Федеральное агентство связи

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Информационная безопасность»

Отчёт по лабораторной работе №1

по дисциплине

«Методы и средства защиты компьютерной информации»

Изучение алгоритмов симметричного шифрования

Выполнил:

студент группы БВТ1301

Редько Е.Ю.

Проверил:

ассистент кафедры ИБ

Барков В.В.

Москва 2016

## Цель работы

Изучить принципы работы алгоритмов симметричного шифрования на примере реализации одного из алгоритмов.

## Индивидуальное задание

DES (Data Encryption Standard)

**Описание алгоритма**

DES – блочный алгоритм, то есть при шифровании исходное сообщение переводится в двоичный код, а затем разбивается на блоки и каждый блок отдельно зашифровывается (расшифровывается). По стандарту размер блока DES равен 64 бита.

Чтобы зашифровать сообщение алгоритмом DES, необходимо выполнить следующую последовательность шагов:

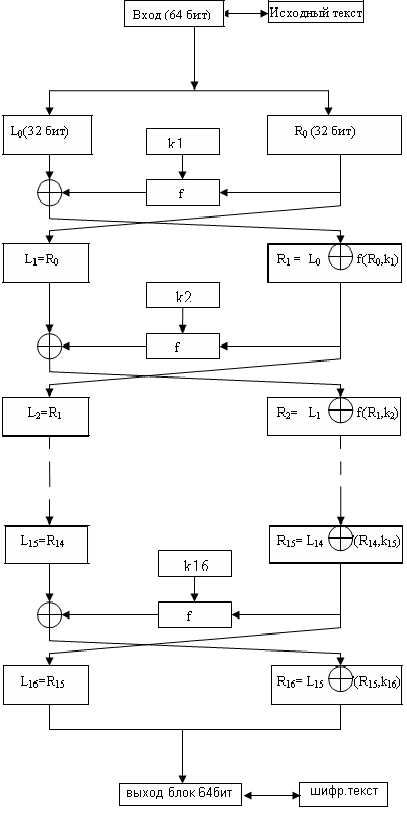
* довести исходное сообщение до такого размера, чтобы оно нацело делилось на размер блока;
* разделить исходное сообщение на блоки;
* довести длину ключа до длины половины блока;
* перевести ключ в бинарный формат;
* провести над каждым блоком прямое преобразование сетью Фейстеля в течение 16-ти раундов. После каждого раунда необходимо выполнять циклический сдвиг ключа на заданное количество символов;
* соединить все блоки вместе; таким образом получим сообщение, зашифрованное алгоритмом DES.

Расшифровка DES производится по аналогии. Используется обратное преобразование сетью Фейстеля.

Рассмотрим один раунд прямого преобразования сетью Фейстеля.

На i-й итерации исходный блок делится пополам – левая часть обозначается L, правая R. Над R и ключом ki вычисляется какая-либо выбранная логическая функция f. Затем выполняется вычисление логической операции “исключающее или” над L и вычисленным ранее значением функции (L xor f). Старое значение R переносится в левую часть блока, а в правую часть заносится значение L xor f. Последняя операция раунда – выполнить циклический сдвиг ключа: keyi+1 = keyi >> shiftKey (при расшифровке keyi-1 = keyi << shiftKey).

**Схема алгоритма**



**Код программы**

IDES.cs (интерфейс):

namespace RedkoLib

{

public interface IDES

{

void Encrypt(Stream inp, Stream outp);

void Decrypt(Stream inp, Stream outp);

}

}

DES.cs:

namespace RedkoLib

{

public class DES : IDES

{

string encryptKey;

string decryptKey;

string data;

string[] Blocks;

private const int blockSize = 64;

private const int charSize = 16;

private const int keyShift = 2;

private const int roundNumber = 16;

public DES(Stream inpKey)

{

byte[] buf = new byte[inpKey.Length];

inpKey.Read(buf, 0, (int)inpKey.Length);

encryptKey = Encoding.Default.GetString(buf, 0, buf.Length);

}

private string CompleteLength(string data)

{

while (((data.Length \* charSize) % blockSize) != 0)

data = "0" + data;

return data;

}

private void SplitBinaryIntoBlocks(string data)

{

Blocks = new string[data.Length / blockSize];

int lengthOfBlock = data.Length / Blocks.Length;

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

Blocks[i] = data.Substring(i \* lengthOfBlock, lengthOfBlock);

}

private void SplitStringIntoBlocks(string data)

{

Blocks = new string[(data.Length \* charSize) / blockSize];

int lengthOfBlock = data.Length / Blocks.Length;

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

{

Blocks[i] = data.Substring(i \* lengthOfBlock, lengthOfBlock);

Blocks[i] = StringToBinary(Blocks[i]);

}

}

private string CompleteKeyLength(string key, int keyLength)

{

if (key.Length > keyLength)

key = key.Substring(0, keyLength);

else

while (key.Length < keyLength)

key = "0" + key;

return key;

