Intel® Trusted Edge Platform (TEP)

User Guide for TEP Container Profile

August 2022

Intel Confidential

Intel Confidential

NDA Required

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Copies of documents, which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or visiting www.intel.com/design/literature.htm.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Contents

Reference Documents

§

Reference	Modules/Owner	Description
1	TEP Container	https://projectTep Container.org/
2	Yocto	https://www.yoctoproject.org/
3	TPM2_PKCS11 Stack	https://github.com/tpm2-software/tpm2-pkcs11
4	TPM2 TSS Stack	https://tpm2-software.github.io/
5	PKCS#11 Spec	http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/csprd02/pkcs11-base-v2.40-csprd02.html
6	P11 kit	https://p11-glue.github.io/p11-glue/p11-kit/manual/
7	Intel-iSecl	https://github.com/intel-secl/intel-secl

Terminology

Term	Description
TPM	Trusted Platform Module
BSP	Boards Support Package
HAL	Hardware Abstraction Layer
RPC	Remote Procedure Call
PKCS11	Public-Key Cryptography Standards
AES	Advance Encryption standard
RSA	Rivest Shamir Adelman
OS	Operating System
VM	Virtual machine
eRPC	Embedded RPC
ECC	Elliptical Curve Cryptography
SFTP	Secure File Transfer Protocol
LUKS	Linux Unified Key Setup
PCR	Platform Configuration Register
TEP	Trusted Edge Platform
TA	Trust Agent

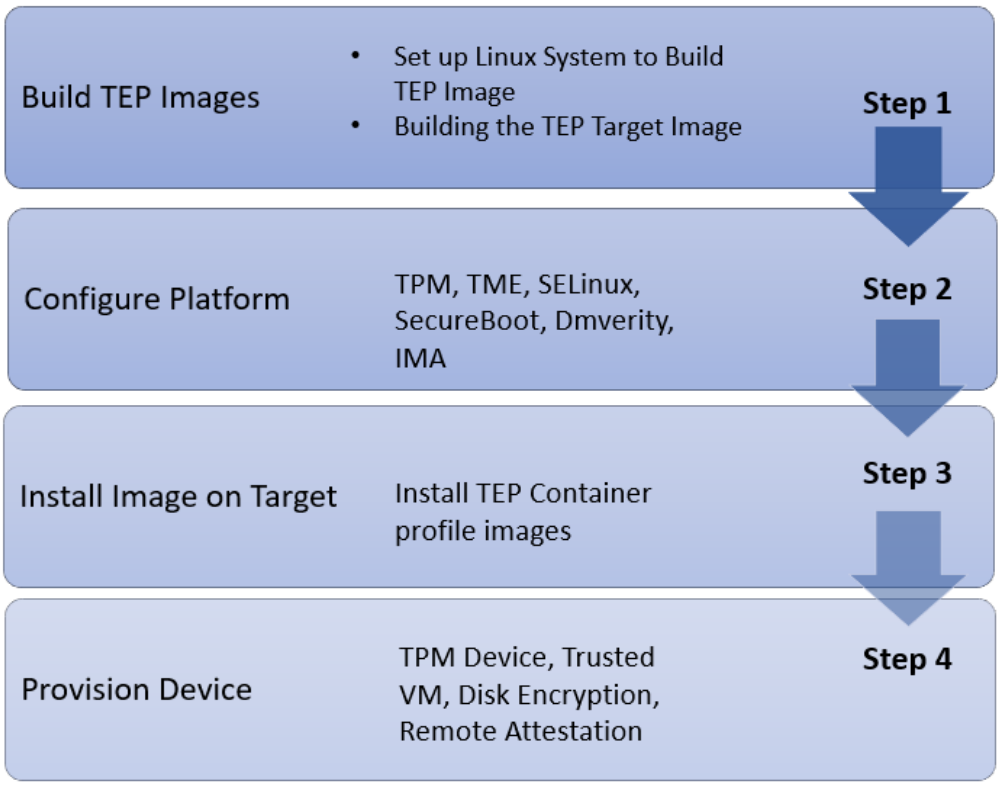
1.0 Introduction

1.1 Purpose

The purpose of this document is to serve as user guide for the Intel® Trusted Edge Platform (TEP) running on Container. For an overview of the Trusted VM, refer to the *Intel TEP Design Guide*.

This version of Trusted Container uses an Intel® Platform Trust Technology (Intel® PTT) as physical hardware TPM. The intent is to provide a TPM based release to customers for their development purpose.

The following flow chart shows the high-level steps for setting up Trusted Edge Platform for the Container profile:



- Build TEP Images (Step 1):
  - Contain steps to set up Linux build system, build TEP image, and TEP target image. There are also steps for disabling microservices and cleaning the target build.
- Configure Platform (Step 2):

  - Contain steps to configure platform for TPM, TME, Secureboot, IMA, Dmverity.
  - Features like Secureboot, IMA, Dmverity can be configured in Yocto and Ubuntu built system.

- Install Image on Target (Step 3):
  - This section gives a brief about how to install TEP Container profile images.
- Provision Device (Step 4):
  - This section contains the steps to provision devices to TEP for Container profile.

1.2 Customer Support

Contact your Intel representative for support or submit an issue to Intel® Premier Support:
https://premiersupport.intel.com

2.0 Building TEP Images

This chapter describes the steps to build TEP images in the following sections:

- Configuring Git
- Setting up Firewall/Proxy
- Adding ssh Public Key to GitHub Account
- Setting up a Linux System to Build TEP Image
- Building the TEP Target Image
- Disabling Microservices
- Cleaning a Build Target

Note: These steps need to be performed Build Machine. See Build System section in the TEP Release Notes for the recommended specs.

2.1 Configuring Git

Ensure that Git is available in your system. To build TEP, configure Git using the following commands:

```
git config --global user.name "username"
git config --global user.email "email"
```

2.2 Setting up Firewall/Proxy

Some environments require network proxies for docker operations (for example, docker pull, docker push, docker run, and so on). Therefore, it is important to set the proxies correctly.

For http_proxy/HTTP_PROXY and https_proxy/HTTPS_PROXY, test whether proxies are working. Also, note that the no_proxy/NO_PROXY list MUST NOT contain spaces between the addresses.

```
Acquire::http::proxy "http://ProxyHOST:ProxyPORT/";
Acquire::https::proxy "https://ProxyHOST:ProxyPORT/";
```

2.3 Adding ssh Public Key to GitHub Account

Set up network proxy in /etc/environment location if you are installing behind the firewall. For secure config update, you must have authorized host public key added into build.

You can generate RSA/ECC keys and replace/add public key at following file before triggering the build:

```
<target/resources>/meta-tep-trusted-os/recipes-core/tep-disk-encryption/files/update_keys/update_user_key.pub.
```

2.4 Setting up a Linux System to Build TEP Image

To set up a Linux build system to build a TEP image, do the following steps:

1. Get the TEP package from Intel representative.
2. Copy the `tep-container-release.tar.gz.bz2` file to installation setup, and extract the package:

```
bzip2 -d tep-container-release.tar.gz.bz2

tar -xvf tep-container-release.tar.gz
```

3. To install the package run with following options:

```
./setup.sh help
```

To get the usage of the package, build a host image with the Trusted Container feature option enabled.

Creating a TEP image that contains the Trusted Container for feature can be accomplished by selecting the 'tep-container' feature option during image setup.

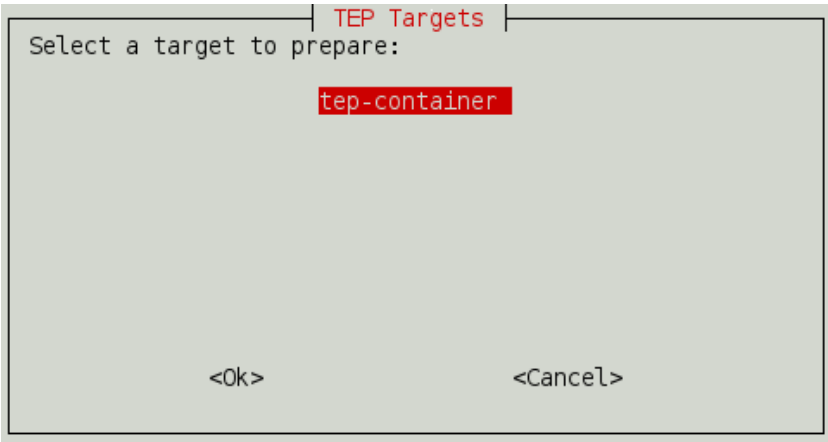
```
NAME
    setup.sh - Setup Yocto environment and start compilation.

SYNOPSIS
    setup.sh [TARGET] [--options OPTION ...]

DESCRIPTION
    A script to setup a Yocto environment on a build system
    and fetch all software layers necessary for a specific
    target as defined in the list below.

TARGETS
    Note that * denotes an experimental beta option.

OPTION HANDLING
    If present, build option with acronym OPTION will be enabled.
    If OPTION is prefixed with -, then it is disabled if it's on
    by default.

    tep-container
   ------------------------------------------------------------------------
    Kernel version: 5.4.61
    Tep trusted OS build build with Host OS with KVM enabled

OPTIONS TO DISABLE MICROSERVIE
==================================
    Example -
    GUI : ./setup.sh [TARGET] [--options OPTION ...] micro-service-gui
    CLI : ./setup.sh [TARGET] [--options OPTION ...] micro-service-cli
```

To capture the logs of the package along with enabling debug logs for installation, use the following command:

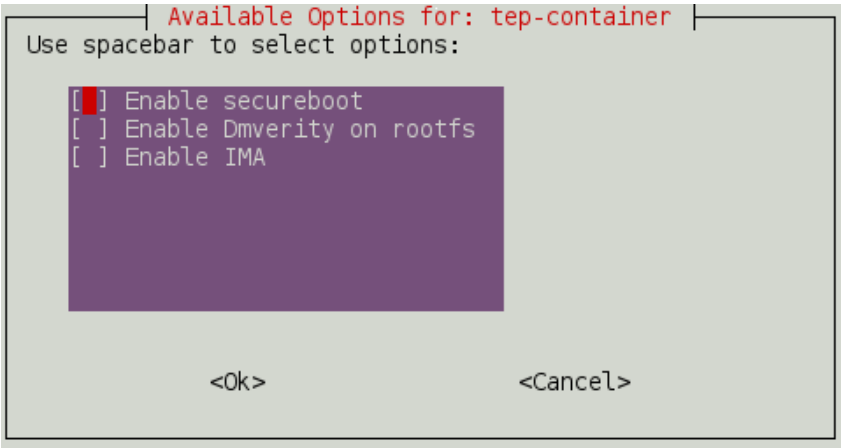
```
bash -x ./setup.sh tep-container 2>&1 | tee debug_logs
```

2.5 Building the TEP Target Image

To install the package, a UI option will pop to build the respective target. Following is an example of 'tep-container' target.



