Университет ИТМО

Факультет программной инженерии и компьютерной техники

Информационная безопасность

Отчет по лабораторной работе № 2

«Блочное симметричное шифрование»

Вариант 5Б

Выполнил студент группы P34302

Ким Даниил Кванхенович

Проверил преподаватель

Рыбаков Степан Дмитриевич

Санкт-Петербург 2023

Содержание

Цель работы:

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

Постановка задачи:

Вариант – 5Б:

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	**Название**	**Номер**	**Режим**
5	Rijndael	Б	CBC

Порядок выполнения работы:

1. Ознакомьтесь с теоретическими основами шифрования данных.
2. Получите вариант задания у преподавателя.
3. Напишите программу согласно варианту задания.
4. Отладьте разработанную программу и покажите результаты работы программы преподавателю.
5. Составьте отчет по лабораторной работе.

Выполнение:

Описание алгоритма:

Rijndael – симметричный алгоритм блочного шифрования (не основанный на сети Фейштеля)

- Для шифрования и расшифрования применяется один ключ, который используется для генерации ключей раунда.
- Данные шифруются и расшифровываются блоками STATE (для *AES* длина блока равна 128 бит)

В процесс шифрования используются 4 основные функции:

- ***SubBytes*** – преобразовывает байты блока с помощью *S-Box*`ов (известных нелинейных таблиц замещения). Обеспечивает преобразование на уровне байтов. Обратная ей функция для дешифрования – ***InvSubBytes***.
- ***ShiftRows*** – применяет циклические смещения строк *STATE* на разные величины. Обеспечивает перемешивание на уровне строк. Обратная функция для дешифрования – ***InvShiftRows***.
- ***MixColumns*** – преобразовывает все столбцы *STATE* и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы. Обеспечивает преобразование и перемешивание на уровне столбцов и преобразовывает байты учитывая соседние байты. Обратная функция для дешифрования – ***InvMixColumns***.
- ***AddRoundKey*** – складывает *STATE* и ключ раунда. Единственное преобразование, которое включает в себя ключ. Обратная функция для дешифрования – сама *AddRoundKey* (за счет обратимости XOR).

Предварительный раунд 0 состоит из одного шага – *AddRoundKey*. Затем, в соответствие с количеством раундов, последовательно выполняются 4 шага:

1. *SubBytes*
2. *ShiftRows*
3. *MixColumns* (во время последнего раунда пропускается)
4. *AddRoundKey*

Листинг разработанного модуля:

Основные методы модуля Rijndael:

- Функция преобразования байтов *SubBytes*:

```
private static void SubBytes(int[][] state) {
    for (int i = 0; i < state.length; i++) {
        for (int j = 0; j < state[0].length; j++) {
            state[i][j] = S_BOX[state[i][j]];
        }
    }
}


private static void InvSubBytes(int[][] state) {
    for (int i = 0; i < state.length; i++) {
        for (int j = 0; j < state[0].length; j++) {
            state[i][j] = INV_S_BOX[state[i][j]];
        }
    }
}
```

*где *Nb* - число столбцов в STATE (для AES *Nb* = 4, т.е. длина блока равна 4 * 32 бит = 128 бит).

Представляет из себя блок замены (substitution) для уничтожения статистической взаимосвязи между зашифрованным текстом и ключом с целью противостояния по-пыткам определить ключ.

Может быть реализована с помощью преобразования в поле Галуа или с помощью предварительно вычисленной таблицы замен, содержащей новое значение для каждого из 256 возможным значений 2 байтового символа.

Сами S-Box`ы представляют из себя массив длиной 256 включающий значения, соответствующих своему индексу:

```
private static final int[] S_BOX;
…

S_BOX = new int[]{
        0x63, 0x7c, 0x77 … 0x54, 0xbb, 0x16
};
```

- Функция сдвига строк ShiftRows

```
private static void ShiftRows(int[][] state) {
    int colNum = state.length;
    for (int i = 0; i < colNum; i++) {
        state[i] = cyclicShiftLeft(state[i], i);
    }
}


private static void InvShiftRows(int[][] state) {
    int colNum = state.length;
    for (int i = 0; i < colNum; i++) {
        state[i] = cyclicShiftRight(state[i], i);
    }
}
```

Значения в строках циклически сдвигаются. Число сдвигов зависит от номера строки.

