1. Introduction

This document describes the required steps to run RTI Connext DDS Secure (DDS Secure) on a computer containing a Trusted Platform Module (TPM) version 2.0 on board. It provides details on how to take advantage of the TPM capability to securely store identity private keys inside the TPM.

This document refers to TPM version 2.0. If your machine has a TPM 1.2, please refer to the document "Using TPM 1.2 with DDS Secure", available in the HOWTO section of the RTI Community Portal.

How to read this document:

Commands executed in a terminal are enclosed in a green box:

```
$ nonRootCommand
# runMeAsRoot
```

Command lines starting with '$' are intended to be executed as a normal unprivileged user. Commands that require root access (either by logging in as root user, or using sudo) start with '#'.

The following exercises are described in this document:

- Ensure the TPM is working on the computer
- Build and install the tools required to access the TPM from the user's space
- Build RTI DDS Secure Plugins that can access the TPM
- Modify the existing security example to use the TPM

This document does not explain, nor attempt to be a substitution to the DDS Securite Plugin User's guide. You must understand the basic concepts of DDS Security in order to understand all the steps.

This document will use the DDS Secure example as a reference, which is located in the following directories:

- Example source code:
  `rti_workspace/6.0.0/examples/connext_dds/c/hello_security`

- Security artifacts: `rti_workspace/6.0.0/examples/dds_security`

1.2 Requirements

The standard DDS Secure for RTI Connext DDS does not allow loading private keys through an external OpenSSL engine for RTI Connext DDS versions prior to 6.0.1. If you are planning to use TPM 2.0 with RTI Connext DDS version 6.0.0 and older, you will need to get the source code of DDS Secure. See Section 3 for more information on how to obtain/build a version of the DDS Security plugin that supports loading keys from the engine.

OpenSSL: It is recommended that you use the same version of OpenSSL used by RTI Connext DDS Security Plugin. Refer to your RTI Connext installation to locate headers and libraries for OpenSSL.

2. Preparing the hardware

This guide has been tested using two different architectures:

- Raspberry PI 3 with an Infineon Optiga SLB 9670 TPM 2.0 Iridium board running Linux Raspbian with kernel 4.4.50
- Dell Inspiron with on-board TPM 2.0 running Linux Mint 19.0 with kernel 4.15.0

If you are using a different hardware or different OS, you might have to change a few things. Try to follow this document as a guideline and be prepared for small variations.

Keep in mind that not all computers (whether laptops or desktops) come with a TPM module. Most of the computer manufacturers typically include a TPM for their professional products lines. Newer computer models are equipped with a TPM2.0. Older machines might have a TPM 1.2. Refer to the computer technical specifications to find out if your computer has a TPM on board (and what version of TPM).
Note that older computers might be equipped with TPM 1.2 chip. This document covers only machines equipped with a TPM version 2.0. TPM 1.2 are not compatible with the TPM 2.0 software stack.

2.1 Verify the OS can access the TPM

Log into your system and verify the TPM is visible by looking either at the kernel boot log, or to see if a character device `/dev/tpm[0-9]` is present:

```
# dmesg | grep tpm
[    1.153786] tpm_tis 00:01: 1.2 TPM (device-id 0xB, rev-id 16)

# ls -l /dev/tpm*
crw-------. 1 root root 10, 224 Oct  5 11:55 /dev/tpm0
```

You might also see a /dev/tpmrm[0-9] device:

```
# ls -l /dev/tpm*
crw-rw----. 1 root root 10, 224 Oct  5 11:55 /dev/tpm0
crw-rw----. 1 root root 10, 224 Oct  5 11:55 /dev/tpmrm0
```

IMPORTANT
The character device `/dev/tpmrm0` identifies the same TPM physical device as `/dev/tpm0`, but its access is resource-managed (RM) by a kernel module that allows concurrent processes to access the physical device `/dev/tpm0`.

The resource-managed device for the TPM was introduced in Linux kernel 4.14. Older kernel version might not have it.

