Thực hành thuật toán DES

Sinh viên thực hiện: Vũ Xuân Phong

I. Mô tả về code thực hiện thuật toán DES.

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

- Thuật toán DES thực hiện với một khối tin nhắn 8x8 bit, tức 64bit và khóa cũng là một khối 8x8 bit, 64bit.

- Chương trình mỗi lần thực hiện với tin nhắn chuỗi 8 kí tự, mỗi kí tự được chuyển sang mã ASCII tương ứng và chuyển sang dạng bit và thành 8 bit. Vì vậy mỗi chuỗi 8 kí tự được chuyển thành một khối 8x8 bit. Đầu vào của chương trình là một chuỗi không xác định (không dấu). Nếu kích thước chuỗi lớn hơn 8 sẽ được chia thành nhiều đoạn nhỏ có kích thước 8 kí tự và thực hiện nhiều lần, nếu đoạn có kích thước nhỏ hơn 8, chương trình sẽ tự động thêm kí tự '!' vào đến khi kích thước là 8. Các đoạn nhỏ 8 kí tự sau khi mã hóa sẽ được ghép lại và thành bản mã cho tin nhắn đầu vào.

- Khóa cũng là một chuỗi 8 kí tự. Nếu chuỗi khóa nhập vào có kích thước nhỏ hơn 8, chương trình sẽ tự động thêm kí tự '!' vào. Còn nếu chuỗi khóa nhập vào có kích thước lớn hơn 8, chương trình sẽ chỉ lấy 8 kí tự đầu của chuỗi nhập vào.

- Thuật toán có thể được chia ra thành ba khối chính: mã hóa, tạo mã và giải mã.

II. Chi tiết về code thực hiện thuật toán.

2.1. Một số khai báo khởi tạo:

- Hai khối hoán vị đầu cuối:

```
46      private final static int[][] IP= {
47              {57,49,41,33,25,17,9,1},
48              {59,51,43,35,27,19,11,3},
49              {61,53,45,37,29,21,13,5},
50              {63,55,47,39,31,23,15,7},
51              {56,48,40,32,24,16,8,0},
52              {58,50,42,34,26,18,10,2},
53              {60,52,44,36,28,20,12,4},
54              {62,54,46,38,30,22,14,6}
55      };
56
57      private final static int [][] FP = {
58              {39,7,47,15,55,23,63,31},
59              {38,6,46,14,54,22,62,30},
60              {37,5,45,13,53,21,61,29},
61              {36,4,44,12,52,20,60,28},
62              {35,3,43,11,51,19,59,27},
63              {34,2,42,10,50,18,58,26},
64              {33,1,41,9,49,17,57,25},
65              {32,0,40,8,48,16,56,24}
66      };
```

- Khối PC1 và PC2 được sử dụng trong thuật toán tạo khóa:

```
68      private final static int[][] PC1 = {
69              {56,48,40,32,24,16,8},
70              {0,57,49,41,33,25,17},
71              {9,1,58,50,42,34,26},
72              {18,10,2,59,51,43,35},
73              {62,54,46,38,30,22,14},
74              {6,61,53,45,37,29,21},
75              {13,5,60,52,44,36,28},
76              {20,12,4,27,19,11,3}
77      };
78
79      private final static int[][] PC2 ={
80              {13,16,10,23,0,4},
81              {2,27,14,5,20,9},
82              {22,18,11,3,25,7},
83              {15,6,26,19,12,1},
84              {40,51,30,36,46,54},
85              {29,39,50,44,32,47},
86              {43,48,38,55,33,52},
87              {45,41,49,35,28,31}
88      };
```

- Tám khối Sbox được sử dụng bên trong hàm Feistel:

```
109     private final static int[][] S1 = {{14,04,13,01,02,15,11,8,03,10,06,12,05,9,00,07},
110             {0,15,7,4,14,2,13,10,3,6,12,11,9,5,3,8},
111             {4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0},
112             {15,2,8,2,4,9,1,7,5,11,3,14,10,0,6,13}
113     };
114     private final static int[][] S2 = {{15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10},
115             {3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5},
116             {0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15},
117             {13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9}
118     };
119     private final static int[][] S3 = {{10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8},
120             {13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1},
121             {13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7},
122             {1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12}
123     };
124     private final static int[][] S4 = {{7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15},
125             {13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9},
126             {10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4},
127             {3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14}
128 };
```

```
129     private final static int[][] S5={{2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9},
130             {14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6},
131             {4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14},
132             {11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3}
133     };
134     private final static int[][] S6={{12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11},
135             {10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8},
136             {9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6},
137             {4,3,2,12,9,5,15,10,11,14,1,7,10,0,8,13}
138 };
139     private final static int[][] S7={{4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1},
140             {13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6},
141             {1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2},
142             {6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12}
143     };
144     private final static int[][] S8={{13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7},
145             {1,15,13,8,10,3,7,4,12,5,6,11,10,14,9,2},
146             {7,11,4,1,9,12,14,2,0,6,10,10,15,3,5,8},
147             {2,1,14,7,4,10,8,13,15,12,9,9,3,5,6,11}
148 };
```