Номер строки	Число сдвигов в право
1	0
2	1
3	2
4	3

Вспомогательная функция циклического сдвига:

```
public static int[] cyclicShiftRight(int[] array, int steps) {
    if (steps == 0) {
        return Arrays.copyOfRange(array, 0, array.length);
    }

    int[] result = new int[array.length];
    int L = array.length;
    steps = steps % array.length;

    if (steps < 0) {
        steps = L + steps;  // L - Abs(steps)
    }

    int[] leftPart = Arrays.copyOfRange(array, 0, L - steps);
    int[] right_part = Arrays.copyOfRange(array, L - steps, L);

    System.arraycopy(right_part, 0, result, 0, steps);
    System.arraycopy(leftPart, 0, result, steps, result.length - steps);

    return result;
}
```

- Функция преобразования столбцов *MixColumns*:

```
private static void MixColumns(int[][] state) {
    int lineNum = 4;
    int[] newCol = {0, 0, 0, 0};

    for (int colId = 0; colId < Nb; colId++) {
        for (int lineId = 0; lineId < lineNum; lineId++) {
            int[] terms = {0, 0, 0, 0};

            for (int i = 0; i < 4; i++) {
                switch (PREDEFINED_MATRIX[lineId][i]) {
                    case 1 -> terms[i] = state[i][colId];
                    case 2 -> terms[i] = GALOIS_MUL_LUT_2[state[i][colId]];
                    case 3 -> terms[i] = GALOIS_MUL_LUT_3[state[i][colId]];
                }
            }
            newCol[lineId] = terms[0] ^ terms[1] ^ terms[2] ^ terms[3];
        }

        for (int lineId = 0; lineId < lineNum; lineId++) {
            state[lineId][colId] = newCol[lineId];
        }
    }
}
```

Для каждого из столбцов в *STATE*, который можно представить как полином, выполняется умножение в поле Галуа на фиксированную матрицу *PREDEFINED_MATRIX*.
Умножение реализовано с помощью массивов длиной 256 включающих значения, соответствующих своему индексу.
Сложение – исключающая дизъюнкция.

```
private static void InvMixColumns(int[][] state) {
    int lineNum = 4;
    int[] newCol = {0, 0, 0, 0};

    for (int colId = 0; colId < Nb; colId++) {
        for (int lineId = 0; lineId < lineNum; lineId++) {
            int[] terms = {0, 0, 0, 0};

            for (int i = 0; i < 4; i++) {
                switch (INV_PREDEFINED_MATRIX[lineId][i]) {
                    case 9 -> terms[i] = GALOIS_MUL_LUT_9[state[i][colId]];
                    case 11 -> terms[i] = GALOIS_MUL_LUT_B[state[i][colId]];
                    case 13 -> terms[i] = GALOIS_MUL_LUT_D[state[i][colId]];
                    case 14 -> terms[i] = GALOIS_MUL_LUT_E[state[i][colId]];
                }
            }
            newCol[lineId] = terms[0] ^ terms[1] ^ terms[2] ^ terms[3];
        }

        for (int lineId = 0; lineId < lineNum; lineId++) {
            state[lineId][colId] = newCol[lineId];
        }
    }
}
```

- Функция сложения с ключом раунда *AddRoundKey*:

```
private static void AddRoundKey(int[][] state, int[][] roundKey) {
    for (int i = 0; i < state.length; i++) {
        for (int j = 0; j < state[0].length; j++) {
            state[i][j] = state[i][j] ^ roundKey[i][j];
        }
    }
}
```

За счет того, что операция XOR сама себе обратна, функция *AddRoundKey* применяется как в процессе шифрования, так и в процессе расшифрования.

Вспомогательные методы модуля Rijndael:

- Функция преобразования байтов *SubWord*:

```
private static int[] SubWord(int[] word) {
    int byteNum = word.length;
    int[] newWord = new int[word.length];

    for (int i = 0; i < byteNum; i++) {
        newWord[i] = S_BOX[word[i]];
    }
    return newWord;
}
```

Функция, используемая в процедуре Key Expansion. Преобразует четырёхбайтовое слово применяя S-Box к каждому из четырёх байтов. Аналогична функции SubBytes.

- Функция сдвига *RotWord*:

```
private static int[] RotWord(int[] word) {
    return cyclicShiftLeft(word, 1);
}
```

Функция, используемая в процедуре Key Expansion. Преобразует четырёхбайтовое слово производя циклический сдвиг на 1 в левую часть.

