Thực hành thuật toán AES 128bit.

Sinh viên thực hiện: Vũ Xuân Phong

I. Mô tả về code thực hiện thuật toán AES 128 bit.

- Thuật toán được thực hiện với code Java và IDE Intellij.

- Thuật toán AES 128 bit mỗi lần thực hiện với một khối 4x4 byte, tức 16 kí tự theo mã Ascii, và khóa cũng có dạng 4x4 byte tức 16 kí tự theo mã Ascii.

- Chương trình mỗi lần thực hiện với một chuỗi 16 kí tự, mỗi kí tự được chuyển sang mã Ascii tương ứng và chuyển thành dạng các block 4x4. Nếu chuỗi kí tự đầu vào có kích thước nhỏ hơn 16, chương trình sẽ tự động thêm vào các kí tự '!' để cho đủ 16 kí tự. Nếu chuỗi đầu vào có kích thước lớn hơn 16 thì sẽ được chia thành nhiều đoạn 16 kí tự và thực hiện mã hóa nhiều lần.

- Khóa đầu vào cũng là một chuỗi 16 kí tự. Nếu khóa nhập vào không đủ 16 kí tự thì chương trình sẽ tự động thêm các kí tự '!'. Còn nếu chuỗi khóa nhập vào lớn hơn 16 kí tự, chương trình sẽ chỉ lấy 16 kí tự đầu của chuỗi nhập vào.

- Thuật toán mã hóa AES 128 bit có thể được chia làm 2 khối chính: Tạo khóa, mã hóa và giải mã.

II. Chi tiết về code thực hiện thuật toán AES 128 bit.

2.1. Một số khai báo khởi tạo.

- Bảng Sub Box được sử dụng trong SubBytes:

```
30      private static final String[][] subBox= {
31              {"63","7c","77","7b","f2","6b","6f","c5","30","01","67","2b","fe","d7","ab","76"},
32              {"ca","82","c9","7d","fa","59","47","f0","ad","d4","a2","af","9c","a4","72","c0"},
33              {"b7","fd","93","26","36","3f","f7","cc","34","a5","e5","f1","71","d8","31","15"},
34              {"04","c7","23","c3","18","96","05","9a","07","12","80","e2","eb","27","b2","75"},
35              {"09","83","2c","1a","1b","6e","5a","a0","52","3b","d6","b3","29","e3","2f","84"},
36              {"53","d1","00","ed","20","fc","b1","5b","6a","cb","be","39","4a","4c","58","cf"},
37              {"d0","ef","aa","fb","43","4d","33","85","45","f9","02","7f","50","3c","9f","a8"},
38              {"51","a3","40","8f","92","9d","38","f5","bc","b6","da","21","10","ff","f3","d2"},
39              {"cd","0c","13","ec","5f","97","44","17","c4","a7","7e","3d","64","5d","19","73"},
40              {"60","81","4f","dc","22","2a","90","88","46","ee","b8","14","de","5e","0b","db"},
41              {"e0","32","3a","0a","49","06","24","5c","c2","d3","ac","62","91","95","e4","79"},
42              {"e7","c8","37","6d","8d","d5","4e","a9","6c","56","f4","ea","65","7a","ae","08"},
43              {"ba","78","25","2e","1c","a6","b4","c6","e8","dd","74","1f","4b","bd","8b","8a"},
44              {"70","3e","b5","66","48","03","f6","0e","61","35","57","b9","86","c1","1d","9e"},
45              {"e1","f8","98","11","69","d9","8e","94","9b","1e","87","e9","ce","55","28","df"},
46              {"8c","a1","89","0d","bf","e6","42","68","41","99","2d","0f","b0","54","bb","16"}
47      };
```

