

中山大学移动信息工程学院

RFID 原理与应用实 验报告

基于 Java 智能卡的电子钱包应用开发

小组成员：

陆 * * *****222

钟 * * *****423

* * * *****438

指导教师：胡建国

目 录

第一章 Java 智能卡的设计概要	1
1.1 项目背景与意义	1
1.2 Java 智能卡发展现状	2
1.3 项目设计内容	2
第二章 电子钱包应用的设计过程	3
2.1 基本文件类型	3
2.1.1 binary file	3
2.1.2 key file	4
2.1.3 EP file	5
2.2 个人化	6
2.2.1 create_file	6
2.2.2 write_key	6
2.2.3 write_binary	7
2.2.4 read_binary	7
2.3 安全管理	8
2.3.1 过程密钥的生成	8
2.3.2 TAC 或 MAC 码的生成	9
2.4 电子钱包基本功能实现	10
2.4.1 圈存	10
2.4.2 消费	12
2.4.3 余额查询	15
第三章 电子钱包应用的功能测试	15

第一章 Java 智能卡的设计概要

1.1 项目背景与意义

智能卡是 IC 卡（集成电路卡）的一种,是由一个或多个集成电路芯片组成的,并封装成便于携带的一种多功能的“电脑片”。根据装载芯片类型的不同、信息通讯方式的不同,又可以分为存储式卡片和微处理器卡片以及接触式卡片、非接触式卡片和双界面卡片。

Java 智能卡技术是 Java 语言编程和智能卡开发相结合的技术,它克服了普通智能卡软硬件开发过于复杂、开发周期长的缺点,在智能卡硬件系统的基础上,通过软件来完成卡内应用程序的下载、安装和运行。Java 智能卡不仅具有普通智能卡的便捷性与安全性,同时还继承了 Java 技术的硬件无关特。与传统的智能卡相比,Java 智能卡技术拥有平台无关性、高灵活性、高安全性以及支持一卡多应用等优点,这些优点使得 Java 智能卡从传统的智能卡中脱颖而出,成为今后智能卡市场的主流产品。Java 智能卡的优点如下:

- (1)平台通用性:卡中应用程序采用 Java 语言开发,不同平台间可灵活移植。
 - (2)一卡多应用性:不同的应用程序可以加载在同一张卡内,相互独立,互不影响。
 - (3)动态下载:以随时对卡中程序进行添加、删除与替换,无需更换卡片。
 - (4)简单灵活性:应用程序可直接利用 Java 开发工具开发,无需了解卡的操作系统。
 - (5)高安全性。卡片既有 Java 语言的安全特性,又支持加密算法及签名功能。
- Java 智能卡的突出优势,决定了它在智能卡市场中的主导地位,也为未来一卡多样化的发展趋势奠定了基础。在信息化技术迅猛发展的今天,智能卡已经在当今的日常生活中得到广泛的应用。Java 智能卡以其超越于普通智能卡“应用易开发、安全性能高、硬件独立性与多功能性”的技术优势,成为应用智能卡中的领跑者。

1.2 Java 智能卡发展现状

针对 Java 智能卡技术的研究，国外的起步早于国内，技术相对成熟，日常的通讯、出行、购物、身份的识别认证等领域都已经应用 Java 智能卡。如斯伦贝谢、金普斯、欧贝特等公司，已经进行了一段时间的合作并推出了一个基于 JavaCard2.1 规范的 SIM 卡,该卡能够让开发商为运营商提供定制的解决方案。

而在国内，针对 Java 智能卡技术的研究起步相对较慢，而且没有完善的具有自主知识产权的 Java 智能卡系统，需要支付巨额版权费。但在一些公司的大力推动下，Java 智能卡的市场正逐渐雄起。Java 智能卡的应用领域已经逐渐涉及到银行业、金融业、通讯业以及人们日常的生活之中，随着时间的推移，技术和应用环境逐渐改善，成本逐渐降低，Java 智能卡必将以其独有的优势在智能卡市场独占鳌头。

1.3 项目设计内容

本项目目的为制作一个可以实现电子钱包功能的 Java 卡片应用。实验开发过程基于 Eclipse 集成开发环境和 JCOP 仿真运行环境。

本实验过程首先建立一个卡内操作系统（Chip Operating System），以进行数据的发送、接收、分析检查、安全测试，进而进行文件读取的功能。接下来对卡内操作系统的安全管理进行设置，包括生成过程密钥，消息验证码（MAC）和交易验证码（TAC）。最后，基于以上功能，实现电子钱包圈存，消费和余额查询的功能。

第二章 电子钱包应用的设计过程

2.1 基本文件类型

在电子钱包应用中，根据文件的结构和用途可以将文件分为二进制文件、EP 文件以及密钥文件。二进制文件为公共应用基本文件以及持卡人基本文件的存储格式。EP 文件用于存储电子钱包信息，如 EP 余额、EP 脱机交易序号、EP 联机序号以及 EP 消费透支限额等信息。密钥文件用于存储安全管理用到的密钥。为便于管理和操作，为各种文件建立相应的辅助类。



2.1.1 binary file

文件结构：

1) 公共应用基本文件

字节	数据元	长度	值
1-8	发卡方标识	8	6264002233330001
9	应用类型标识	1	03
10	发卡方应用版本	1	01
11-20	应用序列号	10	00012001081700000001
21-24	应用启用日期	4	20010101
25-28	应用有效日期	4	20011231
29-30	发卡方自定义 FCI 数据	2	5566

2) 持卡人基本文件

字节	数据元	长度	值
1	卡类型标识	1	00
2	本行职工标识	1	00
3-22	持卡人姓名	20	SAMPLE.CARD.ADF1(当数据的位数没有占满时，数据元左靠齐且右补十六进制“0”)
23-54	持卡人证件号码	32	11010298121800101101029812180010
55	持卡人证件类型	1	05

关键代码：

```
public class BinaryFile {
    private short size;        //二进制文件的大小
    private byte[] binary;     //二进制文件

    public BinaryFile(byte[] pdata);
    /*
     * 功能：写入二进制
     * 参数：off 写入二进制文件 的偏移量； dl 写入的数据长度； data 写入的数据
     * 返回：无
     */
    public final void write_binary(short off, short dl, byte[] data);
    /*
     * 功能：读二进制文件
     * 参数：off 二进制读取的偏移量； len 读取的长度； data 二进制数据的缓冲区
     * 返回：二进制数据的字节长度
     */
    public final short read_binary(short off, short len, byte[] data);
    /*
     * 功能：获取二进制文件大小
     * 参数：无
     * 返回：二进制文件的大小
     */
    public final short get_size();
}
```