In this case you have two options:

- Only one process at a time can access the TPM (by accessing directly /dev/tpm0)
- You install (and configure) the abrmd (Access Broker and Resource Manager Daemon) service

In this guide we will be using the abrmd on the Raspberry PI and the resource-managed device for the Dell laptop.

Raspberry PI only:
At the time this document was written, the latest version of Raspbian did not include support for the Infineon TPM in its kernel.
Refer to this document:
https://www.infineon.com/dgdl/Infineon-TPM20_Embedded_SLB_9670_AppNote-AN-v01_00-EN.pdf?fileId=5546d46265257de8016537f329595e5c
for additional information on how to patch the kernel and gain access to the TPM module.

Most likely your TPM device is owned by the user 'root' and cannot be accessed by any other user. If you want to allow a non-root user to be able to access the TPM, refer to the installation notes for TPM 2.0 TSS (section 2.3.1).

2.2 Install the required software

To use the TPM from Linux, you need to have the following libraries and tools installed:

- TPM 2.0 TSS stack: https://github.com/01org/TPM2.0-TSS.git
- TPM 2.0 Tools: https://github.com/01org/tpm2.0-tools.git
- TPM Access Broker and Resource manager: https://github.com/01org/tpm2-abrmd (only needed if your kernel module does not support resource-managed TPM device `/dev/tpmrm0`)
- TPM 2.0 engine for OpenSSL: https://github.com/tpm2-software/tpm2-tss-engine

If your Linux distribution provides all these libraries and tools from the system's package manager, it is recommended to use them first (and you can skip to the next chapter). If for some reason you need to download, build and install them, follow these instructions.

In general, it is advised to go to each project page and follow the installation instructions, paying particular attention to the prerequisites.

The following sections contains important notes for each step/library you need to install.

2.2.1 TPM 2.0 TSS Stack

Installing the stack is a prerequisite before doing anything. The TSS stack contains the libraries used by all of the other projects.

Assuming the version of OpenSSL to use is installed under $OPENSSLHOME, open the file and build and install with:

```
$ git clone https://github.com/01org/TPM2.0-TSS.git
$ git checkout 2.2.0
$ ./bootstrap
$ CFLAGS=-I$OPENSSLHOME/include CXXFLAGS=$CFLAGS
LDFLAGS=-L$OPENSSLHOME/lib ./configure --disable-doxygen-doc
--disable-doxygen-man
$ make
$ sudo make install
```

NOTES:

- If you don't set CFLAGS, CXXFLAGS, LDFLAGS before invoking configure, the build system will use the default OpenSSL installed on your machine by your package manager. It is recommended that you use the same version of OpenSSL used by RTI Connext DDS Secure plugin.
- The command above uses the release 2.2.0 of the library. Feel free to use a newer version.
- When building on the Raspberry PI, you might have to manually modify the generated Makefile and remove all the references to unexpanded macros such as @CODE_COVERAGE_RULES@
- By default the package will be installed under /usr/local. Make sure the dynamic library loader can search for libraries under /usr/local/lib. Check that one of the files under /etc/ld.so.conf.d contains /usr/local/lib, then re-run:

  ```
  $ sudo ldconfig
  ```

to update the loader cache.

Ownership of TPM device:
By default the TPM device is owned by user 'root':

```
# ls -l /dev/tpm*
crw-------. 1 root root 10, 224 Oct  5 11:55 /dev/tpm0
crw-------. 1 root root 10, 224 Oct  5 11:55 /dev/tpmrm0
```

In this case, only the root user can access the TPM. It is advised to create a specific user and a specific group, then modify the udev rules to change the owner of those devices:

- Create a user 'tss' with no login and no password
- Create a group 'tss' and add any user who can be allowed to access the TPM for this group
- View/edit the file `/usr/local/lib/udev/rules.d/tpm-udev.rules` to ensure the ownership and access mode is as expected
- Create a symlink to the udev rule installed by the TSS stack:

```
# cd /etc/udev/rules.d
# ln -s /usr/local/lib/udev/rules.d/tpm-udev.rules .
```

