

坚石诚信



ET-ARM 开发手册

V1.0

北京坚石诚信科技有限公司

网址: <http://www.jansh.com.cn>

修订记录:

修订日期	版本	修订内容
2013 年 7 月	V1.0	第一版发布

软件开发协议

坚石诚信科技股份有限公司（以下简称坚石）的所有产品，包括但不限于：开发工具包，磁盘，光盘，硬件设备和文档，以及未来的所有定单都受本协议的制约。如果您不愿接受这些条款，请在收到后的 7 天内将开发工具包寄回坚石，预付邮资和保险。我们会把货款退还给您，但要扣除运费和适当的手续费。

1. 许可使用

您可以将本软件合并、连接到您的计算机程序中，但其目的只是如开发指南中描述的那样保护该程序。您可以以存档为目的复制合理数量的拷贝。

2. 禁止使用

除在条款 1 中特别允许的之外，不得复制、反向工程、反汇编、反编译、修改、增加、改进软件、硬件和产品的其它部分。禁止对软件和产品任何部分进行反向工程，或企图推导软件的源代码。禁止使用产品中的磁性或光学介质来传递、存储非本产品的原始程序或由坚石提供的产品升级的任何数据。禁止将软件放在服务器上传播。

3. 有限担保

坚石保证在自产品交给您之日起的 12 个月内，在正常的使用情况下，硬件和软件存储介质没有重大的工艺和材料上的缺陷。

4. 修理限度

当根据本协议提出索赔时，坚石唯一的责任就是根据坚石的选择，免费进行替换或维修。坚石对更换后的任何产品部件都享有所有权。

保修索赔单必须在担保期内写好，在发生故障 14 天内连同令人信服的证据交给坚石。当将产品返还给坚石或坚石的授权代理商时，须预付运费和保险。

除了在本协议中保证的担保之外，坚石不再提供特别的或隐含的担保，也不再对本协议中所描述的产品负责，包括它们的质量，性能和对某一特定目的适应性。

5. 责任限度

不管因为什么原因，不管是因合同中的规定还是由于刑事的原因，包括疏忽的原因，而使您及任何一方受到了损失，由我方产品所造成的损失或该产品是起诉的原因或与起诉有间接关系，坚石对您及任何一方所承担的全部责任不超出您购买该产品所支付的货款。在任何情况下，坚石对于由于您不履行责任所导致的损失，或对于数据、利润、储蓄或其它的后续的和偶然的损失，即使坚石被建议有这种损失的可能性，或您根据第 3 方的索赔而提出的任何索赔均不负责任。

6. 协议终止

当您不能遵守本协议所规定的条款时，将终止您的许可和本协议。但条款 2, 3, 4, 5 将继续有效。

Software Developer's Agreement

All Products of Jansh Technologies Co., Ltd. (Jansh) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subject to the terms of this Agreement. If you do not agree with the terms herein, please return the evaluation package to us, postage and insurance prepaid, within seven days of their receipt, and we will reimburse you the cost of the Product, less freight and reasonable handling charges.

1. Allowable Use – You may merge and link the Software with other programs for the sole purpose of protecting those programs in accordance with the usage described in the Developer's Guide. You may make archival copies of the Software.
2. Prohibited Use – The Software or hardware or any other part of the Product may not be copied, reengineered, disassembled, decompiled, revised, enhanced or otherwise modified, except as specifically allowed in item 1. You may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. You may not use the magnetic or optical media included with the Product for the purposes of transferring or storing data that was not either an original part of the Product, or a Jansh provided enhancement or upgrade to the Product.
3. Warranty – Jansh warrants that the hardware and Software storage media are substantially free from significant defects of workmanship or materials for a time period of twelve (12) months from the date of delivery of the Product to you.
4. Breach of Warranty – In the event of breach of this warranty, Jansh's sole obligation is to replace or repair, at the discretion of Jansh, any Product free of charge. Any replaced Product becomes the property of Jansh.

Warranty claims must be made in writing to Jansh during the warranty period and within fourteen (14) days after the observation of the defect. All warranty claims must be accompanied by evidence of the defect that is deemed satisfactory by Jansh. Any Products that you return to Jansh, or a Jansh authorized distributor, must be sent with freight and insurance prepaid.

EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY OR REPRESENTATION OF THE PRODUCT, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Jansh's Liability – Jansh's entire liability to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of the Product that caused the damages or are the subject of, or indirectly related to the cause of action. In no event shall Jansh be liable for any damages caused by your failure to meet your obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if Jansh has been

advised of the possibility of damages, or for any claim by you based on any third-party claim.

6. Termination – This Agreement shall terminate if you fail to comply with the terms herein. Items 2, 3, 4 and 5 shall survive any termination of this Agreement.

目 录

ET-ARM 开发手册	1
Software Developer's Agreement.....	iii
ET-ARM 用户程序开发.....	2
用户 API 接口	2
1.Dongle_Enum 枚举加密锁	2
2.Dongle_Open 打开加密锁	3
3.Dongle_ResetState 清除状态	3
4.Dongle_Close 关闭加密锁	4
5.Dongle_GenRandom 产生随机数	4
6.Dongle_LEDControl LED 控制	4
7.Dongle_SwitchProtocol 切换通讯协议	5
8.Dongle_RFS 一键恢复	5
9.Dongle_CreateFile 创建文件	6
10.Dongle_WriteFile 写文件	9
11.Dongle_ReadFile 读取数据文件	9
12.Dongle_DownloadExeFile 批量下载可执行文件	10
13.Dongle_RunExeFile 运行可执行程序	11
14.Dongle_DeleteFile 删除文件	11
15.Dongle_ListFile 列文件	12
16.Dongle_GenUniqueKey 唯一化锁	13
17.Dongle_VerifyPIN 校验密码	14
18.Dongle_ChangePIN 更改密码	14
19.Dongle_ResetUserPIN 重置用户 PIN 码	15
20.Dongle_SetUserID 设置用户 ID	15
21.Dongle_GetDeadline 获取到期时间	16
22.Dongle_SetDeadline 设置到期时间	16
23.Dongle_GetUTCTime 获取 UTC 时间	17
24.Dongle_ReadData 读取数据存储区	17
25.Dongle_WriteData 写入数据存储区	18
26.Dongle_ReadShareMemory 读取共享内存数据	18
27.Dongle_WriteShareMemory 写入共享内存数据	18
28.Dongle_RsaGenPubPriKey 产生 RSA 公私钥	19
29.Dongle_RsaPri RSA 私钥运算	20
30.Dongle_RsaPub RSA 公钥运算	21
31.Dongle_EccGenPubPriKey 产生 ECC 公私钥	21
32.Dongle_EccSign ECC 私钥签名	22
33.Dongle_EccVerify ECC 公钥验签	23
34.Dongle_SM2GenPubPriKey 产生 SM2 公私钥	23
35.Dongle_SM2Sign SM2 私钥签名	24
36.Dongle_SM2Verify SM2 公钥验签	24
37.Dongle_TDES 3DES 加解密	25
38.Dongle_SM4 SM4 加解密	25
39.Dongle_HASH HASH 运算	26
40.Dongle_Seed 种子码算法	26
41.Dongle_LimitSeedCount 设置种子码运算次数	27
42.Dongle_GenMotherKey 制作母锁	27
43.Dongle_RequestInit 空锁获取生产请求	28
44.Dongle_GetInitDataFromMother 从母锁获取生产数据	29
45.Dongle_InitSon 生产子锁	29
46.Dongle_SetUpdatePriKey 设置远程升级私钥	30
47.MakeUpdatePacket 制作远程升级包	30
48.Dongle_MakeUpdatePacketFromMother 母锁制作远程升级包	31

49.Dongle_Update 远程升级子锁.....	33
用户 API 错误码.....	33
ET-ARM 锁内可执行程序开发.....	35
编译环境安装	35
编写锁内可执行程序	37
编程内存说明	38
扩展缓冲区.....	38
输入输出缓冲区.....	39
用户编程空间.....	39
锁内系统 API 接口.....	40
1.create_file 创建文件.....	40
2.read_file 读文件	42
3.write_file 写文件	42
4.delete_file 删除文件.....	43
5.get_keyinfo 获取加密锁信息	43
6.led_control LED 控制	44
7.get_random 产生随机数	44
8.get_tickcount 取锁上电时间	45
9.get_realtime 取真实时间	45
10.get_expiretime 到期时间	45
11.get_sharememory 读共享内存	46
12.set_sharememory 写共享内存区.....	46
13.rsa_genkey 产生 RSA 公私钥.....	46
14.rsa_pri RSA 私钥加解密运算.....	47
15.rsa_pub RSA 公钥加解密运算.....	48
16.ecc_genkey 产生 ECC 公私钥	48
17.ecc_sign ECC 私钥运算	49
18.ecc_verify ECC 公钥运算	49
19.sm2_genkey 产生 SM2 公私钥.....	50
20.sm2_sign SM2 私钥运算.....	50
21.sm2_verify SM2 公钥运算	51
22.tdes TDES 加解密运算	51
23.sm4 SM4 加解密运算	51
24.sha1 SHA1 运算.....	52
25.sm3 SM3 运算.....	52
26.seed 种子码运算	52
锁内系统 API 错误码.....	54

说 明

ET-ARM 系列加密锁都是采用的 32 位 ARM 高性能智能卡芯片，对于 ET-ARM 系列加密锁的开发，我们提供了一套丰富的 API 接口和一套多语言的示例程序，最大程度的方便了用户的使用。并且我们还提供了锁内系统的接口函数，方便用户直接调用。

本指南着重为用户讲解 API 接口以及锁内系统函数的使用方法。在此之前，将对我们所提供的示例程序进行一个简单的说明，以便用户使用示例程序进行参考。我们向用户提供的示例程序包括：VC、VB6.0、PB（PB8、PB10）、Java、Delphi7、C#、BCB2007 以及 ARM 可执行程序。每种语言的示例程序都几乎涵盖了 ET-ARM 的所有接口，基本上都实现了 ET-ARM 的所有功能。每种语言的示例程序都提供了十多个示例程序，而且每个例子都有相关的说明。另外，坚石还可以根据用户的需求丰富和完善相关的示例程序。

ET-ARM 用户程序开发

ET-ARM 加密锁是一款 ARM 内核的加密锁，上层提供了丰富的应用程序接口，支持多种系统和多种开发语言。用户 API 接口提供了 ET-ARM 加密锁几乎所有的功能，其中包括开发、生产、远程升级等。在通讯方面，API 通讯采用双向认证和加密方式，有效的防止了通讯数据的模拟破解。下面将一一介绍 ET-ARM 用户 API 接口。

用户 API 接口

1.Dongle_Enum 枚举加密锁

DWORD WINAPI Dongle_Enum(DONGLE_INFO * pDongleInfo, int * pCount);

说明:

本函数最多会枚举出 32 个 HID 设备和 32 个 CCID 设备。

参数:

pDongleInfo [out] 设备信息的数组。当此参数为 NULL 时, pCount 返回找到的设备的数目。

pCount [out] 设备数目。该函数最多可以同时枚举出 32 个 HID 设备和 32 个 CCID 设备。

返回值:

DONGLE SUCCESS 执行成功。

注意：

为使后续的介绍更加方便，先将加密锁信息相关的结构体和定义进行说明：

```
#define CONST_PID 0xFFFFFFFF //出厂时默认的 PID
```

```
#define CONST_USERPIN          "12345678"    //出厂时默认的 USERPIN
```

```
#define CONST_ADMINPIN "FFFFFFFFFFFFFFFF" //出厂时默认的 ADMINPIN
```

默认的 PIN 码重试次数为无限制。根据种子码初始化锁时会同时初始化 PID 和 ADMINPIN (PID 不可更改, ADMINPIN 可更改)。

锁的信息:

```
typedef struct
{
    unsigned short m_Ver;           //COS 版本,比如:0x0201,表示 2.01 版
    unsigned short m_Type;         //产品类型: 0xFF 标准版, 0x00 时钟锁,0x02 U 盘锁
    unsigned char m_BirthDay[8];   //出厂日期
    unsigned long m_Agent;         //代理商编号,比如:默认的 0xFFFFFFFF
    unsigned long m_PID;           //产品 ID
    unsigned long m_UserID;        //用户 ID
    unsigned char m_HID[8];        //8 字节的硬件 ID
    unsigned long m_IsMother;      //母锁标志: 0x01 表示是母锁, 0x00 表示不是母锁
    unsigned long m_DevType;       //设备类型(PROTOCOL_HID 或者 PROTOCOL_CCID)
} DONGLE_INFO;
```

