

Lockbox 例程使用说明

(当前版本 **1.6**)

目录

Lockbox 例程使用说明 1

 (当前版本 1.6) 1

 1. 介绍.....3

 2. 使用需求.....3

 3. Lockbox 演示程序.....4

 4. Lockbox Utility.....6

 5. OTP 烧写程序9

 6. 如何来运行演示程序?10

 7. 如何修改演示程序?11

1. 介绍

本例程向用户介绍了如何使用 Lockbox 技术，整个的例程有两个部分组成：

- PC 端的 Lockbox Utility 工具程序
- DSP 端的 Lockbox 演示程序
- OTP 烧写程序

Lockbox Utility 是一款工具软件，用于帮助那些想要采用 Blackfin Lockbox 技术的用户。该软件采用 MFC 编写，实现了 ECDSA 和 AES 算法，完善了整个方案。

Lockbox 演示程序向用户展示了如何在 Blackfin 上使用 Lockbox，以满足安全性的需要。用户可以根据自身的需求来修改该例程。

OTP 烧写程序提供了烧写 OTP memory 的方法供用户参考。

基于 Blackfin Lockbox 技术，用户可以设计很多安全性非常高的方案，我们在此实现了两个方案，供用户参考：

- **Single Entry:** 这个方案是为那些将应用全部保护起来的情况而设计，整个方案中我们只进入一次安全模式，然后应用将全部在安全模式下运行，直到运行完成，才退回到 Open 模式。这个方案中所有的可能在应用中用到的数据/代码都必须经过验证（通过加密或者 hash 交验）。
- **Multi Entry:** 这个方案只保护关键函数，整个方案我们的例程将多次出入安全模式。对于关键函数，将运行于安全模式，而对于其他函数，则运行于 Open 模式。这个方案更加灵活，用户改动也不大，适用性更加广泛。

用户可以依据自己需要来设计如何使用 Lockbox 技术来保护他们的应用。

2. 使用需求

下面为使用例程的软硬件需求：

- 软件需求：
 - a) Windows XP with SP2
 - b) VDSP++ 5.0 Update 4 或更高
- 硬件需求：
 - a) BF548 EZkit with silicon vision 0.1 or higher.
 - b) HPUSB-ICE

3. Lockbox 演示程序

本演示程序包含两个方案，分别放置在“Multi”和“Single”目录下。

- Multi

在这个方案中，应用程序多次进入安全模式，运行于安全模式的函数采用加密或者 hash 校验的方式来保证安全，他们被放置于 L2/L3 的内存中，运行在 L1 代码段中，为了节省 L1 空间，我们在 L1 代码段中设置了一段共享内存，当这些程序运行时，他们将被拷贝到这段共享内存中来运行。

这个方案存放在“Multi”目录下，文件介绍如下：

文件名	功能
adi_ssl_init.c	SSL 初始化代码
adi_ssl_init.h	SSL 初始化代码头文件
bfrom.h	L1 Rom 中代码的头文件
data.c	数据定义
decryption.c	AES 解密部分代码
init_ssl.c	KeyPad, SSL, LED 初始化代码
Lockbox_example.c	主函数
lockbox_example.h	整个例程共用的头文件
Secure_function.c	ECDSA 部分代码
Share_functions.c	共享部分函数代码
sesr.h	SESR 头文件

- Single

在这个方案中，应用程序只进入安全模式一次。整个应用完全运行于安全模式下，直到退出为止。在安全模式下，所有要访问的代码和数据都必须放置在 L1 或者 L2 内存中，并且经过加密或者 hash 校验，以保证其完整性。

这个方案存放在“Single”目录下，文件介绍如下：

文件名	功能
adi_ssl_init.c	SSL 初始化代码
adi_ssl_init.h	SSL 初始化代码头文件
bfrom.h	L1 Rom 中代码的头文件
data.c	数据定义
decryption.c	AES 解密部分代码
init_ssl.c	KeyPad, SSL, LED 初始化代码
Lockbox_example.c	主函数
lockbox_example.h	整个例程共用的头文件
Secure_function.c	ECDSA 部分代码
sesr.h	SESR 头文件
Application.c	运行于安全模式下的演示函数

