实验手册：Matter 设备 Passcode 及 Spake2p 相关数据的生成和烧录

Matter 设备在 Commissioning 配网的时候，需要使用 Passcode、Discriminator 等相应的参数。此外，不同的设备厂商生产的 Matter 设备对应的 Product Id 和 Vendor Id 也不一样。这些参数都需要设备厂商在工厂生产的时候烧录到产品中去。本次实验帮助开发者掌握如何使用 Silicon Labs 的解决方案对相应参数进行生成和烧录。

实验目的

- 理解 Passcode、Discriminator、PASE 等概念
- 掌握如何利用 Spake2p 工具生成 Verifier
- 掌握如何使用 Silicon Labs 的脚本文件将 Passcode、Discriminator 和 Spake2p 等相关信息烧录到 Matter 设备

Table of Contents

1. 预备知识

本实验中涉及的专有名词及其解释：

- PASE(Passcode-Authenticated Session Establishment): 基于密码认证的会话建立，用于在 Commissioning 的时候 Commissioner 与 Matter 设备之间建立安全信道，生成对称加密秘钥用于 Commissioning 后续通信消息进行加、解密和完整性保护。
- Passcode(Pincode): 密码或口令，用于在 Commissioning 的时候 PASE 会话建立过程的输入参数之一。
- Discriminator: 鉴别码。Matter 设备在 Commissioning 的状态会进行蓝牙广播(Advertising)，广播中就包含鉴别码信息。Commissioner 在 Commissioning 的过程中通过鉴别码来识别加网的 Matter 设备，两者的鉴别码一致才会开始 Commissioning 流程。
- Spake2p: 密码认证的密钥交换算法，用于 PASE 建立过程中使用的算法。
- Verifier: 验证器，用于在 Commissioning 的时候 PASE 会话建立过程的输入参数之一。
- Iteration count: 迭代次数，用于在 Commissioning 的时候 PASE 会话建立过程的输入参数之一。
- Salt: 盐值，用于在 Commissioning 的时候 PASE 会话建立过程的输入参数之一。

2. 实验目的

通过本次《Matter 设备 Passcode 及 Spake2p 相关数据的生成和烧录》实验，开发者可以理解并掌握以下内容：

- 理解 Passcode、Discriminator、PASE 等概念。
- 掌握如何利用 Spake2p 工具生成 Verifier。
- 掌握如何使用 Silicon Labs 的脚本文件将 Passcode、Discriminator 和 Spake2p 等相关信息烧录到 Matter 设备。

3. 实验内容

Matter 设备在 Commissioning 配网的时候，需要使用 Passcode、Discriminator 等相应的参数。此外，不同的设备厂商生产的 Matter 设备对应的 Product Id 和 Vendor Id 也不一样。这些参数都需要设备厂商在工厂生产的时候烧录到产品中去。本次实验帮助开发者掌握如何使用 Silicon Labs 的解决方案对相应参数进行生成和烧录。

本次实验内容包括编译 Spake2p 工具，利用该工具生成 Spake2p 算法的 Verifier，并使用 Silicon Labs 提供的 FactoryDataProvider.py 脚本将以下 Matter 设备相关信息烧录到 Breakout 开发板的 EFR32MG24 芯片内部 Flash 的 NVM3 区域，以供 Matter 设备在 Commissioning 配网时使用。

- Discriminator
- Passcode
- Spake2p Iteration
- Spake2p Salt
- Spake2p Verifier
- Product Id
- Vendor Id

在本实验中，使用以下示例参数进行实验，开发者可以将以下参数设定为自己厂商设备对应的参数。

参数名	本实验使用的测试示例参数
Discriminator	3860
Passcode	45502468
Spake2p Iteration	1000
Spake2p Salt	U1BBS0UyUCBLZXkgU2FsdA==
Spake2p Verifier	利用 Spake2p 工具生成
Product Id	21845
Vendor Id	4169

4. 实验准备

4.1 硬件准备

- 安装 Ubuntu 虚拟机的电脑一台
- MG24 Breakout 开发板一块
- JLink 烧录器一台
- 安装 Chip-Tool 的树莓派一台（可选）
- EFR32MG21 USB Dongle 一块（可选）

4.2 软件准备

- Virtual Box 虚拟机
- Ubuntu 20.04 镜像
- Silicon Labs Matter SDK
- Linux Commander 工具
- 安装 Linux JLink 驱动

