ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА № 1.2

ПО ДИСЦИПЛИНЕ «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

Основы шифрования данных

Выполнил: Давыдов Иван Денисович

Группа: P3400

Вариант 5

Санкт-Петербург
2020/2021

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 заданного блочного шифра в заданном режиме шифрования. Перечень блочных шифров и режимов шифрования приведен в таблице. Номер шифра и режима для реализации получить у преподавателя.

Алгоритм		Режим шифрования	
Номер		Номер	
5	Rijndael	д	OFB

Листинг

Rijndael.java

```
package lab2;

import java.security.MessageDigest;

public class Rijndael {
    private int blockLength = 0;
    private int keyLength = 0;
    private int Nb = 0, Nk = 0, Nr = 0;
    private String password;
    private boolean OFB = false;

    //look-up таблица для операции subBytes
    private int[] Sbox = {
                0x63, ... , 0x16
    };

    //look-up таблица для операции invSubBytes
    private int[] InvSbox = {
        0x52, ... ,0x7d
    };

    //применяется при генерации раундовых ключей
    private int [][] Rcon = {
        {0x00, 0x00, 0x00, 0x00},
        ...
        {0x36, 0x00, 0x00, 0x00}
    };



    Rijndael(int blockLength,int keyLength, String password, boolean ofb){
        this.blockLength = blockLength;
        this.keyLength = keyLength;
        this.password = password;
        this.OFB = ofb;

        this.Nb = blockLength/4;
        this.Nk = keyLength/4;
```

```
        //выбираем количество раундов
        switch (Nb)
        {
            case 4:
                Nr = Nb + Nk +2;
                break;
            case 6:
                Nr = Nk == 8? 14 : 12;
                break;
            case 8:
                Nr = 14;
                break;
        }
    }

     //Метод перестановки байт состояния по look-up таблице
    private int[][] subBytes(int[][] state){
        int[][] res = new int[4][Nb];
        for (int i =0; i<4; i++)
            for (int j = 0; j<Nb; j++)
            res[i][j] = Sbox[state[i][j]];
        return res;
    }

    // обратная операция subBytes
    private int[][] invSubBytes(int[][] state){
        int[][] res = new int[4][Nb];
        for (int i =0; i<4; i++)
            for (int j = 0; j<Nb; j++)
                res[i][j] = InvSbox[state[i][j]];
        return res;
    }

    // сдвиг строк состояния по следующему правилу:
//        {1,2,3,4},        {1,2,3,4},
//        {1,2,3,4},  -->   {2,3,4,1},
//        {1,2,3,4},        {3,4,1,2},
//        {1,2,3,4},        {4,1,3,2},

    private int[][] shiftRows(int[][] state){
        int[][] res = new int[4][Nb];
        for(int i =0; i<Nb; i++) { //столбцы
            res[0][i] = state[0][i];
            res[1][i] = state[1][(i + 1) % Nb];
            res[2][i] = state[2][(i + 2) % Nb];
            res[3][i] = state[3][(i + 3) % Nb];
        }
        return res;
    }

    // сдвиг строк состояния по следующему правилу:
//     {1,2,3,4},       {1,2,3,4},
//     {2,3,4,1},  --> {1,2,3,4},
//     {3,4,1,2},       {1,2,3,4},
//     {4,1,3,2},       {1,2,3,4},
    private int[][] invShiftRows(int[][]state){
        int[][] res = new int[4][Nb];
        for(int i = Nb-1; i > -1; i--) { //столбцы
            res[0][i] = state[0][i];
            res[1][i] = state[1][(i+(Nb-1))%Nb];//(1+(Nb - i)) % Nb];
            res[2][i] = state[2][(i+(Nb-2))%Nb];
            res[3][i] = state[3][(i+(Nb-3))%Nb];
        }
        return res;
    }
```

```
    // Каждая колонка состояния трактуется как полином третьей степени.Над этими полиномами
производится умножение в поле Галуа
    // по модулю x^{4}+1 на фиксированный многочлен c(x)=3x^{3}+x^{2}+x+2
    public int[][] mixColumns(int[][] state){
        int[][] res = new int[4][Nb];

        for(int i =0; i<Nb; i++) { //столбцы
            res[0][i] = (Galois.multiply((byte)2, (byte)state[0][i]) ^
Galois.multiply((byte)3,(byte)state[1][i]) ^ (byte)state[2][i] ^ (byte)state[3][i]) & 0xFF;
            res[1][i] = ((byte)state[0][i] ^ Galois.multiply((byte)2, (byte)state[1][i]) ^
Galois.multiply((byte)3, (byte)state[2][i]) ^ (byte)state[3][i]) & 0xFF;
            res[2][i] = ((byte)state[0][i] ^ (byte)state[1][i] ^
Galois.multiply((byte)2,(byte)state[2][i]) ^ Galois.multiply((byte)3,(byte)state[3][i])) & 0xFF;
            res[3][i] = (Galois.multiply((byte)3,(byte)state[0][i]) ^ (byte)state[1][i] ^
(byte)state[2][i] ^ Galois.multiply((byte)2,(byte)state[3][i])) & 0xFF;