2.1.2 key file

文件结构：

标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
06	0x15	34 TAC 密钥	F0	F0	90	00	CEB726EDC01B793BC37DC09E2F768534
07	0x15	3e 消费密钥	F0	F0	01	00	09F4ACB09131420B8FE1B4CC007AC52B
08	0x15	3f 圈存密钥	F0	F0	01	00	EB9BC6DCDF74FF4E4B43F2E34A6727B6

关键代码：

```
public class KeyFile {
    public short size;        //记录的最大存储数量
    public short recNum;     //当前所存储的记录数量
    private Object [] Key;   //密钥记录

    public KeyFile();
    /* 功能：添加密钥
     * 参数：tag 密钥标识符； length 数值的长度； value 数值（5个字节的密钥头+16个字节的密钥值）
     * 返回：无
     */
    public void addkey(byte tag, short length, byte[] value);
    /* 功能：通过密钥标识符获取密钥记录号
     * 参数：tag 密钥标识符
     * 返回：记录号
     */
    public short findkey(byte Tag);
    /* 功能：通过密钥类型获取密钥记录号
     * 参数：type 密钥类型
     * 返回：记录号
     */
    public short findKeyByType(byte Type);
    /* 功能：通过密钥记录号读取密钥
     * 参数：num 密钥记录号 data 所读取到的密钥缓冲区
     * 返回：密钥的长度
     */
    public short readkey(short num, byte[] data);
}
```

2.1.3 EP file

关键代码:

```
public class EPFile{
    public KeyFile keyfile;
    public byte[] EP_balance;    //电子钱包余额
    public byte[] EP_offline;    //电子钱包联机交易序号
    public byte[] EP_online;    //电子钱包脱机交易序号

    byte keyID;        //密钥版本号
    byte algID;        //算法标识符

    public Randgenerator RandData;    //随机数产生
    public PenCipher EnCipher;    //数据加解密方式实现

    public EPFile (KeyFile keyFile);
    /* 功能: 读取电子钱包的信息
     * 参数: KeyFile
     * 返回: 无
     */
    public final short increase (byte[] data, boolean flag);
    /* 功能: 电子钱包金额的增加
     * 参数: data 所增加的金额  flag 是否真正增加电子钱包余额
     * 返回: 圈存后, 余额是否超过最大限额
     */
    public final short init4load(short num, byte[] data);
    /* 功能: 功能: 圈存初始化功能完成
     * 参数: num 密钥记录号,  data 命令报文中的数据段
     * 返回: 0: 圈存初始化命令执行成功    2: 圈存超过电子钱包最大限额
     */
    public final short load (byte[] data);
    /* 功能: 功能: 圈存功能的完成
     * 参数: data 命令报文中的数据段
     * 返回: 0 圈存命令执行成功; 1 MAC2 校验错误;    2 圈存超过最大限额; 3 密钥未找到
     */

    public final short decrease(byte[] data, boolean flag);
    /* 功能: 电子钱包金额减少
     * 参数: data 消费的金额;  flag 是否真正扣减电子钱包余额
     * 返回: 消费是否超额
     */
    public final short init4purchase(short num, byte[] data);
    /* 功能: 消费初始化命令的完成
     * 参数: num 密钥记录号;  data 命令报文中的数据段
     * 返回: 0 命令执行成功; 2 消费超额
     */
    public final short purchase(byte[] data);
    /* 功能: 消费命令的实现
     * 参数: data 命令报文中的数据段
     * 返回: 0 命令执行成功; 1 MAC 校验错误 2 消费超额; 3 密钥未找到
     */

    public final short get_balance (byte[] data);
    /* 功能: 电子钱包余额获取
     * 参数: data 电子钱包余额的缓冲区
     * 返回: 0
     */
}
```

2.2 个人化

个人化指的是将密钥信息、系统应用信息以及持卡人个人信息等写入卡片的过程，其中 主要涉及的命令有：create_file、write_key、write_binary、read_binary

2.2.1 create_file

命令格式：

CLA	INS	P1	P2	Lc	Data
80	e0	00	文件标识	文件信息长度（07）	文件控制信息

* 文件标识取值：密钥文件“00”，应用基本信息文件“16”，持卡人信息文件“17”，电子钱包文件“18”。

* 文件控制信息格式：

文件类型	BYTE 1	BYTE 2-3	BYTE 4	BYTE 5	BYTE 6	BYTE 7
持卡人基本文件	39	0037	FF	FF	FF	FF
应用基本文件	38	001e	FF	FF	FF	FF
电子钱包文件	2F	FFFF	FF	FF	FF	FF
密钥文件	3F	FFFF	FF	FF	FF	FF

根据 data 的第 0 个元素判断要创建的文件的类型，再调用相应文件的构造函数即可。

```
/*
 * 功能：创建文件
 */
private boolean create_file()
{
    switch (papdu.pdata[0]) {
        //todo: 完成创建密钥文件，持卡人基本文件和应用基本文件
        case condef.EP_FILE: return EP_file();
        case condef.KEY_FILE: return Key_file();
        case condef.CARD_FILE: return CARD_file();
        case condef.PERSON_FILE: return PERSON_file();
        default:
            ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
            return false;
    }
}
```

2.2.2 write_key

命令格式：

CLA	INS	P1	P2	Lc	DATA					
80	D4	00	密钥标识	15	密钥标识	使用权	更改权	密钥版号	算法标识	16 字节密钥

先对命令做合法性检查，再调用 KeyFile 的 addkey 方法添加密钥：

```
private boolean write_key()
{
    if (keyfile == null) //还没密钥文件
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    if (papdu.cla != (byte)0x80)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //文件标识不正确
}
```



```

if(papdu.p2 != (byte)0x06 && papdu.p2 != (byte)0x07 && papdu.p2 != (byte)0x08)
    ISOException.throwIt(ISO7816.SW_WRONG_P1P2);

if(papdu.lc == 0 || papdu.lc > 21) //密钥长度不能为 0 也不能超过 21
    ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

if(keyfile.recNum >= 3) //文件空间已满
    ISOException.throwIt(ISO7816.SW_FILE_FULL);

keyfile.addkey(papdu.p2, papdu.lc, papdu.pdata);

return true;
}

```

2.2.3 write_binary

命令格式

CLA	INS	P1	P2	Lc	Data
00	d6	文件标识符	欲写文件的偏移量	数据域长度	数据

p1 取值：持卡人基本文件为 0x17,应用基本文件为 0x16

根据参数 p1 判断要是写应用基本文件还是持卡人基本文件：

```

if(papdu.p1 == 0x16) { //写应用基本文件
    cardfile.write_binary(papdu.p2, papdu.lc, papdu.pdata);
}
else if(papdu.p1 == 0x17) { //写持卡人基本文件
    personfile.write_binary(papdu.p2, papdu.lc, papdu.pdata);
}

