BiTDB + OP-TEE on Raspberry Pi 3B/3B+

基于交叉编译容器的构建与部署指南

（内部技术文档示例）

2025 年 12 月 3 日

目录

1 文档概述

本文档针对在 Raspberry Pi 3B / 3B+ 平台上部署 BiTDB + OP-TEE 系统的场景，给出一套基于交叉编译容器的完整构建与烧写流程，重点解决以下问题：

- 如何在 x86_64 宿主机上使用预配置的容器环境，避免误用本地编译器生成 x86_64 可执行文件；
- 如何正确交叉编译 ATF、OP-TEE OS、Linux 内核、optee_client、BiTDB 示例程序及 TA；
- 如何将生成的产物刷入 TF 卡的 `boot` 与 `rootfs` 分区；
- 上板后如何验证 OP-TEE、tee_supplicant 与 BiTDB 示例程序；
- 整个过程中容易踩到的坑及对应规避策略。

1.1 目标读者

- 已具备 Linux 基本使用经验的开发者；
- 需要在 Raspberry Pi 3 平台上开发和调试 OP-TEE 及 BiTDB 示例的研究人员或工程师；
- 使用预制交叉编译容器镜像进行构建的后续使用者。

1.2 系统架构简要说明

本系统在 Raspberry Pi 3B / 3B+ 上采用如下典型组合：

- SoC：Raspberry Pi 3B / 3B+（ARMv8）；
- 安全世界（EL3 + Secure EL1）：ARM Trusted Firmware (ATF) 运行在 AArch64 模式；
- 安全世界 TEE Core：OP-TEE OS 运行在 AArch32 模式（Secure EL1）；
- 普通世界内核：32 位 ARM Linux；
- 普通世界用户态组件：AArch32 ELF，可执行程序和共享库。

构建策略上：

- ATF 使用 AArch64 交叉工具链编译；
- Linux 内核、OP-TEE OS Core、optee_client、BiTDB 示例等均使用 AArch32 交叉工具链。

这一组合是 ARMv8 平台上常见且合理的配置，不是“配置混乱”，需要开发者在工具链选择上保持一致性。

2 环境与前提条件

2.1 宿主机环境

- 架构：x86_64；
- 操作系统：Linux（建议 Ubuntu 系列，版本不限于 22.04/24.04）；
- 已安装 Docker 或兼容容器运行环境；
- 有足够磁盘空间存放源码和构建产物。

2.2 开发容器镜像假定

以下内容根据实际镜像略作调整，本指南采用示例设定：

- 镜像名称：`bitdb-optee-dev:latest`；
- 容器中的工程根目录：`/bitdb`；
- 容器内部假定在 `/opt/toolchains` 下可见交叉工具链：
  - AArch32:`/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/`；
  - AArch64:`/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/`。

  这些工具链可以由镜像内置，也可以通过宿主机目录挂载方式提供（见后文“恢复交叉工具链目录”）。
- 已在 `/bitdb/config.mk` 中配置好默认的交叉编译前缀：
  - AArch32 工具链用于 Linux、OP-TEE OS Core、optee_client、示例应用等；
  - AArch64 工具链用于 ATF。

2.3 目录布局建议

在宿主机上准备一个工程根目录，例如：

```
mkdir -p ~/bitdb-optee-dev
# 将 bitdb 源码放在：
# ~/bitdb-optee-dev/bitdb
```

后续会将 `~/bitdb-optee-dev/bitdb` 挂载到容器内的 `/bitdb`，使得容器可随时销毁，而源码与 out 产物由宿主机持久保存。

2.4 从备份压缩包恢复 bitdb 项目目录

在分发本项目时，通常会将“容器镜像”和“源码目录”作为两个独立文件发布，例如：

- 容器镜像：`bitdb-optee-dev-rpi3-optee-v1-20251126.tar`；
- 源码压缩包：`bitdb-src-rpi3-optee-v1-20251126.tar.gz`。