注:

- ◆ 如果用户需要使用自己的密钥，那么请修改 `decrption.c` 文件中 line145-153。
- ◆ 目前的工程进行了 CHIP ID 的校验，如果用户选择校验，那么请保证您已经将 CHIP ID 烧写到 Private OTP 中，并保证烧写地址与 `CHIP_ID_AT_PRIVATE_OTP` 定义的地址一致；如果不选择校验，那么请注释掉 `check_chip_id` 函数。
- ◆ `bfrom.h` 头文件中定义了 L1 Rom 里的函数的实际地址，在不同的芯片版本中，各个函数的地址不完全相同
- ◆ 在当前的工程属性中，Silicon Revision 为 0.1，如果用户要将工程应用到其他版本的芯片，则请修改该属性到相应的芯片版本，不要使用 `any` 或者 `automatic`，因为这样可能导致 `__SILICON_REVISION__` 宏没有定义，而使得工程在连接的时候失败。
- ◆ 任何工程属性的修改，有可能导致 VDSP 重新生成 LDF 文件，请用户使用我们提供的 LDF 文件，或者在新生成的 LDF 文件中加入例程所需的段，请参考我们提供的 LDF 文件。
- ◆ 这两个例程经过 0.1，0.2 版 BF548 芯片测试通过。

4. Lockbox Utility

Lockbox Utility 运行在 windows XP 平台下，下图为其 GUI 界面：



在 Blackfin 的 L1 Rom 中，ADI 实现了部分 ECDSA 和 AES 算法，所以我们需要这个工具来完成整个流程。我们可以通过这个工具来修改我们编译好的 DXE 文件，以使其满足我们在安全方面的需要。

下面介绍下 GUI 的组成，包括五部分，如下：

- 输入参数：用户通过点击“...”按钮来选择想要修改的 DXE 文件。
- ECDSA 选项：用户通过下拉框，选择合适的方式来生成 ECDSA 的钥匙对，并生成对应的数字签名存贮到输入的 DXE 文件中。

我们一共有三种不同的方式供用户选择：

- ◆ Ezkit: 选择该项，本程序将生成与 Ezkit 相同的钥匙对，这样程序修改后的 DXE 文件将可以通过 Ezkit 上面的 ECDSA 认证。
- ◆ Generate: 选择该项，本程序将随机生成钥匙对，用户将生成的公钥烧写到芯片的 Public OTP memory 中，然后修改后的 DXE 文件将能通过烧写有该公钥的芯片的 ECDSA 认证，而对于未烧写该公钥或者烧写了其他公钥的芯片，将不能通过其 ECDSA 认证。
- ◆ Default: 选择该项，程序将不生成钥匙对，而是采用用户提供的钥匙对。用户将其已有的钥匙对放置在与 DXE 文件相同的目录下，名字分别为“public.ecs”和“private.ecs”。