        }
        return res;
    }

    // обратная операция mixColumns
    public int[][] invMixColumns(int[][] state){
        int[][] res = new int[4][Nb];

        for(int i =0; i<Nb; i++) { //столбцы
            res[0][i] = (Galois.multiply((byte)0x0e, (byte)state[0][i]) ^
Galois.multiply((byte)0x0b,(byte)state[1][i]) ^
                            Galois.multiply((byte)0x0d, (byte)state[2][i]) ^
Galois.multiply((byte)0x09,(byte)state[3][i])) & 0xFF;
            res[1][i] = (Galois.multiply((byte)0x09, (byte)state[0][i]) ^
Galois.multiply((byte)0x0e,(byte)state[1][i]) ^
                            Galois.multiply((byte)0x0b, (byte)state[2][i]) ^
Galois.multiply((byte)0x0d,(byte)state[3][i])) & 0xFF;
            res[2][i] = (Galois.multiply((byte)0x0d, (byte)state[0][i]) ^
Galois.multiply((byte)0x09,(byte)state[1][i]) ^
                            Galois.multiply((byte)0x0e, (byte)state[2][i]) ^
Galois.multiply((byte)0x0b,(byte)state[3][i])) & 0xFF;
            res[3][i] = (Galois.multiply((byte)0x0b, (byte)state[0][i]) ^
Galois.multiply((byte)0x0d,(byte)state[1][i]) ^
                            Galois.multiply((byte)0x09, (byte)state[2][i]) ^
Galois.multiply((byte)0x0e,(byte)state[3][i])) & 0xFF;
        }
        return res;
    }

    //скалдываем каждый элемент состояния с соответствующим ему элементом раундового ключа
    private int[][] addRoundKey(int[][] state, int[][] roundKey, int round){
        int[][] res = new int[4][Nb];
        for(int i =0; i<Nb; i++) //столбцы
            for(int j=0; j<4;j++)
                res[j][i] = state[j][i] ^ roundKey[j][round*4+i];
        return res;
    }

    //генерируем cipher key используя MD5hash
    private int[][] generateKey(String word){
        byte[] passwordBytes = word.getBytes();
        int[][] key = new int[4][Nk];

        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("MD5");
        }catch (Exception e){}
        byte[] mdPass = md.digest(passwordBytes);

        for (int i = 0; i < 4; i++)
            for (int j = 0; j < Nk; j++)
                key[i][j] = mdPass[(i+4*j)%16] & 0xFF;

        return key;
```

```
    }

    //Метод генерации раундовых ключей
    private int[][] keyExpansion(int[][] key){
        //содержи в себе ключ шифрования[0] и Nr раунд ключей[1..Nr]
        int[][] schedule = new int[4][(Nr+1)*Nk];

        //копируем ключ шифрования
        for(int i = 0; i<4; i++)
            for(int j = 0; j<Nk; j++)
                schedule[i][j]  = key[i][j];

        int curRound = 1; //[1, Nr]
        while (true) {
            if(curRound > Nr)
                break;

            // i1 = rotate(i Col)
            // i2 = subWord(i1)
            // newColumn  = (i - 4) ^ i2 ^ rcon(i)
            for (int i = 0; i < 4; i++) {

                int[] rotatedWord = rotWord(new int[]{schedule[0][curRound*4 + i - 1],
schedule[1][curRound*4 + i - 1], schedule[2][curRound*4 + i - 1], schedule[3][curRound*4 + i - 1]});
                int[] subbed = subWord(rotatedWord);

                int[] jMinusFour = new int[]{schedule[0][curRound*4 + i - 4], schedule[1][curRound*4 +
i - 4], schedule[2][curRound*4 + i - 4],schedule[3][curRound*4 + i - 4]};

                //bitwise xor and write to schedule
                int[] res = new int[4];
                for (int j = 0; j < 4; j++)
                    schedule[j][curRound*4+i] = jMinusFour[j] ^ subbed[j] ^ Rcon[curRound][j];
            }
            curRound++;
        }
        return schedule;
    }

    //циклический сдвиг колонок, применяется только в расписаниии
    private int[] rotWord(int[] word){
        int[] res = new int[word.length];
        for (int i = 0; i <3; i++){
            res[i] = word[i+1];
        }
        res[word.length-1] = word[0];
        return res;
    }

    //Операция Sub для одного байта
    private int[] subWord(int[] word){
        int[] res = new int[4];
        for (int i =0; i<4; i++)
            res[i] = Sbox[word[i]];
        return res;
    }

    //Bitwise xor для элементов в одинаковых ячейках матриц
    private int[][] xorArrays(int[][]a,int[][]b)
    {
        int[][] res = new int[4][Nb];
        for(int i = 0; i<4; i++)
            for (int j = 0;j<4;j++)
                res[i][j] = a[i][j] ^ b[i][j];
        return res;
    }
```

```
    //метод основной части шифрования
    private int[][] encryptBlock(int[][] stata, int[][]schedule) {