A UI option appears to select the respective configuration. Following is an example of tep-container target.



The setup script will begin configuring the assets needed to build the target image. Depending on the feature options selected and state of the build environment, a few notifications may occur. Some of these notifications are described below.

- Build setup OS version preference

The setup script expects. The Linux OS of choice for Yocto Project build is Ubuntu 18.04 LTS, and above. It will prompt for user confirmation as follows:

```
Do you want to continue the setup? y/[n]
```

- Build directory already exists

If the setup script is not building the target for the first time, the script displays the following message:

```
Build directory tep-container already exists. Do you want to clean the cached build? y/[n]
```

```
--2022-02-24 11:50:50--  https://git.yoctoproject.org/meta-intel/tree/MAINTAINERS
Reusing existing connection to git.yoctoproject.org:443.
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: '/dev/null'

/dev/null                                                    [ <=>

2022-02-24 11:50:51 (446 KB/s) - '/dev/null' saved [11562]

TARGET_FEATURE_SELECTION -->

Build directory tep-container already exists
Do you want to clean the cached build? y/[n]
```

Press 'n' to retain the target build cache, or press 'y' to delete target build cache.

Note: It is advised to delete the target build cache if the target source was modified, or the previous build was incomplete. Retaining the target build cache reduces the build time but may promote build errors if the target source was modified or the previous build was incomplete. If build errors occur, rerun the setup script, and choose 'y' at this prompt.

After setting up the build target, the script shows the following message:
prompt: Do you want to run an automated build? y/[n]

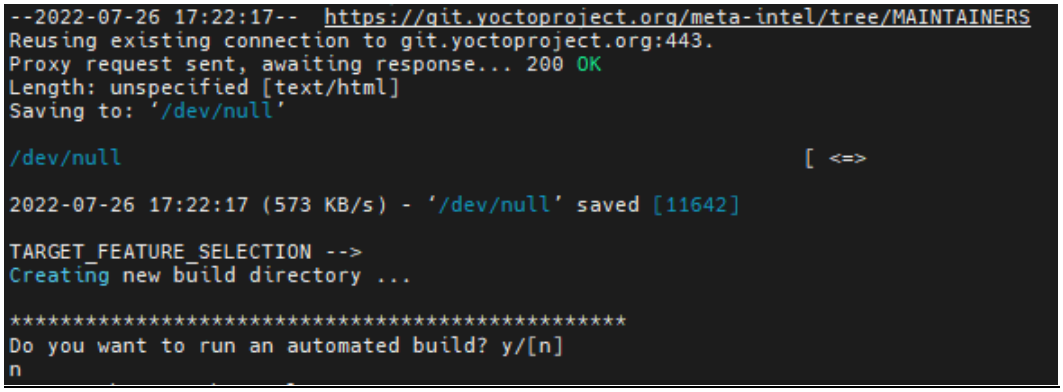
An example is as follows:

```
--2022-07-26 17:52:23--  https://git.yoctoproject.org/meta-intel/tree/MAINTAINERS
Reusing existing connection to git.yoctoproject.org:443.
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: '/dev/null'

/dev/null                                                [  <=>

2022-07-26 17:52:24 (53.1 KB/s) - '/dev/null' saved [11642]

TARGET_FEATURE_SELECTION -->
Creating new build directory ...

**********************************************
Do you want to run an automated build? y/[n]
y
```

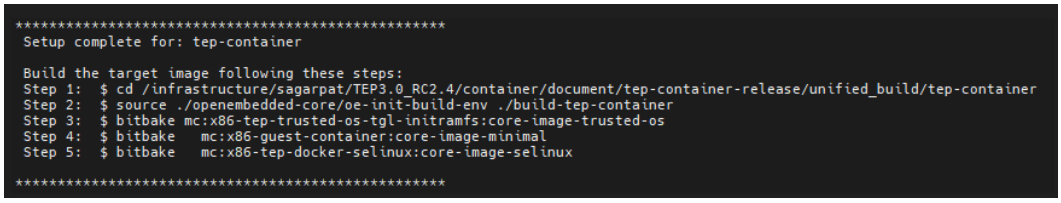
To perform an automated build, press 'y' at the prompt.

The build typically takes a very long time. A Linux build system with the recommended specs may take about 3-4 hours to complete. A Linux build system with the minimum specs may take 6+ hours to complete. See *Build System* section for the recommended specs.

To perform a manual build, press 'n' at the prompt. Example as follows:



- The setup script displays a completion message with the steps necessary to perform a build manually. Example as follows:



With these manual steps, you can perform customized build. TEP provides a select number of precomposed definitions for target platforms to use with platforms. If customization or modifications for build targets are desired, follow the steps to compile and generate the images.

1. After changes the recipes or configurations, select the target, by running the `./setup.sh` script, select any target platform when prompted. **Do not perform an automated build.**
2. Perform the bitbake build process to build the build target image
   - It is recommended to clean the `<build-target>` bitbake recipe if an existing build has already completed. To clean the <build-target> bitbake recipe, use the following commands:

```
cd <your_path>/unified_build/<build-target>/<build-dir>
source oe-init-build-env ../<build-dir>
bitbake -c cleanall <target>
```

The build images are available at `<your path>/<target>-release/unified_build/Output`.

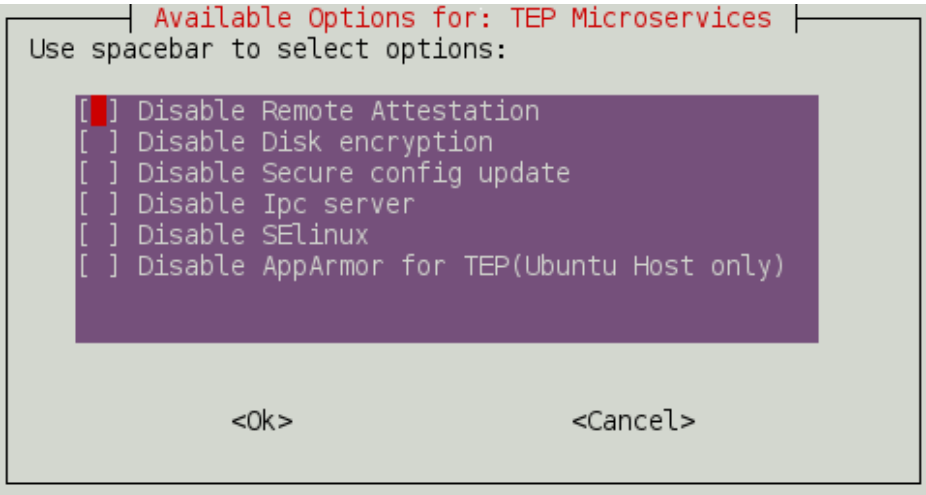
After building a TEP image, see Installing Image section for information on installing the TEP images for respective targets.

2.6 Disabling Microservices

The Microservices are available by default. However, you can choose to disable the microservices. To build a tep-container image without microservices, use the following command:

```
./setup.sh micro-service-gui
```

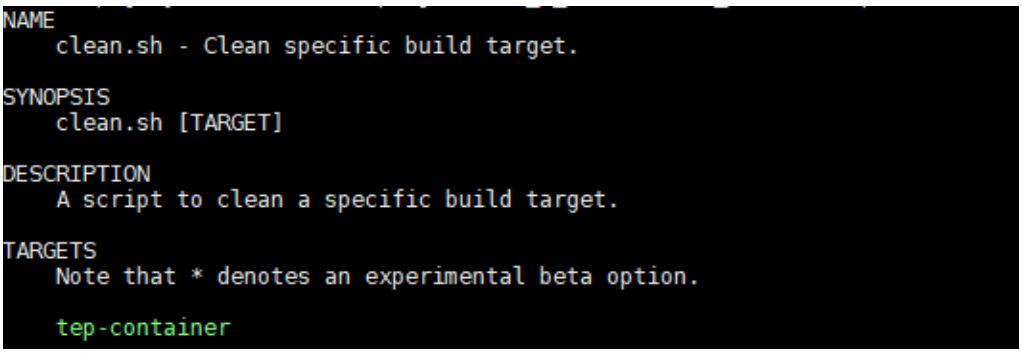
The following UI option appears where you can select the microservices that you want to disable.



2.7 Cleaning a Build Target

You can clean the `tep-container target` by using the `clean.sh` script. Following is the command:

```
./clean.sh help
```



Note: This script removes the complete build directory. If you want to clean Yocto recipes, follow the Yocto clean up method.

3.0 Configuring Platform

Platform with trusted container recommends some configuration to achieve desired security goals. Some of these configurations are described in this section. Ensure that these changes are done at platform level to get system configurations right. TEP supports the following configurations:

- TPM Configuration
- TME Configuration
- Secure Boot Configuration
- TEP VM Disk Partitions
- SELinux Configuration
- Sharing TPM Devices to Guest Containers
- DM-Verity Configuration
- Container TEP Trusted OS Measured Boot Configuration

3.1 Trusted Platform Module (TPM) Configuration

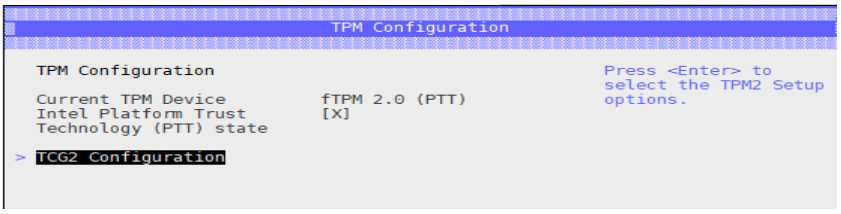
To use TPM as trusted execution environment in the platform, which acts as hardware root of trust for Trusted OS user configurations, disk encryption, and measured boot, platform shall have TPM enabled. The following instructions are specific to Intel PTT enabled platform. These steps help to check if TPM is enabled in 11th Gen Intel® Core™ platform.

Note:

Ensure that we have PTT enabled BIOS/fw image. You can verify it using following steps on 11th Gen Intel® Core™ BIOS.

1. Go to **BIOS menu**->**Intel Advanced Menu**.

The **TPM Configuration** screen is as follows:



Note: Ensure that Intel PTT is enabled.

3.2 Total Memory Encryption (TME) Configuration

Total Memory Encryption (TME) is used to protect DRAM data from physical attacks. Such attacks include, moving DRAM module to another system, probing the DDR to read the cycles, and so on. System memory is encrypted by the TME block attached to the memory controller. All cycles through TME block are encrypted except for the specific exclusion ranges as programed by BIOS.