选择完生成方式后，点击“Generate”按钮以获得钥匙对，并且程序将生成的数字签名写入到上面输入的 DXE 文件中。

- AES 选项: 在这个选项中，包含了很多 AES 加密解密参数: Key length, Mode, Key.
 - ◆ Key Length: 您可以选择的长度有 128/192/256bit.
 - ◆ Mode: ECB, CBC, OFB, CTR 模式. (其中 OFB 和 CTR 现在还不支持)
 - ◆ Key: 在此选项框中输入您的密码，采用 hex 模式，同时请保证密码长度与上面的 key length 中选择的长度一致。
举例来说：如果您想采用 128bit 的密码，并且所有的密码都设置成 1 的话，也就是 “1111 1111 1111 1111”，那么请按如下格式输入：
01010101010101010101010101010101.
(合法输入包括 0-9 和 a-f)
- Operation: 在这个选项框中您选择采用哪种方案和您的应用场景。
 - ◆ Purpose: 这个目的也就是上面说的应用场景，由以下两种组成：
 - Demonstration: 选择这个选项，程序将只显示少数几个段，而隐藏大量不用的段，这样用户看起来更加清晰，更容易理解整个处理过程。
在“Multi”方案中，我们将显示所有按照如下方式命名的段：ShareFunctionSection1, ShareFunctionSection2...，段的命名方式为 ShareFunctionSection + 数字。
在“Single”方案中，我们将显示如下的段：“L1_Code_Encrypt”, “L1_Code_Hash”, “L2_Encrypt” and “L2_Hash”。
注：
ShareFunctionSection1, ShareFunctionSection+数字:
这些段定义在 SDRAM 中，每个段中包含一个函数，这些函数将运行在安全模式下。
L1_Code_Encrypt: 这个段定义在 L1 代码段，该段中的函数将被加密，然后在运行时进入到安全模式下再解密。
L1_Code_Hash: 这个段定义在 L1 代码段，该段中的函数将被计算 hash 值，当我们进入安全模式后，在调用这些函数前，我们将先进行 hash 校验，验证其完整性后再调用。
L2_Encrypt: 这个段定义在 L2 中，该段中的函数将被加密，然后在运行时进入到安全模式下再解密。
L2_Hash: 这个段定义在 L2 代码段，该段中的函数将被计算 hash 值，当我们进入安全模式后，在调用这些函数前，我们将先进行 hash 校验，验证其完整性后再调用。
如果用户想要添加他们自己的函数，那么请将该函数放置在特定的段中。段的命名以及位置如上所述。
 - Practical: 选择这个选项，程序将显示所有的段，这样用户在点击“Load”按钮后，可以看到全部段的信息，然后用户可以依据需要来进行选择何种处理方式。
注：这两种选项的区别只在于显示部分，处理过程完全一样。
 - ◆ Entry: 这个选项决定了所采用的方案。
 - Multi: 多次进入方式

■ Single: 单次进入方式

- Information: 程序将在信息栏中显示您选择的 DXE 文件中所包含的段，然后您可以通过点击 checkbox 来决定如何处理这些段。您可以选择进行加密或者计算 hash 值，在您选择完成后，请点击“Save”按钮保存到一个新的 DXE 文件中。

下面是一个例子：

Hash	AES	Name	Addr	Size
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ShareFunctionSection1	feb0f630	00000060
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ShareFunctionSection2	01001cb0	00000060
<input checked="" type="checkbox"/>	<input type="checkbox"/>	ShareFunctionSection3	01001d10	00000060
<input type="checkbox"/>	<input type="checkbox"/>			
<input type="checkbox"/>	<input type="checkbox"/>			
<input type="checkbox"/>	<input type="checkbox"/>			
<input type="checkbox"/>	<input type="checkbox"/>			
<input type="checkbox"/>	<input type="checkbox"/>			
<input type="checkbox"/>	<input type="checkbox"/>			
<input type="checkbox"/>	<input type="checkbox"/>			

从上图我们可以看到，我们选择了“Multi”方案，并且采用与 Ezkit 相同的钥匙对，采用 128 位的 AES 加密方式，模式为 CBC 模式，密钥为：1111111111111111。

在这个例子中，我们对 ShareFunctionSection1 和 ShareFunctionSection2 进行加密和 hash 校验，而对 ShareFunctionSection3 只进行 hash 校验。

然后我们点击“Save”按钮，将其存到一个新的 DXE 文件中，之后可以在 VDSP 中加载该 DXE 文件，您可以看到如何运行于安全模式下。

5. OTP 烧写程序

本程序提供了一种方法，使用户方便的访问 OTP 内存。

我们在介绍演示程序的时候说到，演示程序可以使用用户自己的 AES 密钥，还可以使用 CHIP ID 进行校验，防止软件被大量复制。

如果用户想要将自己的 AES 密钥写入到 Private OTP 内存中，那么请修改数组 AES_CIPHER_KEY，他要写入的地址为 AES_CIPHER_KEY_AT_PRIVATE_OTP，请依照您想要放置密钥的实际地址修改该宏的定义。

