1. Introduction

This document describes the required steps to run RTI Connext DDS Secure (DDS Secure) on a computer that has a TPM (Trusted Platform Module) 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 1.2. If your machine has a TPM 2.0, please refer to the document "Using TPM 2.0 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 that start 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:

- Prepare a computer with TPM and activate it
- Build and install the tools required to access the TPM from user space
- Build DDS Secure Plugins that can access the TPM
- Generate the required keys, certificates and documents
- Build an example that uses DDS Secure
- Run the examples using the TPM
- Make additional modifications to automatically supply secrets to the engine in order to avoid asking for a password each time the application is run

1.2 Requirements

The standard DDS Secure Plugin for RTI Connext DDS does not allow loading private keys through an external OpenSSL engine RTI Connext DDS versions for version prior to 6.0.1. If you are planning to use TPM 1.2 with RTI Connext DDS version 6.0.0 and older, you will need to get the source code of the DDS Secure Plugins. Your RTI representative can provide you with access to the buildable source.

This document describes the required steps to use the TPM on an HP laptop. For other platforms or other computer brands/BIOS, you may need to change some steps.

All the operations shown here are performed on a machine running CentOS 7. It requires:

- A machine with TPM with CentOS 7 installed with developer tools
- Connext DDS 6.0.0 (or older versions):
    - Connext DDS with target x64Linux3gcc4.8.2
    - DDS Secure buildable source code

Note that Connext DDS 6.0.1 and above doesn't require modification of the DDS Secure Plug-in code.

If you plan to use a different OS, you can still follow these steps, but you may need to change a few things.

2. Preparing the hardware

In this section we illustrate how to use DDS Secure with TPM on an HP EliteBook 8460p. For other manufacturers, or different computer models, please refer to the computer's users manual for instructions on how to enable, clear, and control the TPM from the OS.

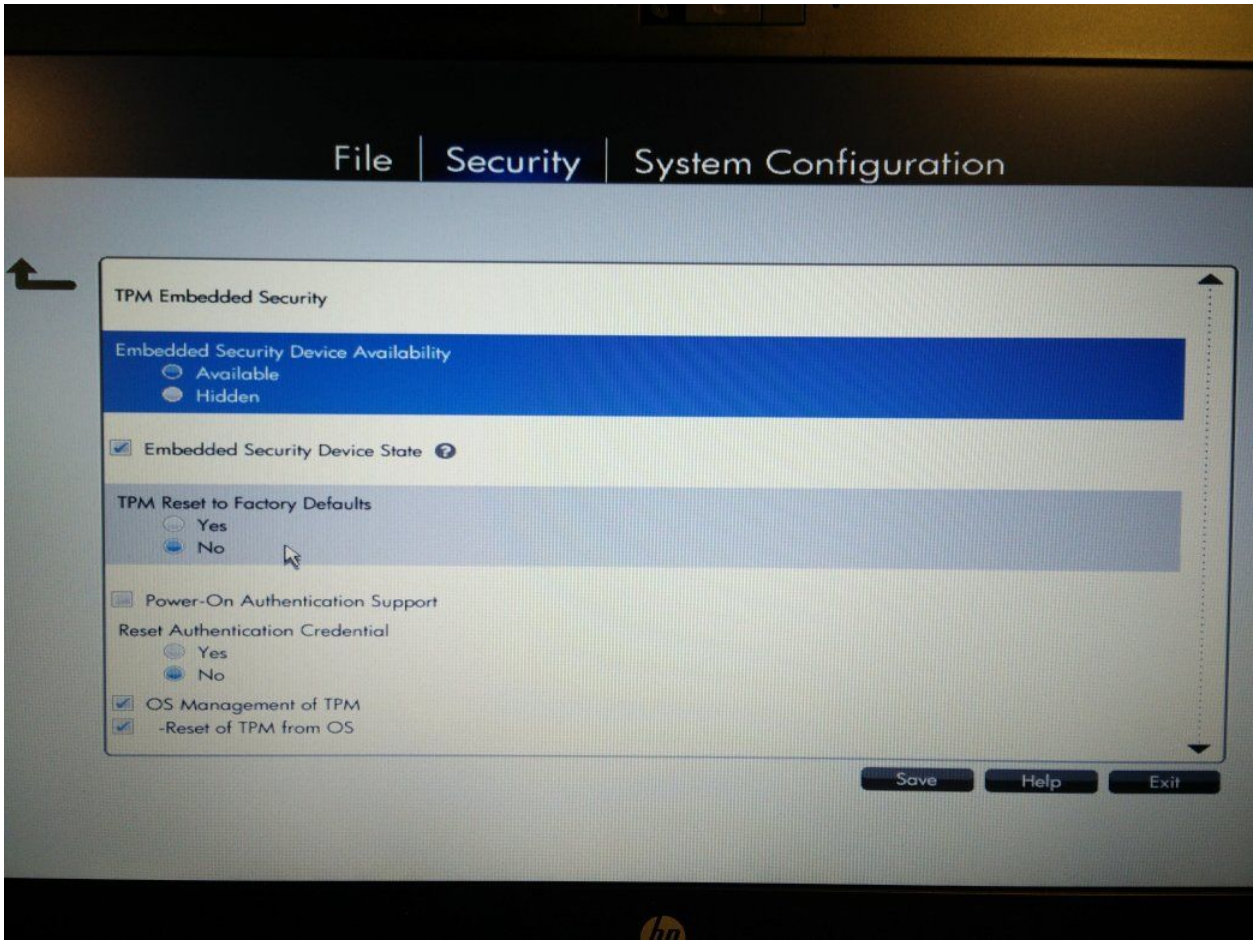
Not all laptops or desktops come with a TPM module. Most computer manufacturers typically include a TPM only in their professional product lines. Refer to your computer's technical specification to find out if your computer has a TPM on Rboard.
Late model computers may be equipped with a TPM 2.0 compatible chip. This document covers only computers equipped with a TPM version 1.2. TPM 2.0 are not compatible with the TPM 1.2 software stack.