This capability is typically enabled in the very early stages of the bios boot. Once configured and locked, will encrypt all the data on external memory buses of a SoC using the NIST standard AES-XTS algorithm with 128-bit keys or 256-bit keys depending on the algorithm availability and selection. The encryption key used for TME uses a hardware random number generator implemented in the Intel SoC, and the keys are not accessible by software or using external interfaces to the Intel SoC.

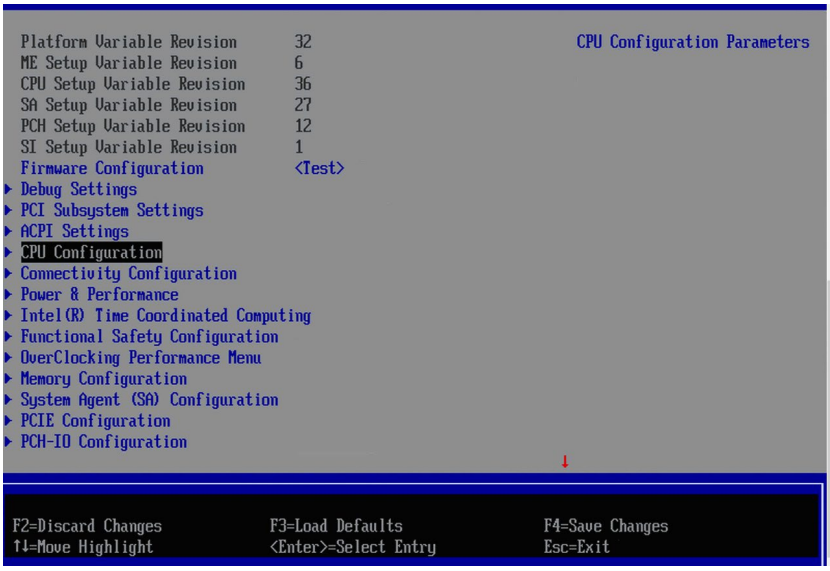
Note:

- **Total Memory Encryption** is not visible in menu options when the processor does not support this feature. This feature is supported for VPro platforms only. Also, this feature is not enabled in “FUSA” enabled SKU’s.
- **Total Memory Encryption** option is not enabled when **In Band ECC** is enabled.

For details refer to TME spec at,
https://software.intel.com/content/dam/develop/external/us/en/documents-tps/multi-key-total-memory-encryption-spec.pdf

To enable the TME capabilities in system, do the following steps:

1. Go to **BIOS** menu->**Intel Advanced Menu**.
2. Select **CPU Configuration**.



3. Select **Enabled** for Total Memory Encryption.

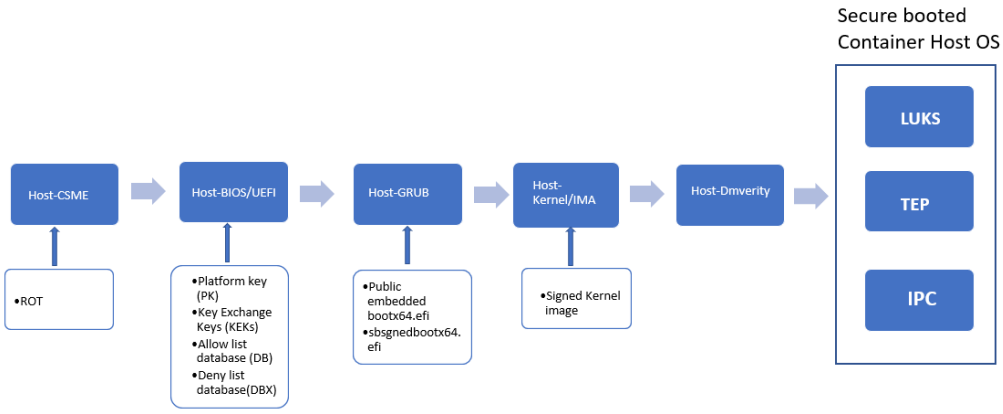
Note:

- **Total Memory Encryption** is unavailable in menu options when the processor does not support this feature. This feature is supported for VPro platforms only. Also, this feature is not enabled in “FUSA” enabled SKU’s.
- **Total Memory Encryption** option is grayed out when the In Band ECC is enabled.

3.3 Secure Boot Configuration

This section describes the secure booting of Container based Trusted Edge Platform (TEP) solution with UEFI FW. The method uses GRUB to securely boot the Container Host OS. The flow diagram for secure boot is as follows.

Figure 1 Secure Boot Flow Diagram



On booting the platform, UEFI verifies the GRUB. GRUB verifies and launches the Container Host.

3.3.1 Enabling Secure Boot for Yocto

This section contains the steps for configuring Secure boot using Yocto. The high-level steps are as follows:

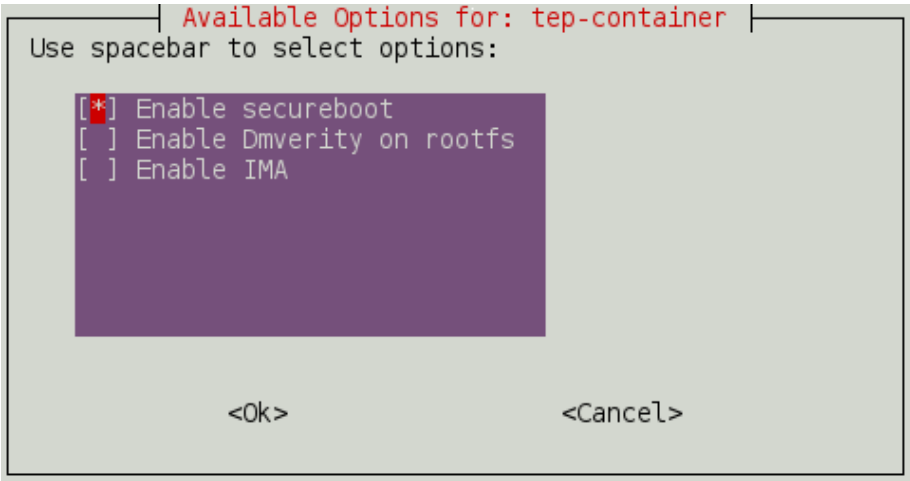
- Configuring Keys for Secure Boot
- Enabling Secure Boot in UEFI Bios

3.3.1.1 Configuring Keys for Secure Boot

Secure boot code is part of a Yocto build system. To configure the secure keys, do the following additional steps:

1. Generate Sample Keys and Certificates. You can also refer to the Sample Keys Generation section to generate key.
2. After GPG keys are generated copy <gpg-keys> folder to meta-tep-container/files/gpg_keys as a /keys.
3. After GPG public key “boot.key“ is generated copy to meta-tep-container/files/boot_keys/ folder.

4. After Certificates are generated, copy “db.der KEK.der PK.der” der formatted certificates to meta-tep-container/files/uefi_keys/ folder.
5. Once keys and certificates are copied, build image by selecting "Enable secure boot" from menu-entry, refer Setting up a Linux Build System to Build TEP Image. Image with TEP Secure Boot feature enabled will be built.



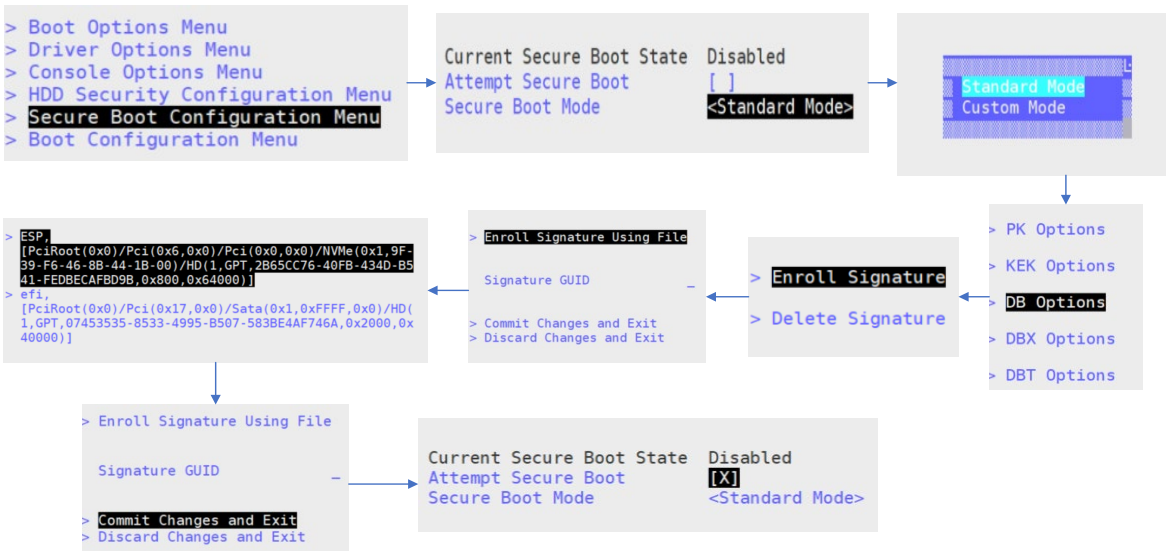
3.3.1.2 Enabling Secure Boot in UEFI BIOS

To enabling Secure boot in UEFI BIOS. Ensure that you have bootguard enabled BIOS to get HW root of trust and chain of trust extended from FW/HW to OS.

1. Copy the DB.der, KEK.der, and PK.der in a USB flash drive and connect it to the target board.
2. Restart the target board and enter the UEFI FW.
3. Go to secure boot settings.
4. Select **Secure Boot Mode** and then select the **Custom Mode** option.
5. Select Custom Secure Boot Options to enroll the certificates in following order - **DB.der → KEK.der → PK.der.**
6. Select **DB Options → KEK Options → PK Options** and select **Enroll Signature**.
7. Select **Enroll Signature Using file**. It lists the partitions.
8. Select the partition, which contains signature files and select the respective signature files.
9. After selecting the respective files, select the setting **Commit Changes and Exit**.
10. Exit after saving and you will see "Enable the secure boot setting (Attempt Secure Boot [X])" and restart/reset the system. Select the drive on which KVM Host OS image is flashed to securely boot.

The following figure shows the flow of deploying the signature files in BIOS.

