OPTIGA™ TPM Application Note

Integration of TLS Functionality for OPTIGA™ TPM SLx 9670 TPM 2.0

Devices

- OPTIGA™ TPM SLB 9670 TPM2.0
- OPTIGA™ TPM SLI 9670 TPM2.0
- OPTIGA™ TPM SLM 9670 TPM2.0

About this document

Scope and purpose

The world we live in is a connected world. Today we rely on our phones, computers and connected devices to communicate, buy goods, travel and work. It is expected that these devices increase exponentially.

All these devices that are connected to the internet have one thing in common – They rely on the protocol called TLS (Transport Layer Security) to protect their information in transit.

TLS is a cryptographic protocol designed to provide secure communication over an insecure infrastructure. This means that, if this protocol is properly deployed, you can open a communication channel to an arbitrary service on the internet and be reasonably sure that you're talking to the correct server, and exchange information safely knowing that your data will not fall into the wrong hands and that it will be received intact.

This is not the case in the real world. Complex systems along with software bugs can open a back door to an attacker. Aside from this, the simplicity of the RSA algorithm (which is widely used in most of the systems running TLS), has known weaknesses, such as the Private Keys being stored in software.

Anyone with the access to the corresponding private key can decrypt the communication between the client and it. This type of attack does not need to happen in real time. An attacker could establish a long-term operation and record all the encrypted traffic and wait until he obtains the Key. After the Key has been compromised, it's possible to decrypt all previously recorded traffic.

The OPTIGA™ TPM SLx 9670 TPM2.0 is a hardware security controller fully compliant with TCG TPM products with CC (EAL4+) and FIPS certification that can be used to harden the TLS connection by managing and keeping the Private Keys secure amongst other security features.

The OPTIGA™ TPM SLx 9670 TPM2.0 products, standard, automotive and industrial versions, differ with regards to supported temperature range, lifetime, quality grades, test environment, qualification and reliability to fit the target applications requirements. For more details refer to Infineon's website [1].

This document contains an overview of what is Transport Security Layer (TLS) and step-by-step instructions on how to use the TLS Stack Software with the Trusted Platform Module OPTIGA™ TPM SLx 9670 TPM2.0 on a Raspberry Pi® 3B+/4 Linux environment by using a Server-Client connection.

The described steps to integrate an OPTIGA™ TPM in a Raspberry Pi® 3B+/4 Linux environment can be performed with one of the Infineon Iridium SLx 9670 TPM2.0 SPI Boards, listed in the Table below.

Iridium Boards:

Supported TPM	Order type	Order number
OPTIGA™ TPM SLB 9670 TPM2.0	IRIDIUM 9670 TPM2.0	SP001596592
OPTIGA™ TPM SLI 9670 TPM2.0	IRIDIUM SLI 9670 TPM2.0	SP004232000
OPTIGA™ TPM SLM 9670 TPM2.0	IRIDIUM SLM 9670 TPM2.0	SP004232004

The 3 Infineon Iridium Boards are referred in the following as "Infineon Iridium SLx 9670 TPM2.0 SPI Board"

About this document

Intended audience

This document is intended for customers who want to increase the security level of their embedded platforms using a OPTIGA™ TPM SLx 9670 TPM2.0 from Infineon Technologies in combination with the Open Source TPM Software Stack 2.0 (TSS 2.0) and like to evaluate how to incorporate TLS with the TPM for their target applications.

This application note was tested using a Raspberry Pi® 3B+/4 and Raspbian Buster with desktop and recommended software with kernel version 4.19 [7].

Table of contents

List of figures

1 The Transport Layer Security (TLS 1.2)

TLS are cryptographic protocols designed to provide secure communication over insecure infrastructure. When these protocols are properly deployed, you can open a communication channel to an arbitrary service on the Internet and have certain level of security and assurance that it will be talking with the correct server and exchange information safely (confidentiality). These protocols protect the communication link.

These are the main objectives of TLS:

- Cryptographic Security
    - Enables Authentication, Confidentiality and Integrity in a communication between two parties that exchange information.
- Interoperability
    - TLS protocols are not system dependent. They can be used for example in Linux, Android, Bare Metal systems.
- Extensibility
    - TLS is a framework for the development of cryptographic protocols. TLS looks to be independent from cryptographic primitives, like ciphers and hashing functions.

We will discuss in section 2.1 [Using SLx 9670 TPM2.0 to Harden the TLS Session] some of the disadvantages of using software based TLS libraries solely and how to achieve a higher level of security and assurance when using SLx 9670 TPM2.0.

2 TLS Hardening by OPTIGA™ SLx 9670 TPM2.0

At the heart of every TPM software implementation there is the TPM Software Stack (TSS). It is a middleware to support, improve and simplify TPM usage for programmers. In general, the TSS features a layered design to fit various requirements from application developers by providing different user APIs with varying abstraction layers and functionality.

2.1 Using SLx 9670 TPM2.0 to Harden the TLS Session

TLS is used by web services and IoT devices to transmit sensitive information between client/Endpoint and Server/Cloud applications. TLS provides authenticated key exchange using asymmetric cryptography, data confidentiality using symmetric encryption and message integrity using message authentication codes scheme. However, these crypto primitives are stored in system memory and do not provide any trustworthiness assurance of the involved endpoint.

The drawback is that their implementation is using software library modules that store private keys in application or secure memory and have proven to contain bugs or vulnerabilities which have been exploited for the last several years.

By using SLx 9670 we can embed crypto operations inside dedicated fixed TPM 2.0 function calls used by TLS protocols, like for example using TPM 2.0 as the source of entropy for the TLS required random number.

The scope of this application note is to show the benefit of using SLx 9670 TPM2.0 to protect the private key involved in the TLS handshake process. This is only one of different ways SLx 9670 TPM2.0 can help harden a TLS session. Token Biding is another.

Token Binding is an extension to TLS that provides stronger authentication and longer, more robust sessions.

The idea behind token binding is “proof of possession”. The challenge with tokens is that they’re only as secure as where you store them. Token binding uses cryptographic key pairs and TPM 2.0 for secure storage.

With token binding, man in the middle attacks can’t forward requests or replay credentials because they can’t prove they have the key bound to the token.

Token Binding will not be covered in this application note. For more information about Token Binding, refer to [9]

2.2 tpm2-tss-engine OpenSSL Plug-In

Enhanced System API (ESAPI) is part of the open source software stack for TPM 2.0 “TSS”. It makes work easier for developers who want to use the Trusted Platform Module (TPM) 2.0 – a standardized hardware-based security solution for securing industrial, automotive and other applications such as network equipment. This is the first open source TPM middleware that complies with the Software Stack (TSS) Enhanced System API (ESAPI) specification of the Trusted Computing Group (TCG), providing significant value to the open source community.

The ease of integration on Linux and other embedded platforms comes with the release of the TPM 2.0 ESAPI stack which speeds up the adoption of TPM 2.0 in embedded systems such as network equipment and industrial systems.

You can find the ESAPI TPM 2.0 TSS stack within the **tpm2-software** open-source project [2].

Note: *Before moving on with this application note a pre-requisite is to have the following components of the **tss2-software** package installed:*

- TPM Software Stack 2.0 (tpm2-tss)
- TPM2 Access Broker & Resource Manager (tpm2-abrmd)
- TPM2 Tools (tpm2-tools)
- TPM2 TSS Engine (tpm2-tss-engine)
- Cryptsetup (cryptsetup)

The details on how to install and test these packages are part of the documentation on GitHub [2] and can be found within the TPM Evaluation Kit. For more information about the TPM Evaluation Kit, please get in touch with your local sales.

The **tpm2-tss-engine** project implements a cryptographic engine for **OpenSSL** [3] for **Trusted Platform Module (TPM 2.0)** [1] using the **tpm2-tss** [2] software stack that follows the Trusted Computing Groups (TCG) **TPM Software Stack (TSS 2.0)** [5]. It uses the **Enhanced System API (ESAPI)** [6] interface of the TSS 2.0 for downwards communication. It supports RSA decryption and signatures as well as ECDSA signatures.

2.3 Using tpm2-tss-engine

We will be using the tpm2-tss-engine as an OpenSSL engine to harden the TLS channel using TPM 2.0.

The development platform used for this task will be the Raspberry Pi® 3B+/4 along the OPTIGA™ IRIDIUM 9670 TPM2.0 board.

To demonstrate the hardening of the TLS session between a Client/Endpoint and Server/Cloud the OpenSSL S_Server and S_Client modules will be used along with the local host capability of Linux running on Raspberry Pi® 3B+/4.

2.3.1 Sanity-Test

The following command can be executed to check if the tpm2-tss-engine has been installed successfully.

Code Listing 1 Sanity test for the TSS Engine

```
001        openssl engine -t -c tpm2tss
```

It should retrieve the engine information about the name and the available functions as shown in Figure 1.

```
pi@raspberrypi:~ $ openssl engine -t -c tpm2tss
(tpm2tss) TPM2-TSS engine for OpenSSL
 [RSA, RAND]
     [ available ]
```

Figure 1 Sanity test for TSS Engine

2.4 OpenSSL Version

Usage of S_Server with HSM-protected private keys is only supported on OpenSSL 1.1.0 and newer. To check the installed version of OpenSSL, run the following command.

```
pi@raspberrypi-os:~ $ openssl version
OpenSSL 1.1.0k  28 May 2019
```

Figure 2 OpenSSL version

2.5 Using OpenSSL and the TPM2-TSS Engine to Create PKI Used in TLS session.

2.5.1 TPM 2.0 Key Management

The Trusted Platform Module (TPM) greatest strength is to enable an application the use of cryptographic keys while keeping them safe inside the TPM. It can both generate and import externally generated keys.

Each key has individual security controls, which can include a password or an enhanced authorization policy. These keys can be certified by the TPM and used to certify other keys as well.

In order to manage internal memory efficiently, the TPM, has the capability to wrap keys (encrypt) with the parent Key and store them (the encrypted key) outside TPM and still not compromising the overall security of the system. When the time comes to use this key, the wrapped Key is loaded into the TPM. Only the specific TPM used to wrap the key can unwrap it and use it, as shown in Figure 3.

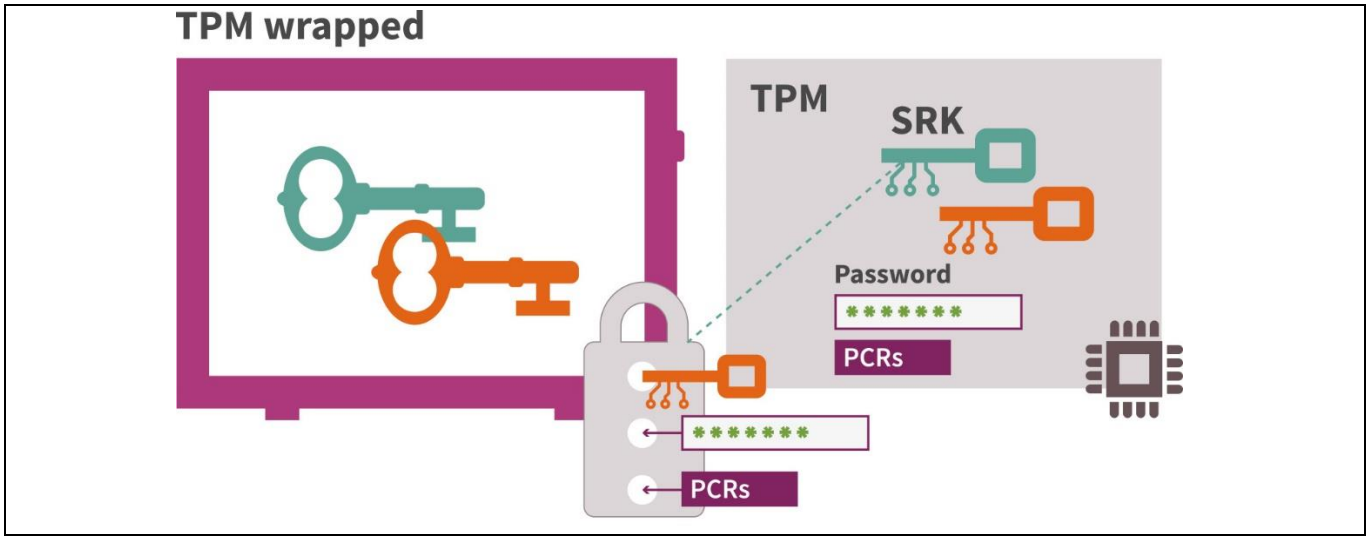

Figure 3 TPM 2.0 Key Wrapping

2.5.2 Creating OpenSSL Configuration File

To establish a TLS session between a Client/Endpoint and the Server/Cloud with OpenSSL and TPM2-TSS engine we need to create an OpenSSL configuration file. This file will have the configuration we will be using to create our rootCA, Server/Cloud, Client/Endpoint Certificates and CSRs. These are part of a Public Key Infrastructure (PKI).

