

The method and process of checking MAC1, calculating MAC2, and checking TAC on the CPU card

原创

One line of Java ⌚ Posted on 2018-08-15 16:10:04 👁 Reading: 7.6k ⭐ Collection 16 👍 Likes 1

copyright

Category Column: CPU card Article Tags: CPU card MAC1 check CPU card MAC2 calculation CPU card TAC check

preface
MAC1 verification, MAC2 computing, and TAC verification processes
The result is as follows
The help class involved
DES tools
ByteUtil

preface

The **key** of the CPU card needs to be obtained through the encryption machine, so the test card of Fudan Microelectronics issued by myself is used here, and now it is clear that the corresponding recharge key and maintenance key are known

The specific process is directly on the code, and there are more detailed comments in it, and you can read it line by line according to the code

MAC1 verification, MAC2 computing, and TAC verification processes

```
1 import java.util.Locale;
2
3 public class CardCenter
4 {
5     public static void main(String[] args)
6     {
7
8         // 圈存的key
9         String loadKey = "3F013F013F013F013F013F013F013F01";
10        System.out.println("圈存的key:" + loadKey);
11        // 验证tac的key
12        String tacKey = "34343434343434343434343434343434";
13        System.out.println("验证tac的key:" + tacKey);
14
15        System.out.println();
16
17        // posid
18        String posid = "112233445566";
19        System.out.println("终端ID:" + posid);
20        // 交易金额
21        String tradeAmount = "00000001";
22        System.out.println("交易金额:" + tradeAmount);
23        // 交易金额十进制
24        int ta = 1;
25        // 交易类型
26        String tradeType = "02";
27        System.out.println("交易类型:" + tradeType);
28
29        System.out.println();
30        // 预充值指令
31        System.out.println("组装预充值指令:805000020b0100000001112233445566");
32        // 预消费指令805000020b0100000001112233445566的指令回复
33        String preTopup = "0000001a0017000106b825d7684c81ce9000";
34        System.out.println("得到预充值响应:" + preTopup);
35        byte[] recvByte = ByteUtil.hexStr2Byte(preTopup);
36
37        // 卡余额
38        String balance = ByteUtil.hexToStr(recvByte, 0, 4);
39        int bal = ByteUtil.hexToInt(recvByte, 0, 4);
40        System.out.println("卡余额:" + balance);
41
42        // 联机计数器
43        String cardCnt = ByteUtil.hexToStr(recvByte, 4, 2);
44        System.out.println("联机计数器:" + cardCnt);
45
46        // 密钥版本
47
```

```

..
48 String keyVersion = ByteUtil.hexToStr(recvByte, 6, 1);
49 System.out.println("密钥版本:" + keyVersion);
50
51 // 算法标识
52 String algIndMark = ByteUtil.hexToStr(recvByte, 7, 1);
53 System.out.println("算法标识:" + algIndMark);
54
55 // 随机数
56 String random = ByteUtil.hexToStr(recvByte, 8, 4);
57 System.out.println("随机数:" + random);
58
59 // mac1
60 String mac1 = ByteUtil.hexToStr(recvByte, 12, 4);
61 System.out.println("mac1:" + mac1);
62
63 System.out.println("");
64 System.out.println("开始验证mac1");
65 // 验证mac1的正确性
66 // 输入的数据为: 随机数+联机计数器+"8000"
67 String inputData = random + cardCnt + "8000";
68 System.out.println("计算过程密钥数据:" + inputData);
69 // 计算过程密钥
70 String sessionKey = Des.getHintKey(inputData, loadKey);
71 System.out.println("过程密钥:" + sessionKey);
72
73 // 计算mac1需要输入的数据
74 // 输入的数据为: 余额+交易金额+交易类型+终端编号
75 inputData = balance + tradeAmount + tradeType + posid;
76 String legitMac1 = Des.PBOC_DES_MAC(sessionKey, "0000000000000000", inputData, 0).substring(0, 8);
77 System.out.println("标准的mac1数据:" + legitMac1);
78
79 if (!mac1.toUpperCase(Locale.getDefault()).equals(legitMac1))
80 {
81     System.out.println("mac1校验失败! ");
82     return;
83 }
84 System.out.println("mac1校验成功! ");
85
86 // 开始计算Mac2用于做充值确认操作
87 System.out.println("");
88 System.out.println("开始计算mac2");
89 // 交易日期
90 String tradeDate = "20170310";
91 // 交易时间
92 String tradeTime = "110734";
93 // 计算mac2的输入数据
94 // 输入数据为: 交易金额+交易类型+终端编号+交易时间+交易日期
95 inputData = tradeAmount + tradeType + posid + tradeDate + tradeTime;
96 String mac2 = Des.PBOC_DES_MAC(sessionKey, "0000000000000000", inputData, 0).substring(0, 8);
97 // 得到mac2
98 System.out.println("计算后的mac2:" + mac2);
99
100 // 组装写卡指令
101 System.out.println("805200000b" + tradeDate + tradeTime + mac2);
102 System.out.println("开始向卡片发送充值确认的指令");
103 // 向卡发送指令
104 System.out.println("apdu:805200000b201703101107349C8A0625");
105 // 响应tac
106 System.out.println("recv:16ead5169000");
107 String tac = "16ead516";
108 System.out.println("得到Tac:" + tac);
109
110 System.out.println("");
111 System.out.println("开始验证tac");
112 System.out.println("tac验证密钥:" + tacKey);
113 // 对tac验证密钥左边8个字节和右边8个字节做异或处理得到tac过程密钥
114 String tacTessionKey = Des.xOr(tacKey.substring(0, 16), tacKey.substring(16, 32));
115 System.out.println("tac过程密钥:" + tacTessionKey);
116 // 充值成功之后新的金额为00000043+00000001=00000044
117 String newBalance = ByteUtil.hexToStr(ByteUtil.intToHex(bal + ta, 4));
118 // 计算标准的tac的输入数据
119 // 输入的数据: 新的余额+旧的联机计数器+交易金额+交易类型+终端编号+交易日期+交易时间
120 inputData = newBalance + cardCnt + tradeAmount + tradeType + posid + tradeDate + tradeTime;
121 String legitTac = Des.PBOC_DES_MAC(tacTessionKey, "0000000000000000", inputData, 0).substring(0, 8);

