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

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

```
import java.util.Locale;

public class CardCenter
{
    public static void main(String[] args)
    {

        // 圈存的key
        String loadKey = "3F013F013F013F013F013F013F013F01";
        System.out.println("圈存的key:" + loadKey);
        // 验证tac的key
        String tacKey = "34343434343434343434343434343434";
        System.out.println("验证tac的key:" + tacKey);

        System.out.println();

        // posid
        String posid = "112233445566";
        System.out.println("终端ID:" + posid);
        // 交易金额
        String tradeAmount = "00000001";
        System.out.println("交易金额:" + tradeAmount);
        // 交易金额十进制
        int ta = 1;
        // 交易类型
        String tradeType = "02";
        System.out.println("交易类型:" + tradeType);

        System.out.println();
        // 预充值指令
        System.out.println("组装预充值指令:805000020b0100000001112233445566");
        // 预消费指令805000020b0100000001112233445566的指令回复
        String preTopup = "0000001a0017000106b825d7684c81ce9000";
        System.out.println("得到预充值响应:" + preTopup);
        byte[] recvByte = ByteUtil.hexStr2Byte(preTopup);

        // 卡余额
        String balance = ByteUtil.hexToStr(recvByte, 0, 4);
        int bal = ByteUtil.hexToInt(recvByte, 0, 4);
        System.out.println("卡余额:" + balance);

        // 联机计数器
        String cardCnt = ByteUtil.hexToStr(recvByte, 4, 2);
        System.out.println("联机计数器:" + cardCnt);

        // 密钥版本

```

```
        String keyVersion = ByteUtil.hexToStr(recvByte, 6, 1);
        System.out.println("密钥版本:" + keyVersion);

        // 算法标识
        String alglndMark = ByteUtil.hexToStr(recvByte, 7, 1);
        System.out.println("算法标识:" + alglndMark);

        // 随机数
        String random = ByteUtil.hexToStr(recvByte, 8, 4);
        System.out.println("随机数:" + random);

        // mac1
        String mac1 = ByteUtil.hexToStr(recvByte, 12, 4);
        System.out.println("mac1:" + mac1);

        System.out.println("");
        System.out.println("开始验证mac1");
        // 验证mac1的正确性
        // 输入的数据为：随机数+联机计数器+“8000”
        String inputData = random + cardCnt + "8000";
        System.out.println("计算过程密钥数据:" + inputData);
        // 计算过程密钥
        String sessionKey = Des.getHintKey(inputData, loadKey);
        System.out.println("过程密钥:" + sessionKey);

        // 计算mac1需要输入的数据
        // 输入的数据为：余额+交易金额+交易类型+终端编号
        inputData = balance + tradeAmount + tradeType + posid;
        String legitMac1 = Des.PBOC_DES_MAC(sessionKey, "0000000000000000", inputData, 0).substring(0, 8);
        System.out.println("标准的mac1数据:" + legitMac1);

        if (!mac1.toUpperCase(Locale.getDefault()).equals(legitMac1))
        {
            System.out.println("mac1校验失败！");
            return;
        }
        System.out.println("mac1校验成功！");

        // 开始计算Mac2用于做充值确认操作
        System.out.println("");
        System.out.println("开始计算mac2");
        // 交易日期
        String tradeDate = "20170310";
        // 交易时间
        String tradeTime = "110734";
        // 计算mac2的输入数据
        // 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
        inputData = tradeAmount + tradeType + posid + tradeDate + tradeTime;
        String mac2 = Des.PBOC_DES_MAC(sessionKey, "0000000000000000", inputData, 0).substring(0, 8);
        // 得到mac2
        System.out.println("计算后的mac2:" + mac2);

        // 组装写卡指令
        System.out.println("805200000b" + tradeDate + tradeTime + mac2);
        System.out.println("开始向卡片发送充值确认的指令");
        // 向卡发送指令
        System.out.println("apdu:805200000b20170310110734 9C8A0625");
        // 响应tac
        System.out.println("recv:16ead5169000");
        String tac = "16ead516";
        System.out.println("得到Tac:" + tac);

        System.out.println("");
        System.out.println("开始验证tac");
        System.out.println("tac验证密钥:" + tacKey);
        // 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
        String tacTessionKey = Des.xOr(tacKey.substring(0, 16), tacKey.substring(16, 32));
        System.out.println("tac过程密钥:" + tacTessionKey);
        // 充值成功之后新的金额为00000043+00000001=00000044
        String newBalance = ByteUtil.hexToStr(ByteUtil.intToHex(bal + ta, 4));
        // 计算标准的tac的输入数据
        // 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
        inputData = newBalance + cardCnt + tradeAmount + tradeType + posid + tradeDate + tradeTime;
        String legitTac = Des.PBOC_DES_MAC(tacTessionKey, "0000000000000000", inputData, 0).substring(0, 8);
```

```
        System.out.println("标准的tac数据:" + legitTac);

        if (!tac.toUpperCase(Locale.getDefault()).equals(legitTac))
        {
            System.out.println("tac校验错误! ");
            return;
        }
        System.out.println("tac校验成功! ");
    }
}
```

The result is as follows

```
圈存的key:3F013F013F013F013F013F013F013F01
验证tac的key:34343434343434343434343434343434

终端ID:112233445566
交易金额:00000001
交易类型:02

组装预充值指令:805000020b0100000001112233445566
得到预充值响应:0000001a0017000106b825d7684c81ce9000
卡余额:0000001a
联机计数器:0017
密钥版本:00
算法标识:01
随机数:06b825d7
mac1:684c81ce

开始验证mac1
计算过程密钥数据:06b825d700178000
过程密钥:6E76E8541E217D9A
D[0]=0000001a00000001
D[1]=0211223344556680
1**************
I=0000001A00000001
O=D42EE2FF647C9F00
I=D63FC0CC2029F980
I=684C81CE47609B5B
标准的mac1数据:684C81CE
mac1校验成功!