Reboot your machine to allow the changes to take effect. After the reboot, you should see the changes when listing the TPM character devices:

```
$ ls -l /dev/tpm*
crw-rw---- 1 tss root  10,   224 Mar 12 11:35 /dev/tpm0
crw-rw---- 1 tss tss  253, 65536 Mar 12 11:35 /dev/tpmrm0
```

2.2.2 TPM 2.0 Tools

Similar to the TSS library, this is how you would download, build and install the TPM 2 tools:

```
$ git clone https://github.com/01org/tpm2.0-tools.git
$ git checkout 3.1.3
$ ./bootstrap
CURL_CFLAGS=-I/usr/local/include CURL_LIBS="-L/usr/local/lib
-lcurl" CRYPTO_CFLAGS=-I${OPENSSLHOME}/include
CRYPTO_LIBS="-L${OPENSSLHOME}/lib -lcrypto" ./configure
```

```
$ make
$ sudo make install
```

NOTE:

- One of the tools (tpm2_getpubek) requires the 'curl' library. On Raspberry PI, the version of CURL with your system is linked to the version of OpenSSL installed; it may be too old and incompatible with the one you are using. If you want to build this tool, you need to download and rebuild curl from source, using the version of OpenSSL you are currently using. Since we are not going to need that tool, we can skip building it.

Test that you can access the TPM correctly using the following command:

```
$ TPM2TOOLS_TCTI_NAME=device TPM2TOOLS_DEVICE_FILE=/dev/tpmrm0
tpm2_getrandom 10
```

All the TPM2 tools need to know how to communicate with the TPM. By default, the TSS library will:

1. Attempt to load the shared library '**libtss2-tcti-default.so**'
2. Attempt to load the shared library '**libtss2-tcti-tabrmd.so**'
3. Windows only: attempt to use the internal **TBS** (TPM Base Service - See https://docs.microsoft.com/en-us/windows/desktop/tbs/about-tbs)
4. Attempt to access the device: **/dev/tpmrm0**
5. Attempt to access the device: **/dev/tpm0**
6. Attempt to reach the Microsoft TPM 2 Software simulator through TCP at: **host=127.0.0.1, port 2321**

The environment variable 'TPM2TOOLS_TCTI_NAME' defines the TCTI (TPM Communication Transport Interface) to use. Refer to the main pages of the TPM2 commands for more information.

For additional information on how to use the TPM 2.0 tools, please refer to this page: https://github.com/tpm2-software/tpm2-tools/wiki/How-to-use-tpm2-tools

2.2.3 ABRMD

The Access Broker and Resource Manager Daemon (ABRMD) is only required for Kernel < 4.14, or where the `/dev/tpmrm0` device is not available. In our case, we built the abrmd on the Raspberry PI.

Pay attention to the prerequisite list of libraries, as abrmd requires dbus for inter-process communication. Install the required packages with:

```
# apt install dbus libdbus-1-dev libglib2.0-dev
```

The abrmd can be installed the same way as previous projects using:

```
$ git clone https://github.com/01org/TPM2.0-TSS.git
$ git checkout 2.1.0
$ ./bootstrap
$ ./configure
make
sudo make install
```

Once installed, you need to modify the dbus rules to pick up the abrmd rules that get installed under `/usr/local/etc/dbus-1`.

Edit the file `/etc/dbus-1/system.conf` and add the directory containing the local rules:

```
...
<!-- Config files are placed here that among other things, punch
holes in the above policy for specific services. -->
<includedir>system.d</includedir>
<includedir>/usr/local/etc/dbus-1/system.d</includedir>
...
```

The default dbus.service (as installed by 'apt') is started by systemd (systemctl) using the following command-line:
`/usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation`

NOTE: The argument '`--address=systemd:`' should not be there and must be removed when the dbus-daemon is invoked.