- Bảng inverse Sub box được sử dụng trong invSubBytes:

```
12      private static final String[][] ivSubbox = {
13              {"52","09","6A","D5","30","36","A5","38","BF","40","A3","9E","81","F3","D7","FB"},
14              {"7C","E3","39","82","9b","2f","ff","87","34","8e","43","44","c4","de","e9","cb"},
15              {"54","7b","94","32","a6","c2","23","3d","ee","4c","95","0b","42","fa","c3","4e"},
16              {"08","2e","a1","66","28","d9","24","b2","76","5b","a2","49","6d","8b","d1","25"},
17              {"72","f8","f6","64","86","68","98","16","d4","a4","5c","cc","5d","65","b6","92"},
18              {"6c","70","48","50","fd","ed","b9","da","5e","15","46","57","a7","8d","9d","84"},
19              {"90","d8","ab","00","8c","bc","d3","0a","f7","e4","58","05","b8","b3","45","06"},
20              {"d0","2c","1e","8f","ca","3f","0f","02","c1","af","bd","03","01","13","8a","6b"},
21              {"3a","91","11","41","4f","67","dc","ea","97","f2","cf","ce","f0","b4","e6","73"},
22              {"96","ac","74","22","e7","ad","35","85","e2","f9","37","e8","1c","75","df","6e"},
23              {"47","f1","1a","71","1d","29","c5","89","6f","b7","62","0e","aa","18","be","1b"},
24              {"fc","56","3e","4b","c6","d2","79","20","9a","db","c0","fe","78","cd","5a","f4"},
25              {"1f","dd","a8","33","88","07","c7","31","b1","12","10","59","27","80","ec","5f"},
26              {"60","51","7f","a9","19","b5","4a","0d","2d","e5","7a","9f","93","c9","9c","ef"},
27              {"a0","e0","3b","4d","ae","2a","f5","b0","c8","eb","bb","3c","83","53","99","61"},
28              {"17","2b","04","7e","ba","77","d6","26","e1","69","14","63","55","21","0c","7d"}
29      };
```

- Hai khối 4x4 được sử dụng trong MixColumns và invMixColumns:

```
61      private static final int[][] mixColumn={
62              {2,3,1,1},
63              {1,2,3,1},
64              {1,1,2,3},
65              {3,1,1,2}
66      };
67      private static final int[][] invMixColumn={
68              {14,11,13,9},
69              {9,14,11,13},
70              {13,9,14,11},
71              {11,13,9,14}
72      };
```

- Các Rcon được sử dụng trong thuật toán tạo khóa:

```
79      private static final int[] Rcon1 = {1,0,0,0};
80      private static final int[] Rcon2 = {2,0,0,0};
81      private static final int[] Rcon3 = {4,0,0,0};
82      private static final int[] Rcon4 = {8,0,0,0};
83      private static final int[] Rcon5 = {16,0,0,0};
84      private static final int[] Rcon6 = {32,0,0,0};
85      private static final int[] Rcon7 = {64,0,0,0};
86      private static final int[] Rcon8 = {128,0,0,0};
87      private static final int[] Rcon9 = {27,0,0,0};
88      private static final int[] Rcon10 = {54,0,0,0};
```

2.2. Một số hàm chức năng.

- Hàm chuyển một chuỗi 16 kí tự sang dạng block 4x4:

```
194     public int[][] toState(String s){
195         int [][] toReturn = new int[4][4];
196         for (int i = 0; i < 4;i++){
197             for (int j = 0; j < 4 ; j ++){
198                 toReturn[i][j] = s.charAt(i*4+j);
199                 if (j==3) break;
200             }
201         }
202         return toReturn;
203     }
```

- Hàm chuyển một số hệ 10 sang dạng hệ cơ số 16:

```
167 @       private int[] toHex(int dec){
168             int [] hex = new int[2];
169             hex[0] = dec/16;
170             hex[1] = dec%16;
171             return hex;
172         }
```

- Hàm thực hiện phép xor 2 byte với nhau:

```
265     public int[]xor(int [] a, int []b){
266         int []toReturn  = new int[8];
267         for (int i =0;i<8;i++){
268             if (a[i]==b[i]){
269                 toReturn[i]=0;
270             }
271             else toReturn[i]=1;
272         }
273         return toReturn;
274     }
```

- Hàm chuyển một số hệ 10 sang 1 byte:

```
334     public int[] toBinary(int dec){
335         int [] toBinary  = new int[8];
336         for(int i = 0; i < 8;i++){
337             toBinary[i] = dec%2;
338             dec=dec/2;
339         }
340         return toBinary;
341     }
```

- Hàm chuyển một byte sang số cơ số 10:

```
347     public int toDec(int []a){
348         return a[0]+a[1]*2+a[2]*4+a[3]*8+a[4]*16+a[5]*32+a[6]*64+a[7]*128;
349     }
```

- Hàm chuyển một khối 4x4 sang dạng chuỗi 16 kí tự:

```java
public String stateToString(int [][] a){
    String s = "";
    for (int i = 0 ; i < 4 ; i ++){
        for(int j = 0 ; j < 4 ; j ++){
            s+=(char)a[i][j];
        }
    }
    return s;
}
```

