

# LKT5103A 32 位加密狗 开发手册

凌科芯安科技（北京）有限公司

## 版本记录

当前版本		V1.0.0	2018.10.10
原始版本		V1.0.0	2018.10.10
升级说明			
升级日期	版本号	新增内容	修改内容

## 联系凌科芯安

**公司名称：**凌科芯安科技（北京）有限公司

**办公地点：**北京市石景山区阜石路 166 号泽洋大厦 1601 室

**电话：**010 - 68864300

**传真：**010-68864300-604

## 目 录

第 1 章 LKT5103A 硬件特性.....	- 1 -
1.1 LKT5103A 参数.....	- 1 -
1.2 LKT5103A 产品特性.....	- 1 -
1.2.1 内部调试资源.....	- 1 -
1.2.2 硬件安全特性.....	- 1 -
1.2.3 硬件安全特性.....	- 1 -
1.2.4 应用领域.....	- 2 -
第 2 章 加密方案介绍.....	- 3 -
2.1 算法移植方案介绍.....	- 3 -
2.1.1 算法移植方案详解.....	- 3 -
2.1.2 算法移植方案特点.....	- 4 -
2.2 对比认证方案介绍.....	- 5 -
2.2.1 对比认证方案详解.....	- 5 -
2.2.2 对比认证方案特点.....	- 6 -
2.3 参数保护方案介绍.....	- 7 -
2.3.1 参数保护方案详解.....	- 7 -
2.3.2 参数保护方案特点.....	- 8 -
第 3 章 通讯协议.....	- 9 -
3.1 缩略语.....	- 9 -
3.2 指令格式.....	- 9 -
3.3 算法调用指令举例说明.....	- 10 -
3.4 通讯接口调用说明.....	- 11 -
3.4.1 Windows 系统下通讯接口调用.....	- 11 -
3.4.2 Linux 系统下通讯接口调用说明.....	- 12 -
第 4 章 加密方案开发说明.....	- 14 -
4.1 编译环境.....	- 14 -
4.2 算法例程中的函数接口说明.....	- 15 -
4.3 算法移植方案的设计开发.....	- 20 -
4.3.1 开发阶段准备工作.....	- 20 -

4.3.2 应用阶段实现流程.....	- 20 -
4.4 对比认证方案的设计开发.....	- 22 -
4.4.1 开发阶段准备工作.....	- 22 -
4.4.2 应用阶段实现流程.....	- 23 -
4.5 参数保护方案的设计开发.....	- 25 -
4.5.1 开发阶段准备工作.....	- 25 -
4.5.2 应用阶段实现流程.....	- 26 -
4.6 编程指南.....	- 28 -
4.6.1 头文件和库文件.....	- 28 -
4.6.2 数据类型.....	- 28 -
4.6.3 编程资源.....	- 28 -
4.6.4 常量使用.....	- 28 -
4.6.5 数据大小端问题.....	- 29 -
4.6.6 算法工程结构解析.....	- 29 -
4.6.7 算法工程代码功能解析.....	- 29 -
4.6.8 编程建议.....	- 31 -
4.6.9 调试技巧.....	- 32 -
4.6.10 注意事项.....	- 33 -
4.7 安全提示.....	- 33 -
4.7.1 不要预留读 NVM 区接口的通道.....	- 33 -
4.7.2 输入输出数据采用变化密文方式.....	- 34 -
4.7.3 尽量避免只在开机阶段进行一次对比认证.....	- 34 -
4.7.4 灵活设置代码陷阱.....	- 34 -
第 5 章 算法下载.....	- 35 -
5.1 连接 LKT5103A.....	- 35 -
5.2 复位 LKT5103A.....	- 36 -
5.3 下载算法.....	- 37 -
5.4 修改下载保护口令.....	- 38 -
5.5 发送算法指令.....	- 38 -
5.6 批量测试算法指令.....	- 39 -

仅限凌科芯安科技（北京）有限公司客户使用，违规必究！

## 第 1 章 LKT5103A 硬件特性

### 1.1 LKT5103A 参数

#### CPU

- 高性能ARM SC000 32位CPU内核
- 32位指令系统
- 低功耗的休眠模式

#### 通讯接口

- USB接口

#### 片上存储

- 128K-Bytes 程序存储区
- 64K-Bytes NVM数据存储区
- 32K-Bytes RAM

#### 操控特性

- 工作电压：4.0V-5.5V
- 工作温度：- 40 °C ~+ 85 °C

#### Flash 寿命

- 不低于10万次擦写次数或10年有效存储

#### 数据安全机制

- 硬件真随机数发生器
- 对称加密算法引擎，DES、3DES、AES128\192\256、RSA1024\1280、ECC(可定制)
- 哈希算法引擎，支持SHA1、SHA256
- FLASH存储区加密
- 优化安全布局

## 1.2 LKT5103A 产品特性

### 1.2.1 内部调试资源

- 采用 32 位 ARM SC000 安全芯片内核，内置 32 位加密操作系统
- 全球唯一硬件 ID 与管理编码
- 具有 128K 字节超大用户程序下载空间
- 64K 字节可定义安全性 NVM 数据存储区
- 32K 字节程序 RAM
- 支持单精度浮点运算
- 具有丰富的系统调用和开发接口，支持多种数学运算
- 硬件 AES/DES/TDES/RSA/ECC（可定制）等协处理器

### 1.2.2 硬件安全特性

- 传感器（电压，时钟，温度，光照）
- 过滤器（防止尖峰/毛刺）
- 独立的内部时钟
- （SFI）的检测机制
- 被动和主动盾牌
- 胶合逻辑（难以逆转工程师电路）
- 握手电路
- 高密度多层技术
- 具有金属屏蔽防护层，探测到外部攻击后内部数据自毁
- 总线和内存加密
- 虚拟地址（SW = 硬件地址！）
- 芯片防篡改设计，唯一序列号
- 硬件错误检测
- 真正的随机数发生器（RNG）
- 噪音的产生（对边信道攻击）
- 预硅功率分析

### 1.2.3 硬件安全特性

- 内部数据不可读取、拷贝



- 敏感信息进行加密（钥匙，别针）
- 双重执行的（如加密解密核查）
- 校验
- 验证程序流
- 不可预知的时序（如随机 NOP）
- 不能直接访问硬件平台，HAL（汇编），C
- 防止缓冲区溢出
- 防止错误的偏移
- 防火墙机制
- 异常计数器
- 执行验证码
- 归零的键和引脚

#### 1.2.4 应用领域

移动支付、电子商务/政务、控制访问、身份识别、控制器，安防监控、游戏机、汽车电子、平板电脑、机顶盒、DVR、路由器、交换机、仪器仪表等各种电子产品终端及应用。

## 第 2 章 加密方案介绍

### 2.1 算法移植方案介绍

#### 2.1.1 算法移植方案详解

LKT5103A 是凌科芯安科技（北京）有限公司行业内独家开发的以 32 位安全处理器为基础的具有高性能高安全性的加密产品（以下简称加密产品），算法移植方案具备方法型发明专利，专利号“ZL 2012 1 0546174.9”。用户将软件程序中一部分关键算法移植到加密产品中运行，如图 2-1 中所示，第一步先将代码 2 移植到加密产品中。用户采用标准 C 语言编写代码，通过 KEIL C 编译器编译并下载到加密产品中。在实际运行中，通过调用函数方式运行加密产品内的程序段，获得运行结果，并以此结果作为用户程序进一步运行的输入数据。因此加密产品成为了产品的一部分，而算法在加密产品内部运算，盗版商无法破解，从根本上杜绝了程序被破解的可能。

软件程序分为两个部分：一部分是在软件中，另一部分在加密产品中。当需要用到加密产品中的算法时软件向其发送指令，加密产品根据指令在内部运行程序并返回结果给软件。

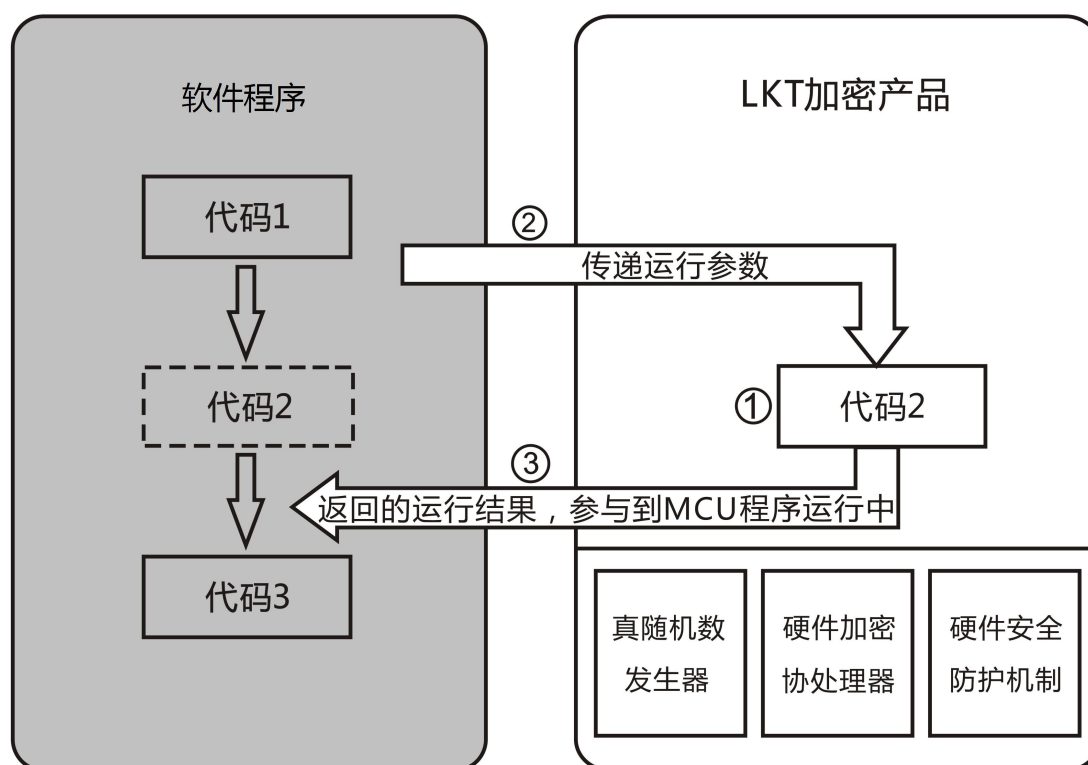


图 2-1：算法移植概念

## 2.1.2 算法移植方案特点

### 2.1.2.1 方案优点

最大限度的发挥了加密产品的安全特性，让加密产品与软件的功能结合使用，软件核心代码分别在软件和加密产品中存储运行，单独破解软件无法获取全部核心关键代码，以此提升整个系统的安全等级，让系统安全达到了质的飞跃。让盗版商和破解团队无从下手分析，从而无法盗版产品。

### 2.1.2.2 方案缺点

因为需要对加密产品算法结构与运行流程有清晰的了解，因此算法调试周期相对较长；加密产品为了防止破解分析不支持在线调试，只能在运行结束后将过程数据打印输出进行分析。

### 2.1.2.3 注意事项

选择移植的算法时，要兼顾考虑实时性、算法自身复杂度等问题。不要将整个软件程序都移植到加密产品中，也不要选择实时性要求过高的算法，同时也要保证算法运行结果是非线性的，不会被轻易分析出来。建议如下：

