

# LKT4200HS 32 位加密芯片

## 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：北京市石景山区古城西街 255 号院 1 号楼中海大厦 B 座 1301

电话：010-68864300

传真：010-68864300-604

## 目 录

|                                      |        |
|--------------------------------------|--------|
| 第 1 章 LKT4200HS 芯片硬件特性 .....         | - 1 -  |
| 1.1 芯片参数 .....                       | - 1 -  |
| 1.2 引脚定义 .....                       | - 1 -  |
| 1.3 电气特性 .....                       | - 1 -  |
| 1.4 芯片安全特性 .....                     | - 2 -  |
| 1.4.1 内部调试资源 .....                   | - 2 -  |
| 1.4.2 硬件安全特性 .....                   | - 2 -  |
| 1.4.3 软件安全特性 .....                   | - 3 -  |
| 1.4.4 应用领域 .....                     | - 3 -  |
| 第 2 章 加密方案介绍 .....                   | - 4 -  |
| 2.1 算法移植方案介绍 .....                   | - 4 -  |
| 2.1.1 算法移植方案详解 .....                 | - 4 -  |
| 2.1.2 算法移植方案特点 .....                 | - 5 -  |
| 2.2 对比认证方案介绍 .....                   | - 6 -  |
| 2.2.1 对比认证方案详解 .....                 | - 6 -  |
| 2.2.2 对比认证方案特点 .....                 | - 7 -  |
| 2.3 参数保护方案介绍 .....                   | - 8 -  |
| 2.3.1 参数保护方案详解 .....                 | - 8 -  |
| 2.3.2 参数保护方案特点 .....                 | - 9 -  |
| 第 3 章 通讯调试说明 .....                   | - 10 - |
| 3.1 通讯电路 .....                       | - 10 - |
| 3.1.1 标准 UART 电路 .....               | - 10 - |
| 3.1.2 简化 UART 电路 .....               | - 10 - |
| 3.1.3 IO 模拟 UART 电路、单线 UART 电路 ..... | - 11 - |
| 3.1.4 加密芯片接口防护 .....                 | - 11 - |
| 3.2 时钟供给电路 .....                     | - 11 - |
| 3.2.1 PWM 波 .....                    | - 11 - |
| 3.2.2 无源晶振加起振电路 .....                | - 11 - |
| 3.2.3 有源晶振 .....                     | - 12 - |

|                        |        |
|------------------------|--------|
| 3.3 复位电路 .....         | - 12 - |
| 3.4 通讯时序 .....         | - 13 - |
| 3.4.1 复位时序 .....       | - 13 - |
| 3.4.2 时钟信号的要求 .....    | - 14 - |
| 3.4.3 串行通讯时序 .....     | - 14 - |
| 3.5 指令协议 .....         | - 15 - |
| 3.5.1 A3 指令解析 .....    | - 15 - |
| 3.5.2 T=0 指令解析 .....   | - 16 - |
| 3.5.3 提速指令说明 .....     | - 20 - |
| 3.5.4 MCU 通讯程序示例 ..... | - 22 - |
| 第 4 章 加密方案开发说明 .....   | - 24 - |
| 4.1 编译环境 .....         | - 24 - |
| 4.2 算法例程中的函数接口说明 ..... | - 25 - |
| 4.3 算法移植方案的设计开发 .....  | - 27 - |
| 4.3.1 开发阶段准备工作 .....   | - 27 - |
| 4.3.2 应用阶段实现流程 .....   | - 27 - |
| 4.4 对比认证方案的设计开发 .....  | - 29 - |
| 4.4.1 开发阶段准备工作 .....   | - 29 - |
| 4.4.2 应用阶段实现流程 .....   | - 30 - |
| 4.5 参数保护方案的设计开发 .....  | - 32 - |
| 4.5.1 开发阶段准备工作 .....   | - 32 - |
| 4.5.2 应用阶段实现流程 .....   | - 33 - |
| 4.6 编程指南 .....         | - 35 - |
| 4.6.1 头文件 .....        | - 35 - |
| 4.6.2 数据类型 .....       | - 35 - |
| 4.6.3 编程资源 .....       | - 35 - |
| 4.6.4 常量使用 .....       | - 35 - |
| 4.6.5 数据大小端问题 .....    | - 36 - |
| 4.6.6 算法工程结构解析 .....   | - 36 - |
| 4.6.7 算法工程代码功能解析 ..... | - 36 - |

|                                |        |
|--------------------------------|--------|
| 4.6.8 编程建议 .....               | - 38 - |
| 4.6.9 调试技巧 .....               | - 39 - |
| 4.6.10 注意事项 .....              | - 40 - |
| 4.7 安全提示 .....                 | - 40 - |
| 4.7.1 不要预留读 NVM 区接口的通道 .....   | - 40 - |
| 4.7.2 输入输出数据采用变化密文方式 .....     | - 41 - |
| 4.7.3 尽量避免只在开机阶段进行一次对比认证 ..... | - 41 - |
| 4.7.4 灵活设置代码陷阱 .....           | - 41 - |
| 第 5 章 算法下载 .....               | - 42 - |
| 5.1 在线下载 .....                 | - 42 - |
| 5.2 脱机下载 .....                 | - 42 - |
| 5.2.1 用 K100 开发板进行脱机下载 .....   | - 42 - |
| 5.2.2 用 P2000 烧录板进行脱机下载 .....  | - 47 - |
| 5.3 批量生产 .....                 | - 48 - |
| 第 6 章 芯片封装 .....               | - 49 - |

## 第 1 章 LKT4200HS 芯片硬件特性

### 1.1 芯片参数

#### CPU

- 高性能ARM SC100 32位CPU内核
- 16 /32位全RISC架构
- CPU内频最高84MHz

#### 片上存储

- 64K-Bytes 程序存储区
- 16K-Bytes NVM数据存储区
- 4K-Bytes RAM

#### 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
- 硬件DES/TDES协处理器
- 异常情况探测
- 256字节安全（只读）和256字节的不可擦Flash区
- 内存数据动态加密
- FLASH存储区加密
- 优化安全布局

#### 串行通讯接口

- 异步半双工，符合ISO7816-3标准
- 单IO T = 0协议

#### 奇偶校验/ CRC计算器

- 8 / 16 / 32位奇偶校验
- CRC-16/32 计算器

#### 时钟控制器

- 外部时钟
- 内部RC时钟

#### 复位

- 上电冷复位
- 热复位

#### 操控特性

- 单电源: 1.62V ~ 5.5V
- 外部时钟输入: 1 to 10MHz
- 工作温度: - 25 °C ~+ 85°C
- 最大电流10mA(Vdd=5.5V, Fclk=5MHz)
- 最大电流6mA(Vdd=3.3V, Fclk=4MHz)
- 最大电流4mA(Vdd=1.98V, Fclk=4MHz)

## 1.2 引脚定义

表1-1: LKT4200HS引脚说明

| 引脚序号 | 引脚名称 | 功能描述 | 引脚类型  |
|------|------|------|-------|
| 1    | GND  | 地    | ---   |
| 2    | NC   | 空置   | ---   |
| 3    | I/O  | 串行通讯 | 输入/输出 |
| 4    | NC   | 空置   | ---   |
| 5    | NC   | 空置   | ---   |
| 6    | CLK  | 时钟   | 输入    |
| 7    | RST  | 复位   | 输入    |
| 8    | VCC  | 电源   | ---   |

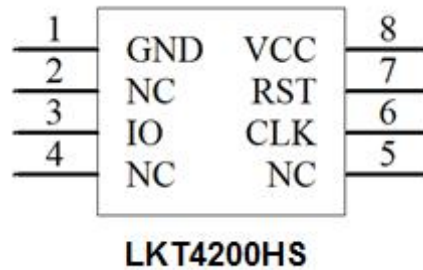


图1-1: LKT4200HS引脚示意图

## 1.3 电气特性

| Parameter                                 | Symbol    | Conditions                                | Min | Typ | Max | Unit |
|---|-----------|---|-----|-----|-----|------|
| Supply current                            | $I_{DD1}$ | $f_{CLK} = 5 \text{ MHz}, 5.5 \text{ V}$  | —   | —   | 10  | mA   |
|   |           | $f_{CLK} = 4 \text{ MHz}, 3.3 \text{ V}$  | —   | —   | 6   |      |
|   |           | $f_{CLK} = 4 \text{ MHz}, 1.98 \text{ V}$ | —   | —   | 4   |      |
| Stop Current                              | $I_{DD2}$ | $f_{CLK} = 1 \text{ MHz}, 5.5 \text{ V}$  | —   | —   | 200 | uA   |
|   | $I_{DD3}$ | $f_{CLK} = \text{GND}, 5.5 \text{ V}$     | —   | —   | 100 |      |
| Cell erase cycle time                     | $T_{EC1}$ | Page                                      | —   | 0.3 | —   | ms   |
|   | $T_{EC2}$ | Sector                                    | —   | 1.5 | —   |      |
|   | $T_{EC3}$ | Bank (Chip)                               | —   | 3.0 | —   |      |
| Cell write cycle time per 32 WU (32 word) | $T_{WC}$  | —1  | —   | 0.3 | —   | ms   |



## 1.4 芯片安全特性

### 1.4.1 内部调试资源

- 采用 32 位智能卡芯片内核，内置 32 位保密操作系统
- 全球唯一硬件 ID 与管理编码
- 具有兼容 uart 的串口
- 支持 ISO7816 T=0 和自定义 A3 通讯协议
- 具有 64K 字节超大用户程序下载空间（可以为客户定制容量）
- 16K 字节可定义安全性 NVM 数据存储区
- 4K 字节用户程序运行 RAM
- 编译环境具有丰富的系统调用和开发接口
- 硬件 3DES 协处理器

### 1.4.2 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（独立CLK）
- （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
- 随机数发生器
- 噪音的产生（对边信道攻击）
- 预硅功率分析

### 1.4.3 软件安全特性

- 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
- 不能直接访问硬件平台
- 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

### 1.4.4 应用领域

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。

## 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

LKT4200HS 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（以下简称加密芯片），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 MCU 程序中一部分关键算法移植到加密芯片中运行，如图 2-1 中所示，第一步先将代码 2 移植到加密芯片中。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芯片中。在实际运行中，通过调用函数方式运行加密芯片内的程序段，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。因此加密芯片成为了产品的一部分，而算法在加密芯片内部运算，盗版商无法破解，从根本上杜绝了程序被破解的可能。

MCU 程序分为两个部分：一部分是在 MCU 中，另一部分在加密芯片中。当需要用到加密芯片中的算法时 MCU 向其发送指令，加密芯片根据指令在内部运行程序并返回结果给 MCU。