Figure 2 Deploy signature in BIOS



3.3.1.3 Sample Keys Generation

Generate required GPG keys as follows:

```
mkdir --mode 0700 keys

gpg --homedir keys --gen-key

gpg --homedir keys --export > boot.key
```

Generate required OpenSSL keys as follows:

```
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=db/"
-keyout db.key  -out db.crt  -days 7300 -nodes -sha256

openssl req -new -x509 -newkey rsa:2048 -subj "/CN=KEK/"
-keyout KEK.key -out KEK.crt -days 7300 -nodes -sha256

openssl req -new -x509 -newkey rsa:2048 -subj "/CN=PK/" -
keyout PK.key -out PK.crt -days 7300 -nodes -sha256

openssl x509 -outform DER -in PK.crt -out PK.der

openssl x509 -outform DER -in KEK.crt -out KEK.der

openssl x509 -outform DER -in db.crt -out db.der

sign-efi-sig-list -k KEK.key -c KEK.crt db db.esl db.auth
```

3.3.2 Enabling the Secure Boot for Ubuntu Host OS

To enable secure boot on Ubuntu system, follow the steps mentioned in the following link: https://wiki.debian.org/SecureBoot.

3.4 TEP VM Disk Partitions

3.4.1 Enabling TEP VM Disk Partitions for Yocto

Following are details on partitions used by TEP Container system.

1. Container host wic image with dm-verity disabled:

Partition number	Mountpoint	Details
P1	/boot	Host os Boot partition contains grub and kernel images.
P2	/	Host os root filesystem.
P3		Host os swap Area.
P4		LUKS partition meant to be used by TEP Secure container.
P5	/tep	Contains binaries required for launching SecureVM like docker and secure docker image.

2. Container host wic image with Dmverity enabled:

With dm-verity enable host rootfs is mounted as read-only. Therefore, the /var and /etc folder are moved to separate partitions mounted as 'rw'.

Partition number	Mountpoint	Details
P1	/boot	Host os Boot partition contains grub and kernel images.
P2	/	Host os root filesystem.
P3		Host os swap Area.
P4		L partition meant to be used by TEP Secure Container.
P5	/tep	Contains binaries required for launching secure container like docker and secure docker image.
P6	/var	Host os /var folder as separate partition with read/write access. Default size reserved for this partition is 1 GB. Use can configure as per the requirement.
P7	/etc	Host os /etc folder as separate partition with read/write access.

Additional partitions user needs to setup required by TEP Secure Container:

- /dev/sda1 (Size: 10 MB, ext4) will be an unencrypted partition (mounted to /home/root/tmp) for TPM2 PKCS11 store.
- /dev/sda2 (Size: 100 MB, ext4)- will be an unencrypted partition (mounted to /home/update/upload/mnt/) for tep_user_config data, Public Keys (SSH trust list).

3. For Measured boot feature/tep partition (P5 in above tables) needs to have fixed universally unique identifier (uuid), which will be used in IMA policy.

3.4.2 Enabling TEP VM Disk Partitions for Ubuntu

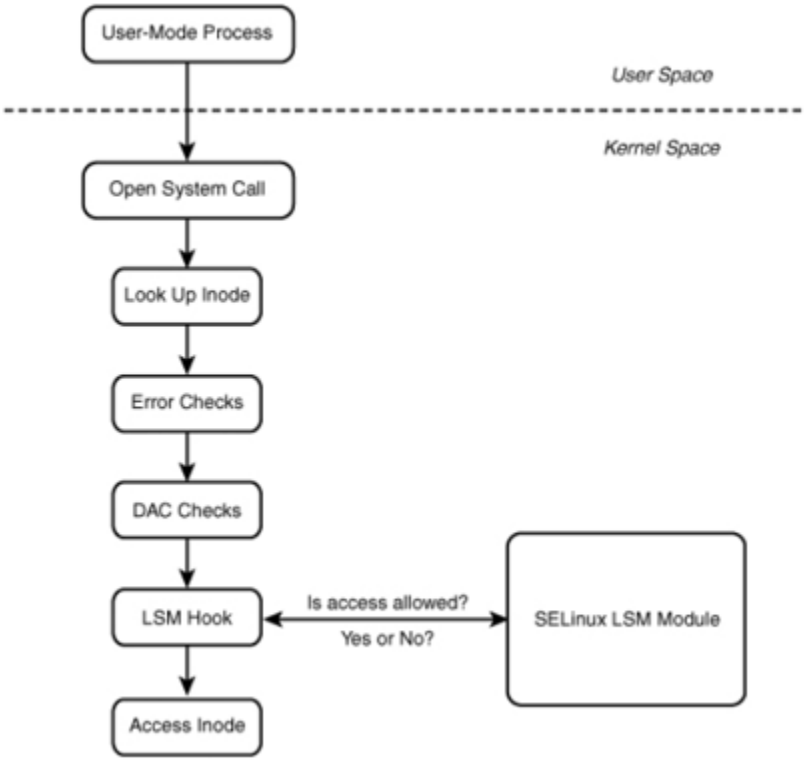
3.5 SELinux Configuration

Security-Enhanced Linux (SELinux) is a security architecture for Linux® systems that allows administrators to have more control over who can access the system.

SELinux defines access controls for the applications, processes, and files on a system. It uses security policies, which are a set of rules that tell SELinux what can or can't be accessed, to enforce the access allowed by a policy.

TEP defines several SELinux policies and rule sets, as a defense-in-depth mechanism, to limit the access to system assets to achieve higher isolation from rest of the system.

Figure 3: LSM hook architecture



3.5.1 SELinux User Personas

Following are the personas defined for a TEP based system. We recommend using ladmin for admin work.

Note: Please change default password of your users at build time.

1. **Security Admin** (**SELinux login–root (system_u, staff_u) mapped with Linux root user**)-is the only one to change SELinux rules. Runtime SELinux rules cannot be modified. Need a physical interface (serial, tty, vga) to access this login. This user could be used for trusted-container setup (one time).
2. **System Admin** (**staff_u SELinux mapped with Linux ladmin**) - Sudo (install applications). This user is part of docker group, which is equivalent to Linux root or sudo. This user shall be used for guest container setup and launch.
3. **User (selinux user_u mapped with Linux guest user)** - ring 3 application access.

#semange login -l

```
Login Name           SELinux User         MLS/MCS Range        Service

__default__          user_u               s0-s0                *
guest                user_u               s0-s0                *
ladmin               staff_u              s0-s0                *
root                 root                 s0-s0:c0.c1023       *
```

Note: You may be able to change SELinux user mapping using SELinux commands at first boot using SELinux security admin “root” from TTY login.

4. **SSH login** with root or ladmin is not same as it is from TTY console, ssh login have privileges which are in sshd_t domain only and will have limited access of that domain only.

Check the status of the SELinux

sestatus

```
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             mcs
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     requested (insecure)
Max kernel policy version:      33
```

3.5.2 Enforcing SELinux to Permissive Mode

To put SELinux in permissive mode, change the SELinux state to 0 as follows:

```
# setenforce 0
```

Note: This command can only be executed by a system admin user with “root” and tty login in default build.

3.6 Sharing TPM Devices to Guest Containers

Sharing of TPM resources across Containers and Host is done through the TPM passthrough device that enables access to the host's TPM for the guest as well as Trusted Container. The Trusted Container must start first and access the TPM device (/dev/tpm0) before any other applications that use Containers are launched.

The resource manager used in sharing mode in the Trusted Container is RM0 which associated with /dev/tpmrm0 devices in the host kernel. By this model, TPM device is shared across host and other guest VMs through RM0's /dev/tpmrm0 device node.

To share the TPM devices to the VMs, verify the TPM availability in the host.

```
root@a4bf0135e9dc:~# ls /dev/tpm*
/dev/tpm0  /dev/tpmrm0
```

TPM devices are shared to the Containers using Docker `--device` Passthrough mechanism.

Note: Only the TEP Trusted Container have a default TPM Passthrough in the launch script. For Guest Container, the user has to add a TPM Passthrough.

3.7 Dmverity Configuration

The Dmverity feature lets you look at a block device and determine if it matches its expected configuration by using a cryptographic hash tree.

3.7.1 Enabling Dmverity for Yocto

This feature is enabled when you select "Enable Dmverity on rootfs" option during the build (refer Setting up a Linux Build System to Build TEP Image section).

No further configuration is required for Yocto.

3.7.2 Enabling Dmverity for Ubuntu

Do the following additional steps on Ubuntu Host to enable DM-Verity.

1. Ensure CONFIG_DM_VERITY is enabled in Ubuntu Kernel.
2. While doing Ubuntu installation, Create Separate Partitions for the directories that need to secure with dm-verity. . To create separate partitions, refer https://askubuntu.com/questions/343268/how-to-use-manual-partitioning-during-installation.
3. Install `cryptsetup` package, use the following command:

```
apt-get install cryptsetup-bin
```

4. Use `veritysetup` format to calculate and hashes and store in Hash device:

```
veritysetup --data-blocks=256 --hash-offset=1052672 format
       <device> <device>

Verification data (hashes) is stored on the same device as
data (starting at hash-offset). Hash-offset must be
greater than number of blocks in data-area.
```

5. Use `veritysetup` to activate the verity device:

```
veritysetup --data-blocks=256 --hash-offset=1052672 create
test-device <device> <device> <root_hash>

Activates the verity device named test-device.

Options --data-blocks and --hash-offset are the same as in
the format command.
The <root_hash> was calculated in format command.
```

3.8 Measured Boot Configuration Using Linux Integrity Measurement Architecture (IMA)

For measured boot of TEP Trusted OS, we use Linux Integrity Measurement Architecture (IMA) to Measure the TEP Secure container and its software collaterals during the boot process and to extend TPM PCR with the hashes. The IMA feature can be enabled in Yocto and Ubuntu based system.

3.8.1 Enabling IMA for Yocto

IMA feature is part of a Yocto code repository. Do the following additional step for enabling IMA for Container Host Kernel:

1. The sample scripts for generate key and certificates is in meta-security/meta-integrity/scripts. Generate keys and certificate for IMA as follows:

```
./ima-gen-self-signed.sh
Generating a RSA private key
..............+++++
......+++++
writing new private key to 'privkey_ima.pem'
-----
./ima-gen-local-ca.sh
Generating a RSA private key
.......+++++
...........+++++
writing new private key to 'ima-local-ca.priv'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
```

2. After the key and certificates are generated copy these files "`ima-local-ca.pem ima-local-ca.priv ima-local-ca.x509 privkey_ima.pem x509_ima.der`" to meta-tep-container/data/debug-keys.
3. While building selects "Enable IMA" from menu-entry, refer Setting up a Linux Build System to Build TEP Image.