```

```

122         System.out.println("标准的tac数据:" + legitTac);
123
124         if (!tac.toUpperCase(Locale.getDefault()).equals(legitTac))
125         {
126             System.out.println("tac校验错误! ");
127             return;
128         }
129         System.out.println("tac校验成功! ");
130     }
}

```

The result is as follows

```

1  圈存的key:3F013F013F013F013F013F013F01
2  验证tac的key:3434343434343434343434343434
3
4  终端ID:112233445566
5  交易金额:00000001
6  交易类型:02
7
8  组装预充值指令:805000020b0100000001112233445566
9  得到预充值响应:0000001a0017000106b825d7684c81ce9000
10 卡余额:0000001a
11 联机计数器:0017
12 密钥版本:00
13 算法标识:01
14 随机数:06b825d7
15 mac1:684c81ce
16
17 开始验证mac1
18 计算过程密钥数据:06b825d700178000
19 过程密钥:6E76E8541E217D9A
20 D[0]=0000001a00000001
21 D[1]=0211223344556680
22 1*****
23 I=0000001A00000001
24 O=D42EE2FF647C9F00
25 I=D63FC0CC2029F980
26 I=684C81CE47609B5B
27 标准的mac1数据:684C81CE
28 mac1校验成功!
29
30 开始计算mac2
31 D[0]=0000000102112233
32 D[1]=4455662017031011
33 D[2]=0734800000000000
34 1*****
35 I=0000000102112233
36 O=A0D9EACF4A0A833D
37 I=E48C8CEF5D09932C
38 2*****
39 I=E48C8CEF5D09932C
40 O=6DF657FD31FBAB05
41 I=6AC2D7FD31FBAB05
42 I=9C8A06256ACBF60A
43 计算后的mac2:9C8A0625
44 80520000b201703101107349C8A0625
45 开始向卡片发送充值确认的指令
46 apdu:80520000b201703101107349C8A0625
47 recv:16ead5169000
48 得到Tac:16ead516
49
50 开始验证tac
51 tac验证密钥:3434343434343434343434343434
52 tac过程密钥:0000000000000000
53 D[0]=0000001b00170000
54 D[1]=0001021122334455
55 D[2]=6620170310110734
56 D[3]=8000000000000000
57 1*****
58 I=0000001B00170000
59 O=F0DE01AF2D71051B
60 I=F0DF03BE0F42414E
61 2*****

```

```
62 I=F0DF03BE0F42414E
63 O=044D5B737F9A0282
64 I=626D4C706F8B05B6
65 3*****
66 I=626D4C706F8B05B6
67 O=2B5ED4A6C483D0BF
68 I=AB5ED4A6C483D0BF
69 I=16EAD5165389360A
70 标准的tac数据:16EAD516
71 tac校验成功!
72
```

涉及的帮助类

DES工具类

```
1 import java.io.ByteArrayOutputStream;
2 import java.io.UnsupportedEncodingException;
3
4 import javax.crypto.Cipher;
5 import javax.crypto.SecretKey;
6 import javax.crypto.spec.SecretKeySpec;
7
8 /**
9  * DES工具类
10  */
11 public class Des
12 {
13     /**
14      * *****压缩替换S-Box*****
15      * *****
16      */
17     private static final int[][] s1 =
18     {
19         { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
20         { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
21         { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
22         { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };
23     private static final int[][] s2 =
24     {
25         { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
26         { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
27         { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
28         { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };
29     private static final int[][] s3 =
30     {
31         { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
32         { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
33         { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
34         { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
35     private static final int[][] s4 =
36     {
37         { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
38         { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },// errorr
39         { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
40         { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
41     private static final int[][] s5 =
42     {
43         { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
44         { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
45         { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
46         { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };
47     private static final int[][] s6 =
48     {
49         { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
50         { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
51         { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
52         { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
53     private static final int[][] s7 =
54     {
55         { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
56         { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
57         { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
58         { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };
59
```

```

60 private static final int[][] s8 =
61 {
62 { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
63 { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
64 { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
65 { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };
66 private static final int[] ip =
67 { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25,
68 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 };
69 private static final int[] _ip =
70 { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52,
71 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25 };
72 // 每次密钥循环左移位数
73 private static final int[] LS =
74 { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
75 private static int[][] subKey = new int[16][48];
76 public static int HEX = 0;
77 public static int ASC = 1;
78
79 /**
80  * 将十六进制A-F转换成对应数
81  *
82  * @param ch
83  * @return
84  * @throws Exception
85  */
86 public static int getIntByChar(char ch) throws Exception
87 {
88     char t = Character.toUpperCase(ch);
89     int i = 0;
90     switch (t)
91     {
92         case '0':
93         case '1':
94         case '2':
95         case '3':
96         case '4':
97         case '5':
98         case '6':
99         case '7':
100        case '8':
101        case '9':
102            i = Integer.parseInt(Character.toString(t));
103            break;
104        case 'A':
105            i = 10;
106            break;
107        case 'B':
108            i = 11;
109            break;
110        case 'C':
111            i = 12;
112            break;
113        case 'D':
114            i = 13;
115            break;
116        case 'E':
117            i = 14;
118            break;
119        case 'F':
120            i = 15;
121            break;
122        default:
123            throw new Exception("getIntByChar was wrong");
124    }
125    return i;
126 }
127
128 /**
129  * 将字符串转换成二进制数组
130  *
131  * @param source
132  *      : 16字节
133  * @return