Create a working directory called "**tpm_hardened_tls**". Within this directory create the "**pki**" directory structure and the "**tpm2**" directory structure.

Code Listing 2

```
001        mkdir tpm_hardened_tls
002        cd tpm_hardened_tls
003        mkdir -p pki/{csr,certs,crl,newcerts,private}
004        mkdir -p tpm2/{csr,certs,tpm_wrapped_keys}
```

```
pi@raspberrypi-os:~ $ mkdir tpm_hardened_tls
pi@raspberrypi-os:~ $ cd tpm_hardened_tls/
pi@raspberrypi-os:~/tpm_hardened_tls $ mkdir -p pki/{csr,certs,crl,newcerts,private}
pi@raspberrypi-os:~/tpm_hardened_tls $ mkdir -p tpm2/{csr,certs,tpm_wrapped_keys}
pi@raspberrypi-os:~/tpm_hardened_tls $
```

Figure 4 Creating directory structure

Create a new file with the name "**openssl.cnf**", "**index.txt**", "**index.txt.attry**", under the tpm_hardened_tls directory.

Code Listing 3

```
001        touch ./pki/openssl.cnf
002        touch ./pki/index.txt
003        touch ./pki/index.txt.attry
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ touch ./pki/openssl.cnf
pi@raspberrypi-os:~/tpm_hardened_tls $ touch ./pki/index.txt
pi@raspberrypi-os:~/tpm_hardened_tls $ touch ./pki/index.txt.attry
```

Figure 5 OpenSSL Configuration File

Then copy and paste the OpenSSL Configuration File [Code Listing 4] into "**openssl.cnf**".

Code Listing 4 OpenSSL Configuration File

```
# OpenSSL intermediate CA configuration file.
# Copy to `/root/ca/intermediate/openssl.cnf`.

[ ca ]
```

```
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = ./pki
certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/private/rootCA.key
certificate       = $dir/private/rootCA.crt

# For certificate revocation lists.
crlnumber         = $dir/crlnumber
crl               = $dir/crl/intermediate.crl
crl_extensions    = crl_ext
default_crl_days  = 30

# SHA-1 is deprecated, so use SHA-2 instead.
default_md        = sha256

name_opt          = ca_default
cert_opt          = ca_default
default_days      = 375
preserve          = no
policy            = policy_loose

[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of `man ca`.
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ policy_loose ]
# Allow the intermediate CA to sign a more diverse range of certificates.
# See the POLICY FORMAT section of the `ca` man page.
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256
```

```
# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address

# Optionally, specify some defaults.
countryName_default             = US
stateOrProvinceName_default     = California
localityName_default            = Milpitas
0.organizationName_default      = Infineon
organizationalUnitName_default  = DSS
emailAddress_default            =

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ usr_cert ]
# Extensions for client certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = client, email
nsComment = "OpenSSL Generated Client Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, emailProtection

[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth

[ crl_ext ]
# Extension for CRLs (`man x509v3_config`).
authorityKeyIdentifier=keyid:always

[ ocsp ]
# Extension for OCSP signing certificates (`man ocsp`).
basicConstraints = CA:FALSE
```

```
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, digitalSignature
extendedKeyUsage = critical, OCSPSigning
```

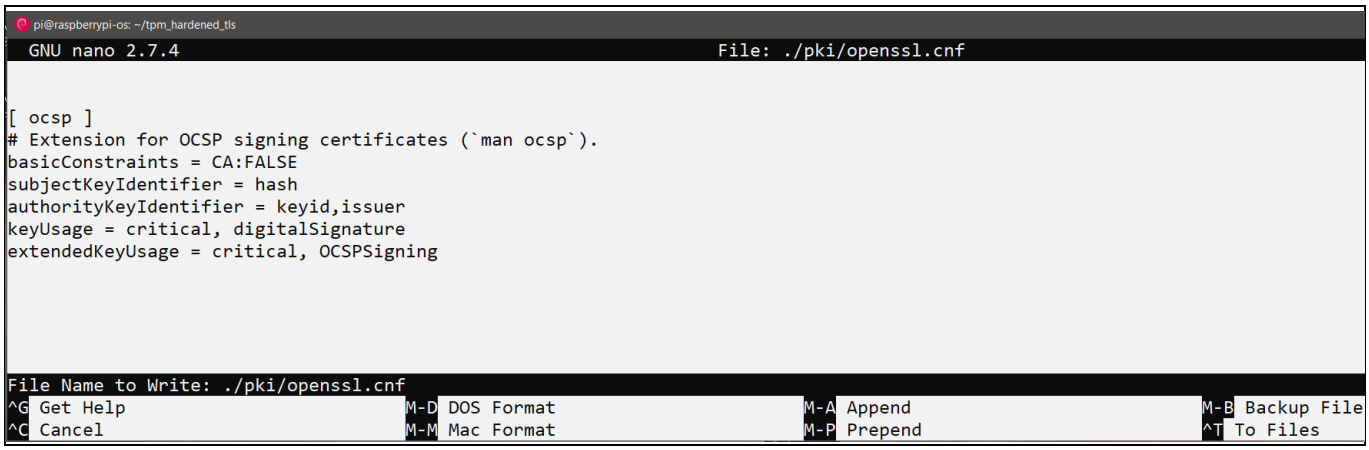

Figure 6 Copying openssl.cnf reference file

2.5.3 Creating the Root CA and Its Certificate

At the core of the PKI there is the Root CA where the chain of trust originates. In normal practice you would use an established CA like for example GlobalSign [10].

For the purpose of this application note we will use OpenSSL to create a Root Certificate Authority. This is not advised for production purposes.

Within our working directory "**tpm_hardened_tls",** create an RSA key pair.

Use OpenSSL to create the Root CA Key pair.

Code Listing 5 Create RootCA key pair

```
001       openssl genrsa -out ./pki/private/rootCA.key 2048
```

Creating a Self-Signed RootCA Certificate

Code Listing 6 Self-Signed RootCA Certificate

```
001       openssl req -config ./pki/openssl.cnf -key
   ./pki/private/rootCA.key -new -x509 -days 7300 -sha256 -extensions
   v3_ca -out ./pki/private/rootCA.crt
002       echo 1000 > ./pki/serial
```

Enter the Root CA certificate information as shown in Figure 7.

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl req -config ./pki/openssl.cnf -key ./pki/private/rootCA.key -new -x509 -days 7300 -sha256 -extensions v3_ca -out ./pki/private/rootCA.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:US
State or Province Name [California]:California
Locality Name [Milpitas]:Milpitas
Organization Name [Infineon]:Infineon
Organizational Unit Name [DSS]:DSS
Common Name []:www.infineon.com
Email Address []:
```

Figure 7 Root CA Distinguished Name or DN

Reading our RootCA

Code Listing 7

```
001        openssl x509 -in ./pki/private/rootCA.crt -noout -text
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl x509 -in ./pki/private/rootCA.crt -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            f7:f8:3e:b5:0c:ae:5a:b4
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Validity
            Not Before: Aug 16 04:10:25 2019 GMT
            Not After : Aug 11 04:10:25 2039 GMT
        Subject: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b0:7b:dd:d1:34:71:0b:e3:18:26:09:54:9c:04:
                    b9:5e:15:a3:cf:64:da:8c:62:86:57:5a:80:0c:fe:
                    7f:c9:35:a1:f9:2a:85:a4:89:df:fc:d0:15:5b:e5:
                    31:24:be:e7:b7:bb:43:95:0e:a4:3e:43:90:11:6c:
                    e6:5c:e5:91:b7:b1:15:86:25:1a:cc:50:50:f3:cf:
                    73:64:d2:2d:d3:38:01:a6:31:26:5e:cf:1f:b7:92:
                    bc:47:b9:d0:7d:c9:d5:a4:26:3f:eb:6b:03:03:21:
                    d4:65:7f:14:40:ed:2b:6e:ac:af:6c:70:36:ba:56:
                    47:b0:d7:95:eb:36:5e:a2:0c:5a:03:41:04:c6:97:
                    61:ba:22:85:cb:1f:1e:20:76:6c:78:fc:82:79:64:
                    1e:f6:06:39:9b:8f:f2:00:94:e3:7d:62:f3:78:9b:
                    f7:23:19:98:62:7b:2b:77:60:ca:0c:e4:fd:de:59:
                    4f:b4:d3:ed:59:bf:3f:95:af:fd:6c:f1:e7:51:57:
                    ee:fd:fb:85:79:a8:1b:50:39:2e:62:e7:1d:b9:86:
                    1d:2f:70:94:9c:52:bf:4c:6b:ae:99:e3:b7:48:87:
                    af:47:0f:8a:1d:33:cb:39:44:a0:b1:ed:3c:a1:4d:
                    06:94:66:1a:ee:f4:89:ba:c3:b9:35:19:f3:05:f9:
                    bd:43
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                71:0D:F7:1F:51:3C:AC:12:6C:8F:66:77:7A:99:9A:25:47:BC:96:EA
            X509v3 Authority Key Identifier:
                keyid:71:0D:F7:1F:51:3C:AC:12:6C:8F:66:77:7A:99:9A:25:47:BC:96:EA

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         9c:d5:5f:10:43:09:a1:c4:6f:25:0d:69:fe:86:b4:2e:06:ac:
         8f:5f:49:b2:14:84:c4:47:d4:2e:fb:89:03:d2:84:1a:3d:3b:
         58:53:b0:9d:5c:e4:b7:60:96:de:c5:ee:2f:64:71:31:d1:93:
         9b:2f:e8:48:03:3b:31:0a:3f:b3:02:e4:9b:15:70:b8:dd:c7:
         81:df:a0:98:fd:7a:0a:05:20:08:e0:25:4a:d7:d8:a9:5e:5d:
         b1:c0:5b:7f:18:bc:a4:94:ea:b8:41:8e:b1:75:4b:e5:5e:a1:
         95:4b:8d:24:0c:46:f7:10:95:29:58:f2:8b:d9:a2:b8:80:f1:
         f4:08:f6:23:4b:e2:64:1b:05:7a:2a:9a:0f:33:86:38:19:43:
         9d:3d:73:ef:fa:f0:b9:26:93:70:bc:10:38:2b:8a:dc:61:d6:
         bb:fd:ed:de:02:6d:41:1d:7e:65:f4:bc:05:74:28:d4:f6:df:
         b3:29:af:95:3f:d2:52:97:84:8f:5b:f6:3c:e6:a8:d7:35:be:
         43:0a:3e:a1:49:3c:95:e7:6d:ab:92:e1:16:ab:25:f1:c1:d0:
         46:46:ef:68:b8:d9:2e:84:e1:48:4c:1b:a2:97:1c:02:36:64:
         e9:7e:ba:07:70:6a:e9:ea:af:6b:dc:27:13:09:60:77:c3:c7:
         5b:28:b2:42
```

Figure 8 Root CA Certificate

2.5.4 Creating the Intermediate CA and Its Certificate

From a security perspective it is always advised to create an intermediate CA signed by the Root CA.

Create the Intermediate CA Key Pair.

Code Listing 8

```
001        openssl genrsa -out ./pki/private/intCA.key 2048
```

Create a CSR for the Intermediate CA

Code Listing 9

```
001        openssl req -config ./pki/openssl.cnf -extensions
   v3_intermediate_ca -new -sha256 -key ./pki/private/intCA.key -out
   ./pki/csr/intCA.csr
```

Enter the CSR information as shown in Figure 9.