开始计算mac2
D[0]=0000000102112233
D[1]=4455662017031011
D[2]=0734800000000000
1**************
I=0000000102112233
O=A0D9EACF4A0A833D
I=E48C8CEF5D09932C
2**************
I=E48C8CEF5D09932C
O=6DF657FD31FBAB05
I=6AC2D7FD31FBAB05
I=9C8A06256ACBF60A
计算后的mac2:9C8A0625
805200000b201703101107349C8A0625
开始向卡片发送充值确认的指令
apdu:805200000b201703101107349C8A0625
recv:16ead5169000
得到Tac:16ead516

开始验证tac
tac验证密钥:34343434343434343434343434343434
tac过程密钥:0000000000000000
D[0]=0000001b00170000
D[1]=0001021122334455
D[2]=6620170310110734
D[3]=8000000000000000
1**************
I=0000001B00170000
O=F0DE01AF2D71051B
I=F0DF03BE0F42414E
2**************
```

```
I=F0DF03BE0F42414E
O=044D5B737F9A0282
I=626D4C706F8B05B6
3**************
I=626D4C706F8B05B6
O=2B5ED4A6C483D0BF
I=AB5ED4A6C483D0BF
I=16EAD5165389360A
标准的tac数据:16EAD516
tac校验成功!
```

涉及的帮助类

DES工具类

```
import java.io.ByteArrayOutputStream;
import java.io.UnsupportedEncodingException;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

/**
 * DES工具类
 */
public class Des
{
    /**
     * **************************压缩替换S-Box************************************
     * ************
     */
    private static final int[][] s1 =
    {
    { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
    { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
    { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
    { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };
    private static final int[][] s2 =
    {
    { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
    { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
    { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
    { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };
    private static final int[][] s3 =
    {
    { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
    { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
    { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
    { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
    private static final int[][] s4 =
    {
    { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
    { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },// erorr
            { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
    private static final int[][] s5 =
    {
    { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
    { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
    { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
    { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };
    private static final int[][] s6 =
    {
    { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
    { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
    { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
    { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
    private static final int[][] s7 =
    {
    { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
    { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
    { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
    { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

```

```
    private static final int[][] s8 =
    {
    { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
    { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
    { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
    { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };
    private static final int[] ip =
    { 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,
            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 };
    private static final int[] _ip =
    { 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,
            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 };
    // 每次密钥循环左移位数
    private static final int[] LS =
    { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
    private static int[][] subKey = new int[16][48];
    public static int HEX = 0;
    public static int ASC = 1;

    /**
     * 将十六进制A--F转换成对应数
     *
     * @param ch
     * @return
     * @throws Exception
     */
    public static int getIntByChar(char ch) throws Exception
    {
        char t = Character.toUpperCase(ch);
        int i = 0;
        switch (t)
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                i = Integer.parseInt(Character.toString(t));
                break;
            case 'A':
                i = 10;
                break;
            case 'B':
                i = 11;
                break;
            case 'C':
                i = 12;
                break;
            case 'D':
                i = 13;
                break;
            case 'E':
                i = 14;
                break;
            case 'F':
                i = 15;
                break;
            default:
                throw new Exception("getIntByChar was wrong");
        }
        return i;
    }

    /**
     * 将字符串转换成二进制数组
     *
     * @param source
     *            : 16字节
     * @return
```

```
     */