在一台全新的宿主机上恢复 `bitdb` 项目目录的推荐步骤如下。

1) 导入容器镜像（一次性操作）

```
# 导入镜像 tar
docker load -i bitdb-optee-dev-rpi3-optee-v1-20251126.tar

# 可选：确认镜像已存在
docker images | grep bitdb-optee-dev
```

2) 在宿主机上解压 bitdb 源码目录

```
# 1. 准备工程根目录
mkdir -p ~/bitdb-optee-dev
cd ~/bitdb-optee-dev

# 2. 解压源码压缩包（文件名根据实际版本调整）
tar xzf ~/Downloads/bitdb-src-rpi3-optee-v1-20251126.tar.gz
```

解压完成后，目录结构应类似：

```
~/bitdb-optee-dev/
    bitdb/
        config.mk
        optee_client/
        optee_examples/
        optee_os/
        arm-trusted-firmware/
        linux/
        out/ # 若压缩包中包含已有构建产物
```

2.5 从备份压缩包恢复交叉工具链目录

除了 bitdb 源码目录之外，还需要恢复 Linaro 交叉工具链，以保证容器内的 /opt/toolchains 路径可用。推荐在宿主机的 ~/bitdb-optee-dev/toolchains 目录中存放解压后的工具链。

典型的工具链压缩包（文件名仅供参考）例如：

- gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf.tar.xz
- gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz

在宿主机上执行：

```
cd ~/bitdb-optee-dev
mkdir -p toolchains
cd toolchains

# 将上述两个工具链压缩包放到当前目录后，解压：
tar xf gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf.tar.xz
tar xf gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
```

解压完成后，目录结构大致为：

```
~/bitdb-optee-dev/toolchains/
    gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/
    gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/
```

后续运行容器时,通过挂载方式将该目录暴露为容器内的 /opt/toolchains,使其与 config.mk 中的工具链路径保持一致:

```
cd ~/bitdb-optee-dev

docker run --rm -it \
  -v "$PWD/bitdb:/bitdb" \
  -v "$PWD/toolchains:/opt/toolchains" \
  bitdb-optee-dev:latest \
  /bin/bash
```

进入容器后，可检查工具链是否就绪:

```
root@<container-id>:/# \
/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-
    gnueabihf-gcc -v 2>&1 | grep Target
# 预期输出: Target: arm-linux-gnueabihf
```

若使用的是“自带工具链”的全量镜像，则也可以不挂载 toolchains 目录，直接使用镜像内部的 /opt/toolchains；但出于可维护性、可升级性和可溯源性考虑，推荐将工具链与源码一起在宿主机显式管理，并通过挂载方式提供给容器。

2.6 核心防坑原则

1) 所有构建动作必须在容器内完成，不在 x86_64 宿主机直接对同一源码树执行 make。

2) 对以下关键二进制必须使用 file 命令检查其架构:

- tee-supplicant;
- libteec.so*;
- optee_example_bitdb;
- BiTDB TA（*.ta，一般为 ARM 32 位，建议抽查)。

3) 刷 TF 卡前，只信任在容器内确认过为 ARM32 的文件，并从对应目录手动拷贝，避免误将历史 x86_64 产物覆盖到根文件系统中。

3 容器启动与工具链检查

3.1 导入镜像并启动容器

如果镜像以 tar 包形式提供，例如 bitdb-optee-dev.tar，可在宿主机执行:

```
# 导入镜像
docker load -i bitdb-optee-dev.tar
```

```
# 确认镜像存在
docker images | grep bitdb-optee-dev
```

随后启动容器并挂载工程目录与工具链目录:

```
cd ~/bitdb-optee-dev

docker run --rm -it \
  -v "$PWD/bitdb:/bitdb" \
  -v "$PWD/toolchains:/opt/toolchains" \
  bitdb-optee-dev:latest \
  /bin/bash
```

进入容器后:

```
root@<container-id>:/# cd /bitdb
root@<container-id>:/bitdb#
```

3.2 检查 AArch32 交叉工具链

在容器内执行:

```
/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-
    gnueabihf-gcc -v 2>&1 | grep Target
```