- Khối E và Pbox được sử dụng bên trong hàm Feistel:

```
private final static  int[][] E = {
        {31,0,1,2,3,4},
        {3,4,5,6,7,8},
        {7,8,9,10,11,12},
        {11,12,13,14,15,16},
        {15,16,17,18,19,20},
        {19,20,21,22,23,24},
        {23,24,25,26,27,28},
        {27,28,29,30,31,0}
};
```

```
private final static int [][] Pbox = {
        {15,6,19,20,28,11,27,16},
        {0,14,22,25,4,17,30,9},
        {1,7,23,13,31,26,2,8},
        {18,12,29,5,21,10,3,24}
};
```

2.2. Một số hàm chức năng.

- Hàm chuyển một số dạng hệ số 10 sang dạng bit:

```
174     public int[] toBinary(int x,int n){
175         int [] a = new int[n];
176         for (int i = 0;i<n;i++){
177             int du = x%2;
178             x = x/2;
179             a[i] = du;
180         }
181         return a;
182     }
```

  - Với x là số cần chuyển sang dạng bit.
  - Với n là số bit của kết quả đầu ra.

- Hàm thực hiện chuyển một chuỗi sang dạng các block 8x8 :

```
373     public ArrayList<int[][]> toState(String s){
374         ArrayList<String> X = inputToArray(s);
375         ArrayList<int[][]> listState = new ArrayList<>();
376         int [][] state = new int[8][8];
377         for (int i = 0; i < X.size();i++){
378             for (int j =0; j < 8;j++){
379                 int x = X.get(i).charAt(j);
380                 state[j] = toBinary(x, n: 8);
381             }
382             listState.add(state);
383             state = new int[8][8];
384         }
385
386         return listState;
387     }
```

  - Với đầu vào s là chuỗi cần chuyển sang dạng block 8x8.
  - Kết quả trả về là một danh sách chứa các block 8x8. Việc mã hóa thực hiện với các block này.

- Hàm hoán vị : thực hiện việc hoán vị các khối theo các khối cho sẵn của thuật toán.

```
184     public int[][] hoanVi(int [][] state, int [][] matrixHoanVi){
185         int n = state[0].length;
186         int [][] stateReturn =  new int[matrixHoanVi.length][matrixHoanVi[0].length];
187         for (int i =0 ; i < matrixHoanVi.length; i++){
188             for (int j = 0; j < matrixHoanVi[0].length;j++){
189                 int x = matrixHoanVi[i][j];
190                 int a = x/n;
191                 int b = x%n;
192                 stateReturn[i][j] = state[a][b];
193             }
194         }
195         return stateReturn;
196     }
```

  - Vỡi state là khối đầu vào cần được hoán vị và matrixHoanVi là khối được cho sẵn bởi thuật toán.
  - Ví dụ với hoán vị đầu với khối IP thì sẽ gọi hàm hoán vị như sau : hoanVi (state, IP). Với IP là khối đã được khai báo phía trên.

- Hàm xor, thực hiện phép xor giữa hai block bit với nhau :

```
361     public int [][] xor(int [][] X, int [][] Y){
362         int [][] xorReturn = new int[X.length][X[0].length];
363         for (int i = 0 ; i < X.length;i ++){
364             for (int j=0;j<X[0].length;j++){
365                 if (X[i][j]==Y[i][j]){
366                     xorReturn[i][j] = 0;
367                 }
368                 else xorReturn[i][j] = 1;
369             }
370         }
371         return xorReturn;
372     }
```

- Hàm stateToString : chuyển một block 8x8 thành dạng chuỗi :

```
304 public String statetoString(int [][] x){
305     StringBuilder stringReturn = new StringBuilder();
306     for (int i = 0 ; i < 8; i++){
307         int c = x[i][0] + x[i][1]*2 + x[i][2]*4+x[i][3]*8+x[i][4]*16 + x[i][5]*32 + x[i][6]*64+x[i][7]*128;
308         stringReturn.append ((char) c);
309     }
310     return stringReturn.toString();
311 }
```