    public static int[] string2Binary(String source)
    {
        int len = source.length();
        int[] dest = new int[len * 4];
        char[] arr = source.toCharArray();
        for (int i = 0; i < len; i++)
        {
            int t = 0;
            try
            {
                t = getIntByChar(arr[i]);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            String[] str = Integer.toBinaryString(t).split("");
            int k = i * 4 + 3;
            for (int j = str.length - 1; j > 0; j--)
            {
                dest[k] = Integer.parseInt(str[j]);
                k--;
            }
        }
        return dest;
    }

    /**
     * 返回x的y次方
     *
     * @param x
     * @param y
     * @return
     */
    public static int getXY(int x, int y)
    {
        int temp = x;
        if (y == 0)
            x = 1;
        for (int i = 2; i <= y; i++)
        {
            x *= temp;
        }
        return x;
    }

    /**
     * s位长度的二进制字符串
     *
     * @param s
     * @return
     */
    public static String binary2Hex(String s)
    {
        int len = s.length();
        int result = 0;
        int k = 0;
        if (len > 4)
            return null;
        for (int i = len; i > 0; i--)
        {
            result += Integer.parseInt(s.substring(i - 1, i)) * getXY(2, k);
            k++;
        }
        switch (result)
        {
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
```

```
            case 7:
            case 8:
            case 9:
                return "" + result;
            case 10:
                return "A";
            case 11:
                return "B";
            case 12:
                return "C";
            case 13:
                return "D";
            case 14:
                return "E";
            case 15:
                return "F";
            default:
                return null;
        }
    }

    /**
     * 将int转换成Hex
     *
     * @param i
     * @return
     * @throws Exception
     */
    public static String int2Hex(int i)
    {
        switch (i)
        {
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
                return "" + i;
            case 10:
                return "A";
            case 11:
                return "B";
            case 12:
                return "C";
            case 13:
                return "D";
            case 14:
                return "E";
            case 15:
                return "F";
            default:
                return null;
        }
    }

    /**
     * 将二进制字符串转换成十六进制字符
     *
     * @param s
     * @return
     */
    public static String binary2ASC(String s)
    {
        String str = "";
        int ii = 0;
        int len = s.length();
        // 不够4bit左补0
        if (len % 4 != 0)
        {
```

```
            while (ii < 4 - len % 4)
            {
                s = "0" + s;
            }
        }
        for (int i = 0; i < len / 4; i++)
        {
            str += binary2Hex(s.substring(i * 4, i * 4 + 4));
        }
        return str;
    }

    /**
     * IP初始置换
     *
     * @param source
     * @return
     */
    public static int[] changeIP(int[] source)
    {
        int[] dest = new int[64];
        for (int i = 0; i < 64; i++)
        {
            dest[i] = source[ip[i] - 1];
        }
        return dest;
    }

    /**
     * IP-1逆置
     *
     * @param source
     * @return
     */
    public static int[] changeInverseIP(int[] source)
    {
        int[] dest = new int[64];
        for (int i = 0; i < 64; i++)
        {
            dest[i] = source[_ip[i] - 1];
        }
        return dest;
    }

    /**
     * 2bit扩展8bit
     *
     * @param source
     * @return
     */
    public static int[] expend(int[] source)
    {
        int[] ret = new int[48];
        int[] temp =
        { 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,
                26, 27, 28, 29, 28, 29, 30, 31, 32, 1 };
        for (int i = 0; i < 48; i++)
        {
            ret[i] = source[temp[i] - 1];
        }
        return ret;
    }

    /**
     * 8bit压缩2bit
     *
     * @param source
     *            (48bit)
     * @return R(32bit) B=E(R)⊕K，将48 位的B 分成8 个分组，B=B1B2B3B4B5B6B7B8
     */
    public static int[] press(int[] source)
    {
        int[] ret = new int[32];
        int[][] temp = new int[8][6];
```

```
        int[][][] s =
        { s1, s2, s3, s4, s5, s6, s7, s8 };
        StringBuffer str = new StringBuffer();
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 6; j++)
            {
                temp[i][j] = source[i * 6 + j];
            }
        }
        for (int i = 0; i < 8; i++)
        {
            // (16)
            int x = temp[i][0] * 2 + temp[i][5];
            // (2345)
            int y = temp[i][1] * 8 + temp[i][2] * 4 + temp[i][3] * 2 + temp[i][4];
            int val = s[i][x][y];
            String ch = int2Hex(val);
            // System.out.println("x=" + x + ",y=" + y + "-->" + ch);