3.8.2 Enabling IMA for Ubuntu

To enable IMA feature on Ubuntu, do the following additional steps:

1. Ensure CONFIG_IMA and CONFIG_IMA_WRITE_POLICY is enabled in Ubuntu Kernel.
2. Create new tep partition copy core-image-trusted-os-intel-corei7-64.tar.bz2 to tep partition. To create separate partitions, refer https://askubuntu.com/questions/343268/how-to-use-manual-partitioning-during-installation .
3. Get the universally unique identifier (uuid) of TEP partition using the following command:

```
lsblk -f
```

4. Add in following command, and execute:

```
echo "measure fsuuid=<tep uuid>"
>/sys/kernel/security/integrity/ima/policy
```

4.0 Installing Image

This section will give a brief about how to install TEP container profile images, which are built by following build instructions.

Note: For prerequisites, refer the *Target System* section in the *TEP Release Notes*.

4.1 Bring-up TEP on Yocto

To install SELinux based Yocto wic image, the prerequisites are described in the following sections. Follow the instructions to bring-up SELinux based Yocto host as bare metal OS for TEP docker Containers.

Installation:

1. Boot alternative Linux OS from USB media.
2. Copy **SELinux enabled Yocto image** (core-image-selinux-intel-corei7-64.wic) from your build machine to an above-mentioned USB booted OS or other media and attach media to 11th Gen Intel® Core™ board.

```
dd if=<path of core-image-selinux-intel-corei7-64.wic >
of= /dev/nvme0n1 status=progress
```

3. Reboot and select NVME from boot device from UEFI.
4. Do ssh setup for accessing TPM simultaneously from Container and host.
   - Generate ssh keypair using ssh-keygen tool
   - Create ssh keys and authorized users on host side. These keys will be passed to TEP Container to perform ssh to host.

```
ssh-keygen
cd /home/root/.ssh
cat id_rsa.pub >> authorized_keys
```

4.1.1 TEP Docker Container Setup

Once bare metal host operating system is up and running after installation, you can install, and set up TEP docker Container as follows:

1. Macvlan Network Creation:

   Prerequisites:
   - Get the subnet, gateway details, Ethernet interface

- Create a macvlan network for Container network access

```
docker network create -d macvlan --
subnet=<10.20.30.0/24> --gateway=<10.20.30.1>  -
o parent=< interface-name> my-macvlan-net
```

- Check for macvlan with the following command:

```
docker network ls
```

- my-macvlan-net should be visible in network list.

2. Create a network file with interface to be used for Host macvlan bridge (**only first boot on SELinux host).** For example,

```
echo enp0s20f0u1 >> /var/network_container.txt
```

Note: The IFName needs be less than 16 characters in length.

3. Run the docker setup for TEP Container:

```
docker import /usr/share/container-img/core-image-trusted-
os-intel-corei7-64.tar.bz2 trusted_container:latest
```

- docker_setup.sh expects three command-line argument as follows:
  - Argument 1–number of TTY connections. Example, “3”
  - Argument 2-The mode of operation for pkcs11 "needed only for grpc" else keep it empty double quotes, example,"" or add erpc. example: "grpc".
  - Argument 3-Core pinning (Optional). Provide the core to pin the docker container. Note: Core numbering starts from 0. example: "0".
  - Example: /usr/bin/docker_setup.sh “3" "grpc"
  - Example1 with optional field: /usr/bin/docker_setup.sh “3” “” "0"
  - Example2 with optional field: /usr/bin/docker_setup.sh "3" "grpc" "0"
  - Example3 with optional field: /usr/bin/docker_setup.sh "3" "erpc" "0" #one with erpc pkcs11 ipc.
- $ /usr/bin/docker_setup.sh “3” “grpc” “0”
  - docker_setup.sh creates a host macvlan bridge interface mac0 interface and assign dynamic ip.
  - Releases the host network interface ip. One can use mac0 macvlan interface.
  - grpc is chosen as an ipc communication.
  - core-pinning of container to core “0”.

4. TEP OS Container execution
   - $ docker exec -it trusted_container /bin/sh
   - docker Container shell will be entered.
   - Check the ip assigned to TEP OS Container
5. SFTP operation and commit: First time boot only

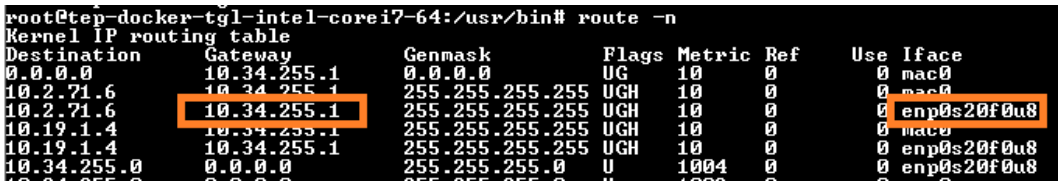
- From Admin machine, perform the SFTP operation for config blob update.
- do device provisioning, follows step–Section 3.1
- Exit Container
- Commit the Container for changes done using the following commands:

```
docker commit trusted_container
trusted_container:latest

docker stop trusted_container
```

6. Relaunch Container
   - $ /usr/bin/docker_setup.sh “3” “grpc” “0”
   - Enter Container shell
     - $ docker exec -it trusted_container /bin/sh
   - Check for LUKS and trustagent
   - For trust agent you should see the logs “tagent start successful”
     - There is trustagent.env required as part of config update.
7. For subsequent SE Linux image boots:
   - TEP OS Container should be launched automatically.
   - LUKS and tagent must start automatically.
8. TEP GRPC Macvlan Network Creation:
   - TEP GRPC socket is set up on local network for which we use VLAN tag based network (<interface>.101) interface and macvlan for name for this bridge is grpc-macvlan-net from host to container. the IP address is fix to 192.168.254.254 and new guest containers except the sample guest container part of release image could be attach on this network for GRPC communication. Following is example.

```
$ docker network ls

$ docker network connect grpc-macvlan-net <container_id>
```

9. Routing table update:
   - The routing table in the TEP OS container must be set up to route the traffic in the default target network IP to the gateway that handles the default target network traffic in the host.
   - Get the gateway configuration from host os using “route -n”,



   - In the trusted container update, the route configurations.

```
ip route change default via 10.34.255.1 dev eth0
```

Note: This is not a mandatory configuration. Network routing table need to be updated when TEP OS container need to access systems outside the subnet.

4.1.2 TEP Guest Container Setup

To set up TEP Guest Container, use the following commands:

```
docker import /usr/share/container-img/core-image-minimal intel-corei7-64.tar.bz2 guest-container:latest
/usr/bin/guest_docker.sh
```

- guest_docker.sh expects 1 command-line argument, which is optional as follows:
  - argument 1 - Core pinning (Optional). Provide the core to pin the docker container. Note: Core numbering starts from 0. example: "0".
  - Example with optional field: /usr/bin/guest_docker.sh “0”.
- Execute `$/usr/bin/guest_docker.sh`
- You will enter docker Container shell.
- Check the ip assigned to TEP guest Container.
- Type Ctrl+P then Ctrl+Q in sequence. It helps to turn exit from Container, without killing (daemon mode).

4.2 TEP Bring up on Ubuntu Host

The following sections describe the method used to bring up AppArmor enabled Ubuntu.

4.2.1 Installation and Setup of TEP Container

Follow the instructions to bring-up Ubuntu host as bare metal OS for TEP docker Containers.

Installation:

1. Boot alternative Linux OS from USB media.
2. Copy **Ubuntu iso image** (usually downloaded from open source) from your build machine to an above mentioned USB booted OS or other media and attach media to 11th Gen Intel® Core™ board.

Note: Use custom.seed file and re-create this iso file to automate the installation process.

```
$dd if=<ubuntu_iot_intel.iso > of=/dev/sdb1
status=progress
```

3. Reboot and select the attached USB storage drive from boot device from UEFI.
4. If automated installation is selected wait until the Ubuntu is installed on NVME. If manual installation is being used, follow through with the required prompts from the installer to finish setup, and wait until the installer completes.
5. Reboot and select the NVME to boot Ubuntu.
6. Do ssh setup for accessing TPM simultaneously from Container and host.

- Generate ssh keypair using ssh-keygen tool.
- Create ssh keys and authorized users on host side. These keys will be passed to TEP Container to perform ssh to host.

4.2.2 Installing TEP Debian Packages

Get the Debian package from the unified build setup by running deb_pack.sh. This will create four Debian packages

- `tep_<release_version>_guest_image_amd64.deb`
- `tep_<release_version>_ipc_amd64.deb`
- `tep_<release_version>_setup_amd64.deb`
- `tep_<release_version>_trusted_image_amd64.deb`

Install the Debian packages in the newly installed Ubuntu.

```
dpkg -i tep.*.deb
```

Note: The next steps assume that following are installed. Use “apt install” to do the install.

- containerd.io
- docker-ce-cli
- docker-ce
- net-tools
- uhdcpc

4.2.2.1 Setting up TEP Docker Container

Follow the same steps mentioned in section Installation and Setup of TEP Container to bring up TEP trusted container and guest containers.

5.0 Provisioning Device

Trusted OS uses Intel PTT as default TPM for root cryptographic keys and root of trust for OS secure operations. If you want to switch to Discrete TPM, you can configure it in the BIOS menu.

TPM device on platform needs to be provisioned with AES-256 user key for confidentiality and ECC-384 public key for verification. These two keys must be provisioned in secure environment before the device is ready for trusted Container. Following are the sample steps that you can follow to provision these attributes into TPM.

5.1 Prerequisite for Device Provisioning

For Admin machine requirement, refer to the TEP Release Notes.

Admin must use the AES-256 symmetric key for confidentiality and the ECC-384 public key for verification.

5.2 TPM Device Provisioning

Refer to the sample_device_provisioning_script for Linux platform. The script is available in the release archive: targets/resources/meta-tep-trusted-os/recipes-core/tep-user-config/files/tpm-authorizations/tep_device_provision_sample.py. These sample operations provide device key creation and provisioning in TPM.

1. To generate a sample ECC key, do this step on the host machine:

```
openssl ecparam -genkey -name secp384r1 -out
pcr_pol_signing_key_priv.pem

openssl ec -in pcr_pol_signing_key_priv.pem -out
pcr_pol_signing_key_pub.pem -pubout
```

2. To generate a 256-bit AES key or use one from the host system, which you are using for encryption in the preceding script.

```
tpm2_getrandom -o tep_config_data_aes_key.bin 32
```

Note:

You can use different random number generator, that is

```
head -c 32 /dev/urandom >
tep_config_data_aes_key.bin
```

1. Use the keys (ECC public key and AES key) from step #1 and #2 and from command-line (that is, Serial port) on TEP machine use tep_device_provision_sample.py to perform device provisioning.

```
python3 tep_device_provision_sample.py -
pol_pub_key pcr_pol_signing_key_pub.pem -enc_key
tep_config_data_aes_key.bin
```

Note:

- Clear TPM before provisioning using command tpm2_clear and also make sure that there is not any stale tpm2_pkcs11.sqlite3 partition-1, which is mounted partition for PKCS11 objects.
- Remove tep_config_data_aes_key.bin from device and store a copy of this key on admin machine for encryption. Note: This is not secure. Preferably add srm/shred to TEP and use that.

