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

姚云龙　14353363
唐珊　14353283
马怡平　14353235

指导教师：胡建国

目　　录

第一章 Java 智能卡的设计概要

1.1 项目背景与意义

无线射频识别技术(Radio Frequency Identification, RFID)的基本原理是利用空间电磁波的耦合或传播进行通信， 以达到自动识别被标识对象，获取标识对象相关信息的目的。与条形码比较，RFID 具有很多优势：不需要视距传输，复用性强，能够快速存储和读取更多数据，具有防水、防磁、耐高温等特性。它不仅可以实现对被识别物体的自动识别与快速读写，还可以同时识别多个标签以及高速移动的标签，操作快捷简单，能工作于各类不同的恶劣环境中，是目前最先进的自动识别技术。

本次项目中利用网络安全知识以及 RFID 知识，让我们实现了一个 Java 卡和终端积极进行通信并且进行圈存、消费、查询的功能。让我们对 EFID 的运行模式和其编程实现有了更加深入的了解。也让我们对信息安全有了一个更加直观的了解。

通过移步一步实现 Java 卡系统功能，我们尝到了仿真环境下成功运行的喜悦。也初步体会了 Java 卡的运行机制，业务逻辑，为以后的项目奠定了基础。

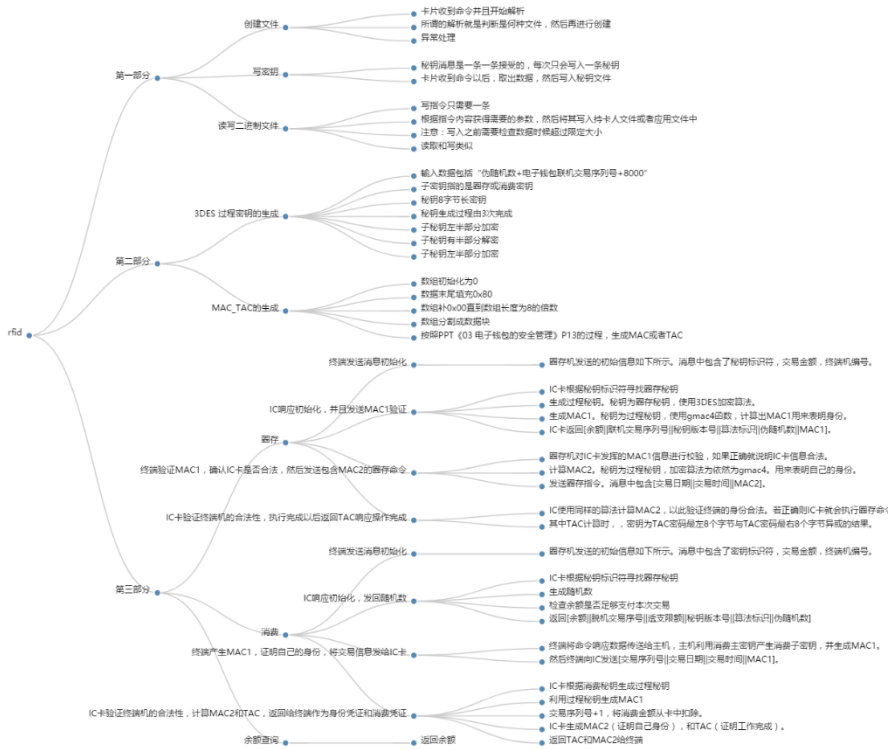
1.2 Java 智能卡发展现状

1、 **“接触式”智能卡**：必须放人“读出器”装置，才能使数据显示到 PC 监视器上。在这一情况下，微芯片必须与能够为数据解码的装置接触。

2、**“非接触式”智能卡**：芯片装有一根天线，在通过一个卡站时天线与基本装置通信，基本装置接着"读取"卡上的数据。

3、**双界面智能卡**:将接触式智能卡的优点与非接触式智能卡的优点结合到一个芯片上。例如，去医院看大夫时，在停车场大门口用你的智能卡对着门挥动以下，栏杆就打开让你进入停车场（非接触式）。进入医生办公室后，把你的智能卡插入读出器后，你的病历就显示到接待护士的荧光屏上（接触式）。

1.3 项目设计内容



1.4 项目设计的人员分工与进度

唐珊	part1 3-4 周
马怡平	part2 5-6 周
姚云龙	part3 7-8 周

第二章 电子钱包应用的设计过程

2.1. 电子钱包设计过程概述

我们看整个项目最重要的 process 函数，通过 apdu.setIncomingAndReceive 方法获取终端发送过来的消息，并将其分别放进 papdu 中各个参数中。接着执行 handleEvent 方法，对指令进行解析，跳转到不同的方法，执行结束之后，将 papdu.data 中的内容分写到 apdu_buffer 中，然后执行 setOutgoingAndSend 将缓冲区发回终端。整个项目的大流程宏观上就是如此。

接下来我们将一点一点对其进行剖析。

2.1.1. 设计阶段综述

整个电子钱包的设计过程分为 3 个阶段：

Step1：实现向电子钱包中写二进制数据，读二进制数据，写密钥的工作。

Step2：第二阶段实现了 3DES 算法以及生成 MAC 和 TAC 的 GMAC 算法。

Step3：第三阶段实现了电子钱包的圈存，花销以及查询功能。

2.1.2. 各个阶段互相之间的关系

第一个阶段的读写二进制数据让 Java 卡能够写入持卡人基本文件和应用基本文件功能，让卡片拥有一个身份信息。写密钥能够将特定的秘钥写入 Java 卡中，是 Java 卡能够实现身份认证功能和安全通信的基础，为第三阶段的工作打下基础。

第二个阶段的加密算法让信息能够以一种比较安全的状态保存下来，是 Java 卡片能够在第三阶段进行身份认证功能和安全通信的实现方法。

第三个阶段就是将理论功能实际实现的过程，通过第一阶段写入的秘钥信息，第二阶段实现的加密算法，通过特定的安全操作来实现钱包的圈存、消费和查询功能。

2.2. 分步讲述设计流程

2.2.1. 第一部分

借助 PPT，我们可以知道这一阶段有三个主要目的和如下解析步骤是：

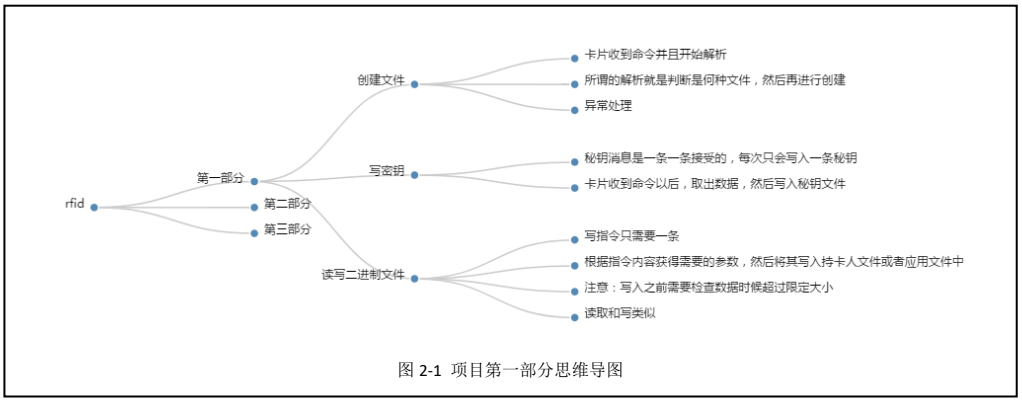


图 2-1 项目第一部分思维导图

前面我们已经知道了 Purse 整个大流程，接下来就是接着我们将执行命令解析工作。这个部分**在 handleEvent 中。利用不同的 ins 跳转到不同的地方**，执行相对应就行了。需要注意的是，在《04 电子钱包的功能》我们知道 load 和 purchase 的 init 函数对应的 ins 都为 0x50。因此，遇到 ins 等于 0x50 的情况的时候，我们还需要判断一下 p1，以此确定是 init_load 还是 init_purchase。