```
pi@raspberrypi-os:~/tpm_hardened_tls $ mkdir ./pki/csr
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl req -config ./pki/openssl.cnf -extensions v3_intermediate_ca -new -sha256 -key ./pki/private/intCA.key -out ./pki/csr/intCA.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:US
State or Province Name [California]:California
Locality Name [Milpitas]:Milpitas
Organization Name [Infineon]:Infineon
Organizational Unit Name [DSS]:DSS
Common Name []:www.infineon.com
Email Address []:
```

Figure 9 Creating Intermediate CA CSR

Create the Intermediate CA certificate

Code Listing 10

```
001        openssl ca -config ./pki/openssl.cnf -extensions
   v3_intermediate_ca -days 3650 -notext -md sha256 -in
   ./pki/csr/intCA.csr -out ./pki/private/intCA.crt
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl ca -config ./pki/openssl.cnf -extensions v3_intermediate_ca -days 3650 -notext -md sha256 -in ./pki/csr/intCA.csr -out ./pki/private/intCA.crt
Using configuration from ./pki/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Aug 16 05:34:58 2019 GMT
            Not After : Aug 13 05:34:58 2029 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = California
            localityName              = Milpitas
            organizationName          = Infineon
            organizationalUnitName    = DSS
            commonName                = www.infineon.com
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                9D:C3:99:85:A8:AA:33:8A:9A:34:DA:F8:57:01:A5:BD:0B:7E:51:3E
            X509v3 Authority Key Identifier:
                keyid:5D:AC:35:6A:29:96:CD:65:3E:2F:ED:3E:67:F2:19:57:FC:F6:06:60

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until Aug 13 05:34:58 2029 GMT (3650 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
```

Figure 10 Intermediate CA Certificate Generation

Verify Signature process

Code Listing 11

```
001        openssl verify -verbose -x509_strict -CAfile
   ./pki/private/rootCA.crt ./pki/private/intCA.crt
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl verify -verbose -x509_strict -CAfile ./pki/private/rootCA.crt ./pki/private/intCA.crt
./pki/private/intCA.crt: OK
```

Figure 11 Verify Intermediate CA vs. Root CA

Reading our Intermediate CA Certificate

Code Listing 12

```
001        openssl x509 -in ./pki/private/intCA.crt -noout -text
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl x509 -in ./pki/private/intCA.crt -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4097 (0x1001)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Validity
            Not Before: Aug 16 05:34:58 2019 GMT
            Not After : Aug 13 05:34:58 2029 GMT
        Subject: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b1:34:90:e2:cb:40:13:b1:ff:8c:ee:37:76:c7:
                    b0:b3:81:72:03:d7:27:a4:f2:50:9f:da:96:19:b6:
                    b8:de:86:12:19:25:4b:b0:ab:70:70:f0:69:de:79:
                    69:ca:d3:40:12:32:dc:05:21:20:97:69:cb:de:5e:
                    3d:f1:31:d3:e2:86:e4:03:ae:cf:06:83:63:b4:1e:
                    e8:9f:fb:e4:81:df:c4:03:b7:cf:e4:a0:81:69:8e:
                    d9:f0:95:fa:03:0f:9d:2d:7b:4b:fe:44:94:7f:3a:
                    44:8f:d8:62:4e:e5:20:67:04:24:54:dc:88:2f:13:
                    51:03:dd:44:bb:39:c5:f4:c1:ab:00:db:36:81:22:
                    a1:5d:dd:78:a6:4e:59:c6:f1:63:77:5b:e1:58:11:
                    ad:a2:b3:3f:9a:c8:dd:01:80:6e:e0:d5:9f:61:7e:
                    ba:3c:ce:78:35:c3:7b:f0:e5:49:7f:90:6e:c2:96:
                    22:72:b2:40:1f:d1:43:d2:1c:e5:e0:0c:42:cb:94:
                    2e:e1:41:e1:b5:fa:6f:59:67:3e:bd:2e:25:7c:13:
                    f3:9d:d2:bb:ee:78:a9:c5:c0:56:ac:d1:e5:c2:97:
                    d3:5a:0d:42:d1:9b:41:f1:3d:dc:3c:e8:c6:b0:ac:
                    9a:96:1c:ca:8e:9b:fb:90:1a:cd:6c:63:c9:da:d4:
                    44:77
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                9D:C3:99:85:A8:AA:33:8A:9A:34:DA:F8:57:01:A5:BD:0B:7E:51:3E
            X509v3 Authority Key Identifier:
                keyid:5D:AC:35:6A:29:96:CD:65:3E:2F:ED:3E:67:F2:19:57:FC:F6:06:60

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         35:88:b4:06:32:b9:e4:86:7d:8f:6e:c5:79:4a:33:89:23:96:
         a7:f0:fe:93:4e:a3:ee:ad:18:47:26:c7:28:29:d8:11:51:7c:
         79:47:79:d3:c9:c2:2b:cb:2f:b2:cd:8d:7f:89:e7:97:e7:a4:
         eb:c1:b6:1c:fc:f1:9a:05:50:9f:18:4a:5f:5c:34:e5:37:7d:
         91:0c:5c:a7:9e:1f:f4:db:0d:2e:d7:6b:2c:01:d1:37:4a:86:
         87:c6:2b:be:b9:6e:18:16:58:f5:a2:90:f1:58:d8:f0:fa:cf:
         e9:a6:07:fc:d2:69:af:99:de:71:6b:3b:95:f3:6f:ce:3b:eb:
         9f:3d:1a:5d:10:18:b6:82:1d:62:56:8c:25:9a:aa:f7:16:86:
         ad:fd:d8:6f:23:32:32:a3:20:e1:8d:5f:fe:32:87:69:a7:81:
         12:c7:45:94:a9:fb:78:b5:df:03:04:4e:d2:4c:dd:4d:d8:8f:
         37:fc:7f:fb:62:38:22:80:14:b0:a0:38:ca:28:14:c0:cc:9b:
         17:37:80:bb:2c:44:87:f5:2f:9e:d3:cf:d6:ed:72:b5:dc:64:
         3e:b0:d7:7f:3b:84:0c:57:f7:69:7a:04:f5:95:b3:7a:9e:3f:
         52:4a:52:5c:27:c8:b3:a3:9a:57:7a:28:5d:6e:ac:a2:7d:78:
         57:8f:79:5f
```

Figure 12 Intermediate CA Certificate

Note: As mentioned before, the exercise of creating the Root CA and Intermediate CA using OpenSSL are for demonstration purposes only. In real applications these would be managed by the Certificate Authority like for example GlobalSign [10].

2.5.5 Creating Client/Endpoint Key Pair Using SLx 9670 TPM2.0

Use SLx 9670 TPM2.0 OpenSSL engine to create a TPM 2.0 key pair for the Client/Endpoint.

Note: ***As explained in Section 2.5.1, the TPM wraps (encrypts) the private key and stores it outside the TPM. The encrypted key blob [Figure 13] is encapsulated between " -----BEGIN TSS2 PRIVATE KEY-----" and "-----END TSS2 PRIVATE KEY-----"***

Create the Client/Endpoint key pair with Password Security Policy using SLx 9670 TPM2.0. Set the password to "abc". This is only for demonstration purposes. ***DO NOT USE FOR PRODUCTION***.

Code Listing 13

```
001       tpm2tss-genkey -a rsa -s 2048 ./tpm2/tpm_wrapped_keys/client.key
   -p abc
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ cat ./tpm2/tpm_wrapped_keys/client.key
-----BEGIN TSS2 PRIVATE KEY-----
MIICDwYGZ4EFCgEDoAMBAQACAQAEggEYARYAAQALAAYEcgAAABAAEAgAAAEAAQEA
9DaEJScgvUX/b6L7kk8Z3AK2QrTpoRTsaMdtOROXi7FvwpTLDKBTiy1p0KcTLUFO
y+RZoUIfA6vMMlPJJjAtnbUfw0oRaQOaNes0bSeyKoutxvoqC14013dRNrnfq3ln
qOqWIyeJDsqsH0drotLU53IvNS4d5U+2nxhYSqGl3f+mKQrX4AasvpC7O6uFk5W4
QUwAN/d7U2jaU/rg6+sV6Pql/ua+XNhKWwFkZmcEJ02k3yFPqaRXZszZ1l5jDymQ
5ra34gokVZBuTZuBU1zmsRlkZAT5fn58mJ46pGlGsdrfPBDsbh+TYfIg1FI5hwTz
u6cneTHM1MpaaXqOdpwWbQSB4ADeACBBKhT+vB+8vBtEfYALRHt3V7VEfq9aIEKp
zBslXRXYEgAQ2nKc+4RppWUyKzlIhXabmu+pF1DYShqXVR6x8Nd7yN0oNOkLj6gf
9T4xhSqXMzd51OUS6FsoSbSDdYxT/h0iQA3i/j0SM+aDsLxKPSUi3AEyDu7JihCA
pSQd/JXViydoFohdCqlnkzb/etFlmJF56UiLIiJnptpZiD8QuprVwW7w+aK9ZTz6
WWqTeX9DazMX+tm11cLlEFUrg4VplJJ3reZMykbbSmjKQd2Ab36bgq72W8uF1H7G
jHIY
-----END TSS2 PRIVATE KEY-----
```

Figure 13 Client/Endpoint Key Pair wrapped by TPM

2.5.6 Creating Client/Endpoint CSR Using SLx 9670 TPM2.0

Create the Client/Endpoint CSR using SLx 9670 TPM2.0. The key pair we just created will be managed by the SLx 9670 TPM2.0.

Note: *Remember that the private key is wrapped outside TPM 2.0 and will be loaded in order to be used for this purpose. Also, the key pair has been configured with a security policy that requires a password for the key to be used.*

Code Listing 14

```
001       openssl req -keyform engine -engine tpm2tss -config
   ./pki/openssl.cnf -key ./tpm2/tpm_wrapped_keys/client.key -new -out
   ./tpm2/csr/client.csr
```

Note: When asked for the password, input the set password "abc".

Fill the Certificate Request Information as shown in Figure 14.

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl req -keyform engine -engine tpm2tss -config ./pki/openssl.cnf -key ./tpm2/tpm_wrapped_keys/client.key -new -out ./tpm2/csr/client.csr
engine "tpm2tss" set.
Enter password for user key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:US
State or Province Name [California]:California
Locality Name [Milpitas]:Milpitas
Organization Name [Infineon]:Infineon
Organizational Unit Name [DSS]:DSS
Common Name []:www.infineon.com
Email Address []:
```