4.2.1 下载 Silicon Labs Matter SDK 源码

- 当 Ubuntu-64 位系统虚拟机安装成功后，更新系统工具，下载 matter 代码，构建编译环境。
    - 更新或安装系统工具。

```
> sudo apt update

> sudo apt upgrade -y

> sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-dev

> sudo reboot
```

    - 下载 matter 代码，更新 submodule，构建编译环境。

```
> git clone https://github.com/SiliconLabs/matter.git

> cd matter
```

```
> git checkout release_1.0.2-1.0

> git pull

> ./scripts/checkout_submodules.py --shallow --recursive --platform efr32

> git submodule update --init --recursive

# 以下步骤需要使用 VPN 网络

> source scripts/bootstrap.sh
```

- 若已下载 matter 代码，则更新。

```
> git fetch

> git checkout release_1.0.2-1.0

> git submodule update --init --recursive

> source scripts/bootstrap.sh
```

- 可能存在代码同步更新失败的问题，导致固件或者工具编译不过，可先保存个人代码，执行如下命令强制同步。（仅供参考，下例强制同步 release_1.0.2-1.0 分支）

```
> git fetch --all

> git reset --hard release_1.0.2-1.0

> git pull
```

4.2.2 安装 Linux Commander 工具

- 在 Ubuntu 中下载 Commander 安装包，并解压。Commander 操作手册请参考文档 UG162: Simplicity Commander Reference Guide。

```
> lesun@Ubuntu20:~/Downloads/SimplicityCommander-Linux$ tar jxvf
Commander_linux_x86_64_1v14p2b1232.tar.bz

# 可以在 commander 目录下看到 commander 执行文件

> cd ./commander
```

- 将 Commander 路径添加到环境变量：

```
> sudo vim /etc/profile

# 将 Commander 的路径添加添加到文件中的最后一行
```

```
> export PATH=$PATH:/home/lesun/Downloads/SimplicityCommander-Linux/commander

# 生效添加的环境变量

> source /etc/profile

 # 在任何目录下，输入 commander 即可打开 Commander 工具。
```

4.2.3 安装 Linux JLink 驱动

- 当前是 Linux Ubuntu 64 位系统，下载 JLink 驱动，选择 JLink_Linux_V784b_x86_64.deb 驱动包下载到~/Downloads 目录，然后使用如下命令安装：

```
> cd ~/Downloads/

> sudo dpkg -i JLink_Linux_V784_x86_64.deb
```

5. 实验步骤

5.1 Ubuntu 虚拟机中编译生成 Spake2p 工具

- 执行如下命令：

```
> cd matter

> source scripts/activate.sh

> gn gen out/host

> ninja -C out/host spake2p
```

- 若需了解命令参数的详细说明，可使用以下命令查看 spake2p 帮助信息

```
> ./out/host/spake2p gen-verifier --help
```

5.2 用 Spake2p 工具生成 Verifier

- 执行如下命令：

```
> ./out/host/spake2p gen-verifier --pin-code 45502468 --iteration-count 1000 --salt
U1BBS0UyUCBLZXkgU2FsdA== --out spake2p-provisioning-data.csv

# 打印 Verifier 等信息，其中 Verifier 在后续步骤中需要使用

> cat spake2p-provisioning-data.csv

Index,PIN Code,Iteration Count,Salt,Verifier

0,45502468,1000,U1BBS0UyUCBLZXkgU2FsdA==,nftJcSzyp6mht4y8oCy+0pDKJaSZhK0BLCN/vICK010EUeP0
XE8ZC8zWjvwXiPE18b5DSyzRILkyLrMfqhUjmJrb/ysXoWmycdrLEOsfp0iWfbbZGDNqp0C8iv6yAQCQkg==
```

- 参数说明：
    - --pin-code 是设定的 pincode，用于生成 spake2_verifier 的参数之一。
    - --iteration-count 是 SPAKE2P PBKDF 的迭代次数，用于生成 spake2_verifier 的参数之一。
    - --spake2_salt 是 SPAKE2P PBKDF 的盐值，用于生成 spake2_verifier 的参数之一。
    - --out 是生成的 csv 文件，包含 Verifier 等信息。