            // String ch = Integer.toBinaryString(val);
            str.append(ch);
        }
        // System.out.println(str.toString());
        ret = string2Binary(str.toString());
        // printArr(ret);
        // 置换P
        ret = dataP(ret);
        return ret;
    }

    /**
     * 置换P(32bit)
     *
     * @param source
     * @return
     */
    public static int[] dataP(int[] source)
    {
        int[] dest = new int[32];
        int[] temp =
        { 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 };
        int len = source.length;
        for (int i = 0; i < len; i++)
        {
            dest[i] = source[temp[i] - 1];
        }
        return dest;
    }

    /**
     * @param R
     *            (2bit)
     * @param K
     *            (48bit的轮子密
     * @return 32bit
     */
    public static int[] f(int[] R, int[] K)
    {
        int[] dest = new int[32];
        int[] temp = new int[48];
        // 先将输入32bit扩展8bit
        int[] expendR = expend(R);// 48bit
        // 与轮子密钥进行异或运
        temp = diffOr(expendR, K);
        // 压缩2bit
        dest = press(temp);
        // printArr(temp);
        return dest;
    }

    /**
     * 两个等长的数组做异或
     *
     * @param source1
```

```
     * @param source2
     * @return
     */
    public static int[] diffOr(int[] source1, int[] source2)
    {
        int len = source1.length;
        int[] dest = new int[len];
        for (int i = 0; i < len; i++)
        {
            dest[i] = source1[i] ^ source2[i];
        }
        return dest;
    }

    /**
     * DES加密--->对称密钥 D = Ln(32bit)+Rn(32bit) 经过16轮置
     *
     * @param D
     *            (16byte)明文
     * @param K
     *            (16byte)轮子密钥
     * @return (16byte)密文
     */
    public static String encryption(String D, String K)
    {
        String str = "";
        int[] temp = new int[64];
        int[] data = string2Binary(D);
        // printArr(data);
        // 第一步初始置
        data = changeIP(data);
        // printArr(data);
        int[][] left = new int[17][32];
        int[][] right = new int[17][32];
        for (int j = 0; j < 32; j++)
        {
            left[0][j] = data[j];
            right[0][j] = data[j + 32];
        }
        // printArr(left[0]);
        // printArr(right[0]);
        setKey(K);// sub key ok
        for (int i = 1; i < 17; i++)
        {
            // 获取(48bit)的轮子密
            int[] key = subKey[i - 1];
            // L1 = R0
            left[i] = right[i - 1];
            // R1 = L0 ^ f(R0,K1)
            int[] fTemp = f(right[i - 1], key);// 32bit
            right[i] = diffOr(left[i - 1], fTemp);
        }
        // 组合的时候，左右调换**************************************************
        for (int i = 0; i < 32; i++)
        {
            temp[i] = right[16][i];
            temp[32 + i] = left[16][i];
        }
        temp = changeInverseIP(temp);
        str = binary2ASC(intArr2Str(temp));
        return str;
    }

    /**
     * DES解密--->对称密钥 解密算法与加密算法基本相同，不同之处仅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
     * 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
     *
     * @param source
     *            密文
     * @param key
     *            密钥
     * @return
     */
    public static String discryption(String source, String key)
```

```
    {
        String str = "";
        int[] data = string2Binary(source);// 64bit
        // 第一步初始置
        data = changeIP(data);
        int[] left = new int[32];
        int[] right = new int[32];
        int[] tmp = new int[32];
        for (int j = 0; j < 32; j++)
        {
            left[j] = data[j];
            right[j] = data[j + 32];
        }
        setKey(key);// sub key ok
        for (int i = 16; i > 0; i--)
        {
            // 获取(48bit)的轮子密
            /** *******不同之处********* */
            int[] sKey = subKey[i - 1];
            tmp = left;
            // R1 = L0
            left = right;
            // L1 = R0 ^ f(L0,K1)
            int[] fTemp = f(right, sKey);// 32bit
            right = diffOr(tmp, fTemp);
        }
        // 组合的时候，左右调换**************************************************
        for (int i = 0; i < 32; i++)
        {
            data[i] = right[i];
            data[32 + i] = left[i];
        }
        data = changeInverseIP(data);
        for (int i = 0; i < data.length; i++)
        {
            str += data[i];
        }
        str = binary2ASC(str);
        return str;
    }

    /**
     * 单长密钥DES(16byte)
     *
     * @param source
     * @param key
     * @param type
     *            0:encrypt 1:discrypt
     * @return
     */
    public static String DES_1(String source, String key, int type)
    {
        if (source.length() != 16 || key.length() != 16)
            return null;
        if (type == 0)
        {
            return encryption(source, key);
        }
        if (type == 1)
        {
            return discryption(source, key);
        }
        return null;
    }

    /**
     * @param source