        int[][] state = new int[4][Nb];
        // начальный раунд
        state = addRoundKey(stata, schedule, 0);

        // основные раунды
        for(int i = 1; i <= Nr - 1 ; i++){
            state = subBytes(state);
            state = shiftRows(state);
            state = mixColumns(state);
            state = addRoundKey(state, schedule, i);
        }

        //финальный раунд
        state = subBytes(state);
        state = shiftRows(state);
        state = addRoundKey(state, schedule, Nr);

        return state;
    }

    public String encrypt(String text){
        int[][] state = new int[4][Nb];
        int[][] ofb_state = new int[4][Nb];
        int[][] text_block = new int[4][Nb];
        String paddedText = "";
        String result = " ";

        if(this.OFB)
            ofb_state = generateKey("OFB");

        paddedText = makePadding(text);
        result = String.format("%"+paddedText.length()+"s",result);
        StringBuilder stringBuilder = new StringBuilder(result);

        int[][] key = generateKey(this.password);
        int[][] schedule = keyExpansion(key);

        int k = 0;
        while(true){ // блоки
            if(k == paddedText.length() / blockLength)
                break;

            // копируем блок текста в матрицу
            for(int i = 0; i < 4; i++)
                for (int j = 0; j < Nb; j++)
                    text_block[i][j] = paddedText.charAt(k*blockLength+Nb*i+j);

            if(this.OFB)
                state = ofb_state;
            else
                state = text_block;

            state = encryptBlock(state, schedule);
            ofb_state = state;

            int[][] tmp_state = state;
            if(this.OFB) {
                tmp_state = xorArrays(state, text_block);
            }

            for(int i = 0; i < 4; i++)
                for (int j = 0; j < Nb; j++)
                    stringBuilder.setCharAt(k*blockLength+Nb*i+j,(char)tmp_state[i][j]);

            k++;
        }
        return stringBuilder.toString();
    }
```

```
public String decrypt(String text){
    int[][] state = new int[4][Nb];

    int[][] ofb_state = new int[4][Nb];
    int[][] text_block = new int[4][Nb];

    if(this.OFB)
        ofb_state = generateKey("OFB");

    //генерим основной и раундовые ключи
    int[][] key = generateKey(this.password);
    int[][] schedule = keyExpansion(key);

    String result = " ";
    result = String.format("%"+text.length()+"s",result);
    StringBuilder stringBuilder = new StringBuilder(result);

    int k = 0;
    while(true){ // блоки
        if(k == text.length() / blockLength)
            break;

        //копируем блок текста в матрицу
        for(int i = 0; i < 4; i++)
            for (int j = 0; j < Nb; j++)
                text_block[i][j] = text.charAt(k*blockLength+Nb *i+j);

        if(this.OFB) {
            //в случае ОФБ нам необходимо заново зашифровать IV и xor его с криптограммой
            state = ofb_state;
            state = encryptBlock(state,schedule);
        }
        else {
            state = text_block;

            // начальный раунд
            state = addRoundKey(state, schedule, Nr);


            // основные раунды
            for (int i = Nr - 1; i >= 1; i--) {
                state = invShiftRows(state);
                state = invSubBytes(state);
                state = addRoundKey(state, schedule, i);
                state = invMixColumns(state);
            }

            //финальный раунд
            state = invShiftRows(state);
            state = invSubBytes(state);
            state = addRoundKey(state, schedule, 0);
        }

        int[][] tmp_state = state;
        if(this.OFB) {
            ofb_state = state;
            tmp_state = xorArrays(state, text_block);
        }

        for(int i = 0; i < 4; i++)
            for (int j = 0; j < Nb; j++)
                stringBuilder.setCharAt(k*blockLength+Nb*i+j,(char)tmp_state[i][j]);
        k++;
    }

    return stringBuilder.toString();
}
```

```
    // добавляем padding
    public String makePadding(String text){
        String result = text;
        int charsToAdd = 0;
        int amountOfBlocks = (int)result.length() / blockLength + ( result.length()%blockLength == 0 ?
0:1);
        System.out.println(amountOfBlocks);

        charsToAdd = blockLength*amountOfBlocks - result.length();


        result += Util.getRandomString(charsToAdd);
        return result;
    }

}
```

