**БЛОЧНЫЕ АЛГОРИТМЫ СИММЕТРИЧНОГО ШИФРОВАНИЯ. РЕЖИМЫ РАБОТЫ**

**Вариант №8**

отчет о лабораторной работе №6

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 две программы для двух режимов шифрования/расшифрования файлов произвольного размера и типа с использованием функции, реализующей базовый цикл блочного шифра, разработанной в ЛР№5. Используемые режимы: **режим простой замены с зацеплением** **и режим гаммирования с обратной связью по выходу**. Размер блока – 128; размер ключа – 512; число основных шагов – 32.

**Листинг 1** – Код программы для шифрования и расшифрования в режиме простой замены с зацеплением

#include "MagmaRounds.h"

int main\_CBC\_ENCRYPT()

{

// 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);

}

// Определение размера файла с данными в байтах

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

dataFile.seekg(0, ios::end);

int fileSize = dataFile.tellg();

dataFile.seekg(0, ios::beg);

**Листинг 1** – Код программы для шифрования и расшифрования в режиме простой замены с зацеплением (продолжение)

dataFile.close();

// Дополнение последнего блока данных при необходимости

int padCount = Padding(fileSize, DATA\_PATH);

// Считывание данных

dataFile.open(DATA\_PATH, ios::binary);

vector <unsigned char> data;

for (int i = 0; i < fileSize + padCount; i++)

{

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

data.push\_back(buff);

}

dataFile.close();

vector<unsigned char> IV = GenerateSinchrosign();

vector<uint64\_t> shiftRegister(M / 8);

// Заполнение регистра сдвига значением синхропосылки

shiftRegister = MakeRegister(IV);

// Буферы для хранения промежуточных данных

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

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

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

uint64\_t L, R;

// Шифрование файла, разделённого на блоки

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

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

{

block.clear();

block.resize(FULL\_BLOCK\_BYTE\_SIZE);

copy(

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i),

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i + 1),

block.begin()

);

// XOR

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

{

xorBlock[q] = block[q] ^ IV[q];

}

// Шифрование

encryptedBlock = EncryptionBlock(key, xorBlock);

// Разделение блока пополам

pair<uint64\_t, uint64\_t> ENCRYPTED\_DATA;

ENCRYPTED\_DATA = MakePairFromBlock(encryptedBlock);

L = ENCRYPTED\_DATA.first;

R = ENCRYPTED\_DATA.second;

**Листинг 1** – Код программы для шифрования и расшифрования в режиме простой замены с зацеплением (продолжение)

// Сдвиг регистра в сторону старших битов

uint64\_t temp\_1 = shiftRegister[2];

uint64\_t temp\_2 = shiftRegister[3];

shiftRegister[0] = temp\_1;

shiftRegister[1] = temp\_2;

shiftRegister[2] = L;

shiftRegister[3] = R;

IV = SplitRegister(shiftRegister);

// reinterpret\_cast применяется для приведения разных по типу указателей

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

}

return EXIT\_SUCCESS;

}

int main\_CBC\_DECRYPT()

{

// 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);

}

// Определение размера файла в байтах

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

dataFile.seekg(0, ios::end);

int fileSize = dataFile.tellg();

dataFile.seekg(0, ios::beg);

// Считывание всех данных

vector <unsigned char> data;

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

{

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

data.push\_back(buff);

}

dataFile.close();

// Считывание использованной при шифровании синхропосылки

ifstream sinchrosignFile(SINCHROSIGN\_PATH, ios::binary);

vector<unsigned char> IV;

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

{

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

IV.push\_back(buff);

}

sinchrosignFile.close();

// Заполнение регистра сдвига значением синхропосылки

**Листинг 1** – Код программы для шифрования и расшифрования в режиме простой замены с зацеплением (продолжение)

vector<uint64\_t> shiftRegister(4);

shiftRegister = MakeRegister(IV);

// Буферы для хранения промежуточных данных

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

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

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

uint64\_t L, R;

// Расшифрование файла, разделённого на блоки

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

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

{

block.clear();

block.resize(FULL\_BLOCK\_BYTE\_SIZE);

copy(

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i),

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i + 1),

block.begin()

);

// Разделение блока пополам

pair<uint64\_t, uint64\_t> ENCRYPTED\_DATA;

ENCRYPTED\_DATA = MakePairFromBlock(block);

L = ENCRYPTED\_DATA.first;

R = ENCRYPTED\_DATA.second;

decryptedBlock = DecryptionBlock(key, block);

// XOR

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

{

xorBlock[q] = decryptedBlock[q] ^ IV[q];

}

// Сдвиг регистра в сторону старших битов

uint64\_t temp\_1 = shiftRegister[2];

uint64\_t temp\_2 = shiftRegister[3];

shiftRegister[0] = temp\_1;

shiftRegister[1] = temp\_2;

shiftRegister[2] = L;

shiftRegister[3] = R;

IV = SplitRegister(shiftRegister);

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

}

decryptedFile.close();

// Усечение последнего блока, если есть необходимость

RemovePadding(DECRYPT\_PATH);

return EXIT\_SUCCESS;

}

**Листинг 2** – Код программы для шифрования и расшифрования в режиме гаммирования с обратной связью по выходу

#include "MagmaRounds.h"

int main\_OFB\_ENCRYPT()

{

// 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);

}

// Определение размера файла с данными в байтах

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

dataFile.seekg(0, ios::end);

int fileSize = dataFile.tellg();

dataFile.seekg(0, ios::beg);

vector <unsigned char> data;

