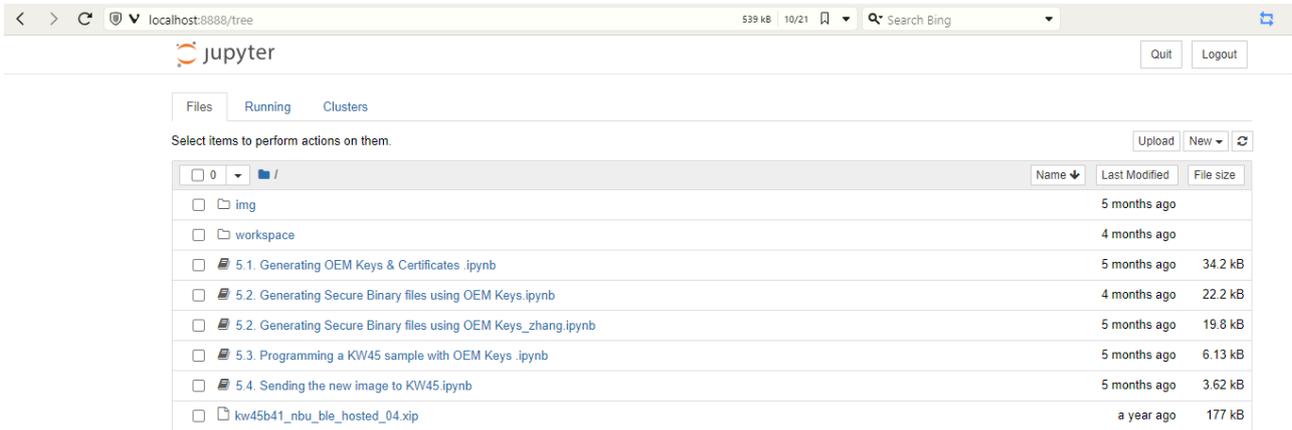


Updating KW45 NBU with SPSDK II –Jupyter Notebook SW Details

- 1 前言
- 2 激活虚拟环境
- 3 创建 OEM 密钥和证书
 - 3.1 准备Keys
 - 3.2 准备Certificates
 - 3.2.1 获取yml模板
 - 3.2.2 修改yml模板并生成证书
 - 3.3 生成SB3KDK
- 4 使用 OEM Key创建 SB3文件
- 5 使用OEM key 烧录KW45 芯片
 - 5.1 设备准备
 - 5.2 使用UART COM口连接设备
 - 5.3 烧写Fuse
- 6 用sb3.sb3更新NBU



3 创建 OEM 密钥和证书

3.1 准备Keys

首先，我们需要生成 RoTKs（信任密钥根）和可选的 ISK（image 签名证书）。为此，我们将使用 `npxcrypto` 应用程序。脚本默认生成 4 个 RoTKs 和 1 个 ISK 密钥（全套可能的密钥）。RoTK 0 生成是强制性的。

根据生成的密钥，会计算 `RoTKTH` 值并将其加载到 fuse 中，这就是设备密钥无法再更改的原因。

`secp384r1` 是一种安全的曲线，它有 384 位的素数域和 256 位的安全等级。脚本生成基于 `secp384r1` 曲线的椭圆曲线加密（ECC）私钥。ECC 私钥可以用来进行数字签名和密钥交换

使用 `npxcrypto`，它是 NXP 提供的一个用于加密协处理器的工具。`npxcrypto` 可以用来生成、管理和使用 ECC 私钥和公钥。

脚本生成了五对 ECC 私钥和公钥，分别命名为 ROTK0、ROTK1、ROTK2、ROTK3 和 ISK。每对私钥和公钥都保存在一个 PEM 格式的文件中，以 `.pem` 或 `.pub` 结尾。

这段脚本使用了以下的命令来生成 ECC 私钥：

```
1 #! npxcrypto $VERBOSITY key generate -k secp384r1  
$ROTK0_PRIVATE_KEY_PATH --force
```