预期输出应包含:

```
Target: arm-linux-gnueabihf
```

说明该工具链是面向 ARM 32 位的 EABI 硬浮点目标。

3.3 config.mk 关键配置示例

在 /bitdb/config.mk 中应存在类似配置（仅示意):

```
AARCH32_TOOLCHAIN_PATH := /opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-
    gnueabihf
AARCH64_TOOLCHAIN_PATH := /opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-
    linux-gnu

CROSS_COMPILE := $(AARCH32_TOOLCHAIN_PATH)/bin/arm-linux-gnueabihf-
CROSS_COMPILE_core := $(CROSS_COMPILE)
CROSS_COMPILE_ta_arm32 := $(CROSS_COMPILE)
```

经验教训 1: 若 CROSS_COMPILE 配错为宿主机 gcc 等，则:

- tee-supplicant、optee_example_bitdb 等会被误编译为 x86_64;
- 上板运行时会看到如 “Syntax error: ”)” unexpected” 之类的错误;
- file 命令会显示 “ELF 64-bit LSB pie executable, x86-64”。

4 全局构建流程

以下所有构建操作均在容器内执行。

4.1 设置并行度

```
cd /bitdb
JOBS=$(nproc 2>/dev/null || echo 4)
```

4.2 步骤一：构建 ATF + U-Boot + OP-TEE （arm-tf-final）

```
make -j"$JOBS" arm-tf-final
```

该目标通常完成以下工作：

- 使用 AArch64 工具链编译 ARM Trusted Firmware；
- 将 OP-TEE 作为 BL32 集成到 ATF；
- 构建并集成 U-Boot（32 位）；
- 生成并导出关键文件：
  - `arm-trusted-firmware/build/rpi3/debug/fip.bin`；
  - `arm-trusted-firmware/build/rpi3/debug/armstub8.bin`；
  - 将 `armstub8.bin` 拷贝为 `out/boot/armstub8.bin`。

构建完成时，日志中会出现 “Built fip.bin successfully” 和拷贝 `armstub8.bin` 的提示。

4.3 步骤二：构建 Linux 内核、rootfs 与示例程序

```
make -j"$JOBS"
```

该步骤通常包括：

- 编译 Linux 内核与模块，生成 `uImage`；
- 编译 OP-TEE OS Core（如前一步未完成）；
- 编译 `optee_client`（`libteec.so` 与 `tee-supplicant`）；
- 编译 `optee_examples` 中的 BiTDB 示例（Host CA 与 TA）；
- 将部分产物拷贝到 `out/boot/` 与 `out/rootfs/`。

需要注意的是，顶层 `make` 可能会复用历史构建产物，后续仍需对关键组件进行架构检查与手动覆盖。

5 optee_client 与 BiTDB 示例的架构确认与覆盖

本节是整个构建过程中最关键的防坑步骤之一。

5.1 optee_client：tee-supplicant 与 libteec.so

在容器内执行：

```
cd /bitdb/optee_client

file out/tee-supplicant/tee-supplicant
file out/libteec/libteec.so.1.0
```

预期输出应类似：

```
out/tee-supplicant/tee-supplicant: ELF 32-bit LSB executable, ARM, EABI5 ...
out/libteec/libteec.so.1.0: ELF 32-bit LSB shared object, ARM, EABI5 ...
```

若检测结果为 x86_64，需重新构建：

```
cd /bitdb/optee_client
make clean
rm -rf out/
JOBS=$(nproc 2>/dev/null || echo 4)

make -j"$JOBS" \
  CROSS_COMPILE=/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/
      bin/arm-linux-gnueabihf- \
  CFG_TEE_BENCHMARK=n

file out/tee-supplicant/tee-supplicant
file out/libteec/libteec.so.1.0
```

5.2 BiTDB Host 示例程序：optee_example_bitdb

```
cd /bitdb/optee_examples/bitdb/host
file optee_example_bitdb
```

预期输出为 ARM 32 位 ELF：

```
optee_example_bitdb: ELF 32-bit LSB executable, ARM, EABI5 ...
```