Figure 14 Client/Endpoint CSR Information

Read the Client/Endpoint CSR

Code Listing 15

```
001        openssl req -in ./tpm2/csr/client.csr -noout -text
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl req -in ./tpm2/csr/client.csr -noout -text
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:f4:36:84:25:27:20:bd:45:ff:6f:a2:fb:92:4f:
                    19:dc:02:b6:42:b4:e9:a1:14:ec:68:c7:6d:39:13:
                    97:8b:b1:6f:c2:94:cb:0c:a0:53:8b:2d:69:d0:a7:
                    13:2d:41:4e:cb:e4:59:a1:42:1f:03:ab:cc:32:53:
                    c9:26:30:2d:9d:b5:1f:c3:4a:11:69:03:9a:35:eb:
                    34:6d:27:b2:2a:8b:ad:c6:fa:2a:0b:5e:34:d7:77:
                    51:36:b9:df:ab:79:67:a8:ea:96:23:27:89:0e:ca:
                    ac:1f:47:6b:a2:d2:d4:e7:72:2f:35:2e:1d:e5:4f:
                    b6:9f:18:58:4a:a1:a5:dd:ff:a6:29:0a:d7:e0:06:
                    ac:be:90:bb:3b:ab:85:93:95:b8:41:4c:00:37:f7:
                    7b:53:68:da:53:fa:e0:eb:eb:15:e8:fa:a5:fe:e6:
                    be:5c:d8:4a:5b:01:64:66:67:04:27:4d:a4:df:21:
                    4f:a9:a4:57:66:cc:d9:d6:5e:63:0f:29:90:e6:b6:
                    b7:e2:0a:24:55:90:6e:4d:9b:81:53:5c:e6:b1:19:
                    64:64:04:f9:7e:7e:7c:98:9e:3a:a4:69:46:b1:da:
                    df:3c:10:ec:6e:1f:93:61:f2:20:d4:52:39:87:04:
                    f3:bb:a7:27:79:31:cc:d4:ca:5a:69:7a:8e:76:9c:
                    16:6d
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         10:11:35:51:f5:d9:ae:2d:3e:ae:ea:e9:69:b0:03:06:87:c2:
         b9:5f:26:f2:3e:d9:99:f0:b8:d3:2e:e2:da:cd:91:7d:7f:c3:
         48:3a:37:5a:ca:43:2d:a9:ce:b5:ca:05:c6:cb:ff:a7:56:ac:
         5d:32:e9:7e:fb:28:95:d1:52:1f:4f:73:dd:68:f9:85:eb:40:
         17:a0:b7:1c:ac:f7:17:2b:1f:d9:2b:94:80:7a:aa:f7:52:bb:
         48:f2:65:82:af:f1:7a:2e:aa:da:da:b0:56:4f:2a:e6:c0:ff:
         39:a5:d4:b6:27:b5:50:8b:e5:16:a5:16:89:23:47:de:75:54:
         67:f5:e0:cc:38:87:62:cf:23:47:87:b1:23:cb:8c:0a:f4:99:
         04:9e:8a:25:7d:77:2a:66:0a:f0:bb:41:33:d9:d2:cd:fe:de:
         16:0c:ad:42:e0:c9:9f:f6:d4:60:a6:de:b4:57:bd:22:8a:29:
         dd:d5:3f:33:ae:1a:8c:b1:38:07:d7:92:db:99:39:fa:c0:0c:
         83:eb:78:0f:d5:81:46:21:62:99:93:00:0e:74:83:b1:86:ce:
         fe:74:3c:5f:11:56:f6:15:c9:81:06:e3:e1:f7:22:22:50:a4:
         2a:73:76:a5:e2:fd:c2:9d:15:7a:27:9e:47:0c:17:79:22:6e:
         98:96:2e:4c
```

Figure 15 Client/Endpoint CSR

2.5.7 Signing Client/Endpoint CSR with RootCA

Use the Intermediate CA to sign the created Client/Endpoint CSR

Code Listing 16

```
001        openssl x509 -req -days 365 -in ./tpm2/csr/client.csr -CA
   ./pki/private/intCA.crt -CAkey ./pki/private/intCA.key -CAcreateserial
   -out ./tpm2/certs/client.crt
```

Figure 16 Signing Client CSR with Intermediate CA

To verify the certificate chain, we use the parameter -untrusted for the Intermediate CA certificate file. This is the parameter OpenSSL has assigned for this verification.

Code Listing 17

```
001        openssl verify -CAfile ./pki/private/rootCA.crt -untrusted
   ./pki/private/intCA.crt ./tpm2/certs/client.crt
```

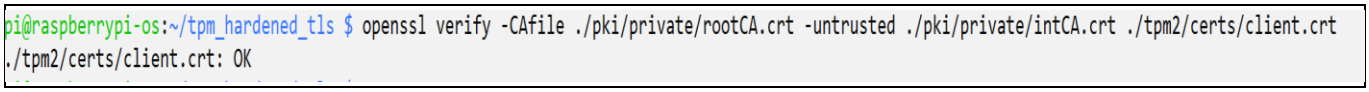

Figure 17 Client Certificate Chain verification

Read the Client/Endpoint Certificate

Code Listing 18

```
001        openssl x509 -in ./tpm2/certs/client.crt -noout -text
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl x509 -in ./tpm2/certs/client.crt -noout -text
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ae:cb:a0:21:a1:9d:43:59
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Validity
            Not Before: Aug 16 06:00:29 2019 GMT
            Not After : Aug 15 06:00:29 2020 GMT
        Subject: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:f4:36:84:25:27:20:bd:45:ff:6f:a2:fb:92:4f:
                    19:dc:02:b6:42:b4:e9:a1:14:ec:68:c7:6d:39:13:
                    97:8b:b1:6f:c2:94:cb:0c:a0:53:8b:2d:69:d0:a7:
                    13:2d:41:4e:cb:e4:59:a1:42:1f:03:ab:cc:32:53:
                    c9:26:30:2d:9d:b5:1f:c3:4a:11:69:03:9a:35:eb:
                    34:6d:27:b2:2a:8b:ad:c6:fa:2a:0b:5e:34:d7:77:
                    51:36:b9:df:ab:79:67:a8:ea:96:23:27:89:0e:ca:
                    ac:1f:47:6b:a2:d2:d4:e7:72:2f:35:2e:1d:e5:4f:
                    b6:9f:18:58:4a:a1:a5:dd:ff:a6:29:0a:d7:e0:06:
                    ac:be:90:bb:3b:ab:85:93:95:b8:41:4c:00:37:f7:
                    7b:53:68:da:53:fa:e0:eb:eb:15:e8:fa:a5:fe:e6:
                    be:5c:d8:4a:5b:01:64:66:67:04:27:4d:a4:df:21:
                    4f:a9:a4:57:66:cc:d9:d6:5e:63:0f:29:90:e6:b6:
                    b7:e2:0a:24:55:90:6e:4d:9b:81:53:5c:e6:b1:19:
                    64:64:04:f9:7e:7e:7c:98:9e:3a:a4:69:46:b1:da:
                    df:3c:10:ec:6e:1f:93:61:f2:20:d4:52:39:87:04:
                    f3:bb:a7:27:79:31:cc:d4:ca:5a:69:7a:8e:76:9c:
                    16:6d
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         62:cf:5f:ef:fb:b0:a5:9b:fb:48:72:fe:4a:64:a7:91:7d:76:
         b8:18:da:b7:a4:3e:31:80:18:31:7e:a8:3d:6c:5e:03:35:d1:
         1b:96:ed:52:7d:1a:27:48:49:3c:bd:62:33:80:7c:b2:26:de:
         53:94:06:db:23:a8:b9:40:4e:c6:67:b0:28:46:0c:8e:79:6c:
         b5:0f:e3:fa:9d:0e:db:16:37:2a:05:3b:65:26:1f:8b:0b:da:
         f5:ec:19:a9:ee:3f:4a:3f:1d:e5:fa:f6:76:b1:a3:19:c8:aa:
         ad:df:ae:5d:0c:ce:89:56:2a:b5:c5:57:8e:a5:19:4f:85:84:
         07:65:fb:e7:0b:ed:ff:02:e1:0a:32:37:d3:68:e6:d3:96:94:
         82:db:b1:a1:51:4f:8b:bd:b6:97:2a:23:8c:a5:b5:2e:96:eb:
         61:87:6d:f9:7f:54:c0:36:a2:70:03:6b:88:cf:58:1d:8e:87:
         71:34:6e:d0:17:db:2e:88:f1:ea:e7:9b:ca:45:7e:12:20:71:
         9b:bd:c7:d9:1e:bf:25:79:93:80:41:37:89:b3:37:9d:cf:57:
         db:ff:d8:1c:4d:e3:dc:7a:d7:38:46:80:18:b0:21:16:7e:51:
         0d:dc:64:68:d3:b7:d9:e1:9e:7c:7a:a9:93:64:3f:b4:85:95:
         6d:eb:7c:79
```

Figure 18 Client/Endpoint Certificate

The just created certificate will be used as part of the TLS handshake process by the client to authenticate the server.

2.5.8 Creating Server Certificate

Now that we have created our Client/Endpoint key pair and certificate using SLx 9670 TPM2.0, we will replicate the process to create the needed certificate for the server.

Code Listing 19

```
001        tpm2tss-genkey -a rsa -s 2048 ./tpm2/tpm_wrapped_keys/server.key
   -p abc
002        openssl req -keyform engine -engine tpm2tss -config
   ./pki/openssl.cnf -key ./tpm2/tpm_wrapped_keys/server.key -new -out
   ./tpm2/csr/server.csr
003        openssl x509 -req -days 365 -in ./tpm2/csr/server.csr -CA
   ./pki/private/intCA.crt -CAkey ./pki/private/intCA.key -CAcreateserial
   -out ./tpm2/certs/server.crt
004        openssl x509 -in ./tpm2/certs/server.crt -noout -text
```

Reading Server Certificate

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl x509 -in ./tpm2/certs/server.crt -noout -text
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ae:cb:a0:21:a1:9d:43:5a
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Validity
            Not Before: Aug 16 06:21:23 2019 GMT
            Not After : Aug 15 06:21:23 2020 GMT
        Subject: C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ba:45:99:4a:b5:23:b5:05:2d:01:c1:5a:37:e3:
                    17:9b:eb:4e:98:dc:36:63:bb:e9:9b:14:9f:76:88:
                    9d:15:25:8e:74:3e:b6:68:07:c1:e2:d8:31:55:82:
                    b5:c2:b5:43:c9:4e:f1:82:f4:fe:82:a7:30:f8:2f:
                    65:dc:0f:89:5a:87:58:35:69:4f:73:cb:89:46:98:
                    e2:25:de:05:75:63:31:61:e4:6e:5e:11:a6:a5:1b:
                    f9:46:7c:06:e3:95:f8:df:39:5a:6d:28:37:0c:94:
                    63:bf:c5:e3:1e:75:4f:9f:e2:a2:cb:3e:d0:bf:0c:
                    36:4f:22:60:20:90:62:2e:c9:d4:8e:4f:26:5f:5e:
                    a6:10:5a:f2:9b:9c:5a:a4:c0:25:d5:c0:88:cd:8f:
                    18:10:e1:32:9d:c3:d1:46:6f:41:c4:ef:be:f0:e9:
                    94:79:32:ae:51:27:9c:c4:69:5d:1e:4e:0c:fb:bb:
                    01:be:cc:72:5a:63:e5:53:3d:04:ef:02:e3:39:66:
                    07:be:87:4a:b9:17:5d:f7:ab:84:dc:d3:73:4b:06:
                    03:96:c5:e5:ff:a0:73:d6:04:4e:d1:9d:bb:d1:3e:
                    b0:3e:91:cd:77:82:27:7d:c0:9b:21:47:a7:0d:59:
                    18:ab:eb:1b:4d:f2:f7:eb:ef:40:bb:08:83:28:37:
                    10:1d
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         af:97:ac:2b:20:d0:6e:f2:23:e0:c1:fb:fa:f1:bd:21:92:9c:
         f8:40:d9:fa:33:38:9b:7a:55:62:94:7e:69:8a:63:6c:8d:be:
         84:00:34:f4:74:49:26:c8:dd:c9:c2:7b:87:b7:87:59:14:c2:
         1a:69:ef:d3:e8:7b:3f:62:b7:c5:a2:65:1e:39:44:44:d1:c3:
         50:23:53:d2:d2:ab:d0:11:bb:96:a5:08:48:a3:f5:13:8e:0f:
         f2:07:ea:a5:5f:94:20:cb:66:35:34:da:96:1d:74:f8:09:5b:
         c3:fb:61:df:dc:1b:5c:4c:cf:dc:34:bb:9b:5d:38:b4:98:c9:
         4f:1c:07:53:db:41:68:4b:a3:7e:c2:52:e4:bf:58:3b:d3:cf:
         43:1d:11:6e:e0:7d:53:68:2d:8f:e4:c3:ef:d4:5f:a4:58:d1:
         50:a6:fe:08:02:31:61:7c:68:59:ff:c5:d8:da:d9:28:12:17:
         85:d1:de:70:e8:48:95:82:04:d4:07:f7:b8:67:96:51:14:36:
         ca:cd:2b:68:60:90:1f:bb:ce:94:4c:a1:a6:b0:85:a9:81:72:
         fc:a9:dd:4a:ef:d2:f3:47:2d:72:3d:ab:ee:c6:f2:0a:3c:df:
         cd:ce:76:80:a1:ee:19:0b:93:42:05:bc:45:28:54:94:21:ed:
         18:1b:23:8b
```

Figure 19 Server Certificate

Verifying the Certificate chain

Code Listing 20

