# Porting Open-TEE to Android and deploying it to devices as a non-root application

Summer internship of 2015 by Pawel Sarbinowski

## Introduction

Open-TEE[1] is a virtual Trusted Execution Environment (TEE) that complies to GlobalPlatform specifications. Open-TEE provides a software runtime where GlobalPlatform-compliant Trusted Applications (TAs) can be run and debugged without the need for special hardware. This, together with the fact that Open-TEE is released under a liberal open source license (Apache License, Version 2.0) makes it a low-cost alternative for the initial stages of TA development. Applications developed using Open-TEE can then be compiled for GlobalPlatform compliant hardware TEEs with no or minor modification. Open-TEE consists of the following components:

- The Manager process multiplexes communication between Client Applications (CAs) and Trusted Applications (TAs). It also provides the backing for TA object storage.
- The Launcher process is responsible for pre-loading library dependencies and launching Tas.
- *Libtee* that provides the API used by CAs to establish communications to TAs.

The goals for the summer internship included:

1. porting Open-TEE to the Android platform, allowing native code CAs to interface with Open-TEE TAs.
2. packaging Open-TEE in an Android library that can be used to deploy Open-TEE without root privileges to off-the-shelf Android 5.x devices.

Porting Open-TEE enables testing of Trusted Applications on the Android platform. In the future, Open-TEE could also be utilized as a fallback TEE in devices that do not provide access to a GlobalPlatform-compliant hardware TEE.

## Deliverables

During this summer internship the following deliverables were produced:

1. A series of patches to Open-TEE that enabled cross-compilation to the CPU architectures supported by Android (arm, arm-v7a, x86). These patches include minor changes to the source code of Open-TEE as well as all the makefiles necessary for the Android build system. These patches have been merged with the Open-TEE upstream in https://github.com/Open-TEE .

   In parallel the Intel Tampere team provided extensive changes to the

code handling the internal communication between the TA manager and libtee in Open-TEE and to the code implementing the TEE secure storage to accommodate restrictions in Android Inter-Process Communication.

2. Documentation added to the official site (http://open-tee.github.io ) describing the process of compiling and deploying Open-TEE for Android.
3. A series of patches to Open-TEE that enable deployment to devices without root privileges. These patches have been merged with the Open-TEE upstream in https://github.com/Open-TEE.
4. Android Studio modules (currently available in https://github.com/Open-TEE/opentee-android) . For bundling Open-TEE with Android applications. The modules contain:
   - **opentee:** This is an android library that packages the Open-TEE compiled files (binaries/libraries) and installs/runs them in a device with application privileges (not root). The necessary files are bundled as assets in the Application. An Android service is included and provides an easy interface for developers to install or run Open-TEE. The service listens for messages from clients that bind to it and executes whatever task it is given sequentially. Multi-threading in the Android Service was avoided because of possible race conditions where a binary might be executed before it is installed in the home directory or before the engine is restarted etc.

     All the files are installed in the home data directory of the app. In android that corresponds to */data/data/<application package name>/opentee/*. Inside that directory three sub-directories are created: *bin/* for the binaries, *ta/* for trusted applications and *tee/* for libraries loaded by the native code dynamically (libManagerApi and libLauncherApi). That directory also keeps the process id file (.pid) of the Open-TEE engine and the socket file that is used for communication between the manager and the client applications. Since all these files are contained inside the Android applications directory and are executed by the applications' user no other permissions are necessary to run Open-TEE.

     All libraries that are dependencies to Open-TEE and are not loaded dynamically are kept in the src/main/libs folder of the module. This is used as the directory where the system linker will search for dependency libraries and is defined by the environmental variable LD_LIBRARY_PATH.

     The module can be bundled along with any Android app that wants to utilize Open-TEE. That app will then bind to the provided Android service and install/run Open-TEE to test any TA on it. Additional functions are also provided for the Application using the service to install TAs in the home directory via byte array streams.
   - **testapp:** A sample Android app that demonstrates the use of the previous module. The sample consists of a single Activity that interfaces with the Open-TEE service to setup Open-TEE.

Further documentation about how each module works and how to use it is available in the README files in the repository.

## Future development

In its current form the project can be used to deploy Open-TEE in a device with Android 5.0+, but there are a few shortcomings. The Client Application must be written in C, as language bindings for other languages are not provided for libtee (nor specified by GlobalPlatform). Additionally the deployment is done in the same application context as the client using Open-TEE (i.e. Open-TEE and the Client App are running as the same Linux user). For a better degree of isolation it would be preferred if the Client Application was running as a different user than Open-TEE, thus taking advantage of the sand-boxing in the Android OS. Both issues could potentially constitute work done during a thesis roughly within the following contexts:

## Java Bindings for Libtee

Using Java also allows for easier integration of the TEE Client Applications with Android Applications that want to utilize TEE functionality. To generate the JNI bindings SWIG[1] could be utilized since it requires fairly minimal instrumentation.

## Fully separated Open-TEE manager that forwards calls to native layer

A proposal to implement this kind of design is to package Open-TEE as a completely separate Android application that will fulfill two functions. The first is to provide an AIDL[2] interface that should be similar to the one libtee offers but adapted to work with Java data types. A simple JNI interface would not be enough since *libtee* uses shared memory to communicate with the TA Manager and shared memory cannot be passed along between two different apps because of the sand-boxing of Android. Data would have to be marshaled from the requests received from the AIDL interface and forwarded to the Open-TEE engine running at a native level. The second function would be to provide multiplexing between Java-level callers, which Open-TEE cannot do by itself due to the marshaling functionality which would be added to the Java-level API.

## References

[1]: http://arxiv.org/abs/1506.07367
[2]: http://www.swig.org
[3]: http://developer.android.com/guide/components/aidl.html

## Appendix

Full instructions on building and deploying Open-TEE on devices:

- Start by following the instructions here to setup your android build environment and download the android source code (e.g. to $HOME/android_source ).
- Add the following to your .bashrc file or to a file you will source on each shell.

```
export ANDROID_ROOT="$HOME/android_source"
export USE_CCACHE=1
export CCACHE_DIR="$HOME/android_source/.ccache"
$ANDROID_ROOT/prebuilts/misc/linux-x86/ccache/ccache -M 50G
source "$HOME/android_source/build/envsetup.sh"
```

- Inside *$ANDROID_ROOT* create a link to the directory you have downloaded Open-TEE to (e.g. *$HOME/Open-TEE* ) like so:

```
ln -s $HOME/Open-TEE $ANDROID_ROOT/Open-TEE
```

- Now to build just run

```
lunch
```

to choose the target and then

```
make clean && make opentee-engine libManagerApi libInternalApi libLauncherApi
libCommonApi libta_conn_test_app conn_test_app libtee
```

to build all the Open-TEE modules. You can find all the available modules by doing `grep -ir "LOCAL_MODULE " Open-TEE/` where Open-TEE/ is the directory containing the Open-TEE source code.

The output files will by default be located in *$ANDROID_ROOT/out/target/product/generic*/* (depending on the architecture). You should also be able to see the output directory path if you do `echo $OUT`.

To deploy those binary files to an Android device you can choose one of two methods:

### ADB (needs root on device)

To copy those files to an android device you can use the script located in *Open-TEE/project/install_android.sh* . The script assumes the adb binary is in your $PATH and that $OUT contains the directory where the binaries were outputted.

*Note: root access on the device is needed for this.*

The files should now be installed on */system/lib/{ta,tee}* and */system/bin* on the device. Do `adb root` to start adb with root privileges and then `adb shell` to get a shell on the device.

**From the adb shell:**

In case the files do not have an execution permission add it with something like:

```
chmod +x /system/bin/opentee-engine
chmod +x /system/bin/conn_test_app
```

And run Open-TEE with

```
/system/bin/opentee-engine
```

Verify that Open-TEE is running with ps:

```
ps | grep tee
```

**Importing and testing a TA via adb**

- Modify *Open-TEE/project/install_android.sh* to also copy your TA .so file in the */system/lib/ta/* directory and your CA to */system/bin/*
- *Run* your CA to test the TA directly via adb shell with */system/bin/<ca_name>* and check logcat or gdb on android for debugging output

## Android Studio

This method uses an Android Studio project that packages Open-TEE inside an application and installs/runs it to the home directory of the app.

- Start by cloning the repo with

```
git clone https://github.com/Open-TEE/opentee-android
cd opentee-android/
```

- Then import all the binaries built to the opentee module (that packages Open-TEE) by using the *opentee/install_opentee_files.sh*. For each architecture compiled (armeabi, armeabi-v7a, x86) you should re-run the install_opentee_files.sh with the appropriate argument (do `./install_opentee_files.sh -h` for help).
- After the import and assumming you have downloaded and installed Android Studio use it to open the opentee-android project (File/Open…). You might need to specify the Android NDK or Android SDK directory in local.properties but the IDE should in most cases detect those by itself.
- You can then build and run the testapp module that is a reference usage implementation and demonstrates how to install/run Open-TEE and other binaries.

**Importing and testing a TA via Android studio**

There are two ways of importing TAs to the Open-TEE android module:

1. By running ./install_opentee_files.sh which will copy your compiled CA and TA to the appropriate directories in the Studio project.
2. Modify OTConstants.java to add the names of your CA/TA
3. Modify the *MSG_INSTALL_ALL* function in OpenTEEService.java to also install your TA to *OTConstants.OPENTEE_TA_DIR* and your CA to *OTConstants.OPENTEE_BIN_DIR*

OR

1. Use only the OpenTEEConnection class and its installByteStreamTA() method to install your TA from any source (file, network etc).
   An example can be seen in MainActivity.java in the opentee-android module (testapp) demonstrating the usage of the service.

In both cases run your CA via an OpenTEEConnection instance. E.g:

mOpenTEEConnection.runOTBinary(OTConstants.STORAGE_TEST_APP_ASSET_BIN_NAME)

### Troubleshooting

If you get errors similar to:

*D/tee_manager(32036): opentee/emulator/opentee-main/main.c:load_lib:166 Failed to load library, /system/lib/libManagerApi.so : dlopen failed: cannot locate symbol "mempcpy" referenced by "libCommonApi.so"...*
*D/tee_launcher(32037): opentee/emulator/opentee-main/main.c:load_lib:166 Failed to load library, /system/lib/libLauncherApi.so : dlopen failed: cannot locate symbol "mempcpy" referenced by "libCommonApi.so"...*

Then most probably the android tree that you are building with does not match the tree on the device and thus you might also have to push the generated libc.so (or other lib*.so files) to the device.

Note: that this is unsafe and might result in your device malfunctioning. It is a good idea to take a backup of the /system/lib/lib*.so files or even a complete ROM backup if you have a custom recovery.