```

2.2.4 read_binary

命令格式

CLA	INS	P1	P2	Le
00	b0	文件标识符	欲读文件的偏移量	数据域长度

p1 取值：持卡人基本文件为 0x17,应用基本文件为 0x16

根据参数 p1 判断是读应用基本文件还是持卡人基本文件，并将读到的文件放到 pdata 中。

```

if(papdu.p1 == 0x16) { //应用基本文件
    cardfile.read_binary(papdu.p2, papdu.le, papdu.pdata);
}
else if(papdu.p1 == 0x17) { //持卡人基本文件
    personfile.read_binary(papdu.p2, papdu.le, papdu.pdata);
}

```

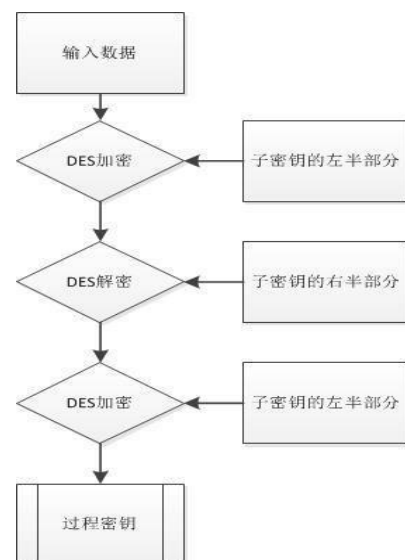
2.3 安全管理

安全管理主要是负责对传送来的数据进行安全性的检验或处理，在电子钱包中包括三个方面：1) 过程密钥的生成，过程密钥是由指定密钥对可变数据加密后产生的单被长密钥，过程密钥只在某一过程中有效。2) 消息验证码 MAC 码的生成，MAC 是报文鉴别码，是中终端和卡片之间的身份认证，用于在交易过程中保证消息的完整性。3) 交易验证码 TAC 的生成，TAC 码用于作为交易的一个有效凭证。

2.3.1 过程密钥的生成

先将数据进行扩充使其长度为 8 的整数倍，然后通过 3DES 算法得到 8 字节长的过程密钥，如右图所示：

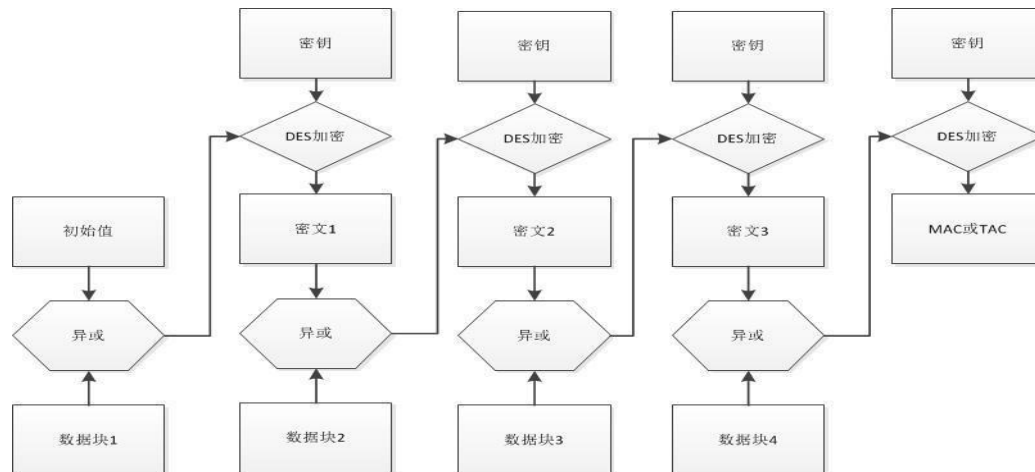
其中输入数据格式为：伪随机数 + 电子钱包联机交易序列号 + 8000。子密钥为：圈存密钥或消费密钥。



```
public final void gen_SESPK(byte[] key,
    byte[] data, short dOff, short dLen,
    byte[] r, short rOff){
    //3次 des 运算
    byte[] buf = JCSysSystem.makeTransientByteArray(
        (short)dLen, JCSysSystem.CLEAR_ON_DESELECT
    );
    short bOff = 0;
    cdes(key, (short)0, data, dOff, dLen, r, rOff, Cipher.MODE_ENCRYPT);
    cdes(key, (short)8, r, rOff, dLen, buf, bOff, Cipher.MODE_DECRYPT);
    cdes(key, (short)0, buf, bOff, dLen, r, rOff, Cipher.MODE_ENCRYPT);
}
```

2.3.2 TAC 或 MAC 码的生成

TAC 和 MAC 计算过程相同，但是输入数据不同。过程如下：1) 将结果的初始值设为 0x0000000000000000 (8byte); 2) 先在数据尾部填入 0x80，再将数据填充至 8 的整数倍长度；3) 将数据按每 8byte 分块；4) 对于每一块先与上一块的结果进行异或再与密钥进行 DES 加密运算。如下图所示：



```

//填充字符串至 8 的倍数
public final short pbocpadding(byte[] data, short
len){ data[len] = (byte)0x80;
len ++;

while(len % 8 !=
0){ data[len] =
(byte)0x00; len++;
}

return len;
}

//两个数据块进行异或，异或结果存入数据块 d1 中
public final void xorblock8(byte[] d1, byte[] d2, short
d2_off){ for(short i=0; i<8; i++){
d1[i] = (byte)(d1[i]^d2[i+d2_off]);
}
}

//MAC 和 TAC 的生成
public final void gmac4(byte[] key, byte[] data, short d1, byte[]
mac){ byte[] buf = JCSysm.makeTransientByteArray(
(short)8, JCSysm.CLEAR_ON_DESELECT
);
for(short i=0; i<(short)8; i++) buf[i] = (byte)0x00;

short d12 = pbocpadding(data,d1);

for(short off=0; off<d12; off+=(short)8){
xorblock8(buf,data,off);
cdes(key,(short)0,buf,(short)0,(short)8,buf,(short)0,Cipher.MODE_ENCRYPT);
}

for(short i=0; i<4; i++) mac[i] = buf[i];
}

```

2.4 电子钱包基本功能实现

2.4.1 圈存

1. 接收圈存初始化命令

CLA	INS	P1	P2	Lc	Data			Le
80	50	00	02		密钥标识符 (1byte)	交易金额 (4byte)	终端机编号 (6byte)	