这个命令的含义：

- `#!` 表示这是一个外部命令，执行 `npxcrypto` 程序。
- `key generate` 是一个子命令，表示要生成一个密钥。

- `-k secp384r1` 是一个选项，表示要使用的曲线的名称。
- `$PRIVATE_KEY_PATH` 是一个变量，表示要保存私钥的文件的名称。
- `-force` 选项表示如果文件已经存在，就覆盖它。

如果断言失败，表示生成私钥出错或者文件不存在，那么脚本会停止并报错。

最终成功生成五对ECC私钥和公钥：

The screenshot shows a file explorer window with the path `spsdk > AN13883SW > AN_SPSDK > workspace`. It displays a list of files generated on 2023/12/26 at 11:51. The files are organized into five pairs, each pair consisting of a private key (pem) and a public key (pub) for a specific certificate (cert0 through cert3, plus a sign_cert). The file types are either PEM File or Microsoft Publisher Certificate.

Name	Date modified	Type
ec_pk_secp384r1_cert0.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert0.pub	2023/12/26 11:51	Microsoft Publisher Certificate
ec_pk_secp384r1_cert1.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert1.pub	2023/12/26 11:51	Microsoft Publisher Certificate
ec_pk_secp384r1_cert2.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert2.pub	2023/12/26 11:51	Microsoft Publisher Certificate
ec_pk_secp384r1_cert3.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert3.pub	2023/12/26 11:51	Microsoft Publisher Certificate
ec_pk_secp384r1_sign_cert.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_sign_cert.pub	2023/12/26 11:51	Microsoft Publisher Certificate

3.2 准备Certificates

为上一步中生成的私钥生成包含公钥的自签名 x509 证书。仍然使用应用程序 `npxcrypto`。

3.2.1 获取yml模板

第一步是获取根证书的配置模板。

根证书是一种用来验证其他证书的有效性的证书，它是证书链的最顶层。根证书的配置模板是一个YAML或JSON格式的文件，它包含了生成根证书所需的信息，例如主题、颁发者、有效期、序列号、密钥类型等。

脚本获取了五个根证书的配置模板，分别命名为 `cert0_template.yml`、`cert1_template.yml`、`cert2_template.yml`、`cert3_template.yml` 和 `sign_cert_template.yml`。

核心命令：

```
1 | %! npxcrypto $VERBOSITY cert get-cfg-template $ROOT0_CERT_CONFIG_PATH
   | --force
```

- `$VERBOSITY` 是一个变量，表示输出的详细程度，可以是0、1或2。
- `cert get-cfg-template` 是一个子命令，表示要获取证书的配置模板。

生成的结果：

spsdk > AN13883SW > AN_SPSDK > workspace

Name	Date modified	Type
sign_cert_template.yml	2023/12/26 12:02	Yaml Source File
cert2_template.yml	2023/12/26 12:02	Yaml Source File
cert3_template.yml	2023/12/26 12:02	Yaml Source File
cert1_template.yml	2023/12/26 12:02	Yaml Source File
cert0_template.yml	2023/12/26 12:02	Yaml Source File
ec_pk_secp384r1_sign_cert.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_sign_cert.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert2.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert2.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert3.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert3.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert0.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert0.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert1.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert1.pub	2023/12/26 11:51	Microsoft Publist

3.2.2 修改yml模板并生成证书

对于获取到的根证书模板，需要自行添加此处的自定义配置，包括修改配置模板文件中的私钥和公钥的路径。

脚本生成了五个根证书，分别命名为`ec_secp384r1_cert0.pem`、`ec_secp384r1_cert1.pem`、`ec_secp384r1_cert2.pem`、`ec_secp384r1_cert3.pem`和`ec_secp384r1_sign_cert.pem`。

脚本首先导入了`yaml`模块，它是一个用来处理YAML格式的文件Python库。修改模板中的配置并重新写入文件：

```

1 # Create configuration for root certificate 0
2 with open(ROOT0_CERT_CONFIG_PATH) as cert_config:
3     # load yaml configuration to dictionary
4     cert = yaml.safe_load(cert_config)
5     # change path to private and public keys
6     cert['issuer_private_key'] = ROTK0_PRIVATE_KEY_PATH
7     cert['subject_public_key'] = ROTK0_PUBLIC_KEY_PATH
8
9 with open(ROOT0_CERT_CONFIG_PATH, "w+") as cert_config:
10    print("Root Certificate config:")
11    pp.pprint(cert)
12    # dump the dictionary back to YAML
13    yaml.dump(cert, cert_config)