- 输入输出数据不能是一成不变的。
- 输入输出之间不能是线性变化的。
- 输出结果返回到软件中，建议作为程序下一步运算的重要参数。
- 输入输出数据的乱序迷惑处理，同时在输入输出数据中加入随机数增大分析难度。
- 线路数据最好以密文形式传递，防止被线路跟踪。

## 2.2 对比认证方案介绍

### 2.2.1 对比认证方案详解

对比认证方案的实现思路如图 2-2 所示。对比认证是基于国际上通用的对称加密算法（3DES、AES 等）对同一组随机数进行加密后，对结果进行比对判断，基于对称算法的特性，只有认证双方使用相同密钥，才可获得相同加密结果，以此来判别另一方身份是否合法。其安全性更多依赖于对称加密算法自身的安全强度以及密钥的安全存储，使用对称密钥对明文数据加密后再进行线路传输，防止线路攻击，保证无法从线路截取通讯数据攻击获得密钥。对比认证方案实现流程如下。软件移植 3DES 或 AES 等对称算法，软件与加密产品端在出厂发行阶段就预置相同的一组密钥。在运行阶段，软件产生随机数 RND 并将其发送给加密产品，然后两端使用预置的密钥同时对 RND 进行 3DES 加密生成密文 C1 和 C2，最后在软件端比较 C1 与 C2，相同则证明加密产品身份合法，软件程序继续运行；不同则证明加密产品身份非法，软件程序退出运行。

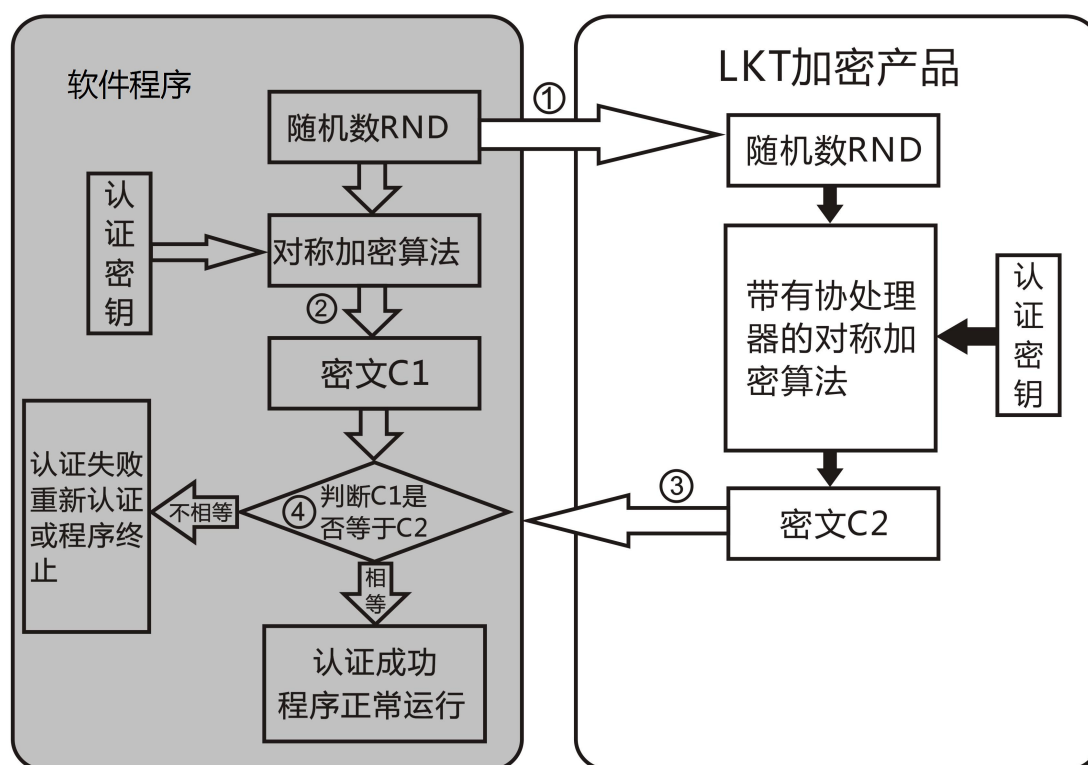


图 2-2：对比认证方案概念

## 2.2.2 对比认证方案特点

### 2.2.2.1 方案优点

该方案应用模式固定，调试简单，不需要对软件端原有程序做大的改动，也不需要了解加密产品内部运行流程。因此调试周期极短，研发投入很小。

### 2.2.2.2 方案缺点

该方案也是目前市面上一些加密产品的主流加密方案，安全性一般，只能防御非侵入式的线路攻击、重放等破解行为。但是通过对软件进行侵入式剖片攻击，可以读出软件端程序进行改写，有效绕过认证对比点。因为软件中的对比认证功能失效，所以在加密产品没有被破解的情况下，仍可完成盗版行为。

### 2.2.2.3 注意事项

由于软件端是不安全的，且需要存储一条用于认证的密钥。建议用户不要将该密钥值存储于连续地址（如单个数组中），可以放到内存不同位置，防止被轻易跟踪到。另外，建议用户每次使用认证密钥时，都经过一系列运算后得出密钥，这样真实密钥临时生成于 RAM 中，掉电即丢失，可有效防止防静态分析。

## 2.3 参数保护方案介绍

### 2.3.1 参数保护方案详解

参数保护方案的实现思路如图 2-3 所示。用户可以把软件中的一部分关键参数移植到加密产品中存储。在实际运行中，软件发送读回参数指令和随机数到加密产品端，后者也产生一组随机数，利用这两组随机数作为输入数据，结合预置的密钥对预存的关键参数进行加密生成密文，然后将密文参数和加密产品产生的随机数回传到软件端。此时，软件使用预置的解密密钥对密文参数进行解密操作，还原出关键参数 M，并将 M 作为程序进一步运行的输入数据参与到后续运行中。因为加密产品存储了软件中的关键数据，因此也成了产品的一部分，只对软件进行破解攻击，不能获取到完整的参数，而缺失的关键参数存储于加密产品中，可有效防止破解，从而起到了有效防护。加密产品的引入，虽然无法阻止盗版商对软件程序进行破解攻击，但通过对软件原有关键参数的有效保护，杜绝了盗版商的抄板行为。

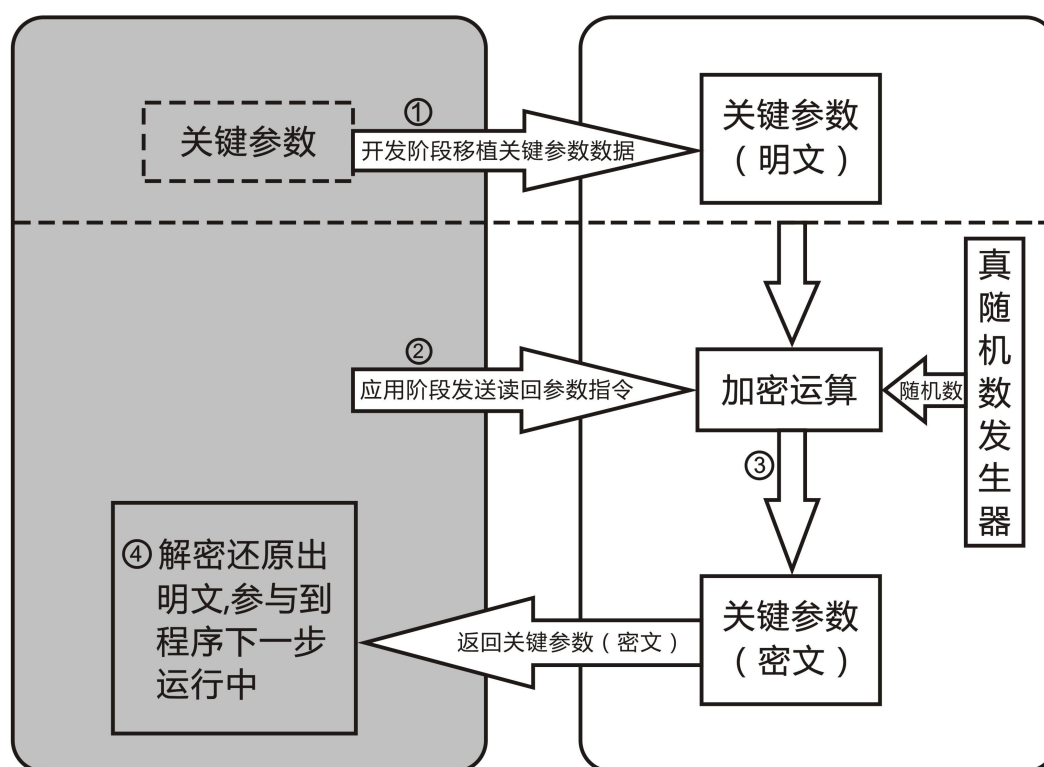


图 2-3：参数保护方案概念

## 2.3.2 参数保护方案特点

### 2.3.2.1 方案优点

该方案应用模式支持用户自定义,包括加密算法的选择和加密模式的设定。安全性仅次于算法移植方案,高于市面上常见的对比认证方案,调试简单,不需要对软件端原有程序做太大的改动,虽然需要了解加密产品内部运行流程,但不需要做太多编程改动,只需了解基本的底层接口应用方式。调试周期介于算法移植和对比认证之间,研发投入相对较小。盗版商没法直接找到比对点,程序执行中没有绝对的对错,而是会影响运行效果。综合比较,其开发难度小于算法移植方案,安全等级高于对比认证。该方案适用于实时性要求高,软件找不到合适的算法进行移植,但又需要高安全防护的项目。

### 2.3.2.2 方案缺点

因为没有像对比认证方案那样在软件端出现比较明显的比对点,所以该方案安全性相较于对比认证有所提升,可以防御非侵入式的线路攻击、重放等破解行为。但无法达到算法移植方案那样高的安全等级。

### 2.3.2.3 注意事项

存储于加密产品中的关键参数可以保证存储安全,当经线路传回到软件端时,要进行线路加扰处理,软件和加密产品端都产生随机数并参与到加密产品内部关键数据的加密过程中,可保证线路数据以随机密文方式传输,切忌使用固定的明文或者固定的密文进行传输。由于软件端是不安全的,且需要存储一条用于解密还原重要参数的密钥,建议用户不要将该密钥值存储于连续地址(如单个数组中),可以放到内存不同位置,防止被轻易跟踪到。

## 第3章 通讯协议

### 3.1 缩略语

APDU	应用协议数据单元
ATR	复位应答
CLA	类字节
CLK	时钟信号
GND	地，基准电压。
INS	指令字节
I/O	串行数据的输入/输出
Lc	在指令中发送的字节长度
Le	接收响应数据的字节长度
P1	参数1
P2	参数2
SW1	状态字节1
SW2	状态字节2
VCC	电源输入

表 3-1：缩略语

### 3.2 指令格式

命令由命令头和命令体两部分构成，如表3-2所示：

命令头				命令体		
CLA	INS	P1	P2	Lc	DATA	Le

表 3-2：指令结构

常见的指令构成形式，如表3-3所示：

形式	指令类型
CASE 1	CLA INS P1 P2
CASE 2	CLA INS P1 P2 Le
CASE3	CLA INS P1 P2 Lc Data
CASE 4	CLA INS P1 P2 Lc Data Le