The first step is to reset the TPM to factory default, in the event that it has been previously enabled or activated. If you have never used your TPM module before, you may be able to skip this step. Just make sure the TPM is enabled and accessible from the OS.

2.1 Reset the TPM to factory default

Resetting the TPM to factory default is done through the BIOS by asserting presence (something that cannot be done remotely). It can also be done from the OS only if the operation is permitted from the BIOS.

On the HP EliteBook BIOS, there is a section in the BIOS for "TPM Embedded Security" as shown below:



IMPORTANT: In order to access and/or clear the TPM, you need to assert physical presence. That is, you must prove that you are physically present in front of the machine. A clear operation cannot be performed from remote operations. Different vendor uses different techniques to assert physical presence. Refer to your computer manual to find out more.
In **HP** laptops, you can assert physical presence and access the TPM Embedded Security section of the BIOS by specifying a BIOS Administrator password. If your computer shows the security options disabled, try to set a BIOS Administrator password.
On **Lenovo** laptops, enabling and disabling the TPM is always accessible through the BIOS settings (with or without the BIOS Administrator password). In order to assert physical presence and show/enable the option to clear the TPM, you need to power on the computer while pressing the Blue key (either the Fn key or ThinkVantage key, depending on the model). After you assert physical presence, enter the BIOS. You should be able to see the Clear TPM option. After clearing the TPM, you should be able to use it directly from the OS. Refer to this

link for additional information

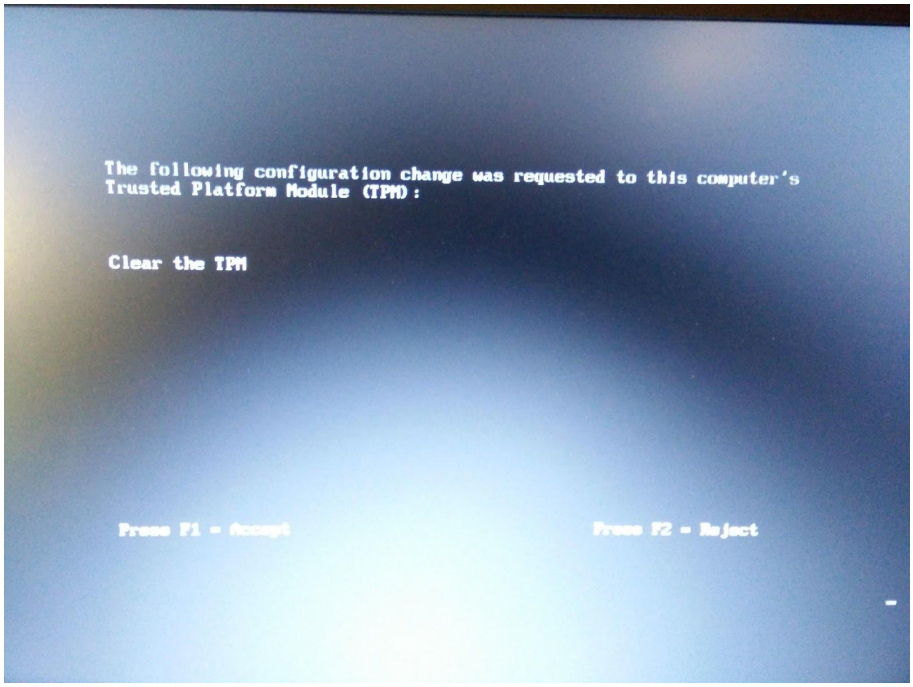
Ensure the TPM is enabled by setting all of the following to available (or checked):

- Embedded Security Device State (enable the TPM)
- Embedded Security Device Availability
- OS Management of TPM (allow taking ownership from OS)

Select “TPM Reset to Factory Default” ("Clear TPM" for Lenovo laptops). The BIOS will warn you about the potential loss of information. Essentially, any previously stored information in the TPM will be lost.

Save the settings and exit the BIOS. The computer will reboot.

At the next reboot, the BIOS will ask you to confirm the reset of the TPM:

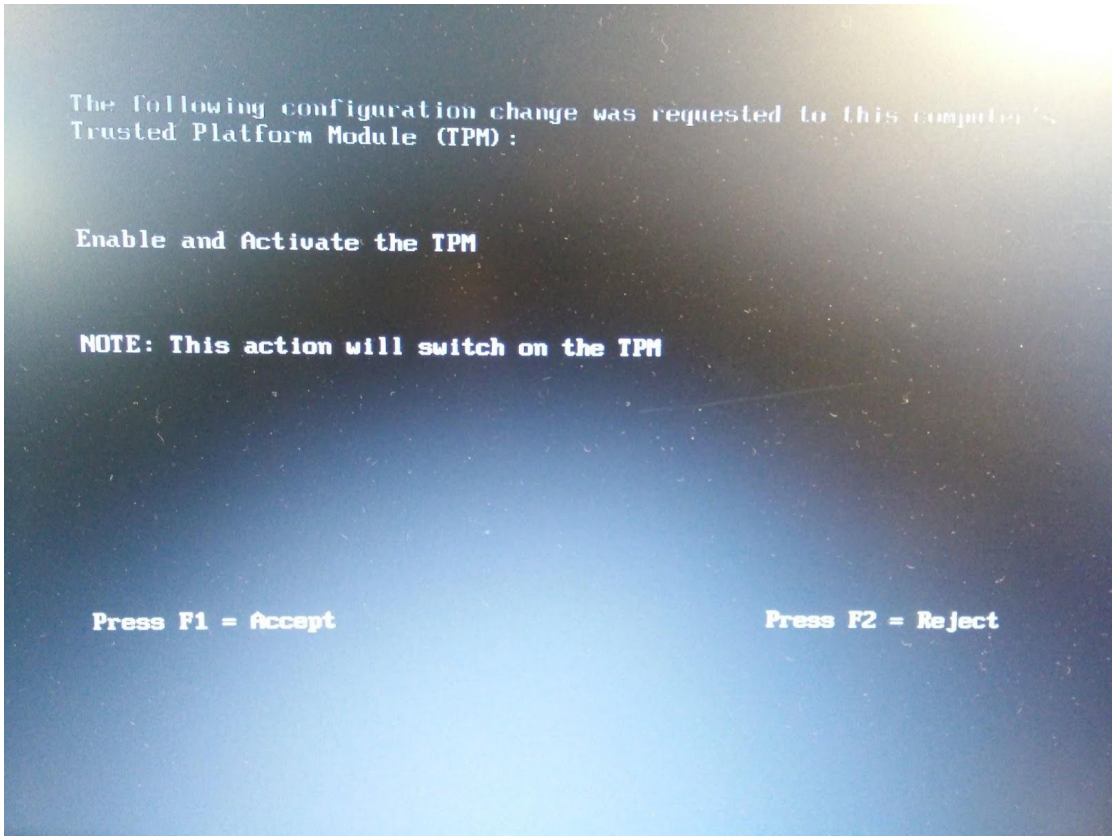


Confirm the operation. The computer will reboot again.

Lenovo laptops will clear the TPM without the reboot.

On HP laptops, after the TPM is reset to factory default, it will be **disabled in the BIOS**. You must go back to the BIOS and re-enable it by checking “Embedded Security Device State”. For Lenovo laptops, you don't need to reboot.

Similarly to resetting the TPM, after the TPM is enabled, the BIOS will reboot and ask to confirm the operation:



Press F1 to activate the TPM and reboot. After this operation, the TPM should be cleared and accessible by the OS.

2.2 Verify the OS can access the TPM

Boot into CentOS and verify the TPM is visible by looking at either the kernel boot log, or if the character device `/dev/tpm0` 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 can get more information about the TPM from the directory: `/sys/class/tpm/tpm0` It will appear as:

Public Endorsement Key	`# cat /sys/class/tpm/tpm0/device/pubek`
	`Algorithm: 00 00 00 01`
	`Encscheme: 00 03`
	`Sigscheme: 00 01`
	`Parameters: 00 00 08 00 00 00 00 02 00 00 00 00`
	`Modulus length: 256`
	`Modulus:`
	`CF 7A C8 CD C3 E9 8A 77 D9 6C 0B 68 C9 5C 48 F7`
	`F7 41 4C 5C 7B 11 9D B7 CE E0 DC 5E A3 3A 43 A3`
	`[...]`

Ownership status	`# cat /sys/class/tpm/tpm0/device/owned` `0`
Enabled status	`# cat /sys/class/tpm/tpm0/device/enabled` `1`
Active status	`# cat /sys/class/tpm/tpm0/device/active` `1`

Dump of all the PCRs	`# cat /sys/class/tpm/tpm0/device/pcrs`
	`PCR-00: 31 1D EC 10 60 95 D3 86 51 0B A6 FB C7 81 4A 1C 0B D2 C7 E0`
	`[...]`

TPM Information	`# cat /sys/class/tpm/tpm0/device/caps`
	`Manufacturer: 0x49465800`
	`TCG version: 1.2`
	`Firmware version: 3.17`

IMPORTANT: Verify the TPM version you have is 1.2 (in bold above). This document covers only 1.2 and not TPM 2.0.

2.3 Install the required software

To use the TPM from Linux, you need to download, build and install the following packages:

- OpenSSL. You can download OpenSSL directly from https://www.openssl.org, or you can use a pre-built version that can be downloaded from your RTI download page:

openssl-1.0.2j-target-<target>.tar.gz (i.e. target=x64Linux3gcc4.8.2)

If you are planning to build or use Connext DDS, you should use the version from your RTI download page.

- From the TrouserS project: http://trousers.sourceforge.net/ download and install the following sub-projects:
  - Trousers: https://sourceforge.net/projects/trousers/files/trousers/
    This will create the **tcsd** daemon that implements an application API on top of the TPM

```
$ ./configure
--with-openssl=/home/local/openssl-1.0.2o-x64Linux2.6gcc4.4.5/releas
e --prefix /usr/local
$ make
# make install
```

  - TPM Tools: https://sourceforge.net/projects/trousers/files/tpm-tools/
    This will create a set of command-line utilities for accessing the TPM (i.e. tpm_nvinfo, tpm_activate, …). These tools require the tcsd daemon to run.

```
$
LD_LIBRARY_PATH=/usr/local/lib:/home/local/openssl-1.0.2o-x64Linux2
.6gcc4.4.5/release/lib ./configure
--with-openssl=/home/local/openssl-1.0.2o-x64Linux2.6gcc4.4.5/relea
se --prefix /usr/local


$
LD_LIBRARY_PATH=/usr/local/lib:/home/local/openssl-1.0.2o-x64Linux2
.6gcc4.4.5/release/lib make

# make install
```

