**БЛОЧНЫЕ АЛГОРИТМЫ СИММЕТРИЧНОГО ШИФРОВАНИЯ**

**Вариант №8**

отчет о лабораторной работе №5

по дисциплине

*МЕТОДЫ И СРЕДСТВА КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ*

Выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ст. гр. №230711, Павлова В.С.

Проверила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доцент каф. ИБ, Басалова Г.В.

**ХОД РАБОТЫ**

**Задание.** Разработать функции, реализующие базовые циклы зашифрования/расшифрования, аналогичные циклам 32-З и 32-Р шифра «Магма». Разработать программы шифрования/расшифрования одного полного блока данных с использованием созданных функций в соответствии с вариантом задания.

По заданию **размер блока** – 128; **размер ключа** – 512; **число основных шагов** – 32.

**Листинг 1** – Код программы для шифрования и расшифрования

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

/\* SIZES IN BYTE (8 BIT) \*/

constexpr int FULL\_KEY\_BYTE\_SIZE = 64;

constexpr int FULL\_BLOCK\_BYTE\_SIZE = 16;

constexpr int SUBKEY\_BYTE\_SIZE = FULL\_KEY\_BYTE\_SIZE / 8;

constexpr int BLOCK\_BYTE\_SIZE = FULL\_BLOCK\_BYTE\_SIZE / 2;

const string KEY\_PATH = "D:\\WORK\\NIX2TWIX\\CODE\\Magma\\key.txt";

const string DATA\_PATH = "D:\\WORK\\NIX2TWIX\\CODE\\Magma\\input.txt";

const string OUTPUT\_PATH = "D:\\WORK\\NIX2TWIX\\CODE\\Magma\\output.txt";

const vector<vector<int>> ReplacementTable = {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

};

// Разделение 64-битного блока пополам

pair<uint64\_t, uint64\_t> MakePairFromBlock(const vector <unsigned char>& data)

{

// BLOCK = {[64], [64]} = {[L], [R]}

uint64\_t L, R;

for (int i = 0; i < FULL\_BLOCK\_BYTE\_SIZE / 2; i++)

{

L = (L << 8) | data[i];

}

for (int i = FULL\_BLOCK\_BYTE\_SIZE / 2; i < FULL\_BLOCK\_BYTE\_SIZE; i++)

{

R = (R << 8) | data[i];

}

//cout << "L: " << hex << L << endl;

//cout << "R: " << hex << R << endl;

return { L, R };

}

// Объединение 64-битного блока в массив char

vector<unsigned char> CombineBlocksToData(uint64\_t L, uint64\_t R)

{

vector<unsigned char> data(FULL\_BLOCK\_BYTE\_SIZE);

for (int i = FULL\_BLOCK\_BYTE\_SIZE / 2 - 1; i >= 0; i--)

{

data[i] = static\_cast<unsigned char>(L & 0xFF); // Взять младший байт L

L >>= 8; // Сдвиг вправо на 8 бит

}

for (int i = FULL\_BLOCK\_BYTE\_SIZE - 1; i >= FULL\_BLOCK\_BYTE\_SIZE / 2; i--)

{

data[i] = static\_cast<unsigned char>(R & 0xFF); // Взять младший байт R

R >>= 8; // Сдвиг вправо на 8 бит

}

return data;

}

vector<uint64\_t> MakeSubkeysArray(const vector <unsigned char>& key)

{

// KEY = SUBKEYS[8]

vector<uint64\_t> SUBKEYS(8, 0);

// Разделение ключа на массив из 8 подключей по 512/8 = 64 бит

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

{

SUBKEYS[i] = 0;

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

{

SUBKEYS[i] = (SUBKEYS[i] << 8) | key[i \* 8 + j];

}

}

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

//{

// cout << "Subkey " << i << ": " << hex << SUBKEYS[i] << endl;

//}

return SUBKEYS;

}

void EncryptionRound(uint64\_t& subKey, uint64\_t& L, uint64\_t& R)

{

// Сохранение начального значения младшей части блока

uint64\_t Rtemp = R;

R = R + subKey;

// Разделение на S-подблоки по 4 бита

vector<char> subBlocks(FULL\_BLOCK\_BYTE\_SIZE, 0);

for (int i = 0; i < FULL\_BLOCK\_BYTE\_SIZE; i++)

{

subBlocks[i] = (R >> (4 \* (15 - i))) & 0x0F;

// Замена по таблице

subBlocks[i] = ReplacementTable[i][subBlocks[i]];

}

// Запись результата замены в правую часть блока

R = 0;

for (int i = 0; i < FULL\_BLOCK\_BYTE\_SIZE; i++)