2.3. Tạo khóa.

- Hàm tạo khóa:

```java
private int [][]createKey(int [][] cipher, int i){
    int [][] toReturn  = new int[4][4];
    int [] columnFinal = new int[4];
    columnFinal[0] = cipher[1][3];
    columnFinal[1] = cipher[2][3];
    columnFinal[2] = cipher[3][3];
    columnFinal[3] =cipher[0][3];
    columnFinal[0] = subADec(columnFinal[0]);
    columnFinal[1] = subADec(columnFinal[1]);
    columnFinal[2] = subADec(columnFinal[2]);
    columnFinal[3] = subADec(columnFinal[3]);
    for (int j = 0 ; j < 4 ; j ++){
        for (int k = 0 ; k < 4 ; k ++){
            int []a = toBinary(cipher[k][j]);
            int []b = toBinary(columnFinal[k]);
            int []c = toBinary(listRcon.get(i)[k]);
            if (j ==0){
                int []d = xor(a,b);
                int x = toDec(xor(c,d));
                columnFinal[k] = x;
                toReturn[k][0] = x;
            } else {
                int y = toDec(xor(a,b));
                columnFinal[k] = y;
                toReturn[k][j] = y;
            }
        }
    }
    return toReturn;
}
```

- Với đầu vào là khối cipherKey 4x4 được tạo thành từ chuỗi khóa nhập vào.
- Với i là thứ tự RCon thứ i được sử dụng.

- Mỗi lần hàm createKey chỉ tạo ra một khóa con.

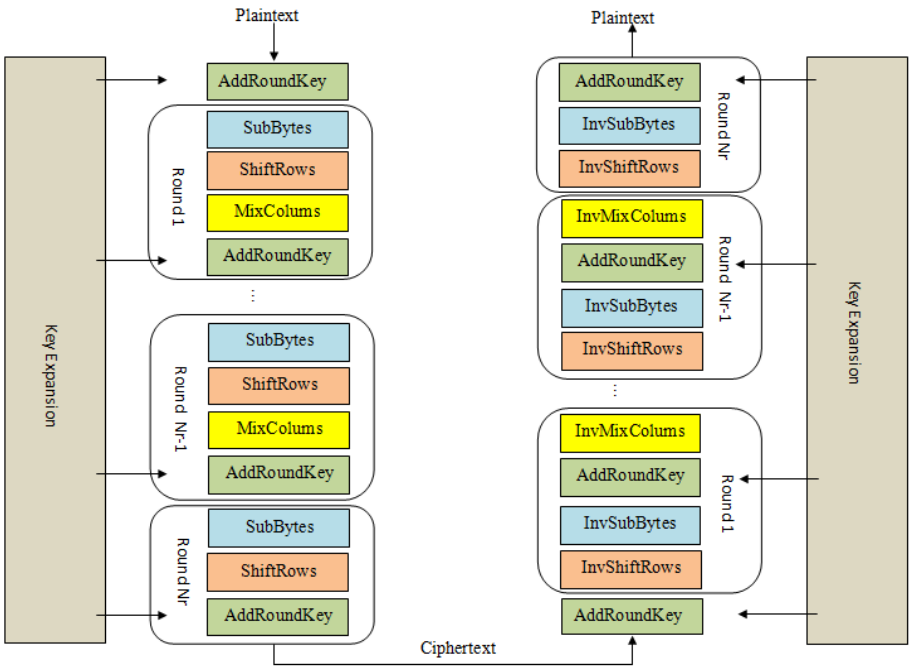
- Hàm subADec thực hiện việc thay thế giá trị bằng một giá trị trong bảng Sub box.

```
373     public int subADec(int dec){
374         int []a = toHex(dec);
375         String s = subBox[a[0]][a[1]];
376         return Integer.parseInt(s, radix: 16);
377     }
```

- Hàm createListKey tạo ra 10 khóa con, phục vụ cho việc mã hóa và giải mã.

```
408     public void  createListKey(int [][] cipher){
409         for (int i = 0 ; i < 10 ; i ++ ){
410             int [][] keySmall = createKey(cipher,i);
411             //printState(keySmall);
412             listKey.add(keySmall);
413             cipher = keySmall;
414         }
415     }
```

2.4. Mã hóa và giải mã.



- Dựa theo mô hình trên, thuật toán AES gồm 4 khối: SubBytes, ShiftRows, MixColumns, AddRoundKey.