```

```

134     */
135     public static int[] string2Binary(String source)
136     {
137         int len = source.length();
138         int[] dest = new int[len * 4];
139         char[] arr = source.toCharArray();
140         for (int i = 0; i < len; i++)
141         {
142             int t = 0;
143             try
144             {
145                 t = getIntByChar(arr[i]);
146             }
147             catch (Exception e)
148             {
149                 e.printStackTrace();
150             }
151             String[] str = Integer.toBinaryString(t).split("");
152             int k = i * 4 + 3;
153             for (int j = str.length - 1; j > 0; j--)
154             {
155                 dest[k] = Integer.parseInt(str[j]);
156                 k--;
157             }
158         }
159         return dest;
160     }
161
162     /**
163     * 返回x的y次方
164     *
165     * @param x
166     * @param y
167     * @return
168     */
169     public static int getXY(int x, int y)
170     {
171         int temp = x;
172         if (y == 0)
173             x = 1;
174         for (int i = 2; i <= y; i++)
175         {
176             x *= temp;
177         }
178         return x;
179     }
180
181     /**
182     * s位长度的二进制字符串
183     *
184     * @param s
185     * @return
186     */
187     public static String binary2Hex(String s)
188     {
189         int len = s.length();
190         int result = 0;
191         int k = 0;
192         if (len > 4)
193             return null;
194         for (int i = len; i > 0; i--)
195         {
196             result += Integer.parseInt(s.substring(i - 1, i)) * getXY(2, k);
197             k++;
198         }
199         switch (result)
200         {
201             case 0:
202             case 1:
203             case 2:
204             case 3:
205             case 4:
206             case 5:
207             case 6:

```

```

200         case 7:
209         case 8:
210         case 9:
211             return "" + result;
212         case 10:
213             return "A";
214         case 11:
215             return "B";
216         case 12:
217             return "C";
218         case 13:
219             return "D";
220         case 14:
221             return "E";
222         case 15:
223             return "F";
224         default:
225             return null;
226     }
227 }
228
229 /**
230  * 将int转换成Hex
231  *
232  * @param i
233  * @return
234  * @throws Exception
235  */
236 public static String int2Hex(int i)
237 {
238     switch (i)
239     {
240         case 0:
241         case 1:
242         case 2:
243         case 3:
244         case 4:
245         case 5:
246         case 6:
247         case 7:
248         case 8:
249         case 9:
250             return "" + i;
251         case 10:
252             return "A";
253         case 11:
254             return "B";
255         case 12:
256             return "C";
257         case 13:
258             return "D";
259         case 14:
260             return "E";
261         case 15:
262             return "F";
263         default:
264             return null;
265     }
266 }
267
268 /**
269  * 将二进制字符串转换成十六进制字符
270  *
271  * @param s
272  * @return
273  */
274 public static String binary2ASC(String s)
275 {
276     String str = "";
277     int ii = 0;
278     int len = s.length();
279     // 不够4bit左补0
280     if (len % 4 != 0)
281     {
282

```

```

283         while (ii < 4 - len % 4)
284         {
285             s = "0" + s;
286         }
287     }
288     for (int i = 0; i < len / 4; i++)
289     {
290         str += binary2Hex(s.substring(i * 4, i * 4 + 4));
291     }
292     return str;
293 }
294
295 /**
296  * IP初始置换
297  *
298  * @param source
299  * @return
300  */
301 public static int[] changeIP(int[] source)
302 {
303     int[] dest = new int[64];
304     for (int i = 0; i < 64; i++)
305     {
306         dest[i] = source[ip[i] - 1];
307     }
308     return dest;
309 }
310
311 /**
312  * IP-1逆置
313  *
314  * @param source
315  * @return
316  */
317 public static int[] changeInverseIP(int[] source)
318 {
319     int[] dest = new int[64];
320     for (int i = 0; i < 64; i++)
321     {
322         dest[i] = source[_ip[i] - 1];
323     }
324     return dest;
325 }
326
327 /**
328  * 2bit扩展8bit
329  *
330  * @param source
331  * @return
332  */
333 public static int[] expend(int[] source)
334 {
335     int[] ret = new int[48];
336     int[] temp =
337     { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25,
338       26, 27, 28, 29, 28, 29, 30, 31, 32, 1 };
339     for (int i = 0; i < 48; i++)
340     {
341         ret[i] = source[temp[i] - 1];
342     }
343     return ret;
344 }
345
346 /**
347  * 8bit压缩2bit
348  *
349  * @param source
350  *      (48bit)
351  * @return R(32bit) B=E(R)⊕K, 将48 位的B 分成8 个分组, B=B1B2B3B4B5B6B7B8
352  */
353 public static int[] press(int[] source)
354 {
355     int[] ret = new int[32];
356     int[][] temp = new int[8][6];

```



```

357     int[][][] s =
358     { s1, s2, s3, s4, s5, s6, s7, s8 };
359     StringBuffer str = new StringBuffer();
360     for (int i = 0; i < 8; i++)
361     {
362         for (int j = 0; j < 6; j++)
363         {
364             temp[i][j] = source[i * 6 + j];
365         }
366     }
367     for (int i = 0; i < 8; i++)
368     {
369         // (16)
370         int x = temp[i][0] * 2 + temp[i][5];
371         // (2345)
372         int y = temp[i][1] * 8 + temp[i][2] * 4 + temp[i][3] * 2 + temp[i][4];
373         int val = s[i][x][y];
374         String ch = int2Hex(val);
375         // System.out.println("x=" + x + ",y=" + y + "-->" + ch);
376         // String ch = Integer.toString(val);
377         str.append(ch);
378     }
379     // System.out.println(str.toString());
380     ret = string2Binary(str.toString());
381     // printArr(ret);
382     // 置换P
383     ret = dataP(ret);
384     return ret;
385 }
386
387 /**
388  * 置换P(32bit)
389  *
390  * @param source
391  * @return
392  */
393 public static int[] dataP(int[] source)
394 {
395     int[] dest = new int[32];
396     int[] temp =
397     { 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };
398     int len = source.length;
399     for (int i = 0; i < len; i++)
400     {
401         dest[i] = source[temp[i] - 1];
402     }
403     return dest;
404 }
405
406 /**
407  * @param R
408  *      (2bit)
409  * @param K
410  *      (48bit的轮子密
411  * @return 32bit
412  */
413 public static int[] f(int[] R, int[] K)
414 {
415     int[] dest = new int[32];
416     int[] temp = new int[48];
417     // 先将输入32bit扩展8bit
418     int[] expendR = expend(R); // 48bit
419     // 与轮子密钥进行异或运
420     temp = diffOr(expendR, K);
421     // 压缩2bit
422     dest = press(temp);
423     // printArr(temp);
424     return dest;
425 }
426
427 /**
428  * 两个等长的数组做异或
429  *
430  * @param source1