You have two options here:
1) Stop the dbus-daemon from the system and run it by hand:

```
# /usr/bin/dbus-daemon --system --nofork --nopidfile
--systemd-activation
```

Alternatively, you can enable the verbosity to troubleshoot any additional issues you might encounter with dbus using:

```
# DBUS_VERBOSE=1 dbus-daemon --system --print-address --nofork
```

2) Modify the systemd startup command that is located under:
`/lib/systemd/system/dbus.service`

Change the value of the 'ExecStart' under [Service] section and remove the `--address=systemd:` parameter:

```
ExecStart=/usr/bin/dbus-daemon --system --nofork --nopidfile
--systemd-activation
```

Then restart the service:

```
# systemctl restart dbus.service
```

Once dbus is configured correctly and running, you can run the abrm daemon with the following command:

```
$ sudo -u tss /usr/local/sbin/tpm2-abrmd --tcti=":/dev/tpm0"
```

Be sure to run the daemon as user 'tss' so it can correctly access the TPM.

Test it using the tpm2_getrandom:

```
$ TPM2TOOLS_TCTI_NAME=tabrmd tpm2_getrandom 10
```

Ensure the library /usr/local/lib/libtss2-tcti-tabrmd.so is accessible either from the default loader search path or by setting the environment library LD_LIBRARY_PATH. Alternatively, you can specify the full path of the library:

```
$ TPM2TOOLS_TCTI_NAME=/usr/local/lib/libtss2-tcti-tabrmd.so
tpm2_getrandom 10
```

2.2.4 TPM 2.0 OpenSSL Engine

Build the OpenSSL engine for the TPM 2.0 using the similar method:

```
$ git clone https://github.com/tpm2-software/tpm2-tss-engine
$ ./bootstrap
$ CRYPTO_CFLAGS=-I${OPENSSLHOME}/include
CRYPTO_LIBS="-L${OPENSSLHOME}/lib -lcrypto" ./configure
--enable-tctienvvar
$ make
$ sudo make install
```

NOTE:

- The option '--enable-tctienvvar' allows the engine to select the TCTI interface through the environment variable 'TPM2TSSENGINE_TCTI' (see below)

To test the engine, try generating some random numbers using Openssl this time:

```
$ LD_LIBRARY_PATH=/usr/local/lib/engines:$LD_LIBRARY_PATH openssl
rand -engine tpm2tss -hex 10
engine "tpm2tss" set.
WARNING:esys:src/tss2-esys/esys_tcti_default.c:137:tcti_from_file()
Could not load TCTI file: libtss2-tcti-default.so
WARNING:esys:src/tss2-esys/esys_tcti_default.c:137:tcti_from_file()
Could not load TCTI file: libtss2-tcti-tabrmd.so
a917a08bef1559884bac
WARNING:esys:src/tss2-esys/esys_tcti_default.c:137:tcti_from_file()
Could not load TCTI file: libtss2-tcti-default.so
WARNING:esys:src/tss2-esys/esys_tcti_default.c:137:tcti_from_file()
Could not load TCTI file: libtss2-tcti-tabrmd.so
```

The engine by default is installed under /usr/local/lib/engines so it won't be found automatically unless you include it in your LD_LIBRARY_PATH.

The warnings listed above are generated because the OpenSSL engine does not know what TCTI mechanism to use in order to communicate with the TPM. It will try to locate and load a series of shared libraries before attempting to reach /dev/tpm0.
If you run the ./configure script with the option '`--enable-tctienvvar`' the engine will look for the environment variable '`TPM2TSSENGINE_TCTI`' that now can be used to define the TCTI to use.

For example, to remove the warnings, use the following command:

```
$ TPM2TSSENGINE_TCTI=device:/dev/tpmrm0
LD_LIBRARY_PATH=/usr/local/lib/engines:$LD_LIBRARY_PATH openssl
rand -engine tpm2tss -hex 10
engine "tpm2tss" set.
726179f5c915d393e74f
```

Refer to the documentation in the TSS library for the format of the `TPM2TSSENGINE_TCTI` value.

2.3.5 Final tests

Before proceeding with the tests, check to ensure your TPM does not have a password set. This guide assumes the TPM does not have any passwords.

If you need to clear the passwords, use the tool '`tpm2_takeownership -c`'. Refer to the main page of the tool to see how to clear the passwords if the TPM already have a password assigned.