到这里我们已经剖开了第一层，接着往下剖析。

经过 handleEvent，我们会跳转到各种不同的方法，这里我们先实现本部分的文件创建 creat_file, write_key, write_read_bin。

这个时候我们将得到这样一张结构图谱。

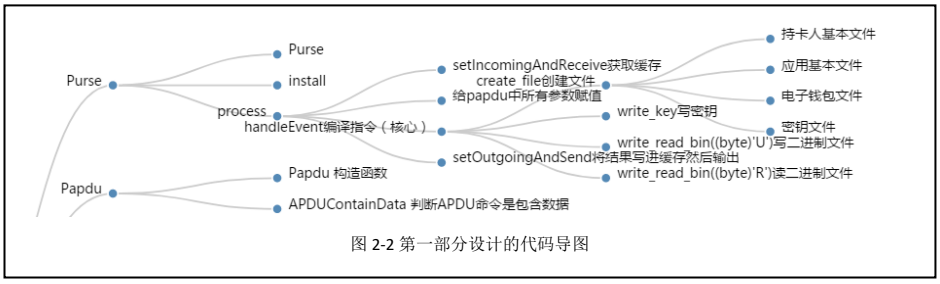


图 2-2 第一部分设计的代码导图

这时候我们就要看我们在开始分析的三个项目的实现步骤了。

首先是 creat_file，它的特点在《实验 2 文档》中已经给出来了，**有 4 种不同的文件可能会被执行创建，每种不同的文件用 AEF 作为标识符**。这里我们同样使用 switch 语句进行判断，然后分出不同的文件创建方法。

利用 ta 已经给了的 EP_file 方法，我们可以一窥这个方法的实现。它首先按照《实验 2 文档》中给定的各个参数，对调用 EP_file 的当前状态进行检验。即**①检验标志位是否合法**（比如 cla, lc），**②检查 EP_file 在之前是否已经被建立过**，然后**③检查 keyfile 是否存在**，因为我们在课程中已经知道，**keyfile 必须最先建立**，所以在以后的文件创建中，必须有这一步检测。**④合法以后就执行 new 操作。这里我为了简化逻辑，对标识为的检测进行了改写，即在 new 之前，对所有特殊的标志位全部检查一遍，简化了逻辑，加强了函数了确定性，也能有效的避免函数被误调用。**不过有点小问题是在文档中，“表文件创建（CREATE FILE）命令报文”部分 **p2 的值应该是一个不确定值而非 0x00**，word 中给的有问题。

```
private boolean EP_file() {
    // CLA: 识别电子卡
    if(papdu.cla != (byte)0x80)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //EP的ins必须为0xE0，校验以免误调用
    if(papdu.ins != condef.INS_CREATE_FILE)
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);

    //p1应该为0x00，p2应该为0x18
    if(papdu.p1 != (byte)0x00 || papdu.p2 != (byte)0x18)
        ISOException.throwIt(ISO7816.SW_WRONG_P1P2);

    // LC: Data Field之长度
    // 文件创建时文件信息长度为0x07
    if(papdu.lc != (byte)0x07)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    // 如果已经被分配
    if(EPfile != null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    //每个应用只能有一个KEY文件，且必须最先建立
    //《02 电子钱包的文件系统》P19
    if(keyfile == null)
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);

    EPfile = new EPFile(keyfile);

    return true;
}
```

图 2-3 EP_file 方法

好了，按照分析出来的结果，**按照上面的①~④的逻辑一步一步写，我们就可以轻而易举的将剩下的 Key_file, Card_file 以及 Person_file**。

接下来就是 write_key 环节，利用我们创建四种文件时候的方法我们同样想校验一遍①**所有的标志位，②检查 key_file，**如果没问题就利用 KeyFile 中的**③addkey 方法**，写入秘钥文件。不过需要注意的是，文档中，**write_key 部分的 lc 应该为 0x07，而非文档中的 0x15**，这个坑在我进行所有标识符匹配的时候造成一定的影响。

然后就是读写二进制文件。这一部分比较简单，**进行标识符匹配**以后，调用 Binary 类中的 read_binary 以及 write_bineary 方法即可。需要注意一点的是，在读写之前，**要根据 p1 的值判断操作的目标文件是 personfile 还是 cardfile，要保证它们不为 null**。另外，在写之前，我**们要保证数据控制在 0-255 之间。**

```
//写入前要先检测写入的数据是否超过文件限定的大小
//写入data长度应该在0-255之间
if(papdu.lc < 0x00 || papdu.lc > 0xFF)
    ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

/*
 * 持卡人或者卡不应该为空，如果不为空就可以开始写文件了
 */
//应用基本文件为0x16，应用文件不应该为空
if(papdu.p1 == (byte)0x16 && cardfile == null)
    ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
//持卡人基本文件为0x17，持卡人不应该为null
if(papdu.p1 == (byte)0x17 && personfile == null)
    ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

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

图 2-4 Write_bin 核心代码

好了，到这里基本的第一部分的项目任务就完成了。在填写完代码之后整个思绪都变得异常清晰，在操作中使用**严格的控制能够良好的避免错误，也能够方便我们整理思路**。让我们的撸代码的时候，思路更加清晰。

2.2.2. 第二部分

首先，还是借助 ppt 捋一下我们的思路，得到如下知识体系。

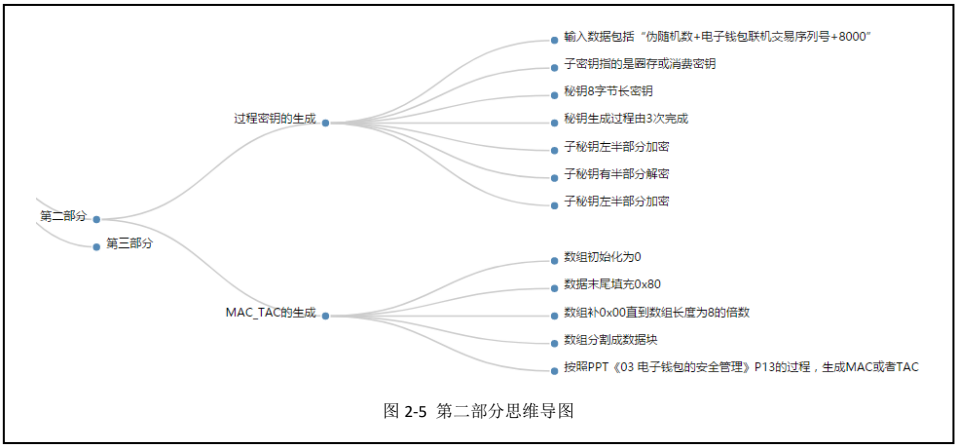


图 2-5 第二部分思维导图

在有了大概思路以后，我们开始阅读源代码。经过简单寻找，我们发现本次实验涉及的代码大多集中在 PenCipher.java 中，少部分在 BinaryFile 和 KeyFile 中。整理出如图 2-6 的结构。

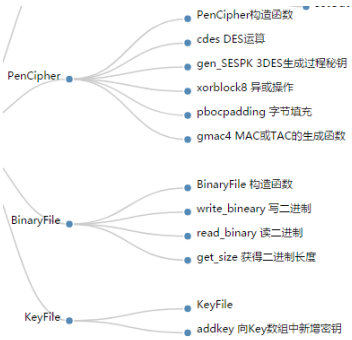


图 2-6 第二部分设计的代码结构导图

构造函数和基本的 DES 运算代码 TA 已经给好了，我们直接调用就可以。按照上面 PPT 中整理出来的知识点，我们开始实现余下的所有函数。

首先是 gen_SESPK 算法实现，很简单，调用三次 des 就行，只不过要**注意偏移量**，还要**注意是加密操作还是解密操作**。

```
//对data使用子秘钥的左半部分加密，写进data
cdes(key,(short)0,data,(short)0,dLen,data,(short)0,Cipher.MODE_ENCRYPT);

//对data使用子秘钥的右半部分解密，所以kOff为0x08，写进data，解密，所以模式为1
cdes(key,(short)8,data,(short)0,dLen,data,(short)0,Cipher.MODE_DECRYPT);