```

```

431     * @param source2
432     * @return
433     */
434     public static int[] diffOr(int[] source1, int[] source2)
435     {
436         int len = source1.length;
437         int[] dest = new int[len];
438         for (int i = 0; i < len; i++)
439         {
440             dest[i] = source1[i] ^ source2[i];
441         }
442         return dest;
443     }
444
445     /**
446     * DES加密-->对称密钥 D = Ln(32bit)+Rn(32bit) 经过16轮置
447     *
448     * @param D
449     *         (16byte)明文
450     * @param K
451     *         (16byte)轮子密钥
452     * @return (16byte)密文
453     */
454     public static String encryption(String D, String K)
455     {
456         String str = "";
457         int[] temp = new int[64];
458         int[] data = string2Binary(D);
459         // printArr(data);
460         // 第一步初始置
461         data = changeIP(data);
462         // printArr(data);
463         int[][] left = new int[17][32];
464         int[][] right = new int[17][32];
465         for (int j = 0; j < 32; j++)
466         {
467             left[0][j] = data[j];
468             right[0][j] = data[j + 32];
469         }
470         // printArr(left[0]);
471         // printArr(right[0]);
472         setKey(K); // sub key ok
473         for (int i = 1; i < 17; i++)
474         {
475             // 获取(48bit)的轮子密
476             int[] key = subKey[i - 1];
477             // L1 = R0
478             left[i] = right[i - 1];
479             // R1 = L0 ^ f(R0, K1)
480             int[] fTemp = f(right[i - 1], key); // 32bit
481             right[i] = diffOr(left[i - 1], fTemp);
482         }
483         // 组合的时候, 左右调换*****
484         for (int i = 0; i < 32; i++)
485         {
486             temp[i] = right[16][i];
487             temp[32 + i] = left[16][i];
488         }
489         temp = changeInverseIP(temp);
490         str = binary2ASC(intArr2Str(temp));
491         return str;
492     }
493
494     /**
495     * DES解密-->对称密钥 解密算法与加密算法基本相同, 不同之处在于轮子密钥的使用顺序逆序, 即解密的第1 轮子密钥为加密的6 轮子密钥, 解密的
496     * 轮子密钥为加密的5 轮子密钥, ..., 解密的第16 轮子密钥为加密的 轮子密钥
497     *
498     * @param source
499     *         密文
500     * @param key
501     *         密钥
502     * @return
503     */
504     public static String discription(String source, String key)
505     {
506

```

```

505 {
506     String str = "";
507     int[] data = string2Binary(source); // 64bit
508     // 第一步初始置
509     data = changeIP(data);
510     int[] left = new int[32];
511     int[] right = new int[32];
512     int[] tmp = new int[32];
513     for (int j = 0; j < 32; j++)
514     {
515         left[j] = data[j];
516         right[j] = data[j + 32];
517     }
518     setKey(key); // sub key ok
519     for (int i = 16; i > 0; i--)
520     {
521         // 获取(48bit)的轮子密
522         /** *****不同之处***** */
523         int[] sKey = subKey[i - 1];
524         tmp = left;
525         // R1 = L0
526         left = right;
527         // L1 = R0 ^ f(L0, K1)
528         int[] fTemp = f(right, sKey); // 32bit
529         right = diffOr(tmp, fTemp);
530     }
531     // 组合的时候, 左右调换*****
532     for (int i = 0; i < 32; i++)
533     {
534         data[i] = right[i];
535         data[32 + i] = left[i];
536     }
537     data = changeInverseIP(data);
538     for (int i = 0; i < data.length; i++)
539     {
540         str += data[i];
541     }
542     str = binary2ASC(str);
543     return str;
544 }
545
546 /**
547  * 单长密钥DES(16byte)
548  *
549  * @param source
550  * @param key
551  * @param type
552  *         0:encrypt 1:discript
553  * @return
554  */
555 public static String DES_1(String source, String key, int type)
556 {
557     if (source.length() != 16 || key.length() != 16)
558         return null;
559     if (type == 0)
560     {
561         return encryption(source, key);
562     }
563     if (type == 1)
564     {
565         return discription(source, key);
566     }
567     return null;
568 }
569
570 /**
571  * @param source
572  * @param key
573  * @param type
574  *         0:encrypt 1:discript
575  * @return
576  */
577 public static String DES_2(String source, String key, int type)
578 {
579