5.3 修改 FactoryDataProvider.py 脚本，使得运行脚本时能正确识别 Matter Breakout 开发板

5.3.1 修改 FactoryDataProvider.py 脚本中的 script_dir 变量

- 执行如下命令：

```
> cd matter/scripts/tools/silabs

# 如果不支持 Vim 工具请自行安装，或者使用其它编辑方法

> vim FactoryDataProvider.py
```

- 将该脚本文件中第 27 行语句" script_dir = os.path.dirname(file)" 修改为当前的绝对路径，例如改为：script_dir = "/home/lesun/matter/scripts/tools/silabs"。因为在某些系统下，可能会存在不兼容的情况导致脚本运行失败。

```
.....

class FactoryDataWriter:

#    script_dir = os.path.dirname(__file__)

   script_dir = "/home/lesun/matter/scripts/tools/silabs"

.....
```

5.3.2 修改 FactoryDataProvider.py 脚本中的 Commander 命令，添加对 "--device" 的支持

```
# 将 python 脚本中执行 commander 命令的所有地方添都加'--device', mcu_family 参数：

# 在第 184 行，添加

mcu_family = self._args.mcu_family

# 第 185 行，改为

cmd = ['commander', 'nvm3', 'read', '-o', inputImage, '--device', mcu_family，]

# 第 191 行，改为

cmd = ['commander', 'device', 'info', '--device', mcu_family, ]

# 第 228 行，改为

"commander", "nvm3", "set", inputImage, '--device', mcu_family,

# 第 277 行，改为

cmd = ['commander', 'flash', '--device', mcu_family, self.OUT_FILE, ]
```

5.4 利用 FactoryDataProvider.py 脚本将 Passcode 和 Spake2p 相关信息烧录到 Breakout 开发板

- 首先，将 JLink 和 Matter Breakout 开发板相连，并将 JLink 的 USB 线插入电脑 USB 端口。
- 在烧录之前，检查 JLink 是否被 Ubuntu 识别到，输入 commander 命令，启动 Commander 工具，点击“Select Kits”，确保能识别到 JLink 的 USB 端口，如下图 1 所示。如果识别不到 USB 端口，需要先解决该问题再进行下一步操作。