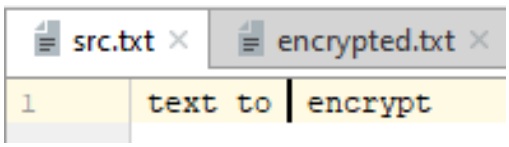
Main.java

```
package lab2;

import org.apache.commons.cli.*;
import org.apache.commons.codec.*;
import org.apache.commons.codec.binary.Hex;

import java.io.IOException;
import java.nio.file.StandardOpenOption;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Main {

    public static void main(String[] args){
        Options options = new Options();

        Option input = new Option("s", "source", true, "input text file path");
        input.setRequired(true);
        options.addOption(input);

        Option encrypted = new Option("e", "encrypted", true, "file path to encrypted text");
        encrypted.setRequired(true);
        options.addOption(encrypted);

        Option decrypted = new Option("d", "decrypted", true, "file path to decrypted text");
        decrypted.setRequired(true);
        options.addOption(decrypted);

        Option blockLength = new Option("bl", "blocklength", true, "size of block, Bite");
        blockLength.setRequired(true);
        options.addOption(blockLength);

        Option keyLength = new Option("kl", "keylength", true, "size of key, Bite");
        keyLength.setRequired(true);
        options.addOption(keyLength);

        Option password = new Option("p", "password", true, "used in key generator");
        password.setRequired(true);
        options.addOption(password);

        Option ofb = new Option("ofb", "output-feedback", false, "sets encryption mode to ofb");
        ofb.setRequired(false);
        options.addOption(ofb);
```

```
        CommandLineParser parser = new DefaultParser();

        try {
            CommandLine cmd = parser.parse(options, args);

            String inputFile = cmd.getOptionValue("source");
            String encryptedFile = cmd.getOptionValue("encrypted");
            String decryptedFile = cmd.getOptionValue("decrypted");
            String pass = cmd.getOptionValue("password");

            int blocksize = Integer.parseInt(cmd.getOptionValue("blocklength"));
            int keysize = Integer.parseInt(cmd.getOptionValue("keylength"));

            if((blocksize != 16 && blocksize != 24 && blocksize != 32) ||
                    keysize != 16 && keysize != 24 && keysize != 32)
            {
                System.out.println("Invalid length of key or block");
                System.exit(1);
            }

//              Files.write(Paths.get("text.txt"), Util.getRandomString(500000).getBytes("ascii"),
StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);

//              шифруем
            Rijndael rijndael = new Rijndael(blocksize, keysize, pass, cmd.hasOption("output-
feedback"));

            String contents = new String(Files.readAllBytes(Paths.get(inputFile)));

            long time = System.currentTimeMillis();
            String encryptedText = rijndael.encrypt(contents);
            System.out.println("Encryption time:"+ (System.currentTimeMillis() - time));

            Files.write(Paths.get(encryptedFile), encryptedText.getBytes("ascii"),
StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);

            //дешифруем
            time = System.currentTimeMillis();
            String decryptedText = rijndael.decrypt(encryptedText);
            System.out.println("Decryption time:"+ (System.currentTimeMillis() - time));
            Files.write(Paths.get(decryptedFile), decryptedText.getBytes("utf-8"),
StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);

        } catch (ParseException | IOException e) {
            System.out.println(e.getMessage());

            System.exit(1);
        }

    }
}
```