表 3-3：指令类型



LKT5103A 调用算法常用指令，如表3-4所示：

命令头				命令体			描述
CLA	INS	P1	P2	Lc	DATA	Le	
00	84	00	00	无	无	01-10	取随机数
80	08	00	00	XX	XX...XX	无	算法调用指令

表3-4：常用指令

算法调用功能返回状态码的具体意义，如表3-5所示：

SW1	SW2	意义
90	00	正确执行
61	XX	有 XX 字节数据返回
67	00	长度错误
69	85	使用条件不满足
6A	80	参数数据域错误
6A	86	参数 P1,P2 错误
6D	00	INS 错误
6E	00	无效的 CLA
6F	00	数据无效

表 3-5：SW

### 3.3 算法调用指令举例说明

注：符号 “->” 代表向 LKT5103A 发送指令，符号 “<-” 代表 LKT5103A 返回数据。

调用算法前，需先向 LKT5103A 中下载算法。现以“LKT5103A 算法例程\LKT5103A\_Ap pDemo\prj”中的“fun\_1”函数为例说明。本文档中的参考指令全部为 16 进制数据。

发送指令举例：

-> 8008 0000 0A 01 112233445566778899      /\*\*发送调用算法命令\*\*/

<- EEDDCCBBAA99887766 9000      /\*\*返回数据以及 SW 值\*\*/

指令结构说明如表 3-6 所示。

命令头	LC	算法函数序号	传入算法函数中的参数
8008 0000	0A	01	112233445566778899

表 3-6：算法指令结构说明

### 3.4 通讯接口调用说明

#### 3.4.1 Windows 系统下通讯接口调用

LKT5103A 在 WIN7、WIN8、XP 下免驱。若要查看设备驱动，详见附录 A。通讯应用到的 4 个函数接口如下表所示，用户也可参考文件夹“Windows 操作例程”中的代码。

##### 3.4.1.1 连接设备

函数原型	int WINAPI EK_Open(int p);	
功能描述	连接 LKT5103A 设备	
参数	p	[IN] 传入整型数据即可
返回值	1：成功 其他值：失败	
备注	无	

表 3-7：连接函数

##### 3.4.1.2 复位操作

函数原型	int WINAPI EK_Reset(int *AtrLen, unsigned char *atr);	
功能描述	对 LKT5103A 进行复位操作	
参数	AtrLen	[OUT] 复位信息长度
	ATR	[OUT] 复位信息
返回值	0：成功 其他值：失败	
备注	每次连接之后都需要先做复位操作	

表 3-8：复位函数

##### 3.4.1.3 算法调用

函数原型	int WINAPI EK_Exchange_APDU(int CmdLen, unsigned char *cmd, int *ResLen, unsigned char *ResBuf);	
功能描述	向 LKT5103A 发送 APDU 指令	
参数	CmdLen	[IN] APDU 指令长度
	cmd	[IN] APDU 指令
	ResLen	[OUT] 返回数据长度
	ResBuf	[OUT] 返回数据
返回值	0：成功 其他值：失败	
备注	APDU 指令均为 16 进制数据	

表 3-9：发送 APDU 指令函数

#### 3.4.1.4 断开设备

函数原型	int WINAPI EK_Close();
功能描述	断开 LKT5103A 设备
参数	无
返回值	0：成功 其他值：失败
备注	无

表 3-10：断开连接函数

#### 3.4.2 Linux 系统下通讯接口调用说明

LKT5103A 在 Linux 系统下免驱。但是必须要将驱动模式切换为 U 盘模式，切换指令详见“\LKT5103A 开发套件\Linux 操作例程”路径下的“驱动切换.txt”文档，切换方法详见第 6 章。通讯应用到的 4 个函数接口如下表所示，用户可参考文件夹“Linux 操作例程”中的代码。

##### 3.4.2.1 连接设备

函数原型	int LKSC_Open(char * SC_Path, int *SC_Handle);	
功能描述	连接 LKT5103A 设备	
参数	SC_Path	[IN] LKT5103 设备的挂载路径
	SC_Handle	[IN] 文件标识符
返回值	0：成功 其他值：失败	
备注（重要）	SC_Path 在各系统下会有差异，如 Ubuntu 下可能为/dev/sg0，Centos 下可能是/dev/sdb，若 PC 上同时接入多个 USB 设备，则 LKT5103 的挂载节点还会发生变化，比如在 Ubuntu 下还可能会变为/dev/sg1。	

表 3-11：连接函数

##### 3.4.2.2 复位操作

函数原型	int LKSC_Reset( int SC_Handle, unsigned char *ATR, int * LenOfAtr);	
功能描述	对 LKT5103A 进行复位操作	
参数	SC_Handle	[IN] 文件标识符

	ATR	[OUT] 复位信息，16 进制数据
	LenOfAtr	[OUT] 复位信息长度
返回值	0：成功 其他值：失败	
备注（重要）	每次连接之后都需要先做复位操作	

表 3-12：复位函数

### 3.4.2.3 算法调用

函数原型	int LKSC_SendAPDU( int SC_Handle, unsigned char *Command, int LenOfCmd, unsigned char *ResBuf, int * LenOfRes);	
功能描述	向 LKT5103A 发送 APDU 指令	
参数	SC_Handle	[IN] 文件标识符
	Command	[IN] APDU 指令
	LenOfCmd	[IN] APDU 指令长度
	ResBuf	[OUT] 返回数据
	LenOfRes	[OUT] 返回数据长度
返回值	0：成功 其他值：失败	
备注	APDU 指令均为 16 进制数据	

表 3-13：发送 APDU 指令函数

### 3.4.2.4 断开设备

函数原型	int LKSC_Close(int SC_Handle);	
功能描述	断开 LKT5103A 设备	
参数	SC_Handle	[IN] 文件标识符
返回值	0：成功 其他值：失败	
备注	无	

表 3-14：断开连接函数

## 第 4 章 加密方案开发说明

本章介绍基于加密产品的三种常用安全方案的设计开发流程。用户可直接使用实例方案开发测试,也可按实际需求修改使用。完整的加密方案是由软件程序和加密产品程序共同组成的。对比认证和参数保护方案可以直接使用我司提供的 demo HEX 文件。算法移植需要用户在加密产品内部用 C 语言编程实现软件中被移植代码的功能,编译成功后将生成的 HEX 文件下载到加密产品中。

注意: HEX 文件就是被移植到加密产品内部的算法,关系到整个系统的安全,一定要妥善保管。

### 4.1 编译环境

可使用美国Keil Software公司出品KEIL C编译器开发编译。

编译器安装成功后,请直接打开我们提供的开发套件中的算法工程文件夹,后缀类型为uvproj 的就是工程文件,如图 4-1 所示。用户可按照注释索引进行调试。







 AppDemo.uvopt	2018/11/6 13:36	UVOPT 文件	8 KB
 AppDemo.uvproj	2018/11/6 13:36	UVPROJ 文件	16 KB
 AppDemo_LKT5103A.dep	2018/11/6 13:37	DEP 文件	2 KB
 AppDemo_uvopt.bak	2018/11/6 13:29	BAK 文件	57 KB
 AppDemo_uvproj.bak	2018/11/6 13:29	BAK 文件	15 KB
 ARM_APP.sc	2018/6/6 15:02	SC 文件	1 KB

图 4-1 : LKT5103A\_AppDemo 工程文件

## 4.2 算法例程中的函数接口说明

LKT5103A 提供 64K 字节的 NVM 数据存储区，从地址“0x0000”到“0xFFFFE”。

写 NVM 区函数如表 4-1 所示。

函数描述	说明
函数形式	void LK_WriteNvm (unsigned short addr, unsigned char * buf , unsigned char len);
参数 1	NVM 区地址
参数 2	写入的数据
参数 3	写入数据的长度

表 4-1：写 NVM 区

读 NVM 区函数如表 4-2 所示。

函数描述	说明
函数形式	void LK_ReadNvm (unsigned short addr, unsigned char * buf , unsigned char len);
参数 1	NVM 区地址
参数 2	存放读出的数据
参数 3	读出数据的长度

表 4-2：读 NVM 区

DES/3DES 写入密钥函数如表 4-3 所示。

函数描述	说明
函数形式	void LK_DesSetKey(unsigned char ByteLenOfKey,unsigned char *pKey);
参数 1	密钥长度（只能为 8 或者 16）
参数 2	密钥值

表 4-3：DES/3DES 写入密钥

DES/3DES 加密函数如表 4-4 所示。

函数描述	说明
函数形式	void LK_DesEncode(unsigned char *pIn, unsigned char *pOut);
参数 1	明文内容（只能为 8 字节）
参数 2	输出的密文值

表 4-4 : DES/3DES 加密

DES/3DES 解密函数如表 4-5 所示。

函数描述	说明
函数形式	void LK_DesDecode(unsigned char *pIn, unsigned char *pOut);
参数 1	需解密的密文值（只能为 8 字节）
参数 2	解密后的明文值

表 4-5 : DES/3DES 解密

获取随机数函数见表 4-6。

函数描述	说明
函数形式	void LK_GetRandom (unsigned char *buf, unsigned char len);
参数1	存放随机数据
参数 2	获取随机数的长度

表 4-6 : 获取随机数

AES 写入加密密钥函数如表 4-7 所示。

函数描述	说明
函数形式	void LK_AesSetKeyEnc(unsigned char ByteLenOfKey, unsigned char *pKey,unsigned char *pRoundKey);
参数 1	密钥长度（只能为 16、24、32 字节）
参数 2	密钥值
参数 3	用于运算的空间（240 字节）

表 4-7 : AES 写入加密密钥

AES 写入解密密钥函数如表 4-8 所示。

函数描述	说明
函数形式	void LK_AesSetKeyDec(unsigned char ByteLenOfKey, unsigned char *pKey,unsigned char *pRoundKey);
参数 1	密钥长度 ( 只能为 16、24、32 字节 )
参数 2	密钥值
参数 3	用于运算的空间 ( 240 字节 )

表 4-8 : AES 写入解密密钥

AES 加密函数如表 4-9 所示。

函数描述	说明
函数形式	void LK_AesEncode(unsigned char *pIn, unsigned char *pOut);
参数 1	明文内容 ( 只能为 16 字节 )
参数 2	输出的密文值

表 4-9 : AES 加密

AES 解密函数如表 4-10 所示。

函数描述	说明
函数形式	void LK_AesDecode(unsigned char *pIn, unsigned char *pOut);
参数 1	需解密的密文值 ( 只能为 16 字节 )
参数 2	解密后的明文值

表 4-10 : AES 解密

HASH 摘要 初始化函数如表 4-11 所示。

函数描述	说明
函数形式	void LK_HashInit(unsigned char HashType);
参数 1	选择的算法 ( 0 表示 SHA-1,1 表示 SHA-256 )

表 4-11 : HASH 摘要 初始化



HASH 摘要 过程数据块输入函数如表 4-12 所示。

函数描述	说明
函数形式	void LK_HashUpdate(unsigned char HashType,unsigned char *buf,unsigned char len);
参数 1	选择的算法 ( 0 表示 SHA-1,1 表示 SHA-256 )
参数 2	需要摘要的中间数据块
参数 3	数据块长度