2.Dongle_Open 打开加密锁

*DWORD WINAPI Dongle_Open(DONGLE_HANDLE * phDongle, int nIndex);*

说明:

打开指定的加密锁。

参数:

phDongle [out] 句柄指针。如果打开成功,会被填充。

index [in] 基于 0 的索引值。指示打开找到的第几把加密锁。

返回值:

DONGLE_SUCCESS 打开成功。

3.Dongle_ResetState 清除状态

DWORD WINAPI Dongle_ResetState(DONGLE_HANDLE hDongle);

说明:

清除 PIN 码验证状态。将加密锁状态变为匿名。

参数:

hDongle [in] 打开的加密锁句柄。

返回值:

DONGLE_SUCCESS 执行成功。

4.Dongle_Close 关闭加密锁

DWORD WINAPI Dongle_Close(DONGLE_HANDLE hDongle);

说明:

关闭打开的加密锁。

参数:

hDongle [in] 打开的加密锁句柄。

返回值:

DONGLE_SUCCESS 关闭成功。

5.Dongle_GenRandom 产生随机数

*DWORD WINAPI Dongle_GenRandom(DONGLE_HANDLE hDongle, int nLen, BYTE * pRandom);*

说明:

匿名权限即可操作。

参数:

hDongle [in] 打开的加密锁句柄。

nLen [in] 要产生的随机数的长度，nLen 的取值范围为 1~128。

pRandomBuffer [out] 随机数缓冲区。

返回值:

DONGLE_SUCCESS 获取随机数成功。

6.Dongle_LEDControl LED 控制

DWORD WINAPI Dongle_LEDControl(DONGLE_HANDLE hDongle, int nFlag);

说明:

LED 灯的控制操作。匿名权限即可操作。

参数:

hDongle [in] 打开的加密锁句柄。

nFlag [in] 控制类型，例如：nFlag = LED_ON，表示控制 LED 为亮的状态；

nFlag = LED_OFF, 表示控制 LED 为灭的状态;

nFlag = LED_BLINK, 表示控制 LED 为闪烁的状态。

返回值:

DONGLE_SUCCESS 命令执行成功。

注意:

LED 灯状态定义如下:

```
#define LED_OFF          0 //灯灭
#define LED_ON           1 //灯亮
#define LED_BLINK        2 //灯闪
```

7.Dongle_SwitchProtocol 切换通讯协议

DWORD WINAPI Dongle_SwitchProtocol(DONGLE_HANDLE hDongle, int nFlag);

说明:

调用执行成功后加密锁会自动重启,打开的句柄 hDongle 会无效,如需操作,请重新枚举并打开锁。该操作必须要验证开发商 PIN 码之后放可使用。

参数:

hDongle [in] 打开的加密锁句柄。

nFlag [in] 协议类型。例如: nFlag == PROTOCOL_HID, 表示将加密锁切换为 HID 设备;

nFlag == PROTOCOL_CCID, 表示将加密锁切换为 CCID 设备

返回值:

DONGLE_SUCCESS 执行成功。

注意:

通讯协议类型定义如下:

```
#define PROTOCOL_HID      0 //hid 协议
#define PROTOCOL_CCID     1 //ccid 协议
```

8.Dongle_RFS 一键恢复

DWORD WINAPI Dongle_RFS(DONGLE_HANDLE hDongle);

说明:

返回出厂状态, 加密锁的 PID、用户 PIN 码、开发商 PIN 码等, 全部恢复到出厂状态, 所有写入的数据都将被清空。另外, 调用执行成功后加密锁会自动重启, 打开的句柄 hDongle 会被自动关闭, 如需操作, 请重新枚举并打开锁。该操作必须要验证开发商 PIN 码之后方可使用。

参数:

hDongle [in] 打开的加密锁句柄。

返回值:

DONGLE_SUCCESS 执行成功。

9.Dongle_CreateFile 创建文件

DWORD WINAPI Dongle_CreateFile(DONGLE_HANDLE hDongle, int nFileType, WORD wFileID, void * pFileAttr);

说明:

该函数不支持可执行文件的创建。该操作需要开发商权限。

参数:

hDongle [in] 打开的加密锁句柄。

nFileType [in] 文件类型, nFileType = FILE_DATA, 表示创建数据文件;

nFileType = FILE_PRIKEY_RSA, 表示创建 RSA 私钥文件;

nFileType = FILE_PRIKEY_ECCSM2, 表示创建 ECCSM2 私钥文件;

nFileType = FILE_KEY, 表示创建 SM4 和 3DES 密钥文件;

不支持 nFileType = FILE_EXE 的文件类型。

wFileID [in] 文件 ID。

pFileAttr [in] 文件的属性。参数的结构为: DATA_FILE_ATTR、PRIKEY_FILE_ATTR 或 KEY_FILE_ATTR。

返回值:

DONGLE_SUCCESS 创建文件成功。

注意:

锁内文件说明

1.RSA 私钥文件允许创建的最大数量为 8 个

2.ECCSM2 私钥文件允许创建的最大数量为 16 个

3.3DES/SM4 密钥文件允许创建的最大数量为 32 个

4.可执行文件允许创建的最大数量为 64 个,总大小不能超过 64K

5..数据文件创建个数受锁内空间大小和文件系统其他因素的影响，最大个数不超过 54 个。例如，文件大小不超过 252 字节，最多可创建 54 个文件；如果文件大小设为 1024 字节，最多可创建 31 个文件，如果文件大小设为 4096 字节，最多可创建 9 个文件。由于实际应用中文件大小情况比较复杂，可自行测试确定文件个数。

文件类型定义如下：

```
#define FILE_DATA           1 //普通数据文件
#define FILE_PRIKEY_RSA     2 //RSA 私钥文件
#define FILE_PRIKEY_ECCSM2  3 //ECC 或者 SM2 私钥文件(SM2 私钥文件时 ECC 私钥文件的子集)
#define FILE_KEY            4 //SM4 和 3DES 密钥文件
#define FILE_EXE            5 //可执行文件
```

数据文件授权结构

```
typedef struct
{
    unsigned short m_Read_Priv; //读权限：0 为最小匿名权限；1 为最小用户权限；2 为最小开发
    商权限
    unsigned short m_Write_Priv; //写权限：0 为最小匿名权限；1 为最小用户权限；2 为最小开发
    商权限
} DATA_LIC;
```

数据文件属性数据结构

```
typedef struct
{
    unsigned long m_Size; //数据文件长度，该值最大不能超过 4096
    DATA_LIC m_Lic; //授权
} DATA_FILE_ATTR;
```

私钥文件授权结构

```
typedef struct
```

```
{
    long          m_Count;      //可调次数: 0xFFFFFFFF 表示不限制, 递减到 0 表示已不可调用
    unsigned char m_Priv;      //调用权限: 0 为最小匿名权限; 1 为最小用户权限; 2 为最小开发
    商权限
    unsigned char m_IsDecOnRAM; //是否是在内存中递减: TRUE 为在内存中递减, FALSE 为在
    FLASH 中递减
    unsigned char m_IsReset;    //用户态调用后是否自动回到匿名态: TRUE 为调后回到匿名态
    (开发商态不受此限制)
    unsigned char m_Reserve;    //保留,用于 4 字节对齐
} PRIKEY_LIC;
```

ECCSM2/RSA 私钥文件属性数据结构

```
typedef struct
{
    unsigned short m_Type;      //数据类型:ECCSM2 私钥 或 RSA 私钥
    unsigned short m_Size;      //数据长度:RSA 是 1024 位 或 2048 位, ECC 是 192 位或 256
    位, SM2 是 0x8100 位
    PRIKEY_LIC     m_Lic;      //授权
} PRIKEY_FILE_ATTR;
```

对称加密算法(SM4/TDES)密钥文件授权结构

```
typedef struct
{
    unsigned long m_Priv_Enc;   //加密时的调用权限: 0 为最小匿名权限 1 为最小用户权限 2
    为最小开发商权限
} KEY_LIC;
```

对称加密算法密钥文件授权结构

```
typedef struct
{
    unsigned short m_Priv_Exe;  //运行的权限: 0 为最小匿名权限 1 为最小用户权限 2 为最小
    开发商权限
} EXE_LIC;
```

对称加密算法(SM4/TDES)密钥文件属性数据结构

```
typedef struct
{
```

```

unsigned long  m_Size;           //密钥数据长度=16
KEY_LIC       m_Lic;           //授权

} KEY_FILE_ATTR;
    
```

10.Dongle_WriteFile 写文件

DWORD WINAPI Dongle_WriteFile(DONGLE_HANDLE hDongle, int nFileType, WORD wFileID, WORD wOffset, BYTE * pInData, int nDataLen);

说明:

该函数不支持可执行文件的写入操作，且该操作需要开发商权限。

参数:

hDongle [in] 打开的加密锁句柄。

nFileType [in] 文件类型。例如，nFileType = FILE_DATA，表示创建数据文件；

nFileType = FILE_PRIKEY_RSA，表示创建 RSA 私钥文件；

nFileType = FILE_PRIKEY_ECCSM2，表示创建 ECCSM2 私钥文件；

nFileType = FILE_KEY，表示创建 SM4 和 3DES 密钥文件；

不支持 nFileType = FILE_EXE 的文件类型。

wFileID [in] 文件 ID。

wOffset [in] 文件偏移。文件写入的起始偏移量。

pInData [in] 准备写入的数据。

nDataLen [in] 参数 pInData 的大小。

返回值:

DONGLE_SUCCESS 写入文件成功。

11.Dongle_ReadFile 读取数据文件

DWORD WINAPI Dongle_ReadFile(DONGLE_HANDLE hDongle, WORD wFileID, WORD wOffset, BYTE * pOutData, int nDataLen);

说明:

读取加密锁内的数据文件。数据文件的读取权限取决于创建时的设定。

参数:

hDongle	[in]	打开的加密锁句柄。
wFileID	[in]	文件 ID。
wOffset	[in]	文件偏移量。
pOutData	[in]	数据缓冲区。
nDataLen	[out]	参数 pOutData 的长度。

返回值:

DONGLE_SUCCESS	读取数据文件成功
----------------	----------

12.Dongle_DownloadExeFile 批量下载可执行文件

*DWORD WINAPI Dongle_DownloadExeFile(DONGLE_HANDLE hDongle, EXE_FILE_INFO * pExeFileInfo, int nCount);*

说明:

锁内可执行文件的数量不能超过 64 个，可执行文件的总大小不能超过 64K，该操作需要验证管理员权限。

参数:

hDongle	[in]	打开的加密锁句柄。
pExeFileInfo	[in]	可执行文件信息的数组。
nCount	[in]	即可执行文件的数量。

返回值:

DONGLE_SUCCESS	批量下载可执行文件成功。
----------------	--------------

可执行文件属性数据结构

```
typedef struct
{
    EXE_LIC      m_Lic;      //授权
    unsigned short m_Len;    //文件长度
} EXE_FILE_ATTR;
```

下载和列可执行文件时填充的数据结构

```
typedef struct
```

```
{
    unsigned short  m_dwSize;           //可执行文件大小
    unsigned short  m_wFileID;         //可执行文件 ID
    unsigned char   m_Priv;            //调用权限：0 为最小匿名权限 1 为最小用户权限 2 为
最小开发商权限
    unsigned char   *m_pData;          //可执行文件数据
}EXE_FILE_INFO;
```

13.Dongle_RunExeFile 运行可执行程序

*DWORD WINAPI Dongle_RunExeFile(DONGLE_HANDLE hDongle, WORD wFileID, BYTE * pInOutBuf, WORD wInOutBufLen, int * pMainRet);*

说明：

运行指定的锁内可执行程序。运行可执行文件的权限，取决于批量下载时每个可执行文件的设置，即，EXE_FILE_INFO 中的 m_Priv 参数。wInDataLen 最大长度不超过 1024，参数 pInOutBuf 对应锁内可执行程序中的 InOutBuf。

参数：

hDongle	[in]	打开的加密锁句柄。
wFileID	[in]	可执行文件的文件 ID。
pInOutBuf	[in,out]	输入数据的缓冲区。
wInOutBufLen	[in]	输入数据缓冲区 pInOutBuf 的大小。
pMainRet	[out]	锁内可执行程序 main 函数的返回值，如果无需改值，可以为传入参数 NULL。