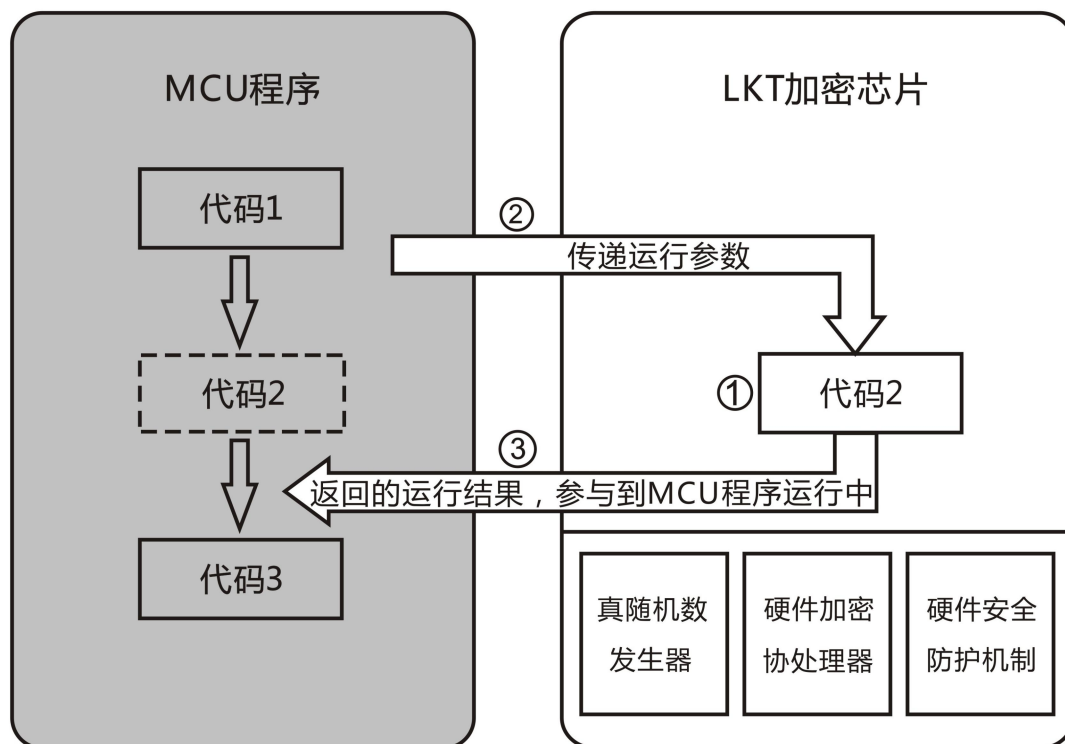


图 2-1：算法移植概念

## 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芯片的安全特性，让加密芯片与主控 MCU 的功能结合使用，MCU 核心代码分别在 MCU 和加密芯片中存储运行，单独破解 MCU 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

因为需要对加密芯片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；加密芯片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 MCU 程序都移植到加密芯片中，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 MCU 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。主控 MCU 移植 3DES 或 AES 等对称算法，主控 MCU 与加密芯片端在出厂发行阶段就预置相同的一组密钥。在运行阶段，MCU 产生随机数 RND 并将其发送给加密芯片，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 MCU 端比较 C1 与 C2，相同则证明加密芯片身份合法，MCU 程序继续运行；不同则证明加密芯片身份非法，MCU 程序退出运行。

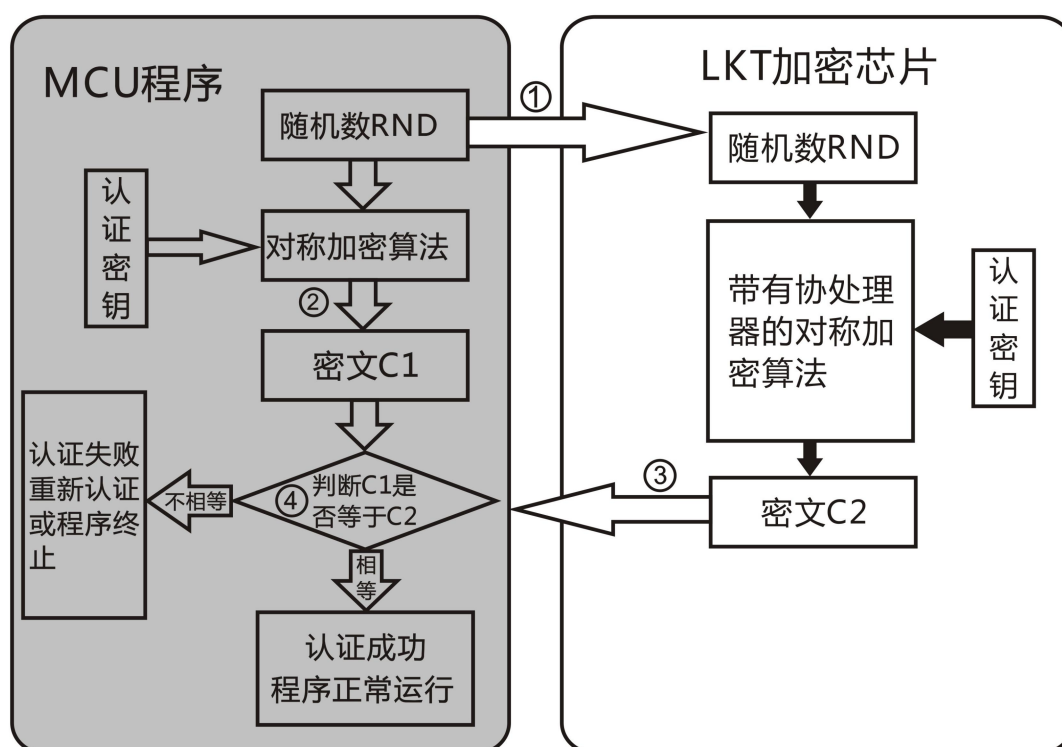


图 2-2: 对比认证方案概念

## 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，不需要对主控 MCU 端原有程序做大的改动，也不需要了解加密芯片内部运行流程。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芯片的主流加密方案，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。但是通过对主控 MCU 进行侵入式剖片攻击，可以读出 MCU 端程序进行改写，有效绕过认证对比点。因为 MCU 中的对比认证功能失效，所以在加密芯片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 MCU 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 MCU 中的一部分关键参数移植到加密芯片中存储。在实际运行中，MCU 发送读回参数指令和随机数到加密芯片端，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芯片产生的随机数回传到 MCU 端。此时，MCU 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。因为加密芯片存储了 MCU 中的关键数据，因此也成了产品的一部分，只对 MCU 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芯片中，可有效防止破解，从而起到了有效防护。加密芯片的引入，虽然无法阻止盗版商对 MCU 程序进行破解攻击，但通过对 MCU 原有关键参数的有效保护，杜绝了盗版商的抄板行为。

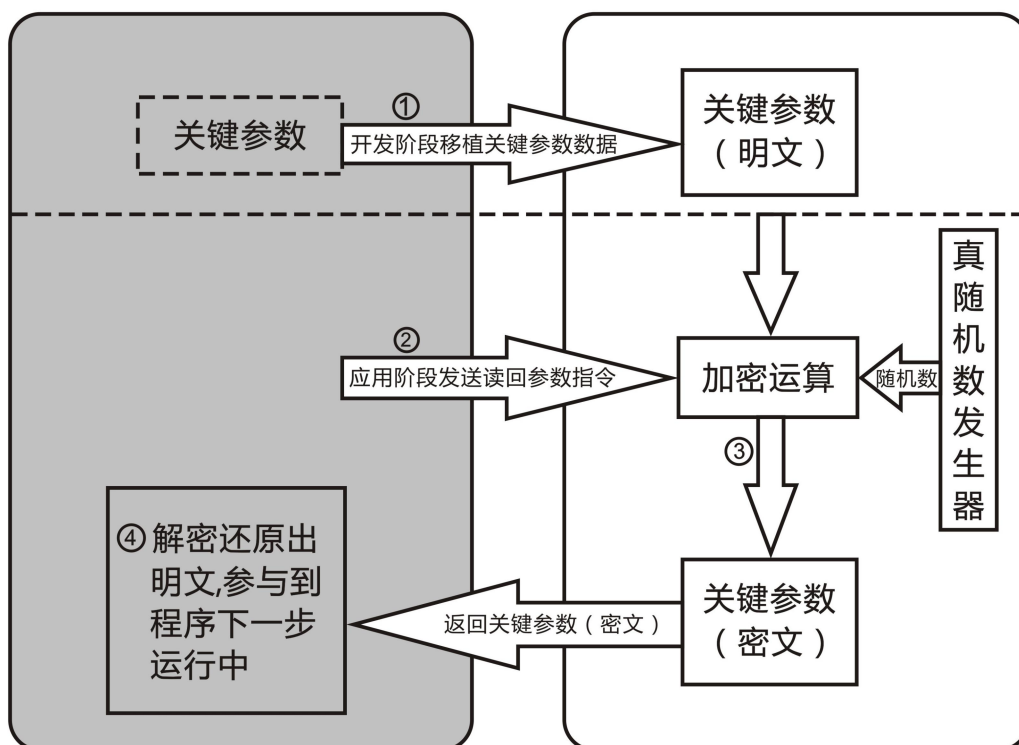


图 2-3：参数保护方案概念

### 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，不需要对主控 MCU 端原有程序做太大的改动，虽然需要了解加密芯片内部运行流程，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，MCU 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 MCU 端出现比较明显的比对点，所以该方案安全性相较对比认证有所提升，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

存储于加密芯片中的关键参数可以保证存储安全，当经线路传回到 MCU 端时，要进行线路加扰处理，MCU 和加密芯片端都产生随机数并参与到加密芯片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 MCU 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 通讯调试说明

### 3.1 通讯电路

### 3.1.1 标准 UART 电路

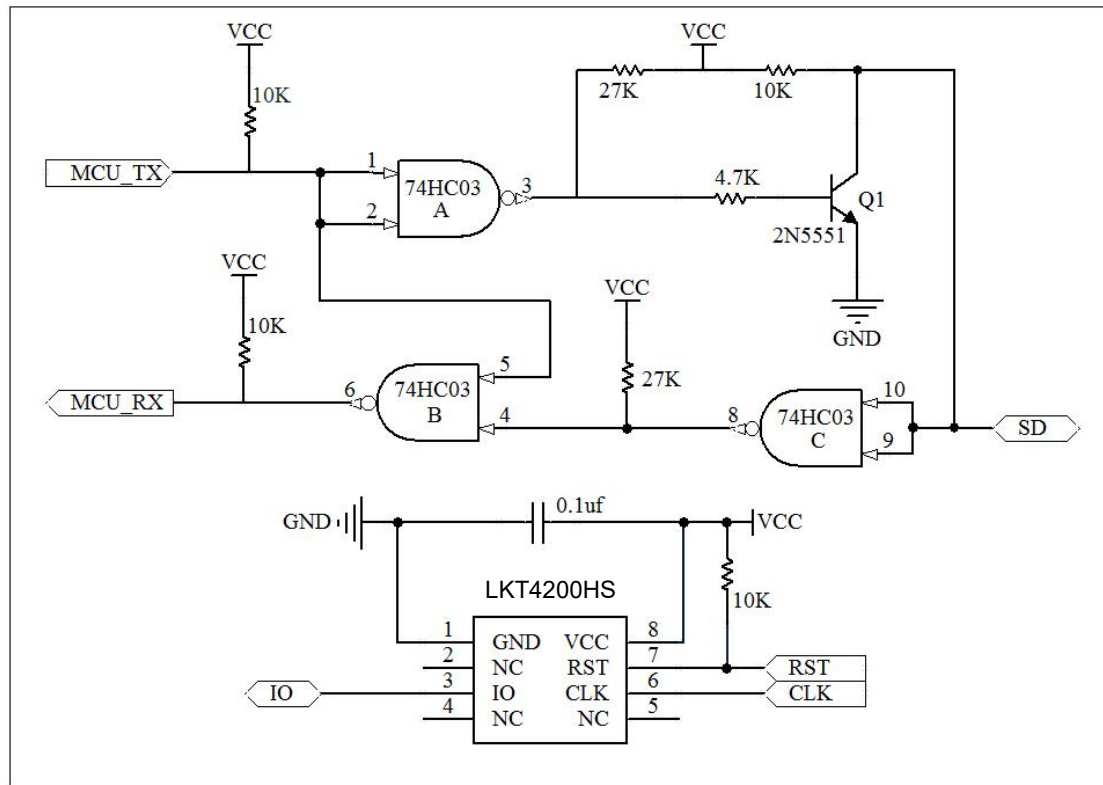


图3-1：标准uart模式

### 3.1.2 简化 UART 电路

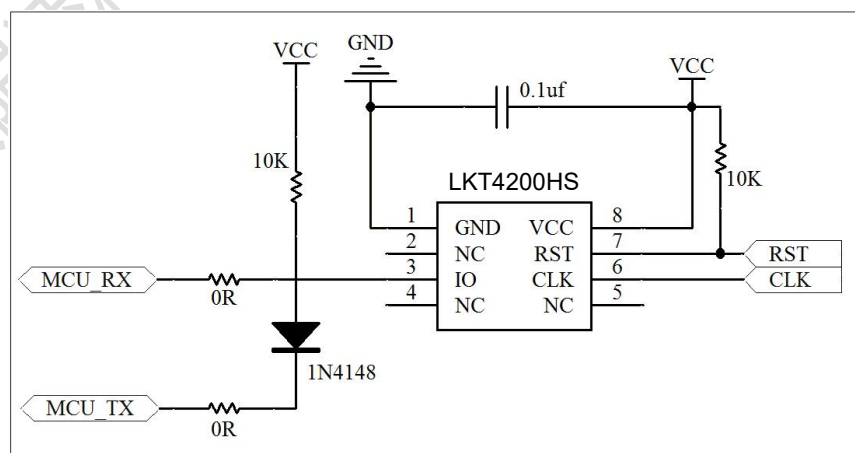


图3-2: 简化 uart 电路

注意：使用 UART 简化电路，单片机 TX 发送数据，同时单片机的 RX 禁止接收数据，目的是为了防止单片机接收到一个无用的冗余数据。也可一直开启，判断为无效数据，不做处理即可。