```

```

580         return null;
581     }
582
583     /**
584      * 三重DES算法(双长密32byte)) 密钥K1和K2 1、先用K1加密明文 2、接K2对上的结果进行解 3、然后用K1对上的结果进行加
585      *
586      * @param source
587      * @param key
588      * @param type
589      *      0:encrypt 1:decrypt
590      * @return
591      */
592     public static String DES_3(String source, String key, int type)
593     {
594         if (key.length() != 32 || source.length() != 16)
595             return null;
596         String temp = null;
597         String K1 = key.substring(0, key.length() / 2);
598         String K2 = key.substring(key.length() / 2);
599         System.out.println("K1-->" + K1);
600         System.out.println("K2-->" + K2);
601         if (type == 0)
602         {
603             temp = encryption(source, K1);
604             System.out.println("step1-->" + temp);
605             temp = decryption(temp, K2);
606             System.out.println("step2-->" + temp);
607             return encryption(temp, K1);
608         }
609         if (type == 1)
610         {
611             temp = decryption(source, K1);
612             temp = encryption(temp, K2);
613             return decryption(temp, K1);
614         }
615         return null;
616     }
617
618     /**
619      * *****48bit的轮子密钥的生成*****
620      * *****
621      */
622     /**
623      * 4bit的密钥转换成56bit
624      *
625      * @param source
626      * @return
627      */
628     public static int[] keyPC_1(int[] source)
629     {
630         int[] dest = new int[56];
631         int[] temp =
632         { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7,
633           62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4 };
634         for (int i = 0; i < 56; i++)
635         {
636             dest[i] = source[temp[i] - 1];
637         }
638         return dest;
639     }
640
641     /**
642      * 将密钥循环左移i
643      *
644      * @param source
645      *      二进制密钥数
646      * @param i
647      *      循环左移位数
648      * @return
649      */
650     public static int[] keyLeftMove(int[] source, int i)
651     {
652         int temp = 0;
653         int len = source.length;

```

```

654     int ls = LS[i];
655     for (int k = 0; k < ls; k++)
656     {
657         temp = source[0];
658         for (int j = 0; j < len - 1; j++)
659         {
660             source[j] = source[j + 1];
661         }
662         source[len - 1] = temp;
663     }
664     return source;
665 }
666
667 /**
668  * 6bit的密钥转换成48bit
669  *
670  * @param source
671  * @return
672  */
673 public static int[] keyPC_2(int[] source)
674 {
675     int[] dest = new int[48];
676     int[] temp =
677     { 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44,
678         49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32 };
679     for (int i = 0; i < 48; i++)
680     {
681         dest[i] = source[temp[i] - 1];
682     }
683     return dest;
684 }
685
686 /**
687  * 获取轮子密钥(48bit)
688  *
689  * @param source
690  * @return
691  */
692 public static void setKey(String source)
693 {
694     if (subKey.length > 0)
695         subKey = new int[16][48];
696     // 装换4bit
697     int[] temp = string2Binary(source);
698     // 6bit均分成两部分
699     int[] left = new int[28];
700     int[] right = new int[28];
701     // 经过PC-14bit转换6bit
702     int[] temp1 = new int[56];
703     temp1 = keyPC_1(temp);
704     // printArr(temp1);
705     // 将经过转换的temp1均分成两部分
706     for (int i = 0; i < 28; i++)
707     {
708         left[i] = temp1[i];
709         right[i] = temp1[i + 28];
710     }
711     // 经过16次循环左移，然后PC-2置换
712     for (int i = 0; i < 16; i++)
713     {
714         left = keyLeftMove(left, LS[i]);
715         right = keyLeftMove(right, LS[i]);
716         for (int j = 0; j < 28; j++)
717         {
718             temp1[j] = left[j];
719             temp1[j + 28] = right[j];
720         }
721         // printArr(temp1);
722         subKey[i] = keyPC_2(temp1);
723     }
724 }
725
726 public static void printArr(int[] source)
727 {
728     ---

```

```

728         int len = source.length;
729         for (int i = 0; i < len; i++)
730         {
731             System.out.print(source[i]);
732         }
733         System.out.println();
734     }
735
736     /**
737      * 将ASC字符串转16进制字符
738      *
739      * @param asc
740      * @return
741      */
742     public static String ASC_2_HEX(String asc)
743     {
744         StringBuffer hex = new StringBuffer();
745         try
746         {
747             byte[] bs = asc.toUpperCase().getBytes("UTF-8");
748             for (byte b : bs)
749             {
750                 hex.append(Integer.toHexString(new Byte(b).intValue()));
751             }
752         }
753         catch (UnsupportedEncodingException e)
754         {
755             e.printStackTrace();
756         }
757         return hex.toString();
758     }
759
760     /**
761      * 16进制的字符串转换成ASC的字符串
762      * 16进制的字符串压缩成BCD(30313233343536373839414243444546)-->(0123456789ABCDEF)
763      *
764      * @param hex
765      * @return
766      */
767     public static String HEX_2_ASC(String hex)
768     {
769         String asc = null;
770         int len = hex.length();
771         byte[] bs = new byte[len / 2];
772         for (int i = 0; i < len / 2; i++)
773         {
774             bs[i] = Byte.parseByte(hex.substring(i * 2, i * 2 + 2), 16);
775         }
776         try
777         {
778             asc = new String(bs, "UTF-8");
779         }
780         catch (UnsupportedEncodingException e)
781         {
782             e.printStackTrace();
783         }
784         return asc;
785     }
786
787     /**
788      * 计算MAC(hex) ANSI-X9.9-MAC(16的整数不补) PBOC-DES-MAC(16的整数补8000000000000000)
789      * 使用单倍长密钥DES算法
790      *
791      * @param key
792      *         密钥 (16byte)
793      * @param vector
794      *         初始向量0000000000000000
795      * @param data
796      *         数据
797      * @return mac
798      */
799     public static String PBOC_DES_MAC(String key, String vector, String data, int type)
800     {
801         if (key.length() != 16)
802