//对data使用子秘钥的左半部分再次加密，写进r结束
cdes(key,(short)0,data,(short)0,dLen,r,rOff,Cipher.MODE_ENCRYPT);
```

图 2-7 3DES 代码

xorblock8 和 pbocpadding 也很简单，报告中就不详细说明了。

MAC/TAC 的生成就有一点麻烦了，不过因为这一步需要用到的基本的函数我们都已经实现完成了。所以这一步只需要按照上面总结的思维导图，一步一步实现就好了。需要注意的是，**我们需要在这里自定义一个初始向量**。另外，整个生成过程也要对**《03 电子钱包的安全管理》P13 有较深的理解。**

```
short new_dl = pbocpadding(data,dl);

//初始向量，存在mac_tac中
//初始变量的值不唯一
byte[] mac_tac = {0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11};

//将这些数据分割为8字节长的数据块组。
short num = (short)(new_dl>>3); //切分成多少块

/*
 * 《03 电子钱包的管理》P13 mac_tac的生成
 * cdes参数：key 密钥；kOff 密钥的偏移量；data 所要进行加解密的数据；dOff 数据偏移量
 */
for(short i = 0; i < num; ++i) {
    xorblock8(mac_tac, data, (short)(i<<3));
    cdes(key,(short)0,mac_tac,(short)0,(short)8,mac_tac,(short)0,Cipher.MODE_ENC
}
//mac只有4个字节，所以只用取前四位进行比较
for(byte i = 0; i < 4; i++) {
    mac[i] = mac_tac[i];
}
```

图 2-8 gmac4 核心代码

另外，本次实验还稍微需要的是，在生成 MAC/TAC 过程中，**填充操作会不会造成越界，是不是需要给数据重新开辟空间。**（虽然我们在最终的检验中发现不会越界，重开空间反而会报错）。这个问题可以在后续中继续完善。

到这里，整个实验二就结束了。在填写完代码之后整个思绪都变得异常清晰，**严格按照 PPT 中的流程，我们就能够填写完本次实验的代码，整体并不难。**

2.2.3. 第三部分

前面两部分，我们知识管窥整个项目的一角。在第三部分，我们将真正深入的分析整个项目了。这一部分代码量多很多很多，实验思路也将更加曲折，调用的方法也将更多。因此需要我们对 TA 给的样例代码，有一个很深刻的了解，才能着手将整个项目填充完成。