```
001        openssl verify -CAfile ./pki/private/rootCA.crt -untrusted
   ./pki/private/intCA.crt ./tpm2/certs/server.crt
```

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl verify -CAfile ./pki/private/rootCA.crt -untrusted ./pki/private/intCA.crt ./tpm2/certs/server.crt
./tpm2/certs/server.crt: OK
```

Figure 20 Client Certificate Chain verification

2.6 Creating an OpenSSL S_Server

We will create now an OpenSSL server. For this purpose, we are using the local host capabilities to run this example on the same Linux machine.

Create an openssl S_Server instance using a terminal window session.

Code Listing 21 OpenSSL S_Server terminal window

```
001       openssl s_server -www -Verify 1 -cert ./tpm2/certs/server.crt -
   key ./tpm2/tpm_wrapped_keys/server.key -keyform engin -engine tpm2tss -
   accept 127.0.0.1:8444
```

Note: When ask for the password to use the key, input the set password during key creation "abc".

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl s_server -www -Verify 1 -cert ./tpm2/certs/server.crt -key ./tpm2/tpm_wra
pped_keys/server.key -keyform engin -engine tpm2tss -accept 127.0.0.1:8444
verify depth is 1, must return a certificate
engine "tpm2tss" set.
Enter password for user key:
Using default temp DH parameters
ACCEPT
```

Figure 21 OpenSSL S_Server

2.7 Creating an OpenSSL S_Client

We will create an OpenSSL S_Client and connect through a TLS session with OpenSSL S_Server (The two terminal windows and services running on the same Linux machine).

Open a new terminal window and go to our root directory for this exercise (tpm_hardened_tls) and run the following command.

Create an OpenSSL S_Client.

Code Listing 22 OpenSSL S_Client terminal window

```
001       openssl s_client -cert ./tpm2/certs/client.crt -key
   ./tpm2/tpm_wrapped_keys/client.key -keyform engine -engine tpm2tss -
   connect localhost:8444
```

Note: Note that both S_Server and S_Client will be using TPM2-TSS Engine alongside SLx 9670 TPM2.0 to interact and establish a TLS session.

The output of the connection is divided in two parts

a) The TLS handshake

b) TLS Cipher

*Note: When using self-signed certificates in OpenSSL, as a precaution it will give a warning stated as an "error": **Verification error: self-signed certificate in certificate chain.***

To verify that our certificate chain is valid we can run the following commands

As shown in Figure 22 and Figure 23 the complete TLS handshake process was successful, and the encrypted channel established.

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl s_client -cert ./tpm2/certs/client.crt -key ./tpm2/tpm_wrapped_keys/client.key -keyf
orm engine -engine tpm2tss -connect localhost:8444
engine "tpm2tss" set.
Enter password for user key:
CONNECTED(00000004)
depth=0 C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
   i:/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDXjCCAkYCCQCuy6AhoZ1DWjANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQGEwJV
UzETMBEGA1UECAwKQ2FsaWZvcm5pYTERMA8GA1UEBwwITWlscGl0YXMxETAPBgNV
BAoMCEluZmluZW9uMQwwCgYDVQQLDANEU1MxGTAXBgNVBAMMEHd3dy5pbmZpbmVv
bi5jb20wHhcNMTkwODE2MDYyMTIzWhcNMjAwODE1MDYyMTIzWjBxMQswCQYDVQQG
EwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTERMA8GA1UEBwwITWlscGl0YXMxETAP
BgNVBAoMCEluZmluZW9uMQwwCgYDVQQLDANEU1MxGTAXBgNVBAMMEHd3dy5pbmZp
bmVvbi5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC6RZlKtSO1
BS0BwVo34xeb606Y3DZju+mbFJ92iJ0VJY50PrZoB8Hi2DFVgrXCtUPJTvGC9P6C
pzD4L2XcD4lah1g1aU9zy4lGmOIl3gV1YzFh5G5eEaalG/lGfAbjlfjfOVptKDcM
lGO/xeMedU+f4qLLPtC/DDZPImAgkGIuydSOTyZfXqYQWvKbnFqkwCXVwIjNjxgQ
4TKdw9FGb0HE777w6ZR5Mq5RJ5zEaV0eTgz7uwG+zHJaY+VTPQTvAuM5Zge+h0q5
F133q4Tc03NLBgOWxeX/oHPWBE7RnbvRPrA+kc13gid9wJshR6cNWRir6xtN8vfr
70C7CIMoNxAdAgMBAAEwDQYJKoZIhvcNAQELBQADggEBAK+XrCsg0G7yI+DB+/rx
vSGSnPhA2fozOJt6VWKUfmmKY2yNvoQANPR0SSbI3cnCe4e3h1kUwhpp79Poez9i
t8WiZR45RETRw1AjU9LSq9ARu5alCEij9ROOD/IH6qVflCDLZjU02pYddPgJW8P7
Yd/cG1xMz9w0u5tdOLSYyU8cB1PbQWhLo37CUuS/WDvTz0MdEW7gfVNoLY/kw+/U
X6RY0VCm/ggCMWF8aFn/xdja2SgSF4XR3nDoSJWCBNQH97hnllEUNsrNK2hgkB+7
zpRMoaawhamBcvyp3Urv0vNHLXI9q+7G8go8383OdoCh7hkLk0IFvEUoVJQh7Rgb
I4s=
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
issuer=/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
---
No client certificate CA names sent
Client Certificate Types: RSA sign, DSA sign, ECDSA sign
Requested Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+SHA256:ECDSA+S
HA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+SHA256:
ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA512
Server Temp Key: X25519, 253 bits
```

Figure 22 OpenSSL S_Client and S_Server TLS Handshake hardened by OPTIGA™ SLx 9670 TPM2.0

```
No client certificate CA names sent
Client Certificate Types: RSA sign, DSA sign, ECDSA sign
Requested Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+SHA256:ECDSA+S
HA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+SHA256:
ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA512
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 2418 bytes and written 1419 bytes
Verification error: self signed certificate
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 4823FB3B980D967741E0542880122ABDBF8D9522BF49002CF020AE05FAFB2AF8
    Session-ID-ctx:
    Master-Key: 7619BC4804F42AE5AB078DCA7F6B2F81072087CC4074807BFF46D3758515B4F910511BD506041144F07AB23538BC439D
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - 17 a1 5e d5 f4 aa 9b 8a-4f 3a ac 85 41 51 99 6e   ..^.....O:..AQ.n
    0010 - ca f6 96 bb 25 55 ee f3-78 2c 1d 45 74 cf d7 54   ....%U..x,.Et..T
    0020 - 66 9f cd a1 cd fa 35 21-0a 6b e3 4d a8 d3 c6 2f   f.....5!.k.M.../
    0030 - 92 c2 5f 1b b3 97 e3 cc-36 a5 2f 6a 11 79 bb b5   .._.....6./j.y..
    0040 - 89 91 4b 9a 6a e4 e1 88-14 04 93 7a 3f b0 d3 96   ..K.j......z?...
    0050 - 31 18 87 18 4a a3 1b ad-fa bf ba 85 05 55 c6 5b   1...J........U.[
    0060 - 6f 3c 15 17 33 c8 6b a3-db 41 c8 2f 1a 34 82 2d   o<..3.k..A./.4.-
    0070 - ec 78 3e bf 5d 77 ff c5-13 37 da fe 88 ec 84 b6   .x>.]w...7......
    0080 - 63 5e 65 28 a8 39 13 a0-b1 2e d0 fe 6c 6a 60 ca   c^e(.9......lj`.
    0090 - 27 a5 5c 2e 41 89 ca 9b-b4 fc 52 54 3c dc 7a 01   '.\.A.....RT<.z.
    00a0 - 10 f0 8a c8 10 2e 8f ae-49 73 65 51 66 dd 23 53   ........IseQf.#S
    00b0 - b1 81 86 bd 27 4a 2c dd-cd 6f 08 bc c6 d7 19 6a   ....'J,..o.....j
    00c0 - b9 1b f2 24 12 ab 27 5f-d7 ca ac 5d 10 8b fd 44   ...$..'_...]...D
    00d0 - d3 38 97 d7 6f 04 71 06-35 54 7d 71 3e 5c e6 16   .8..o.q.5T}q>\..
    00e0 - 5e 97 f6 7f 17 1a 32 af-be 9a f1 82 51 1b dd 65   ^.....2.....Q..e
    00f0 - ca d7 77 e4 0a dc 9c cd-12 3c 1c 74 1a 3d 5e fe   ..w......<.t.=^.
    0100 - 6e de a2 80 cb 16 f3 f7-7f 22 7d 72 02 98 5a 5e   n........"}r..Z^
    0110 - 9d 86 94 72 21 be 9f 9a-27 ff 37 2f 69 37 d3 32   ...r!...'.7/i7.2
    0120 - cd 91 c6 de 57 55 a0 1b-d7 8a 3f 85 00 d6 81 25   ....WU....?....%
    0130 - 38 56 16 7d e2 63 c7 a6-9c 92 ac 3a 69 ec b3 7b   8V.}.c.....:i..{
    0140 - cc 90 ba fd 5f 0b 80 42-40 c7 48 ed 8c b5 96 5f   ...._..B@.H...._
    0150 - 61 8b ab f4 fc 95 2a ae-2a df 74 74 29 c2 52 f8   a.....*.*.tt).R.
    0160 - f3 3e 39 46 cf 2c 48 e0-4d 17 86 93 4f bd 60 0c   .>9F.,H.M...O.`.
    0170 - ae fe 7a 77 9e 07 7c 3a-32 54 f9 0c 2e 7d 46 5a   ..zw..|:2T...}FZ
    0180 - 18 06 2b d8 e2 ff 65 af-4b 4e eb e1 23 07 a8 59   ..+...e.KN..#..Y
    0190 - 0f 8f 5a 8e f4 c3 cd d0-50 56 11 ec 02 c6 06 da   ..Z.....PV......
    01a0 - c9 f4 db e9 b2 f8 d5 94-4d 08 9e e8 2a 9e 1c 0d   ........M...*...
    01b0 - d6 92 dd e5 c8 f3 e7 6c-e5 b5 3f 8c 66 50 d1 d4   .......l..?.fP..
    01c0 - e0 b9 fa 1f b7 34 5a 3a-e3 bb 4b 5e 17 14 5f 47   .....4Z:..K^.._G
    01d0 - e0 ff 8a 57 31 85 84 91-29 7c 0c bf b5 42 65 bf   ...W1...)|...Be.
    01e0 - a2 a9 ca 9e 02 18 de 3e-14 d2 e8 84 fb f3 73 30   .......>......s0
    01f0 - 93 98 8e 70 a7 dd 52 94-1a 35 5c f6 34 27 18 c3   ...p..R..5\.4'..
    0200 - 25 45 76 a6 10 c1 8a d4-c7 2d 4c 33 48 a9 d6 f6   %Ev......-L3H...
```

Figure 23 OpenSSL S_Client and S_Server TLS Cipher

As a summary, this was the process that followed the TLS session.