```

```

803     {
804         return null;
805     }
806     if (type == ASC)
807     {
808         data = ASC_2_HEX(data);
809     }
810     int len = data.length();
811     int arrLen = len / 16 + 1;
812     String[] D = new String[arrLen];
813     if (vector == null)
814         vector = "0000000000000000";
815     if (len % 16 == 0)
816     {
817         data += "8000000000000000";
818     }
819     else
820     {
821         data += "80";
822         for (int i = 0; i < 15 - len % 16; i++)
823         {
824             data += "00";
825         }
826     }
827     for (int i = 0; i < arrLen; i++)
828     {
829         D[i] = data.substring(i * 16, i * 16 + 16);
830         System.out.println("D[" + i + "]=" + D[i]);
831     }
832     // D0 Xor Vector
833     String I = xOr(D[0], vector);
834     String O = null;
835     for (int i = 1; i < arrLen; i++)
836     {
837         System.out.println(i + "*****");
838         System.out.println("I=" + I);
839         O = DES_1(I, key, 0);
840         System.out.println("O=" + O);
841         I = xOr(D[i], O);
842         System.out.println("I=" + I);
843     }
844     I = DES_1(I, key, 0);
845     System.out.println("I=" + I);
846     return I;
847 }
848
849 /**
850  * 计算MAC(hex) PBOC_3DES_MAC(16的整数补8000000000000000) 前n-1组使用单长密钥DES
851  * CBC算法（使用密钥是密钥的左8字节） 最后1组使用双长密钥3DES CBC算法（使用全部16字节密钥）
852  *
853  * @param key
854  *         密钥 (32byte)
855  * @param vector
856  *         初始向量0000000000000000
857  * @param data
858  *         数据
859  * @return mac
860  */
861 public static String PBOC_3DES_MAC(String key, String vector, String data, int type)
862 {
863     if (key.length() != 32)
864     {
865         return null;
866     }
867     if (type == ASC)
868     {
869         data = ASC_2_HEX(data);
870     }
871     int len = data.length();
872     int arrLen = len / 16 + 1;
873     String[] D = new String[arrLen];
874     if (vector == null)
875         vector = "0000000000000000";
876     if (len % 16 == 0)

```

```

877     {
878         data += "8000000000000000";
879     }
880     else
881     {
882         data += "80";
883         for (int i = 0; i < 15 - len % 16; i++)
884         {
885             data += "00";
886         }
887     }
888     for (int i = 0; i < arrLen; i++)
889     {
890         D[i] = data.substring(i * 16, i * 16 + 16);
891         System.out.println("D[" + i + "]=" + D[i]);
892     }
893     // D0 Xor Vector
894     String I = xOr(D[0], vector);
895     String O = null;
896     String k1 = key.substring(0, 16);
897     System.out.println("KL8:" + k1);
898     for (int i = 1; i < arrLen; i++)
899     {
900         System.out.println(i + "*****");
901         System.out.println("I=" + I);
902         O = DES_1(I, k1, 0);
903         System.out.println("O=" + O);
904         I = xOr(D[i], O);
905         System.out.println("I=" + I);
906     }
907     I = DES_3(I, key, 0);
908     return I;
909 }
910
911 /**
912  * 将s1和s2做异或, 然后返回
913  *
914  * @param s1
915  * @param s2
916  * @return
917  */
918 public static String xOr(String s1, String s2)
919 {
920     int[] iArr = diffOr(string2Binary(s1), string2Binary(s2));
921     return binary2ASC(intArr2Str(iArr));
922 }
923
924 /**
925  * 将int类型数组拼接成字符串
926  *
927  * @param arr
928  * @return
929  */
930 public static String intArr2Str(int[] arr)
931 {
932     StringBuffer sb = new StringBuffer();
933     for (int i = 0; i < arr.length; i++)
934     {
935         sb.append(arr[i]);
936     }
937     return sb.toString();
938 }
939
940 /**
941  * 将data分散
942  *
943  * @param data
944  *         数据8字节 16个长度
945  * @param key
946  *         密钥
947  * @param type
948  * @return
949  */
950 public static String divData(String data, String key, int type)

```



```

951 {
952     String left = null;
953     String right = null;
954     if (type == HEX)
955     {
956         left = key.substring(0, 16);
957         right = key.substring(16, 32);
958     }
959     if (type == ASC)
960     {
961         left = ASC_2_HEX(key.substring(0, 8));
962         right = ASC_2_HEX(key.substring(8, 16));
963     }
964     // 加密
965     data = DES_1(data, left, 0);
966     // 解密
967     data = DES_1(data, right, 1);
968     // 加密
969     data = DES_1(data, left, 0);
970     return data;
971 }
972
973 /**
974  * 取反(10001)--->(01110)
975  *
976  * @param source
977  *      数据源
978  * @return
979  */
980 public static String reverse(String source)
981 {
982     int[] data = string2Binary(source);
983     int j = 0;
984     for (int i : data)
985     {
986         data[j++] = 1 - i;
987     }
988     return binary2ASC(intArr2Str(data));
989 }
990
991 /**
992  * 主密钥需要经过两次分散获得IC卡中的子密钥 空圈的通讯类过程密钥使用这种密钥分散机制
993  *
994  * @param issuerFlag
995  *      发卡方标识符
996  * @param appNo
997  *      应用序列号即卡号
998  * @param mpk
999  *      主密钥
1000  * @return
1001  */
1002 public static String getDPK(String issuerFlag, String appNo, String mpk)
1003 {
1004     // 第一次分散
1005     StringBuffer issuerMPK = new StringBuffer();
1006     // 获取Issuer MPK左半边
1007     issuerMPK.append(divData(issuerFlag, mpk, 0));
1008     // 获取Issuer MPK右半边
1009     issuerMPK.append(divData(reverse(issuerFlag), mpk, 0));
1010     // 第二次分散
1011     StringBuffer dpk = new StringBuffer();
1012     // 获取DPK左半边
1013     dpk.append(divData(appNo, issuerMPK.toString(), 0));
1014     // 获取DPK右半边
1015     dpk.append(divData(reverse(appNo), issuerMPK.toString(), 0));
1016     return dpk.toString();
1017 }
1018
1019 /**
1020  * 主密钥需要经过一次分散获得的子密钥 空圈的交易类过程密钥使用这种密钥分散机制
1021  *
1022  * @param mpk
1023  *      主密钥
1024  * @return