Following NVIndexes are used for ECC public and AES symmetric keys:

```
oem_tep_policy_signing_key_nv_idx=0x018A0000

oem_tep_config_data_aes_key_handle=0x8100A000
```

5.3 Trusted Container Provisioning for User Configuration

After device provisioning is completed, you can proceed into Trusted Container provisioning steps. This is the first step required when system first time boots with trusted container and ready for configuration. The prerequisite for this step is to have device provisioned with user keys.

5.3.1 Creating User Configuration Signing and Encryption

Trusted container will accept encrypted and signed user configuration data. Once data is transfer to trusted container, on next reboot tep_user_config daemon will look for a blob at specific location and will verify it and then decrypt (verify-then-decrypt) using keys stored in TPM. For verification we use ECDSA and for decrypt TEP will use AES

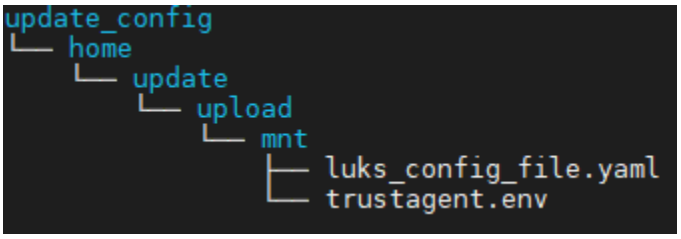
CTR mode. Followings are steps to be followed to create encrypted and signed user config data and then verify it on system.

5.3.1.1 Encryption and Signing of User Config Data at Host Machine.

Following is one sample way to create user configuration.

1. Create user config data in file/files in required folder Hierarchy. Following is an example.

Figure 4: Sample Config Files Tree Structure



2. Create a .tgz file of it. (As this will reduce the size)

```
tar -cvzf update_config.tgz <update_config>
```

Encrypt and signed this .tgz file with given sample host tool. `tep_encrypt_signed _user_config.py` file is available in the release archive: `targets/resources/meta-tep-trusted-os/recipes-core/tep-user-config/files/encrypt_sign_tool/tep_encrypt_singed_user_config.py`

```
python3 tep_encrypt_signed _user_config.py
<update_config.tgz> <ecc_key>  <aes_key>
```

The output is as follows: tep_user_config_data.bin

Note: Refer TPM Device Provisioning section for ECC and AES keys.

- ECC private keys, which are associated with the public key provisioned in the device are used.
- Same Aes-256-bit keys shall be used which is provisioned in the device.

Above will give you encrypted and signed blob, which can be transferred to target platform at 'update' users mount point, which is unencrypted storage partition (/home/update/upload/mnt/).

```
sftp -
o "IdentityFile=../sftp_key/update_user_key_for_dev.pem
" update@tep-machine:upload/mnt/ <<< $'mput *'
```

Note:

Sample asymmetric key files (update_user_key_for_dev.pem) for authorized host key for SFTP is used here. Customer key file names may be different. Refer section 1.2.1 for more details.

Change permission of file update_user_key_for_dev.pem by using the following command:

```
chmod 0444 update_user_key_for_dev.pem.
```

5.3.1.2 Authentication and Decryption of User Config Data at TEP Machine

Trusted OS have the tep_user_config_update.service which, have sample implementation. To authenticate and decrypt user config data, perform following steps in sequence.

1. Check /home/update/upload/mnt/tep_user_config_data.bin file at boot.
2. If this file exists, then this service will start verification process.
3. Parse and authenticate tep_user_config_data.bin file for valid signature.
4. If authentication successful decryption will be followed in /opt/.
5. Apply decrypted user configuration on the system and restart appropriate system services.

Note:

The update of the respective config is end user dependent. The `tep_user_config_update.service` service starts the sample implementation. You can find the script in the following release archive:

```
targets/resources/meta-tep-trusted-os/recipes-core/tep-
user-config/files/tep_user_config_update.sh
```

5.4 Disk Encryption Provisioning

TEP devices use secured drives for storing the confidential information. LUKS encrypted drives are configured as part of the TEP device provisioning.

LUKS uses a signed PCR policy for storing and retrieving LUKS device passphrase. `luks_config_file.yaml` would have all the required configurations required by Disk encryption service. `luks_config_file.yaml` must be updated along with TEP user config data described in Creating User Configuration Signing and Encryption section.

For creation Disk encryption service config files, refer to Disk-Encryption Service.

5.5 Remote Attestation Provisioning

Trustagent is used for performing the Remote attestation in TEP devices. Trustagent must be provisioned with TEP device provisioned. Trustagent answer file `trustagent.env` must be updated along with TEP user config data described in the preceding section.

For creation of `trustagent.env` and its update process, refer to TEP Trustagent.

6.0 Interface Library and Micro-Services

6.1 TEP IPC Service

Trusted Container consists of standard concept of RPC Server and Client. To provide a homogenous application interface to ‘`tpm2 pkcs11 module` from guest container to Trusted container. Server-side listener application is called ‘`tep_ipc_daemon`’ which, will respond on the pkcs11 request from guest CONTAINERs. Client-side example application is compiled to ‘`tep_test_app`’. This demo application provides uses of tpm2 pkcs11 APIs.

TEP IPC service uses gRPC to provide the pkcs11 functionality to the client. gRPC internally uses an IP address to communicate its requests.

Server-Side daemon ‘`tep_server_daemon`’ is started by entrypoint script and links shared libs that encapsulate the RPC server implementation.

On Guest OS, user applications need to link with just one TEP library ‘`libipc_interface.a`’ (part of deliverables), which expose the required pkcs11.h interface, to access PKCS11 APIs. This library implements the RPC client side internally. The TEP implementation also requires gRPC and protobuf to be installed in the guest os.

In the current release, multiple apps can talk to trusted Container. Multiple PKCS11 apps can make use of the same IP address.

6.2 IPC Service and Interface Library

6.2.1 IPC Daemons Service for Trusted Container

The ‘ipc-pkcs11.target’ systemd service is launched at boot time. This initializes TPM stack and run server-daemon. In summary, there is no need to run anything on Trusted Container side, it's ready to accept the PKCS11 calls from Guest OS app.

6.2.2 Interface Library for Service-OS

On guest OS, add pkcs11 sample client apps and library as part of release package. Also add grpc and protobuf.

Following application and binaries are part of target build package:

- libtep_interface.a library
- demo pkcs11app binary (tep_test_App)

Similarly, more user applications can be built and run.

6.2.3 Running the Sample Application

The client apps can make use of IP address to talk to the Sec-OS. More than one process can use them. To choose the require select we need to run the `export` command.

Selecting the right IP address.

```
export TEP_IP_ADDR=192.168.254.254:50051 <choose correct value)
```

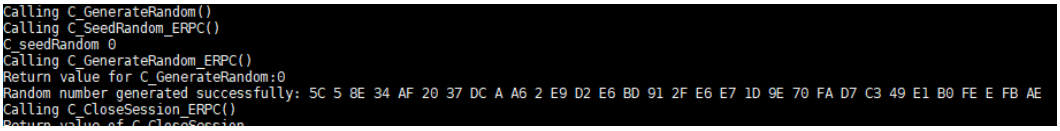
Run the demo app or your own app from same shell

```
tep_test_app
```

The application will exercise RSA, AES, Digest, and random number generation mechanism provided by TPM through pkcs11 interfaces. Following is snapshot of expected results.

RSA:

Random Number Generator:



AES operation on message Digest:

```
digest
5d fb ab ee df 31 8b f3 3c 9 27 c4 3d 76 30 f5 1b 82 f3 51 74 3 1 35 4f a3 d7 fc 51 f0 13 2e
Get AES Key object....
Calling C_FindObjectsInit_ERPC()
Calling C_FindObjects_ERPC()
AES object count 1 found
Calling C_FindObjectsFinal_ERPC()
AES Encrypt ....
Calling C_EncryptInit_ERPC()
Calling C_Encrypt_ERPC()
AES Decrypt ....
Calling C_DecryptInit_ERPC()
Calling C_Decrypt_ERPC()
Message was successfully decrypted!
AES Encrypt using Update & Final....
```

6.3 Disk-Encryption Service

This section contains the details of the provisioning steps required for generating LUKS config files used by LUKS service. It involves steps on the Admin machine and getting data from the target. Following are the steps:

- Retrieving TEP Platform PCR values in the **Target**
- Generating TEP LUKS Config Data file in the **Admin** machine

Note: We assume that customer has golden PCR database of PCR banks of the Target device. You can use highest the available SHA384 or above PCR bank if it is enabled on TEP machine, use "sha384:0,7,10" in PCR operations.

To retrieve TEP platform PCR values in the TEP system, do the following:

1. LUKS partition passphrase is sealed to TPM PCR's 0, 7 and 10. Value in these PCR is a function of BIOS, GRUB, kernel, and Trusted VM code. Follow below steps to gather the PCR values:
2. Copy PCR values from golden PCR database or from a good know device to Admin machine to pcr0_7_10.dat file. This file shall contain pcr values of pcr0, 7, and 10, this need to be provided by customer.

   Retrieve PCR values from target when TPM active PCR bank is SHA256,

   ```
   tpm2_pcrread -o pcr0_7_10.dat "sha256:0,7,10"
   ```

   Retrieve PCR values from target when TPM active PCR bank is SHA384,

   ```
   tpm2_pcrread -o pcr0_7_10.dat "sha384:0,7,10"
   ```

To generate TEP LUKS Config Data file in the **Admin** machine, do the following:

1. Use create_luks_pcr_policy.py to generate various ingredients needed for LUKS passphrase setup. Create_luks_pcr_policy.py is available in the release archive: targets/resources/meta-tep-trusted-os/recipes-core/tep-disk-encryption/files/create_luks_pcr_policy.py
   - **Authorized PCR Policy**: This is a digest of PCR policy which contains info regarding the PCR signing public key and the PCR's included (that is. 0, 7 and 10)
   - **Signed PCR Policy**: This is a digest of current values of PCR 0, 7 and 10. And, this is signed using PCR Signing private key.