```

最后，使用了 `npxcrypto` 来生成根证书：

```

1 # Generate root certificates 0
2 %! npxcrypto $VERBOSITY cert generate -c $ROOT0_CERT_CONFIG_PATH -o
   $ROOT_0_CERT_PATH --force

```

spsdk > AN13883SW > AN_SPSDK > workspace

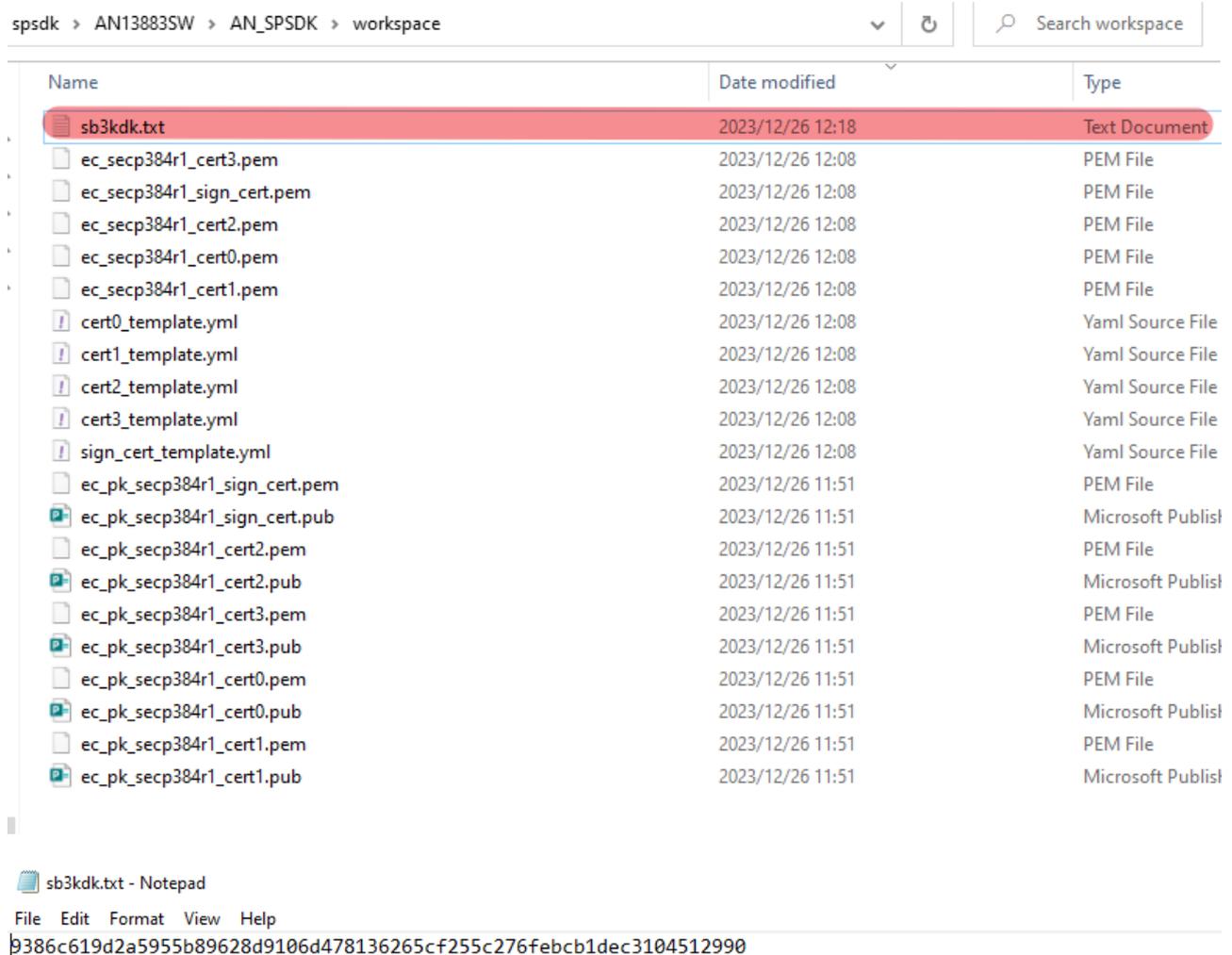
Search workspace

Name	Date modified	Type
ec_secp384r1_cert3.pem	2023/12/26 12:08	PEM File
ec_secp384r1_sign_cert.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert2.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert0.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert1.pem	2023/12/26 12:08	PEM File
cert0_template.yml	2023/12/26 12:08	Yaml Source File
cert1_template.yml	2023/12/26 12:08	Yaml Source File
cert2_template.yml	2023/12/26 12:08	Yaml Source File
cert3_template.yml	2023/12/26 12:08	Yaml Source File
sign_cert_template.yml	2023/12/26 12:08	Yaml Source File
ec_pk_secp384r1_sign_cert.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_sign_cert.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert2.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert2.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert3.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert3.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert0.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert0.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert1.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert1.pub	2023/12/26 11:51	Microsoft Publist

3.3 生成SB3KDK

使用脚本生成 SB3KDK。密钥生成一开始只能进行一次，然后 SB3KDK 密钥会加载到设备熔丝中，并且设备密钥无法再更改。

```
1 import os, binascii
2
3 SB3KDK_KEY_PATH = WORKSPACE + "sb3kdk.txt"
4 with open(SB3KDK_KEY_PATH, "wb") as f:
5     f.write(binascii.b2a_hex(os.urandom(32)))
```



spsdk > AN13883SW > AN_SPSDK > workspace

Name	Date modified	Type
sb3kdk.txt	2023/12/26 12:18	Text Document
ec_secp384r1_cert3.pem	2023/12/26 12:08	PEM File
ec_secp384r1_sign_cert.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert2.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert0.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert1.pem	2023/12/26 12:08	PEM File
cert0_template.yml	2023/12/26 12:08	Yaml Source File
cert1_template.yml	2023/12/26 12:08	Yaml Source File
cert2_template.yml	2023/12/26 12:08	Yaml Source File
cert3_template.yml	2023/12/26 12:08	Yaml Source File