返回值：

DONGLE_SUCCESS 运行指定的可执行文件成功。

14.Dongle_DeleteFile 删除文件

DWORD WINAPI Dongle_DeleteFile(DONGLE_HANDLE hDongle, int nFileType, WORD wFileID);

说明：

需要开发商权限。

参数：

hDongle	[in]	打开的加密锁句柄。
---------	------	-----------

nFileType [in] 文件类型。

wFileID [in] 文件 ID。

返回值:

DONGLE_SUCCESS 删除文件成功。

15.Dongle_ListFile 列文件

DWORD WINAPI Dongle_ListFile(DONGLE_HANDLE hDongle, int nFileType, void pFileList, int * pDataLen);*

说明:

需要开发商权限。

参数:

hDongle [in] 打开的加密锁句柄。

nFileType [in] 指示文件类型。例如, FILE_DATA 等。

pFileList [in] pFileList: 输出文件的列表 (此参数为 NULL 时, pLen 中返回所需的缓冲区长度),

当 nFileType = FILE_DATA 时, pFileList 指向 DATA_FILE_LIST 结构;

当 nFileType = FILE_PRIKEY_RSA 时, pFileList 指向 PRIKEY_FILE_LIST 结构;

当 nFileType = FILE_KEY 时, pFileList 指向 KEY_FILE_LIST 结构;

当 nFileType = FILE_EXE 时, pFileList 指向 EXE_FILE_LIST 结构。

pDataLen [in,out] 参数 pFileList 的输入长度, 执行成功返回 pFileList 的字节长度。

返回值:

DONGLE_SUCCESS 列文件成功。

获取数据文件列表时返回的数据结构

```
typedef struct
{
    unsigned short    m_FILEID; //文件 ID
    unsigned short    m_Reserve; //保留,用于 4 字节对齐
    DATA_FILE_ATTR    m_attr;    //文件属性
}DATA_FILE_LIST;
```

获取私钥文件列表时返回的数据结构

```
typedef struct
{
    unsigned short    m_FILEID;    //文件 ID
    unsigned short    m_Reserve;    //保留,用于 4 字节对齐
    PRIKEY_FILE_ATTR  m_attr;      //文件属性
}PRIKEY_FILE_LIST;
```

获取 SM4 及 TDES 密钥文件列表时返回的数据结构

```
typedef struct
{
    unsigned short    m_FILEID;    //文件 ID
    unsigned short    m_Reserve;    //保留,用于 4 字节对齐
    KEY_FILE_ATTR     m_attr;      //文件属性
}KEY_FILE_LIST;
```

获取可执行文件列表时返回的数据结构

```
typedef struct
{
    unsigned short    m_FILEID;    //文件 ID
    EXE_FILE_ATTR     m_attr;
    unsigned short    m_Reserve;    //保留,用于 4 字节对齐
}EXE_FILE_LIST;
```

16.Dongle_GenUniqueKey 唯一化锁

*DWORD WINAPI Dongle_GenUniqueKey(DONGLE_HANDLE hDongle,int nSeedLen, BYTE * pSeed, char * pPIDstr, char * pAdminPINstr);*

说明:

输入种子码产生 PID 和开发商 PIN 码,需要开发商权限,执行成功后加密锁自动回到匿名态。产生开发商 PIN 码的目的是为了使密码肯定不是弱密码,产生的开发商 PIN 可以借助 Dongle_ChangePIN 进行更改。另外,种子码一定要牢记,否则任何人永远无法得知开发商 PIN 码。

参数:

hDongle	[in]	打开的加密锁句柄。
nSeedLen	[in]	参数 pSeed 的缓冲区长度。
pSeed	[in]	种子码的缓冲区。

pPIDStr [out] 函数执行成功返回 PID。该缓冲区大小至少应该为 8 字节，返回一个 8 字节的以 0 终止的 ansi 字符串。

pAdminPINstr [out] 函数执行成功返回开发商 PIN 码。该缓冲区大小至少应该为 16 字节，返回字符串长度为 16 字节的以 0 终止的 ansi 字符串。

返回值:

DONGLE_SUCCESS 唯一化锁成功。

17.Dongle_VerifyPIN 校验密码

*DWORD WINAPI Dongle_VerifyPIN(DONGLE_HANDLE hDongle, int nFlags, char * pPIN, int * pRemainCount);*

说明:

校验密码。

参数:

hDongle	[in]	打开的加密锁句柄。
nFlags	[in]	PIN 码类型。参数取值为 FLAG_USERPIN 或者 FLAG_ADMINPIN。
pPIN	[in]	PIN 码。
pRemainCount	[out]	剩余重试次数。返回 0 表示已锁死；1~253 表示剩余次数；255 表示 unlimited 重试次数。

返回值:

DONGLE_SUCCESS 校验成功。

如果校验失败，函数的返回值中也含有剩余的重试次数，(错误码 & 0xFFFFF00) == DONGLE_INCORRECT_PIN，即后两位表示剩余次数。

注意:

PIN 码类型定义如下:

```
#define FLAG_USERPIN          0 //用户 PIN
#define FLAG_ADMINPIN         1 //开发商 PIN
```

18.Dongle_ChangePIN 更改密码

*DWORD WINAPI Dongle_ChangePIN(DONGLE_HANDLE hDongle, int nFlags, char * pOldPIN, char **

pNewPIN, int nTryCount);

说明:

更改密码。

参数:

hDongle	[in]	打开的加密锁句柄。
nFlags	[in]	PIN 码类型。参数取值为 FLAG_USERPIN 或者 FLAG_ADMINPIN。
pOldPIN	[in]	旧的 PIN 码缓冲区。必须是一个字符串长度为 16 字节的 0 终止的 ansi 字符串，且可以是中文。
pNewPIN	[in]	新的 PIN 码缓冲区。必须是一个字符串长度为 16 字节的 0 终止的 ansi 字符串。
nTryCount	[in]	重试次数。该参数的取值范围为 1~255,其中 255 表示不限制重试次数。

返回值:

DONGLE_SUCCESS 修改密码成功。

19.Dongle_ResetUserPIN 重置用户 PIN 码

*DWORD WINAPI Dongle_ResetUserPIN(DONGLE_HANDLE hDongle, char * pAdminPIN);*

说明:

重置用户 PIN 码。空锁(即 PID=FFFFFFFF)不能重置密码。执行成功后用户密码恢复为出厂默认 CONST_USERPIN。

参数:

hDongle	[in]	打开的加密锁句柄。
pAdminPIN	[in]	开发商 PIN 码缓冲区，长度为 16 字节的 0 终止的 ansi 字符串。

返回值:

DONGLE_SUCCESS 重置用户 PIN 码成功。

20.Dongle_SetUserID 设置用户 ID

DWORD WINAPI Dongle_SetUserID(DONGLE_HANDLE hDongle, DWORD dwUserID);

说明:

设置用户 PIN 码。需要开发商权限。

参数:

hDongle [in] 打开的加密锁句柄。

dwUserID [in] 用户 ID。

返回值:

DONGLE_SUCCESS 重置用户 PIN 码成功。

21.Dongle_GetDeadline 获取到期时间

*DWORD WINAPI Dongle_GetDeadline(DONGLE_HANDLE hDongle, DWORD * pdwTime);*

说明:

获取加密锁到期时间，匿名权限获取。

参数:

hDongle [in] 打开的加密锁句柄。

pdwTime [out] 获取的到期 UTC 时间值。

若 *pdwTime = 0xFFFFFFFF，表示不限制到期时间

若 (*pdwTime & 0xFFFF0000) == 0，值表示还剩余几小时

若 (*pdwTime & 0xFFFF0000) != 0，值表示到期的 UTC 的时间，可以通过 gmtime 等将该值进行转换。

返回值:

DONGLE_SUCCESS 获取加密锁到期时间成功。

22.Dongle_SetDeadline 设置到期时间

DWORD WINAPI Dongle_SetDeadline(DONGLE_HANDLE hDongle, DWORD dwTime);

说明:

设置加密锁的到期时间。该操作需要管理员权限。

参数:

hDongle [in] 打开的加密锁句柄。

dwTime [in] 时间值，说明：1.设置可用小时数，范围在 1~65535，例如 dwTime = 24；这种情况在校验了用户 PIN 码后开始计时；

- 2.设置到期的年月日时分秒。可通过函数 `time` 或者 `mktime` 取得即时的 `utc` 时间值(`utc` 值都大于 65535);
- 3.取消到期时间限制, 此时 `dwTime` 的值只能为 `0xFFFFFFFF`。

返回值:

`DONGLE_SUCCESS` 设置加密锁到期时间成功。

23.Dongle_GetUTCTime 获取 UTC 时间

DWORD WINAPI Dongle_GetUTCTime(DONGLE_HANDLE hDongle, DWORD * pdwUTCTime);

说明:

获取加密锁的 UTC 时间。

参数:

`hDongle` `[in]` 打开的加密锁句柄。

`pdwUTCTime` `[in]` 时间值。

返回值:

`DONGLE_SUCCESS` 设置加密锁到期时间成功。

24.Dongle_ReadData 读取数据存储区

DWORD WINAPI Dongle_ReadData(DONGLE_HANDLE hDongle, int nOffset, BYTE* pData, int nDataLen);

说明:

读取锁内数据区数据。数据区大小共 8k, 前 4k(0~4095)的读写没有权限限制, 后 4k(4096~8191)任意权限可读, 但是只有开发商权限可写。

参数:

`hDongle` `[in]` 打开的加密锁句柄。

`nOffset` `[in]` 起始偏移。范围在 0~8191。

`pData` `[out]` 读取的数据缓冲区。

`nDataLen` `[in]` 参数 `pData` 的缓冲区大小。

返回值:

`DONGLE_SUCCESS` 读取数据区数据成功。

25.Dongle_WriteData 写入数据存储区

*DWORD WINAPI Dongle_WriteData(DONGLE_HANDLE hDongle, int nOffset, BYTE * pData, int nDataLen);*

说明:

写入锁内数据区数据。数据区大小共 8k，前 4k(0~4095)的读写没有权限限制，后 4k(4096~8191)任意权限可读，但是只有开发商权限可写。

参数:

hDongle	[in]	打开的加密锁句柄。
nOffset	[in]	起始偏移。范围在 0~8191。
pData	[in]	写入的数据缓冲区。
nDataLen	[in]	参数 pData 的缓冲区大小。

返回值:

DONGLE_SUCCESS 写入数据区数据成功。

26.Dongle_ReadShareMemory 读取共享内存数据

*DWORD WINAPI Dongle_ReadShareMemory(DONGLE_HANDLE hDongle, BYTE * pData);*

说明:

获取共享内存数据。共享内存总大小为 32 字节，没有权限限制，掉电后数据自动擦除。

参数:

hDongle	[in]	打开的加密锁句柄。
pData	[out]	输出的数据。输出共享内存的数据，固定为 32 个字节。

返回值:

DONGLE_SUCCESS 获取共享内存数据成功。

27.Dongle_WriteShareMemory 写入共享内存数据

*DWORD WINAPI Dongle_WriteShareMemory(DONGLE_HANDLE hDongle, BYTE * pData, int nDataLen);*

说明:

设置共享内存数据。没有权限限制，掉电后数据自动擦除。

参数:

hDongle	[in]	打开的加密锁句柄。
pData	[in]	输入数据。
nDataLen	[in]	参数 pData 的缓冲区大小。长度不能超过 32。

返回值:

DONGLE_SUCCESS	设置共享内存数据成功。
----------------	-------------

28.Dongle_RsaGenPubPriKey 产生 RSA 公私钥

*DWORD WINAPI Dongle_RsaGenPubPriKey(DONGLE_HANDLE hDongle, WORD wPriFileID, RSA_PUBLIC_KEY * pPubBakup, RSA_PRIVATE_KEY * pPriBakup);*

说明:

产生 RSA 公钥和私钥, 使用该函数之前需要先创建一个 RSA 私钥文件, 需要开发商权限。成功后注意保存 RSA 公私钥数据。

RSA 公钥格式定义如下:

```
typedef struct {
    unsigned int bits;           // length in bits of modulus
    unsigned int modulus;       // modulus
    unsigned char exponent[256]; // public exponent
} RSA_PUBLIC_KEY;
```

