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;
}
```