如果用户想要进行 CHIP ID 校验，那么可以不需修改程序直接运行，如果用户想要修改 CHIP ID 放置的地址，请修改 CHIP_ID_AT_PRIVATE_OTP 的宏定义。

举例说明如下：

本程序将 AES key: 1C82EE0EC12220EC 770FCC952C2130BB 烧写到 Private OTP page 为 0x111 的内存里。

将 CHIP ID 从 Public OTP 读出，然后写入到 Private OTP page 为 0x110 的内存里。

程序的使用方法：

- i. 加载该工程到 VDSP。
- ii. 根据实际需要修改该工程。
- iii. 编译该工程
- iv. 查看 msg_code 段的大小（通过生成的 xml 文件）
- v. 修改 write_otp.c 文件中 14 行的宏 MESSAGE_SIZE 为上述大小
- vi. 重新编译
- vii. 使用 Lockbox Utility 对生成的 DXE 文件作数字签名（只点击 ECDSA 选项框中的 Generate 按钮，不要进行加密操作）
- viii. 加载经过签名的 DXE 文件到 VDSP
- ix. 运行

注：

- OTP memory 只能写一次，所以请保证您选择的 page 没有写过，否则将导致写入失败。
- ECDSA 数字签名是进入安全模式的唯一方式，所以如果用户想要进入安全模式或者访问 Private OTP，那么就一定要通过数字签名认证。

6. 如何来运行演示程序？

下面介绍下如何运行演示程序：

- i. 用户判断是否需要采用自己的密钥，或者进行 CHIP ID 校验，如果需要，那么进入 ii，否则进入 iii。
- ii. 加载 `write_otp` 工程到 VDSP 中，根据需要修改该工程，然后运行该工程。
- iii. 用户依据需要选择“Multi”或者“Single”工程加载到 VDSP 中
- iv. 用户如果不需要校验 CHIP ID，请注释掉 `check_chip_id` 函数，否则，请修改 `CHIP_ID_AT_PRIVATE_OTP` 宏地址为您实际放置 CHIP ID 的地址。
- v. 用户如果使用自己的密钥，请修改 `AES_CIPHER_KEY_AT_PRIVATE_OTP` 宏为您实际放置 AES KEY 的地址。否则，程序将使用 `1111111111111111` 作为默认的密钥。
- vi. 编译该工程
- vii. 查看 `msg_code` 段大小（在 `xml` 文件中）
- viii. 修改位于 `lockbox_example.h` 文件中的宏 `MESSAGE_SIZE` 为上述大小
- ix. 重新编译
- x. 打开 Lockbox Utility 工具
- xi. 加载在 ii 中生成的 DXE 文件
- xii. 选择合适的 ECDSA 钥匙对生成模式
- xiii. 点击“Generate”按钮生成钥匙对
- xiv. 依据用户需要输入 AES 参数
- xv. 在“Operation”选项框中选择合适的方案和用途
- xvi. 点击“Load”按钮
- xvii. 在信息栏，依据需要选择对显示出来的段何种处理方式
- xviii. 点击“Save”按钮，保存为一个新的 DXE 文件。
- xix. 在 VDSP 中加载新生成的 DXE 文件
- xx. 在 VDSP 中点击“Run”按钮
- xxi. 如果选择“Single”方案，您将能够看到 Led 闪烁。
- xxii. 如果选择“Multi”方案，您可以点击 Keypad 上按钮，然后将看到不同的 Led 闪烁，点击 PB4 则退出 demo。

注：

- 在 vi 步骤，我们生成数字签名时，修改了 DXE 文件，所以请不要重复使用相同的 DXE 文件，确保每次操作的 DXE 文件都是重新生成的。
- 请保证您选择的“Entry”与您正在使用的方案的工程一致，不如使用“Multi”方案，那么在“Entry”里必须选择“Multi”。
- 请务必保证每个要处理的段的长度为 16 的倍数，因为 AES 的处理模式是按照每个块 16 个字节来处理的，如果段的长度不是 16 的倍数，那么将导致解密出错。