表 4-12 : HASH 摘要 中间数据块输入

HASH 摘要 最后数据块输入函数如表 4-13 所示。

函数描述	说明
函数形式	void LK_HashLastUpdate(unsigned char HashType,unsigned char *buf,unsigned char len);
参数 1	选择的算法 ( 0 表示 SHA-1,1 表示 SHA-256 )
参数 2	需要摘要的最后一个数据块
参数 3	数据块长度

表 4-13 : HASH 摘要 最后数据块输入

HASH 摘要 生成摘要值函数如表 4-14 所示。

函数描述	说明
函数形式	void LK_HashGetDigest(unsigned char HashType,unsigned char * digest);
参数 1	选择的算法 ( 0 表示 SHA-1,1 表示 SHA-256 )
参数 2	摘要返回的结果

表 4-14 : HASH 摘要 生成摘要值

获取 ID 号 函数如表 4-15 所示。

函数描述	说明
函数形式	void LK_GetChipID (unsigned char *sn);
参数 1	输出的 ID 号(8 字节)

表 4-15 : 获取 ID 号

RSA 公钥运算函数如表 4-16 所示。

函数描述	说明
函数形式	unsigned short RSA_DecryptPubKeyFile(unsigned char* buf,unsigned short dat_len,unsigned short* ret_len, unsigned short ef_addr);
参数 1	数据处理缓冲区
参数 2	待处理数据长度
参数 3	RSA 运算后返回数据长度
参数 4	RSA 公钥存储的地址（NVM 区用户自定义）
返回值	SW 状态码，详见 LKCOS 智能操作系统参考手册

表 4-16：RSA 公钥运算

RSA 私钥运算函数如表 4-17 所示。

函数描述	说明
函数形式	unsigned short RSA_DecryptPrivateKeyFile(unsigned char *datBuf,unsigned short dat_len,unsigned short *ret_len, unsigned short ef_addr);
参数 1	数据处理缓冲区
参数 2	待处理数据长度
参数 3	RSA 运算后返回数据长度
参数 4	RSA 私钥存储的地址（NVM 区用户自定义）
返回值	SW 状态码，详见 LKCOS 智能操作系统参考手册

表 4-17：RSA 私钥运算

## 4.3 算法移植方案的设计开发

### 4.3.1 开发阶段准备工作

实现算法移植加密方案，需要预先完成如下工作：

- 向加密产品中移植算法代码 A ,请参考“\LKT5103A 算法例程\LKT5103A\_AppDemo\Src”路径下的 App\_fun.c 文件中的 Algorithm\_Transplantation 函数。
- 加密产品烧录 hex 的方法详见第 5 章

### 4.3.2 应用阶段实现流程

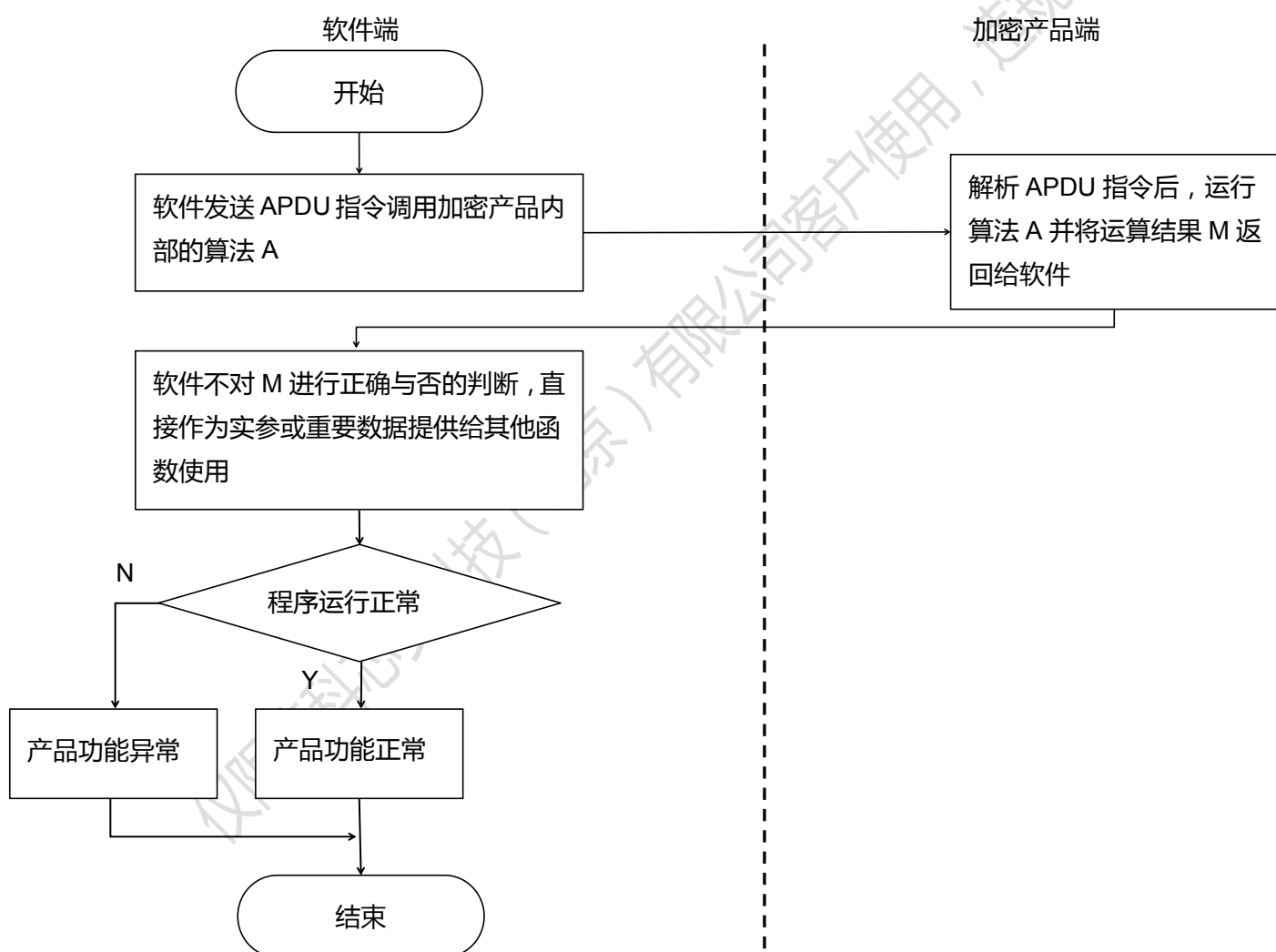


图 4-2：算法移植实现流程图

算法移植实现流程如图 4-2 所示，下面以示例算法进行说明。假设我们从软件中移植到加密产品中的程序功能如图 4-3 所示。一次完整的算法调用执行顺序如下所示：

```

函数名称: Algorithm Transplantation
函数参数: u8 xdata *pIn -----输入数据
          u8 xdata *pOut -----输出数据

函数功能: 方程求解
函数输出: y值

      x          0<= x <2
y= x*x+1        2<= x <6
  sqrt(x+1)      6<= x <10
  1/(x+1)        10< x <=20
  
```

图 4-3：移植到加密产品中的算法

- 软件发送 8008 0000 02 06 01 到加密产品，x=01。
- 加密产品返回 0000 803F 90 00, y=01。指令解析详见 3.3 节。
- 软件将 y 作为重要输入数据，参与到程序下一步运行当中。一次完整的算法调用执行成功。
- 出错处理。当 y 值错误，会引起后续程序异常，用户可在后续代码中做判断处理，也可设置错误上限次数，当达到上限值，程序锁死。

注意：用户在实际使用过程中，一定要保证线路传递数据为随机变化的密文，对明文数据 X 和 Y 实行线路保护，防止通过线路跟踪方式进行破解分析。随机数可作为数据变化的种子，用户可调用加密产品内部接口 LK\_GetRandom 产生，加密可采用加密产品自带的硬件 3DES 加密接口 Des\_encrypt，也可以自行移植 AES、XXTEA 等其他加密算法。

## 4.4 对比认证方案的设计开发

### 4.4.1 开发阶段准备工作

实现对比认证加密方案，需要预先完成如下工作,如图 4-4、4-5 所示。其中加密产品烧录 hex 的方法详见第 5 章，写入密钥 KEY2 的指令为 8008 0000 12 02 10 + KEY2 值，KEY2 的长度为 16 字节。