2.3. Tạo khóa:

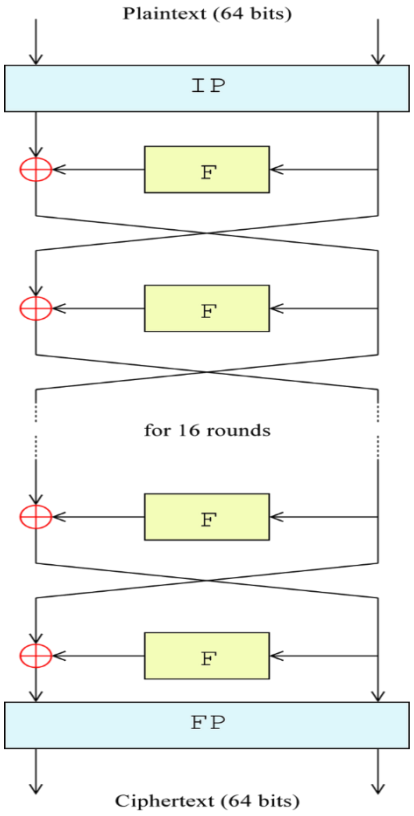
- Khối 64 bit đầu vào được nén thành 56 bit bằng cách cho hoán vị qua block PC1. Sau đó được chia thành hai nửa trên và nửa dưới, và trải qua 16 vòng lặp. Với các vòng thực hiện dịch vòng 2 cả hai nửa trên và dưới, trừ vòng thứ 1,2,9,16 thực hiện dịch vòng 1. Các nửa sau mỗi vòng lặp được ghép lại với nhau và được hoán vị lần nữa qua PC2, sau mỗi vòng lặp một khóa con được tạo thành.

```
198     public void  createKey(){
199         int[][] stateKey = toState(key).get(0);
200         int [][] stateKeyPC1 = hoanVi(stateKey,PC1);
201         int[][] C = new int[4][7];
202         int[][] D = new int[4][7];
203         C[0] = stateKeyPC1[0];
204         C[1] = stateKeyPC1[1];
205         C[2] = stateKeyPC1[2];
206         C[3] = stateKeyPC1[3];
207         D[0] = stateKeyPC1[4];
208         D[1] = stateKeyPC1[5];
209         D[2] = stateKeyPC1[6];
210         D[3] = stateKeyPC1[7];
211         for (int i = 0 ; i < 16;i++){
212             if (i==0||i==1||i==8||i==15){
213                 C = hoanVi(C,XV1);//
214                 D = hoanVi(D,XV1);
215             } else {
216                 C = hoanVi(C,XV2);
217                 D = hoanVi(D,XV2);
218             }
219             int [][] keyTemp = new int[8][7];
220             keyTemp[0] = C[0];
221             keyTemp[1] = C[1];
222             keyTemp[2] = C[2];
223             keyTemp[3] = C[3];
224             keyTemp[4] = D[0];
225             keyTemp[5] = D[1];
226             keyTemp[6] = D[2];
227             keyTemp[7] = D[3];
228             int [][] key = hoanVi(keyTemp,PC2);
229             listKey.add(key);
230         }
231     }
```

- Sau hàm trên 16 khóa con được lưu vào listKey.
- Ở đây em thực hiện các nửa với bằng các block XV1 và XV2 nó hoàn toàn tương tự với việc dịch vòng 1 và dịch vòng 2.

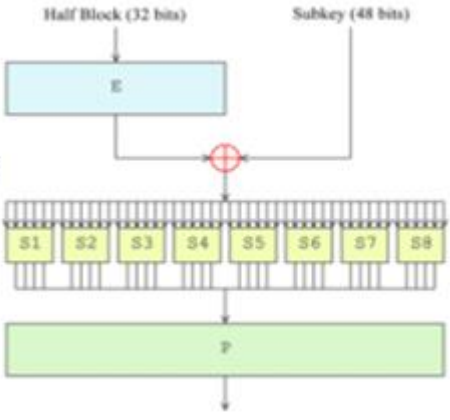
```
89      private final static int[][] XV1 ={{1,2,3,4,5,6,7},
90              {8,9,10,11,12,13,14},
91              {15,16,17,18,19,20,21},
92              {22,23,24,25,26,27,0},
93      };
94      private final static int[][] XV2 ={{2,3,4,5,6,7,8},
95              {9,10,11,12,13,14,15},
96              {16,17,18,19,20,21,22},
97              {23,24,25,26,27,0,1}
98      };
```

2.4. Mã hóa.



- Dựa trên sơ đồ mã hóa, thì ta cần thực hiện hàm F trong mỗi vòng lặp.