sign_cert_template.yml	2023/12/26 12:08	Yaml Source File
ec_pk_secp384r1_sign_cert.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_sign_cert.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert2.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert2.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert3.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert3.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert0.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert0.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert1.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert1.pub	2023/12/26 11:51	Microsoft Publist

sb3kdk.txt - Notepad

File Edit Format View Help

9386c619d2a5955b89628d9106d478136265cf255c276febcb1dec3104512990

4 使用 OEM Key 创建 SB3 文件

为了生成 SB3.1 文件，使用 npximage 工具。npximage 工具根据配置文件生成 SB3.1 文件。让我们为 SB3.1 创建一个模板。根据您的需要修改示例。

1. 获取 SB3 配置模板：

```
%! npximage $VERBOSITY sb31 get-template -f kw45xx -o  
$SB31_TEMPLATE_PATH
```

spsdk > AN13883SW > AN_SPSDK > workspace

Name	Date modified	Type
sb31_config.yml	2023/12/26 12:20	Yaml Source File
sb3kdk.txt	2023/12/26 12:18	Text Document
ec_secp384r1_cert3.pem	2023/12/26 12:08	PEM File
ec_secp384r1_sign_cert.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert2.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert0.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert1.pem	2023/12/26 12:08	PEM File
cert0_template.yml	2023/12/26 12:08	Yaml Source File
cert1_template.yml	2023/12/26 12:08	Yaml Source File
cert2_template.yml	2023/12/26 12:08	Yaml Source File
cert3_template.yml	2023/12/26 12:08	Yaml Source File
sign_cert_template.yml	2023/12/26 12:08	Yaml Source File
ec_pk_secp384r1_sign_cert.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_sign_cert.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert2.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert2.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert3.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert3.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert0.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert0.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert1.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert1.pub	2023/12/26 11:51	Microsoft Publist