     * @param key
     * @param type
     *            0:encrypt 1:discrypt
     * @return
     */
    public static String DES_2(String source, String key, int type)
    {
```

```
        return null;
    }

    /**
     * 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
     *
     * @param source
     * @param key
     * @param type
     *            0:encrypt 1:discrypt
     * @return
     */
    public static String DES_3(String source, String key, int type)
    {
        if (key.length() != 32 || source.length() != 16)
            return null;
        String temp = null;
        String K1 = key.substring(0, key.length() / 2);
        String K2 = key.substring(key.length() / 2);
        System.out.println("K1--->" + K1);
        System.out.println("K2--->" + K2);
        if (type == 0)
        {
            temp = encryption(source, K1);
            System.out.println("step1--->" + temp);
            temp = discryption(temp, K2);
            System.out.println("step2--->" + temp);
            return encryption(temp, K1);
        }
        if (type == 1)
        {
            temp = discryption(source, K1);
            temp = encryption(temp, K2);
            return discryption(temp, K1);
        }
        return null;
    }

    /**
     * **********************************48bit的轮子密钥的生成*************************
     * ********************************
     */
    /**
     * 4bit的密钥转换成56bit
     *
     * @param source
     * @return
     */
    public static int[] keyPC_1(int[] source)
    {
        int[] dest = new int[56];
        int[] temp =
        { 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,
                62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4 };
        for (int i = 0; i < 56; i++)
        {
            dest[i] = source[temp[i] - 1];
        }
        return dest;
    }

    /**
     * 将密钥循环左移i
     *
     * @param source
     *            二进制密钥数
     * @param i
     *            循环左移位数
     * @return
     */
    public static int[] keyLeftMove(int[] source, int i)
    {
        int temp = 0;
        int len = source.length;
```

```
        int ls = LS[i];
        for (int k = 0; k < ls; k++)
        {
            temp = source[0];
            for (int j = 0; j < len - 1; j++)
            {
                source[j] = source[j + 1];
            }
            source[len - 1] = temp;
        }
        return source;
    }

    /**
     * 6bit的密钥转换成48bit
     *
     * @param source
     * @return
     */
    public static int[] keyPC_2(int[] source)
    {
        int[] dest = new int[48];
        int[] temp =
        { 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,
                49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32 };
        for (int i = 0; i < 48; i++)
        {
            dest[i] = source[temp[i] - 1];
        }
        return dest;
    }

    /**
     * 获取轮子密钥(48bit)
     *
     * @param source
     * @return
     */
    public static void setKey(String source)
    {
        if (subKey.length > 0)
            subKey = new int[16][48];
        // 装换4bit
        int[] temp = string2Binary(source);
        // 6bit均分成两部分
        int[] left = new int[28];
        int[] right = new int[28];
        // 经过PC-14bit转换6bit
        int[] temp1 = new int[56];
        temp1 = keyPC_1(temp);
        // printArr(temp1);
        // 将经过转换的temp1均分成两部分
        for (int i = 0; i < 28; i++)
        {
            left[i] = temp1[i];
            right[i] = temp1[i + 28];
        }
        // 经过16次循环左移，然后PC-2置换
        for (int i = 0; i < 16; i++)
        {
            left = keyLeftMove(left, LS[i]);
            right = keyLeftMove(right, LS[i]);
            for (int j = 0; j < 28; j++)
            {
                temp1[j] = left[j];
                temp1[j + 28] = right[j];
            }
            // printArr(temp1);
            subKey[i] = keyPC_2(temp1);
        }
    }

    public static void printArr(int[] source)
    {
```

```
        int len = source.length;
        for (int i = 0; i < len; i++)
        {
            System.out.print(source[i]);
        }
        System.out.println();
    }

    /**
     * 将ASC字符串转16进制字符
     *
     * @param asc
     * @return
     */
    public static String ASC_2_HEX(String asc)
    {
        StringBuffer hex = new StringBuffer();
        try
        {
            byte[] bs = asc.toUpperCase().getBytes("UTF-8");
            for (byte b : bs)
            {
                hex.append(Integer.toHexString(new Byte(b).intValue()));
            }
        }
        catch (UnsupportedEncodingException e)
        {
            e.printStackTrace();
        }
        return hex.toString();
    }

    /**
     * 16进制的字符串转换成ASC的字符串
     * 16进制的字符串压缩成BCD(30313233343536373839414243444546)-->(0123456789ABCDEF)
     *
     * @param hex
     * @return
     */
    public static String HEX_2_ASC(String hex)
    {
        String asc = null;
        int len = hex.length();