老规矩，还是借助 ppt 和 word 实验文档捋一下我们的思路，得到如下知识体系。（这一次实验中，**word 实验文档扮演了很重要的角色**）。

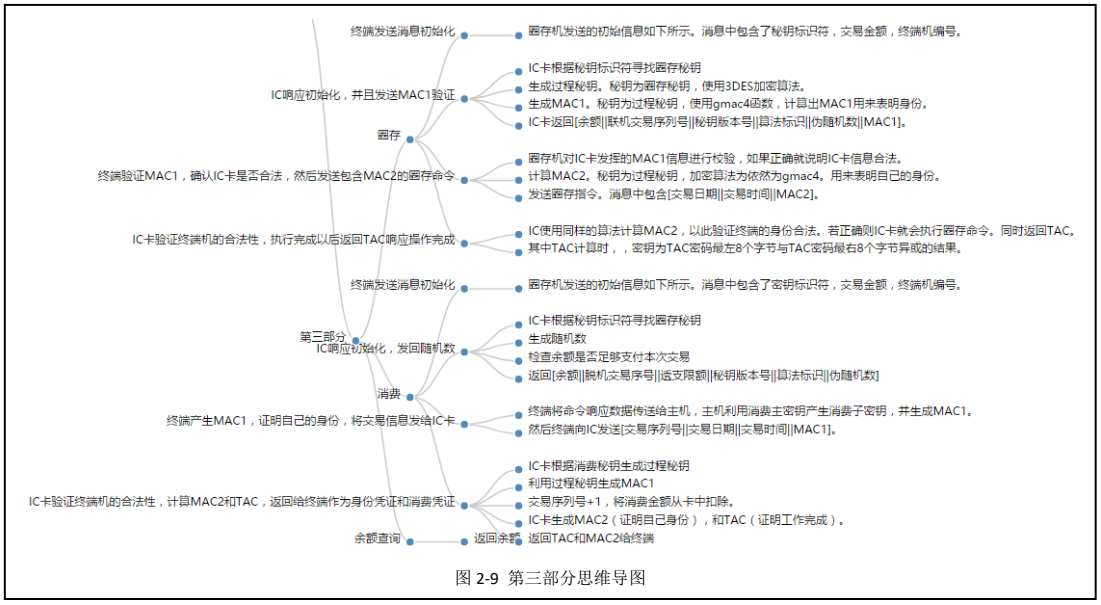


图 2-9 第三部分思维导图

不用说我们都能发现，整个思维简图和前两次相比，复杂了很多很多。我们先整理以下本部分代码中的已经存在的结构吧。

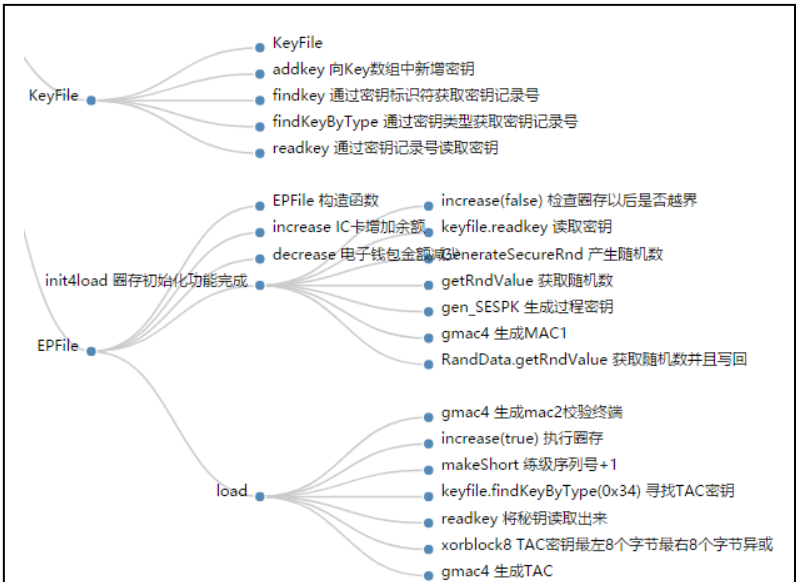


图 2-10 EP_file 中 init4load 以及 load 方法的流程结构

TA 已经给了我们所有的 init4load 和 load 算法的实现方法。我们必须将这两个函数的实现思想方法和 PPT 中分析出来的思维导图进行比，寻找两者之间的对应关系。才能有一个很深刻的了解，也才能够了解 TA 在设计整个电子钱包架构的时候的架设方法。经过细致对比，我们不难发现，**图 2-10 中 init4load 中整个流程刚好就是将图 2-9 中逻辑思维导图一步一步实现的过程。**

这还不够，我们接着对 init4load 和 load 进行进一步分解，同时监视每一个变量存放的信息。我们就会得到如下结构图（部分）。

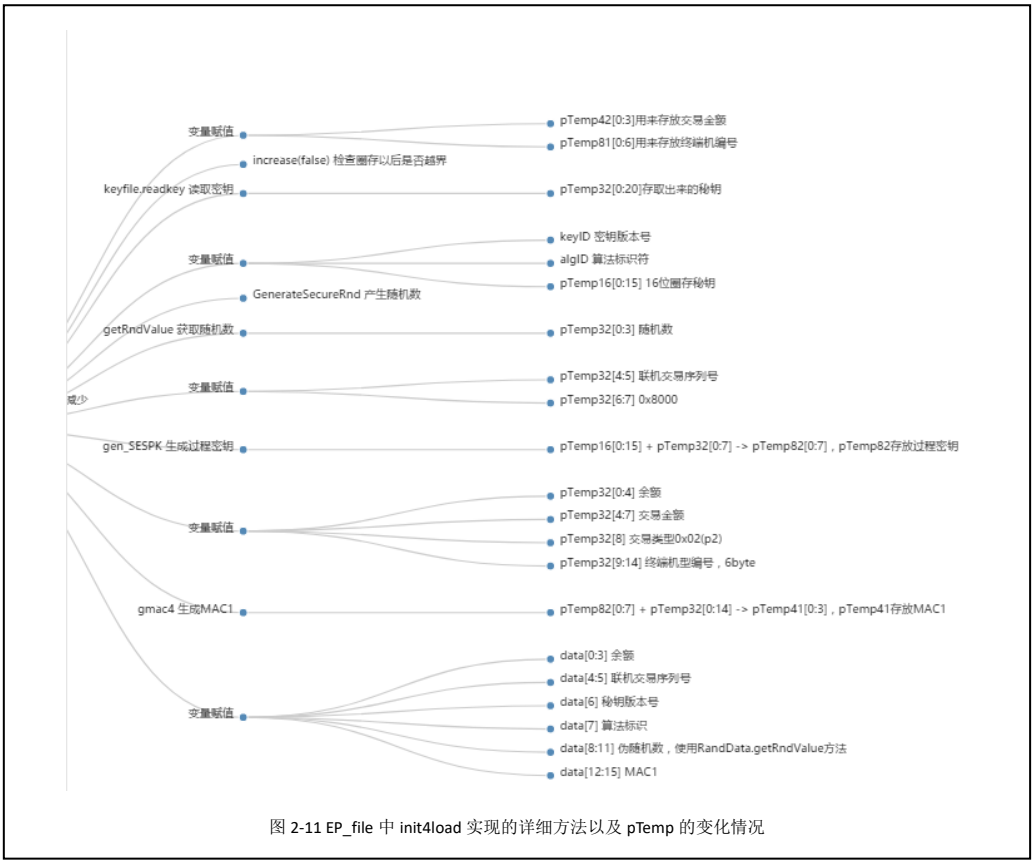


图 2-11 EP_file 中 init4load 实现的详细方法以及 pTemp 的变化情况

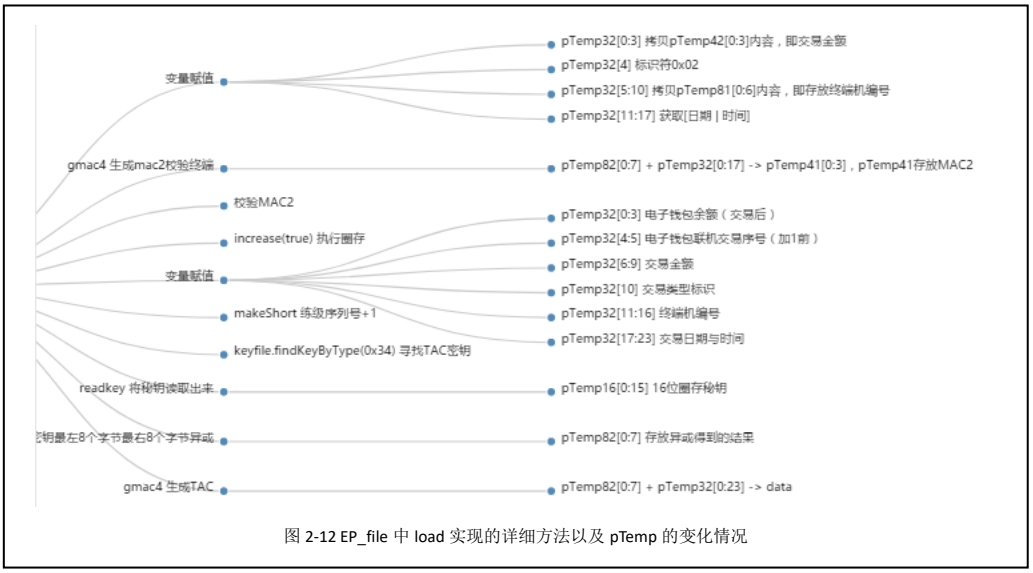


图 2-12 EP_file 中 load 实现的详细方法以及 pTemp 的变化情况

现在整理每一个步骤内，参数的变化情况。

不难发现**①每次刚进入函数调用的时候**，需要对一部分变量赋值（具体是哪些参数和传入的指令有关）。**②每次使用加密算法之前**，不管是生成过程密钥的 3DES 算法之前，还是再生成，要对 pTemp 进行赋值（因为它经常在加密的时候被用来作为输入数据）。记住这些特征，利用这些特征，我们将能够跑很方便的在 init4purchase 和 purchase 中**直接从 init4load 和 load 中抠代码。**

将图 2-11&2-12 中所有的变量经过仔细整理，我们就能够得到如图 2-13 中所示**所有 pTemp 的用途**了。借助这个**表、思维导图、init4load 以及 load 中的代码**，我们将能够快速完成本次代码的编写。

我们首先整理出 EPFile 类下，init4purchse, purchase, get_balance 的思维导图。剩下的就是模仿 init4load 和 load 的模式实现就行了。