```

```

1025 */
1026 public static String getDPK4Once(String data, String mpk)
1027 {
1028     // 第一次分散
1029     StringBuffer dpk = new StringBuffer();
1030     // 获取DPK左半边
1031     dpk.append(divData(data, mpk, 0));
1032     // 获取DPK右半边
1033     dpk.append(divData(reverse(data), mpk.toString(), 0));
1034     return dpk.toString();
1035 }
1036
1037 /**
1038  * @Title: getHintKey 获取过程密钥
1039  * @Description: TODO
1040  * @param: @param data 加密数据
1041  * @param: @param key 密钥
1042  * @param: @return
1043  * @return: String
1044  * @throws:
1045  */
1046 public static String getHintKey(String data, String key)
1047 {
1048     if (null == data || data.length() != 16)
1049         return null;
1050     if (null != key && key.length() > 16)
1051         key = key.substring(0, 16);
1052     if (null == key || key.length() != 16)
1053         return null;
1054     // 加密数据
1055     return encryption(data, key);
1056 }
1057
1058 // 字符串转hex码
1059 public static byte[] hexStr2Byte(String hex)
1060 {
1061     int len = (hex.length() / 2);
1062     byte[] result = new byte[len];
1063     char[] achar = hex.toCharArray();
1064     for (int i = 0; i < len; i++)
1065     {
1066         int pos = i * 2;
1067         result[i] = (byte) (toByte(achar[pos]) << 4 | toByte(achar[pos + 1]));
1068     }
1069     return result;
1070 }
1071
1072 private static byte toByte(char c)
1073 {
1074     return mask[c];
1075 }
1076
1077 static byte[] mask = new byte[128];
1078 private static final String Algorithm = "DESede"; // 定义 加密算法,可用
1079
1080 // DES,DESede,Blowfish
1081 // keybyte为加密密钥, 长度为24字节
1082 // src为被加密的数据缓冲区(源)
1083 public static byte[] encryptMode(byte[] keybyte, byte[] src)
1084 {
1085     try
1086     {
1087         // 生成密钥
1088         SecretKey deskey = new SecretKeySpec(keybyte, Algorithm);
1089         // 从原始密钥数据创建DESKeySpec对象
1090         // DESedeKeySpec dks = new DESKeySpec(key);
1091         // 加密
1092         Cipher c1 = Cipher.getInstance(Algorithm);
1093         c1.init(Cipher.ENCRYPT_MODE, deskey);
1094         return c1.doFinal(src);
1095     }
1096     catch (java.security.NoSuchAlgorithmException e1)
1097     {
1098         e1.printStackTrace();
1099     }

```

```

1100     }
1101     catch (javax.crypto.NoSuchPaddingException e2)
1102     {
1103         e2.printStackTrace();
1104     }
1105     catch (Exception e3)
1106     {
1107         e3.printStackTrace();
1108     }
1109     return null;
1110 }
1111
1112 public static String printHexString(String hint, byte[] b)
1113 {
1114     System.out.print(hint);
1115     String abc = "";
1116     for (int i = 0; i < b.length; i++)
1117     {
1118         String hex = Integer.toHexString(b[i] & 0xFF);
1119         if (hex.length() == 1)
1120         {
1121             hex = '0' + hex;
1122         }
1123         System.out.print(hex.toUpperCase() + " ");
1124         abc = abc + hex;
1125     }
1126     System.out.println("");
1127     return abc.toUpperCase();
1128 }
1129
1130 private static String hexString = "0123456789ABCDEFabcdef";
1131
1132 /**
1133  * 将string(包括中文)转为hex
1134  *
1135  * @param str
1136  * @return
1137  */
1138 public static String encode(String str)
1139 {
1140     byte[] bytes = str.getBytes();
1141     StringBuilder sb = new StringBuilder(bytes.length * 2);
1142     // 转换hex编码
1143     for (byte b : bytes)
1144     {
1145         sb.append(Integer.toHexString(b & 0xFF).substring(1));
1146     }
1147     str = sb.toString();
1148     return str;
1149 }
1150
1151 /**
1152  * 把hex编码转换为string
1153  *
1154  * @param bytes
1155  * @return
1156  */
1157 public static String decode(String bytes)
1158 {
1159     bytes = bytes.toUpperCase();
1160     ByteArrayOutputStream baos = new ByteArrayOutputStream(bytes.length() / 2);
1161     // 将每2位16进制整数组装成一个字节
1162     for (int i = 0; i < bytes.length(); i += 2)
1163     {
1164         baos.write((hexString.indexOf(bytes.charAt(i)) << 4 | hexString.indexOf(bytes.charAt(i + 1))));
1165     }
1166     return new String(baos.toByteArray());
1167 }

```

ByteUtil

```

1 public class ByteUtil
2 {
3
4     static byte[] mask = new byte[128];

```

```

5
6 static
7 {
8     initMask();
9 }
10 static byte[] asciiMask = new byte[] { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
11
12 private static void initMask()
13 { // init mask
14     for (int i = 0; i <= 9; i++)
15     {
16         mask[i + 48] = (byte) i;
17     }
18     for (int i = 0; i <= 5; i++)
19     {
20         mask[i + 97] = (byte) (10 + i);
21     }
22     for (int i = 0; i <= 5; i++)
23     {
24         mask[i + 65] = (byte) (10 + i);
25     }
26 }
27
28 public static int[] byte2Bitmap(int b)
29 {
30     int[] bitmap = new int[8];
31     for (int i = 0; i < 8; i++)
32     {
33         bitmap[i] = ((b >> (8 - i - 1)) & 0x01);
34     }
35     return bitmap;
36 }
37
38 public static byte[] intToBCD(int n, int balen)
39 {
40     byte[] ret = new byte[balen];
41     int tmp;
42     for (int i = 1; i <= balen; i++)
43     {
44         tmp = n % 100;
45         ret[balen - i] = (byte) (tmp / 10 * 16 + tmp % 10);
46
47         n -= tmp;
48         if (n == 0)
49         {
50             break;
51         }
52         n /= 100;
53     }
54     return ret;
55 }
56
57 public static int bcdToInt(byte[] ba, int idx, int len)
58 {
59     int jinwei = len * 2;
60     int ret = 0;
61     int temp = 0;
62     int pow;
63     int posNum; // 正数
64     for (int i = 0, n = len; i < n; i++)
65     {
66         pow = pow(10, (jinwei - 1));
67         posNum = ba[idx + i] >= 0 ? ba[idx + i] : ba[idx + i] + 256;
68         temp = (posNum / 16) * pow + posNum % 16 * pow / 10;
69         ret += temp;
70         jinwei -= 2;
71     }
72     return ret;
73 }
74
75 public static int pow(int x, int y)
76 {
77     int n = x;
78     for (int i = 1; i < y; i++)
79     --