RSA 私钥格式定义如下:

```
typedef struct {
    unsigned int bits;           // length in bits of modulus
    unsigned int modulus;       // modulus
    unsigned char publicExponent[256]; // public exponent
    unsigned char exponent[256];      // private exponent
} RSA_PRIVATE_KEY;
```

参数:

hDongle	[in]	打开的加密锁句柄。
wPriFileID	[in]	RSA 私钥文件 ID。
pPubBakup	[out]	RSA 公钥数据。
pPriBakup	[out]	RSA 私钥数据。

返回值:

DONGLE_SUCCESS 产生 RSA 公私钥成功。

29.Dongle_RsaPri RSA 私钥运算

*DWORD WINAPI Dongle_RsaPri(DONGLE_HANDLE hDongle, WORD wPriFileID, int nFlag, BYTE * pInData, int nInDataLen, BYTE * pOutData, int * pOutDataLen);*

说明:

函数的使用权限取决于锁内 RSA 私钥文件的权限, 在 RSA 私钥文件创建时设定。

参数:

hDongle	[in]	打开的加密锁句柄。
wPriFileID	[in]	RSA 私钥文件 ID。
nFlag	[in]	运算类型。例如, FLAG_ENCODE 表示加密运算; FLAG_DECODE 表示解密运算。
pInData	[in]	输入数据。
nInDataLen	[in]	参数 pInData 的大小
pOutData	[out]	输出数据缓冲区。
pOutDataLen	[in,out]	参数 pOutData 的大小和返回的数据大小。

返回值:

DONGLE_SUCCESS RSA 私钥运算成功。

注意:

- 1.对于加密运算,输入数据长度必须小于私钥 ID 为 wPriFileID 的密钥长度减去 11,以便在函数内部进行 padding
- 2.对于解密运算,输入数据长度必须与 wPriFileID 中指示的密钥长度相一致(比如 1024 位密钥时为 128, 2048 时为 256)
- 3.加密时内部 padding 方式为:PKCS1_TYPE_1 (即第二个字节为 0x01,空数据填充 0xFF)

加解密标志定义如下:

```
#define FLAG_ENCODE                      0 //加密
#define FLAG_DECODE                      1 //解密
```

30.Dongle_RsaPub RSA 公钥运算

DWORD WINAPI Dongle_RsaPub(DONGLE_HANDLE hDongle, int nFlag, RSA_PUBLIC_KEY * pPubKey, BYTE * pInData, int nInDataLen, BYTE * pOutData, int * pOutDataLen);

说明:

RSA 公钥运算，匿名权限可调用。

参数:

hDongle	[in]	打开的加密锁句柄。
nFlag	[in]	运算类型。例如，FLAG_ENCODE 表示加密运算；FLAG_DECODE 表示解密运算。
pPubKey	[in]	RSA 公钥数据。该数据来源于生成 RSA 公钥时的公钥数据。
pInData	[in]	输入数据。
nInDataLen	[in]	参数 pInData 的大小。
pOutData	[out]	输出数据缓冲区。
pOutDataLen	[in,out]	参数 pOutData 的大小和返回的数据大小。

返回值:

DONGLE_SUCCESS RSA 公钥运算成功。

注意:

- 1.对于加密运算,输入数据长度必须小于 pPubKey 中指示的密钥长度-11,以便在函数内部进行 padding ;
- 2.对于解密运算,输入数据长度必须与 pPubKey 中指示的密钥长度相一致(比如 1024 位密钥时为 128, 2048 时为 256) ;
- 3.加密时内部 padding 方式为:PKCS1_TYPE_2 (即第二个字节为 0x02,空数据填充随机数) 。

31.Dongle_EccGenPubPriKey 产生 ECC 公私钥

DWORD WINAPI Dongle_EccGenPubPriKey(DONGLE_HANDLE hDongle, WORD wPriFileID, ECCSM2_PUBLIC_KEY * pPubBakup, ECCSM2_PRIVATE_KEY * pPriBakup);

说明:

产生 ECC 公钥和私钥。使用该函数之前需要先创建一个 ECC 私钥文件。需要开发商权限。成功后注意保存 ECC 公私钥数据。

外部 ECCSM2 公钥格式 ECC(支持 bits 为 192 或 256)和 SM2 的(bits 为固定值 0x8100)公钥格式:

```
typedef struct{
    unsigned int bits;                // length in bits of modulus
    unsigned int XCoordinate[8];      // 曲线上点的 X 坐标
    unsigned int YCoordinate[8];      // 曲线上点的 Y 坐标
} ECCSM2_PUBLIC_KEY;
```

外部 ECCSM2 私钥格式 ECC(支持 bits 为 192 或 256)和 SM2 的(bits 为固定值 0x8100)私钥格式:

```
typedef struct{
    unsigned int bits;                // length in bits of modulus
    unsigned int PrivateKey[8];       // 私钥
} ECCSM2_PRIVATE_KEY;
```

参数:

hDongle	[in]	打开的加密锁句柄。
wPriFileID	[in]	ECC 私钥文件 ID。
pPubBackup	[out]	ECC 公钥数据。
pPriBackup	[out]	ECC 私钥数据。

返回值:

DONGLE_SUCCESS	产生 ECC 公私钥成功。
----------------	---------------

32.Dongle_EccSign ECC 私钥签名

DWORD WINAPI Dongle_EccSign(DONGLE_HANDLE hDongle, WORD wPriFileID, BYTE * pHashData, int nHashDataLen, BYTE * pOutData);

说明:

函数的使用权限取决于锁内 ECC 私钥文件的权限, 在 ECC 私钥文件创建时设定。

参数:

hDongle	[in]	打开的加密锁句柄。
wPriFileID	[in]	ECC 私钥文件 ID。
pHashData	[in]	Hash 数据。
nHashDataLen	[in]	参数 pHashData 的大小。
pOutData	[out]	签名数据。大小固定为 64 字节(256 位 ECC 时是正好,192 位 ECC 时的位会补 0)。

返回值:

DONGLE_SUCCESS 表示签名成功。

33.Dongle_EccVerify ECC 公钥验签

*DWORD WINAPI Dongle_EccVerify(DONGLE_HANDLE hDongle, ECCSM2_PUBLIC_KEY * pPubKey, BYTE * pHashData, int nHashDataLen, BYTE * pSign);*

说明:

函数的使用权限取决于锁内 ECC 私钥文件的权限, 在 ECC 私钥文件创建时设定。

参数:

hDongle	[in]	打开的加密锁句柄。
pPubKey	[in]	ECC 公钥数据。
pHashData	[in]	Hash 数据。
nHashDataLen	[in]	参数 pHashData 的大小。
pSign	[in]	签名数据。大小固定为 64 字节, 为 Dongle_EccSign 函数返回的 pOutData 数据。

返回值:

DONGLE_SUCCESS 表示验签成功, 否则表示验签失败。

注意:

1. 锁内签名算法为: ECDSA_Verify
2. 输入的 Hash 值的长度与 ECC 私钥的密钥长度有关, 如果密钥是 192 位的, 则 hash 值长度不能超过 24(192/8 = 24) 字节, 如果密钥是 256 位的, 则 hash 值长度不能超过 32(256/8 = 32) 字节。
3. 曲线参数为: EC_NIST_PRIME_192 及 EC_NIST_PRIME_256

34.Dongle_SM2GenPubPriKey 产生 SM2 公私钥

*DWORD WINAPI Dongle_SM2GenPubPriKey(DONGLE_HANDLE hDongle, WORD wPriFileID, ECCSM2_PUBLIC_KEY * pPubBakup, ECCSM2_PRIVATE_KEY * pPriBakup);*

说明:

使用该函数之前需要先创建一个 SM2 私钥文件, 需要开发商权限。成功后注意保存 ECC 公私钥数据。

参数:

hDongle	[in]	打开的加密锁句柄。
wPriFileID	[in]	SM2 私钥文件 ID。
pPubBackup	[out]	SM2 公钥数据。
pPriBackup	[out]	SM2 私钥数据。
返回值		
DONGLE_SUCCESS		产生 ECC 公私钥成功。

35.Dongle_SM2Sign SM2 私钥签名

*DWORD WINAPI Dongle_SM2Sign(DONGLE_HANDLE hDongle, WORD wPriFileID, BYTE * pHashData, int nHashDataLen, BYTE * pOutData);*

说明:

函数的使用权限取决于锁内 SM2 私钥文件的权限，在 SM2 私钥文件创建时设定。

参数:

hDongle	[in]	打开的加密锁句柄。
wPriFileID	[in]	SM2 私钥文件 ID。
pHashData	[in]	Hash 数据。
nHashDataLen	[in]	参数 pHashData 的大小。数据长度必须小于 32 个字节。
pOutData	[out]	签名数据。大小固定为 64 字节。

返回值:

DONGLE_SUCCESS	表示签名成功。
----------------	---------

36.Dongle_SM2Verify SM2 公钥验签

*DWORD WINAPI Dongle_SM2Verify(DONGLE_HANDLE hDongle, ECCSM2_PUBLIC_KEY * pPubKey, BYTE * pHashData, int nHashDataLen, BYTE * pSign);*

说明:

函数的使用权限取决于锁内 SM2 私钥文件的权限，在 SM2 私钥文件创建时设定。

参数:

hDongle	[in]	打开的加密锁句柄。
---------	------	-----------

pPubKey	[in]	SM2 公钥数据。
pHashData	[in]	Hash 数据。
nHashDataLen	[in]	参数 pHashData 的大小。
pSign	[in]	签名数据。大小固定为 64 字节, 为 Dongle_EccSign 函数返回的 pOutData 数据。
返回值:		
DONGLE_SUCCESS		表示验签成功, 否则表示验签失败。

37.Dongle_TDES 3DES 加解密

*DWORD WINAPI Dongle_TDES(DONGLE_HANDLE hDongle, WORD wKeyFileID, int nFlag, BYTE * pInData, BYTE * pOutData, int nDataLen);*

说明:

解密运算匿名权限即可, 加密运算的权限取决于密钥文件的权限。

参数:

hDongle	[in]	打开的加密锁句柄。
wKeyFileID	[in]	密钥文件 ID。
nFlag	[in]	运算类型。例如, FLAG_ENCODE 表示加密运算; FLAG_DECODE 表示解密运算。
pInData	[in]	输入数据缓冲区。
pOutData	[out]	输出数据缓冲区。大小至少要和输入数据缓冲区相同, 输入和输出数据缓冲区可以为同一个。
nDataLen	[in]	参数 pInData 的数据大小。数据长度必须是 16 的整数倍, 允许的最大值是 1024。

38.Dongle_SM4 SM4 加解密

*DWORD WINAPI Dongle_SM4(DONGLE_HANDLE hDongle, WORD wKeyFileID, int nFlag, BYTE * pInData, BYTE * pOutData, int nDataLen);*

说明:

解密运算匿名权限即可, 加密运算的权限取决于密钥文件的权限。

参数:

hDongle	[in]	打开的加密锁句柄。
wKeyFileID	[in]	密钥文件 ID。

nFlag	[in]	运算类型。例如, FLAG_ENCODE 表示加密运算; FLAG_DECODE 表示解密运算。
pInData	[in]	输入数据缓冲区。
pOutData	[out]	输出数据缓冲区。大小至少要和输入数据缓冲区相同, 输入和输出数据缓冲区可以为同一个。
nDataLen	[in]	参数 pInData 的数据大小。数据长度必须是 16 的整数倍, 允许的最大值是 1024。

返回值:

DONGLE_SUCCESS SM4 加密或解密运算成功。

39.Dongle_HASH HASH 运算

DWORD WINAPI Dongle_HASH(DONGLE_HANDLE hDongle, int nFlag, BYTE * pInData, int nDataLen, BYTE * pHash);

说明:

HASH 运算。

参数:

hDongle	[in]	打开的加密锁句柄。
nFlag	[in]	Hash 运算算法类型。
pInData	[in]	输入数据缓冲区。
nDataLen	[in]	参数 pInData 的数据大小。
pHash	[out]	输出的 Hash 值。

返回值:

DONGLE_SUCCESS HASH 运算成功。

注意:

nFlag = FLAG_HASH_MD5, 表示 MD5 运算, 此时 pHash 的缓冲区大小为 16 字节。

nFlag = FLAG_HASH_SHA1, 表示 SHA1 运算, 此时 pHash 的缓冲区大小为 20 字节。

nFlag = FLAG_HASH_SM3, 表示国密 SM3 运算, 此时 pHash 的缓冲区大小为 32 字节。

40.Dongle_Seed 种子码算法

DWORD WINAPI Dongle_Seed(DONGLE_HANDLE hDongle, BYTE * pSeed, int nSeedLen, BYTE * pOutData);

说明：

匿名权限可使用，开发商可设置可运算次数。

参数：

hDongle	[in]	打开的加密锁句柄。
pSeed	[in]	输入的种子码数据。
nSeedLen	[in]	种子码长度。取值范围为 1~250 字节。
pOutData	[out]	输出数据缓冲区。输出的大小固定为 16 字节。

返回值：

DONGLE_SUCCESS 种子码运算成功。

注意：

- 1.种子码算法与 PID 有关，空锁(即 PID=FFFFFFFF)不能进行种子码运算。
- 2.如果内部种子码可运算次数不为-1，当其递减到 0 后此函数将不能使用。

41.Dongle_LimitSeedCount 设置种子码运算次数

DWORD WINAPI Dongle_LimitSeedCount(DONGLE_HANDLE hDongle, int nCount);

说明：

设置种子码算法可运算次数。需要开发商权限。

参数：

hDongle	[in]	打开的加密锁句柄。
nCount	[in]	可运算次数，如果此值设置为-1，表示不限制运算次数。

返回值：

DONGLE_SUCCESS 设置成功。

42.Dongle_GenMotherKey 制作母锁

*DWORD WINAPI Dongle_GenMotherKey(DONGLE_HANDLE hDongle, MOTHER_DATA * pInData);*

说明：

制作一把母锁，子母锁的方式是一种可选的初始化锁方式，安全又快速，推荐使用。需要开发商权限。

参数:

hDongle [in] 打开的加密锁句柄。

pInData [in] 输入数据。用于初始化母锁的结构为 MOTHER_DATA 的数据。

返回值:

DONGLE_SUCCESS 制作母锁成功。

注意:

- 1.空锁(即 PID=FFFFFFFF)不能写入母锁数据。
- 2.出于安全考虑,MOTHER_DATA 中的远程升级私钥不允许和母锁自身的远程升级私钥相同,否则会操作失败。

需要发给空锁的初始化数据

```
typedef struct
{
    int m_SeedLen; //种子码长度
    BYTE m_SeedForPID[256]; //产生产品 ID 和开发商密码的种子码 (最长 250 个字节)
    char m_UserPIN[18]; //用户密码(16 个字符的 0 终止字符串)
    BYTE m_UserTryCount; //用户密码允许的最大错误重试次数
    BYTE m_AdminTryCount; //开发商密码允许的最大错误重试次数
    RSA_PRIVATE_KEY m_UpdatePriKey; //远程升级私钥
    DWORD m_UserID_Start; //起始用户 ID
} SON_DATA;
```

母锁数据

```
typedef struct
{
    SON_DATA m_Son; //子锁初始化数据
    long m_Count; //可产生子锁初始化数据的次数 (-1 表示不限制次数, 递减到 0 时会受限)
} MOTHER_DATA;
```

43.Dongle_RequestInit 空锁获取生产请求

*DWORD WINAPI Dongle_RequestInit(DONGLE_HANDLE hDongle, BYTE * pRequest);*

说明:

此函数只对 PID 为 FFFFFFFF 的空锁有效。需要开发商权限。

参数:

hDongle	[in]	打开的加密锁句柄。
pRequest	[out]	输出数据。返回该数据的有效长度为 16 字节,因此需要至少 16 字节的空间。

返回值:

DONGLE_SUCCESS	获取生产请求数据成功。
----------------	-------------

44.Dongle_GetInitDataFromMother 从母锁获取生产数据

DWORD WINAPI Dongle_GetInitDataFromMother(DONGLE_HANDLE hDongle, BYTE * pRequest, BYTE * pInitData, int * pDataLen);

说明:

从母锁获取用于初始化子锁的数据,该函数只对母锁有效。匿名权限可调用。

参数:

hDongle	[in]	打开的加密锁句柄。
pRequest	[in]	请求数据。通过 Dongle_RequestInit 获取的请求数据。
pInitData	[out]	输出数据。函数执行成功返回用于初始化子锁的数据。
pDataLen	[int,out]	参数 pInitData 的有效长度。表示 pInitData 的缓冲区长度,函数执行成功返回 pInitData 的有效长度。

返回值:

DONGLE_SUCCESS	从母锁获取生产数据成功。
----------------	--------------

45.Dongle_InitSon 生产子锁

DWORD WINAPI Dongle_InitSon(DONGLE_HANDLE hDongle, BYTE * pInitData, int nDataLen);

说明:

用子母锁的方式制作子锁。匿名权限即可调用。

参数:

hDongle	[in]	打开的加密锁句柄。
pInitData	[out]	输入数据。函数 Dongle_GetInitDataFromMother 返回的用于初始化子锁的数据。
nDataLen	[in]	参数 pInitData 数据缓冲区的有效长度。

返回值:

DONGLE_SUCCESS

生产子锁成功。

46.Dongle_SetUpdatePriKey 设置远程升级私钥

*DWORD WINAPI Dongle_SetUpdatePriKey(DONGLE_HANDLE hDongle, RSA_PRIVATE_KEY * pPriKey);*

说明:

向锁内设置远程升级私钥。私钥长度为 1024 的 RSA 私钥。需要开发商权限。出于安全考虑, 如果锁是母锁的话, 远程升级私钥不允许和母锁数据区中的子锁远程升级私钥相同, 否则会操作失败。

参数:

hDongle [in] 打开的加密锁句柄。

pPriKey [in] RSA 私钥。

返回值:

DONGLE_SUCCESS

设置远程升级私钥成功。

47.Dongle_MakeUpdatePacket 制作远程升级包

*DWORD WINAPI Dongle_MakeUpdatePacket(DONGLE_HANDLE hDongle, char * pHID, int nFunc, int nFileType, WORD wFileID, int nOffset, BYTE * pBuffer, int nBufferLen, RSA_PUBLIC_KEY * pUPubKey, BYTE * pOutData, int * pOutDataLen);*

说明:

制作远程升级数据包, 匿名权限即可调用。

参数:

hDongle [in] 打开的加密锁句柄。

pHID [in] 硬件序列号。如果不需要绑定该参数可以为 NULL。

nFunc [in] 升级包类型。

nFileType [in] 文件类型。升级有关文件操作时的文件类型。其他升级类型该参数无效。

wFileID [in] 文件 ID。升级有关文件操作时的文件 ID。其他升级类型该参数无效。

nOffset [in] 偏移量。升级有关文件操作时的文件偏移量。其他升级类型该参数无效。

pBuffer [in] 输入数据。

nBufferLen [in] 参数 pBuffer 的缓冲区大小。

pUPubKey	[in]	制作升级包的 RSA 公钥。与设置到锁内的远程升级私钥相对应。该值无论何种升级类型都必须填写。
pOutData	[out]	输出数据。制作的升级包数据。
pOutDataLen	[in,out]	参数 pOutData 输入大小，返回升级包的有效长度。
返回值：		
DONGLE_SUCCESS		制作升级包成功。

注意：

当 nFunc = UPDATE_FUNC_CreateFile，表示创建文件。

当 nFunc = UPDATE_FUNC_WriteFile，写文件。只有锁内已有的文件才可升级写文件操作。

当 nFunc = UPDATE_FUNC_DeleteFile，删除文件。

当 nFunc = UPDATE_FUNC_FileLic，设置文件授权。

当 nFunc = UPDATE_FUNC_SeedCount，设置种子码可运算次数。

当 nFunc = UPDATE_FUNC_DownloadExe，升级可执行文件。

当 nFunc = UPDATE_FUNC_UnlockUserPin，解锁用户 PIN。出于安全考虑解锁用户 PIN 码必须绑定 HID，即 pHID 不能为空，只有这样才能升级成功。升级成功后用户 PIN 码恢复为"12345678"，最大允许重试次数与未升级之前一致。

当 nFunc = UPDATE_FUNC_Deadline，时钟锁升级使用期限。

当 nFunc = UPDATE_FUNC_CreateFile 时, pBuffer 指向要文件的属性结构，例如 KEY_FILE_ATTR。

当 nFunc = UPDATE_FUNC_WriteFile 时, pBuffer 指向要写入的数据。

当 nFunc = UPDATE_FUNC_FileLic 时, pBuffer 指向文件权限的数据结构，例如：DATA_LIC。

当 nFunc = UPDATE_FUNC_SeedCount 时, pBuffer 指向 long 值，表示种子码可运算次数。

当 nFunc = UPDATE_FUNC_DownloadExe 时, pBuffer 指向 EXE_FILE_INFO 结构，与 Dongle_DownloadExeFile 函数用法类似。

当 nFunc = UPDATE_FUNC_Deadline 时, pBuffer 指向 DWORD 值，表示到期的时间。

48.Dongle_MakeUpdatePacketFromMother 母锁制作远程升级包

```
DWORD WINAPI Dongle_MakeUpdatePacketFromMother(DONGLE_HANDLE hDongle, char * pHID, int nFunc, int nFileType, WORD wFileID, int nOffset, BYTE * pBuffer, int nBufferLen, BYTE * pOutData, int * pOutDataLen);
```

说明：

制作远程升级数据包。该函数采用母锁方式制作，与 `Dongle_MakeUpdatePacket` 相比少了远程升级公钥，其他相同。匿名权限即可调用。

参数：

<code>hDongle</code>	[in]	打开的加密锁句柄。
<code>pHID</code>	[in]	硬件序列号。如果不需要绑定该参数可以为 <code>NULL</code> 。
<code>nFunc</code>	[in]	升级包类型。
<code>nFileType</code>	[in]	文件类型。升级有关文件操作时的文件类型。其他升级类型该参数无效。
<code>wFileID</code>	[in]	文件 ID。升级有关文件操作时的文件 ID。其他升级类型该参数无效。
<code>nOffset</code>	[in]	偏移量。升级有关文件操作时的文件偏移量。其他升级类型该参数无效。
<code>pBuffer</code>	[in]	输入数据。
<code>nBufferLen</code>	[in]	参数 <code>pBuffer</code> 的缓冲区大小。
<code>pOutData</code>	[out]	输出数据。制作的升级包数据。
<code>pOutDataLen</code>	[in,out]	参数 <code>pOutData</code> 输入大小，返回升级包的有效长度。

返回值：

`DONGLE_SUCCESS` 制作升级包成功。

注意：

当 `nFunc = UPDATE_FUNC_CreateFile`，表示创建文件。

当 `nFunc = UPDATE_FUNC_WriteFile`，写文件。只有锁内已有的文件才可升级写文件操作。

当 `nFunc = UPDATE_FUNC_DeleteFile`，删除文件。

当 `nFunc = UPDATE_FUNC_FileLic`，设置文件授权。

当 `nFunc = UPDATE_FUNC_SeedCount`，设置种子码可运算次数。

当 `nFunc = UPDATE_FUNC_DownloadExe`，升级可执行文件。

当 `nFunc = UPDATE_FUNC_UnlockUserPin`，解锁用户 PIN。出于安全考虑解锁用户 PIN 码必须绑定 HID，即 `pHID` 不能为空，只有这样才能升级成功。升级成功后用户 PIN 码恢复为"12345678"。

当 `nFunc = UPDATE_FUNC_Deadline`，时钟锁升级使用期限。

当 `nFunc = UPDATE_FUNC_CreateFile` 时，`pBuffer` 指向要文件的属性结构，例如 `KEY_FILE_ATTR`。

当 nFunc = UPDATE_FUNC_WriteFile 时, pBuffer 指向要写入的数据。

当 nFunc = UPDATE_FUNC_FileLic 时, pBuffer 指向文件权限的数据结构, 例如: DATA_LIC。

当 nFunc = UPDATE_FUNC_SeedCount 时, pBuffer 指向 long 值, 表示种子码可运算次数。

当 nFunc = UPDATE_FUNC_DownloadExe 时, pBuffer 指向 EXE_FILE_INFO 结构, 与 Dongle_DownloadExeFile 函数用法类似。

