МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования  
 «Ижевский государственный технический университет имени М.Т.Калашникова»

Факультет «Информатика и вычислительная техника»

Кафедра «Вычислительная техника»

Отчет

по лабораторной работе №2

на тему «Блочное симметричное шифрование»

по дисциплине «Методы и способы защиты компьютерной информации»

Выполнил:

Проверил:

студент группы Б08-781-1

Суровцева А.С.

к.т.н., доцент

Марков Е.М.

Ижевск 2020

**Цель работы**

Изучение структуры и основных принципов работы современных алгоритмов блочного симметричного шифрования, приобретение навыков программной реализации блочных симметричных шифров.

**Задание**

Реализовать систему симметричного блочного шифрования, позволяющую шифровать и дешифровать файл на диске с использованием ГОСТ 28147-89 шифра в EСВ режиме шифрования.

**Основные сведения**

Отечественным стандартом блочного шифрования является алгоритм ГОСТ 28147-89, который использует 256-битный ключ шифрования и 32 цикла преобразования 64-битных блоков исходного сообщения. Алгоритм реализует классическую схему сети Фейштеля. При этом образующая функция реализована следующим образом (рис.1) .

Сначала правая половина блока и ключ раунда складываются по модулю 232. Результат сложения разбивается на восемь 4-битовых последовательностей, каждая из которых поступает на вход соответствующего S-блока. Каждый блок представляет собой таблицу подстановки, которая заменяет поступающее на вход число в диапазоне [0..15] на другое число в том же диапазоне. Выходы всех S-блоков объединяются в 32-битное слово, которое затем циклически сдвигается влево на 11 битов и объединяется с левой частью блока операцией XOR.

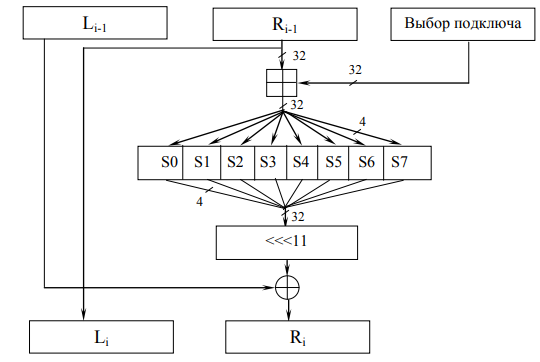


Рисунок 1 – Образующая функция алгоритма ГОСТ 28147-89

Формирование ключей раунда осуществляется по простой схеме. 256-битный ключ разбивается на восемь 32-битных машинных слов. Они нумеруются с К0 по К7. 32 ключа раунда получаются применением этих машинных слов в следующем порядке:

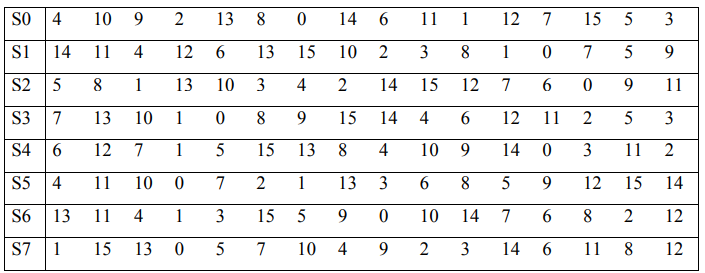


то есть в последних 8 раундах ключи подаются в обратном порядке.

Алгоритм ГОСТ является симметричным, и для дешифрации достаточно подать на вход алгоритма блоки зашифрованных сообщений и ключи раундов в порядке, обратном их следованию при шифрации.

Особенностью алгоритма является отсутствие в стандарте каких-либо рекомендаций по выбору содержимого таблиц подстановок (S-блоков). Первоначально 8\*16\*4=512 бит таблиц подстановок являлись также частью ключевой информации. Впоследствии требование к секретности содержимого таблиц было упразднено, однако статическое, обеспечивающее высокую криптостойкость алгоритма, содержимое таблиц подстановки так и не было опубликовано. Набор S-блоков, рекомендуемый стандартом хеширования ГОСТ Р 34.11-94, использующего блочный шифр ГОСТ 28147-89 в качестве основной преобразующей операции, приведен в таблице 1.

Таблица 1. S-блоки алгоритма ГОСТ 28147-89



Достоинствами алгоритма ГОСТ можно назвать простоту реализации, причем как программной, так и аппаратной, поскольку алгоритм не использует битовых перестановок, большой размер ключа делает малоперспективной силовую атаку на шифр, большое количество раундов затрудняют дифференциальный и линейный криптоанализ. Алгоритм оптимизирован под 32-разрядные процессоры. Единственной проблемой практического применения алгоритма является, как уже упоминалось, формирование таблиц подстановки.

**Приложение**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include "C:\Libraries\cryptopp820\modes.h"

#include "C:\Libraries\cryptopp820\GOST.h"

#include "C:\Libraries\cryptopp820\filters.h"

using namespace std;

using namespace CryptoPP;

byte key[GOST::DEFAULT\_KEYLENGTH], iv[GOST::BLOCKSIZE];

string readFile(const std::string& fileName)

{

ifstream f(fileName, ios::binary);

stringstream ss;

ss << f.rdbuf();

return ss.str();

}

//функция Encrypt:

std::string Encrypt(std::string plaintext)

{

CryptoPP::GOST::Encryption gostEncryption(key, CryptoPP::GOST::DEFAULT\_KEYLENGTH);

CryptoPP::ECB\_Mode\_ExternalCipher::Encryption ebcEncryption(gostEncryption, iv);

// Local variable !

std::string ciphertext;

CryptoPP::StreamTransformationFilter stfEncryptor(ebcEncryption, new CryptoPP::StringSink(ciphertext));

stfEncryptor.Put(reinterpret\_cast<const unsigned char\*>(plaintext.c\_str()), plaintext.length() + 1);

stfEncryptor.MessageEnd();

return ciphertext;

}

//функция Decrypt:

std::string Decrypt(std::string plaintext)

{

CryptoPP::GOST::Decryption gostDecryption(key, CryptoPP::GOST::DEFAULT\_KEYLENGTH);

CryptoPP::ECB\_Mode\_ExternalCipher::Decryption ebcDecryption(gostDecryption, iv);

// Local variable !

std::string decryptedtext;

CryptoPP::StreamTransformationFilter stfDecryptor(ebcDecryption, new CryptoPP::StringSink(decryptedtext));

stfDecryptor.Put(reinterpret\_cast<const unsigned char\*>(plaintext.c\_str()), plaintext.size());

stfDecryptor.MessageEnd();

return decryptedtext;

}

int main(int argc, char\* argv[])

{

memset(key, 0x00, GOST::DEFAULT\_KEYLENGTH);

memset(iv, 0x00, GOST::BLOCKSIZE);

string name\_file = "text.txt";

string my\_file = readFile(name\_file);

string plaintext = my\_file;

string crypt = Encrypt(plaintext);

ofstream fout;

fout.open("crypt\_" + name\_file, ios::trunc | ios::binary);

fout << crypt;

fout.close();

string decrypt = Decrypt(crypt);

ofstream fout1;

fout1.open("decrypt\_" + name\_file, ios::trunc | ios::binary);

fout1 << decrypt;

fout1.close();

return 0;

}

**Результата работы программы**