Optionally, you should be able to clear the TPM through the BIOS or by asserting physical presence. Refer to your computer or TPM board manufacturer for additional information on this task.

To ensure you have access to the TPM through the OpenSSL engine, try generating a RSA key:

```
$ OPENSSL_ENGINES=/usr/local/lib/engines
TPM2TSSENGINE_TCTI=device:/dev/tpmrm0 tpm2tss-genkey -a rsa -s 2048
~/MyTestKey.bin
```

Then extract the public portion of the key using OpenSSL (generate MyTestKeyPub.pem):

```
$ OPENSSL_ENGINES=/usr/local/lib/engines openssl rsa -engine
tpm2tss -inform ENGINE -in MyTestKey.bin -outform PEM -pubout -out
MyTestKeyPub.pem
```

NOTE: As you see from the above command line, you don't need to specify the environment variable TPM2TSSENGINE_TCTI since the public portion is stored directly inside the binary form of the key, and the engine doesn't need to access the physical TPM for this operation.

Try then to sign a document (the /etc/passwd system file) using the key we just created:

```
$ OPENSSL_ENGINES=/usr/local/lib/engines
TPM2TSSENGINE_TCTI=device:/dev/tpmrm0 openssl dgst -sha256 -sign
MyTestKey.bin -out passwd.sha256 -engine tpm2tss -keyform ENGINE
/etc/passwd
```

If everything works, you should get a signature file called '`passwd.sha256`'. Try to verify this now:

```
$ openssl dgst -sha256 -verify MyTestKeyPub.pem -signature
passwd.sha256 /etc/passwd
Verified OK
```

Note that to verify the signature, you only need the public key; you do not need to use the TPM at all.

3. Build DDS Secure

The RTI Connext DDS Secure version 6.0.0 and older does NOT support loading keys from the engine.
If you are using an older version of RTI Connext DDS Secure, contact RTI to obtain either a buildable source (with the required patch) or assistance on enabling this feature in the version of preference.

The following sections assume you have a DDS Security plugin that is able to load keys through the engine.

Additional information can be found on the document "Using TPM 1.2 with DDS Secure" in our RTI Community Portal.

4. Running DDS Security Demo

In this section, we describe all of the steps required to modify the DDS Security demo (located under rti_workspace/<version>/examples/connext_dds/c/hello_security) to run one participant with its identity protected by the TPM.

Build the example application using the appropriate Makefile as described in the RTI Connext DDS Secure Getting Started Guide.

First generate a private key inside the TPM and place it together with the other certificates in the example directory:

```
$ cd ~/rti_workspace/6.0.0/examples/dds_security/cert
$ OPENSSL_ENGINES=/usr/local/lib/engines
TPM2TSSENGINE_TCTI=device:/dev/tpmrm0 tpm2tss-genkey -a rsa -s 2048
peerTPM2key.bin
```

Then generate a CSR:

Copy the file 'example1ECdsa.cnf' into 'exampleTPM' then change it to:

```
prompt=no
distinguished_name      = req_distinguished_name

[ req_distinguished_name ]
countryName=US
stateOrProvinceName=CA
localityName=Santa Clara
organizationName=Real Time Innovations
emailAddress=meECdsa@rti.com
commonName=dtlsexampleTPM
```

Now using openssl, generate the CSR from this configuration file and sign it using the TPM engine:

```
$ OPENSSL_ENGINES=/usr/local/lib/engines
TPM2TSSENGINE_TCTI=device:/dev/tpmrm0 openssl req -engine tpm2tss
-keyform ENGINE -new -key peerTPM2key.bin -out peerTPM2.csr -config
exampleTPM.cnf
```

Now issue an identity certificate using the same CA used by the example.