### 3.1.3 IO 模拟 UART 电路、单线 UART 电路

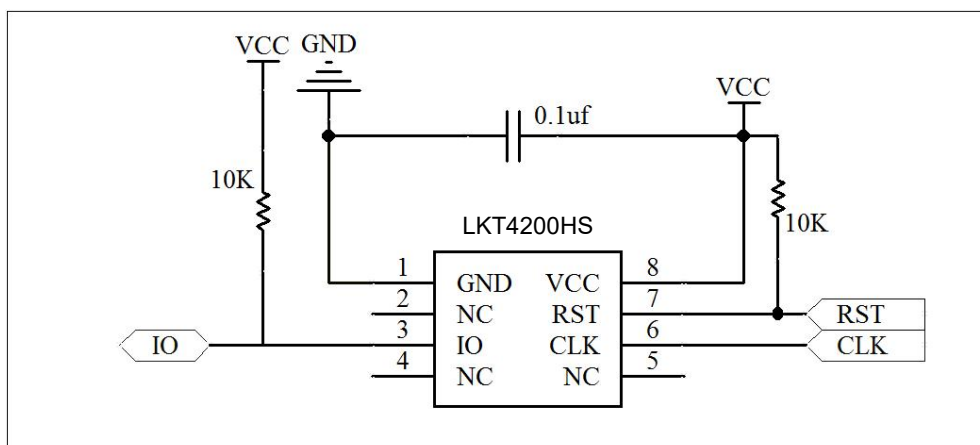


图3-3: IO模拟uart通讯接口

### 3.1.4 加密芯片接口防护

如果应用环境比较复杂, 建议加入TVS二极管对加密芯片的各个GPIO进行抗冲击防护。

## 3.2 时钟供给电路

### 3.2.1 PWM 波

客户MCU产生PWM波给加密芯片提供时钟信号，该方案可降低硬件成本。

### 3.2.2 无源晶振加起振电路

无源晶振及外围电路提供时钟 CLK。Y1 等于 3.579545MHz 为示例，用户可根据实际情况选择晶振频率。如图 3-4 所示，起震电路使用的芯片为 74HC03。

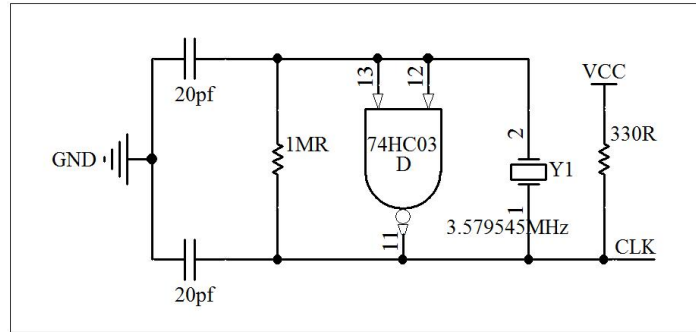


图3-4：无源晶振加起震电路

### 3.2.3 有源晶振

通过有源晶振直接提供时钟，如图 3-5 所示。

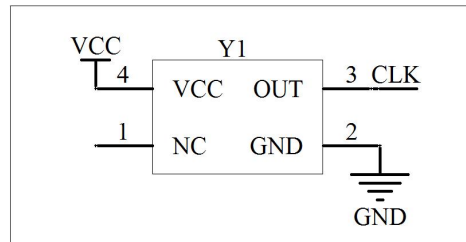


图3-5：有源晶振

### 3.3 复位电路

加密芯片有冷复位和热复位两种复位方式，可以将加密芯片的RST引脚与MCU的GPIO连接，通过MCU进行热复位操作。如果MCU的IO口资源不足，也可以使用冷复位。加密芯片的RST引脚通过外接阻容电路进行上电冷复位，电路如图3-6所示。

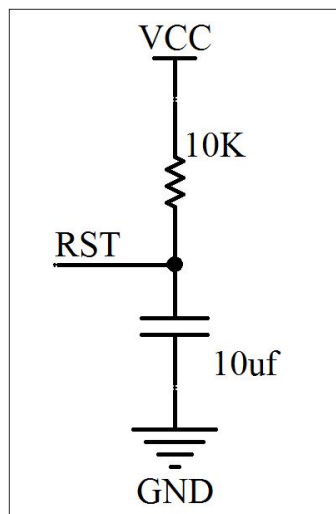


图 3-6：上电复位

## 3.4 通讯时序

芯片每次上电至少要进行一次复位操作，复位成功后才可以进行通讯指令交互。

### 3.4.1 复位时序

#### 3.4.1.1 冷复位（上电复位）

电源和时钟正常供给后，当加密芯片的 RST 引脚出现由低到高的时序后，加密芯片的通讯 IO 口在 400~40000 个时钟周期内返回一串复位信息，时序如图 3-7 所示。

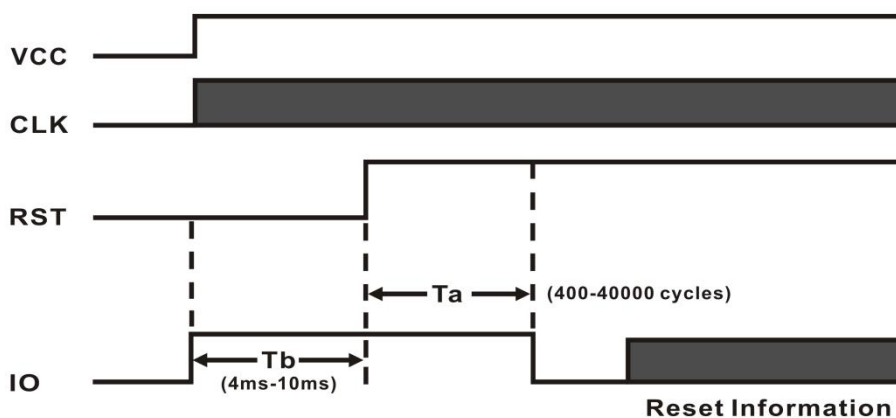


图 3-7：冷复位

#### 3.4.1.2 热复位

电源和时钟正常供给后，RST 引脚保持为高电平。将加密芯片的 RST 引脚由高电平拉低 4~10ms（建议拉低 5ms 即可）后释放为高电平，此后一直保持为高电平。加密芯片的通讯 IO 口在 400~40000 个时钟周期内返回一串复位信息，时序如图 3-8 所示。

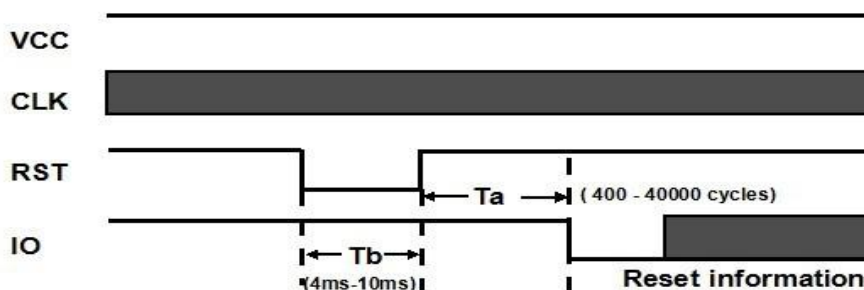


图 3-8：热复位

#### 3.4.1.3 复位信息

正确的复位信息有两种，以 3B 6D 开头的复位信息共 17 字节，以 3B 7D 开头的复位信息共 18 字节。用户可通过前两字节内容和复位信息长度来判断复位操作是否成功。

#### 3.4.2 时钟信号的要求

频率：提供 1M~10MHz 的外部时钟

占空比：40%~60%的方波

幅值： $\geq 0.7 * V_{CC}$

精度：20ppm 以内

#### 3.4.3 串行通讯时序

加密芯片采用 1 起始位、8 数据位（低位在前，高位在后）、1 偶校验位、2 停止位的帧格式。

##### 3.4.3.1 硬件 UART 接口通讯

正确配置 MCU 相关寄存器，确保帧格式和波特率无误，即可正常通讯。当外部时钟频率为 3.579545MHz 时，默认通讯速率约等于 9600bps。

##### 3.4.3.2 IO 模拟 UART 收发数据

当外部时钟频率为 3.579545MHz 时，ETU 为 104us（计算公式详见 3.5.3.1 节）。接收时先判断起始位，为了防止线路上出现干扰，MCU 在接收到第一个下降沿时，延时 30us 左右再读取一次，若仍为低电平，则可判断确为起始位。接下来，以 104us 为单位，做 8 次延时读取操作，将 8 位数据读回。然后判断一下偶校验位，此处也可用 104us 延时略过，不做判断。最后做 208us 延时，等待停止位传送完毕。至此，1 帧数据接收完毕。接收时序如图 3-9 所示。

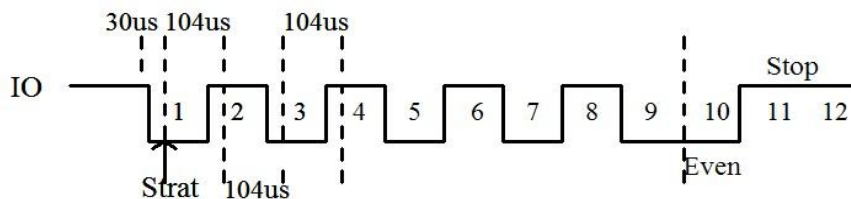


图 3-9：接收时序

MCU 发送指令时，按照 1 起始位、8 数据位（低位在前，高位在后）、1 偶校验位、2 停止位顺序发送即可，如图 3-10 所示。

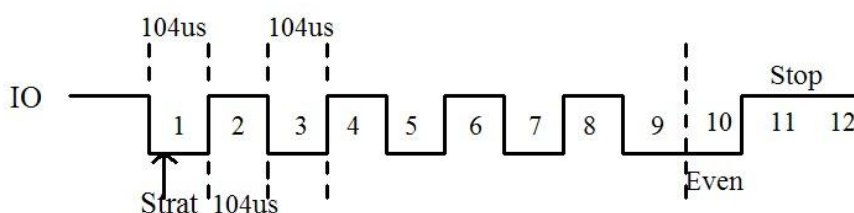


图 3-10：发送时序

### 3.5 指令协议

请注意：以下指令介绍中使用的数据均为 16 进制。11 表示 0x11，即十进制的 17。

加密芯片支持 A3 和 T=0 两种指令协议格式。A3 协议格式为我司自定义。我司推荐优先选用 A3 指令协议，因为交互流程简洁。若用户选用 T=0 协议格式，则支持与标准 PC/SC 接口类型的读卡器（如 LKT-L5 读写器）或者与 LKT-K100 开发板联调。若用户选用 A3 指令协议，只能选用 LKT-K100 开发板作为辅助调试工具。