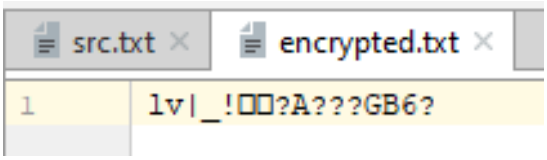
Результаты

Исходный текст:

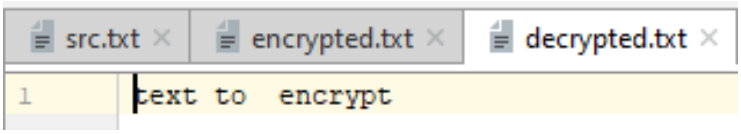


Режим ECB

Текст, полученный в результате шифрования:

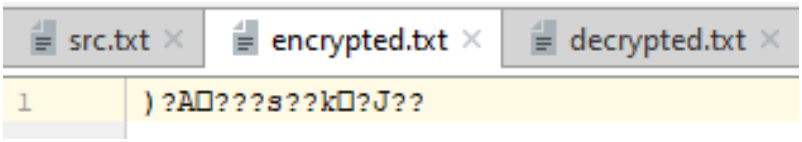


Текст, полученный в результате дешифрования:

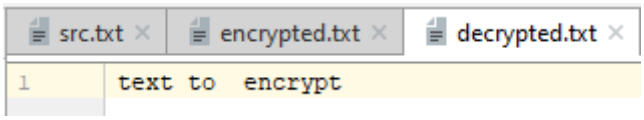


Режим OFB

Текст, полученный в результате шифрования:



Текст, полученный в результате дешифрования:



Выводы

В ходе данной лабораторной работы был изучен алгоритм шифрования Rijndael. Была написана программа, реализующая этот алгоритм в двух режимах работы: ECB & OFB. Работоспособность и корректность результатов была проверена на файлах различной длины.