    - OpenSSL TPM Engine (to be patched as follows): https://sourceforge.net/projects/trousers/files/OpenSSL%20TPM%20Engine/
      This is the OpenSSL engine that offloads asymmetric cryptographic operations to the TPM. It requires the tcsd daemon to run. It also includes

the command-line application `'create_tpm_key'` that you can use to create private keys on the TPM. This code requires patching; see the note below.

```
$ LD_LIBRARY_PATH=/usr/local/lib:/home/local/openssl-1.0.2o-x64Linux
2.6gcc4.4.5/release/lib ./configure --with-openssl=/home/local
/openssl-1.0.2o-x64Linux2.6gcc4.4.5/release --prefix /usr/local

$
LD_LIBRARY_PATH=/usr/local/lib:/home/local/openssl-1.0.2o-x64Linux
2.6gcc4.4.5/release/lib make

# make install
```

Install all the above libraries under /usr/local (use the default `--prefix` value for the configure script).

NOTE TO PATCH OPENSSL TPM ENGINE: Since you are likely to be using a fairly recent version of OpenSSL, you need to download and apply the following patch. This patch contains the following changes:

- Adds the command-line option -z to the utility `create_tpm_key` to allow using the "well-known secret" as TPM authorization data.
- Corrects the exported functions of the engine to return -1 in case of error (instead of 0 which is not a valid error code).
- Adds the libraries 'libcrypto' and 'libdl' to the build files (without these libraries, you will get some unresolved externals)
- Adds a custom error code (TPM_R_TCSD_CONNECT_FAILED) that is returned when the engine cannot connect to the tcsd daemon.

Before building, patch the directory with:

```
$ cd openssl_tpm_engine-0.4.2
$ patch -p1 < ../rti_update_openssl_1_x.patch
```

IMPORTANT: After the build is completed, make sure your ldconfig contains the /usr/local/lib directory either by:

- **Adding /usr/local/lib to the /etc/ld.so.conf.d**
- **Setting LD_LIBRARY_PATH environment variable to include /usr/local/lib**

2.4 Activate the TPM

Although all the required libraries and applications are ready, the TPM right now is still unowned and disabled; it cannot be used yet.

Before doing anything, you need to start the tcsd daemon and make sure it starts after a reboot. To manually start the tcsd daemon, just launch it as root:

```
# tcsd
# ps axuww | grep tcsd
tss      10624  0.0  0.0  15320   528 ?        Ss   14:23   0:00
tcsd
```

To take ownership of the TPM, use the utility `tpm_takeownership` that is part of the tpm-tools package (it should be installed under `/usr/local/sbin`). This tool when launched without arguments will ask you for the owner password and the SRK password.

IMPORTANT: Do not use the well-known secret for owner and SRK (arguments -y and -z) as the TPM engine does not support it without modification. You can modify the source code to pass a well-known secret if you want. For now, just supply an easy to remember password, such as **'rti'** for both owner and SRK.

```
$ tpm_takeownership
Enter owner password: ********
Confirm password: ********
Enter SRK password: ********
Confirm password: ********
```

Verify that everything is good with the command tpm_setpresence:

```
$ tpm_setpresence -s
Enter owner password: ********
Physical Presence Status:
     Command Enable: true
     Hardware Enable: false
     Lifetime Lock: true
     Physical Presence: false
     Lock: true
```

```
```

If you are getting an error activating the TPM, you might need to clear the TPM by asserting physical presence. Refer to section "2.1: Reset TPM to factory default".

If you deactivate the TPM (using the TPM tool `tpm_clear`), the TPM chip will enter into a state that you won't be able to take ownership from, until you clear it from the BIOS. Again, refer to section 2.1 to clear the TPM.

Now you can create a private key that is physically stored inside the TPM:

```
$ create_tpm_key partATpmKey.pem
SRK Password: ********
```

After this step, you should end up with a file 'partATpmKey.pem' containing the KEYBLOB that is used to identify the internal private key inside the TPM. Tthe file obtained is NOT the real private key, as the real key is secured by the TPM:

```
$ cat partATpmKey.pem
-----BEGIN TSS KEY BLOB-----
BIICLwEBAAAAFQAAAAQAAAAAAQACAAMAAAAMAAAIAAAAAAIAAAAAAAAAAAAAAAQCZ
8GjOy3VHHFsQ2N4fJ0kqkte3/BHQbEGetCvwZROsuL5sJK9a6i9Qg5Rfqa7jMbhW
[...]
-----END TSS KEY BLOB-----
```

Put this key blob in a directory where you are planning to store all the various secure items (i.e. `~/tpmdemo`):

```
$ mkdir ~/tpmdemo
$ mv partATpmKey.pem ~/tpmdemo
```

This is a good time to test that your set of libraries previously installed are working. For example, now that you have a private key in the TPM, try to use Openssl to extract the public portion of the key:

```
$ export
LD_LIBRARY_PATH=/home/local/openssl-1.0.2o-x64Linux2.6gcc4.4.5/rele
ase/lib:/usr/local/lib/openssl/engines
$ openssl rsa -engine tpm -inform ENGINE -in partATpmKey.pem
-pubout -out partATpmPub.pem
```

After executing this command, you should have a file 'partATpmPub.pem' containing the public portion of the key.