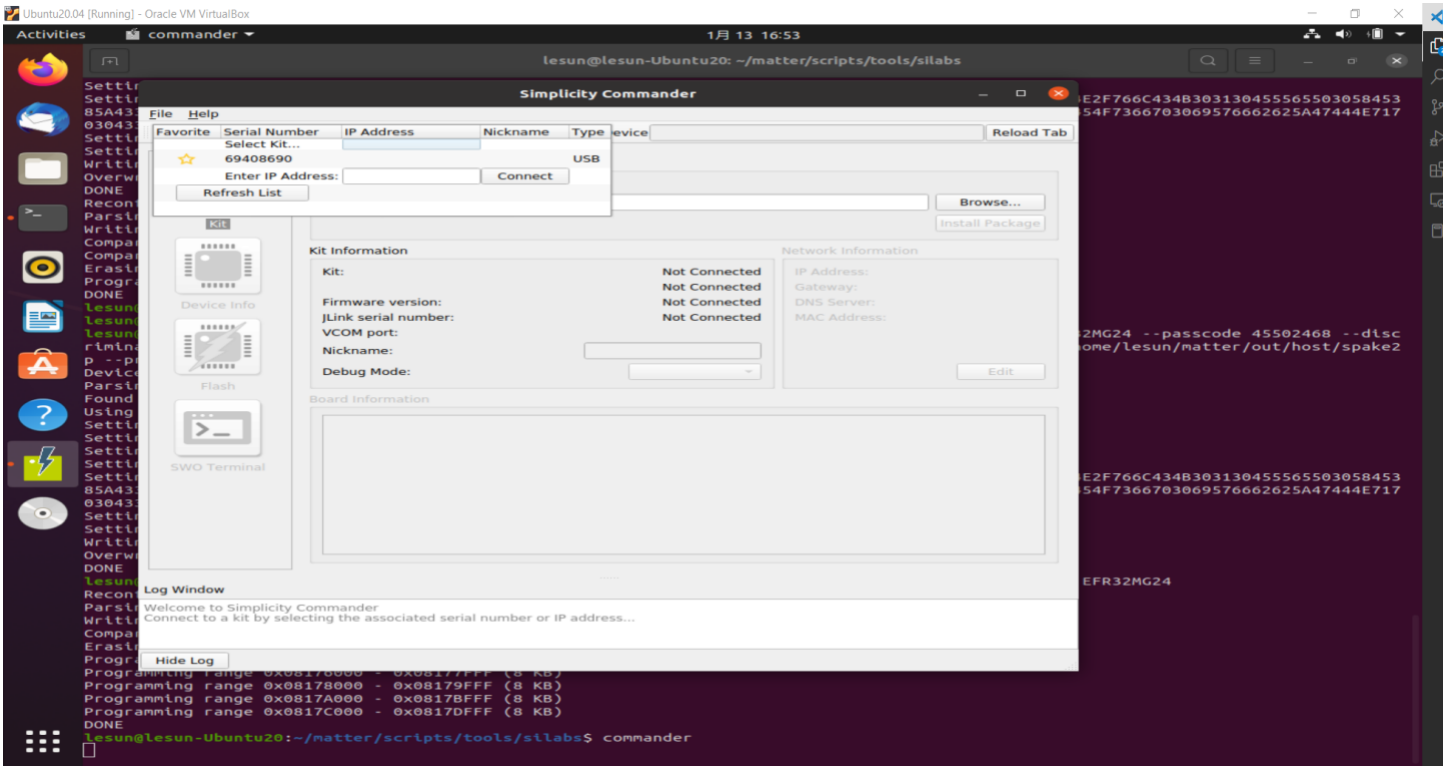

图 1 – 使用 commander 工具识别 J-Link 的 USB 端口

- 执行如下命令：

```
> cd matter/scripts/tools/silabs

> python3 FactoryDataProvider.py --mcu_family EFR32MG24 --passcode 45502468 --discriminator 3860 --spake2_iteration 1000 --spake2_salt U1BBS0UyUCBLZXkgU2FsdA== --spake2_verifier nftJcSzyp6mht4y8oCy+0pDKJaSZhK0BLCN/vICK010EUeP0XE8ZC8zWjvwXiPE18b5DSyzRlLkyLrMfqhUjmJrb/ysXoWmycdrLEOsfp0iWfbbZGDNqp0C8iv6yAQCQkg==  --product_id 21845 --vendor_id 4169
```

- 该脚本实现以下功能：
  - 将 passcode、discriminator、spake2_iteration、spake2_salt、spake2_verifier、product_id、vendor_id 等参数写入 NVM3 区域并生成.s37 固件

- 使用 Commander 工具将.s37 固件烧录到 EFR32MG24 芯片。

- 参数说明：
  - --mcu_family 是设定 EFR32MG24 芯片类型。
  - --passcode 是设定 PASE 会话的 passcode。
  - --iteration-count 是设定 SPAKE2P PBKDF 的迭代次数。
  - --spake2_salt 是设定 SPAKE2P PBKDF 的盐值。
  - --spake2_verifie 是设定 SPAKE2P PBKDF 的的 Verifier.
  - --product_id 是设定产品标识符
  - --vendor_id 是设定设备厂商标识符。
- 执行成功的日志信息：

```
> python3 FactoryDataProvider.py --mcu_family EFR32MG24 --passcode 45502468 --discriminator 3860 --spake2_iteration 1000 --spake2_salt U1BBS0UyUCBLZXkgU2FsdA== --spake2_verifier nftJcSzyp6mht4y8oCy+0pDKJaSZhK0BLCN/vICK010EUeP0XE8ZC8zWjvwXiPE18b5DSyzRlLkyLrMfqhUjmJrb/ysXoWmycdrLEOsfp0iWfbbZGDNqp0C8iv6yAQCQkg==  --product_id 4169  --vendor_id 21845


Parsing file /home/lesun/matter/scripts/tools/silabs/tmp_nvm3.s37...

Found NVM3 range: 0x08174000 - 0x0817E000

Using 4096 B as maximum object size, based on given size of NVM3 area.

Setting NVM3 object: 0x87207 = 140F

Setting NVM3 object: 0x87205 = A8AA4A824080E209A06C05

Setting NVM3 object: 0x87208 = E8030000

Setting NVM3 object: 0x87209 = 553142425330557955434 24C5A586B675532467364413D3D

Setting NVM3 object: 0x8720a = 6E66744A63537A7970366D68743479386F43792B3070444B4A61535A684B30424C434E2F766C434B3031304555655030584538 5A43387A576A767758695045313862354453797A526C4C6B794C724D667168556A6D4A72
```

```
622F7973586F576D796364724C454F7366703069576662625A47444E71703043386976367941514351 6B673D3D

Setting NVM3 object: 0x8720b = 5555

Setting NVM3 object: 0x8720c = 4910

Writing to /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

Overwriting file: /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

DONE

Reconfiguring debug connection with detected device part number: EFR32MG24B210F1536IM48

Parsing file /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

Writing 40960 bytes starting at address 0x08174000

Comparing range 0x08174000 - 0x0817DFFF (40 KB)

Comparing range 0x08176000 - 0x0817DFFF (32 KB)

Erasing range 0x08174000 - 0x08175FFF (1 sector, 8 KB)

Programming range 0x08174000 - 0x08175FFF (8 KB)

DONE
```