1) 根据密钥标识符在密钥文件中找到对应的**圈存密钥**，如果没找到则返回“9403”；

2) 生成随机数，再利用找到的圈存密钥生成过程密钥 输入的数据为：**伪随机数||电子钱包联机交易序号||8000 3)**

根据过程密钥生成 MAC1

输入的数据为：**电子钱包余额（交易前）||交易金额||交易类型标识(0x02)||终端机编号**

4) 返回相应数据：**EP 余额、EP 联机交易序列号、密钥版本号（DPK）、算法标识（DPK）、伪随机数、MAC1。**

4) 返回相应数据：EP 余额、EP 联机交易序列号、密钥版本号（DPK）、算法标识（DPK）、伪随机数、MAC1。

2. 终端确认 MAC1 的合法性后发送圈存命令

3. 接收圈存命令

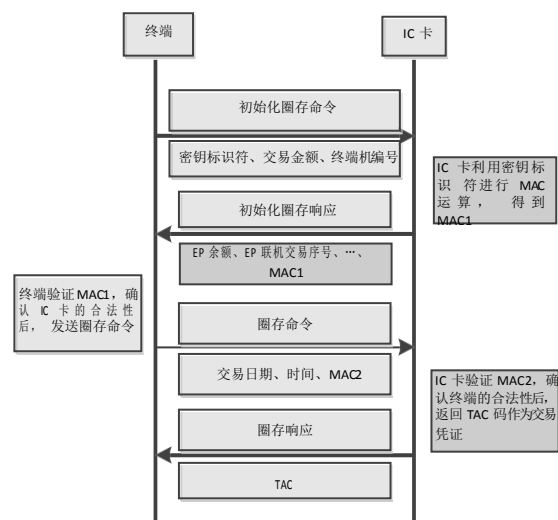
CLA	INS	P1	P2	Lc	Data			Le
80	52	00	00		交易日期 (4byte)	交易时间 (3byte)	MAC2 (4byte)	

1) 用过程密钥验证接收到的 MAC2 输入数据为**交易金额||交易类型标识||终端机编号||交易日期（主机）||交易时间（主机）** 2) 将联机交易号加 1，并且把交易金额加在电子钱包的余额上

3) 计算并返回 TAC 码:

输入的数据：**电子钱包余额（交易后）|| 电子钱包联机交易序号（加 1 前）||交易金额|| 交易类型标识||终端机编号||交易日期（主机）||交易时间（主机）。**

密钥为：**TAC 密码最左 8 个字节与 TAC 密码最右 8 个字节异或的结果。**



```

//圈存判断是否超出最大限额
public final short increase(byte[] data, boolean
    flag){ short i, t1, t2, ads;

    ads = (short)0;
    for (i = 3; i >= 0; i --){
        t1 = (short)(EP_balance[(short)i] & 0xFF);
        t2 = (short)(data[i] & 0xFF);

        t1 = (short)(t1 + t2 + ads);
        if(flag) EP_balance[(short)i] = (byte)(t1 % 256);
        ads = (short)(t1 / 256);
    }
    return ads;
}

//圈存初始化
public final short init4load(short num, byte[]
    data){ short length,rc;

    Util.arrayCopyNonAtomic(data, (short)1, pTemp42, (short)0, (short)4); //交易金额
    Util.arrayCopyNonAtomic(data, (short)5, pTemp81, (short)0, (short)6); //终端机编号

    //判断是否超额圈存
    rc = increase(pTemp42, false);
    if(rc != (short)0)
        return (short)2;

    //密钥获取
    length = keyfile.readkey(num, pTemp32);
    keyID = pTemp32[3];
    algID = pTemp32[4];
    Util.arrayCopyNonAtomic(pTemp32, (short)5, pTemp16, (short)0, length); //圈存密钥

    //产生随机数 (4byte)
    RandData.GenerateSecureRnd();
    RandData.getRndValue(pTemp32, (short)0);

    //产生过程密钥
    //电子钱包联机交易序号
    Util.arrayCopyNonAtomic(EP_online, (short)0, pTemp32, (short)4, (short)2);

    pTemp32[6] = (byte)0x80; //0x8000
    pTemp32[7] = (byte)0x00;

    //          圈存密钥      数据          过程密钥
    Encipher.gen_SESPK(pTemp16, pTemp32, (short)0, (short)8, pTemp82, (short)0);

    //产生 MAC1

    //获取 电子钱包余额(4byte)|| 交易金额(4byte)|| 交易类型标识(1byte)|| 终端机编号(6byte)
    //得到数据 pTemp32
    { . . . }

    //          过程密钥 数据          MAC1
    Encipher.gmac4(pTemp82, pTemp32, (short)0x0F, pTemp41);

    //响应数据
    //电子钱包余额 [0...3]
    //电子钱包联机交易序号 [4...5]
    //密钥版本号 [6]
    //算法标识 [7]
    //随机数 [8...11]
    //mac1 [12...15]
    //将过程密钥赋给 data[16]~data[23];
    Util.arrayCopyNonAtomic(EP_balance, (short)0, data, (short)0, (short)4);
    . . . .

    return 0;
}

//圈存功能的完成
public final short load(byte[]
    data){ short rc;

    //获取 交易金额||交易标识||终端机编号||交易日期与时间 最后用于产生 MAC2
    Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
    pTemp32[4] = (byte)0x02;
    Util.arrayCopyNonAtomic(pTemp81, (short)0, pTemp32, (short)5, (short)6);
    Util.arrayCopyNonAtomic(data, (short)0, pTemp32, (short)11, (short)7);
    Encipher.gmac4(pTemp82, pTemp32, (short)0x12, pTemp41); //产生
    MAC2

```

```

//检验 MAC2
if(Util.arrayCompare(data, (short)7, pTemp41, (short)0, (short)4) != (byte)0x00)
    return (short)1;

//电子钱包数目增加
rc = increase(pTemp42, true);
if(rc != (short)0)
    return 2;

//TAC 数据
//获取 电子钱包余额||电子钱包联机交易序号||交易金额||交易类型||终端机编号||交易日期与时间

Util.arrayCopyNonAtomic(EP_balance, (short)0, pTemp32, (short)0, (short)4);
Util.arrayCopyNonAtomic(EP_online, (short)0, pTemp32, (short)4, (short)2);
Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)6, (short)4);
pTemp32[10] = (byte)0x02;
Util.arrayCopyNonAtomic(pTemp81, (short)0, pTemp32, (short)11, (short)6);
Util.arrayCopyNonAtomic(data, (short)0, pTemp32, (short)17, (short)7);

//联机交易序号加1
rc = Util.makeShort(EP_online[0], EP_online[1]);
rc ++;
if(rc > (short)256)
    rc = (short)1;
Util.setShort(EP_online, (short)0, rc);

//TAC 的计算
short length, num;
num = keyfile.findKeyByType((byte)0x34);
length = keyfile.readkey(num, pTemp16);

if(length == 0)
    return (short)3;

Util.arrayCopyNonAtomic(pTemp16, (short)5, pTemp82, (short)0, (short)8);

Encipher.xorblock8(pTemp82, pTemp16, (short)13);
Encipher.gmac4(pTemp82, pTemp32, (short)0x18, data);

return (short)0;
}