        byte[] bs = new byte[len / 2];
        for (int i = 0; i < len / 2; i++)
        {
            bs[i] = Byte.parseByte(hex.substring(i * 2, i * 2 + 2), 16);
        }
        try
        {
            asc = new String(bs, "UTF-8");
        }
        catch (UnsupportedEncodingException e)
        {
            e.printStackTrace();
        }
        return asc;
    }

    /**
     * 计算MAC(hex) ANSI-X9.9-MAC(16的整数不补) PBOC-DES-MAC(16的整数补8000000000000000)
     * 使用单倍长密钥DES算法
     *
     * @param key
     *            密钥 (16byte)
     * @param vector
     *            初始向量0000000000000000
     * @param data
     *            数据
     * @return mac
     */
    public static String PBOC_DES_MAC(String key, String vector, String data, int type)
    {
        if (key.length() != 16)
```

```
        {
            return null;
        }
        if (type == ASC)
        {
            data = ASC_2_HEX(data);
        }
        int len = data.length();
        int arrLen = len / 16 + 1;
        String[] D = new String[arrLen];
        if (vector == null)
            vector = "0000000000000000";
        if (len % 16 == 0)
        {
            data += "8000000000000000";
        }
        else
        {
            data += "80";
            for (int i = 0; i < 15 - len % 16; i++)
            {
                data += "00";
            }
        }
        for (int i = 0; i < arrLen; i++)
        {
            D[i] = data.substring(i * 16, i * 16 + 16);
            System.out.println("D[" + i + "]=" + D[i]);
        }
        // D0 Xor Vector
        String I = xOr(D[0], vector);
        String O = null;
        for (int i = 1; i < arrLen; i++)
        {
            System.out.println(i + "**************");
            System.out.println("I=" + I);
            O = DES_1(I, key, 0);
            System.out.println("O=" + O);
            I = xOr(D[i], O);
            System.out.println("I=" + I);
        }
        I = DES_1(I, key, 0);
        System.out.println("I=" + I);
        return I;
    }

    /**
     * 计算MAC(hex) PBOC_3DES_MAC(16的整数补8000000000000000) 前n-1组使用单长密钥DES
     * 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）
     *
     * @param key
     *            密钥 (32byte)
     * @param vector
     *            初始向量0000000000000000
     * @param data
     *            数据
     * @return mac
     */
    public static String PBOC_3DES_MAC(String key, String vector, String data, int type)
    {
        if (key.length() != 32)
        {
            return null;
        }
        if (type == ASC)
        {
            data = ASC_2_HEX(data);
        }
        int len = data.length();
        int arrLen = len / 16 + 1;
        String[] D = new String[arrLen];
        if (vector == null)
            vector = "0000000000000000";
        if (len % 16 == 0)
```

```
        {
            data += "8000000000000000";
        }
        else
        {
            data += "80";
            for (int i = 0; i < 15 - len % 16; i++)
            {
                data += "00";
            }
        }
        for (int i = 0; i < arrLen; i++)
        {
            D[i] = data.substring(i * 16, i * 16 + 16);
            System.out.println("D[" + i + "]=" + D[i]);
        }
        // D0 Xor Vector
        String I = xOr(D[0], vector);
        String O = null;
        String kl = key.substring(0, 16);
        System.out.println("KL8:" + kl);
        for (int i = 1; i < arrLen; i++)
        {
            System.out.println(i + "**************");
            System.out.println("I=" + I);
            O = DES_1(I, kl, 0);
            System.out.println("O=" + O);
            I = xOr(D[i], O);
            System.out.println("I=" + I);
        }
        I = DES_3(I, key, 0);
        return I;
    }

    /**
     * 将s1和s2做异或，然后返回
     *
     * @param s1
     * @param s2
     * @return
     */
    public static String xOr(String s1, String s2)
    {
        int[] iArr = diffOr(string2Binary(s1), string2Binary(s2));
        return binary2ASC(intArr2Str(iArr));
    }

    /**
     * 将int类型数组拼接成字符串
     *
     * @param arr
     * @return
     */
    public static String intArr2Str(int[] arr)
    {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < arr.length; i++)
        {
            sb.append(arr[i]);
        }
        return sb.toString();
    }

    /**
     * 将data分散
     *
     * @param data
     *            数据8字节 16个长度
     * @param key
     *            密钥
     * @param type
     * @return
     */
    public static String divData(String data, String key, int type)
```

```
    {
        String left = null;
        String right = null;
        if (type == HEX)
        {
            left = key.substring(0, 16);
            right = key.substring(16, 32);
        }
        if (type == ASC)
        {
            left = ASC_2_HEX(key.substring(0, 8));
            right = ASC_2_HEX(key.substring(8, 16));
        }
        // 加密
        data = DES_1(data, left, 0);
        // 解密
        data = DES_1(data, right, 1);