```
/**
 * 下面数据是计算时需要用到的临时过程数据
 */
    //临时计算数据
    //4个字节的临时计算数据
    private byte[] pTemp41[0:3]; //mac或者tac
    private byte[] pTemp42[0:3]; //交易金额

    //8个字节的临时计算数据
    private byte[] pTemp81[0:5]; //终端机编号
    private byte[] pTemp82[0:7]; //过程秘钥

    //32个字节的临时计算数据
    private byte[] pTemp16[0:15];//圈存秘钥
    private byte[] pTemp32; //输入数据

    init4load中
    //pTemp42 交易金额
    //pTemp81 终端机编号
    //pTemp32 查询圈存秘钥返回的序列
    //pTemp16 所查找到的圈存密钥
    //pTemp32 随机数+EP_online[0:3]+0x8000（输入）
    //pTemp82 过程秘钥
    //pTemp32 EP_balance[0:3]+data[1:4]+0x02+data[5:10]
    //pTemp41 mac1

    load中
    //pTemp32 余额+0x02+终端机编号+交易日期+交易时间
    //pTemp41 mac2
    //pTemp32 余额（交易后）+电子钱包联机交易序号+交易金额+0x02+日期+时间
    //pTemp16 所查找到的圈存密钥
```

图 2-13 pTemp 的变化情况以及每种 pTemp 的不通用处

通过思维导图的引导，我们可以将图 2-9 的内容“翻译”成图 2-10 中的内容，然后进一步“翻译”成图 2-11 和 2-12 中的内容。并且我们很直观的发现了 init4load 中和 load 中的一些规律和明显的块级操作，这将为我们后期实现提供莫大的帮助。

我们通过思维导图分析和结构分析，可以很明显的发现我们需要实现的部分，和 init4load 以及 load 的源码中已经提供的部分之间存在很多相似，这种相似让我们在写代码的时候，能够直**接将那一块操作**（比如取随机数那一块操作）**复制粘贴到另一个地方**，极大地提高了我们的工作效率。只不过复制完以后需要整个检查一下业务逻辑是否有问题，变量是已经赋值，调用的方法是否还需要我们自己实现（比如 decrease）。

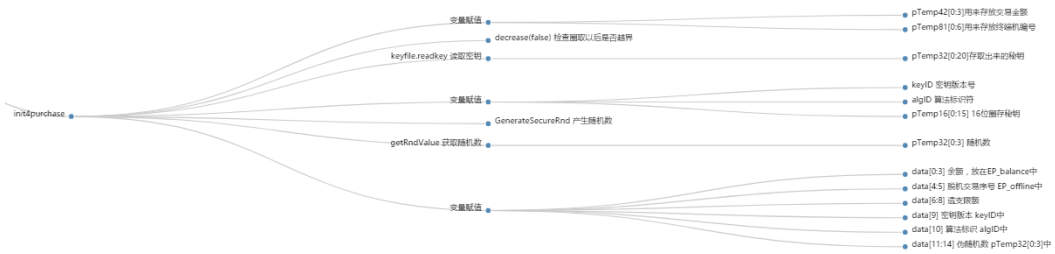


图 2-14 init4purchase 的流程结构及其翻译出来的实现详细方法

同样的方法，我们可以知道将 purchase 的思维导图如下。

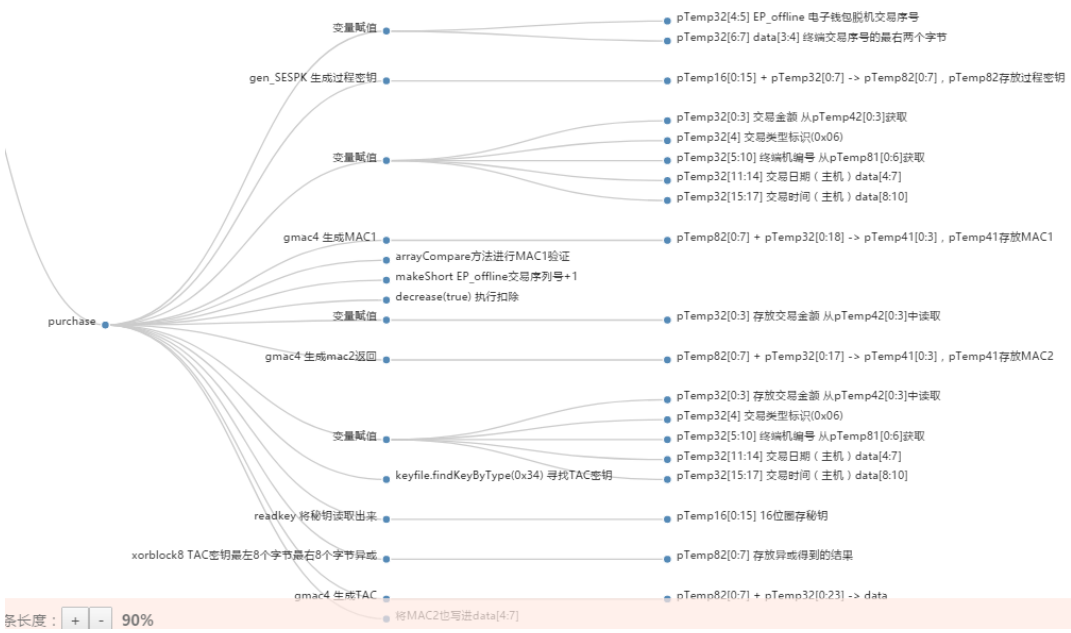


图 2-15 purchase 的流程结构及其翻译出来的实现详细方法

Get_balance 很简单，这里就不特殊分析了。不过，在实现过程中，在 purchase 中有一点问题，那就是我们的 pTemp 连续两次作为输入，并且没有置为 0，导致在最后一步生成 TAC 的时候，会抛出 6D00 的报错信息。我们最后决定重新开一个临时数组代替 pTemp32 的职责，终于成功返回了 TAC。

Purchase 代码（左边为 part1，右边为 part2）

```
/*
 * 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 purchase(byte[] data){
    short rc;
    /*
     * IC卡收到圈存命令后，用所查找到的密钥产生过程密钥
     * 其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
     * 密钥为所查找到的消费密钥
     *
     * 在init4load中，将pTemp32 4bytes用来存放随即变量，将其写进pTemp32[0:3]
     * 电子钱包脱机交易序号2bytes在EP_offline中，将其写进pTemp32[4:5]
     * 终端交易序号在data中，取出就行
     *
     * 对输入pTemp32[0:7]使用消费秘钥pTemp16[0:4]得到过程秘钥，写进pTemp82
     */

    //Util.arrayCopyNonAtomic(pTemp32, (short)0, pTemp32, (short)0, (short)4)
    Util.arrayCopyNonAtomic(EP_offline, (short)0, pTemp32, (short)4, (short)2
    Util.arrayCopyNonAtomic(data, (short)2, pTemp32, (short)6, (short)2);

    //生成过程秘钥
    EnCipher.gen_SESPK(pTemp16, pTemp32, (short)0, (short)8, pTemp82, (short)

    /*
     * IC卡利用过程密钥生成MAC1。
     * 输入数据：交易金额||交易类型标识(0x06)||终端机编号||交易日期（主机）||交易
     * 与消费命令传送的MAC1进行比较，如果相同，则MAC1有效，如果不一致就返回0x01
     */
    Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
    pTemp32[4] = (byte)0x06;
    Util.arrayCopyNonAtomic(pTemp81, (short)0, pTemp32, (short)5, (short)6);
    Util.arrayCopyNonAtomic(data, (short)4, pTemp32, (short)11, (short)7);
    EnCipher.gmac4(pTemp82, pTemp32, (short)0x12, pTemp41);

    if(Util.arrayCompare(data, (short)11, pTemp41, (short)0, (short)4) != (by
        return (short)0x01;    //不相同则返回1

     // IC卡将电子钱包脱机交易序号加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);

    //并且把电子钱包的余额减去交易金额
    rc = decrease(pTemp42, true);
    if(rc != (short)0)
        return 2;
```

```
    /*
     * 在进行上述处理后，IC卡利用过程密钥生成MAC2。
     * 其输入数据为交易金额，密码为过程密钥。
     */
    //byte[] temp = JCSystem.makeTransientByteArray((short)8, JCSystem.CLEAR_
    //Util.arrayCopyNonAtomic(pTemp16, (short)13, temp, (short)0, (short)8);/
    //pTemp32 = JCSystem.makeTransientByteArray((short)32, JCSystem.CLEAR_ON_
    Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
    EnCipher.gmac4(pTemp82, pTemp32, (short)0x4, pTemp41); //pTemp32被拓展

    Util.arrayCopyNonAtomic(pTemp41, (short)0, data, (short)4, (short)4); //i

    /*
     *  IC卡生成TAC码。
     * TAC码的生成方式和MAC码的生成方式一致。
     * 输入：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）||
     * 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。
     */
    //byte[] pTemp32 = JCSystem.makeTransientByteArray((short)32, JCSystem.CL
    Util.arrayCopyNonAtomic(pTemp42, (short)0, pTemp32, (short)0, (short)4);
    pTemp32[4] = 0x06;
    Util.arrayCopyNonAtomic(pTemp81, (short)0, pTemp32, (short)5, (short)6);
    Util.arrayCopyNonAtomic(data, (short)0, pTemp32, (short)11, (short)4);
    Util.arrayCopyNonAtomic(data, (short)4, pTemp32, (short)15, (short)7);

    short length, num;
    num = keyfile.findKeyByType((byte)0x34);//为什么只找0x34？？？？？
    length = keyfile.readkey(num, pTemp16);

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

    //取秘钥的前8位，写进pTemp82[0:7]，覆盖掉过程秘钥
    Util.arrayCopyNonAtomic(pTemp16, (short)5, pTemp82, (short)0, (short)8);/
    //key再搞搞事情，计算tac，写进data
    EnCipher.xorblock8(pTemp82, pTemp16, (short)13);//密钥左8位和右8位异或得到


    //得到tac同时返回tac给终端
    byte[] temp = JCSystem.makeTransientByteArray((short)8, JCSystem.CLEAR_ON
    Util.arrayCopyNonAtomic(pTemp16, (short)13, temp, (short)0, (short)8);//p

    //EnCipher.gmac4(pTemp82, pTemp32, (short)0x22, pTemp41);//得到tac，这个地
    EnCipher.gmac4(temp, pTemp32, (short)22, data);//得到tac直接复制给data返回
    //Util.arrayCopyNonAtomic(pTemp41, (short)0, data, (short)0, (short)4);
    //ISOException.throwIt((short) 0x1234);
    return 0;
}
```

```
init4purchase
//pTemp42 用来存放交易金额
//pTemp81 用来存放终端机编号
//pTemp32 查询圈存秘钥返回的序列
//pTemp16 所查找到的圈存密钥
//pTemp32 放随机数
```