```

2.4.2 消费

1. 终端发出初始化消费命令启动消费交易

CLA	INS	P1	P2	Lc	Data			Le
80	50	01	02	0b	密钥标识符	交易金额 (4byte)	终端机编号 (6byte)	0f

2. IC 卡处理消费初始化命令

- 1) 根据密钥标识符查找密钥，找不到则返回“9403”
- 2) 生成随机数
- 3) 检查电子钱包余额是否大于或等于交易金额，如果小于交易金额，则回送“9401”
- 4) 返回相应数据：EP 余额、EP 脱机交易序列号、透支限额、密钥版本号 DPK、算法标识 DPK、伪随机数

3. 终端生成 MAC1

4. 终端发送消费命令

CLA	INS	P1	P2	Lc	Data				Le
80	54	01	00	0f	终端交易序号	交易日期	交易时间	MAC1	08

5. IC 卡利用所查找到的密钥产生过程密钥。输入数据为：伪随机数||电子钱包脱机交易序

号||终端交易序号的最右两个字节。

6. IC卡利用过程密钥生成 MAC1，验证接收到的 MAC1。输入数据为：交易金额||交易类型标识(0x06)||终端机编号||交易日期（主机）||交易时间（主机）

7. IC卡将电子钱包脱机交易序号加 1，并且把电子钱包的余额减去交易金额。

8. IC卡利用过程密钥生成 MAC2。输入数据为：交易金额。

9. IC卡生成 TAC 码。TAC 码的生成方式和 MAC 码的生成方式一致。其输入的数据：交易金额||交易类型标识||终端机编号||终端交易序号||交易日期（主机）||交易时间（主机）。密钥为 TAC 密码最左 8 个字节与 TAC 密码最右 8 个字节异或的结果。

10. IC卡返回 TAC 码和 MAC2

```
//电子钱包金额的减少
public final short decrease(byte[] data, boolean
flag){ short t1, t2, ads;
ads = (short)0;
for (short i = 3; i >= 0; i--){
//因为 EP_balance[i]和 data[i]是一个字节,而 short 为两个字节,所以要补 0xFF
t1 = (short)(EP_balance[(short)i] & 0xFF);
t2 = (short)(data[i] & 0xFF);
t1 = (short)(t1 - t2 - ads); //当前位 = 原值 - 对应位置消费值 - 借位

if(t1 >= 0) ads = 0; //是否向前一位借
1 else ads = 1;

if(flag) EP_balance[(short)i] = (byte)(t1 % 256);
}
return ads;
}

//消费初始化
public final short init4purchase(short num, byte[]
data){ short length,rc;

Util.arrayCopyNonAtomic(data, (short)1, pTemp42, (short)0, (short)4); //交易金额
Util.arrayCopyNonAtomic(data, (short)5, pTemp81, (short)0, (short)6); //终端机编号

//判断余额是否足够,此时不修改余额
rc = decrease(pTemp42, false);
if(rc != (short)0) return (short)2; //余额不足返回 2

//密钥获取
length = keyfile.readkey(num, pTemp32);
keyID = pTemp32[3];
algID = pTemp32[4];
//获取之前存进文件的消费密钥,存在 pTemp16 中
Util.arrayCopyNonAtomic(pTemp32, (short)5, pTemp16, (short)0, length);

//产生随机数放在 PTemp32
RandData.GenerateSecureRnd();
RandData.getRndValue(pTemp32, (short)0);

//返回响应数据
//电子钱包余额 data[0]~data[3]
//电子钱包脱机交易序号 data[4]~data[5]
//透支限额 data[6]~data[8]
//密钥版本号 data[9]
//算法标识 data[10]
//将随机数赋给 data[11]~data[14]
Util.arrayCopyNonAtomic(EP_balance, (short)0, data, (short)0, (short)4);
Util.arrayCopyNonAtomic(EP_offline, (short)0, data, (short)4, (short)2);
byte[] touzhi = {0x00,0x00,0x00};
Util.arrayCopyNonAtomic(touzhi, (short)0, data, (short)6, (short)3);
data[9] = keyID;
data[10] = algID;
RandData.getRndValue(data, (short)11);

return 0;
}
```

```

//消费的实现
public final short purchase(byte[]
data){ short rc;
//消费密钥之前已经存进了 pTemp16
//伪随机数之前已经存进了 pTemp32

//产生过程密钥
Util.arrayCopyNonAtomic(EP_offline, (short)0, pTemp32, (short)4, (short)2);
//pTemp32[4]~[5]赋值的是脱机交易序列号 EP_offline
Util.arrayCopyNonAtomic(data, (short)2, pTemp32, (short)6, (short)2);
//pTemp32[6]~[7]赋值的是终端交易序号的最后两个字节

Encipher.gen_SESPK(pTemp16, pTemp32, (short)0, (short)8, pTemp82, (short)0);
//pTemp82 为得到的过程密钥

//产生 MAC1
//交易金额 pTemp32[0]~pTemp32[3]
//交易类型标识 pTemp32[4]
//终端机编号 pTemp32[5]~pTemp32[10]
//交易日期和时间 pTemp32[11]~[17]
Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
. . .

Encipher.gmac4(pTemp82, pTemp32, (short)18, pTemp41);
//产生 mac1 并存储在 pTemp41,data 为 pTemp32

//检验 MAC1
if(Util.arrayCompare(data, (short)11, pTemp41, (short)0, (short)4) != (byte)0x00)
    return (short)1; //不相同则返回 1

//脱机交易序号加 1
rc = Util.makeShort(EP_offline[0], EP_offline[1]);
rc ++;
if(rc > (short)256) rc = (short)1;
Util.setShort(EP_offline, (short)0, rc);

//电子钱包金额减少
rc = decrease(pTemp42, true);
if(rc != (short)0) return (short)2;

//MAC2 生成
Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
//交易金额作为数据输入 pTemp32[0]~[3]
Encipher.gmac4(pTemp82, pTemp32, (short)4, pTemp41);
//得到 mac2 存进 pTemp41, 注意这里会把数据 pTemp32 修改!

//TAC 数据
//交易金额[0]~[3]
//交易类型标识[4]
//终端机编号[5]~[10]
//终端交易序号[11]~[14]
//交易日期与时间[15]~[21], 在返回 mac2 之前
Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
. . .

//TAC 的计算
short length, num;
num = keyfile.findKeyByType((byte)0x34);
length = keyfile.readkey(num, pTemp16);

if(length == 0) return (short)3;

//去掉前五位密钥头部
Util.arrayCopyNonAtomic(pTemp16, (short)5, pTemp82, (short)0, (short)8);
Encipher.xorblock8(pTemp82, pTemp16, (short)13); //密钥左 8 位和右 8 位异或得到新密
钥

//得到 tac 同时返回 tac 给终端
byte[] temp = JCSysSystem.makeTransientByteArray((short)8, JCSysSystem.CLEAR_ON_DESELECT);
Util.arrayCopyNonAtomic(pTemp82, (short)0, temp, (short)0, (short)8);

//返回 mac2
Util.arrayCopyNonAtomic(pTemp41, (short)0, data, (short)4, (short)4);
Util.arrayCopyNonAtomic(temp, (short)0, data, (short)8, (short)8);
Encipher.gmac4(temp, pTemp32, (short)22, data); //得到 tac 直接复制给 data 返回了

return 0;
}

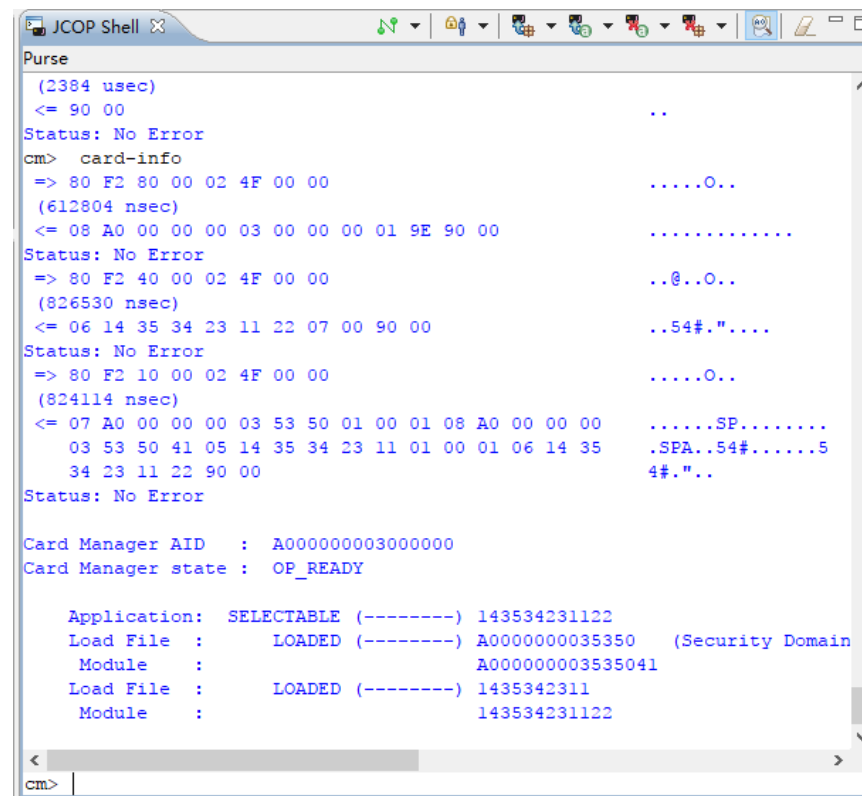
```