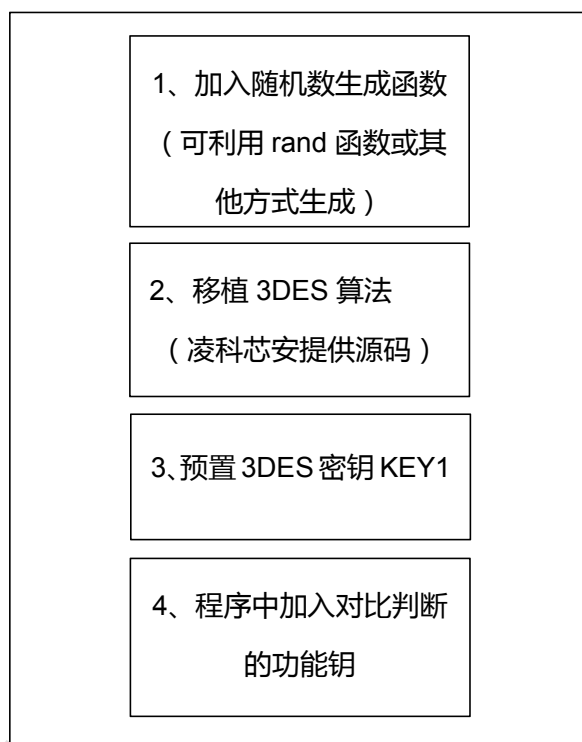


图 4-4：软件端准备工作



图 4-5：加密产品端准备工作

#### 4.4.2 应用阶段实现流程

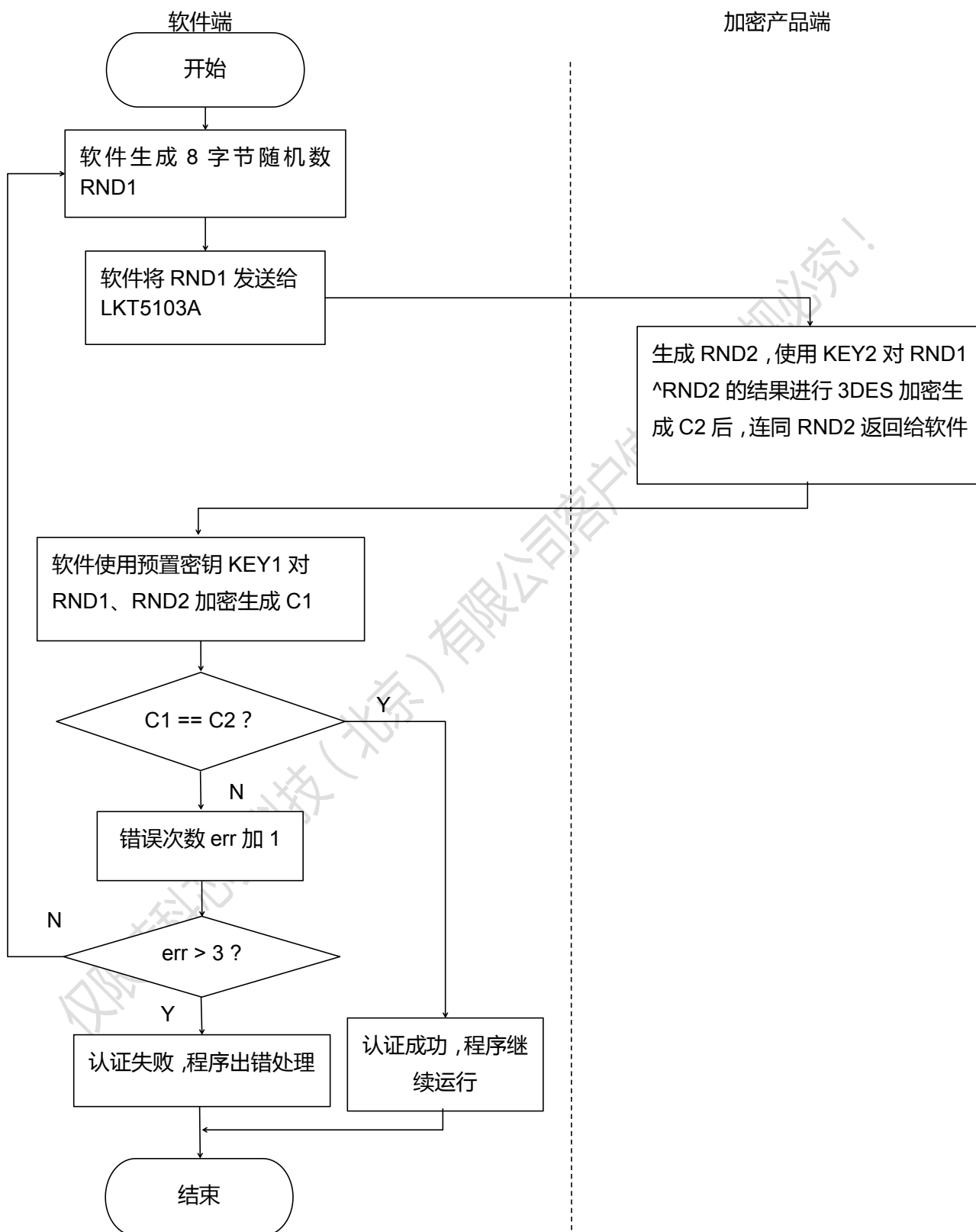


图 4-6：对比认证流程图

- 对比认证实现流程如图 4-6 所示，下面以实例进行说明。假设各参数如下：  
RND1 = 1122334455667788  
RND2 = 0102030405060708  
KEY1 = 11111111111111112222222222222222  
KEY2 = KEY1
- 软件发送 8008 0000 09 03 1122334455667788 到加密产品。
- 加密产品返回 0102030405060708 1735509EA362EDDE 9000，其中 C2 = 1735509EA362EDDE。
- 软件使用预置的密钥 KEY1 对 RND1 异或 RND2 的值进行 3DES 加密生成 C1，  
 $C1 = 3DES\_EN(RND1 \oplus RND2, KEY1)$ 。
- 软件对比 C1 与 C2 是否相等，如果相等，则认证成功，程序继续执行；如果 C1 不等于 C2，错误计数器 err 加 1，程序重新恢复至第 1 步，以此循环，当错误计数器为 3 时（错误次数 err 可由用户自行设定），认证失败，程序进行出错处理。

## 4.5 参数保护方案的设计开发

### 4.5.1 开发阶段准备工作

实现参数保护加密方案，需要预先完成如下工作,如图 4-7、4-8 所示。

- 软件端写入密钥 KEY1，用来解密重要数据。
- 加密产品烧录 hex 的方法详见第 5 章。
- 向加密产品写入密钥 KEY2，指令为 8008 0000 12 02 10 + KEY2 值。
- 向加密产品写入重要参数 D1 的指令为 8008 0000 09 04 + D1 值，D1 值为 8 字节。

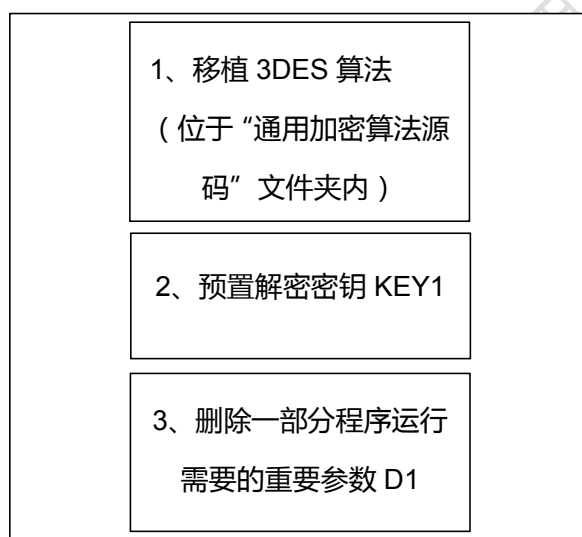


图 4-7：软件端准备工作

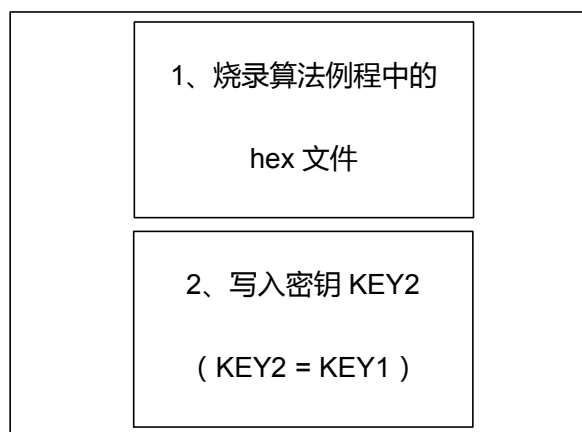


图 4-8：加密产品端准备工作



#### 4.5.2 应用阶段实现流程

软件端

加密产品端

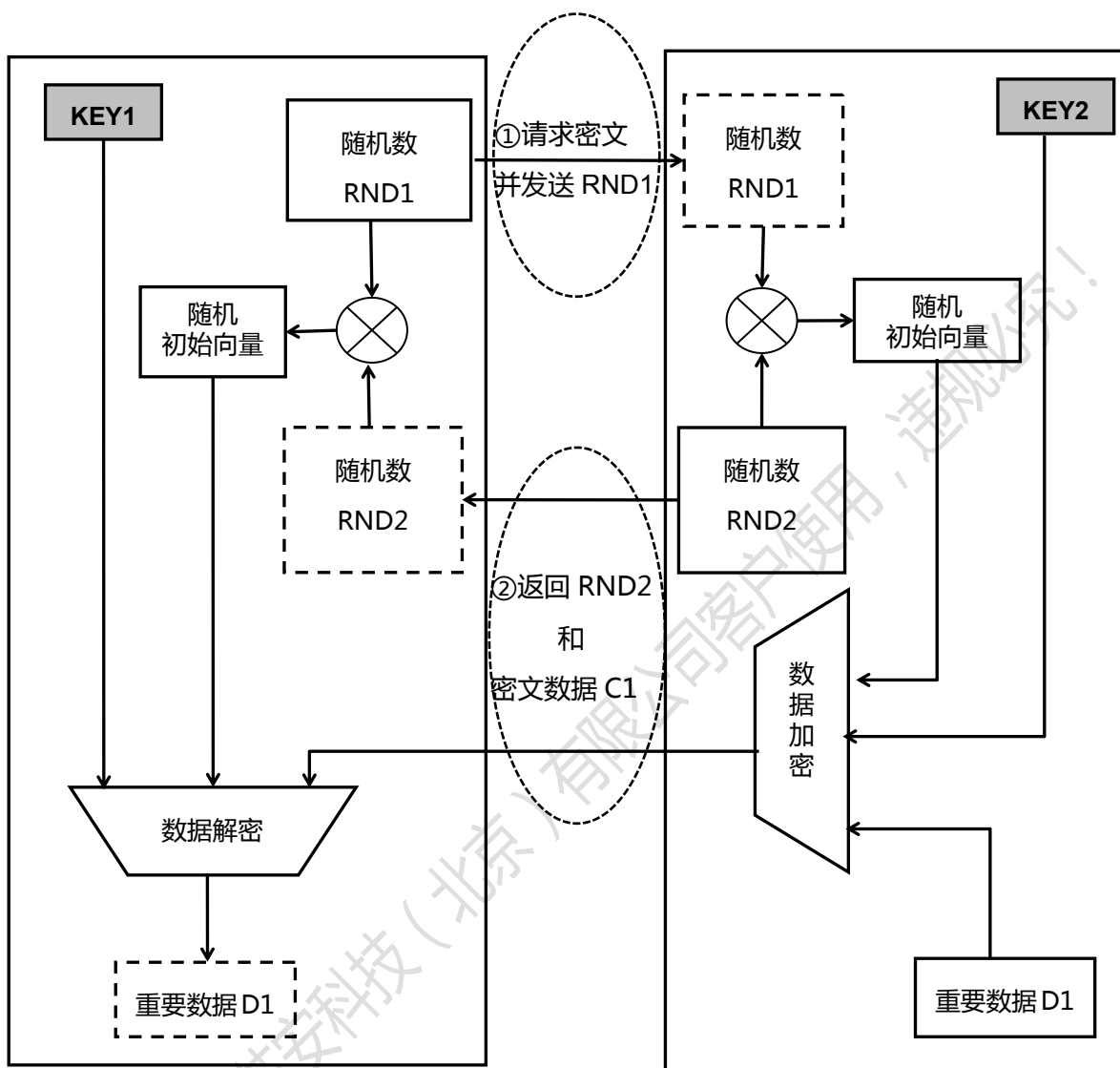


图 4-9：参数保护方案流程图

- 参数保护加密实现流程如图 4-9 所示，下面以实例进行说明。假设各参数如下：  
 RND1 = 1122334455667788  
 RND2 = 0102030405060708  
 D1 = 1234567812345678  
 KEY1 = 11111111111111112222222222222222  
 KEY2 = KEY1
- 软件发送 8008 0000 09 05 + 1122334455667788 ( RND1 ) 到加密产品。
- 加密产品返回 0102030405060708 ( RND2 ) + 8FF76B64C412F3C9 ( 密文 ) + 9000

( SW 状态码 ) , 其中  $C1 = 8FF76B64C412F3C9$ 。

$C1 = 3DES\_EN(D1 \oplus RND1 \oplus RND2, KEY2)$

( 使用密钥 KEY2 对 D1 异或 RND1 异或 RND2 的结果进行 3DES 加密后生成 C1 )。

- 软件使用预置的密钥 KEY1 对 C1 进行 3DES 解密生成数据 M1,再分别异或 RND1 和 RND2 , 还原出重要参数 D1。

$D1 = 3DES\_DE(C1, KEY1) \oplus RND1 \oplus RND2$

( 使用密钥 KEY1 对 C1 进行 3DES 解密后的结果再与 RND1 和 RND2 分别作异或操作得到 D1 )。

- 软件是否要对 D1 的正确性进行判断由用户自行决定 ( 可选择判断 D1 值, 或 D1 的区间范围是否正确 ), 最主要的是 D1 要作为重要参数, 参与到软件程序运行的各个阶段。