- Функция расширения ключа *keyExpansion*:

```
private static int[][][] keyExpansion(int[][] cipherKey) {
    // Массив Nr + 1 раундовых ключей.
    // Первые Nk слов расширенного ключа аналогичны Ключу
    int[][] w = new int[Nb * (Nr + 1)][Nk];
    for (int i = 0; i < Nk; i++) {
        for (int j = 0; j < Nk; j++) {
            w[i][j] = cipherKey[j][i];
        }
    }

    int wordId = Nk;
    int roundKeyNum = Nr + 1;
    int wordNum = Nb * roundKeyNum;

    while (wordId < wordNum) {
        int[] prevWord = w[wordId - 1];
        int[] nkBeforeWord = w[wordId - Nk];

        if (wordId % Nk == 0) {
            prevWord = RotWord(prevWord);
            prevWord = SubWord(prevWord);
            int[] rCon = R_CON[wordId / Nk];
            for (int i = 0; i < 4; i++)
                prevWord[i] = prevWord[i] ^ rCon[i]; // XOR prev_word u r_con
        } else if (Nk == 8 & (wordId - 4 % Nk == 0)) {
            prevWord = SubWord(prevWord);
        }
        for (int i = 0; i < 4; i++)
            w[wordId][i] = prevWord[i] ^ nkBeforeWord[i]; // XOR w[i-1] u w[i-Nk]
        wordId++;
    }

    int[][][] roundKeys = new int[Nr + 1][Nk][4];
    for (int i = 0; i < Nr + 1; i++) {
        int[][] cols = Arrays.copyOfRange(w, 4 * i, 4 * (i + 1));

        int[][] roundKey = new int[Nk][4];
        for (int col = 0; col < Nk; col++) {
            for (int line = 0; line < Nk; line++) {
                roundKey[col][line] = cols[line][col];
            }
        }
        roundKeys[i] = roundKey;
    }

    return roundKeys;
}
```

Интерфейс модуля Rijndael:

- Вспомогательные функции шифрования *encryptBlock* и дешифрования блока *decryptBlock*:

```
public static int[][] encryptBlock(int[][] state, int[][] cipherKey) {
    int[][][] roundKeys = keyExpansion(cipherKey);

    AddRoundKey(state, roundKeys[0]);

    for (int roundId = 1; roundId <= Nr; roundId++) {
        SubBytes(state);

        ShiftRows(state);

        if (roundId < Nr) {
            MixColumns(state);
        }

        AddRoundKey(state, roundKeys[roundId]);
    }
    return state;
}


public static int[][] decryptBlock(int[][] state, int[][] cipherKey) {
    int[][][] roundKeys = keyExpansion(cipherKey);

    for (int roundId = Nr; roundId > 0; roundId--) {
        AddRoundKey(state, roundKeys[roundId]);

        if (roundId < Nr) {
            InvMixColumns(state);
        }

        InvShiftRows(state);

        InvSubBytes(state);
    }

    AddRoundKey(state, roundKeys[0]);
    return state;
}
```

- Функции шифрования *encrypt* и дешифрования *decrypt* для режима ECB:

```
public static int[][][] encrypt(int[][][] blocks, int[][] cipherKey) {
    for (int i = 0; i < blocks.length; i++) {
        int[][] encryptedBlock = encryptBlock(blocks[i], cipherKey);
        blocks[i] = encryptedBlock;
    }
    return blocks;
}

public static int[][][] decrypt(int[][][] blocks, int[][] cipherKey) {
    for (int i = 0; i < blocks.length; i++) {
        int[][] decryptedBlock = decryptBlock(blocks[i], cipherKey);
        blocks[i] = decryptedBlock;
    }
    return blocks;
}
```

Все блоки шифруются и дешифруются независимо.

Преимущества режима ECB:

- Все блоки шифруются и расшифровываются быстро [нет доп.расходов кроме непосредственно самой E() & D()] и с постоянной скоростью
- Процесс шифрования и расшифрования можно распараллелить

Недостатки режима ECB:

- сохраняется статистика языка [одинаковым блокам шифротекста соответствуют одинаковые блоки открытого текста]

- Функции шифрования *encryptCBC* и дешифрования *decryptCBC* для режима CBC:

```
public static int[][][] encryptCBC
(int[][][] blocks, int[][] cipherKey, int[][] initializationVector) {
    int[][] xorVector = initializationVector;

    for (int i = 0; i < blocks.length; i++) {
        int[][] plainText = blocks[i];

        AddRoundKey(plainText, xorVector);

        int[][] encryptedBlock = encryptBlock(plainText, cipherKey);

        blocks[i] = encryptedBlock;
        xorVector = encryptedBlock;
    }
    return blocks;
}

public static int[][][] decryptCBC
(int[][][] blocks, int[][] cipherKey, int[][] initializationVector) {
    int[][] xorVector = initializationVector;
    int[][][] result = new int[blocks.length][][];

    for (int i = 0; i < blocks.length; i++) {
        int[][] cipherText = Blocks.copyBlock(blocks[i]);

        int[][] decryptedBlock = decryptBlock(cipherText, cipherKey);

        if (i != 0) {
            xorVector = Blocks.copyBlock(blocks[i - 1]);
        }
        AddRoundKey(decryptedBlock, xorVector);

        result[i] = decryptedBlock;
    }
    return result;
}
```

