log.ToString());

}

// Общая последовательность S-DES

private static BitArray MakeTransformation(BitArray openTextBitArray, BitArray key1, BitArray key2)

{

BitArray ip = RefreshArray(openTextBitArray, Permutations.IP);

log.Append(String.Format("{0, -5} : {1}", "IP", ToString(ip))).Append(Environment.NewLine); ;

BitArray round1 = MakeRound(ip, key1);

log.Append(String.Format("{0, -5} : {1}", "RND1", ToString(round1))).Append(Environment.NewLine);

SwapParts(round1);

log.Append(String.Format("{0, -5} : {1}", "SW", ToString(round1))).Append(Environment.NewLine);

BitArray round2 = MakeRound(round1, key2);

log.Append(String.Format("{0, -5} : {1}", "RND2", ToString(round2))).Append(Environment.NewLine);

BitArray ip\_1 = RefreshArray(round2, Permutations.IP\_1);

log.Append(String.Format("{0, -5} : {1}", "IP\_1", ToString(ip\_1))).Append(Environment.NewLine).Append(Environment.NewLine);

return ip\_1;

}

// Функция раунда

private static BitArray MakeRound(BitArray blockBitArray, BitArray key)

{

Tuple<BitArray, BitArray> tuple = Divide(blockBitArray);

BitArray leftPart = tuple.Item1;

BitArray rightPart = tuple.Item2;

log.Append(String.Format("{0, -5} : {1}", "L", ToString(leftPart))).Append(Environment.NewLine);

log.Append(String.Format("{0, -5} : {1}", "R", ToString(rightPart))).Append(Environment.NewLine);

BitArray e\_p = RefreshArray(rightPart, Permutations.E\_P);

log.Append(String.Format("{0, -5} : {1}", "E\_P", ToString(e\_p))).Append(Environment.NewLine);

BitArray \_ = e\_p.Xor(key);

log.Append(String.Format("{0, -5} : {1}", "XOR", ToString(\_))).Append(Environment.NewLine);

tuple = Divide(\_);

BitArray coLeft = MakeS\_Matrix(tuple.Item1, s\_l);

log.Append(String.Format("{0, -5} : {1}", "SL", ToString(coLeft))).Append(Environment.NewLine);

BitArray coRight = MakeS\_Matrix(tuple.Item2, s\_r);

log.Append(String.Format("{0, -5} : {1}", "SR", ToString(coRight))).Append(Environment.NewLine);

BitArray afterP4 = RefreshArray(Join(coLeft, coRight), Permutations.P4);

log.Append(String.Format("{0, -5} : {1}", "P4", ToString(afterP4))).Append(Environment.NewLine);

\_ = leftPart.Xor(afterP4);

log.Append(String.Format("{0, -5} : {1}", "XOR", ToString(\_))).Append(Environment.NewLine);

BitArray out\_ = Join(\_, rightPart);

return out\_;

}

// Преобразование числа в массив битов

public static BitArray ToBitArray(Int32 number, Int32 size)

{

String bitArrayStr = Convert.ToString(number, 2).PadLeft(size, '0');

BitArray arrayBit = new BitArray(size);

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

{

arrayBit.Set(i, bitArrayStr[i].Equals('1') ? true : false);

}

return arrayBit;

}

// Выбор последовательности для перестановки

private static BitArray RefreshArray(BitArray bitArray, Permutations permutation)

{

Int32[] sequence = new Int32[] { };

switch (permutation)

{

case Permutations.P10:

{

sequence = new Int32[] { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6 };

break;

}

case Permutations.P8:

{

sequence = new Int32[] { 6, 3, 7, 4, 8, 5, 10, 9 };

break;

}

case Permutations.IP:

{

sequence = new Int32[] { 2, 6, 3, 1, 4, 8, 5, 7 };

break;

}

case Permutations.E\_P:

{

sequence = new Int32[] { 4, 1, 2, 3, 2, 3, 4, 1 };

break;

}

case Permutations.P4:

{

sequence = new Int32[] { 2, 4, 3, 1 };

break;

}

case Permutations.IP\_1:

{

sequence = new Int32[] { 4, 1, 3, 5, 7, 2, 8, 6 };

break;

}

}

return \_RefreshArray(bitArray, sequence);

}

// Перестановки в массиве

private static BitArray \_RefreshArray(BitArray bitArray, Int32[] sequence)

{

BitArray buf = new BitArray(sequence.Length);

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

{

buf.Set(i, bitArray.Get(sequence[i] - 1));

}

return new BitArray(buf);

}

// Сдвиг влево на количество позиций

public static void MoveLeft(BitArray bitArray, Int32 position)

{

for(Int32 i = 0; i < position; i++)

{

Int32 center = bitArray.Length / 2;

Boolean bufLeft = bitArray.Get(0), bufRight = bitArray.Get(center);

for (Int32 j = 1; j < center; j++)

{

bitArray.Set(j - 1, bitArray.Get(j));

bitArray.Set(j - 1 + center, bitArray.Get(j + center));

}

bitArray.Set(center - 1, bufLeft);

bitArray.Set(center \* 2 - 1, bufRight);

}

}

// Деление массива на две равные части

public static Tuple<BitArray, BitArray> Divide(BitArray bitArray)

{

Int32 newLength = bitArray.Length / 2;

BitArray leftPart = new BitArray(newLength);

BitArray rightPart = new BitArray(newLength);

for (Int32 i = 0; i < newLength; i++)

{

leftPart.Set(i , bitArray.Get(i));

rightPart.Set(i, bitArray.Get(i + newLength));

}

return Tuple.Create(leftPart, rightPart);

}

// S-матрица

public static BitArray MakeS\_Matrix(BitArray bitArray, Int32[,] table)

{

Int32 row = Convert.ToInt32(bitArray.Get(0)) \* 2 + Convert.ToInt32(bitArray.Get(3)) ;

Int32 column = Convert.ToInt32(bitArray.Get(1)) \* 2 + Convert.ToInt32(bitArray.Get(2)) ;

Int32 number = table[row, column];

return ToBitArray(number, 2);

}

// Объединение двух массивов

public static BitArray Join(BitArray bitArray1, BitArray bitArray2)

{

Int32 size1 = bitArray1.Length;

Int32 size2 = bitArray2.Length;

BitArray newArray = new BitArray(size1 + size2);

for(int i = 0; i < size1; i++)

{

newArray.Set(i, bitArray1.Get(i));

}

for (int i = 0; i < size2; i++)

{

newArray.Set(i + size1, bitArray2.Get(i));

}

return newArray;

}

// Перестановка левой и правой части массива

private static void SwapParts(BitArray bitArray)

{

int center = bitArray.Length / 2;

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

{

Boolean buf = bitArray.Get(i);

bitArray.Set(i, bitArray.Get(i + center));

bitArray.Set(i + center, buf);

}

}

public static String ToString(List<BitArray> bitArrayList)

{

StringBuilder sb = new StringBuilder();

foreach (BitArray bitArray in bitArrayList)

{

foreach (Boolean b in bitArray)

{

sb.Append(b ? 1 : 0);

}

}

return sb.ToString();

}

public static String ToString(BitArray bitArray)

{

StringBuilder sb = new StringBuilder();

foreach (Boolean b in bitArray)

{

sb.Append(b ? 1 : 0);

}

return sb.ToString();

}

}