2.3.1. SubBytes và invSubBytes.

- Thực hiện việc thay thế các giá trị trong khối 4x4 bằng các giá trị trong một bảng SubBox hoặc bảng invSubBox co sẵn.
- Bảng SubBox được sử dụng trong mã hóa, còn bảng invSubBox được sử dụng trong giải mã.
- Hàm thực hiện việc subBytes và invSubBytes.

```
214     private int [][] subByte(int [][] tosub, String[][] subbox){
215         int [][] toReturn  = new int[tosub.length][tosub[0].length];
216         for (int i = 0 ; i < tosub.length;i++){
217             for (int j = 0 ;j < tosub[0].length;j++){
218                 int []a = toHex(tosub[i][j]);
219                 toReturn[i][j] = Integer.parseInt(subbox[a[0]][a[1]], radix: 16);
220             }
221         }
222         return toReturn;
223     }
```

- Với toSub là khối 4x4 cần thực hiện việc thay thế.
- Và subbox là bảng được sử dụng. Nếu là mã hóa thì subbox là SubBox, còn giải mã thì subbox là invSubBox.

2.3.2. ShiftRows và invShiftRows.

- Là việc thực hiện dịch tùy theo hàng, hàng thứ nhất không dịch, hàng thứ 2 dịch 1, hàng thứ 3 dịch 2 và hàng cuối cùng dịch 3.
- Hàm thực hiện ShiftRows và invShiftRows:

```
232     public int[][] shiftRow(int[][] toShiftRow, int[][] shiftBox){
233         int [][] toReturn = new int[toShiftRow.length][toShiftRow[0].length];
234         for (int i=0;i<toShiftRow.length;i++){
235             for (int j=0;j<toShiftRow[0].length;j++){
236                 int x = shiftBox[i][j];
237                 int a = x/4;
238                 int b = x%4;
239                 toReturn[i][j] = toShiftRow[a][b];
240             }
241         }
242         return toReturn;
243     }
```

- Với toShiftRow là khối 4x4 cần được thực hiện shiftrows.
- Và shiftBox là khối được sử dụng để thực hiện việc shiftrow hoặc invshiftrow, shiftBox sẽ nhận giá trị là một trong hai khối.

```
48      private static final int[][] invshiftRow = {
49              {0,1,2,3},
50              {7,4,5,6},
51              {10,11,8,9},
52              {13,14,15,12}
53      };
54
55      private static final int[][] shiftRow = {
56              {0,1,2,3},
57              {5,6,7,4},
58              {10,11,8,9},
59              {15,12,13,14}
60      };
```

- Khối shiftRow tương ứng với quá trình mã hóa, còn invshiftRow tương ứng với quá trình giải mã.

2.3.3. MixColums và invMixColumns.

- Là quá trình thực hiện việc trộn các hàng của một khối 4x4 bằng cách nhân với một khối 4x4 khác được cho sẵn (mixColumn với quá trình mã hóa và invMixColumn với quá trình giải mã).
- Để thực hiện MixColumns và invMixColumns ta cần thực hiện các phép nhân.
- Hàm thực hiện phép nhân 2.

```
251     public int[] mul2(int [] abyte){
252         int[] toReturn = new int[abyte.length];
253         toReturn[7] = abyte[6];
254         toReturn[6] = abyte[5];
255         toReturn[5] = abyte[4];
256         toReturn[4] = abyte[3];
257         toReturn[3] = abyte[2];
258         toReturn[2] = abyte[1];
259         toReturn[1] = abyte[0];
260         toReturn[0] = 0;
261         if (abyte[7]==1) return xor(toReturn,b);
262         else return toReturn;
263     }
```

- Các phép nhân với 3, 9, 11, 13, 14 đều dựa trên phép nhân 2. Giả sử phép nhân x với 3.

$$3 \times x = (2 \oplus 1) \times x = (2 \times x) \oplus x$$

- Hàm thực hiện các phép nhân:

```
    public int[] Mulx(int[] a, int x){
        int[] toReturn = new int[8];
        switch (x){
            case 1:
                toReturn= a;
                break;
            case 2:
                toReturn = Mul2(a);
                break;
            case 3:
                int [] temp = Mul2(a);
                toReturn = xOr(temp,a);
                break;
            case 9:
                int[] temp1 = Mul2(a);
                 temp1 = Mul2(temp1);
                temp1 = Mul2(temp1);
                toReturn = xOr(temp1,a);

                break;
            case 11:
                int []temp2 = Mul2(a);
                temp2 = Mul2(temp2);
                 temp2 = xOr(temp2,a);
                 temp2 = Mul2(temp2);
                toReturn = xOr(temp2,a);
                break;
            case 13:
                int[] temp3 = Mul2(a);
                temp3 = xOr(temp3,a);
                temp3 = Mul2(temp3);
                temp3 = Mul2(temp3);
                toReturn = xOr(temp3,a);
                break;
            case 14:
                int []temp4 =  Mul2(a);
                temp4 = xOr(temp4,a);
                temp4 = Mul2(temp4);
                temp4 =xOr(temp4,a);
                toReturn = Mul2(temp4);
                break;
            default:
                toReturn = null;
                break;
        }

        return toReturn;
    }
```