В данном коде метод AddroundKey используется для побитового XOR блоков текста и вектора инициализации/шифротекста из предыдущих блоков.
При использовании режима CBC не сохраняется статистика языка, все блоки шифруются и расшифровываются быстро [операция XOR очень простая] и процесс расшифрования можно распараллелить. Однако, если в IV допущена ошибка, то дешифрование приведет к неправильному первому блоку, а шифрование – к целиком неправильному тексту. Процесс шифрования нельзя распараллелить. Если первые блоки в двух открытых тестах совпадают, то и первые блоки в соответствующих широтекстах тоже будут совпадать.

Вывод программы:

1. Шифрование одного блока в режиме ECB (без использования вектора инициализации) по шагам:

   Исходные данные:

   - Блок STATE (hex):

```
int[][] state = {
        {0x32, 0x88, 0x31, 0xe0},
        {0x43, 0x5a, 0x31, 0x37},
        {0xf6, 0x30, 0x98, 0x07},
        {0xa8, 0x8d, 0xa2, 0x34}};
```

   - Ключ (hex):

```
int[][] cipherKey = {
        {0x2b, 0x28, 0xab, 0x09},
        {0x7e, 0xae, 0xf7, 0xcf},
        {0x15, 0xd2, 0x15, 0x4f},
        {0x16, 0xa6, 0x88, 0x3c}};
```

   Выполнение:

   - Предварительный раунд:

```
------------------------------------------------------------------------------
Предварительный Раунд
    Шаг 0 (AddRoundKey):
        19 a0 9a e9
        3d f4 c6 f8
        e3 e2 8d 48
        be 2b 2a  8
------------------------------------------------------------------------------
```

- Основные раунды:

```
------------------------------------------------------------------------
Раунд 1
        1. SubBytes     2. ShiftRows      3. MixColumns     4. AddRoundKey
        d4 e0 b8 1e     d4 e0 b8 1e        4 e0 48 28       a4 68 6b  2
        27 bf b4 41     bf b4 41 27       66 cb f8  6       9c 9f 5b 6a
        11 98 5d 52     5d 52 11 98       81 19 d3 26       7f 35 ea 50
        ae f1 e5 30     30 ae f1 e5       e5 9a 7a 4c       f2 2b 43 49

------------------------------------------------------------------------
Раунд 2
        1. SubBytes     2. ShiftRows      3. MixColumns     4. AddRoundKey
        49 45 7f 77     49 45 7f 77       58 1b db 1b       aa 61 82 68
        de db 39  2     db 39  2 de       4d 4b e7 6b       8f dd d2 32
        d2 96 87 53     87 53 d2 96       ca 5a ca b0       5f e3 4a 46
        89 f1 1a 3b     3b 89 f1 1a       f1 ac a8 e5        3 ef d2 9a

------------------------------------------------------------------------
…
```

- Последний раунд:

```
------------------------------------------------------------------------
Раунд 10
        1. SubBytes     2. ShiftRows      4. AddRoundKey
        e9 cb 3d af     e9 cb 3d af       39  2 dc 19
         9 31 32 2e     31 32 2e  9       25 dc 11 6a
        89  7 7d 2c     7d 2c 89  7       84  9 85  b
        72 5f 94 b5     b5 72 5f 94       1d fb 97 32

------------------------------------------------------------------------
```

2. Шифрование текста (длина текста больше длины блока) в режиме ECB:

```
[MAIN] Open Text:   hello world! My name is Daniel, and I am 21 years old. Bye Bye

[MAIN] Cipher Key:  elena valentinovna

[ENCRYPTION] Cipher Text:   ◆◆◆
◆◆S◘"Nn◆A◆%D◆◆sZ◆◘◆c◘◆]ъ◆◘◆3◆◘◘@◘◆AA'o◆◘◆◆◘◆◆◘y◆◆&sD◘I◆m◘

[DECRYPTION] Decrypted Text:    hello world! My name is Daniel, and I am 21 years old. Bye Bye◘◘
```

Из результата работы программы видно, что при делении текста на блоки, открытый текст было дополнен так, чтобы его длина была кратна длине блока. При дешифровке эти дополняющие нулевые символы никуда не пропали (последняя строчка).