若检测结果为 x86_64，按如下方式重编：

```
cd /bitdb/optee_examples/bitdb/host
make clean
rm -f optee_example_bitdb

cd /bitdb/optee_examples

JOBS=$(nproc 2>/dev/null || echo 4)
make -j"$JOBS" \
```

```
  HOST_CROSS_COMPILE=/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-
      gnueabihf/bin/arm-linux-gnueabihf- \
  TEEC_EXPORT=/bitdb/optee_client/out/export \
  TA_DEV_KIT_DIR=/bitdb/optee_os/out/arm/export-ta_arm32

cd /bitdb/optee_examples/bitdb/host
file optee_example_bitdb
```

5.3 BiTDB TA 文件检查

可选地对 TA 文件进行检查:

```
cd /bitdb/optee_examples/bitdb/ta
file *.ta
```

一般应为 ARM 32 位共享对象或 OP-TEE TA 格式。

5.4 将已确认组件覆盖到 out/rootfs

在容器内执行:

```
cd /bitdb

# 1) tee-supplicant -> /usr/local/bin
mkdir -p out/rootfs/usr/local/bin
cp -v optee_client/out/tee-supplicant/tee-supplicant \
     out/rootfs/usr/local/bin/tee-supplicant

# 2) libteec.so* -> /usr/local/lib
mkdir -p out/rootfs/usr/local/lib
cp -v optee_client/out/libteec/libteec.so* \
     out/rootfs/usr/local/lib/

# 3) TA -> /lib/optee_armtz + /usr/local/lib/optee_armtz
mkdir -p out/rootfs/lib/optee_armtz
cp -v optee_examples/bitdb/ta/*.ta \
     out/rootfs/lib/optee_armtz/

mkdir -p out/rootfs/usr/local/lib/optee_armtz
cp -v optee_examples/bitdb/ta/*.ta \
     out/rootfs/usr/local/lib/optee_armtz/

# 4) optee_example_bitdb -> /usr/local/bin
cp -v optee_examples/bitdb/host/optee_example_bitdb \
     out/rootfs/usr/local/bin/optee_example_bitdb
```

经验教训 2: 不要直接信任 `out/rootfs/bin` 下的自动复制结果，可能残留历史 x86_64 产物。应以 `optee_client/out` 与 `optee_examples` 中经过 `file` 检查的 ARM32 文件为准，再手动覆盖到 `out/rootfs/usr/local/...`。

6 TF 卡刷写流程

本节操作在宿主机完成。

6.1 首次准备：刷写基础 Raspbian 镜像

在进行 OP-TEE 相关文件覆盖之前，需先在 TF 卡上刷入一份官方 Raspbian 基础镜像，本文以 2019-06-20-raspbian-buster.img 为例。该镜像负责提供完整的根文件系统和基础用户空间环境，我们后续只替换其中极少量与启动和 OP-TEE 相关的文件。

1) 准备镜像文件 从官方渠道下载 2019-06-20-raspbian-buster.img（或对应的 .img.zip / .img.xz 压缩包），解压后得到 2019-06-20-raspbian-buster.img，假定放在：

```
~/Downloads/2019-06-20-raspbian-buster.img
```

2) 确认 TF 卡设备名 插入 TF 卡后，在宿主机上执行：

```
lsblk
```

假定 TF 卡对应的整盘设备为 /dev/sdb（注意 **不是** /dev/sdb1、/dev/sdb2），实际名称请以 lsblk 输出为准。

3) 确保未挂载任何分区 如果系统自动挂载了 TF 卡分区，需要先卸载：

```
sudo umount /dev/sdb1 2>/dev/null || true
sudo umount /dev/sdb2 2>/dev/null || true
```

如还有其它分区（/dev/sdb3 等），也一并卸载。

4) 使用 dd 刷写基础镜像

```
sudo dd if=~/Downloads/2019-06-20-raspbian-buster.img \
       of=/dev/sdb \
       bs=4M conv=fsync status=progress
```

该步骤会 **覆盖整张 TF 卡**，请务必确认 of= 指向的是 TF 卡而不是系统盘。

完成后建议执行：

```
sync
```

5) 重新插拔或重新扫描分区 刷写完成后，可以将 TF 卡拔出再插入一次，或者触发内核重新扫描分区表（部分发行版自动完成）。再次执行：

```
lsblk
```