```
purchase
//pTemp32 输入
//pTemp82; //过程秘钥
//pTemp41 mac1
//pTemp16 返回的秘钥序列
//pTemp82 tac圈存序列
//pTemp82 异或得到的
```

图 2-16 init4purchase 和 purchase 中参数变化情况

我们在写代码的时候也要记得及时校对参数是都是否使用正确，同时及时记录表中每一个参数的更新情况。图 2-16 是我记录的参数的变化情况

到这里，整个实验中需要我们填写的功能函数全部填写完成了。实验过程中，**了解业务逻辑**是一个很重要的过程，阅读源码，了解源码中**每一个函数的具体意义**，以及**每一个参数中，存放的信息有哪些**，这些都最好都在一个 txt 或者什么地方器记录下来，方便自己查阅和提取（毕竟一个没有 debug 的 IDE 插件，你不能要求什么，只能人脑 debug），另一方面，每实现一个功能，就尽可能在注释中写清楚这个部分的功能，输入信息，输出，同时写清楚关键信息在哪个参数的哪个位置位置，这样你在实现的过程中才不会头晕脑胀。也能够方便你最后 debug 的时候找错误信息从哪来。

[最终的逻辑流程图点这里查看]

第三章 电子钱包应用的功能测试

3.1. 功能测试概诉

整块功能测试主要分为两部分。

第一部分为 Java 卡的初始化的时候，文件的创建功能检测，二进制数据读写检测，秘钥写入检测。这部分的测试样例 TA 已经给好了，我们只需要将其改成.jcsh 后缀，执行/set-var path 之后，就能够进行检测了。

第二部分我们将模拟终端机和 IC 卡的通信，进行圈存和读取以及查询功能测试。查看整个系统是否能够正常运行。

3.2. 功能测试详细说明

3.2.1. 初始化功能测试

```
cm>  /set-var path "D:\Eclipse_cpp_workspace\purse"
cm>  purse_script
/select 1235318401
 => 00 A4 04 00 05 12 35 31 84 01 00
 (258862 nsec)
 <= 90 00
Status: No Error
```

输入第一行制定了脚本的工作路径，第二行指定了执行的脚本对象。首行选择了 1235318401 对象。

```
/send 80e00000073ffffffffffff
 => 80 E0 00 00 07 3F FF FF FF FF FF FF
 (497198 nsec)
 <= 90 00
Status: No Error
/send 80e000160738001effffffff
 => 80 E0 00 16 07 38 00 1E FF FF FF FF
 (583295 nsec)
 <= 90 00
Status: No Error
/send 80e00017073900370ffffffff
 => 80 E0 00 17 07 39 00 37 FF FF FF FF
 (497768 nsec)
 <= 90 00
Status: No Error
/send 80e00018072ffffffffffff
 => 80 E0 00 18 07 2F FF FF FF FF FF FF
 (1649 usec)
 <= 90 00
Status: No Error
```

接下来左图中显示分别是**秘钥建立**（80E0 开头，AEF 为 3F，该文件必须最先建立），下一行为**应用基本文件的文件建立**（AEF 为 38），**持卡人文件创建**（AEF 为 39），以及**电子钱包的建立**（AEF 为 2F）。到这里 create_file 方法检验完毕。全部返回 9000，无误。

```
/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....
 (480663 nsec)
 <= 90 00                                              ..
Status: No Error
/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.
 (409389 nsec)
 <= 90 00                                              ..
Status: No Error
/send 80d400061534f0f09000CEB726EDC01B793BC37DC09E2F768534
 => 80 D4 00 06 15 34 F0 F0 90 00 CE B7 26 ED C0 1B    .....
    79 3B C3 7D C0 9E 2F 76 85 34                      y;.}.
 (338118 nsec)
 <= 90 00                                              ..
Status: No Error
```

右图 3 行指令执行**秘钥写入**，分别为**消费密钥**（标识符 p2 为 07）、**圈存密钥**（标识符 p2 为 08）以及 **TAC 密钥**（标识符 p2 为 06）。写入成功，返回 9000。所以 write_key 部分检验完毕，没问题。

```
/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
 (421934 nsec)
 <= 90 00                                              ..
Status: No Error
/send 00D617003700005341 4D504C452E4341524 2E4144463100000001101029812180010110102
 => 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 00 05                ............
 (297065 nsec)
 <= 90 00                                              ..
Status: No Error
/send 00B0170037
 => 00 B0 17 00 37                                     ....7
 (791410 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
/send 00B0160037
 => 00 B0 16 00 1E                                     .....
 (340398 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
/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
 (421934 nsec)
 <= 90 00                                              ..
Status: No Error
/send 00D6170037000053414D504C452E434152442E4144463100000001101029812180010110102
 => 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 00 05                ............
 (297065 nsec)
 <= 90 00                                              ..
Status: No Error
/send 00B0170037
 => 00 B0 17 00 37                                     ....7
 (791410 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
```

左图 4 行分别测试了 Java 卡**基本文件**（p1 为 0x16）和**持卡人基本文件**（p1 位 0x16）的**二进制文件写入**（ins 为 0xD6）**读出**（ins 为 B0）。经过校验发现读写都能够正常执行，并且读入的数据和输出的数据完全一致，并且返回了 9000，故而我们认为 read_write_bin 部分检验完成，没问题。

因此我们认为本部分是成功的，没有问题的。

3.2.2. 3DES 算法和 MAC 生成算法检查

为了执行这一步，我们首先对 process 进行如下修改。

```
if(papdu.ins == (byte)0x62){

    PenCipher penci = new PenCipher();
    byte[] key = new byte[]{0x12, 0x12, 0x12, 0x12, 0x12, 0x12, 0x12, 0x12};
    byte[] data = new byte[]{0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22};
    byte[] mac = new byte[4];

    penci.gmac4(key, data, (short)8, mac);

    //ISOException.throwIt((short) mac.length);

    Util.arrayCopyNonAtomic(mac, (short)0, apdu_buffer, (short)0, (short) 4);