当 nFunc = UPDATE_FUNC_Deadline 时, pBuffer 指向 DWORD 值, 表示到期的时间。

49.Dongle_Update 远程升级子锁

DWORD WINAPI Dongle_Update(DONGLE_HANDLE hDongle, BYTE * pUpdateData, int nDataLen);

说明:

远程升级子锁中的数据。匿名权限即可。升级数据 pUpdateData 对一把锁只能使用一次。

参数:

hDongle	[in]	打开的加密锁句柄。
pUpdateData	[in]	输入数据。升级数据, 由 Dongle_MakeUpdatePacket 或者 Dongle_MakeUpdatePacketFromMother 产生。
nDataLen	[in]	参数 pUpdateData 的大小。必须为 1024 的整数倍。

返回值:

DONGLE_SUCCESS	升级成功。
----------------	-------

注意:

- 1.本函数内部是按 1024 字节的分块机制发送,如遇返回值不是 DONGLE_SUCCESS 会立即中断发送并返回。
- 2.如果需要进行流程控制,可由调用方来分块(每块 1024 字节)发送,并保证数据块的顺序不被打乱,根据返回的错误码来控制流程。

用户 API 错误码

#define	DONGLE_SUCCESS	0x00000000	// 操作成功
#define	DONGLE_NOT_FOUND	0xF0000001	// 未找到指定的设备
#define	DONGLE_INVALID_HANDLE	0xF0000002	// 无效的句柄
#define	DONGLE_INVALID_PARAMETER	0xF0000003	// 参数错误

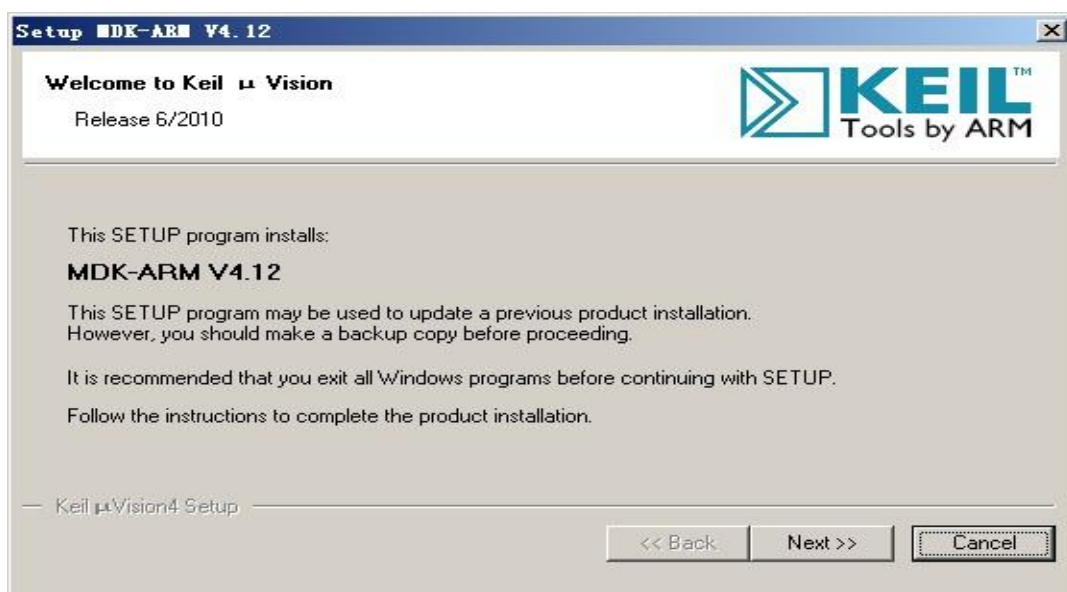
#define	DONGLE_COMM_ERROR	0xF0000004	// 通讯错误
#define	DONGLE_INSUFFICIENT_BUFFER	0xF0000005	// 缓冲区空间不足
#define	DONGLE_NOT_INITIALIZED	0xF0000006	// 产品尚未初始化 (即没设置 PID)
#define	DONGLE_ALREADY_INITIALIZED	0xF0000007	// 产品已经初始化 (即已设置 PID)
#define	DONGLE_ADMINPIN_NOT_CHECK	0xF0000008	// 开发商密码没有验证
#define	DONGLE_USERPIN_NOT_CHECK	0xF0000009	// 用户密码没有验证
#define	DONGLE_INCORRECT_PIN	0xF000FF00	// 密码错 (后 2 位指示剩余次数)
#define	DONGLE_PIN_BLOCKED	0xF000000A	// PIN 码已锁死
#define	DONGLE_ACCESS_DENIED	0xF000000B	// 访问被拒绝
#define	DONGLE_FILE_EXIST	0xF000000E	// 文件已存在
#define	DONGLE_FILE_NOT_FOUND	0xF000000F	// 未找到指定的文件
#define	DONGLE_READ_ERROR	0xF0000010	// 读取数据错误
#define	DONGLE_WRITE_ERROR	0xF0000011	// 写入数据错误
#define	DONGLE_FILE_CREATE_ERROR	0xF0000012	// 创建文件错误
#define	DONGLE_FILE_READ_ERROR	0xF0000013	// 读取文件错误
#define	DONGLE_FILE_WRITE_ERROR	0xF0000014	// 写入文件错误
#define	DONGLE_FILE_DEL_ERROR	0xF0000015	// 删除文件错误
#define	DONGLE_FAILED	0xF0000016	// 操作失败
#define	DONGLE_CLOCK_EXPIRE	0xF0000017	// 加密锁时钟到期
#define	DONGLE_ERROR_UNKNOWN	0xFFFFFFFF	// 未知的错误

ET-ARM 锁内可执行程序开发

编译环境安装

本款加密锁采用 ARM 内核的芯片，所以锁内可执行程序采用 keil4 进行开发。如果初次进行开发，需要先将 SDK 目录下的 keil4\MDKSetup.exe 运行并安装。安装步骤如下：

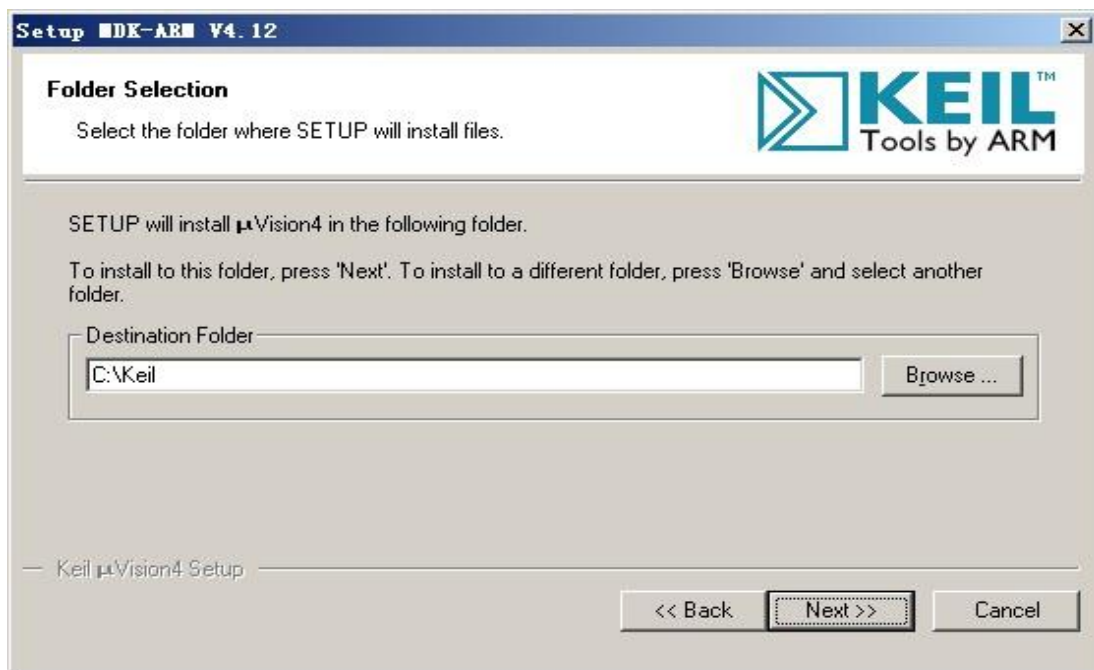
一、双击 MDKSetup.exe 进行安装



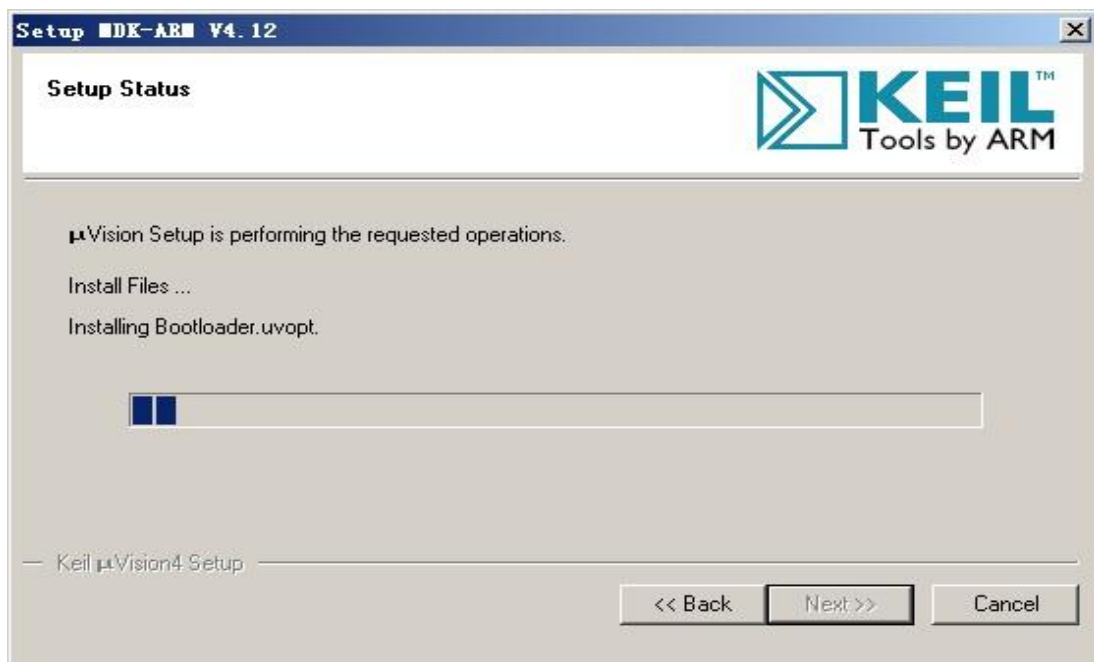
二、点击 Next



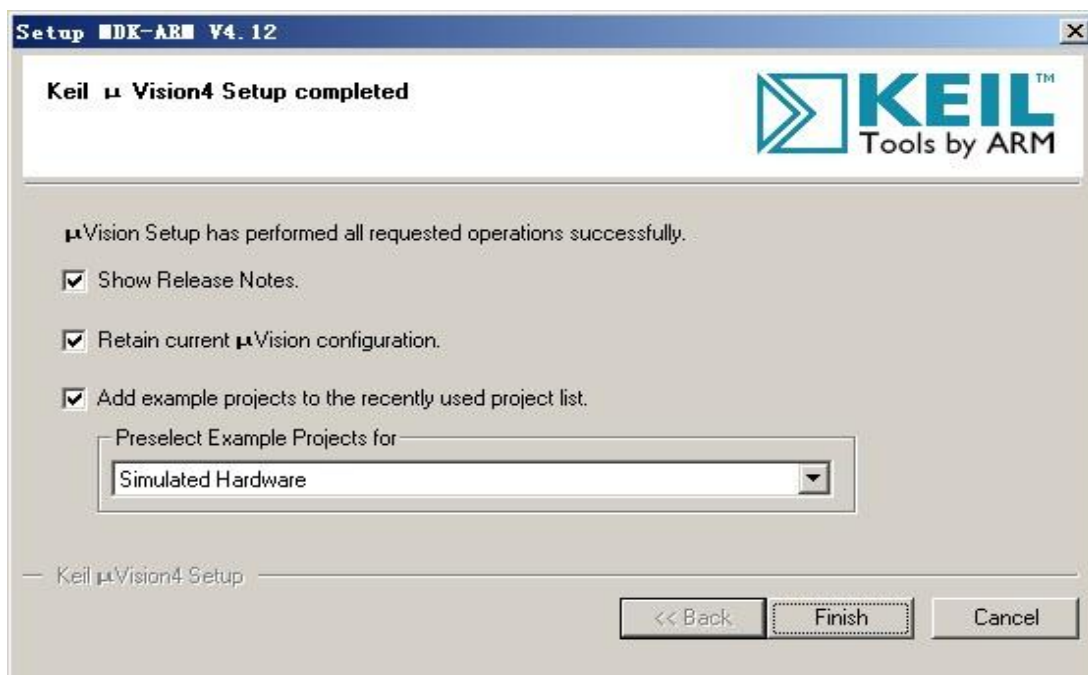
三、继续 Next



四、选择你准备安装的路径后继续 Next



五、安装完成后显示如下图所示



最后点击 Finish，完成了 Keil4 开发环境的安装。

编写锁内可执行程序

编译环境安装完成之后就可以进行开发工作了，SDK 目录中的 Tools\ARMProjWizard.exe 是 ARM 可执行程序的工程向导，通过这个工程向导可以生成一个配置好的 ET-ARM 锁内可执行文件的基本工程，用户只需要在里面添加代码即可。具体操作步骤如下：