5.5 （可选）也可以将步骤 2 和步骤 4 合并，直接将 Spake2p 工具路径作为 FactoryDataProvider.py 脚本的输入参数，生成 verifier 并将 Passcode 和 Spake2p 相关信息烧录到 Breakout 开发板

- 执行如下命令：

```
> cd matter/scripts/tools/silabs

> python3 FactoryDataProvider.py --mcu_family EFR32MG24 --passcode 45502468 --discriminator 3860 --spake2_iteration 1000 --spake2_salt U1BBS0UyUCBLZXkgU2FsdA== --gen_spake2p_path /home/lesun/matter/out/host/spake2p --product_id 21845 --vendor_id 4169
```

- 执行成功的日志信息如下。通过对比，可以发现烧录的内容和步骤 4 烧录的内容一致。

```
> python3 FactoryDataProvider.py --mcu_family EFR32MG24 --passcode 45502468 --discriminator 3860 --spake2_iteration 1000 --spake2_salt U1BBS0UyUCBLZXkgU2FsdA== --gen_spake2p_path /home/lesun/matter/out/host/spake2p  --product_id 21845 --vendor_id 4169

Parsing file /home/lesun/matter/scripts/tools/silabs/tmp_nvm3.s37...
```

```
Found NVM3 range: 0x08174000 - 0x0817E000

Using 4096 B as maximum object size, based on given size of NVM3 area.

Setting NVM3 object: 0x87207 = 140F

Setting NVM3 object: 0x87205 = A8AA4A824080E209A06C05

Setting NVM3 object: 0x87208 = E8030000

Setting NVM3 object: 0x87209 = 553142425330557955434 24C5A586B675532467364413D3D

Setting NVM3 object: 0x8720a =
6E66744A63537A7970366D68743479386F43792B3070444B4A61535A684B30424C434E2F766C434B3031304
5556550305845385A43387A576A767758695045313862354453797A526C4C6B794C724D667168556A6D4A72
622F7973586F576D796364724C454F736670306956766 2625A47444E71703043386976367941514 3516B673D
3D

Setting NVM3 object: 0x8720b = 5555

Setting NVM3 object: 0x8720c = 4910

Writing to /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

Overwriting file: /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

DONE

Reconfiguring debug connection with detected device part number: EFR32MG24B210F1536IM48

Parsing file /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

Writing 40960 bytes starting at address 0x08174000

Comparing range 0x08174000 - 0x0817DFFF (40 KB)

Comparing range 0x08176000 - 0x0817DFFF (32 KB)

Erasing range 0x08174000 - 0x08175FFF (1 sector, 8 KB)

Programming range 0x08174000 - 0x08175FFF (8 KB)

DONE
```

5.6 （可选）也可以将步骤 4 分解成两步，首先生成.s37 固件，然后利用 Commander 工具将.s37 固件烧录到 Breakout 开发板

5.6.1 利用 FactoryDataProvider.py 脚本将 Passcode 和 Spake2p 相关信息写入到.s37 固件

- 首先，将 JLink 的 USB 线从电脑 USB 端口移除。
- 执行如下命令：

```
> cd matter/scripts/tools/silabs

> python3 FactoryDataProvider.py --mcu_family EFR32MG24 --passcode 45502468 --discriminator 3860 --spake2_iteration 1000 --spake2_salt U1BBS0UyUCBLZXkgU2FsdA== --gen_spake2p_path /home/lesun/matter/out/host/spake2p --product_id 21845 --vendor_id 4169
```