#### 3.5.1 A3 指令解析

当加密芯片内部已经下载 hex 算法文件后，就可以使用 A3 指令进行功能测试了。A3 协议发送指令组成详见表 3-1 所示，命令头固定为 A3，发送数据长度 Lc 为命令体内容 CMD 的长度。

表 3-1：A3 协议发送命令结构

| 命令头 | 命令体长度 | 命令体内容 |
|-----|-------|-------|
| A3  | Lc    | CMD   |

A3 协议返回数据结构组成详见表 3-2 所示。命令头固定为 A3，返回数据长度 Le 为数据内容 DATA 加上状态码 SW 的总长度。其中 SW 固定为 2 字节长度，故  $Le = \text{DATA 长度} + 2$ 。

表 3-2: A3 协议返回数据结构

| 命令头 | 数据长度 | 数据内容 | 状态码 |
|-----|------|------|-----|
| A3  | Le   | DATA | SW  |

下面以算法例程中取反操作功能为例进行说明，指令详见表 3-3。命令体长度 Lc 为 09，CMD 为 011122334455667788。

表 3-3: A3 取反功能发送指令

| 命令头 | 命令体长度 | 数据内容                       |
|-----|-------|----------------------------|
| A3  | 09    | 01 11 22 33 44 55 66 77 88 |

加密芯片返回结果详见表 3-4 所示。数据长度 Le 为 0A（数据内容 8 字节 + 状态码 2 字节），数据内容为 EEDDCCBBAA998877。

表 3-4: A3 取反功能返回值

| 命令头 | 数据长度 | 数据内容                    | 状态码   |
|-----|------|-------------------------|-------|
| A3  | 0A   | EE DD CC BB AA 99 88 77 | 90 00 |

完整交互流程如图 3-11 所示，“->”表示 MCU 发送指令，“<-”表示 MCU 接收数据。

```

-> A3 09 011122334455667788          // A3 + Lc + DATA

<- A3 0A EEDDCCBBAA998877 9000      // A3 + Le + DATA
  
```

图 3-11: 加密芯片对输入数据取反

### 3.5.2 T=0 指令解析

#### 3.5.2.1 指令格式

APDU 命令由命令头和命令体两部分构成，如表 3-5 所示：

表 3-5: 指令结构

| 命令头 |     |    |    | 命令体 |      |
|-----|-----|----|----|-----|------|
| CLA | INS | P1 | P2 | Lc  | DATA |

SW 返回状态码的具体含义，如表 3-6 所示。

表 3-6: SW 状态码解析

| SW1 | SW2 | 意义                               |
|-----|-----|----------------------------------|
| 90  | 00  | 正确执行 (算法运行 return 1 )            |
| 63  | 00  | 密钥验证失败 (口令或密钥错误)                 |
| 61  | XX  | 有 XX 字节数据返回, 需要发送 00C0 0000XX 取出 |
| 67  | 00  | 长度错误                             |
| 69  | 85  | 没有下载算法                           |
| 6A  | 80  | 运行出错退出 (算法运行 return 0)           |
| 6A  | 86  | 指令中的第 3、4 字节错误                   |
| 6D  | 00  | 指令中的第 2 字节错误                     |
| 6E  | 00  | 指令中的第 1 字节错误                     |
| 6F  | 00  | 缓冲区内无数据, 但发送了 00C0 0000 XX 指令    |

### 3.5.2.2 指令处理流程说明

- 大于 5 字节的 APDU 指令, MCU 先发送前 5 字节, 如果收到 INS, 继续发送后续数据并接收 SW; 如果 MCU 收到 6X/9X, 只需再接收一字节数据 (X 表示 1~F 之间任意值); 如果 MCU 收到 60, 则继续接收直至收到 SW 为止。举例说明如图 3-12 所示。

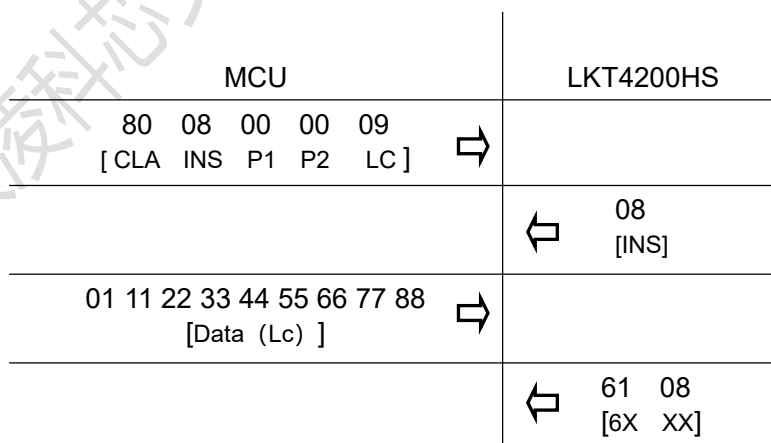


图 3-12: 大于 5 字节 APDU 指令交互流程

- 等于 5 字节的 APDU 指令, MCU 一次性全部发送后, 如果收到 INS, 则继续接收数



据和 SW；如果 MCU 收到 6X/9X，只需再接收一字节数据（X 表示 1~F 之间任意值）；

如果 MCU 收到 60，则继续接收直至收到 SW 为止。举例说明如图 3-13 所示。

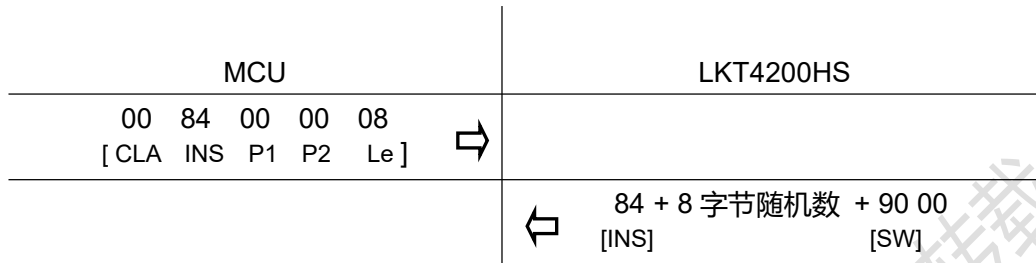


图 3-13：等于 5 字节 APDU 指令交互流程

### 3.5.2.3 获取随机数指令

LKT4200HS 在未下载算法前，只能进行获取随机数功能测试。MCU 发送指令格式如表 3-7 所示，加密芯片返回数据格式如表 3-8 所示。

表 3-7：获取随机数指令结构

| 命令头(4字节)    | 数据内容 (1字节) |
|-------------|------------|
| 00 84 00 00 | XX(01~10)  |

表 3-8：返回随机数格式说明

| INS 过程字节 | 数据内容     | SW   |
|----------|----------|------|
| 84       | XX 字节随机数 | 9000 |

指令 0084 0000 XX 中，前 4 字节固定不变，最后一字节 XX 取值范围是 01~10，执行成功后会从加密芯片中获 XX 字节随机数，交互流程如图 3-14 所示。

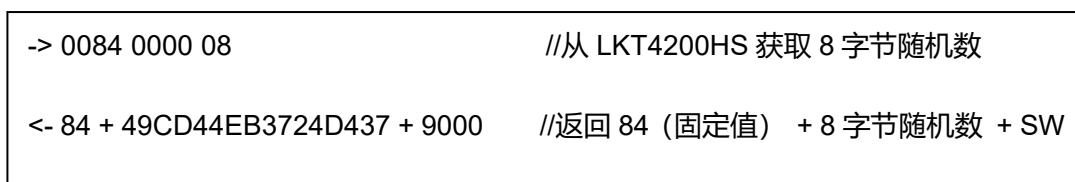


图 3-14：获取随机数完整交互流程

### 3.5.2.4 算法调用指令

在测试之前先保证加密芯片已经下载算法，下载方法详见第五章。现在以算法例程中取反操作功能为例进行说明。MCU 发送算法调用指令结构说明如表 3-9 所示，MCU 接收加密芯片返回 SW 数据格式如表 3-10 所示。MCU 获取加密芯片缓冲区内数据指令如表 3-11 所示。MCU 接收加密芯片返回缓冲区内内容格式如表 3-12 所示。

表 3-9：调用算法指令结构说明

| 命令头       | LC | 算法函数序号 | 传入算法函数中的参数              |
|-----------|----|--------|-------------------------|
| 8008 0000 | 09 | 01     | 01 02 03 04 05 06 07 08 |

表 3-10：加密芯片返回 SW 状态码格式说明

| SW1 | SW2 | SW 含义                      |
|-----|-----|----------------------------|
| 61  | 08  | 有 8 (sw2) 字节数据在加密芯片缓冲区等待取出 |

表 3-11：获取加密芯片缓冲区内数据指令说明

| 命令头      | 数据内容 |
|----------|------|
| 00C00000 | 08   |

表 3-12：加密芯片返回缓冲区内内容格式说明

| INS 过程字节 | 缓冲区数据内容          | SW   |
|----------|------------------|------|
| C0       | FEFDFCFBFAF9F8F7 | 9000 |

使用 MCU 向加密芯片发送调用算法命令的流程如图 3-15 所示。

|                            |                             |
|----------------------------|-----------------------------|
| -> 80080000 09             | /**发送命令头+LC**/              |
| <- 08                      | /**返回过程字节 INS**/            |
| -> 01 0102030405060708     | /**发送后续数据**/                |
| <- 6108                    | /**有 8 字节应答数据等待读出**/        |
| -> 00C0000008              | /**发送获取数据命令**/              |
| <- C0 FEFDFCFBFAF9F8F79000 | /**取出数据 (INS+算法返回数据+SW) **/ |

图 3-15：MCU 执行过程

### 3.5.3 提速指令说明

#### 3.5.3.1 默认通讯速率

MCU 对加密芯片进行复位操作并接收完整复位信息后,如果不向其发送任何提速指令,则加密芯片使用缺省速率通讯,计算公式为  $S = F/C$  (S: 通信速率、 F: 外部时钟频率、 C: 分频系数), 每位有效数据时间  $ETU = 1 / S$ 。外部时钟频率 S 由用户选择提供, 芯片的默认分频系数 C 为 372。

外部时钟频率若为 3.579545MHz, 则  $S = F / C = 3.579545M / 372 \approx 9600\text{bps}$ ,  $ETU = 372/3.579(M) \approx 104\mu\text{s}$ 。

#### 3.5.3.2 临时提速

加密芯片支持临时提速和永久提速两种提速方式。在外部时钟不变的前提下, 用户可以通过修改分频系数来提速。

临时提速指令见表 3-13。

表 3-13: PPS 值

| 提速指令     | PPS 值 (修改后的分频系数) |
|----------|------------------|
| FF10947B | 64               |
| FF10957A | 32               |
| FF109679 | 16               |

先对加密芯片进行复位操作,使用缺省通讯速率接收完整复位信息,然后发送提速指令。当加密芯片返回与提速指令一样的数据后, 提速操作完成。用户必须采用提速后波特率进行通讯。注意: 一旦进行复位操作或重新上电, 分频系数重新恢复为 372, 通讯速率又恢复为缺省速率, 需要重新进行提速。下面实际举例说明完整提速流程。

假设外部时钟频率为 3.579MHz, 需要将通讯速率提升到 115200。交互流程如图 3-16 所示。

|                                       |   |
|---------------------------------------|---|
| 复位操作                                  | /**复位操作**/                                    |
| 使用 9600bps 进行通讯                       |   |
| <- 3B6D00004C4B917002201212170008021D | /**返回 17 字节复位信息, 最后 8 字节为芯片唯一 ID 号, 不会出现重复**/ |
| -> FF10957A                           | /**将分频系数临时修改为 32**/                           |
| <- FF10957A                           | /**修改成功, 通讯速率提升为 115200bps**/                 |