2.4.3 余额查询

```
/*
 * 功能: 电子钱包余额获取
 * 参数: data 电子钱包余额的缓冲区
 * 返回: 0
 */
public final short get_balance(byte[]
    data){ for(short i = 0;i < 4;i++)
    {
        data[i] = EP_balance[i];
    }
    return 0;
}
```

第三章 电子钱包应用的功能测试

直接运行的运行结果



```
JCop Shell X
Purse
(2384 usec)
<= 90 00
Status: No Error
cm> card-info
=> 80 F2 80 00 02 4F 00 00
(612804 nsec)
<= 08 A0 00 00 00 03 00 00 00 01 9E 90 00
Status: No Error
=> 80 F2 40 00 02 4F 00 00
(826530 nsec)
<= 06 14 35 34 23 11 22 07 00 90 00
Status: No Error
=> 80 F2 10 00 02 4F 00 00
(824114 nsec)
<= 07 A0 00 00 00 03 53 50 01 00 01 08 A0 00 00 00
03 53 50 41 05 14 35 34 23 11 01 00 01 06 14 35
34 23 11 22 90 00
Status: No Error
Card Manager AID : A000000003000000
Card Manager state : OP_READY

Application: SELECTABLE (-----) 143534231122
Load File : LOADED (-----) A0000000035350 (Security Domain
Module : A000000003535041
Load File : LOADED (-----) 1435342311
Module : 143534231122
cm>
```

3.1. 选择电子钱包文件，命令为: /select 1435342311 (14353423 为小组一成员学号)

```
cm> /select 1435342311
=> 00 A4 04 00 05 14 35 34 23 11 00
(447980 nsec)
<= 90 00
Status: No Error
```

3.2. 由实验二文档可知， 文件创建命令报文格式为：

CLA INS P1 P2 LC DATA
80 e0 00 P2 07 DATA

3.2.1. 建立密钥文件: /send 80e00000073fffffffffff
创建密钥文件的 P2 为 00， DATA 为 3fffffffffff

```
cm> /send 80e00000073fffffffffff
=> 80 E0 00 00 07 3F FF FF FF FF FF
(795739 nsec)
<= 90 00
Status: No Error
```

3.2.2. 建立发卡方基本信息文件: /send 80e000160738001effffffff
应用基本文件的 P2 为 16， DATA 为 38001effffffff

```
cm> /send 80e000160738001effffffff
=> 80 E0 00 16 07 38 00 1E FF FF FF FF .....8.....
(668953 nsec)
<= 90 00 ..
Status: No Error
```

3.2.3. 建立持卡人基本信息文件:/send 80e0001707390037fffffffff
持卡人基本信息文件 P2 为 17， DATA为 390037fffffffff

```
cm> /send 80e0001707390037fffffffff
=> 80 E0 00 17 07 39 00 37 FF FF FF FF
(708197 nsec)
<= 90 00
Status: No Error
```