   ```
   python3 create_luks_pcr_policy.py -pol_pub_key
   pcr_pol_signing_key_pub.pem -pol_priv_key
   pcr_pol_signing_key_priv.pem -pcr_val_file
   <pcr0_7_10.dat> -tpm_type ptt
   ```

2. Use `create_luks_config.py` to generate a YAML configuration file, which stores all relevant luks configuration data. Create_luks_pcr_policy.py is available in the release archive: targets/resources/meta-tep-trusted-os/recipes-core/tep-disk-encryption/files/create_luks_pcr_policy.py

   ```
   python3 create_luks_config.py -dev_part /dev/sda3 -
   auth_pol authorized.policy -pol_file
   pcr0_7_10.pcr.policy -pol_file_sign
   pcr0_7_10.pcr.signature -pcr_bank sha256
   ```

   Note: `-dev_part` is configurable partition and `-pcr_bank` also can be changed in case it is enabled in grub.

3. Copy luks_config_file.yaml (from above step) into update_config/home/update/upload/mnt directory.
4. Follow Creating User Configuration Signing and Encryption for signed configuration.

Trusted OS ships with a few scripts in the file system to aid in the initial setup of the LUKS functionality. The `tep_luks_module.py` script exist in /opt/tep-luks/ tep_luks_module.py.

- Decryption of `tep_user_config_update.bin` will generate `luks_config_update.yaml` and copy to /home/update/upload/mnt folder, which will be used for tep_luks_module.py execution.
- tep_luks_module.py to setup LUKS initialization.
- tep_luks_module.py will read luks_config_file.yaml and encrypt partition first time.

- In subsequent boots, tep_luks.service systemd service will invoke tep_luks_module.py and do LUKS decryption to unseal LUKS passphrase and provide to dm driver in kernel for decryption and integrity protection.

Note:

Once LUKS partition is configured and mounts, remove old sshd host keys, and regenerate new keys in mounted partition as follows. This only needs to be done at first boot. After that these keys will remain persistent in LUKS drive.

Remove older keys and regenerate sshd hostkeys in LUKS mounted partition. (/home/root/tep_luks_dev is our luks mount directory)

1. rm
   - `/home/root/tep_luks_dev/ssh_host_rsa_key`
   - `/home/root/tep_luks_dev/ssh_host_ecdsa_key`
   - `/home/root/tep_luks_dev/ssh_host_ed25519_key`
2. systemctl restart sshdgenkeys.service

Following NVIndex is used for the sealed LUKS passphrase:

- `luks_passphrase_handle=0x8100_A001`

6.3.1 Debugging LUKS Failure

Re-use a partition for repeat testing for LUKS enablement in the following scenarios:

- If crypt setup detects presence of LUKS header in the beginning of a partition, it will not set up LUKS again. It checks using the following command:
  `cryptsetup isLuks /dev/sda3 && echo $?` (If 0, LUKS header is present)
- To remove this Luks header, execute the following command on the USB from a Linux machine. This will wipe out LUKS header:
  `dd if=/dev/zero of=/dev/<luks_parition_id> bs=100M count=1`

6.4 Remote Attestation Service

Attestation refers to the process of authenticating and attesting to the state of a remote platform and its operating system. This use case is for foundational security.

TEP uses the iSecL framework for remote attestation use cases.

6.4.1 Attestation Components

To setup attestation use cases, you require the following:

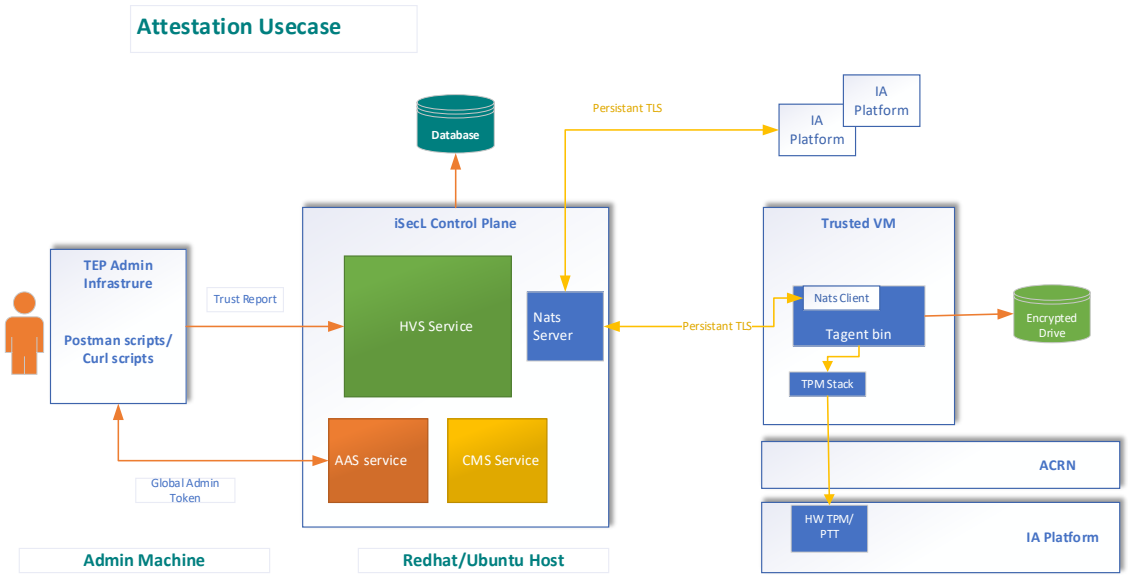
- Remote attestation verifier system (iSecL control plane).

Refer "Remote Attestation verifier system" section.

- TEP Admin infrastructure setup
- Trust agent component integrated with TEP target system.

Following figure shows the attestation components for TEP:

Figure 4: System View of the Attestation Infrastructure



6.4.2 TEP Trust Agent

Trust Agent resides on TEP trusted VM and enables both remote attestation and the extended chain of trust capabilities.

- It provides host specific information.
- It provides secure attestation quotes.
- Allows secure attestation quotes to be sent to the Verification Service.

6.4.2.1 TrustAgent Provisioning and Setup Configuration

Trustagent answer file (`trustagent.env`) is a configuration file with details for provisioning. It has attestation verifier URLs to connect, TEP device identifier name, jwt token, and TLS digest details. A sample `trustagent.env` is provided.

Following are the steps for configuring the trustagent answer file and its update process:

- TEP admin must create a Trustagent answer file `trustagent.env`. Reference file is provided with release package.

```
TA_TLS_CERT_CN=Trust Agent TLS Certificate
HVS_URL=https://<Ip address or hostname of HVS>:30443/hvs/v2

AAS_API_URL=https://<Ip address or hostname of AAS>:30444/aas/v1

CMS_BASE_URL=https://<Ip address or hostname of CMS>:30445/cms/v1
SAN_LIST=<Comma-separated list of IP addresses and hostnames for
the TAgent matching the SAN list specified in the populate-users
script; may include wildcards>

CMS_TLS_CERT_SHA384=<CMS TLS Digest>

BEARER_TOKEN=<CUSTOM CLAIM TOKEN>

TA_SERVICE_MODE=outbound
NATS_SERVERS=< nats-server-ip>:4222

#unique HOST ID
TA_HOST_ID=< Any unique identifier for the host>
```

- Update `trustagent.env` to TEP OS,
  - Copy the trustagent.env to update_config/home/update/upload/mnt in TEP update package.
  - Follow Creating User Configuration Signing and Encryption section.
- Align system time, which aligns to TEP attestation server and save using hwclock.
- Init service will invoke to setup of Trustagent once LUKS initialization and decrypt the storage drive is complete.
- Trustagent saves the certs and logs files generated during providing on to LUKS encrypted drive at `/home/root/tep_luks_dev/trustagent`.

6.4.3 Remote Attestation Verifier System (Intel® SecL Control Plane)

iSecL control plane is the server/system for attesting the platform integrity. Following are the components required for foundational security:

- Postgres database
- Certificate Management library component
- Authentication and Authorization components
- Host verification service
- Nats Server configuration

Follow the iSecl GitHub documentation and product guides for building and setting up the control plane. In depth parameter for Attestation verifier components can be

checked in the documentation. Remote Attestation Service section has a simplified version of setup that you can follow. Please change the parameter in the environment file to your own configuration.

6.4.3.1 Intel® SecL Information

TEP 3.0 uses the Intel® SecL v4.2 and following are the documentations.

Document Type	Document Reference
Product Guide	https://intel-secl.github.io/docs/4.2/product-guides/Foundational%20%26%20Workload%20Security/01/
Quick Start Guide - Build	https://intel-secl.github.io/docs/4.2/quick-start-guides/Foundational%20%26%20Workload%20Security%20-%20Containerization/5Build/
Quick Start Guide - Deployment	https://intel-secl.github.io/docs/4.2/quick-start-guides/Foundational%20%26%20Workload%20Security%20-%20Containerization/TWP%20Control%20Plane%20-%20Helm%20based%20deployment/

6.4.3.2 Remotes Server Configuration and Prerequisites

- Machine: Refer to the *Attestation Verifier System Requirements* section of the TEP release notes.
- Preferred user for build and deployment: Root
- Create a docker user https://hub.docker.com/

6.4.3.3 Building Attestation Server Components

Build Docker Images reference guide - https://intel-secl.github.io/docs/4.2/quick-start-guides/Foundational%20%26%20Workload%20Security%20-%20Containerization/5Build/

- Install the following as per the above quick start build guide.
  - Development Tools and Utilities
  - Repo Tool
  - GoLang
  - Docker
  - Apply the required proxy settings for docker service if you are running behind proxy. Do it for http_proxy, https_proxy, no_proxy, HTTP_PROXY, HTTPS_PROXY, NO_PROXY.
  - Add the proxies for building TEP VM.

```
mkdir -p ~/.docker/

touch ~/.docker/config.json

cat << EOF > ~/.docker/config.json

{

  "proxies": {

    "default": {

      "httpProxy": <proxy>,

      "httpsProxy": <proxy>,

      "noProxy": "<no proxy ip's>"

    }

  }

}

EOF
```

- Following the section and "Build OCI Container images and K8s Manifests" as per quick start guide,
  - Images are located at /root/intel-secl/build/fs/k8s/
- Build util docker images (run from repo root)
  - docker build -t isecl/nats-init:v4.2.0 -f utils/tools/containers/nats/Dockerfile
  - docker build -t isecl/init-wait:v4.2.0 -f utils/tools/containers/init-wait/Dockerfile
- Check the docker image list:

```
root@ubuntu20vm:~/taas/isecl_42# docker image ls -a | grep isecl
isecl/nats-init                      v4.2.0    8610cd62d68b   About a minute ago   183MB
localhost:5000/isecl/nats-init       v4.2.0    8610cd62d68b   About a minute ago   183MB
isecl/init-wait                      v4.2.0    9485daa64791   4 minutes ago        10.9MB
localhost:5000/isecl/init-wait       v4.2.0    9485daa64791   4 minutes ago        10.9MB
localhost:5000/isecl/aas-manager     v4.2.0    8aac5cb76a2c   30 minutes ago       230MB
isecl/aas-manager                    v4.2.0    8aac5cb76a2c   30 minutes ago       230MB
isecl/authservice                    v4.2.0    a2828b15cfa9   20 hours ago         86.5MB
isecl/hvs                            v4.2.0    c320703a4f3d   20 hours ago         155MB
localhost:5000/isecl/hvs             v4.2.0    c320703a4f3d   20 hours ago         155MB
isecl/cms                            v4.2.0    b49ecdb001fc   20 hours ago         85.2MB
localhost:5000/isecl/cms             v4.2.0    b49ecdb001fc   20 hours ago         85.2MB
localhost:5000/isecl/authservice     v4.2.0    0abd9f9a2b85   20 hours ago         86.5MB
```

6.4.3.4 Deploying Attestation Server Components

Quick start reference guide: https://intel-secl.github.io/docs/4.2/quick-start-guides/Foundational%20%26%20Workload%20Security%20-%20Containerization/TWP%20Control%20Plane%20-%20Helm%20based%20deployment/

Helm-chart and use case for TEP 3.0: Trusted Workload Placement - Control Plane

Follow the instructions for deployment of attestation server components.