    apdu.setOutgoingAndSend((short)0, (short)4);//把缓冲区的数据返回给终端
}else{
```

输入测试样例“**/send 80620000073fffffffffffff**”， ins 为 0x62 跳转进这个分支，因为本次检验中，gmac4 传入的参数都是预定义好的，所以 62 后面参数无论怎么设置都可以。校验中，key 为 0x1212121212121212，输入数据为 0x2222222222222222。初始向量在 gmac4 中已经定义为 0x1111111111111111。

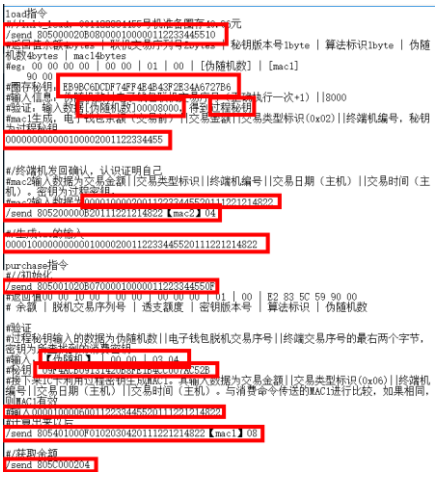
控制台（IC 卡）返回的数据	DES 算法工具返回的数据
18 00 10 11 01 02 98 12 18 00 10 00 00 00 00 00 .. 00 00 00 00 00 00 00 00 00 00 00 05 .. (355792 nsec) <= 90 00 .. Status: No Error /send 00B016001E => 00 B0 16 00 1E .. (299345 nsec) <= 62 64 00 22 33 33 00 01 03 01 00 01 20 01 08 17 bd 00 00 00 01 20 01 01 01 20 01 12 31 55 66 90 00 .. Status: No Error /send 00B0170037 => 00 B0 17 00 37 .. (282810 nsec) <= 00 00 53 41 4D 50 4C 45 2E 43 41 52 44 2E 41 44 .. 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 cm> /send 80620000073fffffffffffff => 80 62 00 00 07 3F FF FF FF FF FF FF .b (2992 usec) <= 79 5D B6 6C 90 00 y] Status: No Error	DES算法工具 加解密 MAC运算 XOR运算 关于 算法选择 ANSI-X9.9-MAC PBOC-DES-MAC PBOC-3DES-MAC 密钥 1212121212121212 初始向量 1111111111111111 数据 HEX ASCII 22222222222222280000000000000000 MAC 795DB66C0F3467BB 计算MAC

因为在本次实验中，我们**只取计算出来的 MAC 的前 4bytes 进行校验**。经过查看确实**均为 0x795DB66C。3DES 算法检测类似，篇幅所限就不赘述了，经过验证两边的结果仍然一致**。

故而我们可以认为，本部分的 3DES 算法和 MAC 生成算法完成，没有错误。

3.2.3. 圈存、消费、查询功能测试

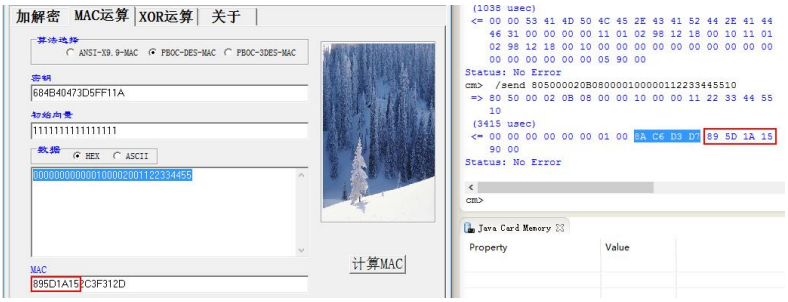
这个部分的测试最好新建一个 txt 用来存放能够预先设置好的值，以及一些必要的参数和说明。如下图所示。



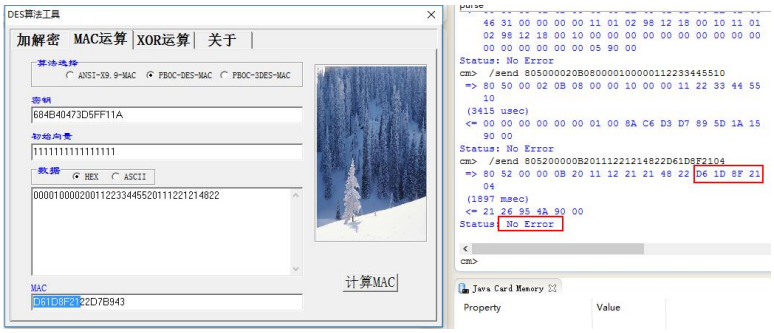
首先输入“/send 805000020B0800001000001122334455 10”，作为终端发送给 Java 卡的 **init4load 激活指令**。返回 00000000 00001000 8AC6D3D7 895D1A15，我们得到了 IC 返回的伪随机数 8AC6D3D7 和 MAC1 895D1A15。（前两个参数分别为电子钱包的余额以及本次交易金额）。

在 DES 算法工具中，使用圈存秘钥作为秘钥，[伪随机数||电子钱包联机交易序号（第一次为 0000）||0x8000]作为输入，得到过程秘钥。

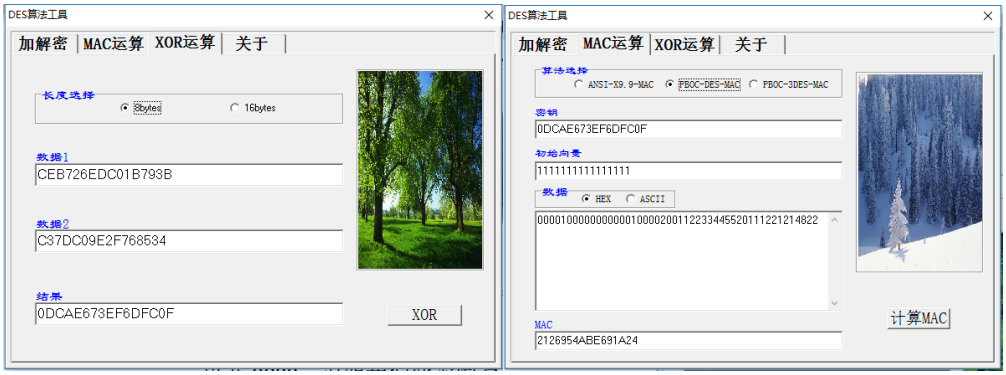
将过程秘钥复制到 MAC 运算中。输入 00000000000010000200112233445 5（[电子钱包余额（交易前）||交易金额||交易类型标识(0x02)||终端机编号]）。得到的 MAC1 和 Java 卡返回的 MAC1 一致，认证身份。



输入[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）]，输出得到 MAC2 为 0xD61D8F21，将其作为参数，替换掉“/send 805200000B20111221214822【mac2】04”中的 mac2。传送当前日期、时间和 MAC2 给 IC 卡证明终端身份。**让 Java 卡执行 load 指令进行圈存。**



IC 卡执行完毕并且返回 TAC 值。为了验证 TAC 是否正确，我们对其进行验证。



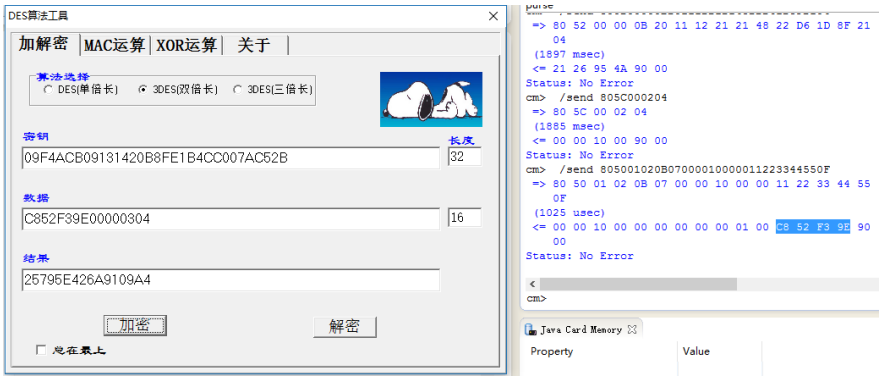
不难发现，利用使用[电子钱包余额（交易后）||电子钱包联机交易序号（加 1 前）||交易金额||交易类型标识||终端机编号||交易日期（主机）||交易时间（主机）]作为输入，TAC 密码最左 8 个字节与 TAC 密码最右 8 个字节异或的结果作为秘钥，得到的 TAC 值为 2126954A。和终端机返回的 TAC 一致。再加上结尾的 9000，说明我们此次圈存指令正确。

为了确保圈存的钱确实已经写入 Java 卡。我们执行“/send 805C000204”指令**查询余额**，这部分 PPT 中有说明，这里不赘述。返回 0x000010009000。说明我们余额已经变为 0x00001000，init4load 和 load 确实执行无误。