## 4.6 编程指南

### 4.6.1 头文件和库文件

库文件和头文件已经添加于算法工程文件中，用户无需做增删操作。

DEF\_TYPE.H 头文件中包含了变量的别名定义，例如 char 类型被定义为 u8。

LK\_API.H 头文件中包含了所以系统函数的声明。与 4.2 节对应一致。

LKTAPI\_5103A.LIB 库文件中包含了系统函数的定义。

### 4.6.2 数据类型

主要数据类型占用空间详见表 4-18

表 4-18：数据类型占用空间

类型	重定义名称	长度（字节）
unsigned char	u8	1
Signed char	s8	1
unsigned short	u16	2
signed int	s16	2
unsigned int	u32	4
--	--	--

### 4.6.3 编程资源

- CODE 区空间共 128K 字节，支持用户自定义编程。
- NVM 区 64K 字节，可用于存储数据参数，掉电不丢失。
- RAM：32K 字节。

### 4.6.4 常量使用

因为加密产品内部 RAM 空间有限，对于内部运行过程中不变的参数，用户可选择写入到

NVM 区，需要时读出使用。也可以将其定义为常量或常量数组。应用示例：

```
unsigned char code testbuf [6] = {1,2,3,4,5,6};
```

#### 4.6.5 数据大小端问题

大小端与 CPU 或软件的架构有关，在此不做赘述。LKT5103A 的数据存储为小端模式。

用户移植算法时要特别注意原有 CPU 存储模式与 LKT5103A 是否一致。

#### 4.6.6 算法工程结构解析

算法工程文件夹目录结构和主要文件功能描述详见表 4-19。

表 4-19：工程文件详解

文件夹名称	包含文件	功能
head	DEF_TYPE.h LK_API.h	系统函数声明和变量定义
Lib	LKT5103A_API.LIB	系统函数定义
Out	LKT5103A_AppDemo.hex .....	编译后生成的算法目标文件和过程文件，用于下载到加密产品中。
prj	AppDemo.uvproj .....	Keil 项目工程文件等
Src	App_Main.c	算法运行主函数文件
	App_fun.c	用户私有算法子函数等

#### 4.6.7 算法工程代码功能解析

加密产品的算法运行主函数定义在 App\_Main.c 文件中，算法示例中一些嵌套调用的子函数定义在 App\_fun.c 种，用户可以直接在上述两个文件中编程，也可将自己的 C 文件添加到工程中。但注意不要对入口函数名称和地址进行修改。

##### 4.6.7.1 主函数

App\_Main.c 文件中的 APP\_Command 为主函数。加密产品应用层算法的入口就是该函数，不能对该函数的名称和地址进行修改。

#### 4.6.7.2 输入输出缓冲区

加密产品输入输出缓冲区大小均为 0xFF 字节，因其属于被动运行模式，收到软件发来的指令数据后才会解析指令，运行并返回数据结果。其自身是不会主动向外部输出数据的。软件与 LKT5103A 进行数据交互需要利用 APDU 指令作为数据通道。主函数 APP\_Command 声明如下：

```
u32 APP_Command(u8 LenOfIn,u8 *pInBuf,u8 * LenOfOut,u8 *pOutBuf)。
```

其中，pInBuf 为输入缓冲区，pOutBuf 为输出缓冲区。

我们以开发套件算法中的取反功能为例进行说明。当软件向加密产品发送指令时，APDU 指令数据域中的内容会被传送到加密产品的 pInBuf 中，T=0 协议指令的第五字节开始为数据域内容。APDU 发送指令解析如表 4-20 所示。

表 4-20：软件向 LKT5103A 发送 APDU 指令并传入数据

通讯方向	协议类型	命令头	数据域	输入缓冲区地址	输入缓冲区大小
软件发送	T = 0	8008 0000	09 01 1122334455667788	pInBuf	FF

加密产品接收到指令后，会执行 APP\_Command 中的程序，当运行完毕后，将输出的数据放入 pOutBuf 中，加密产品自动向软件输出数据。APDU 接收数据解析如表 4-21 所示。

表 4-21：LKT5103A 向软件返回数据

通讯方向	协议类型	LKT5103A 输出缓冲区地址	输出缓冲区大小	输出给软件的数据内容	软件处理方式
软件	T = 0	pOutBuf	FF	6108	发送 00C0000008

接收				EEDDCCBBAA998877	直接使用
----	--	--	--	------------------	------

这里需要特殊说明的是,使用 T=0 协议的指令时,加密产品运行完毕后会先返回 61XX(XX 为待输出数据的长度),此时软件需要发送 00C0 0000 XX,将加密产品缓冲区内的数据取出。

#### 4.6.7.3 程序起始运行

采用 C 语言编程,程序按照 C 语言语法标准执行。每次软件发送 APDU 指令后,都将触发主函数 APP\_Command 的运行。因此,用户需在该函数内实现算法功能或嵌套调用。示例代码中调用了 App\_fun.c 文件中的一些子函数,仅为起到示例作用。用户可以选择直接在 App\_Main.c 或 App\_fun.c 中实现自定义功能,也可以新加入 C 文件和头文件,但一定要在 App\_Main.c 中做函数声明或引入头文件。

#### 4.6.7.4 程序终止

加密产品程序运行正常的终止有两种方式,即 return 0 或 return 1。

当加密产品检测到返回值为 1 时,会使用 T=0 协议将 pOutBuf 中的数据输出给软件(与软件发送 APDU 指令协议一致)。

当加密产品检测到返回值为 0 时,会使用 T=0 协议输出 2 字节数据 6A80 给软件(与软件发送 APDU 指令协议一致)。

当加密产品程序运行期间,不会对外输出任何数据。若程序进入死循环,则不会响应外部 APDU 指令,而加密产品内部无看门狗程序,此时只能采用硬件复位才可恢复正常。

#### 4.6.8 编程建议

- 函数嵌套不宜过多,避免造成堆栈溢出。
- 注意考虑大小端问题。
- 尽量减少第三方库文件的使用和头文件的引入。
- 通讯数据内容加入校验字节。

- 善于利用 NVM 区数据掉电不丢失的特性存储重要数据。
- 运行阶段减少对 NVM 区的写操作。
- 尽量不要直接套用 demo 算法，修改后再使用。
- 当算法单次调用的输入或输出数据超出 255 字节时，可设置全局标志位。例如：需要传入 1K 字节指令，运算并输出 1K 字节数据。可设置全局变量 Cnt\_Flag。将发送指令 1K 字节分 4 包数据传入加密产品中，每传一包数据，Cnt\_Flag 累加一次，不运行自定义算法。当 Cnt\_Flag 值为 4 时，运行正式算法。输出数据同理，将数据分 4 次拷贝到 pOutBuf 中，完成数据输出。

#### 4.6.9 调试技巧

因为安全机制等原因，加密产品不支持在线调试。当用户自定义编程调试出错时，无法动态实时监测过程变量和数据值，因此在一定程度上增加了调试难度。所以建议客户移植或者修改现有 demo 算法时，如果算法运行异常，尽量将较大程序切割成 N 个小算法模块，逐级调试，定位问题。下面介绍两种常用调试技巧。

##### 4.6.9.1 直接输出法

用户可将加密产品内部算法运行过程中产生的数据拷贝到输出缓冲区 pOutBuf 中，当完成一次算法调用后，软件可读出加密产品运行的所有过程数据，然后分析定位哪部分算法出现了问题。

该方法适用于加密产品内部算法代码量与复杂度适中，嵌套调用较少的程序，且算法运行过程中不会挂掉，因为一旦挂掉，加密产品无法输出任何数据，也就无从分析了。

##### 4.6.9.2 间接输出法

用户单独预留一个读 NVM 区数据的算法接口。运行正式算法时，可随时将各级嵌套调用过程中产生的数据分次写入到 NVM 区。在需要的时候，通过读 NVM 区接口获取到全部过程

数据，进行算法故障分析定位。

该方法适用于分析复杂且过程数据很多的算法程序，可以规避 pOutBuf 单次仅能传递 0xFF 字节的问题。尤其适用于调试过程中会挂掉的算法，因为写入的数据存入了 NVM 区，掉电不会丢失，用户仅需复位加密产品后，即可将算法发生异常之前的数据从 NVM 区中读回进行分析定位。

#### 4.6.10 注意事项

用户按标准方式定义全局变量。加密产品上电后，会自动调用 App\_Main.c 文件的 APP\_Init 函数完成全局变量初始化操作。例如定义全局变量 g\_var1 并初始化赋值为 0x77，正确的编程方法如下：

```
unsigned char g_var1 ;  
  
void APP_Init(void)  
{  
    g_var1 = 0x77;  
}
```

### 4.7 安全提示

以上章节对总体应用方案和调试技巧做了归纳总结，本节主要针对安全隐患的细节做出补充说明。

#### 4.7.1 不要预留读 NVM 区接口的通道

NVM 区一般会被用户定义为重要数据的存储区，对于破解商的反向分析可能会起到至关重要的作用。用户在调试阶段为了方便，可能会预留读取 NVM 区的接口，通过 APDU 指令将 NVM 区数据读取到外部。但实际量产应用阶段，一定要将通过 APDU 指令输出 NVM 区数据的通道关闭。即用户可以在算法内部调用读取 NVM 区的接口，但是不要将数据放入 pOutBuf 中向外部输出。



#### 4.7.2 输入输出数据采用变化密文方式

软件与加密产品通过通讯接口进行数据传递时，可以轻易的被破解商截获并进行分析。因此，软件与加密产品进行通讯数据交互时，要采用变化密文的方式。此处有两个含义，变化指的是明文数据自身要变化，如果无法做到，就要借助随机数参与，使明文每次都会变化；密文指的是要对明文进行加密处理，即使截获也无法直接分析数据功能含义。加密产品内部自带真随机数发生器与 3DES 加密算法硬件协处理器，并且为用户提供了编程调用接口。用户要尽可能利用起来这些良好的资源。这样，仅仅通过线路跟踪是无法分析数据的，通过线路重放也是无法轻易破解攻击软件的。

#### 4.7.3 尽量避免只在开机阶段进行一次对比认证

在现有的加密方案中，应用最广泛的就是开机阶段进行一次对比认证的方案。因此，破解商对该方案的破解也是最为熟悉。但只要适当改动升级该方案，也能一定程度上提高整体安全水平。例如将认证环节加入到软件程序运行的各个环节，可以在开机后 10 秒、1 分钟或者 1 小时内认证一次，增加认证次数，增加软件端反汇编后代码的分析难度。

#### 4.7.4 灵活设置代码陷阱

设置代码陷阱，当认证错误达到一定次数后，程序锁死，或者让功能处于部分异常的状态。还可以设置一些无效的对比点或者算法调用，参数调用。执行一些错误检查，当加密产品返回的认证数据无误，或者参数、算法运行结果错误时，程序继续运行，这样一来，当破解人员修改代码，程序即可判断到有人攻击新片，让程序锁死或出错。