图 3-16: 临时提速操作

### 3.5.3.3 永久提速

通过提速后通讯速率 S 和外部时钟频率 F 即可计算出期望的分频系数 C。通过 APDU 指令即可将分频系数永久设定为目标值。永久的含义是复位或者断电不会将分频系数修改为默认值, 永久提速支持重复修改分频系数 C。

假设期望的提速后通讯速率为  $S = 57600\text{bps}$ , 外部时钟频率  $F = 3.579545\text{Mhz}$ 。根据公式  $S = F/C$  (S: 通信速率、F: 外部时钟频率、C: 分频系数,  $16 \leq C \leq 372$ ), 可计算出期望的分频系数  $C = F / S = 3.579545\text{M} / 57600 \approx 62$ , 可通过发送 80 CC 00 00 04(固定格式) + 003E(需要修改成的默认分频系数, 16 进制) + FFC1 (前两字节取反值)。发送成功后芯片返回 9000, 至此提速工作完成, 只需重新上电或复位即可生效, 提速流程如图 3-17 所示。

|                                       |   |
|---------------------------------------|---|
| 复位操作                                  | /**复位操作**/                                    |
| 使用 9600bps 进行通讯                       |   |
| <- 3B6D00004C4B917002201212170008021D | /**返回 17 字节复位信息, 最后 8 字节为芯片唯一 ID 号, 不会出现重复**/ |
| -> 80CC 0000 04 003E FFC1             | /**将分频系数永久修改为 62**/                           |
| <- 9000                               | /**修改成功, 通讯速率提升为 57600bps**/                  |

图 3-17: 永久提速操作

### 3.5.4 MCU 通讯程序示例

用户可直接使用我司编写封装的 APDU 指令处理函数。兼容 A3、T=0 指令协议，用户无需自己编写指令处理函数。具体参考示例代码位于开发套件“串行通讯接口例程”文件夹中。APDU 指令交互函数说明详见表 3-14，APDU 发送函数说明详见表 3-15，APDU 接收函数说明详见表 3-16。用户可以直接移植这三个功能函数，即可完成 APDU 指令处理的封装工作。

```
typedef uint32_t u32;
typedef uint16_t u16;
typedef uint8_t u8;
```

表 3-14：指令交互函数

| 函数描述 | 说明  |        |
|------|---|--------|
| 函数形式 | u16 Exchange_APDU(u16 SendCmdLen,u8 *SendCmdData,u16 *ReceDataLen,u8 *ReceData) |        |
| 参数 1 | [IN]  | 发送指令长度 |
| 参数 2 | [IN]  | 发送指令内容 |
| 参数 3 | [OUT]   | 接收数据长度 |
| 参数 4 | [OUT]   | 接收数据内容 |
| 返回值  | SW 状态码  |        |

表 3-15：发送指令函数

| 函数描述 | 说明  |        |
|------|---|--------|
| 函数形式 | void Send_Cmd(u16 SendCmdNum,u8 *SendStr) |        |
| 参数 1 | [IN]                                      | 发送指令长度 |
| 参数 2 | [IN]                                      | 发送指令数据 |
| 返回值  | 无   |        |

表 3-16：接收数据函数

| 函数描述 | 说明  |        |
|------|---|--------|
| 函数形式 | u8 Rece_Data(u16 RecedataNum,u8 *ReceStr) |        |
| 参数 1 | [IN]                                      | 接收数据长度 |
| 参数 2 | [OUT]                                     | 接收数据内容 |
| 返回值  | 正确返回 0，错误返回 1                             |        |

## 第 4 章 加密方案开发说明

本章介绍基于加密芯片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 MCU 程序和加密芯片程序共同组成的。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芯片内部用 C 语言编程实现 MCU 中被移植代码的功能，编译成功后将生成的 HEX 文件下载到加密芯片中。

注意：HEX 文件就是被移植到加密芯片内部的算法，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                                      | 2018/9/28 15:33 | UVOPT 文件  | 8 KB  |
| <input checked="" type="checkbox"/> AppDemo.uvproj | 2018/3/27 16:32 | UVPROJ 文件 | 15 KB |
| AppDemo_LKT4200HS.dep                              | 2018/9/28 15:33 | DEP 文件    | 2 KB  |
| AppDemo_uvopt.bak                                  | 2018/9/28 15:32 | BAK 文件    | 8 KB  |

图 4-1：算法工程文件

## 4.2 算法例程中的函数接口说明

LKT4200HS 提供 16K 字节的 NVM 数据存储区，从地址“0x0000”到“0x3FFE”。

写 NVM 区函数如表 4-1 所示。

u16 等价于 unsigned short

u8 等价于 unsigned char

| 函数描述 | 说明  |
|------|---|
| 函数形式 | void LK_WriteNvm (u16 addr, u8 *buf, u8 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 (u16 addr,u8 *buf, u8 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 加密函数。注意这三个参数都是 LV 结构(数据长度+数据，如加密数据时 08 (长度) 1122334455667788(数据内容)) 如表 4-3 所示。

| 函数描述 | 说明  |
|------|---|
| 函数形式 | void LK_DESEncrypt(u8 *plain, u8 *k, u8 *cipher); |
| 参数 1 | 明文长度+明文值  |
| 参数 2 | 密钥长度+密钥值  |
| 参数 3 | 输出的密文长度+密文值                                       |

表 4-3：DES/3DES 加密



DES/3DES 解密函数。注意这三个参数都是 LV 结构(数据长度+数据 , 如解密数据时 08 (长度) 1122334455667788(数据内容)) 如表 4-4 所示。

| 函数描述 | 说明   |
|------|--|
| 函数形式 | void LK_DESDecrypt(u8 *plain, u8 *k , u8 *cipher); |
| 参数 1 | 需解密的密文长度+密文值                                       |
| 参数 2 | 密钥长度+密钥值   |
| 参数 3 | 解密后的明文长度+明文值                                       |

表 4-4: DES/3DES 解密

获取随机数函数见表 4-5。

| 函数描述 | 说明                                    |
|------|---------------------------------------|
| 函数形式 | void LK_GetRandom (u8 *buf, u8 len ); |
| 参数1  | 存放随机数据                                |
| 参数 2 | 获取随机数的位数                              |

表 4-5: 获取随机数

获取芯片 ID 号函数见表 4-6。

| 函数描述 | 说明                         |
|------|----------------------------|
| 函数形式 | void LK_GetChipID(u8 *sn); |
| 参数1  | 存放芯片 ID 号                  |

表 4-6: 获取芯片 ID 号

## 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：

- 向加密芯片中移植算法代码 A，请参考 “\LKT4200HS 算法例程 \LKT4200HS\_AppDemo\mdk” 路径下的 App\_fun.c 文件中的 Algorithm\_Transplantation 函数。
- 加密芯片烧录 hex 的方法详见第 5 章

### 4.3.2 应用阶段实现流程

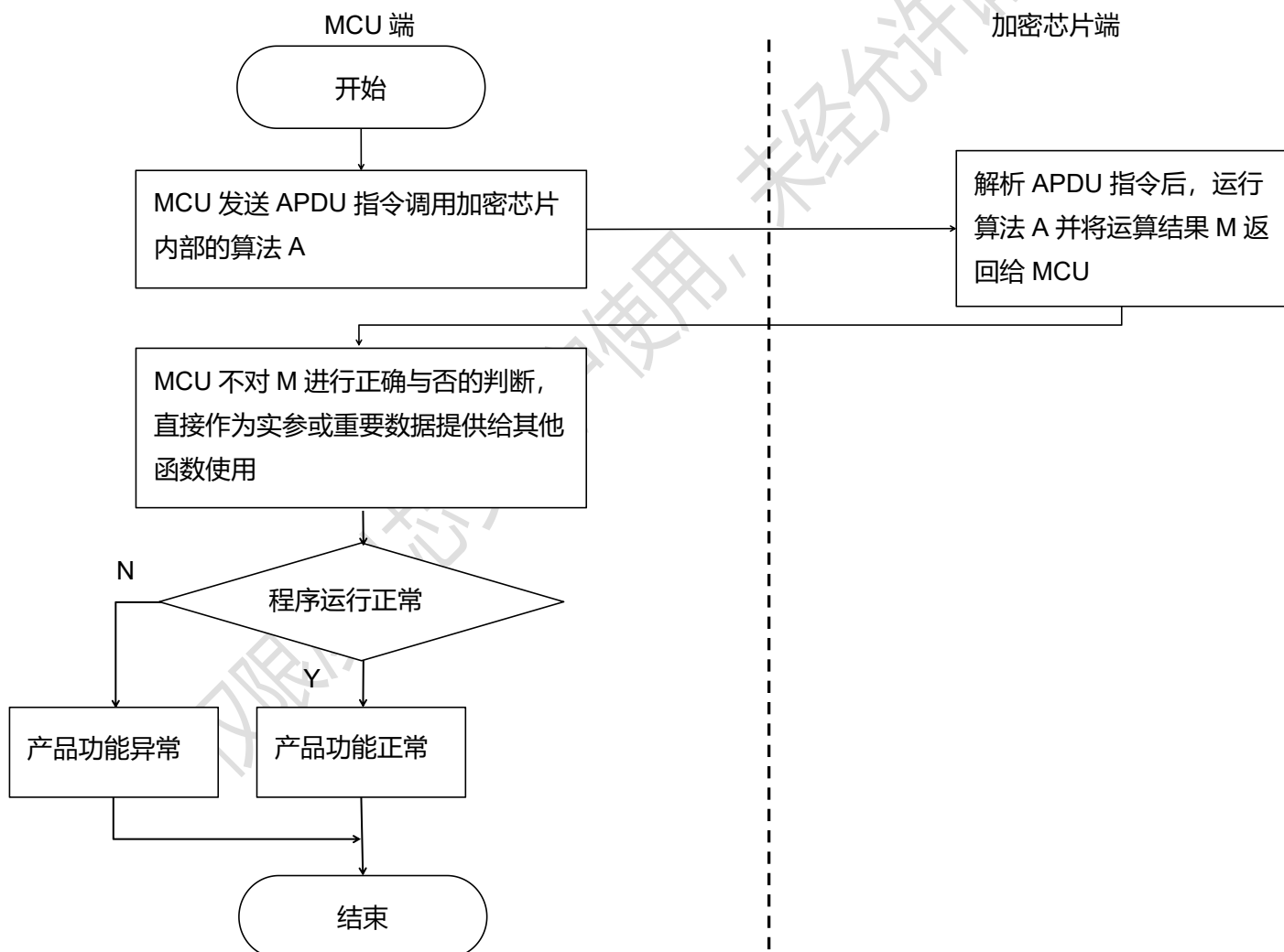


图 4-2：算法移植实现流程图

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 MCU 中移植到加密芯片中的程序功能如图 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：移植到加密芯片中的算法

- MCU 发送 A3 02 06 01 到加密芯片，x=01。
- 加密芯片返回 A3 06 3F80 0000 9000，y=01。指令解析详见 3.5.1 节。
- MCU 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，用户可调用加密芯片内部接口 LK\_GetRandom 产生，加密可采用加密芯片自带的硬件 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。其中加密芯片烧录 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: MCU 端准备工作