```

```

79     {
80         n *= x;
81     }
82     return n;
83 }
84
85 public static byte[] hexStr2Byte(String hex)
86 {
87     int len = (hex.length() / 2);
88     byte[] result = new byte[len];
89     char[] achar = hex.toCharArray();
90     for (int i = 0; i < len; i++)
91     {
92         int pos = i * 2;
93         result[i] = (byte) (toByte(achar[pos]) << 4 | toByte(achar[pos + 1]));
94     }
95     return result;
96 }
97
98 public static int revAsciiHexToInt(byte[] ah)
99 {
100     int ret = 0;
101     for (int i = 0; i < ah.length / 2; i++)
102     {
103         int hex = (mask[ah[i * 2]] << 4) + (mask[ah[i * 2 + 1]]);
104         ret += (hex << (8 * i));
105     }
106     return ret;
107 }
108
109 public static int revHexToInt(byte[] data, int off, int len)
110 {
111     int ret = 0;
112     for (int i = 0; i < len; i++)
113     {
114         ret += (data[off + i] & 0xff) << (8 * i);
115     }
116
117     return ret;
118 }
119
120 public static int revAsciiHexToInt(byte[] ah, int off, int len)
121 {
122     int ret = 0;
123     for (int i = 0, n = len / 2; i < n; i++)
124     {
125         int hex = (mask[ah[i * 2 + off]] << 4) + (mask[ah[i * 2 + off + 1]]);
126         ret += (hex << (8 * i));
127     }
128     return ret;
129 }
130
131 public static byte[] intToRevAsciiHex(int value, int hexlen)
132 {
133     byte[] ret = new byte[hexlen * 2];
134
135     for (int i = 0; i < hexlen; i++)
136     {
137         if (value > 0)
138         {
139             ret[i * 2] = asciiMask[((value & 0xf0) >> 4)];
140             ret[i * 2 + 1] = asciiMask[(value & 0xf)];
141             value >>= 8;
142         }
143         else
144         {
145             ret[i * 2] = '0';
146             ret[i * 2 + 1] = '0';
147         }
148     }
149     return ret;
150 }
151
152 public static String intToRevHexString(int value, int hexlen)
153

```

```

---
154 {
155
156     byte[] ret = new byte[hexlen];
157     for (int i = 0; i < hexlen; i++)
158     {
159         ret[i] = (byte) (value & 0xff);
160         value >>= 8;
161         if (value == 0)
162             break;
163     }
164     return hexToStr(ret);
165 }
166
167 public static byte[] intToAsciiHex(int value, int len)
168 {
169     byte[] ret = new byte[len * 2];
170     for (int i = 0; i < len; i++)
171     {
172         ret[i * 2] = asciiMask[((value & 0xf0) >> 4)];
173         ret[i * 2 + 1] = asciiMask[value & 0xf];
174         value >>= 8;
175     }
176     return ret;
177 }
178
179 public static byte[] intToHex(int value, int len)
180 {
181     byte[] ret = new byte[len];
182     for (int i = 0; i < len; i++)
183     {
184         ret[i] = (byte) ((value >> 8 * (len - i - 1)) & 0xff);
185     }
186     return ret;
187 }
188
189 public static byte[] intToRevHex(int value, int len)
190 {
191     byte[] ret = new byte[len];
192     for (int i = 0; i < len; i++)
193     {
194         ret[i] = (byte) (value & 0xff);
195         value >>= 8;
196     }
197     return ret;
198 }
199
200 public static int hexToInt(byte[] buf, int idx, int len)
201 {
202     int ret = 0;
203
204     final int e = idx + len;
205     for (int i = idx; i < e; ++i)
206     {
207         ret <<= 8;
208         ret |= buf[i] & 0xFF;
209     }
210     return ret;
211 }
212
213 public static int hexToInt(byte[] buf)
214 {
215     return hexToInt(buf, 0, buf.length);
216 }
217
218 public static String hexToStr(byte[] buf)
219 {
220     return hexToStr(buf, 0, buf.length);
221 }
222
223 public static String hexToStr(byte[] buf, int idx, int len)
224 {
225     StringBuffer sb = new StringBuffer();
226     int n;
227     for (int i = 0; i < len; i++)

```

```

228     {
229         n = buf[i + idx] & 0xff;
230         if (n < 0x10)
231         {
232             sb.append("0");
233         }
234         sb.append(Integer.toHexString(n));
235     }
236
237     return sb.toString();
238 }
239
240 private static byte toByte(char c)
241 {
242     return mask[c];
243 }
244
245
246 public static String strToHex(String str)
247 {
248     try
249     {
250         byte[] bytes = str.getBytes("GBK");
251         StringBuilder sb = new StringBuilder(bytes.length * 2);
252         // 转换hex编码
253         for (byte b : bytes)
254         {
255             sb.append(Integer.toHexString(b + 0x800).substring(1));
256         }
257         return sb.toString();
258     }
259     catch (Exception e)
260     {
261         e.printStackTrace();
262     }
263     return null;
264 }
265
266 public static String toHex(byte b) {
267     String result = Integer.toHexString(b & 0xFF);
268     if (result.length() == 1) {
269         result = '0' + result;
270     }
271     return result;
272 }
273
274 }

```