Hàm F có dạng



- Hàm F: Bao gồm việc hoán vị bằng block E, thực hiện phép xor với khóa, thay đổi từ 6 bit thành 4 bit qua các Sbox và cuối cùng là hoán vị qua PBox.

```
346     public int[][] Fiestel(int [][] inputStateBottom, int[][] key){
347
348         int [][] Ex = hoanVi(inputStateBottom,E);
349         int [][] F = xor(Ex,key);
350         int [][] Freturn = new int[8][4];
351         for (int j=0;j < 8;j++){
352             Freturn[j] = Sbox(F[j],j);
353         }
354         return hoanVi(Freturn,Pbox);
355     }
```

- Hàm Sbox: Thực hiện việc chuyển một chuỗi 6 bit thành 4 bit qua Sbox.

```
339     public  int[] Sbox(int [] unKnow, int i ){
340         int x = unKnow[5] + unKnow[0]*2;
341         int y = unKnow[4] + unKnow[3]*2 + unKnow[2]* 4 + unKnow[1]*8;
342         int hex = listSbox.get(i)[x][y];
343         return toBinary(hex, n: 4);
344     }
```

  - Với i là Sbox thứ i (từ 1 đến 8), unknown là một mảng 6 kí tự bit cần chuyển sang dạng 4 bit.

- Hàm thực hiện mã hóa:

```
246     public int [][] EncodeState(int i){
247         int[][] halfTop = new int[4][8];
248         int[][] halfBottom = new int[4][8];
249         divInput(liststateIP.get(i),halfTop,halfBottom);
250         for (int j = 0 ; j < 16 ; j ++){
251             int [][] left = Fiestel(halfBottom, listKey.get(j));
252             int [][] right = xor(halfTop, left);
253             if (j==15) {
254                 halfTop = right;
255                 break;
256             }
257             halfTop = halfBottom;
258             halfBottom = right;
259         }
260         int [][] stateEncode = new int[8][8];
261         stateEncode[0] = halfTop[0];
262         stateEncode[1] = halfTop[1];
263         stateEncode[2] = halfTop[2];
264         stateEncode[3] = halfTop[3];
265         stateEncode[4] = halfBottom[0];
266         stateEncode[5] = halfBottom[1];
267         stateEncode[6] = halfBottom[2];
268         stateEncode[7] = halfBottom[3];
269
270         return hoanVi(stateEncode,FP);
271     }
```

- Vì ban đầu chuỗi nhập vào được chia thành danh sách các block. Nên giá trị i đầu vào ở đây là block thứ i trong danh sách các block cần mã hóa.
- Đầu tiên khối block được chia thành 2 nửa.
- Thực hiện 16 vòng lặp. Trong mỗi vòng lặp, một nửa được thực hiện hàm F, sau đó được xor với nửa còn lại. Sau đó được tráo đổi cho nhau và thực hiện vòng tiếp theo.
- Sau 16 vòng, 2 nửa được ghép lại và được hoán vị qua khối hoán vị cuối FP.

2.6. Giải mã: Việc giải mã thực hiện hoàn toàn ngược lại so với việc mã hóa.

```
272     public int [][] DecodeState(int i){
273         int[][] halfTop = new int[4][8];
274         int[][] halfBottom = new int[4][8];
275         divInput(listStateForDecode.get(i),halfTop,halfBottom);
276         for (int j = 15 ; j >=0 ; j --){
277             int [][] left = Fiestel(halfBottom, listKey.get(j));
278             int [][] right = xor(halfTop, left);
279             if (j==0) {
280                 halfTop = right;
281                 break;
282             }
283             halfTop = halfBottom;
284             halfBottom = right;
285         }
286         int [][] stateDecode = new int[8][8];
287 
288         stateDecode[0] = halfTop[0];
289         stateDecode[1] = halfTop[1];
290         stateDecode[2] = halfTop[2];
291         stateDecode[3] = halfTop[3];
292         stateDecode[4] = halfBottom[0];
293         stateDecode[5] = halfBottom[1];
294         stateDecode[6] = halfBottom[2];
295         stateDecode[7] = halfBottom[3];
296 
297         return hoanVi(stateDecode,FP);
298     }
```

Em xin chân thành cảm ơn ạ.