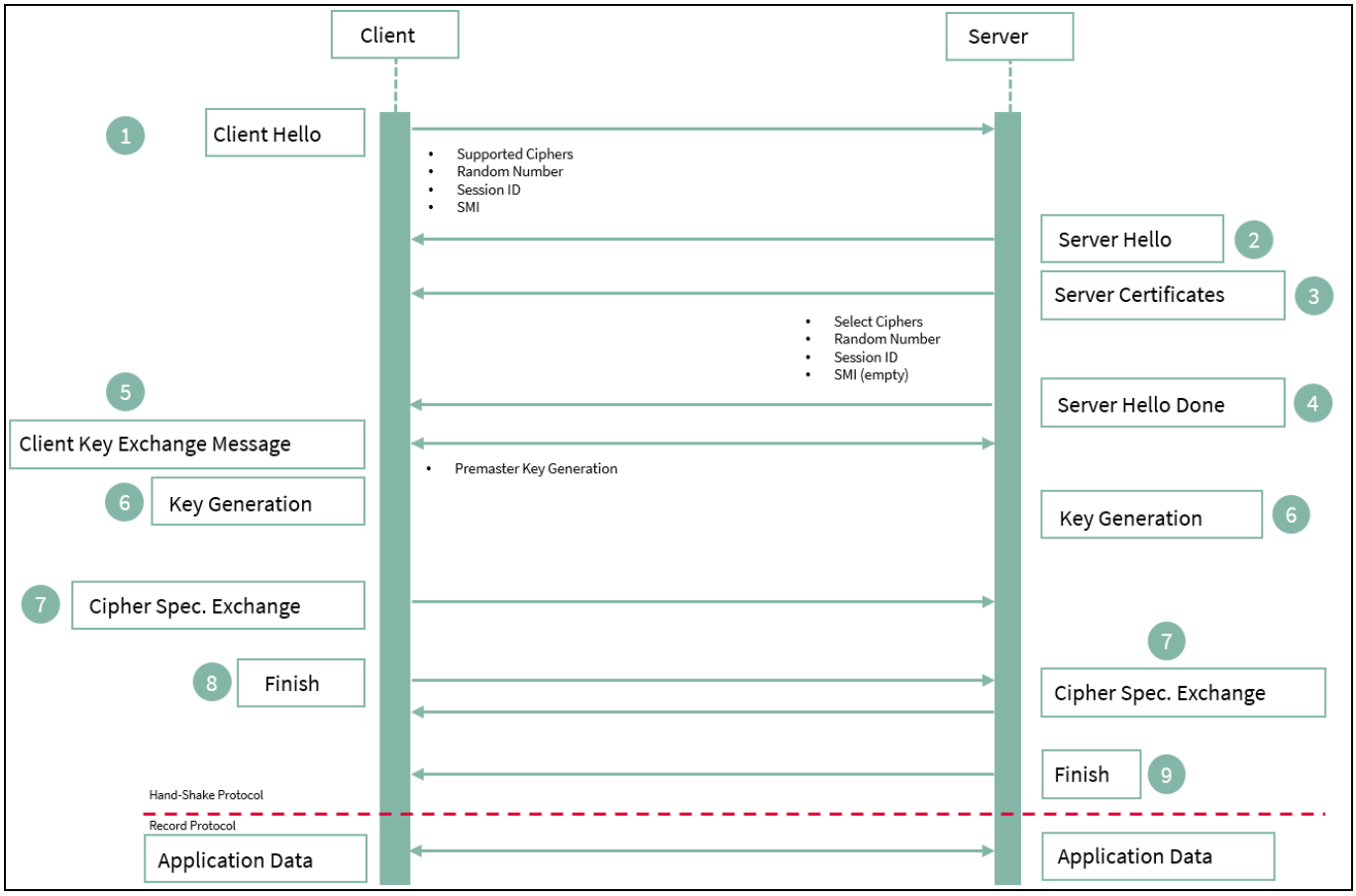

Figure 24 OpenSSL S_Client and S_Server TLS session flow

- **Client Hello**: The Client Hello is the first message in the TLS handshake from the client to the server. The Client Hello message includes the highest version of the TLS protocol the client supports, a random number generated by the client, cipher suites and the compression algorithm supported by the client, and an optional session identifier.
- **Server Hello** message is the response from the Server once it receives the Client Hello. The Server Hello is the first message from the server to the client. To be precise, the Server Hello is the first message from the server to the client, which is generated at the TLS layer. The Server Hello message includes the highest version of TLS protocol that both the client and the server can support, a random number generated by the server, the strongest cipher suite, and the compression algorithm that both the client and the server can support. The Server can use the TPM as source of entropy, that is, use TPM to generate the random number.
- **Server Certificates**: After the Server Hello message is sent to the client, the server sends its public certificate, along with other certificates, up to the root certificate authority (CA) in the certificate chain. The client must validate these certificates to accept the identity of the server.
- **Key Generation**: At this point, the client and the server have exchanged all the required materials to generate the master secret. The master secret is generated using the client random number, the server random number, and the premaster secret.
- **Change Cipher Spec:** Message to the server to indicates that all messages generated from here onward are protected with the keys already established.

- **Client Finish Message**: Last message from the client to the server. It's the hash of the complete message flow in the TLS handshake encrypted by the already-established keys. Once the server receives the Finished message from the client, it responds back with the **Change Cipher Spec Message**. This indicates to the client that the server is ready to start communicating with the secret keys already established.
- **Server Finish Message**: This is like the Finished message generated by the client and includes the hash of the complete message flow in the handshake encrypted by the generated cryptographic keys. This completes the TLS handshake and here onward both the client and the server can send data over an encrypted channel.

Note: When the server demands TLS mutual authentication, then the server will request the client certificate. The client certificate request message from the server includes a list of certificate authorities trusted by the server and the type of the certificate. After that, the server sends the Server Hello Done message to the client. This is an empty message that only indicates to the client that the server has completed its initial phase in the handshake.

If the server demands the client certificate, now the client sends its public certificate along with all other certificates in the chain up to the root certificate authority (CA) required to validate the client certificate.

- **Client Key Exchange Message**: After the Server Hello message is sent to the client, the server sends its public certificate, along with other certificates, up to the root certificate authority (CA) in the certificate chain. The client must validate these certificates to accept the identity of the server.

In the next section will use TShark as a sniffer to capture and log the TLS session. This tool will enable us to see the actual process that is followed as described in Figure 24.

3 Decoding SSL/TLS Traffic using TShark

TShark is a network protocol analyzer. It lets the capture of data packets from a live network or read packets from a previously save captured.

TShark native format of captured files is "**pcapng**", which is also the format used by WireShark[8].

We will be using TShark to decode the traffic between the S_Server and S_Client. This will allow to monitor and follow the TLS handshake exchange.

3.1 Installing TShark

Execute the following commands to install TShark on Raspberry Pi® 3B+/4 and verify that it installed correctly.

Code Listing 23

```
001        sudo apt-get update
002        sudo apt-get install tshark -y
003        tshark -v
```

```
pi@raspberrypi-os:~/tpm_sserver $ tshark -v
TShark (Wireshark) 2.6.7 (Git v2.6.7 packaged as 2.6.7-1~deb9u1)
```

Figure 25 TShark Install

3.2 Available Network Interfaces to use with TShark

Displaying available network interfaces that TShark can use

Code Listing 24

```
001        sudo tshark -D
```

```
pi@raspberrypi-os:~/tpm_sserver $ sudo tshark -D
Running as user "root" and group "root". This could be dangerous.
1. eth0
2. wlan0
3. any
4. lo (Loopback)
5. bluetooth0
6. nflog
7. nfqueue
8. usbmon1
9. ciscodump (Cisco remote capture)
10. randpkt (Random packet generator)
11. sshdump (SSH remote capture)
12. udpdump (UDP Listener remote capture)
```

Figure 26 Network Interface detected in Raspberry Pi® 3B+/4

3.3 Testing the Capture of Network Traffic with TShark

To test that all is working before moving on, we will capture a loopback transaction using the available Local Host: 127.0.0.1.

Open an additional terminal window (additional to the one we are using for TShark)

Code Listing 25 TShark terminal window

```
001        tshark -i 4
```

Code Listing 26 Terminal window 2

```
001        ping 127.0.0.1
```

To escape the capturing by TShark and the ping sequence use Linux terminal escape sequence "**ctrl + c**"

```
pi@raspberrypi-os:~/tpm_sserver $ tshark -i 4
Capturing on 'Loopback'
    1 0.000000000    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) request  id=0x04b2, seq=1/256, ttl=64
    2 0.000113953    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) reply    id=0x04b2, seq=1/256, ttl=64 (request in 1)
    3 1.043414058    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) request  id=0x04b2, seq=2/512, ttl=64
    4 1.043463379    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) reply    id=0x04b2, seq=2/512, ttl=64 (request in 3)
    5 2.083403746    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) request  id=0x04b2, seq=3/768, ttl=64
    6 2.083453379    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) reply    id=0x04b2, seq=3/768, ttl=64 (request in 5)
    7 3.123400772    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) request  id=0x04b2, seq=4/1024, ttl=64
    8 3.123447176    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) reply    id=0x04b2, seq=4/1024, ttl=64 (request in 7)
    9 4.163401855    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) request  id=0x04b2, seq=5/1280, ttl=64
   10 4.163446227    127.0.0.1 → 127.0.0.1    ICMP 98 Echo (ping) reply    id=0x04b2, seq=5/1280, ttl=64 (request in 9)
^C10 packets captured
```

Figure 27 TShark capturing terminal window

At the end of the capture TShark will indicate the number of packets that were captured. In this case 10.

```
pi@raspberrypi-os:~ $ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.194 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.113 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.112 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.102 ms
^C
--- 127.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4163ms
rtt min/avg/max/mdev = 0.102/0.128/0.194/0.035 ms
```

Figure 28 Terminal window 2

3.4 Capturing a TLS Session using TShark

Within the working directory (**tpm_hardened_tls**), create a new directory name for example "**tls_log**".

Additionally, we will open three terminal windows:

1. TShark terminal window
2. S_Server terminal window
3. S_Client terminal window

Code Listing 27 TShark terminal window

```
001        mkdir tls_log
002        cd tls_log
```

Open the S_Server as previously demonstrated [Code Listing 21].

Note: Remember that we are using TPM password policy to secure the use of the private key. The password is "abc"

Code Listing 28 S_Server terminal window

```
001        openssl s_server -www -Verify 1 -cert ./tpm2/certs/server.crt -
   key ./tpm2/tpm_wrapped_keys/server.key -keyform engine -engine tpm2tss
   -accept 127.0.0.1:8444
```

```
pi@raspberrypi-os:~/tpm_hardened_tls/tls_log $ openssl s_server -www -Verify 1 -cert ../tpm2/certs/server.crt -key ../tp
m2/tpm_wrapped_keys/server.key -keyform engin -engine tpm2tss -accept 127.0.0.1:8444
verify depth is 1, must return a certificate
engine "tpm2tss" set.
Enter password for user key:
Using default temp DH parameters
ACCEPT
```

Figure 29 Opening S_Server

On the TShark terminal window we need to start the capturing task.

