



How to configure OP-TEE



Contents

1. How to configure OP-TEE	3
2. Cross-compile with OpenSTLinux SDK	13
3. How to use USB mass storage in U-Boot	23
4. OP-TEE overview	33
5. STM32CubeProgrammer	43
6. STM32MP1 Developer Package	53
7. STM32MP1 Distribution Package	63
8. STM32MP15 secure boot	73
9. U-Boot overview	83



A quality version of this page, approved on *10 June 2020*, was based off this revision.

Contents

1 Purpose	4
2 Overview	5
3 Build with the Distribution Package	6
4 Build with the Developer Package or a Bare Environment	7
4.1 Initialize the cross compile environment	7
4.2 Build OP-TEE OS	7
4.2.1 Developer Package SDK	7
4.2.2 Bare Environment	8
4.2.3 Generated Files	8
4.2.4 Details on build directives	8
4.2.5 Troubleshoot	8
4.3 Build commands for other OP-TEE components	9
4.3.1 Build the secure components	9
4.3.2 Build the non-secure components	9
5 Update software on board	11
5.1 Update OP-TEE Core Boot Partitions in an SD card	11
5.2 Update OP-TEE Linux Files in a SD card	11
5.3 Update via USB mass storage on U-boot	12
5.4 Update your boot device (including SD card on the target)	12
6 References	13



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target. The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```
PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                # OP-TEE test suite (optional)
PC $> bitbake optee-examples           # TA and CA examples
```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```
meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*
```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```

PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>

```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```

PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-

```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```

PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts

```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The [Developer Package](#) toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-cflagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3      (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5      (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4      (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6      (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10     (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 17.11.2021 - 16:11 / Revision: 16.11.2021 - 11:23

Contents

1 Purpose	14
2 Overview	15
3 Build with the Distribution Package	16
4 Build with the Developer Package or a Bare Environment	17
4.1 Initialize the cross compile environment	17
4.2 Build OP-TEE OS	17
4.2.1 Developer Package SDK	17
4.2.2 Bare Environment	18
4.2.3 Generated Files	18
4.2.4 Details on build directives	18
4.2.5 Troubleshoot	18
4.3 Build commands for other OP-TEE components	19
4.3.1 Build the secure components	19
4.3.2 Build the non-secure components	19
5 Update software on board	21
5.1 Update OP-TEE Core Boot Partitions in an SD card	21
5.2 Update OP-TEE Linux Files in a SD card	21
5.3 Update via USB mass storage on U-boot	22
5.4 Update your boot device (including SD card on the target)	22
6 References	23



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target.

The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```
PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                 # OP-TEE test suite (optional)
PC $> bitbake optee-examples            # TA and CA examples
```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```
meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*
```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```
PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>
```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```
PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-
```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```
PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts
```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The [Developer Package](#) toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3       (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5       (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4       (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6       (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10     (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 22.03.2021 - 14:32 / Revision: 16.02.2021 - 14:56

Contents

1 Purpose	24
2 Overview	25
3 Build with the Distribution Package	26
4 Build with the Developer Package or a Bare Environment	27
4.1 Initialize the cross compile environment	27
4.2 Build OP-TEE OS	27
4.2.1 Developer Package SDK	27
4.2.2 Bare Environment	28
4.2.3 Generated Files	28
4.2.4 Details on build directives	28
4.2.5 Troubleshoot	28
4.3 Build commands for other OP-TEE components	29
4.3.1 Build the secure components	29
4.3.2 Build the non-secure components	29
5 Update software on board	31
5.1 Update OP-TEE Core Boot Partitions in an SD card	31
5.2 Update OP-TEE Linux Files in a SD card	31
5.3 Update via USB mass storage on U-boot	32
5.4 Update your boot device (including SD card on the target)	32
6 References	33



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target.