## 第 5 章 算法下载

### 5.1 连接 LKT5103A

操作步骤如下：

- 打开 LCS SAM 软件，在“通讯设置”选项页中，“通信方式”选择“DLL”。
- 在弹出的对话框中，选择开发套件 tool 文件夹目录下的“LKT5103A.DLL”动态库，如图 5-1 所示。
- 如果设备已正常接入，点击“连接”，会显示连接成功及时间，如图 5-2 所示。

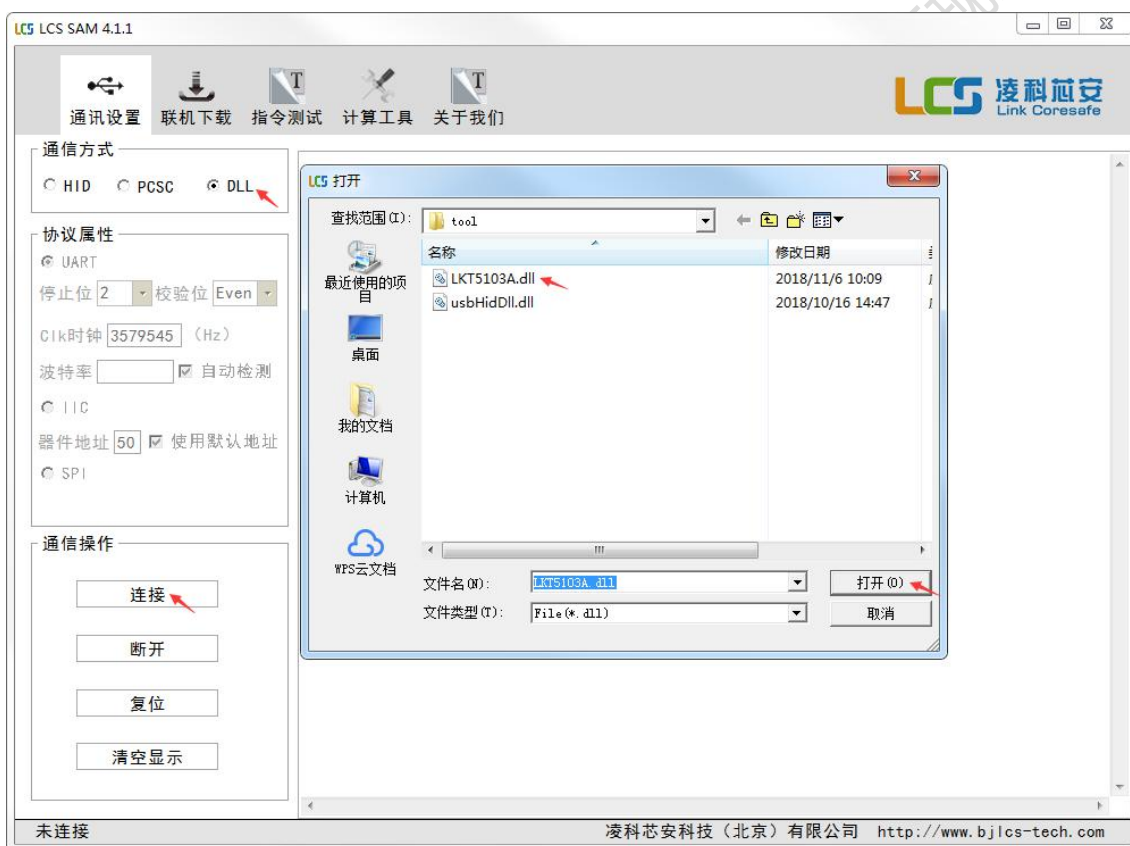


图 5-1：连接

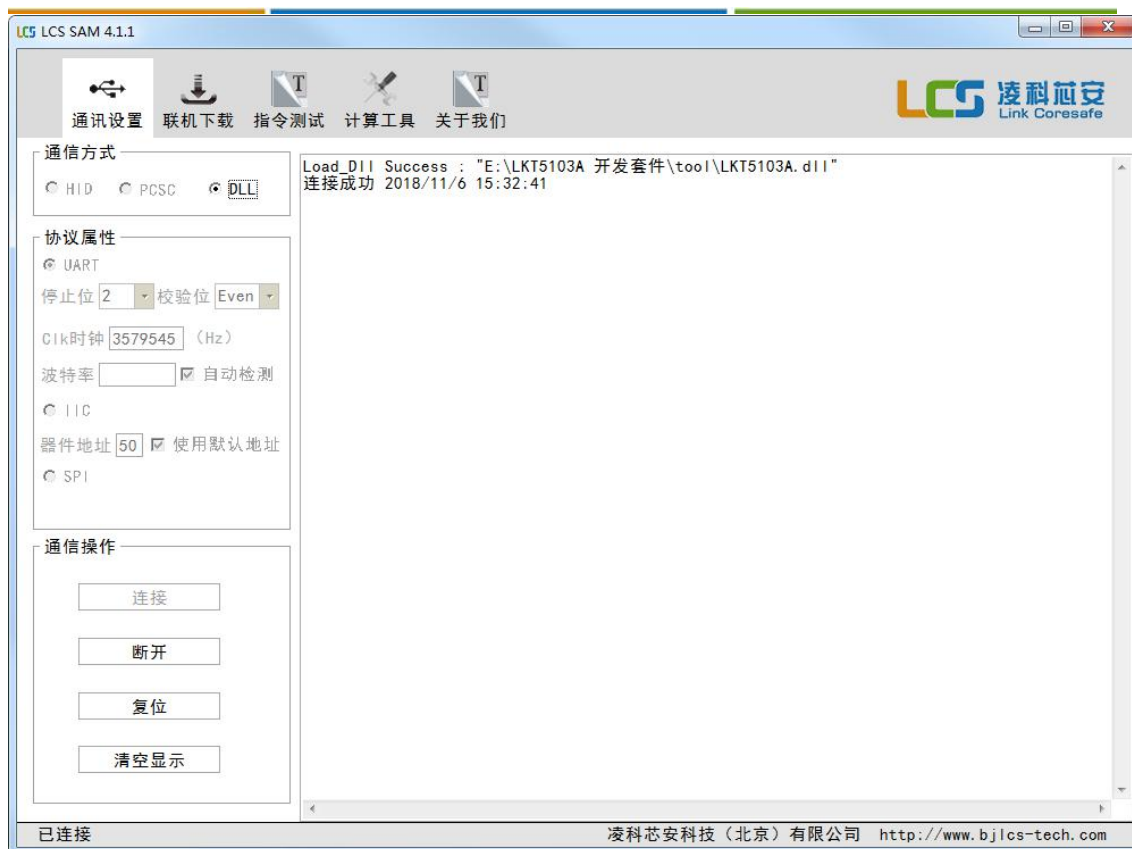


图 5-2 : 连接成功

## 5.2 复位 LKT5103A

连接成功后，点击“复位”按钮，即可打印出 LKT5103A 的复位信息，如图 5-3 所示。

LKT5103A 连接成功后，必须要先做一次复位操作，才能进行正常的指令交互和算法调用。

复位成功后，会返回“3B 6D”开头的 17 字节复位信息。LKT5103A 的最后 8 字节是唯一 ID 号，不会出现重复的情况。复位操作可对加密锁进行初始化，当算法运行异常时，可选择进行复位操作，重新调用算法。

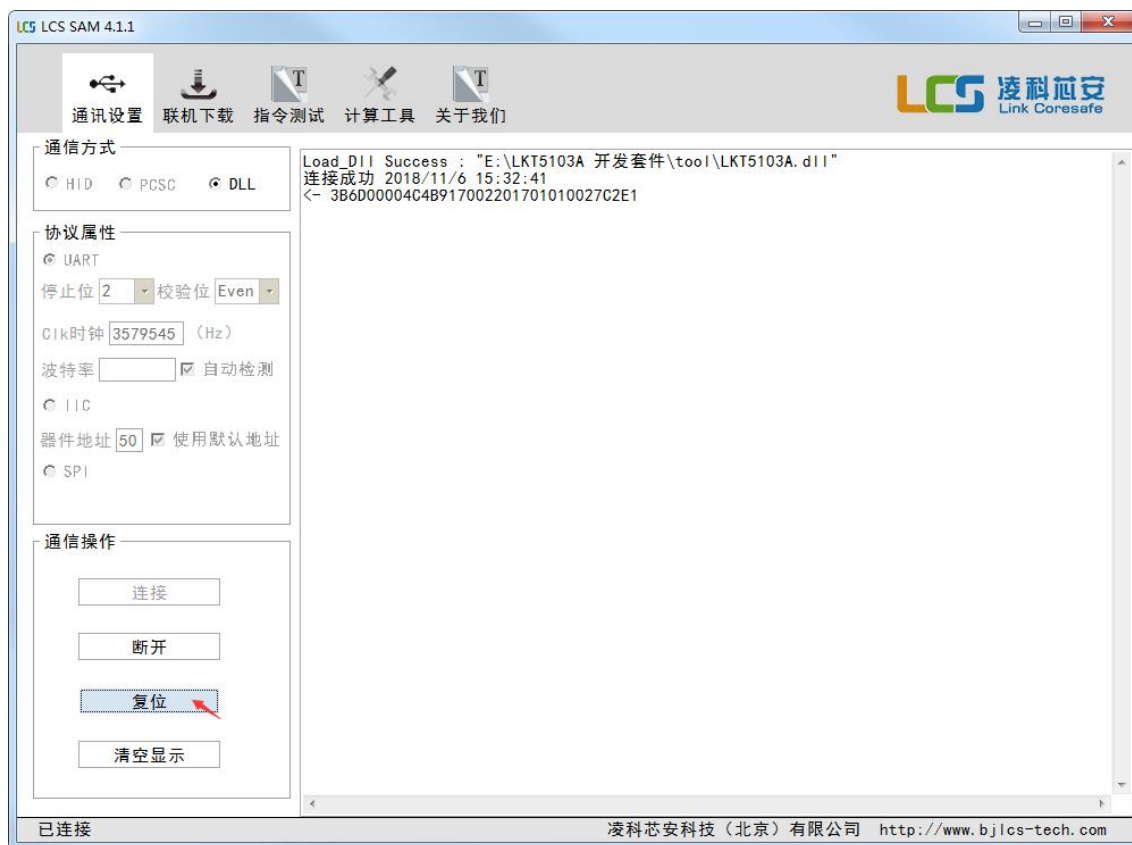


图 5-3

### 5.3 下载算法

- 点击“联机下载”选项页。
- 在“当前口令”中填写下载口令，默认下载口令为“0000000000000000”（口令长度必须为 8 字节）。
- 点击“选择算法文件”按钮，选择目标 hex 文件。
- 点击“下载算法”按钮下载算法，如图 5-4 所示。