一、双击 ARMProjWizard.exe



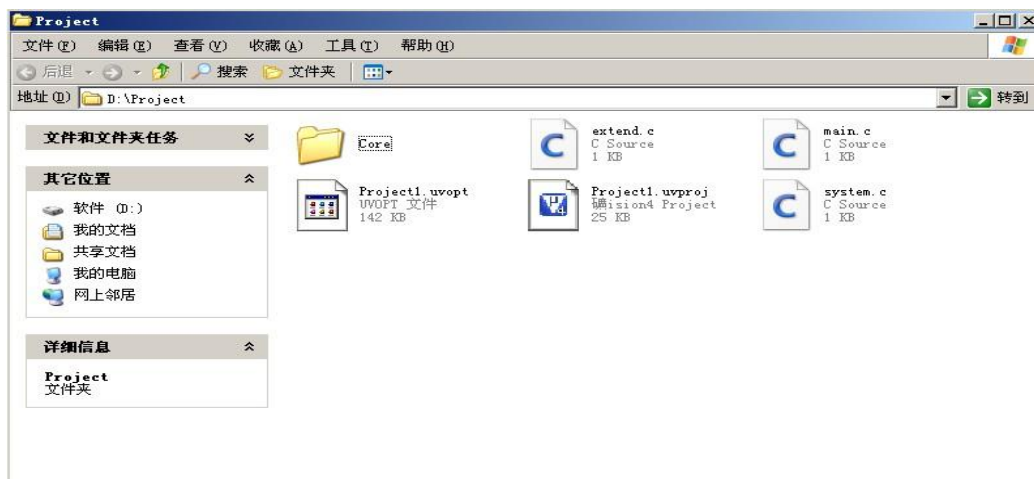
二、选择工程存放的路径



三、点击创建工程



四、选择“是”之后进入创建完成后的工程目录



创建完成之后，双击 Project1.uvproj 工程文件就可以用 Keil 环境进行 ARM 可执行文件的开发了。

编程空间说明

扩展缓冲区

```
extern unsigned char ExtendBuf[0x400];
```

ExtendBuf 为扩展缓冲区，总大小为 1024 字节。该缓冲区是锁内系统 API 使用的空间，用户也可以使用。但是，在每次调用锁内系统 API 之后，该区域的数据可能会被擦除。所以，在不使用锁内系统 API 的时候，用户可以使用该缓冲区。

输入输出缓冲区

```
extern unsigned char InOutBuf[0x400];
```

InOutBuf 为输入输出缓冲区,总大小为 1024 字节。上层用户 API 传到锁内的数据保存在这个缓冲区中,可执行程序运行完成之后,也通过 InOutBuf 缓冲区将数据返回给上层用户 API。例如下面这段代码

```
int main(void)
{
    WORD wError = 0;
    DWORD dwLen = 0;
    memcpy(&dwLen, InOutBuf, 4); //输入数据的前四个字节作为长度,获取随机数
    wError = get_random(InOutBuf, dwLen); //获取长度为 dwLen 的随机数
    return wError;
}
```

上面程序实现的功能为,外部将一个随机数的长度传入锁内,锁内程序依据这个长度获取随机数,最后将随机数返回到锁外。整个过程都是通过对 InOutBuf 的拷贝、填充来实现数据的交换。

用户 API 中 **Dongle_RunExeFile** 的参数 *plnOutBuf* 对应了此处的 InOutBuf,参数 *wlnOutBufLen* 决定了使用 InOutBuf 多大的长度,因此 *wlnOutBufLen* 的值不能超过 InOutBuf 的最大值 0x400 (1024);参数 *pMainRet* 对应了上例中的 *wError* 的值。

用户编程空间

除了以上的扩展缓冲区和输入输出缓冲区外,还有一个 2K 的用户编程空间,供用户定义变量。也就是说,用户最多可以自定义使用的缓冲区大小为 2048 字节。

用户定义的变量分为局部变量和全局变量。ET-ARM 中这 2048 字节的分配方案为,局部变量 1024 字节,全局变量 1024 字节。但是在用户的实际使用中,有可能需要在这两部分的空间做出调整,这里将提供一种调整局部变量和全局变量的大小的方法。在工程目录的 Core 文件夹下有一个名称为 startup.s 的文件,如图



所示

, 打开该文件后的第一行内容为:

Stack_Size	EQU	0x00000400	;1024 Bytes
------------	-----	------------	-------------

这部分是定义了局部变量的大小 1024 字节,相应的全局变量的大小为 2048 减去该值。用户可以修改该值来调整局部变量空间的大小,相应的全局变量空间大小也会自动的改变,修改完后保存 startup.s 文件重新编译 ARM 可执行程序即可。需要注意的是,一般情况下不要随意修改该文件的数据,如果必须要修改的话,修改范围也只能是 0~2048 之间。

另外,需要对全局变量做以下说明: 全局变量不支持定义时初始化,只能在主函数中进行初始化。

知道了以上内存空间的说明，就可以使用下面的系统函数进行编程开发锁内可执行程序了。

锁内系统 API 接口

1.create_file 创建文件

WORD create_file(**WORD** type, **WORD** fileid, **BYTE*** pattr, **int** len_attr);

说明：

本函数不支持创建可执行文件。

参数：

type [in] 文件类型，type = FILE_DATA，表示创建数据文件；

type = FILE_PRIKEY_RSA，表示创建 RSA 私钥文件；

type = FILE_PRIKEY_ECCSM2，表示创建 ECCSM2 私钥文件；

type = FILE_KEY，表示创建 SM4 和 3DES 密钥文件；

不支持 type = FILE_EXE 的文件类型。

fileid [in] 文件 ID。

pattr [in] 文件的属性。参数的结构为：DATA_FILE_ATTR、PRIKEY_FILE_ATTR 或 KEY_FILE_ATTR。

len_attr [in] 参数 pattr 的长度

返回值：

返回 0x9000，表示创建成功。

文件类型定义如下：

```
#define FILE_DATA      1 //普通数据文件
#define FILE_PRIKEY_RSA 2 //RSA 私钥文件
#define FILE_PRIKEY_ECCSM2 3 //ECC 或 SM2 私钥文件
#define FILE_KEY       4 //SMS4 或 TDES 密钥文件
#define FILE_EXE       5 //可执行文件
```

数据文件授权结构

```
typedef struct
{
    unsigned short m_Read_Priv; //读权限：0 为最小匿名权限 1 为最小用户权限 2 为最小
    管理员权限
```



```
    unsigned short  m_WritePriv;      //写权限: 0 为最小匿名权限  1 为最小用户权限  2 为最小
    管理员权限

} DATA_LIC;
```

数据文件属性数据结构

```
typedef struct
{
    unsigned long    m_Size;          //数据文件长度
    DATA_LIC        m_Lic;          //授权

} DATA_FILE_ATTR;
```

私钥文件授权结构

```
typedef struct
{
    long            m_Count;          //可调次数: 0xFFFFFFFF 表示不限制, 递减到 0 表示已不可调用
    unsigned char   m_Priv;          //调用权限: 0 为最小匿名权限, 1 为最小用户权限 , 2 为最小管
    理员权限
    unsigned char   m_IsDecOnRAM;    //是否是在内存中递减: 1 为在内存中递减, 0 为在 FLASH 中递减
    unsigned char   m_IsReset;      //用户态调用后是否自动回到匿名态: 1 为调后回到匿名态 (管理
    员态不受此限制)
    unsigned char   m_Reserve;      //保留,用于 4 字节对齐

} PRIKEY_LIC;
```

SM2、ECC 及 RSA 私钥文件属性数据结构

```
typedef struct
{
    unsigned short  m_Type;          //数据类型:ECC 私钥 或 RSA 私钥
    unsigned short  m_Size;          //数据长度:对 rsa 来说是 1024 位或 2048 位, 对 ecc 来说是 192
    位或 256 位, 对 SM2 来说是 256 位
    PRIKEY_LIC      m_Lic;          //授权

} PRIKEY_FILE_ATTR;
```

SM4 及 TDES 密钥文件授权结构

```
typedef struct
{
    unsigned long   m_Priv_Enc;      //加密调用权限: 0 为最小匿名权限, 1 为最小用户权限 , 2 为最
    小管理员权限
```



```
} KEY_LIC;
```

SM4 及 TDES 密钥文件属性数据结构

```
typedef struct
{
    unsigned long  m_Size;          //密钥数据长度, 正常是 16
    KEY_LIC        m_Lic;          //授权
} KEY_FILE_ATTR;
```

2.read_file 读文件

WORD read_file(WORD fileid, WORD offset, WORD len, BYTE* pbuf);

说明:

读取数据文件, 当 fileid=0xFFFF 时表示读取 8K 数据区, 不能读取可执行文件。

参数:

fileid	[in]	文件 ID。
offset	[in]	文件偏移量。
len	[in]	数据长度。
pbuf	[in]	数据缓冲区。

返回值:

返回 0x9000 表示读取数据存储区成功。

3.write_file 写文件

WORD write_file(WORD type, WORD fileid, WORD offset, WORD len, BYTE* pbuf);

说明:

写入数据文件, 当 fileid=0xFFFF 时表示写入 8K 数据区, 不能写入可执行文件。

参数:

type	[in]	文件类型。
fileid	[in]	文件 ID。
offset	[in]	文件偏移量。
len	[in]	文件长度。

pbuf [in] 数据缓冲区。

返回值:

返回 0x9000 表示写入数据存储区成功。

4.delete_file 删除文件

WORD delete_file(WORD type, WORD fileid);

说明:

删除文件。

参数:

type [in] 文件类型。

fileid [in] 文件 ID。

返回值:

返回 0x9000 表示删除文件成功。

5.get_keyinfo 获取加密锁信息

WORD get_keyinfo(KEY_INFO *pKI);

参数:

pKI [out] DONGLE_INFO 结构，加密锁信息。

锁的信息定义的结构体如下:

```
typedef struct
{
    unsigned short m_Ver;           //COS 版本,比如:0x0201,表示 2.01 版
    unsigned short m_Type;          // 0xFF 为标准锁, 0x00 为时钟锁, 0x02 为 U 盘锁
    unsigned char m_BirthDay[8];    //出厂日期
    unsigned long m_Agent;           //代理商编号,比如:默认的 0x00000000
    unsigned long m_PID;             //产品 ID
    unsigned long m_UserID;          //用户 ID
    unsigned char m_HID[8];          //8 字节的硬件 ID
    unsigned long m_IsMother;        //母锁标志: 0x01 表示是母锁, 0x00 表示不是母锁
} DONGLE_INFO;
```

6.get_pinstate 获取 PIN 码验证状态

WORD *get_pinstate(DWORD * pState);*

说明:

获取当前加密锁的 PIN 码验证状态，状态包括匿名态，用户态和开发商态。

参数:

pState [out] 状态值。

状态值的定义如下:

```
#define PIN_NONE              0x00              //匿名态
#define PIN_USER              0x01              //用户态
#define PIN_ADMIN              0x02              //开发商态
```

7.led_control LED 控制

WORD *led_control(BYTE flag);*

说明:

LED 灯的控制，可以选择 LED 的开、关和闪烁。

参数:

flag [in] 控制类型。

控制类型的定义如下:

```
#define LED_OFF              0x00              //LED 关
#define LED_ON              0x01              //LED 开
#define LED_BLINK              0x02              //LED 闪烁
```

返回值:

返回 0x9000，表示设置成功。

8.get_random 产生随机数

WORD *get_random(BYTE* pbuf, BYTE len);*

参数:

Pbuf [out] 随机数缓冲区。

len [in] 要产生的随机数的长度，nLen 的取值范围为 1~128。

返回值:

返回 0x9000, 表示产生随机数成功。

9.get_tickcount 取锁上电时间

WORD *get_tickcount(DWORD * pCount);*

说明:

取锁的上电时间, 单位是 ms, 精度是 100ms。

参数:

pCount [out] 上电以来的毫秒数

返回值:

返回 0x9000, 表示获取上电时间成功。

10.get_realtime 取真实时间

WORD *get_realtime(DWORD * pTime);*

说明:

获取真实时钟时间。

参数:

pTime [out] 锁内 utc 时间。

返回值:

返回 0x9000 表示获取真实时钟时间成功。

11.get_expiretime 到期时间

WORD *get_expiretime(DWORD * pTime);*

说明:

获取时钟锁的到期时间。

参数:

pTime [out] 到期时间值。

返回值:

返回 0x9000，表示获取时钟到期时间成功。

12.get_sharememory 读共享内存

WORD *get_sharememory*(**BYTE** * *pbuf*);

说明:

读取共享内存中的数据，大小为 32 字节。

参数:

pbuf [out] 数据缓冲区。

返回值:

返回 0x9000，表示读取共享内存数据成功。

13.set_sharememory 写共享内存区

WORD *set_sharememory*(**BYTE** * *pbuf*);

说明:

写入数据到共享内存区，大小为 32 字节。

参数:

pbuf [in] 数据缓冲区。

返回值:

返回 0x9000，表示写入共享内存数据成功。

14.rsa_genkey 产生 RSA 公私钥

WORD *rsa_genkey*(**WORD** *fileid*, **RSA_PRIVATE_KEY** * *prpk*);

说明:

在产生 RSA 公私钥对前，需要先创建 RSA 私钥文件。

RSA 公私钥格式定义如下:

RSA 公钥格式(兼容 1024,2048)

```
typedef struct {
    unsigned int  bits;           // length in bits of modulus
    unsigned int  modulus;       // modulus
}
```

```
    unsigned char exponent[256];          // public exponent
} RSA_PUBLIC_KEY;
```

RSA 私钥格式(兼容 1024,2048)

```
typedef struct {
    unsigned int  bits;                  // length in bits of modulus
    unsigned int  modulus;               // modulus
    unsigned char publicExponent[256];   // public exponent
    unsigned char exponent[256];         // private exponent
} RSA_PRIVATE_KEY;
```

参数:

fileid	[in]	RSA 私钥文件 ID。
pRPK	[out]	RSA 公私钥数据。

返回值:

返回 0x9000, 表示产生 RSA 公私钥成功。

15.rsa_pri RSA 私钥加解密运算

WORD *rsa_pri*(**WORD** fileid, **BYTE*** pln, **WORD** len, **BYTE*** pOut, **WORD*** plen_Out, **WORD** mode);

参数:

Fileid	[in]	RSA 私钥文件 ID。
pln	[in]	输入数据缓冲区。
len	[in]	输入数据大小。
pOut	[out]	输出数据缓冲区。
plen_Out	[in,out]	输出数据大小。
mode	[in]	运算类型。

返回值:

返回 0x9000, 表示 RSA 私钥运算成功。

加解密标志定义如下:

```
#define  MODE_ENCODE      0 //加密
#define  MODE_DECODE     1 //解密
```

16.rsa_pub RSA 公钥加解密运算

WORD *rsa_pub*(**BYTE*** *pln*, **WORD** *len*, **RSA_PUBLIC_KEY*** *pPub*, **BYTE*** *pOut*, **WORD*** *plen_Out*, **WORD** *mode*);

参数:

<i>pln</i>	[in]	输入数据缓冲区。
<i>len</i>	[in]	输入数据大小。
<i>pPub</i>	[in]	RSA 公钥数据。该数据来源于生成 RSA 公私钥时的公钥数据。
<i>pOut</i>	[out]	输出数据缓冲区。
<i>plen_Out</i>	[in,out]	参数 <i>pOut</i> 的大小和返回的数据大小。
<i>mode</i>	[in]	运算类型。

返回值

返回 0x9000 表示 RSA 公钥运算成功。

17.ecc_genkey 产生 ECC 公私钥

WORD *ecc_genkey*(**WORD** *fileid*, **ECCSM2_KEY_PAIR *** *pEKP*);

说明:

在产生 ECC 公私钥对之前, 需要产生 ECC 私钥文件。

ECC/SM2 公私钥格式定义如下:

ECC/SM2 公私钥对格式

```
typedef struct {
    ECCSM2_PRIVATE_KEY Prikey;
    ECCSM2_PUBLIC_KEY Pubkey;
} ECCSM2_KEY_PAIR;
```

外部 ECC/SM2 公钥格式

```
typedef struct {
    unsigned int bits;
    unsigned int XCoordinate[8];
    unsigned int YCoordinate[8];
}ECCSM2_PUBLIC_KEY;
```

外部 ECC/SM2 私钥格式

```
typedef struct {
    unsigned int bits;
    unsigned int PrivateKey[8];
} ECCSM2_PRIVATE_KEY;
```

参数:

fileid [in] ECC 私钥文件 ID。

pEKP [out] ECC 公私钥数据。

返回值:

返回 0x9000, 表示产生 ECC 公私钥成功。

18.ecc_sign ECC 私钥运算

WORD *ecc_sign(WORD fileid, BYTE* pIn, WORD len, BYTE* pOut, WORD* plen_Out);*

说明:

ECC 私钥仅进行签名操作, 即只提供加密运算。

参数:

fileid [in] ECC 私钥文件 ID。

pIn [in] 输入数据缓冲区。

len [in] 参数 pIn 的大小。

pOut [out] 签名数据。大小固定为 64 字节(256 位 ECC 时是正好,192 位 ECC 时的位会补 0)。

plen_Out [in] 参数 pOut 的大小。

返回值:

返回 0x9000, 表示签名成功。

19.ecc_verify ECC 公钥运算

WORD *ecc_verify(ECCSM2_PUBLIC_KEY* pEPK, BYTE* pHash, WORD len_Hash, BYTE* pSign);*

说明:

ECC 公钥仅进行验签操作, 即仅提供解密运算。

参数:

pEPK	[in]	ECC 公钥数据。
pHash	[in]	Hash 数据。
len_Hash	[in]	参数 pHash 的大小。
pSign	[in]	签名数据。大小固定为 64 字节，为 ecc_sign 函数返回的 pOut 数据。

返回值：

返回 0x9000，表示验签成功。

20.sm2_genkey 产生 SM2 公私钥

WORD sm2_genkey(**WORD** fileid, **ECCSM2_KEY_PAIR** * pEKP);

说明：

产生 SM2 公私钥之前，需要创建 SM2 私钥文件。

参数：

fileid	[in]	SM2 私钥文件 ID。
pEKP	[out]	SM2 公私钥数据。

返回值

返回 0x9000，表示产生 SM2 公私钥成功。

21.sm2_sign SM2 私钥运算

WORD sm2_sign(**WORD** fileid, **BYTE*** pIn, **WORD** len, **BYTE*** pOut, **WORD*** plen_Out);

说明：

SM2 私钥仅进行签名操作，即只提供加密运算。

参数：

fileid	[in]	SM2 私钥文件 ID。
pIn	[in]	输入数据缓冲区。
len	[in]	参数 pIn 的大小。数据长度必须小于 32 个字节。
pOut	[out]	签名数据。大小固定为 64 字节。
plen_Out	[in]	输出数据长度。

返回值：

返回 0x9000，表示签名成功，否则表示签名失败。

22.sm2_verify SM2 公钥运算

WORD *sm2_verify(ECCSM2_PUBLIC_KEY* pEPK, BYTE* pHash, WORD len_Hash, BYTE* pSign);*

说明：

SM2 私钥仅进行验签操作，即只提供解密运算。

参数：

pEPK [in] SM2 公钥数据。

pHash [in] Hash 数据。

len_Hash [in] 参数 pHash 的大小。

pSign [in] 签名数据。大小固定为 64 字节，为 Dongle_EccSign 函数返回的 pOutData 数据。

返回值：

返回 0x9000，表示验签成功，否则表示验签失败。

23.tdes TDES 加解密运算

WORD *tdes(BYTE* pdata, int len, int mode, WORD fileid);*

参数：

pdata [in] 数据缓冲区。

len [in] 参数 pdata 的数据大小。数据长度必须是 16 的整数倍，允许的最大值是 1024。

mode [in] 运算类型。FLAG_ENCODE 表示加密运算；FLAG_DECODE 表示解密运算。

fileid [in] 密钥文件 ID。

返回值：

返回 0x9000，表示 TDES 加解密运算成功。

24.sm4 SM4 加解密运算

WORD *sm4(BYTE* pdata, int len, int mode, WORD fileid);*

参数：

pdata [in] 数据缓冲区。

len [in] 参数 pdata 的数据大小。数据长度必须是 16 的整数倍,允许的最大值是 1024。

mode [in] 运算类型。FLAG_ENCODE 表示加密运算; FLAG_DECODE 表示解密运算。

fileid [in] 密钥文件 ID。

返回值:

返回 0x9000, 表示 SM4 加解密运算成功。

25.sha1 SHA1 运算

WORD sha1(BYTE* pdata ,int len, BYTE* phash);

参数:

pdata [in] 数据缓冲区。

len [in] 参数 pdata 的数据大小。

phash [in] 输出的 Hash 值, 固定为 20 字节大小。

返回值:

返回 0x9000, 表示 SHA1 运算成功。

26.sm3 SM3 运算

WORD sm3(BYTE* pdata ,int len, BYTE* phash);

参数:

pdata [in] 数据缓冲区。

len [in] 参数 pdata 的数据大小。

phash [in] 输出的 Hash 值, 固定为 32 字节大小。

返回值:

返回 0x9000, 表示 SM3 运算成功。

27.seed 种子码运算

WORD seed(BYTE* pseed ,int len, BYTE* presult);

参数:

pseed [in] 输入的种子码数据。

len [in] 种子码长度。取值范围为 1~250 字节。

presult [out] 输出数据缓冲区。输出的大小固定为 16 字节。

返回值:

返回 0x9000，表示种子码运算成功。

锁内系统 API 错误码

#define ERR_SUCCESS	0x9000	//操作成功
#define ERR_INVALID_INS	0x6C00	//无效的 INS
#define ERR_INVALID_P1	0x6C01	//无效的 P1
#define ERR_INVALID_P2	0x6C02	//无效的 P2
#define ERR_INVALID_LEN	0x6C03	//无效的 LEN
#define ERR_INVALID_PARAM	0x6C04	//无效的参数(如:数据中的一些参数)
#define ERR_FAILED	0x6C05	//操作失败
#define ERR_READ_ERR	0x6B00	//读数据错误
#define ERR_WRITE_ERR	0x6B01	//写数据错误
#define ERR_FILE_EXIST	0x6A82	//文件已存在
#define ERR_FILE_NOTFOUND	0x6A83	//文件不存在
#define ERR_FILE_CFILE_ERR	0x6A86	//创建文件失败
#define ERR_FILE_READ_ERR	0x6A87	//读文件失败
#define ERR_FILE_WRITE_ERR	0x6A88	//写文件失败
#define ERR_FILE_DFILE_ERR	0x6A89	//删文件失败
#define ERR_FILE_CDIR_ERR	0x6A8A	//创建文件夹失败
#define ERR_FILE_DDIR_ERR	0x6A8B	//删除文件夹失败
#define ERR_NOT_INITED	0x6980	//尚未初始化
#define ERR_ALREADY_INITED	0x6981	//已初始化过了
#define ERR_ADMINPIN_NOTCHECK	0x6982	//管理员 PIN 没有校验
#define ERR_USERPIN_NOTCHECK	0x6983	//用户 PIN 没有校验
#define ERR_PIN_BLOCKED	0x6984	//PIN 码已被锁定
#define ERR_RUN_LIMITED	0x6985	//运行已受限(如, 种子码运算等)