The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```
PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                # OP-TEE test suite (optional)
PC $> bitbake optee-examples            # TA and CA examples
```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```
meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*
```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```

PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>

```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```

PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-

```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```

PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts

```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The *Developer Package* toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3      (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5      (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4      (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6      (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10     (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 13.05.2020 - 08:56 / Revision: 13.05.2020 - 08:54

Contents

1 Purpose	34
2 Overview	35
3 Build with the Distribution Package	36
4 Build with the Developer Package or a Bare Environment	37
4.1 Initialize the cross compile environment	37
4.2 Build OP-TEE OS	37
4.2.1 Developer Package SDK	37
4.2.2 Bare Environment	38
4.2.3 Generated Files	38
4.2.4 Details on build directives	38
4.2.5 Troubleshoot	38
4.3 Build commands for other OP-TEE components	39
4.3.1 Build the secure components	39
4.3.2 Build the non-secure components	39
5 Update software on board	41
5.1 Update OP-TEE Core Boot Partitions in an SD card	41
5.2 Update OP-TEE Linux Files in a SD card	41
5.3 Update via USB mass storage on U-boot	42
5.4 Update your boot device (including SD card on the target)	42
6 References	43



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target. The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```

PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                # OP-TEE test suite (optional)
PC $> bitbake optee-examples           # TA and CA examples
  
```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```

meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*
  
```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```

PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>

```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```

PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-

```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```

PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts

```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The [Developer Package](#) toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-suppllicant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-suppllicant target/sbin/tee-suppllicant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-suppllicant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-suppllicant -> ../bin/tee-suppllicant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3       (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5       (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4       (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6       (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10     (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 26.11.2021 - 14:35 / Revision: 16.11.2021 - 08:35

Contents

1 Purpose	44
2 Overview	45
3 Build with the Distribution Package	46
4 Build with the Developer Package or a Bare Environment	47
4.1 Initialize the cross compile environment	47
4.2 Build OP-TEE OS	47
4.2.1 Developer Package SDK	47
4.2.2 Bare Environment	48
4.2.3 Generated Files	48
4.2.4 Details on build directives	48
4.2.5 Troubleshoot	48
4.3 Build commands for other OP-TEE components	49
4.3.1 Build the secure components	49
4.3.2 Build the non-secure components	49
5 Update software on board	51
5.1 Update OP-TEE Core Boot Partitions in an SD card	51
5.2 Update OP-TEE Linux Files in a SD card	51
5.3 Update via USB mass storage on U-boot	52
5.4 Update your boot device (including SD card on the target)	52
6 References	53



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target.

The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```

PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                # OP-TEE test suite (optional)
PC $> bitbake optee-examples           # TA and CA examples

```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```

meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*

```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```
PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>
```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```
PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-
```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```
PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts
```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The *Developer Package* toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-cflagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted rootfs.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3      (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5      (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4      (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6      (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10    (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 17.11.2021 - 16:14 / Revision: 16.11.2021 - 17:54

Contents

1 Purpose	54
2 Overview	55
3 Build with the Distribution Package	56
4 Build with the Developer Package or a Bare Environment	57
4.1 Initialize the cross compile environment	57
4.2 Build OP-TEE OS	57
4.2.1 Developer Package SDK	57
4.2.2 Bare Environment	58
4.2.3 Generated Files	58
4.2.4 Details on build directives	58
4.2.5 Troubleshoot	58
4.3 Build commands for other OP-TEE components	59
4.3.1 Build the secure components	59
4.3.2 Build the non-secure components	59
5 Update software on board	61
5.1 Update OP-TEE Core Boot Partitions in an SD card	61
5.2 Update OP-TEE Linux Files in a SD card	61
5.3 Update via USB mass storage on U-boot	62
5.4 Update your boot device (including SD card on the target)	62
6 References	63



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target.

The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```
PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                 # OP-TEE test suite (optional)
PC $> bitbake optee-examples            # TA and CA examples
```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```
meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*
```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```

PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>

```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```

PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-

```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```

PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts

```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The [Developer Package](#) toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3       (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5       (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4       (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6       (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10     (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 17.11.2021 - 16:16 / Revision: 09.11.2021 - 11:24

Contents

1 Purpose	64
2 Overview	65
3 Build with the Distribution Package	66
4 Build with the Developer Package or a Bare Environment	67
4.1 Initialize the cross compile environment	67
4.2 Build OP-TEE OS	67
4.2.1 Developer Package SDK	67
4.2.2 Bare Environment	68
4.2.3 Generated Files	68
4.2.4 Details on build directives	68
4.2.5 Troubleshoot	68
4.3 Build commands for other OP-TEE components	69
4.3.1 Build the secure components	69
4.3.2 Build the non-secure components	69
5 Update software on board	71
5.1 Update OP-TEE Core Boot Partitions in an SD card	71
5.2 Update OP-TEE Linux Files in a SD card	71
5.3 Update via USB mass storage on U-boot	72
5.4 Update your boot device (including SD card on the target)	72
6 References	73



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target.

The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm®v7-A and Arm®v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```
PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                # OP-TEE test suite (optional)
PC $> bitbake optee-examples           # TA and CA examples
```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```
meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*
```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```
PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>
```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```
PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-
```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```
PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts
```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The *Developer Package* toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-suppllicant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-suppllicant target/sbin/tee-suppllicant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-suppllicant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-suppllicant -> ../bin/tee-suppllicant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3      (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5      (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4      (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6      (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10    (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 02.04.2021 - 09:52 / Revision: 02.04.2021 - 09:49

Contents

1 Purpose	74
2 Overview	75
3 Build with the Distribution Package	76
4 Build with the Developer Package or a Bare Environment	77
4.1 Initialize the cross compile environment	77
4.2 Build OP-TEE OS	77
4.2.1 Developer Package SDK	77
4.2.2 Bare Environment	78
4.2.3 Generated Files	78
4.2.4 Details on build directives	78
4.2.5 Troubleshoot	78
4.3 Build commands for other OP-TEE components	79
4.3.1 Build the secure components	79
4.3.2 Build the non-secure components	79
5 Update software on board	81
5.1 Update OP-TEE Core Boot Partitions in an SD card	81
5.2 Update OP-TEE Linux Files in a SD card	81
5.3 Update via USB mass storage on U-boot	82
5.4 Update your boot device (including SD card on the target)	82
6 References	83



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target. The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The Distribution Package provides means to build the following OP-TEE components from their related bitbake target:

```

PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client             # OP-TEE client
PC $> bitbake optee-test               # OP-TEE test suite (optional)
PC $> bitbake optee-examples           # TA and CA examples

```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```

meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*

```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```
PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>
```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```
PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-
```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```
PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts
```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The *Developer Package* toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```

PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>

```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```

PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install

```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```

PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant

```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```

PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant

```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3       (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5       (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4       (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6       (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10     (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8    (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the `STM32CubeProgrammer` documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 19.10.2021 - 13:54 / Revision: 19.10.2021 - 13:54

Contents

1 Purpose	84
2 Overview	85
3 Build with the Distribution Package	86
4 Build with the Developer Package or a Bare Environment	87
4.1 Initialize the cross compile environment	87
4.2 Build OP-TEE OS	87
4.2.1 Developer Package SDK	87
4.2.2 Bare Environment	88
4.2.3 Generated Files	88
4.2.4 Details on build directives	88
4.2.5 Troubleshoot	88
4.3 Build commands for other OP-TEE components	89
4.3.1 Build the secure components	89
4.3.2 Build the non-secure components	89
5 Update software on board	91
5.1 Update OP-TEE Core Boot Partitions in an SD card	91
5.2 Update OP-TEE Linux Files in a SD card	91
5.3 Update via USB mass storage on U-boot	92
5.4 Update your boot device (including SD card on the target)	92
6 References	93



1 Purpose

This article describes the process used for building several OP-TEE components from sources and deploying them the target.

The build example is based on the OpenSTLinux [Developer Package](#) or [Distribution Package](#), and also presents build instructions for a bare environment.



2 Overview

OP-TEE is a trusted execution environment for Arm[®]v7-A and Arm[®]v8-A platforms. OP-TEE is made of several components described in [OP-TEE architecture overview](#).

OP-TEE components generate boot images and files stored in the filesystem embedded in the target.

- OP-TEE OS generates 3 boot image files to be loaded in the platform boot media, in the predefined partitions. The generated boot images include a [STM32 binary header](#) enabling the use of the authenticated boot and flash programming facilities.
- OP-TEE client (package `optee_client`) can be built to generate non-secure services for the OP-TEE OS. The files generated from `optee_client` build are stored in the embedded filesystem.
- OP-TEE project releases other packages intended for test and demonstration. These can be built and embedded in the target filesystem. Building `optee_examples` and `optee_test` generates client and trusted applications together with libraries which are all stored in the target filesystem. Note the OP-TEE Linux driver is built into the Linux kernel image and is part of the OP-TEE ecosystem.

OP-TEE can be embedded in the STM32MP1 platform for the ST trusted configuration.



3 Build with the Distribution Package

The **Distribution Package** provides means to build the following OP-TEE components from their related bitbake target:

```

PC $> bitbake optee-os-stm32mp           # OP-TEE core firmware
PC $> bitbake optee-os-sdk-stm32mp      # OP-TEE development kit for Trusted
Applications
PC $> bitbake optee-client              # OP-TEE client
PC $> bitbake optee-test                # OP-TEE test suite (optional)
PC $> bitbake optee-examples           # TA and CA examples

```

Distribution Package build process includes fetching the source files, compiling them and installing them to the target images.

The Yocto recipes for the OP-TEE packages can be found in:

```

meta-st/meta-st-stm32mp/recipes-security/optee/optee-os-stm32mp*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-client*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-examples*
meta-st/meta-st-openstlinux/recipes-security/optee/optee-test*

```



4 Build with the Developer Package or a Bare Environment

Both [Developer Package](#) and bare build environments expect you to fetch/download the OP-TEE package source file trees in order to build the embedded binary images.

The instruction set below assumes all OP-TEE package source trees are available in the base directory referred as <sources>/. The source files are available from the github repositories:

```
PC $> cd <sources>/
PC $> git clone https://github.com/STMicroelectronics/optee_os.git
PC $> git clone https://github.com/OP-TEE/optee_client.git
PC $> git clone https://github.com/OP-TEE/optee_test.git
PC $> git clone https://github.com/linaro-swg/optee_examples.git
PC $> ls -l <sources>/
optee_client
optee_examples
optee_os
optee_test
PC $>
```

Warning

The STM32MP1 platform is not yet fully merged in the official OP-TEE repository ^[1] hence the URL provided above refers to the ST distribution ^[2]

4.1 Initialize the cross compile environment

The compilation toolchain provided by the [Developer Package](#) can be used, refer to [Setup Cross Compile Environment](#).

Alternatively other bare toolchains can be used to build the OP-TEE **secure** parts. In such case, the instructions below expect the toolchain to be part of the **PATH** and its prefix is defined by **CROSS_COMPILE**. One can use something like:

```
PC $> export PATH=<path-to-toolchain>:$PATH
PC $> export CROSS_COMPILE=<toolchain-prefix>-
```

4.2 Build OP-TEE OS

4.2.1 Developer Package SDK

The OP-TEE OS can be built from the [Developer Package](#) **Makefile.sdk** script that is present in the tarball. It automatically sets the proper configuration for the OP-TEE OS build. To build from shell command:

```
PC $> make -f Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=<board_dts_file_name>.dts
```



4.2.2 Bare Environment

Alternatively one can also build OP-TEE OS based a bare cross compilation toolchains, for example for the stm32mp157c-ev1 board:

```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 0=out all
```

4.2.3 Generated Files

The 3 OP-TEE boot images are generated at following paths:

```
<optee-os>/out/core/tee-header_v2.stm32
<optee-os>/out/core/tee-pageable_v2.stm32
<optee-os>/out/core/tee-pager_v2.stm32
```

One can get the configuration directives used for the build are available in this file:

```
<optee-os>/out/conf.mk
```

The build also generates a development kit used to build Trusted Application binaries:

```
<optee-os>/out/export-ta_arm32/
```

4.2.4 Details on build directives

Mandatory directives to build OP-TEE OS:

- **PLATFORM=stm32mp1**: builds an stm32mp1 platform
- **CFG_EMBED_DTB_SOURCE_FILE=<device-tree-source-file>**: in-tree (core/arch/arm/dts/) device tree filename with its **.dts** extension.

Common optional directives:

- **CFG_TEE_CORE_DEBUG={n|y}**: disable/enable debug support
- **CFG_TEE_CORE_LOG_LEVEL={0|1|2|3|4}**: define the trace level (0: no trace, 4: overflow of traces)
- **CFG_UNWIND={n|y}**: disable/enable stack unwind support

Note: internal memory size constrains the debug support level that can be provided.

4.2.5 Troubleshoot

The *Developer Package* toolchain may report dependency error in the traces such as:

```
PC $> make PLATFORM=stm32mp1 ...
arm-openstlinux_weston-linux-gnueabi-ld.bfd: cannot find libgcc.a: No such file or
directory
```

To overcome the issue, add the directive **comp-flagscore=--sysroot=\$SDKTARGETSYSROOT**. I.e:



```
PC $> cd <optee-os>
PC $> make PLATFORM=stm32mp1 \
           CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts \
           CFG_TEE_CORE_LOG_LEVEL=2 \
           comp-cflagscore=-sysroot=$SDKTARGETSYSROOT \
           O=out all
```

4.3 Build commands for other OP-TEE components

This section describes how the several OP-TEE components (excluding OP-TEE OS described in above section) can be built. All those components generate files targeting the embedded Linux OS based filesystem (i.e the rootfs). These files are the secure Trusted Applications (TAs) binaries as well as non-secure Client Applications (CAs), libraries and test files.

There are several ways to build the OP-TEE components. The examples given below refer to OP-TEE client, test and examples source file tree paths as <optee-client>, <optee-test> and <optee-examples>.

Building these components expect, at least for the trusted applications, that the OP-TEE OS was built and the generated TA development kit is available at <optee-os>/out/export-ta_arm32/.

It is recommended to use CMake for building the Linux userland part whereas secure world binaries (TAs) must be build from their GNU makefiles as the OP-TEE project has not yet ported the secure world binaries build process over CMake.

4.3.1 Build the secure components

Build the TAs: This step expects OP-TEE OS is built to generate the 32bit TA development kit. Assuming OP-TEE OS was built at path <optee-os>/out, the TA development kit is available from path <optee-os>/out/export-ta_arm32/.

Instructions below build and copy the Trusted Application binaries to a local **.target/** directory that can be used to populate the target filesystem.

```
PC $> export TA_DEV_KIT_DIR=$PWD/optee_os/out/export-ta_arm32
PC $> mkdir -p ./target/lib/optee_armtz
PC $> for f in optee_test/ta/*/Makefile; do \
           make -C `dirname $f` O=out; \
           cp -f `dirname $f`/out/*.ta ./target/lib/optee_armtz; \
       done
```

Content in local directory **.target/** are the TA binary files:

```
PC $> tree target/
target
├── lib
│   └── optee_armtz
│       ├── 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
│       ├── 731e279e-aafb-4575-a771-38caa6f0cca6.ta
│       └── (...)
└── ...
```

These files need to be copied to the the target filesystem.

4.3.2 Build the non-secure components

Download the OP-TEE source files in a base directory and create a **CMakeLists.txt** file in the base directory that lists all package to be built through CMake. For example:



```
PC $> ls
optee_client
optee_examples
optee_os
optee_test
CMakeLists.txt
PC $> cat CMakeLists.txt
add_subdirectory (optee_client)
add_subdirectory (optee_test)
add_subdirectory (optee_examples)
PC $>
```

From base directory, run **cmake** then **make**. The example below also creates the tree file system **./target/** that is populated with files generated that need to be installed in the target file system.

Note this examples also sets the toolchain environment:

```
PC $> cmake -DOPTEE_TEST_SDK=$PWD/optee_os/out/export-ta_arm32 \
            -DCMAKE_INSTALL_PREFIX= -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=y
PC $> make
PC $> make DESTDIR=target install
```

Note the empty **CMAKE_INSTALL_PREFIX** value to get thing installed from root **/**, not from **/usr/**. **DESTDIR=target** makes the embedded files be populated in the local **./target/** directory.

Note also that stm32mp15 expects tool **tee-supplciant** to be located in directory **/usr/bin** whereas CMake installs it in directory **/usr/sbin**. To overcome this issue, one can build a link to the effective location, i.e:

```
PC $> ln -s ../bin/tee-supplciant target/sbin/tee-supplciant
```

Once done, local directory **./target/** contains the files to be copied in the target filesystem.

```
PC $> tree target/
target/
├── bin
│   ├── benchmark
│   ├── optee_example_acipher
│   ├── optee_example_aes
│   ├── optee_example_hello_world
│   ├── optee_example_hotp
│   ├── optee_example_random
│   ├── optee_example_secure_storage
│   ├── tee-supplciant
│   └── xtest
├── include
│   ├── tee_bench.h
│   ├── tee_client_api_extensions.h
│   ├── tee_client_api.h
│   └── teec_trace.h
├── lib
│   ├── libteec.so -> libteec.so.1
│   ├── libteec.so.1 -> libteec.so.1.0.0
│   ├── libteec.so.1.0.0
│   └── optee_armtz
│       └── (...) # This directory was previously filled with TAs
└── sbin
    └── tee-supplciant -> ../bin/tee-supplciant
```



5 Update software on board

The OP-TEE OS boot images shall be loaded into the related partitions of the boot media.
The other OP-TEE images are stored in the target filesystem.

For example, if using an SD card as target boot media, the card can be plugged in its PC card reader and the images copied.
OP-TEE core boot images can be loaded using tool **dd** while other files can be simply copied into the mounted roots.

5.1 Update OP-TEE Core Boot Partitions in an SD card

If booting the target from an SD card, the core OP-TEE firmware can be updated using the tool **dd**.
Plug the SD card into the computer slot reader and copy the binary to the dedicated partition; on an SDCard/USB disk the OP-TEE OS boot partitions are partition 4 to 6.

The target partition is located from the partition labels of the SD card, i.e:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 28 16:35 bootfs -> ../../sdd7      (Linux kernel boot
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl1 -> ../../sdd1      (part#1 is TF-A)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 fsbl2 -> ../../sdd2      (part#2 is TF-A backup)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 rootfs -> ../../sdd9     (Linux kernel root
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 ssbl -> ../../sdd3       (part#3# is U-Boot)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teed -> ../../sdd5       (OP-TEE OS paged data)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teeh -> ../../sdd4       (OP-TEE OS header
image)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 teex -> ../../sdd6       (OP-TEE OS resident
core)
lrwxrwxrwx 1 root root 11 Jan 28 16:35 userfs -> ../../sdd10    (Linux kernel user
filesystem)
lrwxrwxrwx 1 root root 10 Jan 28 16:35 vendorfs -> ../../sdd8   (Linux kernel vendor
filesystem)
```

For the SD card described above, the 3 OP-TEE core images can then be updated with:

```
PC $> dd conv=fdatasync of=/dev/sdd4 if=<optee-os>/out/core/tee-header_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd5 if=<optee-os>/out/core/tee-pageable_v2.stm32
PC $> dd conv=fdatasync of=/dev/sdd6 if=<optee-os>/out/core/tee-pager_v2.stm32
```

5.2 Update OP-TEE Linux Files in a SD card

The OP-TEE files that need to be copied to the target filesystem were installed in a local directory **./target/**.

They can now be copied to the target SD card rootfs partition once the SD card is plugged to the host computer and its filesystems are mounted in the host, i.e

```
PC $> cp -ar target/* /media/$USERNAME/rootfs/
```



5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot and follow the previous sections to load binary files `tee-*_v2.stm32` onto target partitions.

5.4 Update your boot device (including SD card on the target)

Refer to the [STM32CubeProgrammer](#) documentation to update your target.



6 References

- https://github.com/OP-TEE/optee_os
- https://github.com/STMicroelectronics/optee_os

Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))