IMPORTANT:
If this is the first time you are using the demo CA under examples/dds_security/cert, you MUST create a directory structure required by OpenSSL CA. This is done by running the following commands:

```
$ cd ~/rti_workspace/6.0.0/examples/dds_security/cert
$ mkdir demoCA
$ touch demoCA/index.txt
$ echo "01" > demoCA/serial
```

Now you can generate the certificate with:

```
$ cd ~/rti_workspace/6.0.0/examples/dds_security/cert
$ openssl ca -config openssl.cnf -days 365 -in peerTPM2.csr -out
peerTPM2.pem -keyfile cakey.pem -cert cacert.pem
```

Now modify the file:
`~/rti_workspace/6.0.0/examples/connext_dds/c/hello_security/USER_QOS_PROFILES.xml`

and add a new QoS participant profile called 'TPM_A' at the end of the <qos_library> definition:

```
<qos_profile name="TPM_A" base_name="RSA_A">
  <participant_qos>
    <property>
      <value>
        <element>
          <name>dds.sec.auth.identity_certificate</name>
          <value>file:../../../dds_security/cert/peerTPM2.pem</value>
        </element>
        <element>
          <name>dds.sec.auth.private_key</name>
          <value>file:../../../dds_security/cert/peerTPM2key.bin</value>
        </element>
        <element>
          <name>com.rti.serv.secure.library</name>
          <value>nddssecurity-tpm</value>
        </element>
        <element>
          <name>com.rti.serv.secure.authentication.keyform</name>
```

```
          <value>engine</value>
        </element>
        <element>
          <name>com.rti.serv.secure.openssl_engine</name>
          <value>tpm2tss</value>
        </element>
        <element>
          <name>com.rti.serv.secure.openssl_engine.tpm2tss.SET_TCTI</name>
          <value>device:/dev/tpmrm0</value>
        </element>
      </value>
    </property>
  </participant_qos>
</qos_profile>
```

Now change the permissions file to add a grant rule for the participant with the new certificate.

Edit the file:
`~/rti_workspace/6.0.0/examples/dds_security/xml/PermissionsA.xml`
and add the following section at the end of the grant rules:

```
        <grant name="ParticipantA_TPM">
            <subject_name>C=US, ST=CA, O=Real Time Innovations,
CN=dtlsexampleTPM/emailAddress=meECdsa@rti.com</subject_name>
            <validity>
                <!-- Format is CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm] in GMT -->
                <not_before>2014-06-01T13:00:00</not_before>
                <not_after>2034-06-01T13:00:00</not_after>
            </validity>
            <default>ALLOW</default>
        </grant>
```

Note the Common Name (in bold) that must match the certificate we just created.

After this change you need to sign the Permissions file again:

```
$ cd ~/rti_workspace/6.0.0/examples/dds_security/xml
$ openssl smime -sign -in PermissionsA.xml -text -out
signed/signed_PermissionsA.p7s -signer ../cert/cacertECdsa.pem
-inkey ../cert/cakeyECdsa.pem
```

Start a publisher (that is not configured to use the TPM):

```
$ ./objs/x64Linux3gcc4.8.2/HelloWorld_publisher 0 0 rsa
Writing HelloWorld Secure, count 0
Writing HelloWorld Secure, count 1
Writing HelloWorld Secure, count 2
Writing HelloWorld Secure, count 3
Writing HelloWorld Secure, count 4
...
```

Note that you need to supply the arguments "0 0 rsa" in order for the application to use the 'RSA_B' profile that shares the same identity CA as the TPM_A profile.

Now start the subscriber:

```
$ LD_LIBRARY_PATH=/usr/local/lib/engines:$LD_LIBRARY_PATH
./objs/x64Linux3gcc4.8.2/HelloWorld_subscriber 0 0 tpm
HelloWorld subscriber sleeping for 4 sec...
HelloWorld subscriber sleeping for 4 sec...
Received data:

   msg: "Hello World Secure (1)"
HelloWorld subscriber sleeping for 4 sec...
Received data:

   msg: "Hello World Secure (2)"
HelloWorld subscriber sleeping for 4 sec...
Received data:

   msg: "Hello World Secure (3)"
HelloWorld subscriber sleeping for 4 sec...
...
```

Conclusions

RTI Connext DDS Secure (DDS Secure) can be customized to use TPM 2.0 in order to securely store identity private keys inside the device. To use DDS Secure with a TPM version 1.2, please refer to the document "Using TPM 1.2 with DDS Secure" on the RTI Community Portal.