        // 加密
        data = DES_1(data, left, 0);
        return data;
    }

    /**
     * 取反(10001)--->(01110)
     *
     * @param source
     *            数据源
     * @return
     */
    public static String reverse(String source)
    {
        int[] data = string2Binary(source);
        int j = 0;
        for (int i : data)
        {
            data[j++] = 1 - i;
        }
        return binary2ASC(intArr2Str(data));
    }

    /**
     * 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
     *
     * @param issuerFlag
     *            发卡方标识符
     * @param appNo
     *            应用序列号即卡号
     * @param mpk
     *            主密钥
     * @return
     */
    public static String getDPK(String issuerFlag, String appNo, String mpk)
    {
        // 第一次分散
        StringBuffer issuerMPK = new StringBuffer();
        // 获取Issuer MPK左半边
        issuerMPK.append(divData(issuerFlag, mpk, 0));
        // 获取Issuer MPK右半边
        issuerMPK.append(divData(reverse(issuerFlag), mpk, 0));
        // 第二次分散
        StringBuffer dpk = new StringBuffer();
        // 获取DPK左半边
        dpk.append(divData(appNo, issuerMPK.toString(), 0));
        // 获取DPK右半边
        dpk.append(divData(reverse(appNo), issuerMPK.toString(), 0));
        return dpk.toString();
    }

    /**
     * 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
     *
     * @param mpk
     *            主密钥
     * @return
```

```
     */
    public static String getDPK4Once(String data, String mpk)
    {
        // 第一次分散
        StringBuffer dpk = new StringBuffer();
        // 获取DPK左半边
        dpk.append(divData(data, mpk, 0));
        // 获取DPK右半边
        dpk.append(divData(reverse(data), mpk.toString(), 0));
        return dpk.toString();
    }

    /**
     * @Title: getHintKey 获取过程密钥
     * @Description: TODO
     * @param: @param data 加密数据
     * @param: @param key 密钥
     * @param: @return
     * @return: String
     * @throws:
     */
    public static String getHintKey(String data, String key)
    {
        if (null == data || data.length() != 16)
            return null;
        if (null != key && key.length() > 16)
            key = key.substring(0, 16);
        if (null == key || key.length() != 16)
            return null;
        // 加密数据
        return encryption(data, key);
    }

    // 字符串转hex玛
    public static byte[] hexStr2Byte(String hex)
    {
        int len = (hex.length() / 2);
        byte[] result = new byte[len];
        char[] achar = hex.toCharArray();
        for (int i = 0; i < len; i++)
        {
            int pos = i * 2;
            result[i] = (byte) (toByte(achar[pos]) << 4 | toByte(achar[pos + 1]));
        }
        return result;
    }

    private static byte toByte(char c)
    {
        return mask[c];
    }

    static byte[] mask = new byte[128];
    private static final String Algorithm = "DESede"; // 定义 加密算法,可用

    // DES,DESede,Blowfish
    // keybyte为加密密钥，长度为24字节
    // src为被加密的数据缓冲区（源）
    public static byte[] encryptMode(byte[] keybyte, byte[] src)
    {
        try
        {
            // 生成密钥
            SecretKey deskey = new SecretKeySpec(keybyte, Algorithm);
            // 从原始密钥数据创建DESKeySpec对象
            // DESedeKeySpec dks = new DESKeySpec(key);
            // 加密
            Cipher c1 = Cipher.getInstance(Algorithm);
            c1.init(Cipher.ENCRYPT_MODE, deskey);
            return c1.doFinal(src);
        }
        catch (java.security.NoSuchAlgorithmException e1)
        {
            e1.printStackTrace();
```

```
        }
        catch (javax.crypto.NoSuchPaddingException e2)
        {
            e2.printStackTrace();
        }
        catch (Exception e3)
        {
            e3.printStackTrace();
        }
        return null;
    }

    public static String printHexString(String hint, byte[] b)
    {
        System.out.print(hint);
        String abc = "";
        for (int i = 0; i < b.length; i++)
        {
            String hex = Integer.toHexString(b[i] & 0xFF);
            if (hex.length() == 1)
            {
                hex = '0' + hex;
            }
            System.out.print(hex.toUpperCase() + " ");
            abc = abc + hex;
        }
        System.out.println("");
        return abc.toUpperCase();
    }

    private static String hexString = "0123456789ABCDEFabcdef";

    /**
     * 将string(包括中文)转为hex
     *
     * @param str
     * @return
     */
    public static String encode(String str)
    {
        byte[] bytes = str.getBytes();
        StringBuilder sb = new StringBuilder(bytes.length * 2);
        // 转换hex编码
        for (byte b : bytes)
        {
            sb.append(Integer.toHexString(b + 0x800).substring(1));
        }
        str = sb.toString();
        return str;
    }

    /**
     * 把hex编码转换为string
     *
     * @param bytes
     * @return
     */
    public static String decode(String bytes)
    {
        bytes = bytes.toUpperCase();
        ByteArrayOutputStream baos = new ByteArrayOutputStream(bytes.length() / 2);