Кроме того видно, что шифротекст не представляет из тебя текст, т.к. алгоритм шифрования работает не на уровне символов, а на уровне байт.

3. Шифрование текста в режиме CBC по шагам (открытый текст и ключ тот же, что и в примере 2):

```
[ENCRYPTION]
        IV              Plain text   After XOR    Cipher Text
        65 6c 65 6e     68 65 6c 6c   d  9  9  2  66 ff d6 98
        61 20 76 61     6f 20 77 6f   e  0  1  e   7 63 2b  5
        6c 65 6e 74     72 6c 64 21  1e  9  a 55  1d dc 9e f4
        69 6e 6f 76     20 4d 79 20  49 23 16 56  96 82 d8 4a

        66 ff d6 98     6e 61 6d 65   8 9e bb fd  18 f3 cf  0
         7 63 2b  5     20 69 73 20  27  a 58 25  2b 60  f 48
        1d dc 9e f4     44 61 6e 69  59 bd f0 9d  fa 36 ec 45
        96 82 d8 4a     65 6c 2c 20  f3 ee f4 6a  78 54  8 1d

        18 f3 cf  0     61 6e 64 20  79 9d ab 20  fe 80 82 d7
        2b 60  f 48     49 20 61 6d  62 40 6e 25  86 6b ce  a
        fa 36 ec 45     20 32 31 20  da  4 dd 65  e5 1e 9b  f
        78 54  8 1d     79 65 61 72   1 31 69 6f  4b 48 30 83

        fe 80 82 d7     73 20 6f 6c  8d a0 ed bb   d f7 ce bb
        86 6b ce  a     64 2e 20 42  e2 45 ee 48  63  3 8d 4e
        e5 1e 9b  f     79 65 20 42  9c 7b bb 4d  e6 88 f7 65
        4b 48 30 83     79 65  0  0  32 2d 30 83  e0 28 18 bf

[CIPHER TEXT]      f□c□□□□□J□□ +`H□6□Ex□□□k□
□λc□N□□e□(□

[DECRYPTION]
        Cipher Text     XORed Text   IV           Plain text
        66 ff d6 98      d  9  9  2  65 6c 65 6e  68 65 6c 6c
         7 63 2b  5      e  0  1  e  61 20 76 61  6f 20 77 6f
        1d dc 9e f4     1e  9  a 55  6c 65 6e 74  72 6c 64 21
        96 82 d8 4a     49 23 16 56  69 6e 6f 76  20 4d 79 20

        18 f3 cf  0      8 9e bb fd  66 ff d6 98  6e 61 6d 65
        2b 60  f 48     27  a 58 25   7 63 2b  5  20 69 73 20
        fa 36 ec 45     59 bd f0 9d  1d dc 9e f4  44 61 6e 69
        78 54  8 1d     f3 ee f4 6a  96 82 d8 4a  65 6c 2c 20

        fe 80 82 d7     79 9d ab 20  18 f3 cf  0  61 6e 64 20
        86 6b ce  a     62 40 6e 25  2b 60  f 48  49 20 61 6d
        e5 1e 9b  f     da  4 dd 65  fa 36 ec 45  20 32 31 20
        4b 48 30 83      1 31 69 6f  78 54  8 1d  79 65 61 72

         d f7 ce bb     8d a0 ed bb  fe 80 82 d7  73 20 6f 6c
        63  3 8d 4e     e2 45 ee 48  86 6b ce  a  64 2e 20 42
        e6 88 f7 65     9c 7b bb 4d  e5 1e 9b  f  79 65 20 42
        e0 28 18 bf     32 2d 30 83  4b 48 30 83  79 65  0  0

[DECRYPTED TEXT]   hello world! My name is Daniel, and I am 21 years old. Bye Bye
```

4. Демонстрация использования отличающихся векторов инициализации в процессе шифрования и расшифрования:

```
[MAIN] Open Text:       hello world! My name is Daniel, and I am 21 years
old. Bye Bye

[MAIN] Cipher Key 1:    elena valentinovna

[MAIN] Cipher Key 2:    larisa Vladimirovna

[CIPHER TEXT]     f□ c̆+  ⍚□□□□J □□ +` H□6□Ex □□□ɪk□
□λc □N□□e□( □

[DECRYPTED TEXT]  ah{k}a!xrhn<$Jd9name is Daniel, and I am 21 years old. Bye
Bye
```

Вывод:

В ходе данной лабораторной работы был закреплен материал об алгоритмах симметричного блочного шифрования и режимах шифрования.

Был создан программный модуль на языке программирования Java реализующий алгоритм Rijndael в режиме CBC.