2. 修改yml模板:

```
1 # Create configuration for sb31  
2 with open(SB31_TEMPLATE_PATH) as sb31_config:  
3     # load yaml configuration to dictionary  
4     sb31 = yaml.safe_load(sb31_config)  
5     # change paths  
6     sb31['containerOutputFile'] = SB31_FILE_PATH  
7     sb31['containerKeyBlobEncryptionKey'] = SB3KDK_KEY_PATH  
8     sb31['description'] = "384_none_nbu_only"  
9     sb31['kdkAccessRights'] = 3  
10    sb31['containerConfigurationWord'] = 0  
11    sb31['rootCertificate0File'] = ROOT_0_CERT_PATH  
12    sb31['rootCertificate1File'] = ROOT_1_CERT_PATH  
13    sb31['rootCertificate2File'] = ROOT_2_CERT_PATH  
14    sb31['rootCertificate3File'] = ROOT_3_CERT_PATH  
15    sb31['mainRootCertId'] = 0  
16    sb31['mainRootCertPrivateKeyFile'] = ROTK0_PRIVATE_KEY_PATH  
17    sb31['rootCertificateEllipticCurve'] = "secp384r1"  
18    sb31['useIsk'] = False  
19  
20    del sb31['binaryCertificateBlock']
```

```

21     del sb31['signingCertificatePrivateKeyFile']
22     del sb31['signingCertificateFile']
23     del sb31['signCertData']
24     del sb31['commands']
25     del sb31['iskCertificateEllipticCurve']
26     sb31['commands'] = [
27         {
28             "erase": {
29                 "address": "0x48800000 ",
30                 "size": "0x30000"
31             }
32         },
33         {
34             "load": {
35                 "address": "0x48800000 ",
36                 "file": "kw45b41_nbu_ble_hosted_04.xip"
37             }
38         }
39     ]
40
41     with open(SB31_TEMPLATE_PATH, "w+") as sb31_config:
42         print("SB31:")
43         pp.pprint(sb31)
44         # dump the dictionary back to YAML
45         yaml.dump(sb31, sb31_config)

```

修改一些变量，以及写入sb31命令，包括KW45擦除的起始地址和要载入的xip文件。修改之后重新写入这么yaml文件中。

3. 由之前的Key和Certificates创建SB3文件：

- (a) 之前的yaml修改中，已经将key和certificate的路径写入了；
- (b) 这里，直接通过 `nxpimage` 这个程序就可以由密钥和证书创建 `sb3.sb3` 文件；
- (c)
 - 1 | `SB31_FILE_PATH = "sb3.sb3"`
 - 2 | `SB31_TEMPLATE_PATH = WORKSPACE + "sb31_config.yml"`
 - 3 |
 - 4 | `%! nxpimage $VERBOSITY sb31 export $SB31_TEMPLATE_PATH`