}

private string EncodeOneRound(string data, string key)

{

string L = data.Substring(0, data.Length / 2);

string R = data.Substring(data.Length / 2, data.Length / 2);

return (R + XOR(L, AND(R, key)));

}

private string DecodeOneRound(string data, string key)

{

string L = data.Substring(0, data.Length / 2);

string R = data.Substring(data.Length / 2, data.Length / 2);

return (XOR(AND(L, key), R) + L);

}

private string AND(string s1, string s2)

{

string result = "";

for (int i = 0; i < s1.Length; i++)

{

bool a = Convert.ToBoolean(Convert.ToInt32(s1[i].ToString()));

bool b = Convert.ToBoolean(Convert.ToInt32(s2[i].ToString()));

if (a & b)

result += "1";

else

result += "0";

}

return result;

}

private string XOR(string s1, string s2)

{

string result = "";

for (int i = 0; i < s1.Length; i++)

{

bool a = Convert.ToBoolean(Convert.ToInt32(s1[i].ToString()));

bool b = Convert.ToBoolean(Convert.ToInt32(s2[i].ToString()));

if (a ^ b)

result += "1";

else

result += "0";

}

return result;

}

private string NextKey(string key)

{

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

{

key = key[key.Length - 1] + key;

key = key.Remove(key.Length - 1);

}

return key;

}

private string PrevKey(string key)

{

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

{

key = key + key[0];

key = key.Remove(0, 1);

}

return key;

}

private string StringToBinary(string data)

{

string result = "";

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

{

string binaryChar = Convert.ToString(data[i], 2);

while (binaryChar.Length < charSize)

binaryChar = "0" + binaryChar;

result += binaryChar;

}

return result;

}

private string BinaryToString(string data)

{

string result = "";

while (data.Length > 0)

{

string char\_binary = data.Substring(0, charSize);

data = data.Remove(0, charSize);

int a = 0;

int degree = char\_binary.Length - 1;

foreach (char c in char\_binary)

a += Convert.ToInt32(c.ToString()) \* (int)Math.Pow(2, degree--);

result += ((char)a).ToString();

}

return result;

}

public void Encrypt(Stream inp, Stream outp)

{

byte[] buf = new byte[inp.Length];

inp.Read(buf, 0, (int)inp.Length);

data = Encoding.Default.GetString(buf, 0, buf.Length);

data = CompleteLength(data);

SplitStringIntoBlocks(data);

encryptKey = CompleteKeyLength(encryptKey, data.Length / (2 \* Blocks.Length));

encryptKey = StringToBinary(encryptKey);

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

{

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

Blocks[i] = EncodeOneRound(Blocks[i], encryptKey);

encryptKey = NextKey(encryptKey);

}

encryptKey = PrevKey(encryptKey);

decryptKey = BinaryToString(encryptKey);

String result = "";

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

result += Blocks[i];

result = BinaryToString(result);

byte[] byteArray = Encoding.Default.GetBytes(result);

outp.Write(byteArray, 0, Convert.ToInt32(byteArray.Length));

outp.Position = 0;

}

public void Decrypt(Stream inp, Stream outp)

{

byte[] buf = new byte[inp.Length];

inp.Read(buf, 0, (int)inp.Length);

data = Encoding.Default.GetString(buf, 0, buf.Length);

decryptKey = StringToBinary(decryptKey);

data = StringToBinary(data);

SplitBinaryIntoBlocks(data);

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

{

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

Blocks[i] = DecodeOneRound(Blocks[i], decryptKey);

decryptKey = PrevKey(decryptKey);

}

decryptKey = NextKey(decryptKey);

string result = "";

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

result += Blocks[i];

result = BinaryToString(result);

result = result.TrimStart('0');

byte[] byteArray = Encoding.Default.GetBytes(result);

outp.Write(byteArray, 0, byteArray.Length);

outp.Position = 0;

}

}

}

TDES.cs (тест):

namespace RedkoTest

{

[TestClass()]

public class TDES

{

[TestMethod()]

public void TestDES()

{

string data = "test message";

string key = "f93hnt57w";

byte[] inpByteArray = Encoding.Default.GetBytes(data);

MemoryStream inpStream = new MemoryStream(inpByteArray);

byte[] keyByteArray = Encoding.Default.GetBytes(key);

MemoryStream keyStream = new MemoryStream(keyByteArray);

Stream encrStream = new MemoryStream();

Stream decrStream = new MemoryStream();

DES target = new DES(keyStream);

target.Encrypt(inpStream, encrStream);

target.Decrypt(encrStream, decrStream);

byte[] buf = new byte[decrStream.Length];

decrStream.Read(buf, 0, (int)decrStream.Length);

string decrypted = Encoding.Default.GetString(buf, 0, buf.Length);

Assert.AreEqual(data, decrypted);

}

}

}