3.2.4. 建立电子钱包文件:/send 80e00018072Ffffffffffffff

3.2.5. 建立电子钱包文件 P2 为 18， DATA 为 2Ffffffffffffff

```
Status: No Error
cm> /send 80e00018072Ffffffffffffff
=> 80 E0 00 18 07 2F FF FF FF FF FF FF ...../.....
(1835 usec)
<= 90 00 ..
Status: No Error
```

3.3. 写入密钥文件

CLA INS P1 P2 LC DATA
80 d4 00 密钥标识 15 3e/3f/34+使用权+更改权+密钥版本号+算法标识+16 字节密钥

文件体（密钥）							
标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
06	0x15	34（TAC 密钥）	F0	F0	90	00	CEB726EDC01B793BC37DC09E2F768534
标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
07	0x15	3e（消费密钥）	F0	F0	01	00	09F4ACB09131420B8FE1B4CC007AC52B
标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
08	0x15	3f（圈存密钥）	F0	F0	01	00	EB9BC6DCDF74FF4E4B43F2E34A6727B6

3.3.1. 增加消费密钥:

/send 80d40007153ef0f0010009F4ACB09131420B8FE1B4CC007AC52B

```
cm> /send 80d40007153ef0f0010009F4ACB09131420B8FE1B4CC007AC52B
=> 80 D4 00 07 15 3E F0 F0 01 00 09 F4 AC B0 91 31 .....>....
42 0B 8F E1 B4 CC 00 7A C5 2B B.....Z.+
(713027 nsec)
<= 90 00 ..
Status: No Error
```

3.3.2. 增加圈存密钥:

```
/send 80d40008153ff0f00100EB9BC6DCDF74FF4E4B43F2E34A6727B6
cm> /send 80d40008153ff0f00100EB9BC6DCDF74FF4E4B43F2E34A6727B6
=> 80 D4 00 08 15 3F F0 F0 01 00 EB 9B C6 DC DF 74 .....?.....
    FF 4E 4B 43 F2 E3 4A 67 27 B6 .....NKC..Jg'.
(649029 nsec)
<= 90 00 ..
Status: No Error
```

3.3.3. 增加 TAC 密钥

```
/send 80d400061534f0f09000CEB726EDC01B793BC37DC09E2F768534
cm> /send 80d400061534f0f09000CEB726EDC01B793BC37DC09E2F768534
=> 80 D4 00 06 15 34 F0 F0 90 00 CE B7 26 ED C0 1B .....4....
    79 3B C3 7D C0 9E 2F 76 85 34 y:.)../v.4
(862152 nsec)
<= 90 00 ..
Status: No Error
```

3.4. 写二进制文件

CLA	INS	P1	P2	LC	DATA
00	D6	文件标识符	欲写文件的偏移量	数据域长度	数据

3.4.1. 写入发卡方基本信息:

/send 00D616001E626400223333000103010001200108170000000120010101200112315566
写入应用基本文件的 P1 为 0x16，再有

文件体			
字节	数据元	长度	值
1-8	发卡方标识	8	6264002233330001
9	应用类型标识	1	03
10	发卡方应用版本	1	01
11-20	应用序列号	10	00012001081700000001
21-24	应用启用日期	4	20010101
25-28	应用有效日期	4	20011231
29-30	发卡方自定义 FCI 数据	2	5566

所以偏移量为 00，数据域长度为：8+1+1+10+4+4+2=30=0x1E

```
cm> /send 00D616001E626400223333000103010001200108170000000120010101200112315566
=> 00 D6 16 00 1E 62 64 00 22 33 33 00 01 03 01 00 .....bd."33....
    01 20 01 08 17 00 00 00 01 20 01 01 01 20 01 12 . .... ..
    31 55 66 .....1Uf
(787287 nsec)
<= 90 00 ..
Status: No Error
```

3.4.2. 写入持卡人基本信息:/send

00D6170037000053414D504C452E434152442E4144463100000000110102981218001
01101029812180010000000000000000000000000000000000000005
写入持卡人信息的 P1 为 0x17，再有

文件体			
字节	数据元	长度	值
1	卡类型标识	1	00
2	本行职工标识	1	00
3-22	持卡人姓名	20	SAMPLE.CARD.ADF1(当数据的位数没有占满时，数据元左靠齐且右补十六进制"0")
23-54	持卡人证件号码	32	11010298121800101101029812180010
55	持卡人证件类型	1	05

偏移量为 00，数据域长度为 55=0x37

```
cm> /send 00D6170037000053414D504C452E434152442E4144463100000000110102981218001011010298
=> 00 D6 17 00 37 00 00 53 41 4D 50 4C 45 2E 43 41 ....7..SAMPLE.CA
    52 44 2E 41 44 46 31 00 00 00 00 11 01 02 98 12 RD.ADF1.....
    18 00 10 11 01 02 98 12 18 00 10 00 00 00 00 00 .....
    00 00 00 00 00 00 00 00 00 00 05 .....
(609181 nsec)
<= 90 00 ..
Status: No Error
```

3.5. 读取信息

CLA INS P1 P2 LC
00 B0 文件标识符 欲读文件的偏移量 数据域长度

3.5.1. 读取发卡方基本信息: /send 00B016001E

从上面 4 可以知道，发卡方基本信息文件的标识为 0x16，偏移量为 00，长度为 1E

```
cm> /send 00B016001E
=> 00 B0 16 00 1E .....
(752874 nsec)
<= 62 64 00 22 33 33 00 01 03 01 00 01 20 01 08 17 bd."33.....
    00 00 00 01 20 01 01 01 20 01 12 31 55 66 90 00 .... ..1Uf..
Status: No Error
```

用返回的数据和上面发卡方的基本信息进行对比，可以看出是一致的

文件体			
字节	数据元	长度	值
1-8	发卡方标识	8	6264002233330001
9	应用类型标识	1	03
10	发卡方应用版本	1	01
11-20	应用序列号	10	00012001081700000001
21-24	应用启用日期	4	20010101
25-28	应用有效日期	4	20011231
29-30	发卡方自定义 FCI 数据	2	5566

3.5.2. 读取持卡人基本信息: /send 00B0170037

持卡人基本信息文件的标识为 0x17，偏移量为 00，长度为 0x37

```
cm> /send 00B0170037
=> 00 B0 17 00 37 .....7
(578391 nsec)
<= 00 00 53 41 4D 50 4C 45 2E 43 41 52 44 2E 41 44 ..SAMPLE.CARD.AD
    46 31 00 00 00 00 11 01 02 98 12 18 00 10 11 01 F1.....
    02 98 12 18 00 10 00 00 00 00 00 00 00 00 00 00 .....
    00 00 00 00 00 00 05 90 00 .....
Status: No Error
```