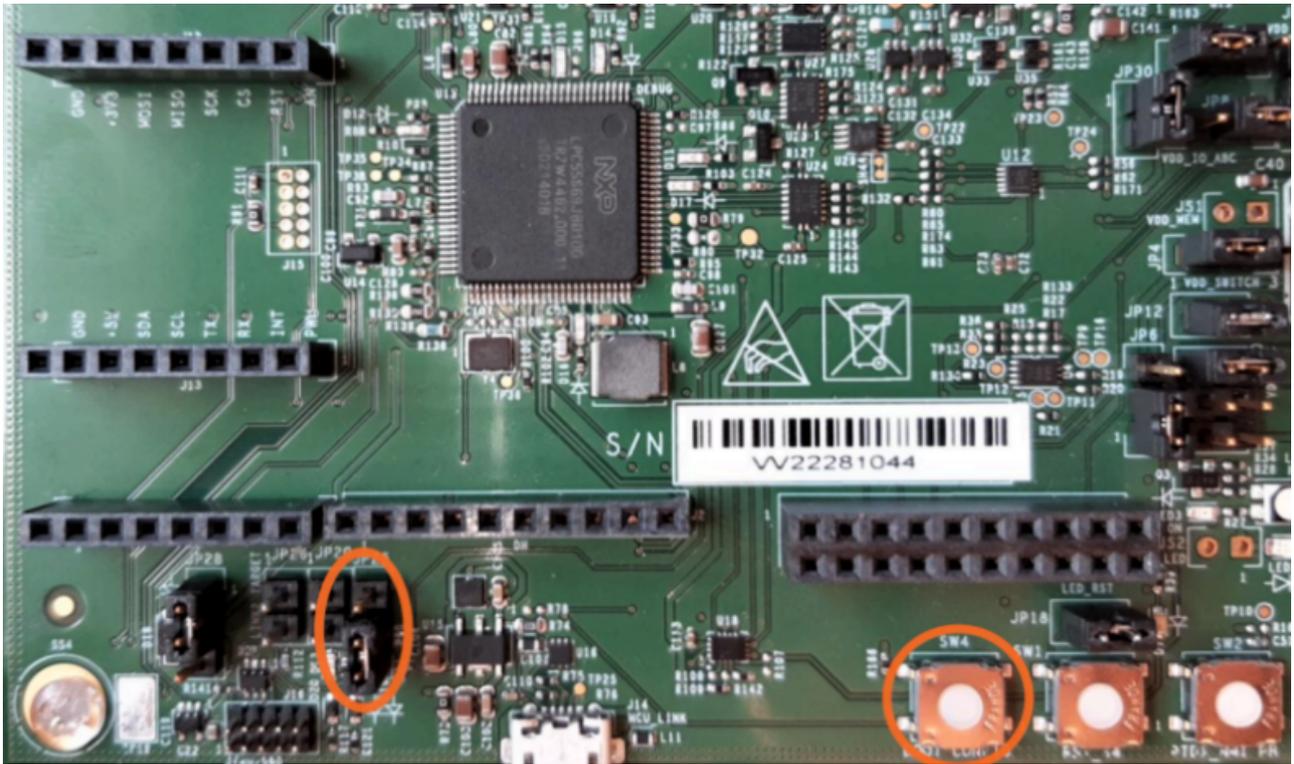
Name	Date modified	Type
sb3.sb3	2023/12/26 12:34	SB3 File
sb31_config.yml	2023/12/26 12:25	Yaml Source File
sb3kdk.txt	2023/12/26 12:18	Text Document
ec_secp384r1_cert3.pem	2023/12/26 12:08	PEM File
ec_secp384r1_sign_cert.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert2.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert0.pem	2023/12/26 12:08	PEM File
ec_secp384r1_cert1.pem	2023/12/26 12:08	PEM File
cert0_template.yml	2023/12/26 12:08	Yaml Source File
cert1_template.yml	2023/12/26 12:08	Yaml Source File
cert2_template.yml	2023/12/26 12:08	Yaml Source File
cert3_template.yml	2023/12/26 12:08	Yaml Source File
sign_cert_template.yml	2023/12/26 12:08	Yaml Source File
ec_pk_secp384r1_sign_cert.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_sign_cert.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert2.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert2.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert3.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert3.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert0.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert0.pub	2023/12/26 11:51	Microsoft Publist
ec_pk_secp384r1_cert1.pem	2023/12/26 11:51	PEM File
ec_pk_secp384r1_cert1.pub	2023/12/26 11:51	Microsoft Publist

5 使用OEM key 烧录KW45 芯片

5.1 设备准备

使用 KW45-EVK 板。第一步是进入 ISP 模式，方法如下

1. 将 JP25 置于 (2-3) 位置
2. 按下 SW4 复位电路板



然后，使用应用程序 `nxpdevscan` 检查设备是否以 ISP 模式连接到电脑。

```
# check if the device is connected and detected by PC
%! nxpdevscan

Created `%!` as an alias for `execute`.
nxpdevscan
----- Connected NXP SDIO Devices -----

----- Connected NXP USB Devices -----

----- Connected NXP UART Devices -----

----- Connected NXP SIO Devices -----
```