{

R = R | (static\_cast<uint64\_t>(subBlocks[i]) << (4 \* (15 - i)));

}

// Циклический сдвиг на 11 бит влево

R = (R << 11) | (R >> 53);

// Сложение со старшей частью блока

R ^= L;

L = Rtemp;

}

void DecryptionRound(uint64\_t& subKey, uint64\_t& L, uint64\_t& R)

{

uint64\_t temp = R;

R = L;

L = L + subKey;

// Разделение на S-подблоки по 4 бита

vector<char> subBlocks(FULL\_BLOCK\_BYTE\_SIZE, 0);

for (int i = 0; i < FULL\_BLOCK\_BYTE\_SIZE; i++)

{

subBlocks[i] = (L >> (4 \* (15 - i))) & 0x0F;

// Замена по таблице

subBlocks[i] = ReplacementTable[i][subBlocks[i]];

}

// Запись результата замены в правую часть блока

L = 0;

for (int i = 0; i < FULL\_BLOCK\_BYTE\_SIZE; i++)

{

L = L | (static\_cast<uint64\_t>(subBlocks[i]) << (4 \* (15 - i)));

}

// Циклический сдвиг на 11 бит влево

L = (L << 11) | (L >> 53);

L = L ^ temp;

}

void EncryptionBlock(const vector <unsigned char>& key, const vector <unsigned char>& data)

{

ofstream encryptedFile(OUTPUT\_PATH, ios::binary);

// Разделение блока пополам

pair<uint64\_t, uint64\_t> DATA;

DATA = MakePairFromBlock(data);

uint64\_t L, R;

L = DATA.first;

R = DATA.second;

// Разделение ключа на массив подключей

vector<uint64\_t> SUBKEYS;

SUBKEYS = MakeSubkeysArray(key);

// Первые 24 раунда шифрования K0-7

for (int j = 0; j < 3; j++)

{

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

{

// Определение подключа раунда

uint64\_t subKey = SUBKEYS[i];

EncryptionRound(subKey, L, R);

}

}

// Последние 8 раундов шифрования K7-0

for (int i = 7; i >= 0; i--)

{

// Определение подключа раунда

uint64\_t subKey = SUBKEYS[i];

EncryptionRound(subKey, L, R);

}

DATA = { L,R };

vector <unsigned char> newData = CombineBlocksToData(L, R);

encryptedFile.write(reinterpret\_cast<const char\*>(newData.data()), newData.size());

}

void DecryptionBlock(const vector <unsigned char>& key, const vector <unsigned char>& data)

{

ofstream decryptedFile(OUTPUT\_PATH, ios::binary);

// Разделение блока пополам

pair<uint64\_t, uint64\_t> DATA;

DATA = MakePairFromBlock(data);

uint64\_t L, R;

L = DATA.first;

R = DATA.second;

// Разделение ключа на массив подключей

vector<uint64\_t> SUBKEYS;

SUBKEYS = MakeSubkeysArray(key);

// Первые 8 раундов расшифрования K0-7

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

{

// Определение подключа раунда

uint64\_t subKey = SUBKEYS[i];

DecryptionRound(subKey, L, R);

}

// Последние 24 раунда шифрования K7-0

for (int j = 0; j < 3; j++)

{

for (int i = 7; i >= 0; i--)

{

// Определение подключа раунда

uint64\_t subKey = SUBKEYS[i];

DecryptionRound(subKey, L, R);

}

}

DATA = { L,R };

vector <unsigned char> newData = CombineBlocksToData(L, R);

decryptedFile.write(reinterpret\_cast<const char\*>(newData.data()), newData.size());

}

int main()

{

// Считывание ключа

ifstream keyFile(KEY\_PATH, ios::binary);

vector <unsigned char> key;

char buff;

for (int i = 0; i < FULL\_KEY\_BYTE\_SIZE; i++)

{

keyFile.read(&buff, sizeof(unsigned char));

key.push\_back(buff);

}

/\*

Для шифрования прописать путь: DATA\_PATH

Для расшифрования результата шифрования

прописать путь для dataFile: OUTPUT\_PATH

\*/

ifstream dataFile(DATA\_PATH, ios::binary);

vector <unsigned char> data;

// Считывание блока данных

for (int i = 0; i < FULL\_BLOCK\_BYTE\_SIZE; i++)

{

dataFile.read(&buff, sizeof(unsigned char));

data.push\_back(buff);

}

EncryptionBlock(key, data);

// DecryptionBlock(key, data);

return EXIT\_SUCCESS;

}