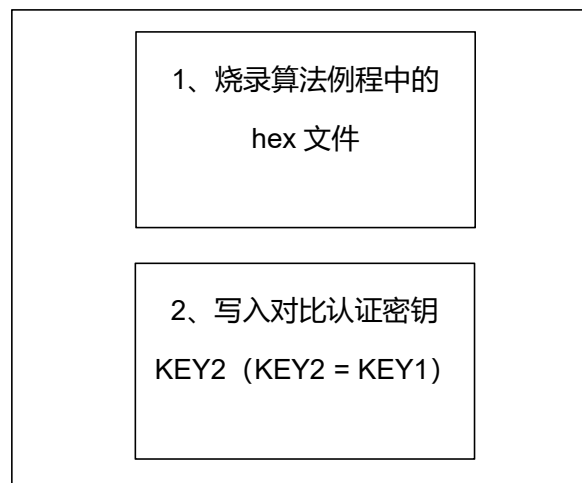


图 4-5: 加密芯片端准备工作

#### 4.4.2 应用阶段实现流程

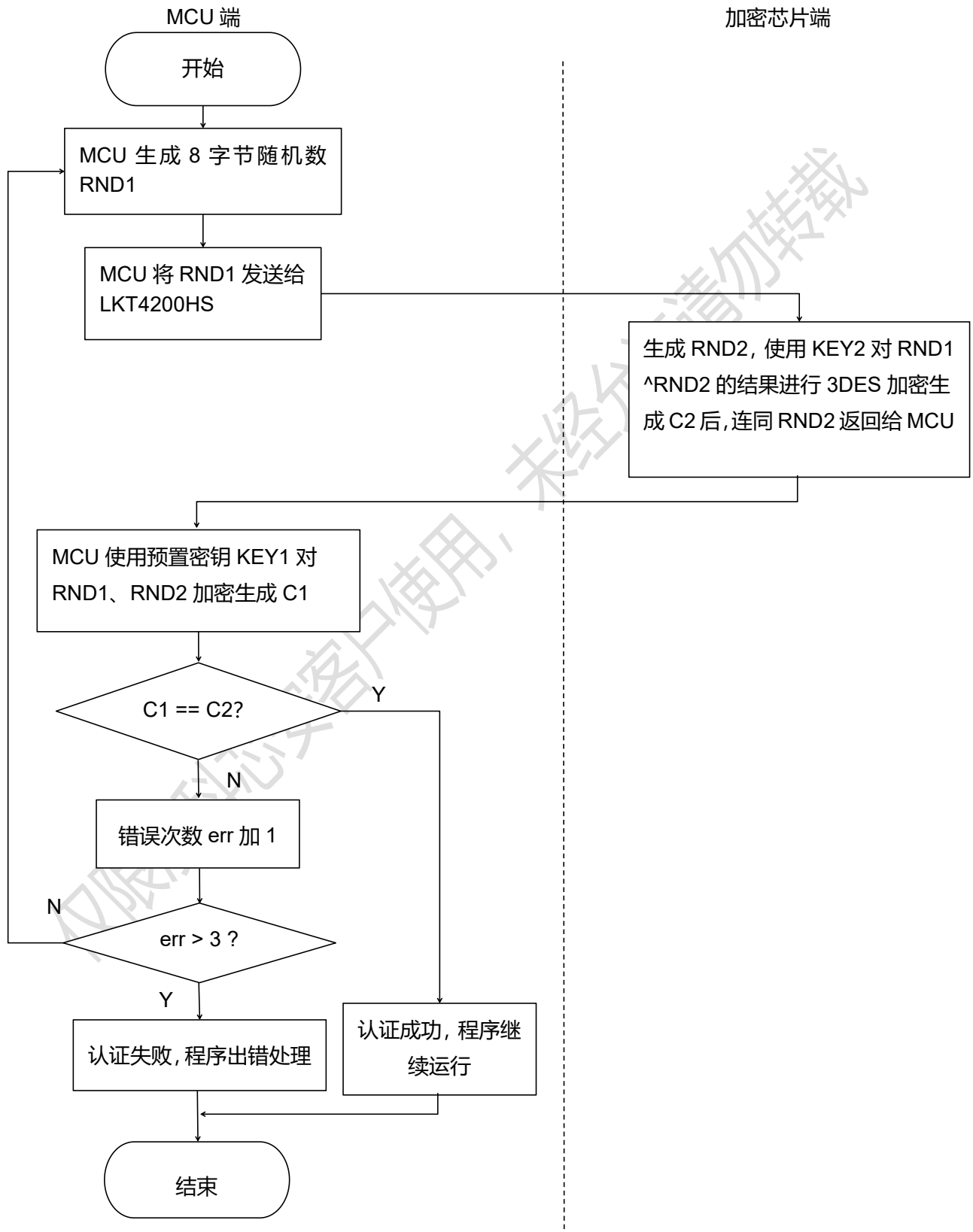


图 4-6： 对比认证流程图

- 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
- MCU 发送 A3 09 03 1122334455667788 到加密芯片。
- 加密芯片返回 A3 10 0102030405060708 1735509EA362EDDE，其中 C2 = 1735509EA362EDDE。
- MCU 使用预置的密钥 KEY1 对 RND1 异或 RND2 的值进行 3DES 加密生成 C1，  
 $C1 = 3DES\_EN(RND1 \oplus RND2, KEY1)$ 。
- MCU 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。

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

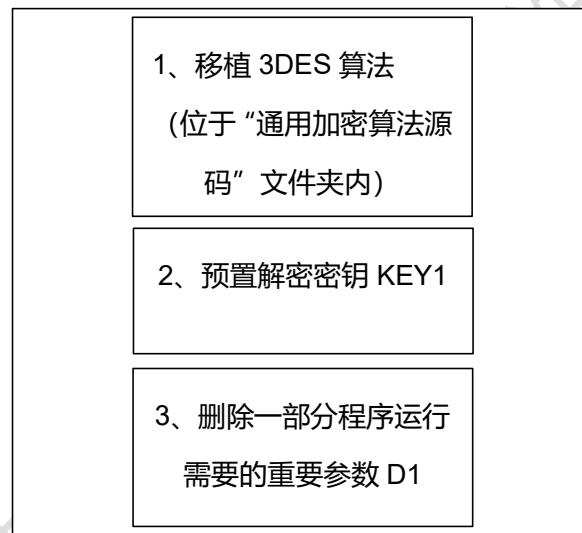


图 4-7: MCU 端准备工作



图 4-8: 加密芯片端准备工作

#### 4.5.2 应用阶段实现流程

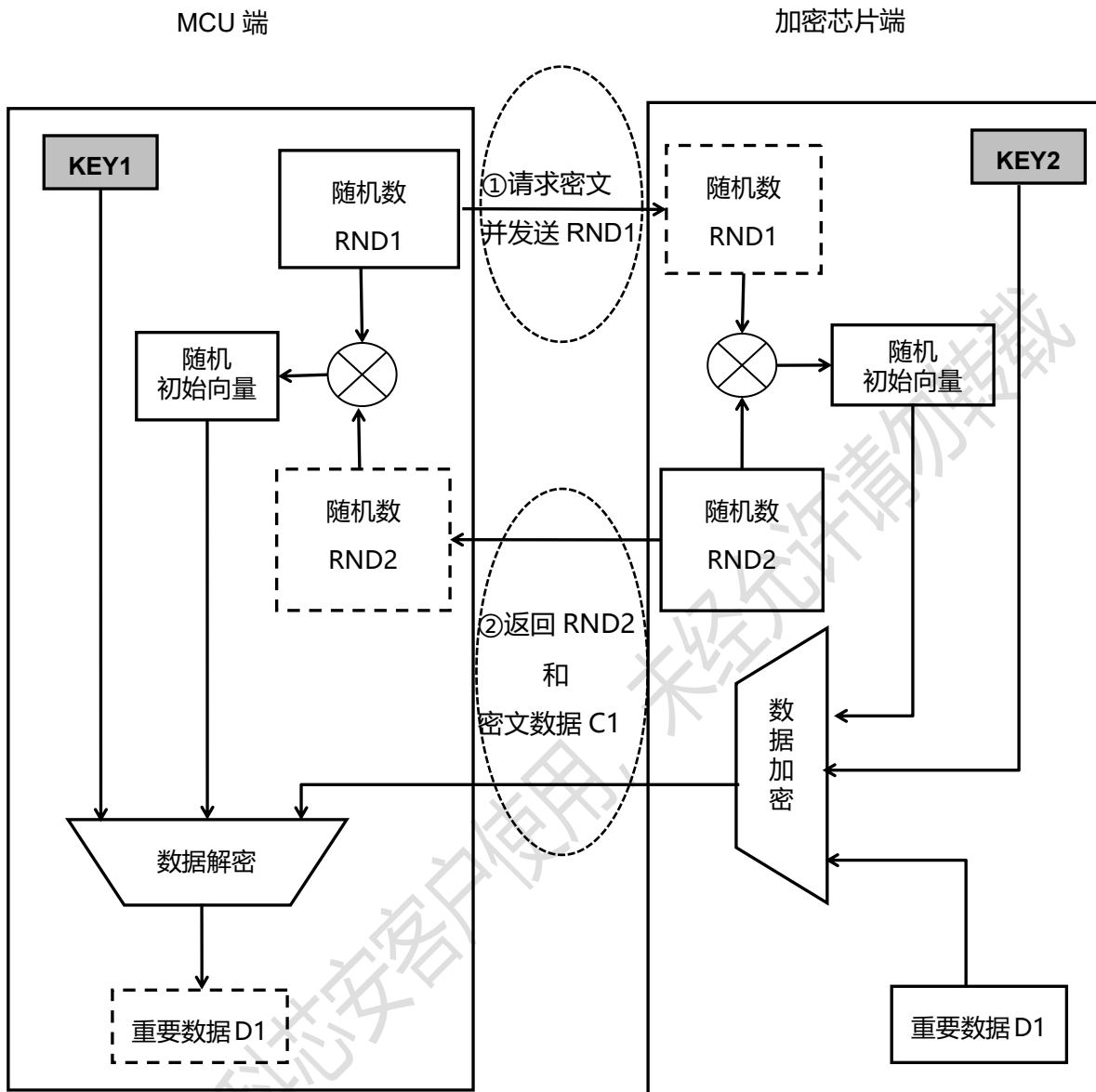


图 4-9：参数保护方案流程图

- 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：  
 $RND1 = 1122334455667788$   
 $RND2 = 0102030405060708$   
 $D1 = 1234567812345678$   
 $KEY1 = 11111111111111112222222222222222$   
 $KEY2 = KEY1$
- MCU 发送 A3 09 05 + 1122334455667788 (RND1) 到加密芯片。
- 加密芯片返回 A3 10 + 0102030405060708 (RND2) + 8FF76B64C412F3C9 (密文) ,



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)。

- MCU 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)。

- MCU 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 MCU 程序运行的各个阶段。

## 4.6 编程指南

### 4.6.1 头文件

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。

lkapi.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。

### 4.6.2 数据类型

主要数据类型占用空间详见表 4-7

表 4-7：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   | DWORD | 4      |

### 4.6.3 编程资源

- CODE 区空间共 64K 字节，支持用户自定义编程。
- NVM 区 16K 字节，可用于存储数据参数，掉电不丢失。
- RAM: 4K 字节。

### 4.6.4 常量使用

因为加密芯片内部 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 const testbuf [6] = {1,2,3,4,5,6};
```

#### 4.6.5 数据大小端问题

大小端与 CPU 或 MCU 的架构有关，在此不做赘述。LKT4200HS 的数据存储为大端模式。用户移植算法时要特别注意原有 CPU 存储模式与 LKT4200HS 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 4-8。

表 4-8：工程文件详解

| 文件夹名称 | 包含文件                                 | 功能                       |
|-------|--------------------------------------|--------------------------|
| head  | DEF_Type.h<br>DEF_Macro.h<br>lkapi.h | 系统函数声明和变量定义              |
| Out   | AppDemo.hex<br>.....                 | 编译后生成的算法目标文件，用于下载到加密芯片中。 |
| prj   | AppDemo.uvproj<br>.....              | 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

加密芯片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\_FUNCTION 为主函数。加密芯片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

加密芯片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 MCU 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。MCU 与 LKT4200HS 进行数据交互需要利用 APDU 指令作为数据通道。主函数 APP\_FUNCTION 声明如下：

```
u32 APP_FUNCTION(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 MCU 向加密芯片发送指令时，APDU 指令数据域中的内容会被传送到加密芯片的 pInBuf 中，A3 协议指令的第二字节开始为数据域内容，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 4-9 所示。