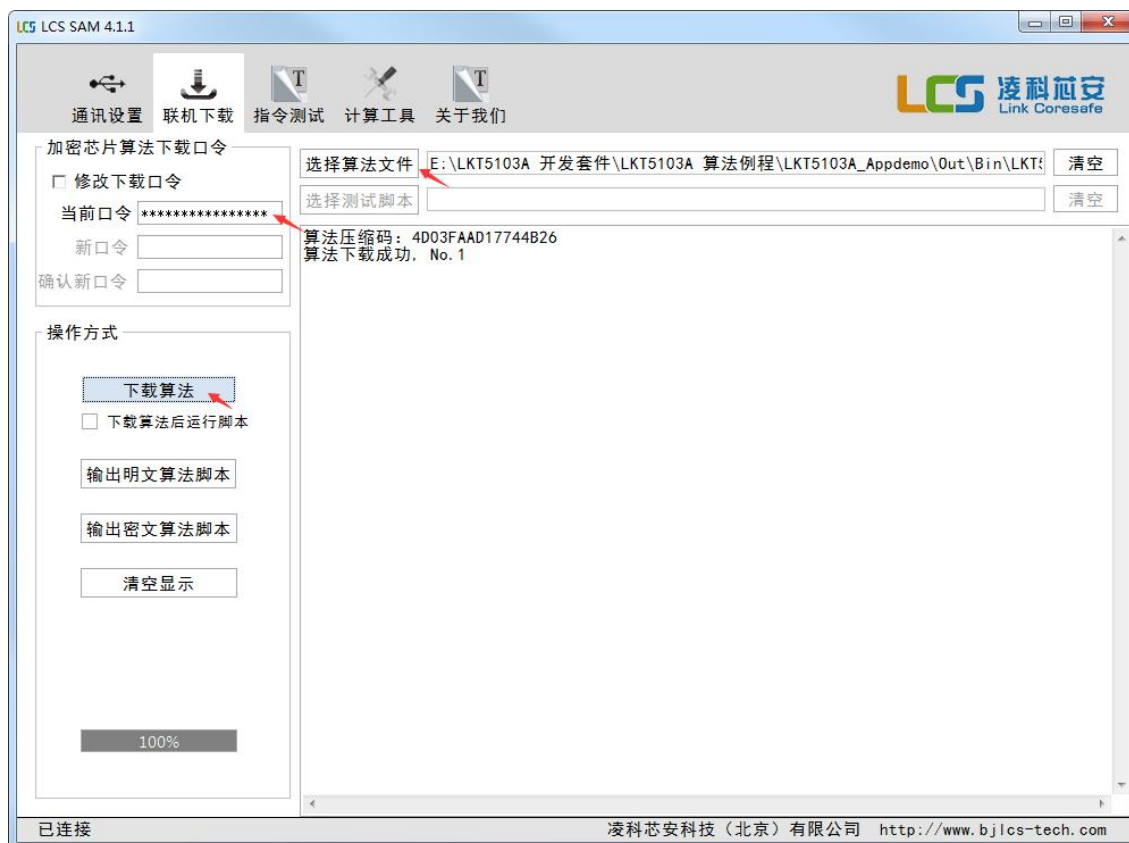


图 5-4：下载算法

## 5.4 修改下载保护口令

- 在“联机下载”选项页中的“当前口令”输入框内，填入当前使用的下载口令，勾选“修改下载口令”选项框，在“新口令”和“确认新口令”中，填入修改后的下载口令(口令长度必须为8字节)。
- 点击“下载算法”，算法下载成功后完成修改。
- 修改下载口令后，该芯片只能用新口令下载算法，新口令与其它芯片无关（其他芯片默认口令仍为00000000000000000000）。

## 5.5 发送算法指令

- 点击“指令测试”选项页。
- 在“APDU 指令”中输入算法指令。
- 点击“执行 APDU 指令”，如图 5-5 所示。

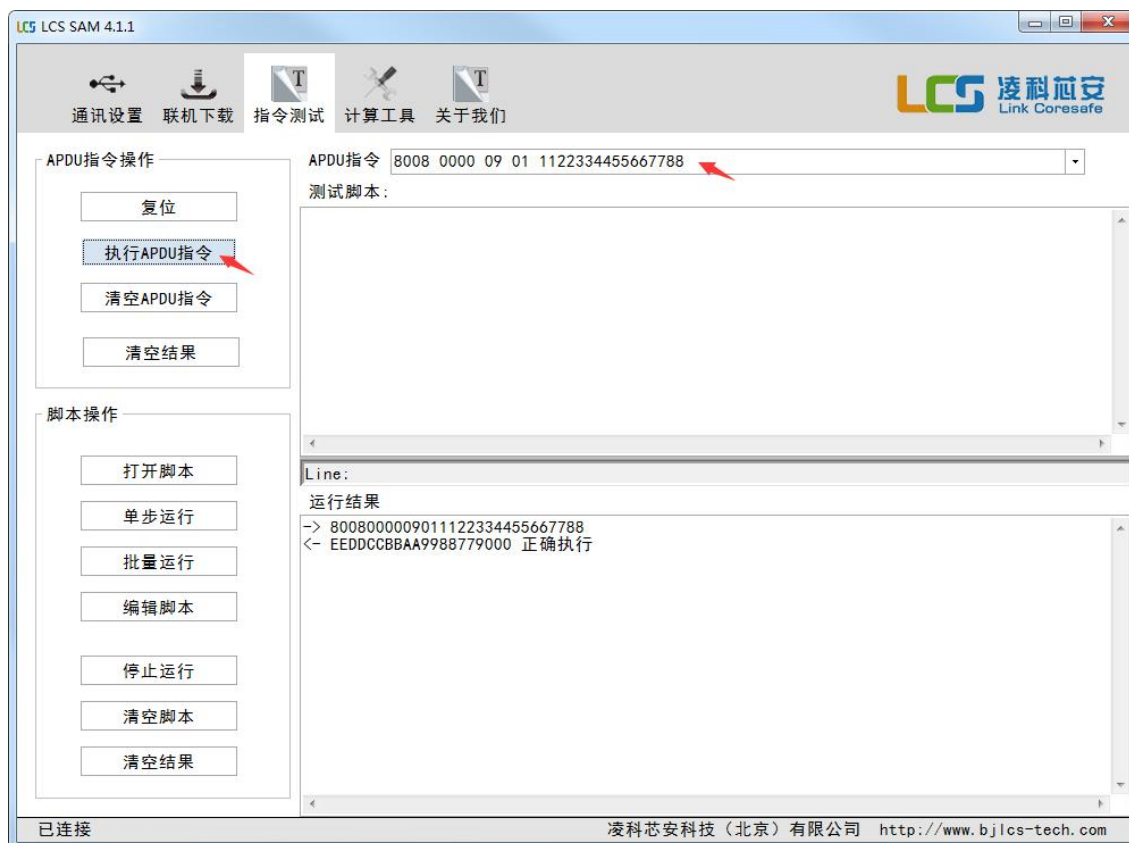


图 5-5：发送指令

## 5.6 批量测试算法指令

批量测试例程中的几个算法指令步骤如下：

- 在“指令测试”选项页中，点击“打开脚本”，选择脚本文件。
- 点击“批量运行”按钮，如图 5-6 所示。

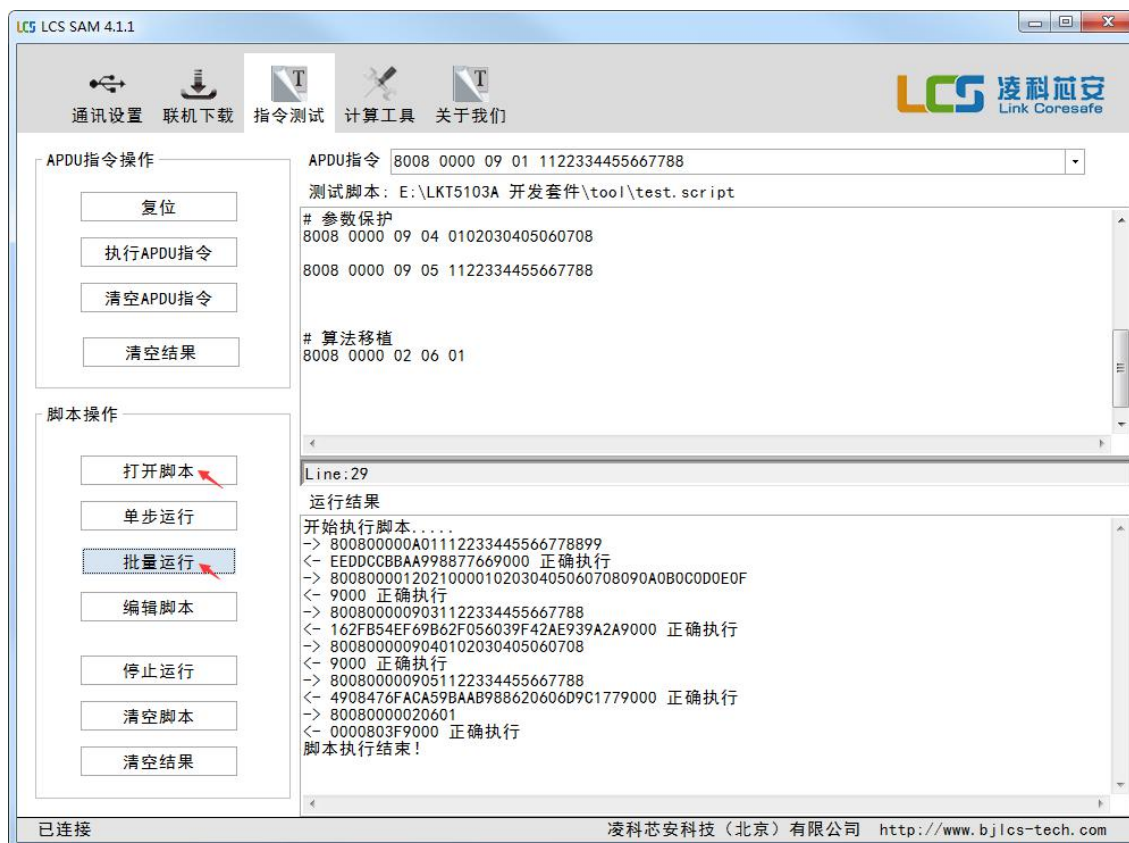


图 5-6：运行脚本



## 附录 A：驱动说明

LKT5103A 在 winxp、win7、win8 下免驱。驱动模式分为两种，U 盘驱动和光盘驱动。出厂默认为光盘模式。

注意：Windows 系统下两种驱动模式均可工作，但在 Linux 系统下只能使用 U 盘模式，下面详细讲解。

当 LKT5103A 接入到 PC 端 USB 接口之后，不会有提示信息。打开设备管理器之后，会在 DVD/CD-ROM 驱动器下看到新增了一个名为 Generic Usb Key USB Device 的设备。如图 A-1 中的红色框内所示。



图 A-1：设备名称

可以通过 APDU 指令来切换驱动模式。参看 5.1 节说明来连接 LKT5103A，参看 5.4 节说明发送指令“80CC 0000 04 DC00 23FF”，返回 9000 表示切换驱动成功。返回其他值为切换失败。切换成功后，重新拔插 LKT5103A 设备，打开设备管理器，会在通用串行总线控制器中看到新增了一个 USB 大容量存储设备，如图 A-2 中的红色框内所示，至此成功切换至 U 盘驱动模式。若要切回光盘驱动模式，发送指令“80CC 0000 04 DC02 23FD”，然后重新拔插 LKT5103A 设备即可。





图 A-2：设备名称