- Start and push images to local registry for hvs, aas, cms, aas-manager, init-wait, nats-init)
  - Start registry using the following command:

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

  - Push container images to registry, using the following command:

```
docker tag isecl/hvs:v4.2.0 localhost:5000/isecl/hvs:v4.2.0

docker push localhost:5000/isecl/hvs:v4.2.0

docker tag isecl/authservice:v4.2.0 localhost:5000/isecl/authservice:v4.2.0

docker push localhost:5000/isecl/authservice:v4.2.0

docker tag isecl/aas-manager:v4.2.0 localhost:5000/isecl/aas-manager:v4.2.0

docker push localhost:5000/isecl/aas-manager:v4.2.0

docker tag isecl/cms:v4.2.0 localhost:5000/isecl/cms:v4.2.0

docker push localhost:5000/isecl/cms:v4.2.0

docker tag isecl/init-wait:v4.2.0 localhost:5000/isecl/init-wait:v4.2.0

docker push localhost:5000/isecl/init-wait:v4.2.0

docker tag isecl/nats-init:v4.2.0 localhost:5000/isecl/nats-init:v4.2.0

docker push localhost:5000/isecl/nats-init:v4.2.0
```

- Install the Helm and add helm charts as per the deployment guide
- Set up NFS for persistent volume as per the deployment guide
- Create values.yaml file for TEP/K3s (reference: https://raw.githubusercontent.com/intel-secl/helm-charts/v4.2.0-Beta/usecases/twp-control-plane/values.yaml)
- Setup the Kubernetes K8s or light weight Kubernetes K3s. Following are the instructions for K3s.

6.4.3.5 Setting up K3s

To install K3s, use the following commands:

```
export INSTALL_K3S_EXEC="server --no-deploy traefik"
export INSTALL_K3S_CHANNEL=stable
export INSTALL_K3S_VERSION=v1.21.9+k3s1
export K3S_KUBECONFIG_MODE="644"
export http_proxy=http://proxy-us.intel.com:911
export https_proxy=http://proxy-us.intel.com:911
export
no_proxy=localhost,127.0.0.0/8,10.0.0.0/8,.intel.com
curl -sfL https://get.k3s.io | sh -
exit
```

To check if K3s is active and running, use the following command:

```
sudo systemctl status k3s
```

To get k3s nodes, use the following commands:

```
# Update KUBECONFIG in $HOME/.profile
cat << EOF >> $HOME/.profile
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
EOF
source $HOME/.profile
# Talk to K8s cluster
kubectl get nodes
```

To create registry credential secret for k8s, use the following command:

```
cp /root/.docker/config.json $HOME/docker_config.json
kubectl create secret generic registrycreds --from-
file=.dockerconfigjson=$HOME/docker_config.json --
type=kubernetes.io/dockerconfigjson
rm -rf $HOME/docker_config.json
```

To restart k3s, use the following command:

```
sudo systemctl restart k3s
```

6.4.3.6 Use Case Based Helm Deployment

Install the helm chart for TEP remote attestation server control plane,

```
helm pull isecl-helm/Trusted-Workload-Placement-Control-Plane

helm install isecl-4.2 isecl-helm/Trusted-Workload-Placement-Control-Plane --create-namespace -n isecl -f values.yaml
```

To uninstall helm deployment, use the following command:

```
helm uninstall isecl-4.2 -n isecl
```

:

To check micro services, use the following command:

```
curl --location --request GET 'https://<IP>:30445/cms/v1/version' -k --noproxy "*"

curl --location --request GET 'https://<IP>:30444/aas/v1/version' -k --noproxy "*"

curl --location --request GET 'https://<IP>:30443/hvs/v2/version' -k --noproxy "*"
```

6.4.4 TEP Admin Attestation Infrastructure

TEP admin has the following tasks for setting up the attestation use cases:

1. Create `trustagent.env` required for TEP device trustagent provisioning.
2. Get global admin token using the respective user id and password for communicating with Attestation verifier (iSecL control plane).
3. Creation of TEP device flavors,
   - Post flavor group templates - create a flavor template for TEP project and post to HVS.
   - Post flavor group - create a flavor group required and post to HVS.

- Boot a golden host with Trust agent provisioned. Import flavors from golden host.

4. Host registration and generation of reports
   - Register hosts required. Use the host names TA_HOST_ID used in with provisioning Trustagent on TEP targets.
5. Following are the various report generation options:
   - Create Trust report - use the TA_HOST_ID. This creates trust report speaking to respective TEP devices
   - List Reports - This generates reports for all available devices register
   - Security Assertion Markup Language Report
   - All Hosts - Provides status of all TEP devices connected.
   - Report generation details are captured in intel-secl product guide at the following location: https://intel-secl.github.io/docs/4.2/product-guides/Foundational%20%26%20Workload%20Security/60%20Platform%20Integrity%20Attestation/#attestation-reporting

6.4.4.1 Postman Scripts

Intel-iSecl provides post man scripts and API collection's for iSecL control plane to be used in postman environment. Postman collection provides majority of the functionality.

https://github.com/intel-secl/docs/blob/v4.0/develop/quick-start-guides/Foundational%20&%20Workload%20Security.md#5-usecase-workflows-api-collections

Admin user can customize or add these scripts as per the TEP use case requirements. Sample collections for TEP are provided with release package with trustagent recipe at `os.security.tep.meta-tep-trusted-os/recipes-ta/api-collections/.`

6.4.4.2 Flavor Configuration

A Flavor is a standardized set of expectations that determines what platform measurements will be considered “trusted.” Following are the configurations required for TEP:

- Flavor templates
- Flavor groups
- Flavor import

Link for flavor configuration
https://intel-secl.github.io/docs/4.2/product-guides/Foundational%20%26%20Workload%20Security/90%20Flavor%20Management/

TEP uses custom flavor templates and flavor groups. A reference JSON is provided with release package. These templates are to be updated along with postman collection provided in preceding section.

Flavor Templates expose the backend logic that determines which PCRs and event log measurements will be used for specific Flavor parts. These are conditional rules that apply to a Flavor part cumulatively based on defined conditions. Use the reference template to create custom flavor template.

A Flavor Group represents a set of rules to satisfy for a set of flavors to be matched to a host for attestation. TEP supports flavor groups consisting of Platform and OS flavor types.

6.4.4.3 Bearer Token

Bearer token is used to authorize the TEP device while performing the TrustAgent provisioning, and this token must be updated as part of `trustagent.env`.

Intel-secl supports various tokens and superAdminUsername can be used to do trustagent provisioning.

It is recommended to create a new user with isecl attestation verifier, which would have only limited permissions for performing the provision steps as download-ca-cert, download privacy ca, EK & AIK provision. superAdminUsername can be used for engineering purposes. Intel-secl product guide provides steps for creation on new user.

6.4.4.4 Example for Getting Bearer Token

```
curl --header "Content-Type: application/json" --request
POST --data '{"username":"
superAdminUsername","password":"superAdminPassword"}'
https://<hostname>:30444/aas/v1/token  --noproxy "*" -k
```

Token generated can be used as BEARER_TOKEN in trustagent.env.

6.4.5 Debug Logs

- Before setting up LUKS Encrypted drive:

Debug logs from device provisioning, user config update, disk encryption and TPM2_TSS are logged at /var/log/tep_logs.

- After setting up LUKS Encrypted drive:

Once LUKS encrypted drive is setup debug logs up to that point from device provisioning, user config update, disk encryption and TPM2_TSS are copied to it, whose file path is /home/root/tep_luks_dev/tep_logs, and new logs will be appended to this.

TrustAgent related logs are logged at /home/root/tep_luks_dev/logs/trustagent

7.0 Intel Recommendations

Following are the Intel recommendations for system security.

- Change 'root/ladmin/guest' and 'update' user's password in your Yocto build recipe. Also, make sure that you set root and user password on servicer-OS, Guest OS/containers.
  - 'root/ladmin/guest' users
    find recipe in the release archive:
    `targets/resources/meta-tep- trusted-os/recipes-core/images/core-image-trusted-os.inc`
  - 'update' user
    find recipe in the release archive:
    `targets/resources/meta-tep-trusted-os/recipes-core/tep-disk-encryption/add-tep-update-user.bb`
- Keep the Trusted OS, GRUB, and BIOS stacks up to date with patches.
- In the production system:
  - Close all debug interfaces, including JTAG and Serial connections.
  - Disable memory dumps.
  - BIOS Menu lock down with password.
  - Out-Of-band provisioning of UEFI keys must be disabled.
- Recommend changing the HOST name during provision to device unique value, this can be either achieve using customized installer for Yocto image or change at build time by adding/modifying following in your local.conf and have a system service to make it device unique at boot.
  - `hostname_pn-base-files = "your_hostname_here"`
- It is recommended to use AES-CTR-256, and ECC-384 crypto algorithms for better resistant soon.
- To protect against an adversary with physical access, the system needs to support TME, VxD with the encrypted disk.
- It is recommended that Ethernet interface name should not be more that 13 characters. If host system Ethernet interface name is more than above limit use, change the interface name as follows:
  - Update file /etc/udev/rules.d/70-persistent-net.rules and reboot system

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="<MAC_ADDRESS>", ATTR{dev_id}=="0x0",
ATTR{type}=="1", KERNEL=="eth*", NAME="eth1"
```