此时通常会看到类似：

- /dev/sdb1：boot 分区（vfat），容量约几十 ~ 一百多 MB；
- /dev/sdb2：rootfs 分区（ext4），容量为 TF 卡其余空间。

后续章节中对 boot / rootfs 的覆盖操作，都是在这张已经刷入 2019-06-20-raspbian-buster.img 的卡的基础上进行，从而只替换少量启动文件和 OP-TEE 相关组件，而无需自己从零构建完整 rootfs。

6.2 挂载 TF 卡分区

在完成基础镜像刷写并确认 /dev/sdb1、/dev/sdb2 存在后，创建挂载点并挂载：

```
sudo mkdir -p /media/$USER/boot /media/$USER/rootfs
sudo mount /dev/sdb1 /media/$USER/boot
sudo mount /dev/sdb2 /media/$USER/rootfs

mount | grep /media/$USER
```

6.3 覆盖 boot 分区文件

在宿主机工程根目录（例如 ~/bitdb-optee-dev/bitdb）执行：

```
cd ~/bitdb-optee-dev/bitdb

sudo cp -v out/boot/armstub8.bin \
          out/boot/bcm2710-rpi-3-b.dtb \
          out/boot/bcm2710-rpi-3-b-plus.dtb \
          out/boot/config.txt \
          out/boot/u-boot.bin \
          out/boot/uboot.env \
          out/boot/uImage \
          /media/$USER/boot/
```

经验教训 3：关于 vfat 与 cp -a。

- boot 分区通常为 vfat，不支持完整的 UID/GID/权限语义；
- 使用 cp -a 会产生 “无法保留所有者: 不允许的操作” 之类的提示；
- 因此对 boot 分区建议只使用 cp -v，不要加 -a。

6.4 覆盖 rootfs 分区关键组件

```
cd ~/bitdb-optee-dev/bitdb

# 1) tee-supplicant 与 optee_example_bitdb
sudo mkdir -p /media/$USER/rootfs/usr/local/bin

sudo cp -v out/rootfs/usr/local/bin/tee-supplicant \
        /media/$USER/rootfs/usr/local/bin/tee-supplicant

sudo cp -v out/rootfs/usr/local/bin/optee_example_bitdb \
        /media/$USER/rootfs/usr/local/bin/optee_example_bitdb

# 2) libteec.so* (推荐只放在 /usr/local/lib)
sudo mkdir -p /media/$USER/rootfs/usr/local/lib
sudo cp -v out/rootfs/usr/local/lib/libteec.so* \
        /media/$USER/rootfs/usr/local/lib/
```

```
# 3) TA 文件
sudo mkdir -p /media/$USER/rootfs/lib/optee_armtz
sudo cp -v out/rootfs/lib/optee_armtz/*.ta \
        /media/$USER/rootfs/lib/optee_armtz/

sudo mkdir -p /media/$USER/rootfs/usr/local/lib/optee_armtz
sudo cp -v out/rootfs/usr/local/lib/optee_armtz/*.ta \
        /media/$USER/rootfs/usr/local/lib/optee_armtz/
```

刷新缓存并卸载分区：

```
sync
sudo umount /media/$USER/boot
sudo umount /media/$USER/rootfs
```

至此，TF 卡准备完毕，可以插入树莓派启动。

7 板端验证与运行 BiTDB 示例

7.1 验证 OP-TEE 驱动加载情况

树莓派启动后在终端执行：

```
dmesg | grep -i tee
```

预期看到类似输出：

```
optee: probing for conduit method from DT.
optee: initialized driver
```

同时检查设备节点：

```
ls /dev/tee*
# 常见输出：/dev/tee0 /dev/teepriv0
```

若无相关日志或设备节点，说明 OP-TEE 驱动未正常初始化，需回溯检查内核配置、ATF/设备树和启动参数。

7.2 启动 tee-supplicant

```
sudo /usr/local/bin/tee-supplicant -d &
```

若二进制是 x86_64，则可能出现如下错误：

- “Syntax error: ”)” unexpected”；
- 或 “cannot execute binary file” 等。

此时可用 `file` 再次确认：

```
file /usr/local/bin/tee-supplicant
```

若仍是 x86_64，说明之前刷入的文件有误，需要回到容器环境重新检查并覆盖。

7.3 运行 BiTDB 示例程序

在树莓派上执行:

```
sudo /usr/local/bin/optee_example_bitdb
```

若程序正常运行并输出 BiTDB 相关日志，说明:

- ATF + OP-TEE OS + Linux 内核协同正常;
- `tee-supplicant` 能通过 `/dev/tee0` 与 TEE 通信;
- `libteec.so` 与 TA 的版本匹配且架构正确;
- 容器构建环境配置正确，关键二进制均为 ARM 32 位。

8 常见坑点与经验总结

8.1 在宿主机误编译导致 x86_64 可执行文件混入

症状

- `file` 显示为 “ELF 64-bit LSB pie executable, x86-64”;
- 在树莓派上运行时出现 “Syntax error: ”)” unexpected”、“cannot execute binary file” 等错误。

原因

- 在 x86_64 宿主机上直接对源码执行 `make`，使用宿主编译器生成了 x86_64 产物;
- 顶层 `make` 又将这些历史产物打包进 `out/rootfs`，最终被刷入 TF 卡。

规避

- 所有构建命令一律在容器内执行;
- 构建完成后，用 `file` 检查 `tee-supplicant`、`optee_example_bitdb`、`libteec.so*` 等关键文件;
- 刷 TF 卡前，在宿主机上再次抽查 `/media/$USER/rootfs/usr/local/bin/` 等路径中的文件架构，确保为 ARM32。

8.2 顶层 make 复用历史产物导致 “以为更新，实际没更新”

症状

- 在 `optee_client/out` 中已重新编译并确认了 ARM32 版本;
- 但 `out/rootfs/bin` 或最终 TF 卡中的文件仍为旧版本（甚至是 x86_64）。

规避

- 不依赖 `out/rootfs/bin` 下自动 copy 的结果；
- 始终从 `optee_client/out` 与 `optee_examples` 下经过 `file` 检查的 ARM32 文件出发，手动覆盖到 `out/rootfs/usr/local/bin` 和 `out/rootfs/usr/local/lib`；
- 刷卡时只复制 `usr/local/...` 下的这些已确认文件。

8.3 在 vfat 分区上使用 cp -a 引发的权限报错

症状

- 出现大量 “无法保留所有者: 不允许的操作”、“Operation not permitted” 等提示。

原因

- vfat 文件系统不支持 Linux 的 UID/GID 和完整权限语义；
- `cp -a` 尝试保留所有者和权限，导致操作失败。

规避

- 对 `boot` 分区仅使用 `cp -v` 复制文件；
- `rootfs` 为 ext4 时，可按需使用 `cp -a`，但一般 `cp -v` 即可满足需求。

8.4 关于 “32 位 Linux + 64 位 ATF” 的疑惑

说明

- ARMv8 SoC 允许在 EL3 / Secure World 运行 AArch64 固件（如 ATF），同时在 Normal World 运行 32 位 Linux；
- 本项目采用的组合为：
  - ATF：AArch64；
  - OP-TEE Core、Linux 与用户态组件：AArch32。

结论

- 这种组合在工程实践中十分常见，并且完全合理；
- 对开发者而言最关键的是：**ATF 使用 AArch64 工具链，其余全部使用 AArch32 工具链**，保持一致即可。

9 快速操作清单（TL;DR）

如果已对上述内容较为熟悉，可直接按本节的步骤进行 “流水线式” 操作。

9.1 宿主机：导入镜像与准备目录

```
docker load -i bitdb-optee-dev.tar
mkdir -p ~/bitdb-optee-dev

# 将 bitdb 源码置于 ~/bitdb-optee-dev/bitdb
# 将交叉工具链压缩包解压至 ~/bitdb-optee-dev/toolchains

# 准备 Raspbian 基础镜像：
# ~/Downloads/2019-06-20-raspbian-buster.img
```

9.2 宿主机：首次刷写 Raspbian 基础镜像

```
lsblk # 假设 TF 卡为 /dev/sdb

sudo umount /dev/sdb1 2>/dev/null || true
sudo umount /dev/sdb2 2>/dev/null || true

sudo dd if=~/Downloads/2019-06-20-raspbian-buster.img \
       of=/dev/sdb \
       bs=4M conv=fsync status=progress

sync
# 可拔出再插入一次 TF 卡
```

9.3 宿主机：启动容器

```
cd ~/bitdb-optee-dev

docker run --rm -it \
  -v "$PWD/bitdb:/bitdb" \
  -v "$PWD/toolchains:/opt/toolchains" \
  bitdb-optee-dev:latest \
  /bin/bash
```

9.4 容器内：构建全部组件

```
cd /bitdb
JOBS=$(nproc 2>/dev/null || echo 4)

make -j"$JOBS" arm-tf-final
make -j"$JOBS"
```

9.5 容器内：确认并覆盖 optee_client 与 BiTDB 示例

```
# 检查 optee_client
cd /bitdb/optee_client
file out/tee-supplicant/tee-supplicant
file out/libteec/libteec.so.1.0

# 检查 BiTDB host
cd /bitdb/optee_examples/bitdb/host
file optee_example_bitdb

# 将确认无误的 ARM 版本覆盖到 out/rootfs
cd /bitdb
mkdir -p out/rootfs/usr/local/bin out/rootfs/usr/local/lib \
        out/rootfs/lib/optee_armtz out/rootfs/usr/local/lib/optee_armtz

cp -v optee_client/out/tee-supplicant/tee-supplicant \
     out/rootfs/usr/local/bin/
cp -v optee_client/out/libteec/libteec.so* \
     out/rootfs/usr/local/lib/
cp -v optee_examples/bitdb/host/optee_example_bitdb \
     out/rootfs/usr/local/bin/
cp -v optee_examples/bitdb/ta/*.ta \
     out/rootfs/lib/optee_armtz/
cp -v optee_examples/bitdb/ta/*.ta \
     out/rootfs/usr/local/lib/optee_armtz/
```

9.6 宿主机：挂载并覆盖 TF 卡

```
lsblk # 找出 /dev/sdb1 /dev/sdb2

sudo mkdir -p /media/$USER/boot /media/$USER/rootfs
sudo mount /dev/sdb1 /media/$USER/boot
sudo mount /dev/sdb2 /media/$USER/rootfs

cd ~/bitdb-optee-dev/bitdb

# 覆盖 boot
sudo cp -v out/boot/armstub8.bin \
          out/boot/bcm2710-rpi-3-b*.dtb \
          out/boot/config.txt \
          out/boot/u-boot.bin \
          out/boot/uboot.env \
          out/boot/uImage \
          /media/$USER/boot/

# 覆盖 rootfs 关键组件
sudo cp -v out/rootfs/usr/local/bin/tee-supplicant \
         /media/$USER/rootfs/usr/local/bin/
```

```
sudo cp -v out/rootfs/usr/local/bin/optee_example_bitdb \
        /media/$USER/rootfs/usr/local/bin/
sudo cp -v out/rootfs/usr/local/lib/libteec.so* \
        /media/$USER/rootfs/usr/local/lib/
sudo cp -v out/rootfs/lib/optee_armtz/*.ta \
        /media/$USER/rootfs/lib/optee_armtz/
sudo cp -v out/rootfs/usr/local/lib/optee_armtz/*.ta \
        /media/$USER/rootfs/usr/local/lib/optee_armtz/

sync
sudo umount /media/$USER/boot
sudo umount /media/$USER/rootfs
```

9.7 树莓派：验证运行

```
dmesg | grep -i tee
ls /dev/tee*

sudo /usr/local/bin/tee-supplicant -d &
sudo /usr/local/bin/optee_example_bitdb
```

若 BiTDB 示例能够正常运行，即表明：

- 容器交叉编译环境配置正确；
- 关键可执行文件与共享库架构均为 ARM 32 位；
- ATF + OP-TEE OS + Linux 内核 + 用户态栈整体打通。