表 4-9： MCU 向 LKT4200HS 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 |
|--------|-------|-----------|------------------------|---------|---------|
| MCU 发送 | A3    | A3        | 09 01 1122334455667788 | pInBuf  | FF      |
|        | T = 0 | 8008 0000 | 09 01 1122334455667788 | pInBuf  | FF      |

加密芯片接收到指令后，会执行 APP\_FUNCTION 中的程序，当运行完毕后，将输出的数据放入 pOutBuf 中，加密芯片自动向 MCU 输出数据。APDU 接收数据解析如表 4-10 所示。

表 4-10： LKT4200HS 向 MCU 返回数据

| 通讯方向 | 协议类型  | LKT4200HS 输出缓冲区地址 | 输出缓冲区大小 | 输出给 MCU 的数据内容 | MCU 处理方式      |
|------|-------|-------------------|---------|---------------|---------------|
| MCU  | T = 0 | pOutBuf           | FF      | 6108          | 发送 00C0000008 |

|    |    |         |    |                  |      |
|----|----|---------|----|------------------|------|
| 接收 |    |         |    | EEDDCCBBAA998877 | 直接使用 |
|    | A3 | pOutBuf | FF | 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，加密芯片运行完毕后会先返回 61XX(XX 为待输出数据的长度)，此时 MCU 需要发送 00C0 0000 XX，将加密芯片缓冲区内数据取出。使用 A3 协议不存在该情况，加密芯片运行完毕后会直接输出数据。

#### 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 MCU 发送 APDU 指令后，都将触发主函数 APP\_FUNCTION 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

加密芯片程序运行正常的终止有两种方式，即 return 0 或 return 1。

当加密芯片检测到返回值为 1 时，会使用 A3 或 T=0 协议将 pOutBuf 中的数据输出给 MCU（与 MCU 发送 APDU 指令协议一致）。

当加密芯片检测到返回值为 0 时，会使用 A3 或 T=0 协议输出 2 字节数据 6A80 给 MCU（与 MCU 发送 APDU 指令协议一致）。

当加密芯片程序运行期间，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，而加密芯片内部无看门狗程序，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 包数据传入加密芯片中，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，加密芯片不支持在线调试。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

用户可将加密芯片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，MCU 可读出加密芯片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。

该方法适用于加密芯片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，加密芯片无法输出任何数据，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，用户仅需复位加密芯片后，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。加密芯片上电后，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

MCU 与加密芯片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，MCU 与加密芯片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。加密芯片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 MCU 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 MCU 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 MCU 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，当加密芯片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 在线下载

在线下载即先贴片，然后通过 MCU 进行在线下载算法。在线下载又分为明文下载和密文下载两种方式。用户可以使用该方式灵活升级加密芯片中的算法，保证产品的不断更新完善。客户若想进行远程更新算法，可以采用密文下载方式，将算法 hex 文件转换成密文格式在线传输给 MCU，后者将密文指令转发给加密芯片完成算法升级，该方法可有效防止线路跟踪，避免截获算法。

下面仅对方法一进行说明。如需在线下载算法，请与凌科芯安技术支持人员联系，获取在线下载算法说明文档。

### 5.2 脱机下载

#### 5.2.1 用 K100 开发板进行脱机下载

加密芯片放入 SOP8 的转接座（芯片的凹点或白点与图 5-1 中红圈对应）。将开发板与 PC 连接。

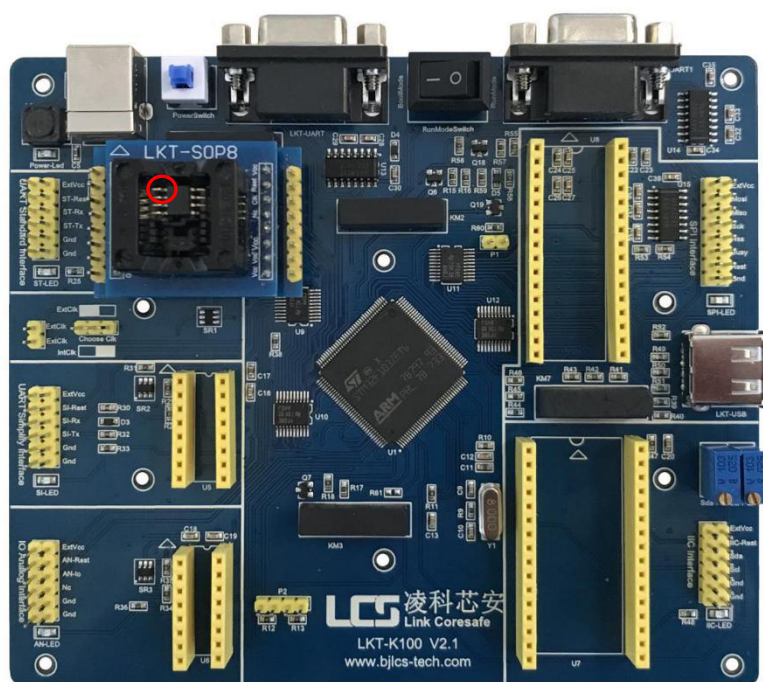


图 5-1：放入芯片

- 打开 LCS SAM 软件，如图 5-2 所示。
- 在“通讯设置”页面下选择“通信方式”为“HID”，“协议属性”为“UART”。
- 点击“连接”，会弹出“HID 读写器”的对话框，如图 5-2 所示。
- 点击“确定”按钮，会显示当前的连接状态和波特率，如图 5-3 所示。