5.2 使用UART COM口连接设备

```
# choose com port
UART_CONNECTION = "-p com5"

%! blhost $UART_CONNECTION get-property current-version
assert _exit_code == 0

blhost -p com5 get-property current-version
Response status = 0 (0x0) Success.
Response word 1 = 1258488064 (0x4b030100)
Current Version = K3.1.0
```

----> 注意，这里要改为自己在使用的COM口编号。

5.3 烧写Fuse

使用前面步骤中生成的Key/RoTKTH 对设备保险丝进行编程。使用 blhost对fuse进行编程，设备需要处于 ISP 模式，可以与 blhost 通信并处理 blhost 命令。

Note: 这一步是不可逆的，请确保每一个细节都是正确的。

1. 增加fuse电压用于烧写：

```
%! blhost $UART_CONNECTION set-property 0x16 1
```

2. 使用之前的 sb3kdk.txt 中的值，烧写 0x20 这个fuse；

```
%! blhost $UART_CONNECTION fuse-program 0x20
```

```
[[7aa7ef9813b3561257b8837dab26225301df3511217f2733c71dadcd447722d1]]
```

3. 使用ROTKTH烧写 0x1F 这个fuse：

SB3.1 generation

We have created certificates and keys required for the creation of SB3.1 file. Let's create a SB3.1.

```
In [7]: %! nxpimage $VERBOSEITY sb31 export $SB31_TEMPLATE_PATH
assert _exit_code == 0
assert os.path.exists(WORKSPACE+SB31_FILE_PATH)

nxpimage sb31 export workspace/sb31_config.vml
SB3KDK: 1d5a43bc0adb4cf6c1e6c642ea5b8cb2fa4f1297017fc94d703f00f07dc7e41f
RoTKTH: 9cafd4417941784fd0754b08238bae5793ab8074a6f9df7b93114d01102434a356fab761bd303773c6c6880895a643
Success. (Secure binary 3.1: C:/Users/ )
```

```
%! blhost $UART_CONNECTION fuse-program 0x1F
```

```
[[650d8097079ff27a3e8a2da14781b922fd8295b6c00bfa067f00e87f1a16b8b304b
f710d45cbd591e2e24be83183922c]]
```

4. 烧写TZM_EN这个fuse，它是一个配置位，用来启用或禁用TZM（TrustZone Memory）功能。TZM是一种内存保护机制，可以将内存分为安全区域和非安全区域，防止非安全区域的代码访问安全区域的数据。这个fuse在第一批KW45的样品中没有被设置，导致S3MUA（Secure 3rd-Party Microcontroller Authentication）这个模块的信号量（semaphore）不能正常工作，从而导致写入TR（Trust Register）寄存器时发生总线错误。S3MUA是一种安全认证机制，可以让设备与第三方的微控制器进行安全的通信。如果不使用S3MUA的信号量（例如只有一个线程与S3MUA通信），那么TZM_EN这个fuse可以保持为0，不需要烧写。

```
%! blhost $UART_CONNECTION fuse-program 0xD [[01]]
```

5. 烧写完成，把电压置低：

```
%! blhost $UART_CONNECTION set-property 0x16 0
```

6 用sb3.sb3更新NBU

和之前一样，要将板子置于ISP模式。操作方法一致。使用 `nxpdevscan` 检查是否成功进入ISP模式：

```
nxpdevscan
----- Connected NXP USB Devices -----

----- Connected NXP UART Devices -----

Port: COM5
Type: mboot device

----- Connected NXP SIO Devices -----
```

最后一步是将带有NBU映像的SB3.1文件上传到设备。大约需要几秒钟，到此为止大功告成！

```
%! blhost $UART_CONNECTION receive-sb-file $SB31_FILE_FINAL
```

```
blhost -p com5 receive-sb-file workspace/sb3.sb3
Sending SB file
Response status = 0 (0x0) Success.
```