To get the details of the public key directly from the private key (through the TPM), use the same 'openssl rsa' command:

```
$ openssl rsa -engine tpm -inform ENGINE -in partATpmKey.pem -noout
-text
engine "tpm" set.
SRK authorization:
Public-Key: (2048 bit)
Modulus:
    00:81:c6:9d:7b:d1:5f:56:f2:68:80:73:4e:b6:ea:
    d4:2e:4d:12:8e:e2:7a:7c:cd:d9:cd:bb:75:98:b8:
    [...]
    d6:da:4c:fc:bb:e4:8c:a0:39:c4:c3:e9:4b:f3:c4:
    ee:7f
Exponent: 65537 (0x10001)
```

IMPORTANT:

If you are getting an error from Openssl locating the dynamic library of the TPM engine (i.e. openssl looks under `/usr/lib64/openssl/engines`), just create a symbolic link of the built `libtpm.so` into the directory where openssl is looking for engines.

3. Build DDS Secure

When using TPM, all the cryptographic operations that require access to the private key will have to go through the TPM device, since only the TPM knows the key. DDS Secure will have to load the private key blob identifier and not the real private key, then use this key blob when performing the operations through the TPM engine.

Unfortunately the RTI Connext DDS Secure prior to 6.0.1 always loads the full private key, so we will have to make some modifications to the source code to allow the key blob to be loaded through the engine. If you are using RTI Connext DDS Secure 6.0.1 and above, you can skip this chapter.

The first step is to gather all the required software needed to rebuild DDS Secure:

- GNU C compiler (gcc)
- Linker, librarian (binutils)
- Make (make)

You can install those packages through the platform's package manager (yum, apt).

If you already installed openssl in the previous step, you can skip this one. Otherwise you will need OpenSSL. You can download OpenSSL directly from https://www.openssl.org, or you can use a pre-built version available from your RTI download page:

- openssl-1.0.2j-target-<target>.tar.gz (i.e. target=x64Linux3gcc4.8.2)

NOTE: You only need the target file; you do not need the host rtipackage file (i.e. openssl-1.0.2j-5.3.0-host-<arch>.rtipkg).

Un-tar the openssl target archive in your work directory (i.e. ~/tpmbuild):

```
$ mkdir ~/tpmbuild
$ tar xzf
~/Downloads/openssl-1.0.2j-target-x64Linux3gcc4.8.2.tar.gz
```

Download and extract the RTI Connext DDS Secure Buildable sources. If you don't have access to this file, please contact your RTI representative.

```
$ cd ~/tpmbuild
$ tar xzf ~/Downloads/rti_secure_dds_plugins-5.3.0-buildsrc.tar.gz
$ cd ndds530-security-buildsrc
```

Edit the file 'personal.mk'. Be sure the following variables are correctly set:

```
export OS_ARCH = x86_64Linux2.6
WAVEWORKSHOME = /home/fabrizio/tpmbuild/ndds530-security-buildsrc
NDDSHOME = /home/fabrizio/rti_connext_dds-5.3.0
RTI_OPENSSLHOME = /home/fabrizio/tpmbuild/openssl-1.0.2j
```

Finally, build the sources:

```
$ make x64Linux3gcc4.8.2.extract
$ make x64Linux3gcc4.8.2
```

If everything goes well, there should be no build errors, and you should end up with the DDS Secure library under: `security.1.0/lib/x64Linux3gcc4.8.2`:

- `libnddssecurity.so` - Shared library, release version
- `libnddssecurityd.so` - Shared library, debug version
- `libnddssecurityz.a` - Static library, release version
- `libnddssecurityzd.a` - Static library, debug version

Now download the patch file called '[securedds-5.3.0-tpm.patch]' and apply the patch with:

```
$ patch -p1 < ../securedds-5.3.0-tpm.patch
```

The patch affects the following files:

- `security.1.0/include/share/security/security_default.h`
- `security.1.0/srcC/authentication/Authentication.c`
- `security.1.0/srcC/authentication/h/Authentication.pkg.h`
- `security.1.0/srcC/authentication/peer/authentication.peer.h`
- `security.1.0/srcC/common/CertHelper.c`
- `security.1.0/srcC/common/h/CertHelper.pkg.h`
- `security.1.0/srcC/common/peer/common.peer.h`

Then rebuild:

```
$ make x64Linux3gcc4.8.2
```

This new version of the DDS Secure Plugin has a new configuration parameter (settable through the property QoS of the domain participant):

`com.rti.serv.secure.authentication.keyform`

The possible values for this parameter are:

- `pem` (default) - This is the standard format for keys generated using openssl

- `engine` - This value tells the plugin to load the key through the engine. When using keys stored in the TPM, you need to specify `keyform=engine`

At this point, we will install this newly modified version of the DDS Secure plugin in our RTI Connext DDS installation (let's rename it to '`libnddssecuritytpm.so`' to avoid conflict with the original version). You can manually execute those commands:

```
$ cp security.1.0/lib/x64Linux3gcc4.8.2/libnddssecurity.so
$NDDSHOME/lib/x64Linux3gcc4.8.2/libnddssecuritytpm.so

$ cp security.1.0/lib/x64Linux3gcc4.8.2/libnddssecurityd.so
$NDDSHOME/lib/x64Linux3gcc4.8.2/libnddssecuritytpmd.so

$ cp security.1.0/lib/x64Linux3gcc4.8.2/libnddssecurityz.a
$NDDSHOME/lib/x64Linux3gcc4.8.2/libnddssecuritytpmz.a

$ cp security.1.0/lib/x64Linux3gcc4.8.2/libnddssecurityzd.a
$NDDSHOME/lib/x64Linux3gcc4.8.2/libnddssecuritytpmzd.a
```

Or you can do this by creating a simple script to do it with a single operation (i.e. install-securedds.sh)

IMPORTANT:

TPM 1.2 does not support RSA PSS padding as required by the Object Management Group (OMG) standard for DDS Secure. The patch file contains modifications that disable RSA PSS padding when signing data between participants.