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

{

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

data.push\_back(buff);

}

dataFile.close();

int S = fileSize % FULL\_BLOCK\_BYTE\_SIZE;

// Генерация синхропосылки

vector<unsigned char> IV = GenerateSinchrosign();

// Заполнение регистра сдвига значением синхропосылки

vector<uint64\_t> shiftRegister(M / 8);

shiftRegister = MakeRegister(IV);

// Буферы для хранения промежуточных данных

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

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

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

uint64\_t L, R;

// Шифрование файла, разделённого на целые блоки

int blockCount =

(fileSize % FULL\_BLOCK\_BYTE\_SIZE == 0)

? fileSize / FULL\_BLOCK\_BYTE\_SIZE

: fileSize / FULL\_BLOCK\_BYTE\_SIZE + 1;

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

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

**Листинг 2** – Код программы для шифрования и расшифрования в режиме гаммирования с обратной связью по выходу (продолжение)

{

// Шифрование гаммы

vector <unsigned char> gamma = EncryptionBlock(key, IV);

// Разделение блока пополам

pair<uint64\_t, uint64\_t> ENCRYPTED\_GAMMA;

ENCRYPTED\_GAMMA = MakePairFromBlock(gamma);

L = ENCRYPTED\_GAMMA.first;

R = ENCRYPTED\_GAMMA.second;

// Сдвиг регистра в сторону старших битов

uint64\_t temp\_1 = shiftRegister[2];

uint64\_t temp\_2 = shiftRegister[3];

shiftRegister[0] = temp\_1;

shiftRegister[1] = temp\_2;

shiftRegister[2] = L;

shiftRegister[3] = R;

IV.clear();

IV.resize(M);

IV = SplitRegister(shiftRegister);

block.clear();

if (FULL\_BLOCK\_BYTE\_SIZE \* (i + 1) <= data.size())

{

block.resize(FULL\_BLOCK\_BYTE\_SIZE);

copy(

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i),

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i + 1),

block.begin()

);

}

else

{

block.resize(S);

copy(

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i),

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i) + (S),

block.begin()

);

}

// XOR

for (int q = 0; q < block.size(); q++)

{

xorBlock[q] = block[q] ^ gamma[q];

}

// reinterpret\_cast применяется для приведения разных по типу указателей

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

}

encryptedFile.close();

return EXIT\_SUCCESS;

}

int main\_OFB\_DECRYPT()

{

// Считывание ключа

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

**Листинг 2** – Код программы для шифрования и расшифрования в режиме гаммирования с обратной связью по выходу (продолжение)

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);

}

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

vector <unsigned char> data;

// Определение размера файла в байтах

dataFile.seekg(0, ios::end);

int fileSize = dataFile.tellg();

dataFile.seekg(0, ios::beg);

// Считывание всех данных

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

{

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

data.push\_back(buff);

}

dataFile.close();

int S = fileSize % FULL\_BLOCK\_BYTE\_SIZE;

// Считывание использованной при шифровании синхропосылки

ifstream sinchrosignFile(SINCHROSIGN\_PATH, ios::binary);

vector<unsigned char> IV;

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

{

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

IV.push\_back(buff);

}

sinchrosignFile.close();

// Заполнение регистра сдвига значением синхропосылки

vector<uint64\_t> shiftRegister(4);

shiftRegister = MakeRegister(IV);

// Буферы для хранения промежуточных данных

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

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

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

uint64\_t L, R;

// Расшифрование файла, разделённого на блоки

int blockCount =

(fileSize % FULL\_BLOCK\_BYTE\_SIZE == 0)

? fileSize / FULL\_BLOCK\_BYTE\_SIZE

: fileSize / FULL\_BLOCK\_BYTE\_SIZE + 1;

**Листинг 2** – Код программы для шифрования и расшифрования в режиме гаммирования с обратной связью по выходу (продолжение)

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

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

{

// Шифрование гаммы

vector <unsigned char> gamma = EncryptionBlock(key, IV);

// Разделение блока пополам

pair<uint64\_t, uint64\_t> ENCRYPTED\_GAMMA;

ENCRYPTED\_GAMMA = MakePairFromBlock(gamma);

L = ENCRYPTED\_GAMMA.first;

R = ENCRYPTED\_GAMMA.second;

// Сдвиг регистра в сторону старших битов

uint64\_t temp\_1 = shiftRegister[2];

uint64\_t temp\_2 = shiftRegister[3];

shiftRegister[0] = temp\_1;

shiftRegister[1] = temp\_2;

shiftRegister[2] = L;

shiftRegister[3] = R;

IV.clear();

IV.resize(M);

IV = SplitRegister(shiftRegister);

block.clear();

if (FULL\_BLOCK\_BYTE\_SIZE \* (i + 1) <= data.size())

{

block.resize(FULL\_BLOCK\_BYTE\_SIZE);

copy(

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i),

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i + 1),

block.begin()

);

}

else

{

block.resize(S);

copy(

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i),

data.begin() + FULL\_BLOCK\_BYTE\_SIZE \* (i)+(S),

block.begin()

);

}

// XOR

for (int q = 0; q < block.size(); q++)

{

xorBlock[q] = block[q] ^ gamma[q];

}

// reinterpret\_cast применяется для приведения разных по типу указателей

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

}

decryptedFile.close();

return EXIT\_SUCCESS;

}