Tls.pcap will be the log file created by TShark

Code Listing 29 TShark terminal window

```
001        tshark -s0 -w tls.pcap -i 4
```

```
pi@raspberrypi-os:~/tpm_sserver/tls_log $ tshark -s0 -w tls.pcap -i 4
Capturing on 'Loopback'
```

Figure 30 Start of TShark capture task

In our workspace root directory (where OpenSSL conf file and self-signed certificate are located). Open the S_Client as previously demonstrated [Code Listing 22]

Code Listing 30 S_Client terminal window

```
001       openssl s_client -cert ./tpm2/certs/client.crt -key
   ./tpm2/tpm_wrapped_keys/client.key -keyform engine -engine tpm2tss -
   connect localhost:8444
```

We will get the same interaction as in our previous section [Code Listing 22]

```
pi@raspberrypi-os:~/tpm_hardened_tls $ openssl s_client -cert ./tpm2/certs/client.crt -key ./tpm2/tpm_wrapped_keys/client.key -keyf
orm engine -engine tpm2tss -connect localhost:8444
engine "tpm2tss" set.
Enter password for user key:
CONNECTED(00000004)
depth=0 C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = California, L = Milpitas, O = Infineon, OU = DSS, CN = www.infineon.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
   i:/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDXjCCAkYCCQCuy6AhoZ1DWjANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQGEwJV
UzETMBEGA1UECAwKQ2FsaWZvcm5pYTERMA8GA1UEBwwITWlscGl0YXMxETAPBgNV
BAoMCEluZmluZW9uMQwwCgYDVQQLDANEU1MxGTAXBgNVBAMMEHd3dy5pbmZpbmVv
bi5jb20wHhcNMTkwODE2MDYyMTIzWhcNMjAwODE1MDYyMTIzWjBxMQswCQYDVQQG
EwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTERMA8GA1UEBwwITWlscGl0YXMxETAP
BgNVBAoMCEluZmluZW9uMQwwCgYDVQQLDANEU1MxGTAXBgNVBAMMEHd3dy5pbmZp
bmVvbi5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC6RZlKtSO1
BS0BwVo34xeb606Y3DZju+mbFJ92iJ0VJY50PrZoB8Hi2DFVgrXCtUPJTvGC9P6C
pzD4L2XcD4lah1g1aU9zy4lGmOIl3gV1YzFh5G5eEaalG/lGfAbjlfjfOVptKDcM
lGO/xeMedU+f4qLLPtC/DDZPImAgkGIuydSOTyZfXqYQWvKbnFqkwCXVwIjNjxgQ
4TKdw9FGb0HE777w6ZR5Mq5RJ5zEaV0eTgz7uwG+zHJaY+VTPQTvAuM5Zge+h0q5
F133q4Tc03NLBgOWxeX/oHPWBE7RnbvRPrA+kc13gid9wJshR6cNWRir6xtN8vfr
70C7CIMoNxAdAgMBAAEwDQYJKoZIhvcNAQELBQADggEBAK+XrCsg0G7yI+DB+/rx
vSGSnPhA2fozOJt6VWKUfmmKY2yNvoQANPR0SSbI3cnCe4e3h1kUwhpp79Poez9i
t8WiZR45RETRw1AjU9LSq9ARu5alCEij9ROOD/IH6qVflCDLZjU02pYddPgJW8P7
Yd/cG1xMz9w0u5tdOLSYyU8cB1PbQWhLo37CUuS/WDvTz0MdEW7gfVNoLY/kw+/U
X6RY0VCm/ggCMWF8aFn/xdja2SgSF4XR3nDoSJWCBNQH97hnllEUNsrNK2hgkB+7
zpRMoaawhamBcvyp3Urv0vNHLXI9q+7G8go8383OdoCh7hkLk0IFvEUoVJQh7Rgb
I4s=
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
issuer=/C=US/ST=California/L=Milpitas/O=Infineon/OU=DSS/CN=www.infineon.com
---
```

Figure 31 S_Client transaction

On the TShark window we will see that 13 packages were captured. Stop the TShark task as we will work with the captured file

```
pi@raspberrypi-os:~ $ tshark -s0 -w tls.pcap -i 4
Capturing on 'Loopback'
13
```

Figure 32 TShark captured packets

On the **tls_log** directory we will work with the captured file to get the information we want. For this we will use TShark as well.

Code Listing 31 TShark terminal window

```
001        tshark -r tls.pcap -V -x -o "ssl.debug_file:ssldebug.log" -o
   "ssl.desegment_ssl_records: TRUE" -o
   "ssl.desegment_ssl_application_data: TRUE" -o
   "ssl.keys_list:127.0.0.1,8444,http,server.pem" >> TLS_HandShake.log
```

What the above command did is that it read the captured file with TLS handshake process and saved it to TLS_HandShake.log. The reason we do this is that the file format used by TShark is not readable by normal means.

```
pi@raspberrypi-os:~/tpm_hardened_tls/tls_log $ tshark -r tls.pcap -V -x -o "ssl.debug_file:ssldebug.log" -o "ssl.desegment_ssl_records: TRUE" -o "ssl.
desegment_ssl_application_data: TRUE" -o "ssl.keys_list:127.0.0.1,8444,http,server.pem" >> TLS_HandShake.log
```

Figure 33 Reading tls.pcap file using TShark

Now that we have the transaction between the S_Server and S_Client in a workable form we will filter the "**Secure Socket Layer**" interactions which are the ones of interest to us.

Code Listing 32 TShark terminal window

```
001        grep -A70 "Secure Sockets Layer" TLS_HandShake.log
```

These are the different steps that happened during the S_Server and S_Client TLS handshake process.

```
pi@raspberrypi-os:~/tpm_hardened_tls/tls_log $ grep -A70 "Secure Sockets Layer" TLS_HandShake.log
Secure Sockets Layer
    TLSv1 Record Layer: Handshake Protocol: Client Hello
        Content Type: Handshake (22)
        Version: TLS 1.0 (0x0301)
        Length: 171
        Handshake Protocol: Client Hello
            Handshake Type: Client Hello (1)
            Length: 167
            Version: TLS 1.2 (0x0303)
            Random: fbe25a2b781d5993b5edfe79968a01f91f93093e74383b1c...
                GMT Unix Time: Oct 24, 1967 11:55:39.000000000 PDT
                Random Bytes: 781d5993b5edfe79968a01f91f93093e74383b1ca883bdf9...
            Session ID Length: 0
            Cipher Suites Length: 56
            Cipher Suites (28 suites)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
                Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
                Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
                Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
                Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
                Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
                Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
                Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
                Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
                Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
                Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
                Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
                Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
                Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
                Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
                Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
            Compression Methods Length: 1
            Compression Methods (1 method)
                Compression Method: null (0)
            Extensions Length: 70
            Extension: ec_point_formats (len=4)
                Type: ec_point_formats (11)
                Length: 4
                EC point formats Length: 3
                Elliptic curves point formats (3)
                    EC point format: uncompressed (0)
                    EC point format: ansiX962_compressed_prime (1)
                    EC point format: ansiX962_compressed_char2 (2)
            Extension: supported_groups (len=10)
                Type: supported_groups (10)
                Length: 10
                Supported Groups List Length: 8
                Supported Groups (4 groups)
                    Supported Group: x25519 (0x001d)
                    Supported Group: secp256r1 (0x0017)
                    Supported Group: secp521r1 (0x0019)
                    Supported Group: secp384r1 (0x0018)
            Extension: SessionTicket TLS (len=0)
                Type: SessionTicket TLS (35)
                Length: 0
                Data (0 bytes)
            Extension: encrypt_then_mac (len=0)
                Type: encrypt_then_mac (22)
                Length: 0
```

Figure 34 TLS Client Hello

```
Secure Sockets Layer
    TLSv1.2 Record Layer: Handshake Protocol: Server Hello
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 65
        Handshake Protocol: Server Hello
            Handshake Type: Server Hello (2)
            Length: 61
            Version: TLS 1.2 (0x0303)
            Random: f8ed6856138131c3a65aedb96810759531ff217fac1781f5...
                GMT Unix Time: Mar 29, 1966 07:10:14.000000000 PST
                Random Bytes: 138131c3a65aedb96810759531ff217fac1781f5829e96bc...
            Session ID Length: 0
            Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
            Compression Method: null (0)
            Extensions Length: 21
            Extension: renegotiation_info (len=1)
                Type: renegotiation_info (65281)
                Length: 1
                Renegotiation Info extension
                    Renegotiation info extension length: 0
            Extension: ec_point_formats (len=4)
                Type: ec_point_formats (11)
                Length: 4
                EC point formats Length: 3
                Elliptic curves point formats (3)
                    EC point format: uncompressed (0)
                    EC point format: ansiX962_compressed_prime (1)
                    EC point format: ansiX962_compressed_char2 (2)
            Extension: SessionTicket TLS (len=0)
                Type: SessionTicket TLS (35)
                Length: 0
                Data (0 bytes)
            Extension: extended_master_secret (len=0)
                Type: extended_master_secret (23)
                Length: 0
    TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 876
        Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 872
            Certificates Length: 869
            Certificates (869 bytes)
                Certificate Length: 866
                Certificate: 3082035e30820246020900aecba021a19d435a300d06092a... (id-at-commonName=www.infineon.com,id-at-organizationalUnitName=DSS,i
d-at-organizationName=Infineon,id-at-localityName=Milpitas,id-at-stateOrProvinceName=California,id-at-cou
                    signedCertificate
                        serialNumber: 12595336849177527130
                        signature (sha256WithRSAEncryption)
                            Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
                        issuer: rdnSequence (0)
                            rdnSequence: 6 items (id-at-commonName=www.infineon.com,id-at-organizationalUnitName=DSS,id-at-organizationName=Infineon,i
d-at-localityName=Milpitas,id-at-stateOrProvinceName=California,id-at-countryName=US)
                                RDNSequence item: 1 item (id-at-countryName=US)
                                    RelativeDistinguishedName item (id-at-countryName=US)
                                        Id: 2.5.4.6 (id-at-countryName)
                                        CountryName: US
                                RDNSequence item: 1 item (id-at-stateOrProvinceName=California)
                                    RelativeDistinguishedName item (id-at-stateOrProvinceName=California)
                                        Id: 2.5.4.8 (id-at-stateOrProvinceName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: California
                                RDNSequence item: 1 item (id-at-localityName=Milpitas)
                                    RelativeDistinguishedName item (id-at-localityName=Milpitas)
                                        Id: 2.5.4.7 (id-at-localityName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: Milpitas
                                RDNSequence item: 1 item (id-at-organizationName=Infineon)
                                    RelativeDistinguishedName item (id-at-organizationName=Infineon)
                                        Id: 2.5.4.10 (id-at-organizationName)
                                        DirectoryString: uTF8String (4)
```