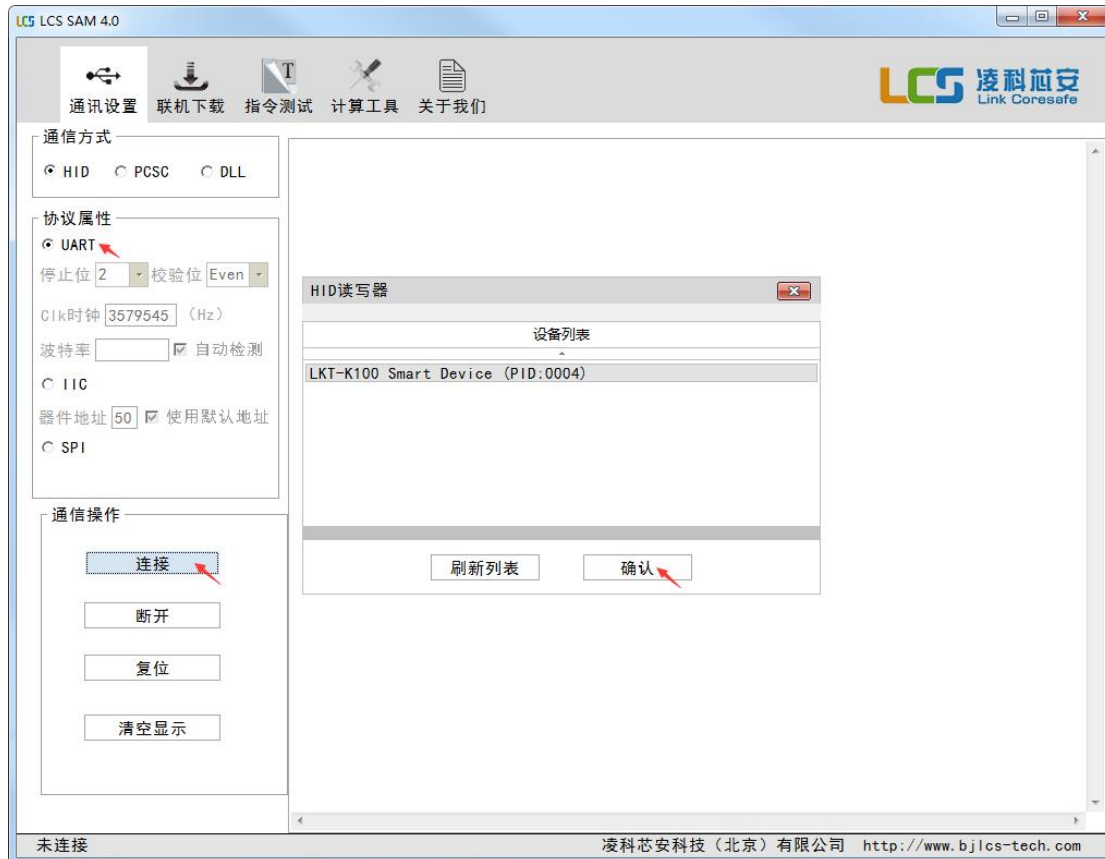


图 5-2

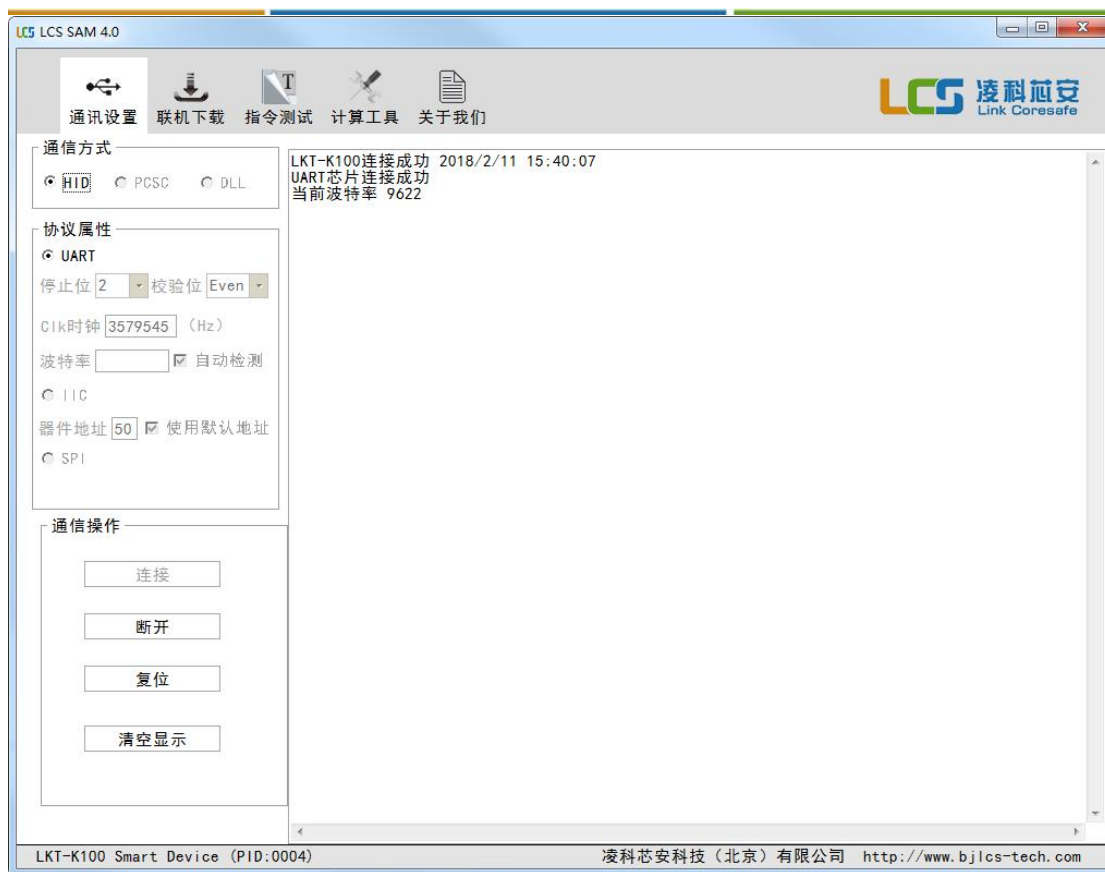


图 5-3

### 5.2.1.1 下载算法

通过 LCS SAM 软件配合 LKT-K100 开发板对加密芯片进行算法下载，然后贴片测试。

- 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。

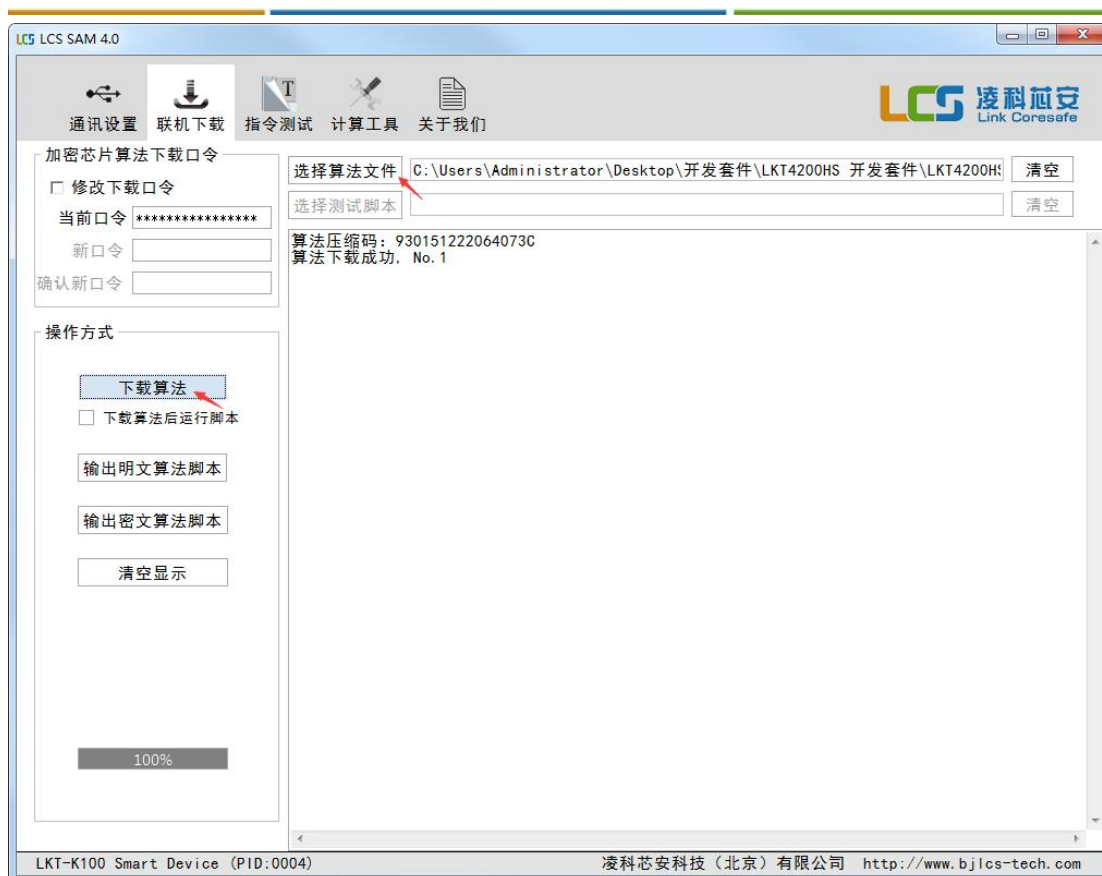


图 5-4：下载算法

### 5.2.1.2 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0000000000000000)。

### 5.2.1.3 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。

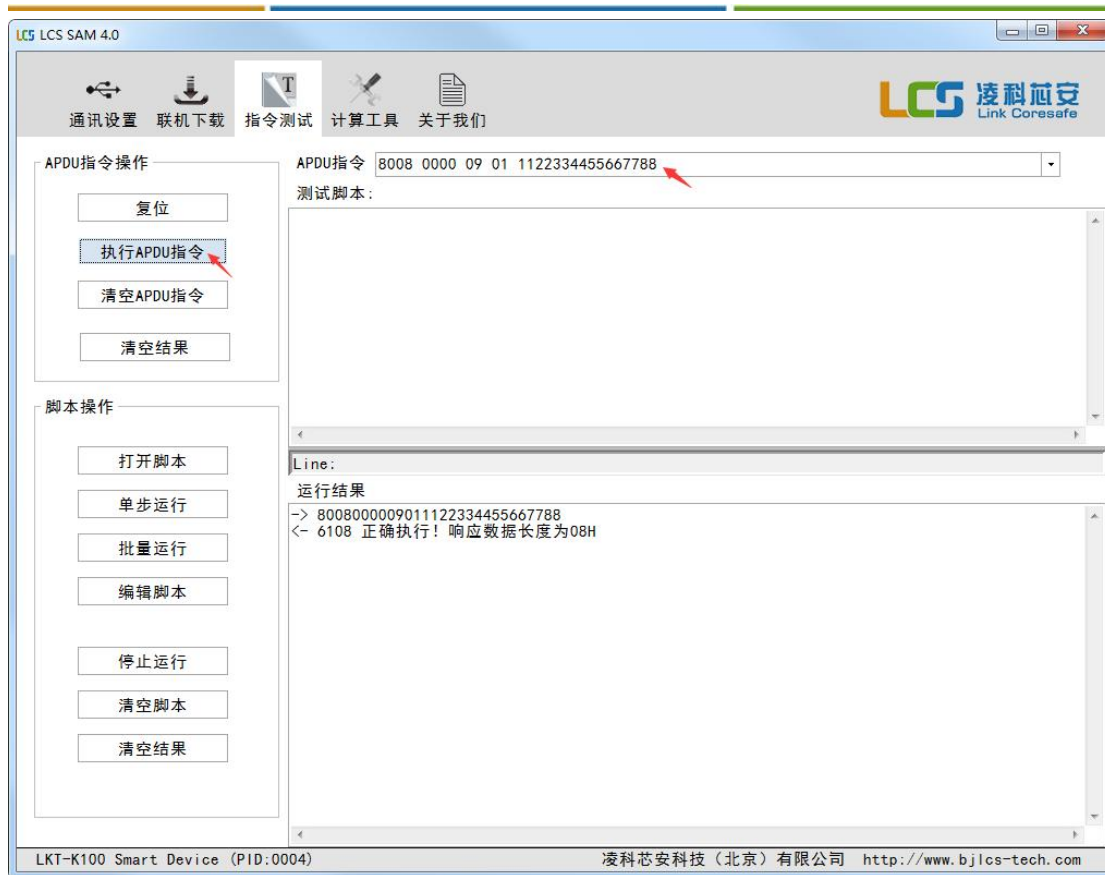


图 5-5: 发送指令

#### 5.2.1.4 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

## 5.2.2 用 P2000 烧录板进行脱机下载

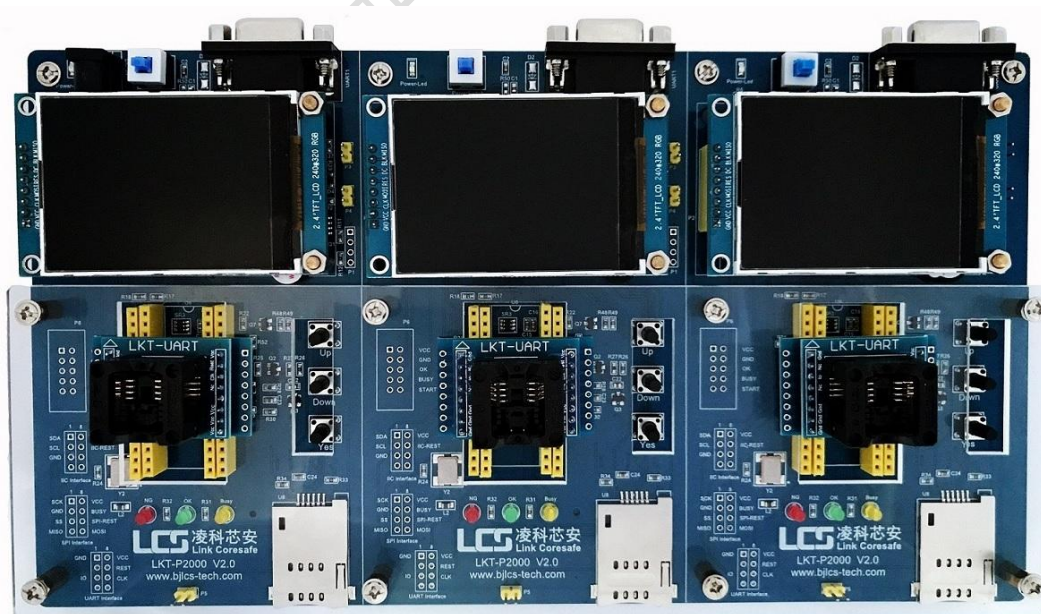


图 5-7：P2000 下载器

P2000 烧录板可同时烧录 3 颗芯片，详情请联系技术支持。

### 5.3 批量生产

用户可自行采购 LKT-L5 读写器、LKT-K100 开发板、LKT-K200 开发板、LKT-P2000 脱机烧录板、LKT-P2000 脱机烧录板+机械手等方式完成芯片烧录算法的工作，也可以委托我司完成芯片的算法烧录工作。

## 第 6 章 芯片封装

LKT4200HS 标准封装为 SOP8，如图 6-1 所示。也支持定制其他封装形式。

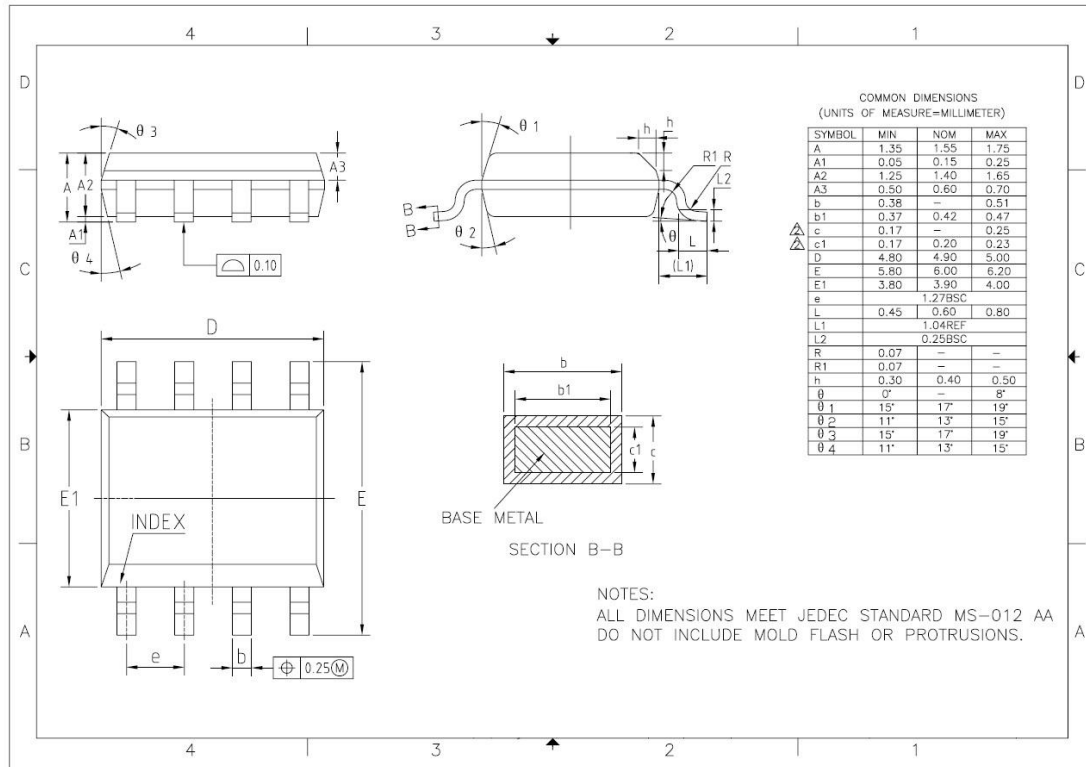


图 6-1: SOP8 封装图