This will break compatibility with all the applications running standard DDS Secure.

If you need to interoperate with other machines running RTI Connext DDS with Security plugin, you need to disable RSA PSS padding as well on all those other participants. You are not going to be compliant with the OMG specifications, but you will be able to communicate securely with participants where identities are safely stored inside the TPM.

In DDS Secure version 6.0.1 and above you can use the property: '`com.rti.serv.secure.authentication.rsa_pss_pad`' to control whether to enable (default value) or disable RSA PSS padding.

4. Certificate Creation

In this section, we will create a certificate for the participant identity running on the machine with the TPM and identified by the private key previously created (`tpmdemo/partATpmKey.pem`).

To request the certificate from a certificate authority, you need to generate a CSR (Certificate Signing Request) signed using the private key of the requestor. Since the private key is only known by the TPM, only the TPM can produce the CSR.

Fortunately this operation can be done using openssl and the TPM engine.

First, create a configuration file for Openssl containing all the information about the certificate. Create a file like this:

```
[ req ]
prompt=no
distinguished_name      = req_distinguished_name

[ req_distinguished_name ]
countryName=US
stateOrProvinceName=CA
localityName=Sunnyvale
organizationName=Real Time Innovations
commonName=TpmParticipant_A
```

tpmdemo/partATpmCert.conf

Pay attention to the values used in the section `'req_distinguished_name'` (in bold above) as you will need to reuse those values for the Permissions file this participant will use.

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

```
$ export
LD_LIBRARY_PATH=/home/local/openssl-1.0.2o-x64Linux2.6gcc4.4.5/rele
ase/lib:/usr/local/lib/openssl/engines

$ openssl req -engine tpm -keyform engine -new -key partATpmKey.pem
-out partATpmCert.csr -config partATpmCert.cnf
engine "tpm" set.
SRK authorization: ********
```

After this, you will end up with the certificate signing request file `partCert.csr`.

If you want to analyze the content of the CSR file, you can always use Openssl:

```
$ openssl req -in partATpmCert.csr -noout -text
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=CA, L=Sunnyvale, O=Real Time Innovations,
CN=TpmParticipant_A
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:81:c6:9d:7b:d1:5f:56:f2:68:80:73:4e:b6:ea:
                    [...]
```

Notice the value of the Subject field: "`C=US, ST=CA, L=Sunnyvale, O=Real Time Innovations, CN=TpmParticipant_A`". This exact string needs to be specified in the Permissions file (see below).

Given the CSR, you can now obtain the certificate using your own certificate authority.
For this example, we are using a previously configured CA using OpenSSL:

```
$ openssl ca -config openssl.cnf -days 365 -in
~/tpmdemo/partATpmCert.csr -out ~/tpmdemo/partATpmCert.pem
Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 12 (0xc)
        Validity
            Not Before: Oct  6 22:51:10 2017 GMT
            Not After : Oct  6 22:51:10 2018 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = CA
            localityName              = Sunnyvale
            organizationName          = Real Time Innovations
            commonName                = TpmParticipant_A
```

```
Certificate is to be certified until Oct  6 22:51:10 2018 GMT (365
days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

NOTE: The above command require setting up a CA using OpenSSL. In particular before you can issue certificates, you need to set up a directory structure with a few required files that Openssl will use to keep a database of issues certificates. You also need to create a private key and a valid (even self-signed) certificate for the CA.

After this step, we have a full identity of our participant that will use the TPM for authentication.

5. Prepare the test application

For this tutorial, we will use one of the examples supplied with ConnextDDS. Let's use the one located under your workspace '`examples/connext_dds/c/hello_security`'.

This example will create two applications, a publisher and a subscriber.

The publisher will publish the string "Hello World Secure (*xx*)" (where xx is a progressive integer) for the topic "Example HelloWorld".

The subscriber will simply subscribe to "Example HelloWorld" and print the received samples on the console.

You need to modify the source code of the application to take the profile name as the third argument. You also need to ensure the demo application does not link statically with the security plugin. We want to be able to dynamically choose which version to use from the USER_QOS_PROFILES.xml.

Download and patch the hello_security directory using the following patch:

```
$ cd rti_workspace/5.3.0/examples/connext_dds/c/hello_security
$ patch -p1 < ../hello_security_dynamic.patch
```

After the patch is applied, you can specify the profile name (without the library QoS profile name) as the third argument of the generated applications. (The first argument is the domain number, the second one is the number of samples to write/receive).

Each application will have its own identity (private key + certificate + permissions file).

In this example, we are planning to prepare four profiles that can be chosen at run-time. Only two of them use the TPM. The other two can be used in another machine (or on the same HP laptop, but they won't be using TPM).

NOTE: On the machine without the TPM, you still need to use the modified version of the RTI DDS Secure Plugin due to the incompatibility with RSA PSS padding as described above.

Proceed with building the example as described in the READ_ME.txt included file, making sure you are using the dynamic libraries of RTI Connext DDS:

```
$ export NDDSHOME=<....>
```

```
$ make -f make/makefile_HelloWorld_x64Linux3gcc4.8.2 SHAREDLIB=1
Checking directory objs
Making directory objs
Checking directory objs/x64Linux3gcc4.8.2
Making directory objs/x64Linux3gcc4.8.2
[...]
```

Let's now take a look at the `USER_QOS_PROFILE.xml`, and let's define four profiles (see the example here):

- A: Standard profile using OpenSSL without a specific engine. This will use the following identity files:
  - `partAKey.pem`: private key
  - `partACert.pem`: certificate
  - `PermissionsA.p7s`: signed permissions file
- B: Similar to A, a profile that uses OpenSSL without TPM. This will use the following files:
  - `partBKey.pem`: private key
  - `partBCert`: certificate
  - `PermissionsB.p7s`: signed permissions file
- ATpm: Profile that uses TPM. Files:
  - `partATpmKey.pem`: private key
  - `partATpmCert.pem`: certificate
  - `PermissionsATpm.p7s`: signed permissions file
- BTpm: Profile that uses TPM. Files:
  - `partBTpmKey.pem`: private key
  - `partBTpmCert.pem`: certificate
  - `PermissionsBTpm.p7s`: signed permissions file

We have already created the key and certificate for one of the participants with TPM before. Now we will complete the creation of all the other files.

First the key, config, csr, and certificate for BTpm:

```
# Create private key
$ create_tpm_key partBTpmKey.pem

# Manually create the partBTpmCert.cnf file and use:
#      commonName=TpmParticipant_B

# Now create CSR
```

```
openssl req -engine tpm -keyform engine -new -key partBTpmKey.pem
-out partBTpmCert.csr -config partBTpmCert.cnf

# Finally create the certificate from CSR
openssl ca -config openssl.cnf -days 365 -in
~/tpmdemo/partBTpmCert.csr -out ~/tpmdemo/partBTpmCert.pem
```

Now let's create identities A and B that don't use the TPM:

```
# Create private key
$ openssl genpkey -algorithm RSA -out partAKey.pem -pkeyopt
rsa_keygen_bits:2048

# Manually create the partACert.cnf file and use:
#      commonName=Participant_A

# Now create CSR
openssl req -new -key partAKey.pem -out partACert.csr -config
partACert.cnf

# Finally create the certificate from CSR
openssl ca -config openssl.cnf -days 365 -in
~/tpmdemo/partACert.csr -out ~/tpmdemo/partACert.pem

# Repeat the same commands for partB
```

After this, we need to prepare the four Permissions files, one for each profile.

Each permission file in this example will be exactly the same, with the exception of the <subject_name> tag in each file, which must match the SubjectName field in the participant identity certificate:

- Permissions_ATpm.xml: `C=US, ST=CA, L=Sunnyvale, O=Real Time Innovations, CN=TpmParticipant_A`
- Permissions_BTpm.xml: `C=US, ST=CA, L=Sunnyvale, O=Real Time Innovations, CN=TpmParticipant_B`
- Permissions_A.xml: `C=US, ST=CA, L=Sunnyvale, O=Real Time Innovations, CN=Participant_A`
- Permissions_B.xml: `C=US, ST=CA, L=Sunnyvale, O=Real Time Innovations, CN=Participant_B`

For the scope of this example, we will use a very simple permissions and governance file where we don't accept any participant that does not offer an encrypted connection:

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/5.3.
0/dds_security_permissions.xsd">
    <permissions>
        <grant name="ParticipantA">
            <subject_name>C=US, ST=CA, L=Sunnyvale, O=Real Time
Innovations, CN=Participant_A</subject_name>
            <validity>
                <not_before>2013-06-01T13:00:00</not_before>
                <not_after>2023-06-01T13:00:00</not_after>
            </validity>
            <allow_rule>
                <domains>
                    <id>0</id>
                </domains>
                <publish>
                    <topics>
                        <topic>*</topic>
                    </topics>
                    <partitions>
                        <partition>*</partition>
                    </partitions>
                </publish>
                <subscribe>
                    <topics>
                        <topic>*</topic>
                    </topics>
                    <partitions>
                        <partition>*</partition>
                    </partitions>
                </subscribe>
            </allow_rule>
            <default>ALLOW</default>
        </grant>
    </permissions>
</dds>
```

PermissionsA.xml

Once you have the four Permissions files, proceed with signing it using your document CA:

```
$ openssl smime -sign -in Permissions_A.xml -text -out
Permissions_A.p7s -signer ~/CA/documentCA.pem -inkey
~/CA/private/documentCAKey.pem
```

Repeat the above command for all the four permissions files.

NOTE: you don't need to have a different CA to sign those documents. You can use the same CA that released the certificates for the identities.

Prepare also the governance file:

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/5.3.
0/dds_security_governance.xsd">
    <domain_access_rules>
        <domain_rule>
            <domains>
                <id_range>
                    <min>0</min>
                </id_range>
            </domains>

<allow_unauthenticated_participants>false</allow_unauthenticated_pa
rticipants>

<enable_join_access_control>true</enable_join_access_control>

<discovery_protection_kind>ENCRYPT</discovery_protection_kind>

<liveliness_protection_kind>ENCRYPT</liveliness_protection_kind>
            <rtps_protection_kind>SIGN</rtps_protection_kind>
            <topic_access_rules>
                <topic_rule>
                    <topic_expression>*</topic_expression>

<enable_discovery_protection>true</enable_discovery_protection>
```

```
<enable_read_access_control>true</enable_read_access_control>

<enable_write_access_control>true</enable_write_access_control>

<metadata_protection_kind>ENCRYPT</metadata_protection_kind>

<data_protection_kind>ENCRYPT</data_protection_kind>
                </topic_rule>
            </topic_access_rules>
        </domain_rule>
    </domain_access_rules>
</dds>
```

PermissionsA.xml

And sign it:

```
$ openssl smime -sign -in Governance.xml -text -out Governance.p7s
-signer ~/CA/documentCA.pem -inkey ~/CA/private/documentCAKey.pem
```

To recap, after this step you should have the following files under `~/tpmdemo`:

File Name	Description
documentCA.pem	The certificate of the Authority used to sign the Permissions and Governance files
identityCA.pem	The certificate of the Authority that generates the identity certificate (this can be the same as the documentCA).
part<X>Cert.pem	The identity certificate for the participant <X>
part<X>Key.pem	The private key for the participant <X>. TPM Keys are only a key blob (a binary buffer identifying the private key stored inside the TPM device). Non-TPM keys are full private keys.
Governance.p7s	The governance file signed by the document authority
Permissions<X>.p7s	The permissions files signed by the document authority

Where the value of <X> is the profile name: `A`, or `B`, or `ATpm`, or `BTpm`

As a final step, we will edit the `USER_QOS_PROFILE.xml` now, and specify the above files in the configuration. We need to remember to:

- Select the '`nddssecuritytpm.so`' as DDS Secure plugin (even for the configuration where we don't use the TPM. As previously noted, the issue is with the RSA PSS padding.)
- Be sure to specify the property 'keyform' with value 'engine' for the TPM configurations
- For RTI Connext 6.0.1 and above, disable also RSA PSS Padding by defining: `com.rti.serv.secure.authentication.rsa_pss_pad` to `false`

You can download the full USER_QOS_PROFILE.xml here.

6. Run the test applications

Verify that you have the following libraries in your dynamic library search path (LD_LIBRARY_PATH):

- The RTI Connext DDS core and API libraries (libnddscore.so, libnddsc.so)
- The newly built DDS Secure plugin with the TPM support (libnddsecuritytpm.so)
- OpenSSL crypto libraries (libcrypto.so)
- TrouserS libraries (libtspi.so)
- The OpenSSL TPM Engine (libtpm.so)

Also ensure that the Trousers daemon (tcsd) is running. If not, start it as root.

Open a terminal and start a publisher that uses the TPM (on the computer with the TPM):

```
$ ./HelloWorld_publisher 0 0 ATpm
Using profile: ATpm
Loading private key through engine...
SRK authorization: ********
Writing HelloWorld Secure, count 0
[...]
```

You must type the SRK password as you defined it, when you took ownership of the TPM.

On another computer (perhaps without TPM), start a terminal and start a subscriber:

```
$ ./HelloWorld_subscriber 0 0 B
Using profile: B
HelloWorld subscriber sleeping for 4 sec...
HelloWorld subscriber sleeping for 4 sec...
Received data:
```

```
   msg: "Hello World Secure (15)"
HelloWorld subscriber sleeping for 4 sec...
Received data:

   msg: "Hello World Secure (16)"
HelloWorld subscriber sleeping for 4 sec...
[...]
```

As you see, the two applications communicate securely through the first one that uses the TPM for the RSA cryptographic operations, and the second one that uses a standard OpenSSL.

If you are getting an error indicating the function "ENGINE_set_default(ENGINE_METHOD_ALL)" is failing, make sure the Trousers daemon (tcsd) is running on your machine.

6.1 Remove SRK Prompt

When executing an application that uses either the ATpm or BTpm profile, you will be prompted for the SRK Authorization password. This might not be suitable for an application that need to start automatically. This section demonstrates who to remove it or take control of providing the SRK Authorization password from the code.

The method used by the DDS Secure plugin to load the private key is the following:

```
EVP_PKEY *ENGINE_load_private_key(ENGINE *e,
                                  const char *key_id,
                                  UI_METHOD *ui_method,
                                  void *callback_data);
```

This is invoked in the modified version of the DDS Secure plugin, in `security.1.0/srcC/common/CertHelper.c:399`.
Right now the last two arguments ui_method and callback_data are not being used (set as NULL). Those two arguments can be used to supply a callback function that can provide the SRK password to the engine automatically.

As an example, let's modify the code to add a `ui_method` to supply a password that will be stored in `callback_data`:

```
static int myReadString(UI *ui, UI_STRING *uis) {
    if (UI_get_string_type(uis) == UIT_PROMPT) {
        printf("Automatically setting password for '%s' to '%s'",
                UI_get0_output_string(uis),
                (char *)UI_get0_user_data(ui));
        if (UI_set_result(ui, uis, UI_get0_user_data(ui)) >= 0) {
            return 1;
        }
    }
    return 0;
}

Peer
EVP_PKEY * RTI_Security_CertHelper_loadPrivateKey(const char *file,
ENGINE *engine, int keyFormat)
{
    EVP_PKEY *pkey = NULL;
    if (keyFormat == RTI_SECURITY_KEY_FORMAT_ENGINE) {
        if (engine == NULL) {
            // Cannot load ENGINE keyform: OpenSSL engine not
defined
            goto done;
        }
        printf("Loading private key through engine...\n");
        {
            UI_METHOD *ui_method;
            ui_method = UI_create_method("Custom Password Provider");
            UI_method_set_reader(ui_method, myReadString);

            pkey = ENGINE_load_private_key(engine, file, ui_method,
"rti");
            UI_destroy_method(ui_method);
        }

[...]
```

security.1.0/srcC/common/CertHelper.c

The bolded text above indicates the new sections added. The SRK password hardcoded is the string “rti” (highlighted).

Rebuild and re-run the test using the profile ATpm:

```
$ ./HelloWorld_publisher 0 0 ATpm
Using profile: ATpm
Loading private key through engine...
Automatically setting password for 'SRK authorization: ' to 'rti'
Writing HelloWorld Secure, count 0
Writing HelloWorld Secure, count 1
[...]
```

This time the application will start without asking for the SRK authorization password.

Conclusions

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