Figure 35 TLS Server Hello

```
Secure Sockets Layer
    TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 876
        Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 872
            Certificates Length: 869
            Certificates (869 bytes)
                Certificate Length: 866
                Certificate: 3082035e30820246020900aecba021a19d4359300d06092a... (id-at-commonName=www.infineon.com,id-at-organizationalUnitName=DSS,i
d-at-organizationName=Infineon,id-at-localityName=Milpitas,id-at-stateOrProvinceName=California,id-at-cou
                    signedCertificate
                        serialNumber: 12595336849177527129
                        signature (sha256WithRSAEncryption)
                            Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
                        issuer: rdnSequence (0)
                            rdnSequence: 6 items (id-at-commonName=www.infineon.com,id-at-organizationalUnitName=DSS,id-at-organizationName=Infineon,i
d-at-localityName=Milpitas,id-at-stateOrProvinceName=California,id-at-countryName=US)
                                RDNSequence item: 1 item (id-at-countryName=US)
                                    RelativeDistinguishedName item (id-at-countryName=US)
                                        Id: 2.5.4.6 (id-at-countryName)
                                        CountryName: US
                                RDNSequence item: 1 item (id-at-stateOrProvinceName=California)
                                    RelativeDistinguishedName item (id-at-stateOrProvinceName=California)
                                        Id: 2.5.4.8 (id-at-stateOrProvinceName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: California
                                RDNSequence item: 1 item (id-at-localityName=Milpitas)
                                    RelativeDistinguishedName item (id-at-localityName=Milpitas)
                                        Id: 2.5.4.7 (id-at-localityName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: Milpitas
                                RDNSequence item: 1 item (id-at-organizationName=Infineon)
                                    RelativeDistinguishedName item (id-at-organizationName=Infineon)
                                        Id: 2.5.4.10 (id-at-organizationName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: Infineon
                                RDNSequence item: 1 item (id-at-organizationalUnitName=DSS)
                                    RelativeDistinguishedName item (id-at-organizationalUnitName=DSS)
                                        Id: 2.5.4.11 (id-at-organizationalUnitName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: DSS
                                RDNSequence item: 1 item (id-at-commonName=www.infineon.com)
                                    RelativeDistinguishedName item (id-at-commonName=www.infineon.com)
                                        Id: 2.5.4.3 (id-at-commonName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: www.infineon.com
                        validity
                            notBefore: utcTime (0)
                                utcTime: 19-08-16 06:00:29 (UTC)
                            notAfter: utcTime (0)
                                utcTime: 20-08-15 06:00:29 (UTC)
                        subject: rdnSequence (0)
                            rdnSequence: 6 items (id-at-commonName=www.infineon.com,id-at-organizationalUnitName=DSS,id-at-organizationName=Infineon,i
d-at-localityName=Milpitas,id-at-stateOrProvinceName=California,id-at-countryName=US)
                                RDNSequence item: 1 item (id-at-countryName=US)
                                    RelativeDistinguishedName item (id-at-countryName=US)
                                        Id: 2.5.4.6 (id-at-countryName)
                                        CountryName: US
                                RDNSequence item: 1 item (id-at-stateOrProvinceName=California)
                                    RelativeDistinguishedName item (id-at-stateOrProvinceName=California)
                                        Id: 2.5.4.8 (id-at-stateOrProvinceName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: California
                                RDNSequence item: 1 item (id-at-localityName=Milpitas)
                                    RelativeDistinguishedName item (id-at-localityName=Milpitas)
                                        Id: 2.5.4.7 (id-at-localityName)
                                        DirectoryString: uTF8String (4)
                                            uTF8String: Milpitas
                                RDNSequence item: 1 item (id-at-organizationName=Infineon)
                                    RelativeDistinguishedName item (id-at-organizationName=Infineon)
                                        Id: 2.5.4.10 (id-at-organizationName)
```

Figure 36 TLS Handshake Protocol Certificate

```
Secure Sockets Layer
    TLSv1.2 Record Layer: Handshake Protocol: New Session Ticket
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 1050
        Handshake Protocol: New Session Ticket
            Handshake Type: New Session Ticket (4)
            Length: 1046
            TLS Session Ticket
                Session Ticket Lifetime Hint: 7200 seconds (2 hours)
                Session Ticket Length: 1040
                Session Ticket: f9799e8d8b8c8f19d19afd2e70f880fc151d125c0cead678...
    TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
        Content Type: Change Cipher Spec (20)
        Version: TLS 1.2 (0x0303)
        Length: 1
        Change Cipher Spec Message
    TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 40
        Handshake Protocol: Encrypted Handshake Message

0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00   ..............E.
0010  04 86 e0 00 40 00 40 06 58 6f 7f 00 00 01 7f 00   ....@.@.Xo......
0020  00 01 20 fc db 0a 4f 85 61 3f 36 45 7f b9 80 18   .. ...O.a?6E....
0030  05 5d 02 7b 00 00 01 01 08 0a b4 6c c2 f6 b4 6c   .].{.......l...l
0040  c2 b5 16 03 03 04 1a 04 00 04 16 00 00 1c 20 04   .............. .
0050  10 f9 79 9e 8d 8b 8c 8f 19 d1 9a fd 2e 70 f8 80   ..y.........p..
0060  fc 15 1d 12 5c 0c ea d6 78 c4 7d 36 60 36 b8 51   ....\...x.}6`6.Q
0070  a1 98 7f 04 ca e7 82 59 cd 33 81 4c 8c f8 c9 4b   .......Y.3.L...K
0080  7d b3 fb 94 2f cb c7 23 ee 87 c3 cb ab f6 4a c9   }.../..#......J.
0090  c7 2c 06 23 6d 9a ed 83 fc f4 90 e4 ef ba 94 14   .,.#m...........
00a0  0b 32 7d 40 cc f9 4f 22 5b ce e5 64 25 66 35 be   .2}@..O"[..d%f5.
00b0  90 4e 78 4c b7 46 88 93 fb a9 98 36 3f bd 89 44   .NxL.F.....6?..D
00c0  90 d5 61 b3 f4 a8 b4 53 0a 8a 4e da b0 26 62 c3   ..a....S..N..&b.
00d0  a7 87 53 82 ec a4 68 e8 e6 0d ce 54 35 1f d7 3d   ..S...h....T5..=
00e0  fc 71 86 5f 7a cb 11 b8 db 4d 3e 20 a4 d3 07 b3   .q._z....M> ....
00f0  73 65 8e 41 3b 7c 2a c8 1c e8 bd c4 c7 3f dd ad   se.A;|*......?..
0100  e5 5e 2a 3b d8 f6 06 30 02 ca 2e 8e 5c 03 05 e4   .^*;...0....\...
0110  7f 13 63 6a 7d 39 88 4a f0 94 36 fc 0d 13 8e 2f   ..cj}9.J..6..../
0120  7d 5b 47 a9 1f 68 08 8d 07 97 bf 33 38 f4 83 d5   }[G..h.....38...
0130  c6 c5 46 fc 2f 5c 10 d7 bf 16 9c 56 e1 74 b3 27   ..F./\.....V.t.'
0140  2d d5 6d c4 9c 5c a0 c3 16 6c 1a fa 9b df eb 26   -.m..\...l.....&
0150  f2 ec 90 df 39 c2 55 88 94 90 3d 6a 77 b2 1f 16   ....9.U...=jw...
0160  25 9f c4 74 82 32 09 31 4d 53 b3 37 7c d4 29 7b   %..t.2.1MS.7|.){
0170  8e bd b6 98 8e 56 3f f4 06 7c fc e8 d5 6d 09 4f   .....V?..|...m.O
0180  d1 f8 3c 79 d7 4f 80 2a 3b 70 bd cc 32 ab de 6f   ..<y.O.*;p..2..o
0190  1b 01 36 e7 2b f5 74 b2 69 26 f9 76 a6 df 26 9d   ..6.+.t.i&.v..&.
01a0  5d 1b 5c f1 67 6f f9 0b 8f 54 bd 90 a3 72 0d 04   ].\.go...T...r..
01b0  63 bb 88 5e 51 4b de 3c 13 25 90 8a 6d f4 8d c5   c..^QK.<.%..m...
01c0  12 0a ec 52 dd d9 37 29 42 6f c1 28 b1 52 20 2c   ...R..7)Bo.(.R ,
01d0  58 ab 0e 65 a2 98 f3 d6 75 4f e9 0f 8b d2 4e cf   X..e....uO....N.
01e0  22 f3 c4 2c d4 ac be ee bb 67 ca 22 95 f4 bc d3   "..,.....g."....
01f0  2f f8 3d f2 b4 ff 15 8f 1e c0 c7 20 f7 2a 7c 6b   /.=........ .*|k
0200  e7 99 bd 2f 00 fd 27 70 a9 d2 85 26 bb 2f 29 c4   .../..'p...&./).
0210  38 80 3c 95 f7 68 e1 19 8d ff 6f b6 01 18 ac 9a   8.<..h....o.....
0220  44 3a 73 4d 0a f1 a4 37 56 5c a9 54 b4 12 cd a8   D:sM...7V\.T....
0230  17 bb 84 61 02 94 a5 5b dc f3 24 79 55 5c b6 1f   ...a...[..$yU\..
0240  61 48 28 4c 00 ec 6e 95 7c 89 d4 80 37 a1 d0 ad   aH(L..n.|...7...
0250  7f f6 94 f0 e4 f9 8c d1 20 20 48 49 46 7c c6 5d   ........  HIF|.]
0260  41 b2 91 7b fe 70 8c c6 dd df 71 4d 52 2a c3 b9   A..{.p....qMR*..
0270  fd c0 97 36 f1 58 9a fa 7d e2 18 74 c7 77 af 55   ...6.X..}..t.w.U
0280  dc 50 98 ac 56 ba 22 b6 f8 9f a5 ee dd d3 5e a2   .P..V."......^.
0290  46 5d fd 5a 2f 4d 00 c8 e5 e9 01 02 ef 87 4e 07   F].Z/M........N.
02a0  6b 86 56 b1 5d cf 16 ed eb 3f 14 fc b0 8d f1 ca   k.V.]....?......
02b0  05 de ba 2d bc a0 09 6f 57 3f e5 06 8f e9 af 2a   ...-...oW?.....*
02c0  13 b2 3a f9 02 6f 53 c8 f1 9b 55 72 61 97 9c 49   ..:..oS...Ura..I
02d0  06 86 c6 39 42 dd 1f 8d 5a 75 a6 3d 90 6f 49 77   ...9B...Zu.=.oIw
02e0  1e 55 06 56 56 22 25 c9 0c e6 7f 5e 77 3e 66 67   .U.VV"%....^w>fg
02f0  16 e2 7d ea 9d 2c dd aa e8 b4 44 9e 47 b4 e4 b7   ..}..,....D.G...
```

Figure 37 TLS Creation of Session Ticket : All communication are encrypted at this point

TShark is a useful tool that enables engineers and hackers to monitor transactions in a network. The important fact of using OPTIGA™ SLx 9670 TPM 2.0 as hardening element of a TLS session is that the private key used to enable the session is guarded by OPTIGA™ SLx 9670 TPM 2.0. The use of the Private Key is also protected by a TPM 2.0 password policy. More complex policies can be created around TPM 2.0 objects. These are out of the scope of this application note.

As we have mentioned the TLS layer sits on top of the Transport Layer and provides the means of encrypting the communication channel between two entities. Additionally, TLS is a point to point transaction and not end to end. In other words, the TLS channel is established between two parties.

A higher level of security can be achieved by implementing Token Binding which is an extension to TLS.Token Binding Protocol is established by the user-agent generating a Private Key-Public Key pair (managed by TPM) per target server, and proving possession of the private key on every TLS connection to the target server.

4 References

[1] http://www.infineon.com/tpm

[2] https://github.com/tpm2-software

[3] https://www.openssl.org/

[4] https://trustedcomputinggroup.org/work-groups/trusted-platform-module/

[5] https://trustedcomputinggroup.org/work-groups/software-stack/

[6] https://trustedcomputinggroup.org/wp-content/uploads/TSS_ESAPI_Version-0.9_Revision-04_reviewEND030918.pdf

[7] https://www.raspberrypi.org/downloads/raspbian/

[8] https://www.wireshark.org/

[9] https://www.rfc-editor.org/rfc/rfc8472.txt

[10] https://www.globalsign.com/en/

Revision history

Document version	Date of release	Description of changes
1.0	27.04.2020	Initial version

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2020-04-27

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:
dsscustomerservice@infineon.com