```
cm>  /send 805C000204
 => 80 5C 00 02 04
 (1885 msec)
 <= 00 00 10 00 90 00
Status: No Error
```

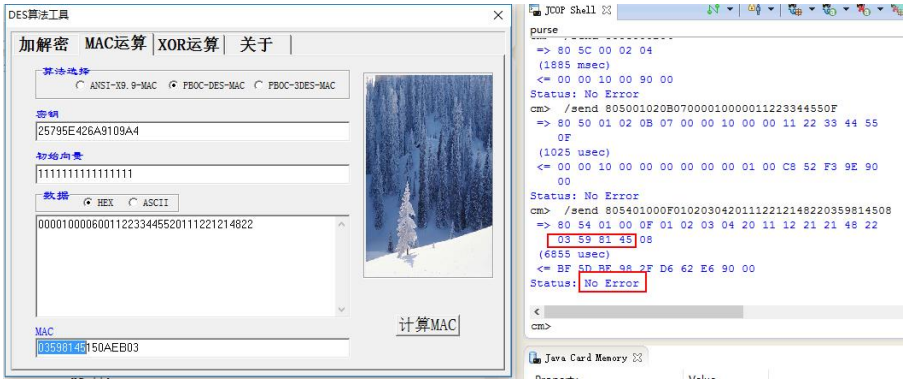
接下来执行消费指令，/send 805001020B0700001000011223344550F，data 中包含[密钥标识符||交易金额||终端机编号]。让 Java 卡**执行 init4purchase 方法，对本次消费进行预处理**。

得到[余额 || 脱机交易序号 || 透支限额 || 密钥版本 || 算法标识 || 伪随机数]。提取[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]，消费秘钥作为秘钥生成过程秘钥。

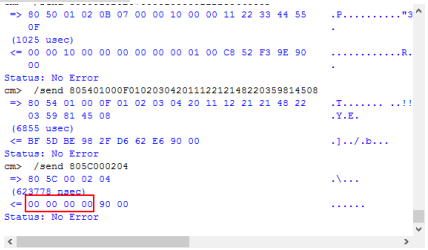


使用[交易金额||交易类型标识(0x06)||终端机编号||交易日期（主机）||交易时间（主机）]作为输入，过程秘钥作为秘钥，得到 MAC1。

从终端发送出去/send 805401000F0102030420111221214822【mac1】08，分别包含[终端交易序号||交易日期（主机）||交易时间（主机）|| MAC1]。作为输入，**执行 purchase 进行开始消费**。

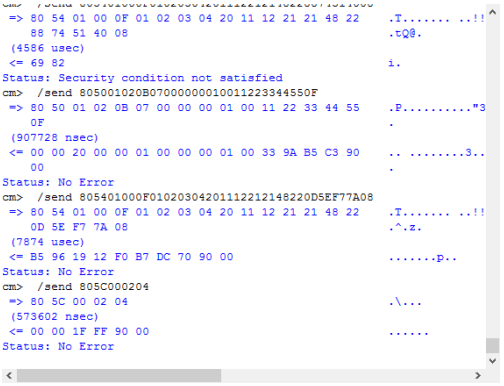


返回 MAC2+TAC+9000。MAC 计算我们已经验过很多次了，这里就不再验算了。再次执行“/send 805C000204”指令**查询余额。**



返回余额 0x00000000，之前圈存的前已经消费了。故而我们认为本次 **init_purchase 和 purchase 成功**。

另外，我们执行多次圈存和消费的指令进行测试，篇幅所限就不一一截图说明了。检验过程中我们**执行两次圈存指令**（均充值 0x00001000）和**一次消费指令**（消费金额为 0x00000001）。得到最终的结果为 0x00001FFF。**另外我们在测试过程中，有时候会误输入交易序号会有报错信息，**比如图中最上方那行指令。**更正后也能够正常执行**。这更加说明了我们的 **init4purchase、purchase、init4load、load 以及 get_balance 是没有问题的。**



到这里，整个实验所有实验要求已经全部实现并且验证完毕。

第四章 项目总结与感想

【唐珊】

在本项在这次大作业中，我主要参与完成了实验二，也就是电子钱包的文件系统这部分。

第一次接触 JavaCard 编程，而且上学期没有选到物联网这门课，所以刚开始做这个实验感觉很陌生不知从何下手。不过万事开头难，当理清一个思路后按照

步骤一步一步来问题也就解决了。

首先是装软件配置环境，说难也不难可相当麻烦，由于之前电脑已经配过 jdk 环境和新版本的 eclipse，但不符合实验要求，所以卸载重装这过程也费了一番功夫。

然后就是根据实验课的文档了解思路。实验二的主要目的概括为有三个：

创建文件。智能卡收到命令并且解析，以及一些异常处理。

写密钥。卡片收到命令以后，取出数据，然后写入秘钥文件，秘钥消息是一条一条接受的所以每次也只写一条。

读写二进制文件。根据指令内容获得需要的参数，然后将其写入持卡人文件或者应用文件中，读文件同理。

接下来就是写代码了，好在老师给了源码，可以参考和修改，中间也遇到了一些问题，比如说在写入二进制文件之前要检查数据是否超过限定大小。因为代码能力比较弱所以写和 debug 都挺费力的，有时候因为一点小问题耽误很久找不出来 bug，所幸有队友一起，帮忙提出问题，互相讨论，才使我们的实验能顺利进行，非常感激！

从这次实验中，主要学习了 COS，和智能卡通信 API，了解到日常生活中的校园卡等智能卡的工作原理，并且通过实验练习了如何实现，其实越是贴近生活的实验越是有趣，希望接下来的实验我们组也能互相学习共同进步！

【马怡平】

在本次项目中，实现了电子钱包安全管理模块的基本功能，包括：过程密钥和消息验证码的生成。过程密钥为输入数据（伪随机数+电子钱包联机交易序列号+8000）经过 Des 加密形成的交易特有的 8 字节长密钥，Mac 则是不停用过程密钥对消息数据进行加密和亦或操作。实现过程主要是按照实验步骤来填充函数，问题在于对于各种数据报文位数的控制，不同标志数据段的位数一不注意就会发生位溢出的状况导致 bug。最后实验的验证结果和验证软件结果一致。通过本次实验，我了解到电子钱包安全管理的具体的加密过程以及为什么这些加密信息可以和和特定交易信息对应；学习了 des 和 mac 生成原理，但对它们生成的具体操作原因还不甚清楚。

【姚云龙】

第三部分怼完简直想吐的心都有了。

这一个实验需要我们对整个项目代码有很深的了解，同时对 TA 已经给的源码一定要好好理解，并且仔细分析。我在实验之前光源码就细细的读了两遍。

第一遍读源码的时候，我读的很爽，因为整个项目的代码写的很整洁，代码封装的也很好，读起来很容易，但是我没有仔细考虑源码中每一个步骤和我们《实验 3 文档》中每一步骤的关系，而只是单纯看了代码的实现。所以在我看完代码，准备实现的时候，对项目的实现逻辑没什么头绪，而且本次实验中 EP_file 中，参数异常之多，同时还牵涉到了 KeyFile 以及 Randgenarator 类以及之前队友实现的功能中的一些类。所以不得不重新读源码。

第二遍读源码的过程中，我对照《实验 3 文档》，一步一步看 init4load 和 load

的实现过程，同时在一个 txt 中记录下来变量的变化情况，统计出来每个变量的使用情况以后。重新实现代码也就开始得心应手起来。

第三部分虽然代码很多，但是基本是从 init4load 和 load 中复制粘贴下来的。需要自己实现的方法也一般比较简单，计算时注意一点小心 coding 就行。

但是就算稳稳的开车，还是翻了。EP_file.purchase 最后计算 TAC 的时候怎么搞都不对，加上 Papdu.APDUContainData 这个地方不知道要修改。总是 debug 了很久才能找到并且修改问题。没错，这次实验一般时间都是在 debug，剩下的时间里面的一半时间在读源码，再剩下的时间里面，一半整理报告，一半 coding。有点像吐血。不过能够读完这种比较完整的 Java 卡的源代码，还是挺幸运的，毕竟代码风格真的很符合我的口味_(:３ 」∠)_

另外，由于篇幅限制，更加详细的实验报告（吐槽）可以在我的博客上看到。

http://blog.csdn.net/michael753951/article/details/70481546

基于本周周一夜间验收过程中，在进行文件创建过程中返回 P1P2 报错的情况。我们在 create_file 所指向的 4 个文件创建方法中，去掉了 p1 必须 0x00 的验证。同时还去掉了不必要的 ins 检测，避免出现某种奇奇怪怪的错误。如下图所示。

表- 文件创建（CREATE-FILE）命令报文

代码	值
CLA	80
INS	E0
P1	
P2	00
Lc	文件信息长度 07
Data	文件控制信息

龙云尧
P2 不确定，有 4 种可能

表- P2 值

密钥文件	应用基本文件	持卡人基本信息文件	电子钱包文件
00	16	17	18

表- DATA 域文件控制信息（AEF）

	命令报文数据域					
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

至此才能在真机中正确的运行，目前 p1p2 报错的具体原因还不太清楚，因为实验文档中并未提及创建文件中 p1 的意义。不过可能是因为不同的系统在指令的声明中，使用的 p1 不一定均为 0。