- Cuối cùng, hàm thực hiện mixColumns:

```
351     private int[][] mixColumn(int [][] toMix, int[][] mixBox){
352         int [][] toReturn = new int[4][4];
353         for (int j = 0; j < 4 ; j ++) {
354             for (int i = 0; i < 4; i++) {
355                 int[] a1 = mulx(toBinary(toMix[0][j]), mixBox[i][0]);
356                 int[] a2 = mulx(toBinary(toMix[1][j]), mixBox[i][1]);
357                 int[] a3 = mulx(toBinary(toMix[2][j]), mixBox[i][2]);
358                 int[] a4 = mulx(toBinary(toMix[3][j]), mixBox[i][3]);
359                 int[] a12 = xor(a1, a2);
360                 int[] a34 = xor(a3, a4);
361                 int[] a = xor(a12, a34);
362                 toReturn[i][j]= toDec(a);
363             }
364         }
365         return toReturn;
366     }
```

- Với mixBox nhận giá trị là mixcolumn hoặc invcolumn tùy thuộc vào quá trình là mã hóa hay giải mã.

2.3.4. AddRoundKey.

- Thực hiện công khối 4x4 với khóa tương ứng của vòng.

```
418     public int[][] addRoundKey(int [][] toAdd, int [][] keySmall){
419         int [][] toReturn = new int[4][4];
420         for (int i = 0 ; i < 4; i ++){
421             for (int j = 0 ; j < 4 ; j ++) {
422                 int[] a = toBinary(toAdd[i][j]);
423                 int[] b = toBinary(keySmall[i][j]);
424                 toReturn[i][j] = toDec(xor(a,b));
425             }
426         }
427         return toReturn;
428     }
```

2.3.5. Mã hóa.

- Như trong mô hình, đầu tiên là thực hiện addRoundKey sau đó bao gồm 10 vòng, mỗi vòng thực hiện subBytes, shiftRows, mixColumns và addRoundKey. Riêng vòng cuối cùng không dung mixColumns.

```
457     public String encode(int [][]state){
458         //printState(state);
459          int [][] first = addRoundKey(state,cipherkey);
460          //printState(first);
461        for (int i = 0 ; i < 9;i++){
462             first = subByte(first,subBox);
463            // printState(first);
464             int[][] b = shiftRow(first,shiftRow);
465            // printState(b);
466             int [][] c = mixColumn(b,mixColumn);
467            // printState(c);
468             first = addRoundKey(c,listKey.get(i));
469            // printState(first);
470        }
471        int [][] a1 = subByte(first,subBox);
472         //printState(a1);
473        int [][] a2 = shiftRow(a1,shiftRow);
474        //printState(a2);
475        int [][] toReturn = addRoundKey(a2,listKey.get(9));
476        printState(toReturn);
477         return stateToString(toReturn);
478 
479     }
```

2.3.6. Giải mã.

- Cũng thực hiện theo mô hình và thực hiện ngược lại so với quá trình mã hóa.

```
480     public String decode(int [][] state){
481         //printState(state);
482         int [][] decode = addRoundKey(state,listKey.get(9));
483         //printState(decode);
484         for (int i = 8; i >=0 ; i --){
485             decode = shiftRow(decode,invshiftRow);
486           //  printState(decode);
487             int [][] a = subByte(decode,ivSubbox);
488             //printState(a);
489             int [][] b = addRoundKey(a,listKey.get(i));
490             //printState(b);
491             decode = mixColumn(b,invMixColumn);
492             //printState(decode);
493         }
494         int[][] a1 = shiftRow(decode,invshiftRow);
495         //printState(a1);
496         int [][] a2 = subByte(a1,ivSubbox);
497         //printState(a2);
498         int[][] toReturn  = addRoundKey(a2,cipherkey);
499         printState(toReturn);
500         return stateToString(toReturn);
501     }
502 }
```

Em cảm ơn ạ.