7. 如何修改演示程序？

演示程序向用户介绍了如何在 Blackfin 上使用 Lockbox 技术。用户可以通过修改这个例程，来满足用户自己的需要。

我们将分别对两种不同的方案来做介绍。

- “Multi” 方案

在使用这个方案时，用户将要运行于安全模式的函数封装到每个独立的段中，段的命名按照 `ShareFunctionSection+数字`。

您可以将这些段放置在 L2/L3，然后这些程序将运行在 L1 代码段的 `ShareBufferForL1Code` 段中。

请查看 `ShareBufferForL1Code` 段的长度，务必保证其长度比 `ShareFunctionSection` 段中最大的段的长度长。我们在运行时将按如下模式操作这个段：

Open->Secure Mode: 我们将调用的函数从 `ShareFunctionSection` 段中拷贝到 `ShareBufferForL1Code` 段中，然后执行。

Secure->Open Mode: 我们将 `ShareBufferForL1Code` 清 0，以保证代码不被外界看到。

用于 ECDSA 认证的信息大小定义在 `lockbox_example.h` 中，请务必保证该大小与实际 DXE 文件中 `msg_code` 的段的大小一致。

所有的解密函数在 `decryption.c` 中，您可以在这里看到关于 AES 密钥的使用和 CHIP ID 的检查，您可以依据需要来修改这段代码。

演示程序的关键函数放置在 `share_functions.c` 中，函数名为 `funA,funB,funC`，您可以用您的关键函数替换这些函数，然后通过 Lockbox Utility 来编辑他们，以达到保证这些函数安全的目的。

- “Single” 方案

在这个方案中，应用的主要函数运行于安全模式下，所以您需要检查所有有可能在安全模式中被调用的函数，确保他们没有被修改，再调用之前检查他们的一致性。

您可以在 `application.c` 中看到细节描述，所有的在安全模式下运行的函数被放置在 L1 或者 L2 中，您可以在 Lockbox Utility 中选择对他们进行 hash 或者加密处理。

举例来说，您可一将您的关键函数放置在如 `L2_Encrypt` 段中，然后通过 Lockbox Utility 来对他们进行加密，最后加载到 Blackfin 上。在运行时，当系统进入到安全模式后，解密函数将对其进行解密，然后验证其 hash 值，通过验证后，运行该段代码，当退出时，我们将这些段清零，以保证不被外界看到。

注：

- 用于 ECDSA 认证的信息大小定义在 `lockbox_example.h` 中，请务必保证该大小与实际 DXE 文件中 `msg_code` 的段的大小一致。
- 所有的解密函数在 `decryption.c` 中，您可以在这里看到关于 AES 密钥的使用和 CHIP ID 的检查，您可以依据需要来修改这段代码。
- 演示程序的关键函数放置在 `share_functions.c` 中，函数名为 `funA,funB,funC`，您可以用您的关键函数替换这些函数，然后通过 Lockbox Utility 来编辑他们，以达到保证这些函数安全的目的。

- 当您改变工程选项时，VDSP 会重新生成 LDF 文件，请修改新的 LDF 文件以保证内存布局和原来的一致。
- 固定的段：msg_code, msg_data, digital_signature, AESData. 请务必保证您的 DXE 文件中包含这些段。
 - ◆ “msg_code”：这个段里放置的是 ECDSA 要验证的信息，它应该放置在 L1 代码段中。
 - ◆ “msg_data”：这个段是用来在运行时放置验证信息的，它应该被放置在 L1 数据段中。
 - ◆ “digital_signature”：这个段中放置的是数字签名，它应该放置在 msg_data 之前。
 - ◆ “AESData”：这个段中存放的是处理的信息，应用程序将在运行时读取这个段来对其他的段进行处理。