- 执行成功的日志信息：

```
> python3 FactoryDataProvider.py --mcu_family EFR32MG24 --passcode 45502468 --discriminator 3860 --spake2_iteration 1000 --spake2_salt U1BBS0UyUCBLZXkgU2FsdA== --gen_spake2p_path /home/lesun/matter/out/host/spake2p --product_id 4169 --vendor_id 21845


Device not connected

Parsing file /home/lesun/matter/scripts/tools/silabs/base_matter_mg24_nvm3.s37...

Found NVM3 range: 0x08174000 - 0x0817E000

Using 4096 B as maximum object size, based on given size of NVM3 area.

Setting NVM3 object: 0x87207 = 140F

Setting NVM3 object: 0x87205 = A8AA4A824080E209A06C05

Setting NVM3 object: 0x87208 = E8030000

Setting NVM3 object: 0x87209 = 553142425330557955434 24C5A586B675532467364413D3D

Setting NVM3 object: 0x8720a =
6E66744A63537A7970366D68743479386F43792B3070444B4A61535A684B30424C434E2F766C434B3031304
5556550305845385A43387A576A767758695045313862354453797A526C4C6B794C724D667168556A6D4A72
622F7973586F576D796364724C454F736670306957666262 5A47444E71703043386976367941514 3516B673D
3D
```

```
Setting NVM3 object: 0x8720b = 5555

Setting NVM3 object: 0x8720c = 4910

Writing to /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

Overwriting file: /home/lesun/matter/scripts/tools/silabs/matter_factorydata.s37...

DONE
```

5.6.2 利用 Commander 工具将.s37 固件烧录到 Breakout 开发板

- 首先，将 JLink 的 USB 线插入电脑 USB 端口，并确保 Ubuntu 下能识别 JLink 的 USB 端口。如果识别不到 USB 端口，需要先解决该问题再进行下一步操作。
- 执行如下命令：

```
> commander flash matter_factorydata.s37 --device EFR32MG24
```

- 执行成功的日志信息：

```
commander flash matter_factorydata.s37 --device EFR32MG24

Reconfiguring debug connection with detected device part number: EFR32MG24B210F1536IM48

Parsing file matter_factorydata.s37...

Writing 40960 bytes starting at address 0x08174000

Comparing range 0x08174000 - 0x0817DFFF (40 KB)

Erasing range 0x08174000 - 0x0817DFFF (5 sectors, 40 KB)

Programming range 0x08174000 - 0x08175FFF (8 KB)

Programming range 0x08176000 - 0x08177FFF (8 KB)

Programming range 0x08178000 - 0x08179FFF (8 KB)

Programming range 0x0817A000 - 0x0817BFFF (8 KB)

Programming range 0x0817C000 - 0x0817DFFF (8 KB)

DONE
```

6. 利用树莓派的 Chip-Tool 进行 Commissioning 配网验证

- 在树莓派中，使用 chip-tool 工具配网 Matter 设备，如果设备成功入网，则表明烧录的 NVM3 区域的 Spake2p 等参数成功生效。
- 本实验指定 pincode 为 45502468，discriminator 为 3860，使用 chip-tool 命令触发设备入网的指令如下：

```
> cd connectedhomeip

> sudo rm -rf /tmp/chip_*

> mattertool startThread

> sudo ot-ctl dataset active -x

> ./out/standalone/chip-tool pairing ble-thread 1234 hex:<thread-dataset> 45502468 3860

> ./out/standalone/chip-tool onoff toggle 1234 1
```

7. 常见问题

- 在 Ubuntu 中安装完 Simplicity Commander 工具之后，需要设置环境变量。
- 如果 JLink 的 USB 端口无法在 Ubuntu 下识别，检查虚拟机的 USB 设置是否正确。也可以重新拔插与电脑相连的 JLink 的 USB 线。
- 在做 Commissioning 配网验证开始前，需要先烧入匹配的 Bootloader 和 Matter 设备的固件。

8. 参考资料

- SPAKE2+算法 —— Spake2p 算法介绍。
- Matter 源码 —— Silicon Labs 提供的 Matter 源码。
- Spake2p 工具 —— CSA 联盟提供的 Spake2p 工具，用于生成 Spake2p 算法的 Verifier。
- FactoryDataProvider.py 脚本 —— Silicon Labs 提供的 Python 脚本，用于烧录 Matter 设备安全相关的信息。