        // 将每2位16进制整数组装成一个字节
        for (int i = 0; i < bytes.length(); i += 2)
            baos.write((hexString.indexOf(bytes.charAt(i)) << 4 | hexString.indexOf(bytes.charAt(i + 1))));
        return new String(baos.toByteArray());
    }
}
```

ByteUtil

```
public class ByteUtil
{

    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++)
```

```
        {
            n *= x;
        }
        return n;
    }

    public static byte[] hexStr2Byte(String hex)
    {
        int len = (hex.length() / 2);
        byte[] result = new byte[len];
        char[] achar = hex.toCharArray();
        for (int i = 0; i < len; i++)
        {
            int pos = i * 2;
            result[i] = (byte) (toByte(achar[pos]) << 4 | toByte(achar[pos + 1]));
        }
        return result;
    }

    public static int revAsciiHexToInt(byte[] ah)
    {
        int ret = 0;
        for (int i = 0; i < ah.length / 2; i++)
        {
            int hex = (mask[ah[i * 2]] << 4) + (mask[ah[i * 2 + 1]]);
            ret += (hex << (8 * i));
        }
        return ret;
    }

    public static int revHexToInt(byte[] data, int off, int len)
    {
        int ret = 0;
        for (int i = 0; i < len; i++)
        {
            ret += (data[off + i] & 0xff) << (8 * i);
        }

        return ret;
    }

    public static int revAsciiHexToInt(byte[] ah, int off, int len)
    {
        int ret = 0;
        for (int i = 0, n = len / 2; i < n; i++)
        {
            int hex = (mask[ah[i * 2 + off]] << 4) + (mask[ah[i * 2 + off + 1]]);
            ret += (hex << (8 * i));
        }
        return ret;
    }

    public static byte[] intToRevAsciiHex(int value, int hexlen)
    {
        byte[] ret = new byte[hexlen * 2];

        for (int i = 0; i < hexlen; i++)
        {
            if (value > 0)
            {
                ret[i * 2] = asciiMask[((value & 0xf0) >> 4)];
                ret[i * 2 + 1] = asciiMask[(value & 0xf)];
                value >>= 8;
            }
            else
            {
                ret[i * 2] = '0';
                ret[i * 2 + 1] = '0';
            }
        }
        return ret;
    }

    public static String intToRevHexString(int value, int hexlen)
```

```
    {

        byte[] ret = new byte[hexlen];
        for (int i = 0; i < hexlen; i++)
        {
            ret[i] = (byte) (value & 0xff);
            value >>= 8;
            if (value == 0)
                break;
        }
        return hexToStr(ret);
    }

    public static byte[] intToAsciiHex(int value, int len)
    {
        byte[] ret = new byte[len * 2];
        for (int i = 0; i < len; i++)
        {
            ret[i * 2] = asciiMask[((value & 0xf0) >> 4)];
            ret[i * 2 + 1] = asciiMask[value & 0xf];
            value >>= 8;
        }
        return ret;
    }

    public static byte[] intToHex(int value, int len)
    {
        byte[] ret = new byte[len];
        for (int i = 0; i < len; i++)
        {
            ret[i] = (byte) ((value >> 8 * (len - i - 1)) & 0xff);
        }
        return ret;
    }

    public static byte[] intToRevHex(int value, int len)
    {
        byte[] ret = new byte[len];
        for (int i = 0; i < len; i++)
        {
            ret[i] = (byte) (value & 0xff);
            value >>= 8;
        }
        return ret;
    }

    public static int hexToInt(byte[] buf, int idx, int len)
    {
        int ret = 0;

        final int e = idx + len;
        for (int i = idx; i < e; ++i)
        {
            ret <<= 8;
            ret |= buf[i] & 0xFF;
        }
        return ret;
    }

    public static int hexToInt(byte[] buf)
    {
        return hexToInt(buf, 0, buf.length);
    }

    public static String hexToStr(byte[] buf)
    {
        return hexToStr(buf, 0, buf.length);
    }

    public static String hexToStr(byte[] buf, int idx, int len)
    {
        StringBuffer sb = new StringBuffer();
        int n;
        for (int i = 0; i < len; i++)
```

```
        {
            n = buf[i + idx] & 0xff;
            if (n < 0x10)
            {
                sb.append("0");
            }
            sb.append(Integer.toHexString(n));
        }

        return sb.toString();
    }

    private static byte toByte(char c)
    {
        return mask[c];
    }


    public static String strToHex(String str)
    {
        try
        {
            byte[] bytes = str.getBytes("GBK");
            StringBuilder sb = new StringBuilder(bytes.length * 2);
            // 转换hex编码
            for (byte b : bytes)
            {
                sb.append(Integer.toHexString(b + 0x800).substring(1));
            }
            return sb.toString();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return null;
    }

    public static String toHex(byte b) {
        String result = Integer.toHexString(b & 0xFF);
        if (result.length() == 1) {
            result = '0' + result;
        }
        return result;
    }


}
```