同样，与之前写入的持卡人信息完全一致

文件体			
字节	数据元	长度	值
1	卡类型标识	1	00
2	本行职工标识	1	00
3-22	持卡人姓名	20	SAMPLE.CARD.ADF1(当数据的位数没有占满时，数据元左靠齐且右补十六进制"0")
23-54	持卡人证件号码	32	11010298121800101101029812180010
55	持卡人证件类型	1	05

3.6. 圈存

3.6.1. 圈存初始化: /send 8050000020B080000100000112233445510

代码	值	
CLA	80	
INS	50	
P1	00	
P2	02	
Lc	08	
Data	密钥标识符	08
	交易金额	00 00 10 00 (40.96 元)
	终端机编号	00 11 22 33 44 55
Le	10	

```
cm> /send 8050000020B080000100000112233445510
=> 80 50 00 02 08 08 00 00 10 00 00 11 22 33 44 55 .P.....
    10 联机交易 算法
(4518 nsec) 金额 序列号 标识 随机数 MAC1前四位
<= 00 00 00 00 00 00 01 00 DA 23 50 6D BB 69 08 24 .....
    90 00 密钥
Status: No Error 版本号
```

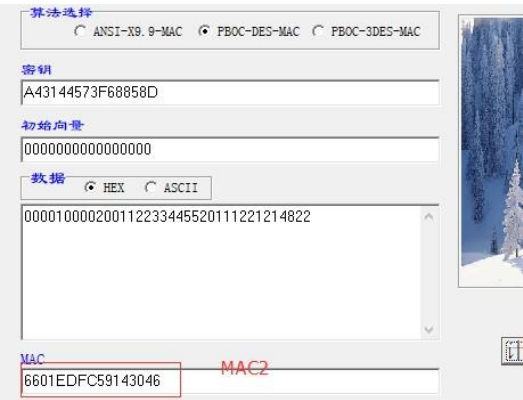
用 Des.exe 对 MAC1 进行验证



MAC1 正确无误

3.6.2. 圈存:

借由刚才得到的过程密钥，利用 Des.exe 计算 MAC2，输入数据为 00001000020011223344552011221214822(交易金额||交易类型标识||终端机编号||交易日期||交易时间)



取计算所得 MAC2 的前四字节放到命令中得到:

/send 805200000B20112212148226601EDFC04

代码	值	
CLA	80	
INS	52	
P1	00	
P2	00	
Lc	0B	
Data	交易日期（主机）	20 11 12 21（2011 年 12 月 11 日）

Le	交易时间（主机）	21 48 22（21 点 48 分 22 秒）
	MAC2	MAC2
		04

结果如下:

```
cm> /send 805200000B20112212148226601EDFC04
=> 80 52 00 00 0B 20 11 12 21 21 48 22 66 01 ED FC .R.
04 .
(5864 usec)
<= 14 62 AD 13 90 00 .b.
Status: No Error
```

成功圈存并返回 TAC 码

3.6.3. 查询余额: /send 805C000204

代码	值
CLA	80
INS	5C
P1	00
P2	02
Lc	不存在
Data	不存在
Le	04

```
cm> /send 805C000204
=> 80 5C 00 02 04
(924338 nsec)
<= 00 00 10 00 90 00
Status: No Error
```

确实存入了 40.96 元

3.7. 消费

3.7.1. 消费初始化: /send 805001020B07000010000011223344550F

代码	值	
CLA	80	
INS	50	
P1	01	
P2	02	
Lc	0B	
Data	密钥标识符	07
	交易金额	00 00 10 00 (40.96 元)
	终端机编号	00 11 22 33 44 55
Le	0F	

运行结果为

```
cm> /send 805001020B07000010000011223344550F
=> 80 50 01 02 0B 07 00 00 10 00 00 11 22 33 44 55 .P.....
    0F .
(1043 usec)
<= 00 00 10 00 00 00 00 00 01 00 DF 58 6A 8C 90 .....
    00 .
Status: No Error
```

由实验三文档可以知道

说明	值
EP 余额	00 00 10 00
EP 脱机交易序列号	00 00
透支限额	00 00 00
密钥版本号 DPK	01
算法标识 DPK	00
伪随机数 (IC 卡)	伪随机数

得到随机数 DF 58 6A 8C, 用于消费的 MAC1 的计算

3.7.2. 消费的实现

一样用 Des.exe 求出 MAC1, 将其前四个字节放到命令中



去计算得到 MAC1 的前四个字节，替换到命令中：

/send 805401000F010203042011122121482203EB30D108

代码	值	
CLA	80	
INS	54	
P1	01	
P2	00	
Lc	0F	
Data	终端交易序号	01 02 03 04
	交易日期（主机）	20 11 12 21（2011 年 12 月 11 日）
	交易时间（主机）	21 48 22（21 点 48 分 22 秒）
	MAC1	MAC1
Le	08	

```
cm> /send 805401000F010203042011122121482203EB30D108
=> 80 54 01 00 0F 01 02 03 04 20 11 12 21 21 48 22 .T..... ..
    03 EB 30 D1 08 ..0..
(7545 usec)
<= 11 83 BB A1 47 BA 93 3A 90 00 ....G....
Status: No Error
```



返回 TAC 的前四个字节，和 MAC2 的前四个字节，用 Des.exe 分别对其验证 TAC:

长度选择

☒ 8bytes

☐ 16bytes

数据1

CEB726EDC01B793B TAC密钥左8字节

数据2

C37DC09E2F768534 TAC密钥右8字节

结果

0DCAE673EF6DFC0F 异或结果作为密钥

XOR

算法选择

☐ ANSI-X9.9-MAC

☒ FB0C-DES-MAC

☐ FB0C-3DES-MAC

密钥

0DCAE673EF6DFC0F

初始向量

0000000000000000

数据

☒ HEX

☐ ASCII

00001000060011223344550102030420111221214822
交易金额 || 交易类型 || 终端机编号 || 终端交易序号
|| 交易日期 || 交易时间

MAC

1183BBA1D794B346 TAC

计算MAC

```
/send 805401000F010203042011122121482203EB30D108
80 54 01 00 0F 01 02 03 04 20 11 12 21 21 48 22 .T..... ..
03 EB 30 D1 08 ..0..
545 usec) 返回的TAC
11 83 BB A1 47 BA 93 3A 90 00 ....G....
tus: No Error
```

MAC2:

