TRUSTONIC

Getting Started with <t-sdk

<t-sdk

PREFACE

VERSION HISTORY

Version	Date	Modification
1.0	2013-02-06	First Issued version
1.1	2013-03-12	Only minor changes
1.2	2013-03-19	Added Appendix with change SUID
1.3	2013-07-10	Merged Arndale and QEMU Getting Started's into one unique document. Restructured document. Added Section for Samples and APK Applications.

TABLE OF CONTENTS

1 INTRODUCTION

This document explains how to run <t-base on the Arndale board or the QEMU emulator.

Arndale uses an ARM Cortex A15 processor with a Trustzone implementation.

QEMU can be used to emulate an ARM processor with a Trustzone implementation.

Therefore a <t-base-OS can be executed on both Arndale and QEMU and the board/emulator can be used to test and develop TAs for the <t-base-OS TEE.

1.1 GLOSSARY AND ABBREVIATIONS

ADB	**A**ndroid **D**ebug **B**ridge is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device.
TA	**T**rusted **A**pplication on the SWd side executing specific security services in the <t-base runtime environment. The security services provided by a Trusted Application are called by NWd client applications. <t-base differentiates between System Trusted Application like the CM Trusted Application and Trusted Applications of Service Providers. Formerly called Trustlet.
TLC	**T**rusted Application **L**ayer **C**onnector. Software running in the Normal-World providing a high level convenience interface to access a Trusted Application in the Secure-World by the Normal-World client.
logcat	The Android logging system provides a mechanism for collecting and viewing system debug output. Logs from various applications and portions of the system are collected in a series of circular buffers, which then can be viewed and filtered by the logcat command. You can use logcat from an ADB shell to view the log messages.
SUID	The SUID is a 16 byte unique identifier of the SoC. It is used to identify a device in the t-base ecosystem.

1.2 <T-SDK PACKAGE STRUCTURE AND SETUP

After uncompressing the archive received from Trustonic the <t-sdk package provides everything you need for Trusted Application (TA) and Trusted Application Layer Connector (TLC) development under Linux:

- Documentation/: Documentation including API references and this guide.
- Samples/: Sample projects to get started quickly
- t-sdk/: Contains header files and libraries to compile TAs and TLCs.
- ./index.html: Documentation entry point to get started using the <t-sdk.
- **setup.sh**: This script is called when building samples.

When receiving the <t-sdk Release Package please edit setup.sh script for specifying tools directories needed for building samples:

- Android NDK and SDK
- Java and Ant applications directories

```
# Android NDK Directory
export NDK_BUILD=

# Android SDK Directory
export ANDROID_HOME=

# Java Home Directory
export JAVA_HOME=

# Ant application for building TSdkSample
export ANT_PATH=
export PATH=${ANT_PATH}:$PATH
```

Depending on the toolchain you want to use, please specify either your GNU or ARM working directory. By default GNU toolchain is used.

If you want to use GNU toolchain, please specify your GCC directory here:

```
# GCC Compiler variables
export CROSS_GCC_PATH=
export CROSS_GCC_PATH_INC=${CROSS_GCC_PATH}/arm-none-eabi/include
export CROSS_GCC_PATH_LIB=${CROSS_GCC_PATH}/lib
export CROSS_GCC_PATH_BIN=${CROSS_GCC_PATH}/bin
```

If you want to use ARM toolchain please specify your ARM compiler path and license file here:

```
# TOOLCHAIN : ARM, GNU
# If variable is not set, use GNU by default
export TOOLCHAIN=ARM
[...]
export ARM_RVCT_PATH=    # ARM DS-5 - e.g.: /usr/local/DS-5
export LM_LICENSE_FILE=  # DS-5 license - e.g.: /home/user/DS5PE-*.dat
```

1.3 SAMPLES AND TOOLS APPLICATIONS

Some applications are provided with the <t-sdk package to help you getting started with the development of Trusted Applications.

1.3.1 Aes Sample

This sample provides an example of AES encryption/decryption operation.

It is preinstalled in QEMU and Arndale packages.

Execution on QEMU:

```
$ cd <your-tsdk-QEMU-package-path>/Tools
$ ./adb_qemu.sh
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleAes -l 32
```

Execution on Arndale:

```
$ adb shell
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleAes -l 32
```

Release mode output(logcat):

```
I/TLC AES ( 5225): Copyright (c) Trustonic Limited 2013
I/TLC AES ( 5225): Success
```

1.3.2 Rot13 Sample

This sample makes a rotation of a string by shifting each character of 13 positions forward.

Execution on QEMU:

```
$ cd <your-tsdk-QEMU-package-path>/Tools
$ ./adb_qemu.sh
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleRot13
Plain text: The quick brown fox jumps over the lazy dog
Cipher text: Gur dhvpx oebja sbk whzcf bire gur ynml qbt
TLC exit code: 00000000
```

Execution on Arndale:

```
$ adb shell
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleRot13
Plain text: The quick brown fox jumps over the lazy dog
Cipher text: Gur dhvpx oebja sbk whzcf bire gur ynml qbt
TLC exit code: 00000000
```

Release mode output(logcat):

```
I/TLC SAMPLE ROT13( 9615): Copyright (c) Trustonic Limited 2013
I/TLC SAMPLE ROT13( 9615): Success
```

1.3.3 Rsa Sample

This sample shows how to generate RSA key pair.

It also maps additional memory to Trusted Application (cf. Developer Guide Section "Memory Mapping").

Execution on QEMU:

```
$ cd <your-tsdk-QEMU-package-path>/Tools
$ ./adb_qemu.sh
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleRsa
```

Execution on Arndale:

```
$ adb shell
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleRsa
```

Release mode output(logcat):

```
I/TLC RSA (10172): Copyright (c) Trustonic Limited 2013
I/TLC RSA (10172): Success
```

1.3.4 Sha256 Sample

This sample shows how to hash some data passed as plain text.

It also maps additional memory to Trusted Application (cf. Developer Guide Section “Memory Mapping”).

Execution on QEMU:

```
$ cd <your-tsdk-QEMU-package-path>/Tools
$ ./adb_qemu.sh
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleSha256
```

Execution on Arndale:

```
$ adb shell
root@android:/ # cd /data/app/Samples
root@android:/ # ./tlcSampleSha256
```

Release mode output(logcat):

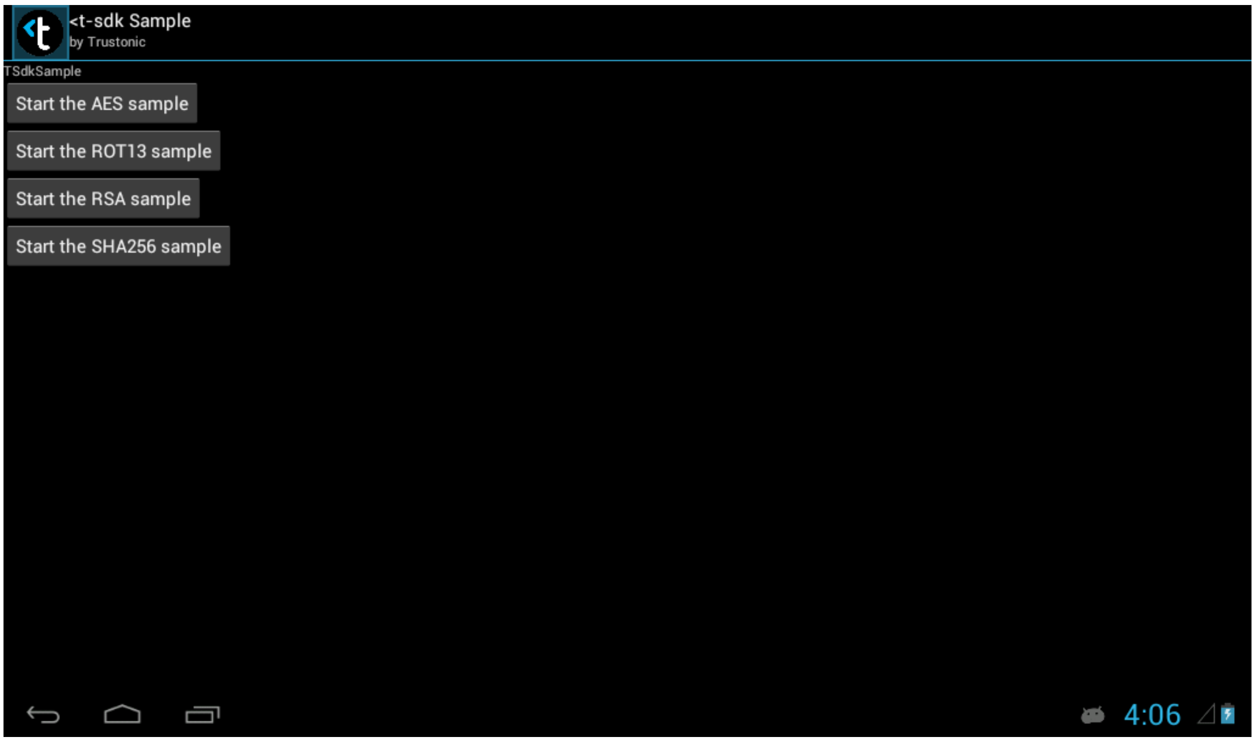
```
I/TLC SAMPLE SHA256(10146): Copyright (c) Trustonic Limited 2013
I/TLC SAMPLE SHA256(10146): Success
```

1.3.5 TSdkSample

This sample is a launcher for executing AES, Rot13, RSA and Sha256 samples from Android graphical environment on Arndale board.

Due to QEMU limitations in terms of performance this application cannot be run on QEMU.

To launch it on Android go to "Applications" menu and click on it. You will get following screen:



1.3.6 TlcInfo and TBaseInfo Tools

1.3.6.1 TlcInfo

TlcInfo is a command line tool that can be used during development/testing.

It is preinstalled in QEMU and Arndale images.

Installation:

```
$ adb push Out/Bin/Debug/tlcInfo /data/app
```

Execution:

```
$ cd <your-tsdk-QEMU-or-Arndale-package-path>/
$ ./adb_qemu.sh
already connected to localhost:5555
root@android:/ #
root@android:/ # /data/app/tlcInfo
```

Output:

```
Copyright (c) Trustonic Limited 2013.
*** jenkins-Generic_integration_build-265 ###, Aug  5 2013, 21:08:34.

mcGetMobiCoreVersion:
productId        = <t-sdk-r4
versionMci       = 0x00000005
versionSo        = 0x00020002
versionMclf      = 0x00020003
versionContainer = 0x00020001
versionMcConfig  = 0x00000001
versionTlApi     = 0x0001000D
versionDrApi     = 0x00010000
versionCmp       = 0x00000000
```

1.3.6.2 TBaseInfo

TBaseInfo is the equivalent of TlcInfo in APK format and can be used on Commercial products with <t-base. It is preinstalled on Arndale board.

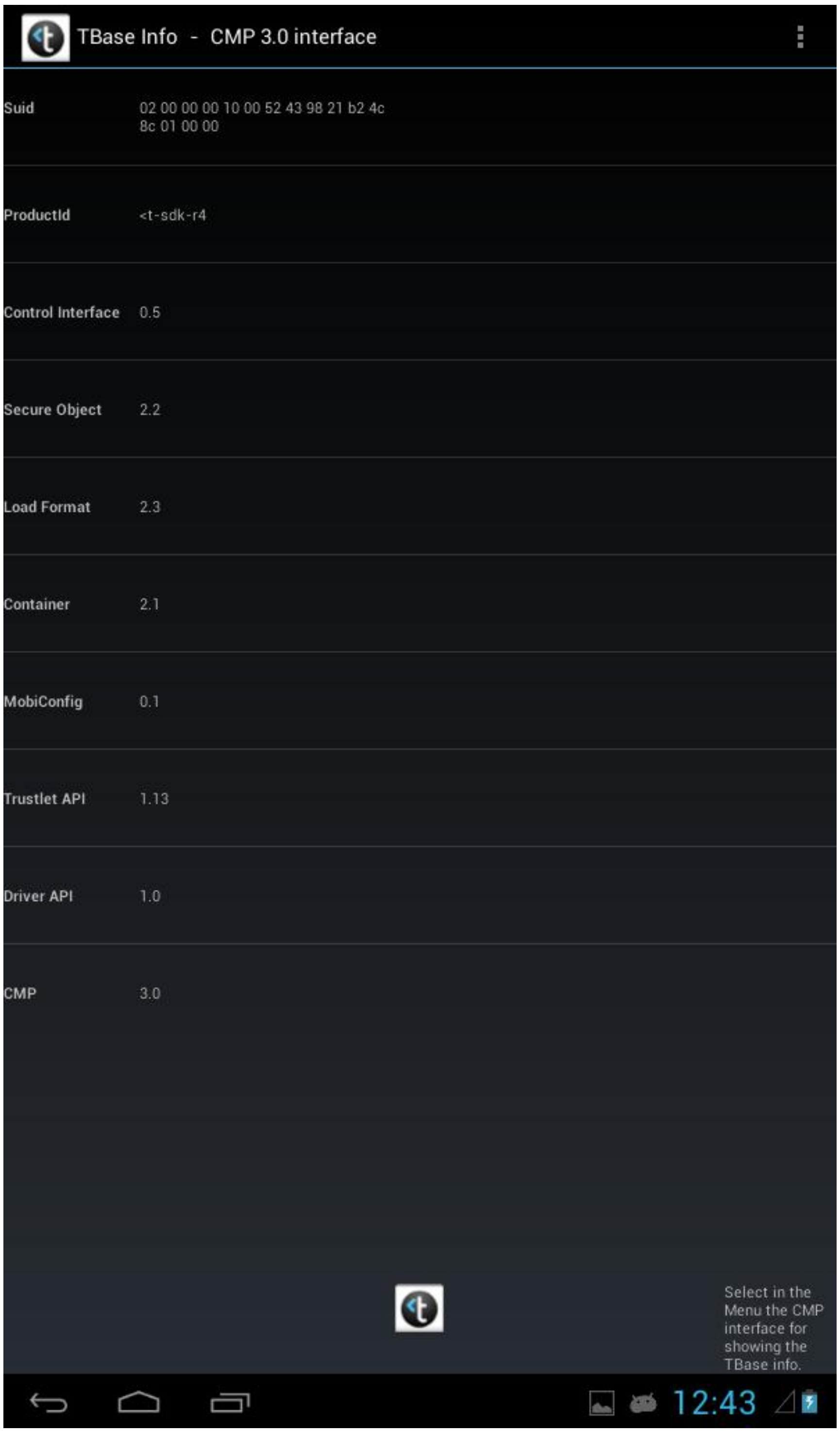
Execution:

TBaseInfo is an Android application. You can start it via a command line or by launching it from the Android Applications Menu.

Command Line execution:

```
$ adb shell am start -a android.intent.action.MAIN -n
com.trustonic.android.tbaseinfo/com.trustonic.android.tbaseinfo.TBaseInfo
```

Output data is the same as TlcInfo as following:



```
Suid               02 00 00 00 10 00 52 43 98 21 b2 4c
                   8c 01 00 00
ProductId          <t-sdk-r4
Control Interface  0.5
Secure Object      2.2
Load Format        2.3
Container          2.1
MobiConfig         0.1
Trustlet API       1.13
Driver API         1.0
CMP                3.0
```

Select in the Menu the CMP interface for showing the TBase info.

1.3.6.3 Output Parameters

Both TlcInfo and TBAseInfo applications give information relative to the components of <t-base environment:

- Suid(TBaseInfo only): SUID used, cf. GLOSSARY AND ABBREVIATIONS
- productId: Name of the <t-sdk release.
- versionMci: Version of the <t-base Interface.
- versionSo: Version of the Secure Object API.
- versionMclf : Version of the <t-base Load Format
- versionContainer : Version of the Container Format.
- versionMcConfig : Version of the <t-base Configuration.
- versionTlApi : Version of the Trusted Application API.
- versionDrApi : Version of the <t-base Driver API.
- versionCmp : Version of the Content Management Protocol.

1.3.7 RootPA

The Root Provisioning Agent is a component running on a device and acts as a remote agent interacting with Trustonic backend.

To install (use debug version, release version needs to be signed by OEM):

```
adb install RootPA-debug.apk
3779 KB/s (592294 bytes in 0.153s)
         pkg: /data/local/tmp/RootPA-debug.apk
Success
```

2 Running Rot13 Sample on QEMU and Board

In this section we will run the Rot 13 sample in order to indicate the procedure to use <t-sdk samples with QEMU or Arndale board.

The Rot13 sample shifts characters of a string of 13 characters ahead in the alphabetic order.

1) Compile the Rot13 sample TA

```
$ cd <your-tsdk-package-path>/Samples/Rot13/TlSampleRot13
$ ./Locals/Build/build.sh
```

Output:

```
ANDROID_HOME=/home/developer1/t-sdk-r4/Tools/adt-bundle-linux-x86-20130522/sdk
JAVA_HOME=/home/developer1/t-sdk-r4/Tools/jdk1.7.0_25
Running make...
******************************************
Trustlet Build
******************************************
******************************************
MODE      : Debug
TOOLCHAIN : GNU
******************************************
mkdir -p Locals/Code/../../Out/Bin/GNU/Debug/obj-local/Locals/Code/
/home/developer1/t-sdk-r4/Tools/gcc-arm-none-eabi-4_7-2013q1/bin/arm-none-eabi-gcc    -
std=c99 -DARCH=__arm32__ -D__arm32__ -mthumb -D__THUMB__ -O3 -Os -fno-builtin -mthumb-
interwork   -fno-short-enums     -DPLAT=ARMV7_A_PB   -DARM_ARCH=ARMv7   -D__ARMv7__   -
D__ARMV7_A__  -D__ARMV7_A_PB__  -DARMV7_A_SHAPE=PB   -mcpu=generic-armv7-a  -mfpu=vfp  -
mfloat-abi=softfp    -DDEBUG    --debug    -DTRUSTLET        -ILocals/Code/public    -
I/home/developer1/t-sdk-r4/t-sdk/TlSdk/Public           -I/home/developer1/t-sdk-r4/t-
sdk/TlSdk/Public/MobiCore/inc  -I/home/developer1/t-sdk-r4/Tools/gcc-arm-none-eabi-4_7-
2013q1/arm-none-eabi/include    -c    -o    Locals/Code/../../Out/Bin/GNU/Debug/obj-
local/Locals/Code/tlSampleRot13.o Locals/Code/tlSampleRot13.c
[…]
******************************************
** READELF & MobiConvert *****************
******************************************
/home/developer1/t-sdk-r4/Tools/gcc-arm-none-eabi-4_7-2013q1/bin/arm-none-eabi-readelf
-a                Locals/Code/../../Out/Bin/GNU/Debug/tlSampleRot13.axf                >
Locals/Code/../../Out/Bin/GNU/Debug/tlSampleRot13.lst2
java   -jar   /home/developer1/t-sdk-r4/t-sdk/TlSdk/Bin/MobiConvert/MobiConvert.jar   -
servicetype  2  -numberofthreads  1  -numberofinstances  1  -memtype  2  -flags  0  -bin
Locals/Code/../../Out/Bin/GNU/Debug/tlSampleRot13.axf                            -output
Locals/Code/../../Out/Bin/GNU/Debug/04010000000000000000000000000000.tlbin    -keyfile
Locals/Build/keySpTl.xml
>Locals/Code/../../Out/Bin/GNU/Debug/04010000000000000000000000000000.tlbin.log
```

2) Compile the Rot13 sample TLC

```
$ cd <your-tsdk-package-path>/Samples/Rot13/TlcSampleRot13
$ ./Locals/Build/build.sh
```

Output:

```
Mode          : Debug
Refresh Sources:
ANDROID_HOME=/home/developer1/t-sdk-r4/tools/adt-bundle-linux-x86-20130522/sdk
JAVA_HOME=/home/developer1/t-sdk-r4/tools/jdk1.7.0_25
Gdbserver       : [arm-linux-androideabi-4.6] libs/armeabi/gdbserver
Gdbsetup        : libs/armeabi/gdb.setup
Compile++ thumb: TlcSampleRot13 <= client-jni.cpp
Prebuilt : libMcClient.so <= /home/developer1/t-sdk-r4/t-sdk/TlcSdk/Bin/Generic/Debug/
Compile++ thumb: tlcSampleRot13Lib <= tlcSampleRot13.cpp
StaticLibrary   : libtlcSampleRot13Lib.a
SharedLibrary   : libTlcSampleRot13.so
Install         : libTlcSampleRot13.so => libs/armeabi/libTlcSampleRot13.so
Compile++ thumb   : tlcSampleRot13 <= main.cpp
Executable      : tlcSampleRot13
Install         : tlcSampleRot13 => libs/armeabi/tlcSampleRot13
Install         : libMcClient.so => libs/armeabi/libMcClient.so
```

3) Launch QEMU or power up the board:

For QEMU, run the **run_qemu.sh** script from t-sdk-qemu package delivered by Trustonic:

```
$ cd <your-tsdk-QEMU-package-path>/tools
$ ./run_qemu.sh
```

Note : <t-base will be automatically started

4) Connect to the emulator/board:

For QEMU, execute the **adb_qemu.sh** script:

```
$ cd <your-tsdk-QEMU-package-path>/tools
$ ./adb_qemu.sh
```

For Arndale board, execute the **adb_arndale.sh** script:

```
$ adb connect
$ ./adb_arndale.sh
```

5) Push binaries to the emulator/board from your workstation:

```
$ cd <your-tsdk-package-path>/Samples/Rot13
$ adb push TlcSampleRot13/Out/Bin/Debug/tlcSampleRot13 /data/app/Samples/
$ adb push
TlcSampleRot13/Out/Bin/Debug/tlcSampleRot13/04010000000000000000000000000000.tlbin
/data/app/Samples/
```

6) Executing Rot13 sample:

```
$ adb shell
root@android:/ # cd /data/app/Samples
root@android:/data/app/Samples # ./tlcSampleRot13
Plain text: The quick brown fox jumps over the lazy dog
Cipher text: Gur dhvpx oebja sbk whzcf bire gur ynml qbt
TLC exit code: 00000000
```

7) Getting Rot13 execution traces by using logcat:

   Use following command to get log after the execution of sample Rot13 (or any sample execution):

```
$ adb logcat
--------- beginning of /dev/log/main
I/TLC SAMPLE ROT13( 3331): Success
```

Use the following command to clean system debug traces between 2 consecutive executions:

```
 $ adb logcat -c
```

The same steps can be applied to execute other samples of the <t-sdk: Aes, Rsa and Sha256.

3 Building a new Trusted Application

You can start your own development by starting from Rot13 TA and TLC.

After an eventual setup of your environment (in case you want to use your own toolchain as described in 0) you are ready for building your first Trusted Application (TA).

Now you have to apply the following steps:

1. Update the demo makefile.mk for your own TA:

```
# output binary name without path or extension
OUTPUT_NAME := <yourTrustedApplicationName>
```

2. Execute the build script:

```
$ ./build.sh
```

When a TA is built using this command, it will use the default options and will compile a TA in Debug with the GNU compiler.

It is possible to specify which MODE and TOOLCHAIN will be used to build the TA:

`MODE`	:	Debug or Release
`TOOLCHAIN`	:	ARM or GNU

The script should be called this way:

```
$ MODE=Debug   TOOLCHAIN=ARM ./build.sh      #default values
$ MODE=Release TOOLCHAIN=GNU ./build.sh
```

3. Push the TA you built to your Emulator/Arndale Board (once running):

```
$ adb connect localhost:5555
$ adb push
<yourTrustedApplicationDirectory>/<yourTrustedApplicationUUID>.tlbin
/data/app/t-base/mcRegistry
```

For example:

```
$ adb push
/data/app/Samples/Aes/tlAes/Out/Bin/GNU/Debug/
07020000000000000000000000000000.tlbin /data/app/t-base/
```

4 Building a new Trusted Application Layer Connector

After an eventual setup of your environment (in case you want to use your own toolchain as described in 0) you are ready for building your first Trusted Application Layer Connector (TLC).

Now you have to apply the following steps:

1. Update the demo Android.mk and build.sh for your own TLC
2. To get the TLC build running, you have to export an additional path to the output folder of your TA:

```
Export COMP_PATH_<yourTAName>= ${COMP_PATH_ROOT}/<yourTAdir>/Out
```

3. Call the build script:

```
./build.sh
```

4. Push the TLC you built to your running Emulator/Arndale Board:

```
adb connect localhost:5555
adb push
<yourTrustedApplicationConnectorDirectory>/<yourTrustedApplicationConn
ectorName> /data/app/t-base/mcRegistry
```

For example:

```
adb push
/data/app/Samples/Aes/tlcAes/Out/Bin/Debug/tlcAes /data/app/t-base/
```

5 DEFINING YOUR TRUSTED APPLICATION TYPE

5.1 ON A COMMERCIAL DEVICE (RUNNING AS SERVICE PROVIDER-TRUSTED APPLICATION)

System TAs cannot be loaded on commercial devices for testing; the only way of running your TA is loading it as a SP-TA.

On commercial smartphones and tablets you need connection to the <t-base backend using the Provisioning Agent (PA).

Hint: working with commercial devices usually implies having root access rights i.e. to push files to /data/app/mcRegistry.

Use a SP Test Trusted Application container:

There is a TA existing in Backend that can be used for testing your TA.

1. Use the XML key file that comes with the <t-sdk `(t-sdk-rXXX\Samples\Sha256\TlSampleSha256\ Locals\Build\keySpTl.xml)` it contains the AES key of the 04010200..00 sample SP-TA container in Backend:

```
<?xml version="1.0"?>
<Key>000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d
0e0f</Key>
```

2. Compile your TA as Service Provider with following options (`makefile.mk`)

```
TRUSTLET_UUID := 04010200000000000000000000000000
TRUSTLET_SERVICE_TYPE := 2 # SP TA
TRUSTLET_KEYFILE := <your path>/keySpTl.xml
```

3. Install the Provisioning Agent Android app.

4. A SP-container for your TA ("<UUID>.spcont") is needed on your device in `data/app/mcRegistry/` to run it as SP-TA
   ➔ Trigger the backend to create it:

```
adb shell ls /data/app/mcRegistry
adb shell am start -a "com.secunet.android.t-base.install_sp_tl"
-e "TL_UUID" "04010200-0000-0000-0000-000000000000"
```

5. load both modules (from TA/TLC out-folders) onto your device
   If mcOpenSession() is used by the TLC, the TA must be pushed to /data/app/mcRegistry Otherwise when mcOpenTrustlet() is used by the TLC, the TA must be pushed to the location expected by the TLC.

```
adb push 04010200000000000000000000000000.tlbin
data/app/mcRegistry/ #mcOpenSession() case
adb push <myTLCname> data/app/
```

6. start your TLC

```
adb shell data/app/<myTLCname>
```

5.2 ON A DEVELOPMENT BOARD (RUNNING AS SYSTEM-TA OR SP-TA)

Development boards (Arndale are typically not known to the Backend but 16 SP-TA containers are predefined.

To execute your Trusted Application on a Development Board it has to be compiled either as System Trusted Application and signed by the dummy OEMs RSA private key or as Service Provider Trusted Application and signed with the development key provided in the <t-sdk.

Hint: <t-base images are available for some development platforms / boards. See [DevDev] for details or contact support@trustonic.com .

Use a signed System TA:

1. Use the RSA System TA sample private key that comes with the <t-sdk `(t-sdk-rXXX\Samples\Sha256\TlSampleSha256\ Locals\Build\pairVendorTltSig.pem)`

2. Compile your TA as a System TA (adapt `makefile.mk`)

```
TRUSTLET_UUID := 06010000000000000000000000000000
TRUSTLET_SERVICE_TYPE := 3 # System Trusted Application
TRUSTLET_KEYFILE := <your path>/pairVendorTltSig.pem
```

3. load both modules (from TA/TLC out-folders) onto your device

```
adb push 06010000000000000000000000000000.tlbin data/app/mcRegistry/
adb push <myTLCname> data/app/
```

4. start your TLC

```
adb shell data/app/<myTLCname>
```

Use a signed SP TA: see previous chapter.

6 Using Emulator or Development Board

6.1 QEMU EMULATOR

6.1.1 Installing the static Emulator

The static emulator is already installed in your package and is located in **`qemu/`** folder.

6.1.2 Running the emulator

All you have to do is to run the following script located in the **`tools/`** folder of the QEMU package.

```
$ ./run_qemu.sh
```

This script contains the command line launching the QEMU emulator.

NOTES:

Please wait some a moment until Linux loads!
It's not possible to use CTRL+C in the console, CTRL+C will shut down the emulator completely!
Entering command "stty intr ^p" before starting qemu CTRL+C does not shutdown the emulator (CTRL+P does it)

6.1.3 Using ADB

When QEMU is up and running you can connect to QEMU with ADB (Android Debug Bridge) with the following script located in the **`tools/`** folder of the QEMU package:

```
$ ./adb_qemu.sh
```

The ADB application has been installed previously in the <t-sdk package by the **`setup.sh`** script.

6.2 ARNDALE BOARD

The Arndale boards TRUSTONIC delivers to customers are already set up.

Following steps have to be executed:

- Connect a mini USB cable between your workstation and your Arndale board.
- power on your board
- connect to your board using adb:

```
$ adb devices
List of devices attached
0123456789ABCDEF device
$ adb shell
```

Customers willing to update t-base image or reinstall their board can go to Appendix I.

7 DEBUGGING <T-BASE-OS

7.1 TRACE32 SETUP

Add these lines to your cmm config to be able to connect:

```
System.CPU CORTEXA15
Sys.mc debugaccessport 0x1
Sys.Config.COREBASE 0x80010000
```

7.2 GDB

The emulator has a working GDB stub server via the host's port 1234 that can be enabled by using the parameters below.

Execution stops at the first instruction and then any local variation of `arm-eabi-gdb` or `arm-eabi-gdbtui` can be used, GUI debugging via KDBG and Code Sourcery Code Bench are known to work.

```
 -s -S
```

7.2.1 The parameters explained in detail

QEMU debug parameters (see also http://wiki.qemu.org/download/qemu-doc.html):

```
-S    : Freeze CPU at startup (use 'c' to start execution)

-s    : Start GDB server listening on the host's port 1234
```

7.3 EXECUTION TRACING

In some cases it's quite useful to get a log of all the instructions the emulator executed and the current CPU context.

To do that you have to start the emulator with the additional parameters:

```
-d in_asm,cpu -D ./qemu.log -singlestep
```

This instructs the emulator to create qemu.log with debug information about every instruction it ran and also the CPU context for every instruction.

NOTE:

Running with full logging enabled will slow down your system and the size of qemu.log will end up in the GB range!

7.3.1 The parameters explained in detail

`-d in_asm,cpu`

log every instruction and the CPU context for every instruction

`-D ./qemu.log`

write log to qemu.log

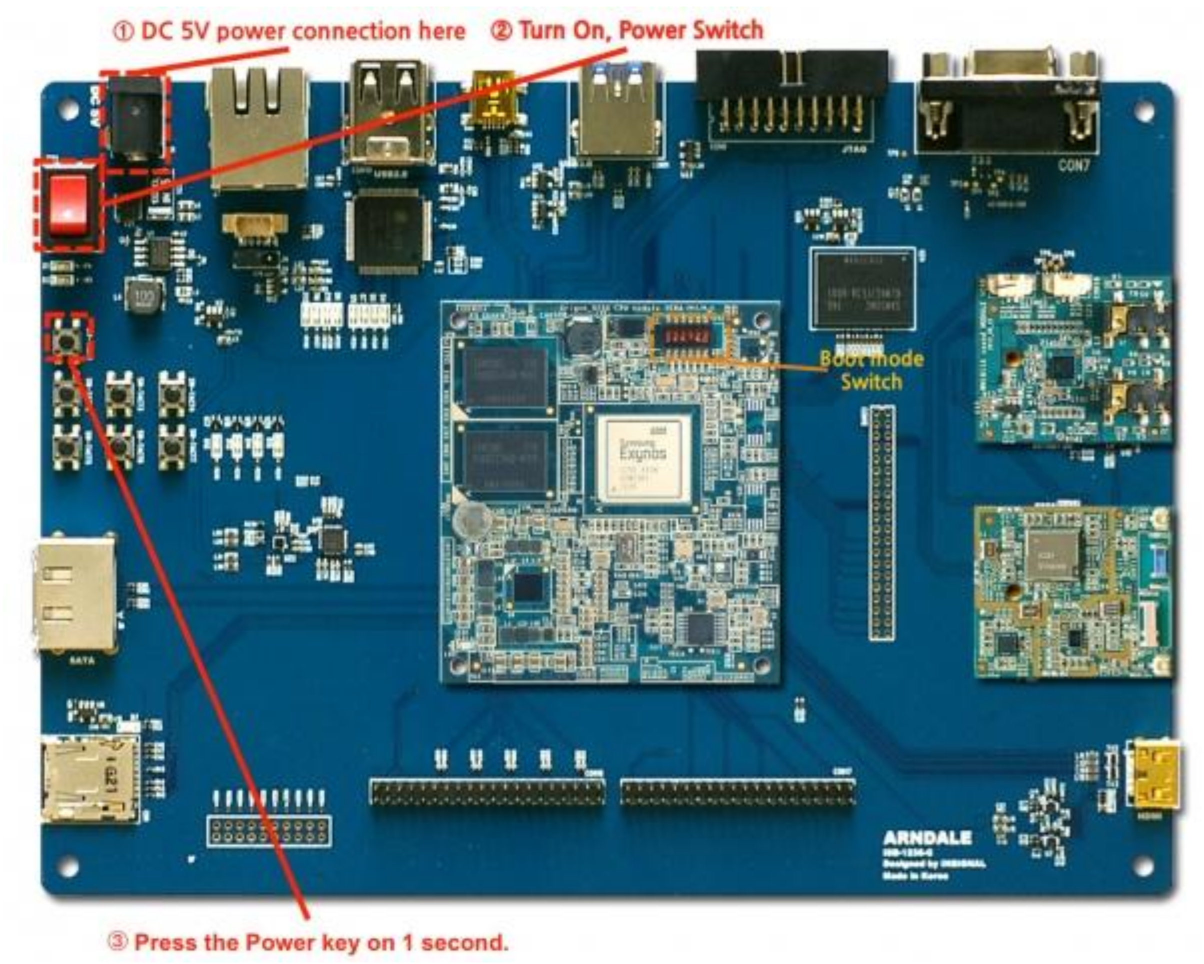
`-singlestep`

run in singe step mode, which does not use any internal optimizations

Appendix I.

INSTALLING/FLASHING ARNDALE BOARD

This chapter explains how to setup an Arndale board from scratch.


① DC 5V power connection here
② Turn On, Power Switch
Boot mode Switch
③ Press the Power key on 1 second.
ARNDALE

A) Mounting the LCD panel

The LCD panel (optional) comes with a flat ribbon terminated by a daughter board (LCD-MIPI2LVDS-A ver1.0). The daughter board must be connected to CON16 of the main board, with its arrow pointing outward. The LCD panel should be mounted on top of the main board using legs at least 3cm long.

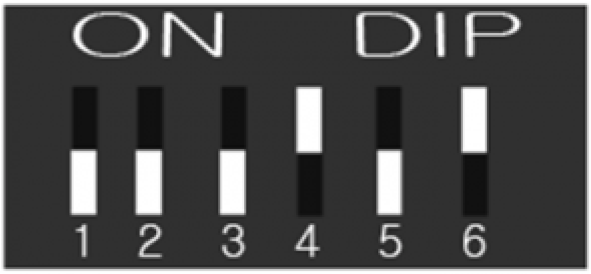
B) Boot mode setup

Arndale board provides two types of boot mode.

eMMC is the default boot mode.
The DIP switch is not accessible once the optional LCD display is mounted.

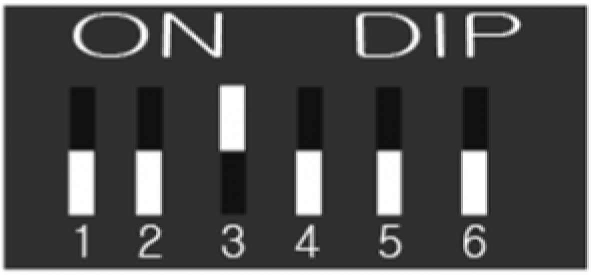
For eMMC bootmode configuration (it is the default bootmode):

DIP switch (CFG1/SW1) CFG1[6:1] : 101000



For SD/MMC bootmode (Use SD/MMC bootmode when eMMC bootmode is unstable):

DIP switch (CFG1/SW1) CFG1[6:1] : 000100



C) Turning on the board

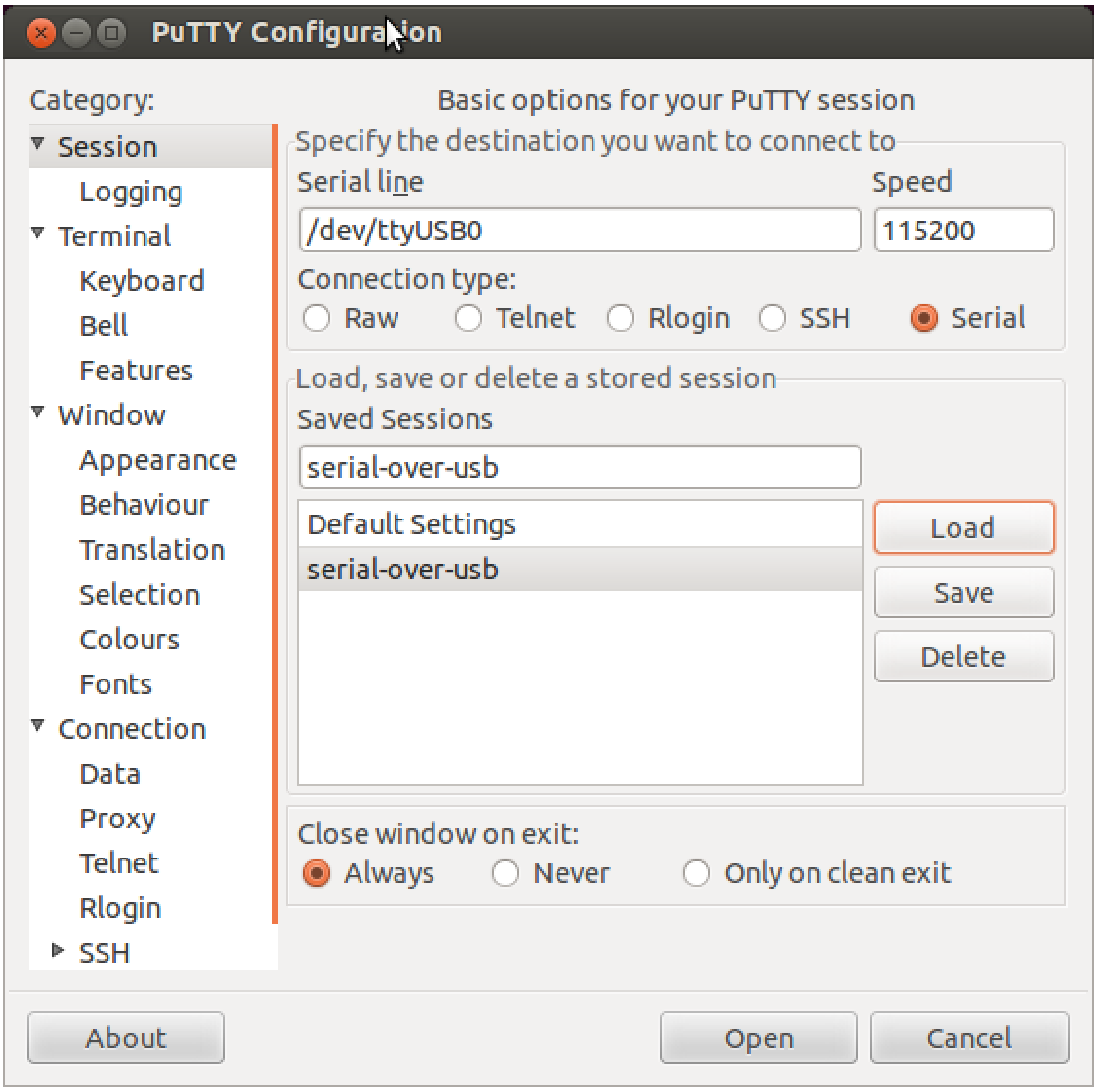
Turn the power switch on, then press and hold the power key for 1 second to boot the board (see picture above).
The board is delivered with android system already flashed and ready to use.

D) Setting serial port

Serial port is used to reflash the Arndale board.

Putty can be used to configure the serial port with the following configuration:

- Serial Port : /dev/ttyS0
- Serial over USB: /dev/ttyUSB0
- Baud Rate : 115200
- Parity : 8bit
- Hardware Flow Control : No
- Stop bit : 1bit



Specify the destination you want to connect to
Serial line: /dev/ttyUSB0
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Load, save or delete a stored session
Saved Sessions: serial-over-usb
Default Settings
serial-over-usb
Close window on exit: [✓ Always] Never Only on clean exit

D) Downloading software on target system

When booting the board, press any key (of Host PC Keyboard, focused on serial terminal window) within ~3 seconds to enter u-boot prompt.

```
OOK

U-Boot 2010.12 (Oct 11 2012 - 18:54:08) for Insignal Arndale

CPU: S5PC520 Rev1.0 [Samsung SOC on SMP Platform Base on ARM CortexA15]
APLL = 1400MHz, MPLL = 800MHz, EPLL = 96MHz, VPLL = 300MHz, BPLL = 666MHz
DRAM:  2047 MiB

TrustZone Enabled BSP
BL1 version: 20120430
```

```
PMIC: S5M8767

Checking Boot Mode ... EMMC4.41
REVISION: 1.0
REVISION: 1.0
NAME: S5P_MSHC0
MMC Device 0: 3816 MB
MMC Device 1: 0 MB
MMC Device 2 not found
*** Warning - using default environment

Hit any key to stop autoboot:  0
Arndale #
```

Type the 'fastboot' command on bootloader. The following message appears. Enter into the download mode.

```
Arndale # fastboot
 [Partition table on MoviNAND]
ptn 0 name='fwbl1' start=0x2 len=N/A (use hard-coded info. (cmd: movi))
ptn 1 name='bl2' start=N/A len=N/A (use hard-coded info. (cmd: movi))
ptn 2 name='bootloader' start=N/A len=N/A (use hard-coded info. (cmd: movi))
ptn 3 name='tzsw' start=N/A len=N/A (use hard-coded info. (cmd: movi))
ptn 4 name='kernel' start=N/A len=N/A (use hard-coded info. (cmd: movi))
ptn 5 name='ramdisk' start=N/A len=0x0(~27262976KB) (use hard-coded info.
(cmd: movi))
ptn 6 name='system' start=0x2 len=0x0(~512065536KB)
ptn 7 name='userdata' start=0x2 len=0x0(~512032768KB)
ptn 8 name='cache' start=0x2 len=0x0(~512032768KB)
ptn 9 name='fat' start=0x2 len=0x0(~-1845690368KB)
OTG cable Connected!
```

E) Flashing

Get the Android image archive arndale-jb.zip
Extract all file in the archive.
Get the Arndale sdk release package (i.e Arndale-tbase-XXX-release-tsdk_rX.tar.bz2)
Extract all file in the arndale-jb directory
Execute the following script from the host system: `fastboot_flash_all.sh`

F) Launching Android

Enter the following command to boot now.

```
sudo fastboot reboot
```

G) Configuring USB Access

Under GNU/linux systems (and specifically under Ubuntu systems), regular users can't directly access USB devices by default. The system needs to be configured to allow such access. The recommended approach is to create a file /etc/udev/rules.d/51-android.rules (as the root user) and to copy the following lines in it. <username> must be replaced by the actual username of the user who is authorized to access the phones over USB.

```
# adb protocol on passion (Nexus One)
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="4e12",
MODE="0600", OWNER="<username>"
# fastboot protocol on passion (Nexus One)
SUBSYSTEM=="usb",   ATTR{idVendor}=="0bb4",   ATTR{idProduct}=="0fff",
MODE="0600", OWNER="<username>"
# adb protocol on crespo/crespo4g (Nexus S)
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="4e22",
MODE="0600", OWNER="<username>"
# fastboot protocol on crespo/crespo4g (Nexus S)
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="4e20",
MODE="0600", OWNER="<username>"
# adb protocol on stingray/wingray (Xoom)
SUBSYSTEM=="usb",   ATTR{idVendor}=="22b8",   ATTR{idProduct}=="70a9",
MODE="0600", OWNER="<username>"
# fastboot protocol on stingray/wingray (Xoom)
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="708c",
MODE="0600", OWNER="<username>"
# adb protocol on maguro/toro (Galaxy Nexus)
SUBSYSTEM=="usb",   ATTR{idVendor}=="04e8",   ATTR{idProduct}=="6860",
MODE="0600", OWNER="<username>"
# fastboot protocol on maguro/toro (Galaxy Nexus)
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="4e30",
MODE="0600", OWNER="<username>"
# adb protocol on panda (PandaBoard)
SUBSYSTEM=="usb",   ATTR{idVendor}=="0451",   ATTR{idProduct}=="d101",
MODE="0600", OWNER="<username>"
# fastboot protocol on panda (PandaBoard)
SUBSYSTEM=="usb",   ATTR{idVendor}=="0451",   ATTR{idProduct}=="d022",
MODE="0600", OWNER="<username>"
# usbboot protocol on panda (PandaBoard)
SUBSYSTEM=="usb",   ATTR{idVendor}=="0451",   ATTR{idProduct}=="d00f",
MODE="0600", OWNER="<username>"
# usbboot protocol on panda (PandaBoard ES)
SUBSYSTEM=="usb",   ATTR{idVendor}=="0451",   ATTR{idProduct}=="d010",
MODE="0600", OWNER="<username>"
# adb protocol on grouper (Nexus 7)
```

```
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="4e42",
MODE="0600", OWNER="<username>"
# fastboot protocol on grouper (Nexus 7)
SUBSYSTEM=="usb",   ATTR{idVendor}=="18d1",   ATTR{idProduct}=="4e40",
MODE="0600", OWNER="<username>"
```

Those new rules take effect the next time a device is plugged in. It might therefore be necessary to unplug the device and plug it back into the computer. After you modify, the udev rules will be activated with restarting Host PC or "sudo udevadm control --reload-rules" command.

H) Setting up the network connection

a. WiFi

WiFi network is not working with the setup we currently use for t-base.

b. Ethernet network

Cable based network must be used. Unfortunately it is not working out of the box with the Insignal kernel. Reason is that the MAC address of eth0 is empty ().

c. Setting MAC address manually

```
$ /system/bin/netcfg eth0 hwaddr [MACADDR, e.g. 00:01:02:03:04:05]
```

Response will always be: “action 'hwaddr' failed (Invalid argument)”
but never mind, new MACADDR usually was set correctly, please verify withfollowing command:

```
$ netcfg
…
eth0   DOWN                 0.0.0.0/0      0x00001002 00:01:02:03:04:05
```

d. Starting DHCP

```
$ /system/bin/netcfg eth0 dhcp
```

There should be no error message in response
If response is “action 'dhcp' failed (Cannot assign requested address)” - choose a different MACADDR and try again

Please note:

This step has to be executed manually after each reboot of the board!

The MACADDR is not saved anywhere in flash memory

I) Setting DATE and TIME

Use the Android settings to set the current date and time of the device.

Otherwise the browser can’t verify the certificates of the web pages, and browsing is almost impossible.

Please note:

This step has to be executed manually after each reboot of the board!

There is no automatic setting of date and time.

J) Taking screenshots

Arndale uses the generic Android way of taking screenshots:

→ Press Volume down key and Power key simultaneously and hold for 1 second.

Volume down key (SW-TACT5) is two keys below Power key (SW-TACT1) on Arndale board.

Appendix II. Changing the SUID of <t-base image

There is a script provided that allows changing the device specific SUID in the TEE image `t-base.img`. TSM_ID and SUID can be changed in `run_qemu.sh` as showed in sample code below:

```
#parameters:
#-----------
# TSM_ID = 2 Byte Hex
# DEV_SUID = 10 Byte Hex device specific part of SUID
#
TSM_ID=0001
SUID=010000000000517fb45c
MODE=Release
#
#result:
#------- ARM-SiPID|TSM_ID|DEV_SUID
# new SUID in "mobicore.img.out" consists of 00000000-XXXX-YYYYYYYYYYYYYYYYYYYY
# .1.2.3.4 .1.2 .1.2.3.4.5.6.7.8.9.A
```

Note: TSM customers shall use their own TSM_ID provided by Trustonic when they create QEMU instances for connection to the backend servers. This avoids collisions with other test devices from other parties.

Appendix III. Using the Android display on QEMU

By Default the emulator starts a VNC connection that can be used to manipulate the Android Display.

You just need to start your favorite VNC viewer and connect to localhost:5900