Trusted Edge Platform for IoT Platforms

User Guide

August 2021

Intel Confidential

Document Number:

Contents

Revision History

Date	Revision	Reference #	Description
August 2020	0.1		First draft
October 2020	0.2		Changes for PV
Jan-2021	0.3		Updated for PV2.0 release
May-2021	0.4		Updated for PV2.1 release
Augist-2021	0.5		Updated for PV2.2 release

References

Reference	Modules/Owner	Description
1	ACRN	https://projectacrn.org/
2	Yocto	https://www.yoctoproject.org/
3	ECS	http://wheeljack.ch.intel.com/ECS-Documentation/index.html
4	TPM2_PKCS11 Stack	https://github.com/tpm2-software/tpm2-pkcs11
5	TPM2 TSS Stack	https://tpm2-software.github.io/
6	PKCS#11 Spec	http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/csprd02/pkcs11-base-v2.40-csprd02.html
7	P11 kit	https://p11-glue.github.io/p11-glue/p11-kit/manual/
8	Intel-isecl	https://github.com/intel-secl/intel-secl

Definitions and Acronyms

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 Introduction

1.1 Document Description

This document is intended to serve as an integration and user guide for the TEP trusted VM and container configuration in IoTG projects. For an overview of the Trusted VM and container configuration, please refer to the **Trusted VM Quick Start guide**.

This document describes features supported for production release. This release is tested on Tiger Lake rvp platform based on TGL-U and Yocto Linux based guest-VM for hypervisor and SELinux enabled yocto host for container flavor.

This version of Trusted VM/container uses an Intel PTT as physical hardware TPM as POR and limited testing done on dTPM. The intent is to provide an TPM based release to customers for their development purpose.

1.2 Build Trusted OS as standalone image

Below are the instructions to build a Trusted OS as standalone or with acrn as a unified build system. ACRN build is for reference integration example, end customer may change it as per their requirements. For acrn build with trusted OS we are referring meta-acrn from acrn opensource yocto layer in our sample integrations. Packaging of this standalone image into acrn target need to be done as part of acrn build. In next section we will explain how could unified build be make with acrn and trusted os image. Building stand-alone trusted OS is still important for development and component wise deployment.

Note: you may need to setup your build host for yocto build. You can follow standard yocto guidelines . Following few common packages may need to be installed in ubuntu. These build instructions are based on TGL yocto bkc.

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib
build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils
debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev
pylint3 xterm
```

1.2.1 Clone Trusted OS meta-layers

1. Trusted OS standalone build instructions are here, these instructions are assuming that you have access to the Intel Gitlab repositories.
2. Refer to the latest README from: https://gitlab.devtools.intel.com/OWR/IoTG/SMIE/Security/secure-computing/tep-docs/-/blob/tep_2.2_release/docs/BUILDING.md#trusted-os-standalone-build

Make a new directory.

```
$ mkdir <work_dir>
$ cd <work_dir>
```

Use repo to pull the repositories.

```
$ repo init -u
ssh://git@gitlab.devtools.intel.com:29418/OWR/IoTG/SMIE/Security/secure-
computing/meta-tep-manifests.git -b refs/heads/trusted-os_2.2 -g all
```

Pull meta-layers

```
$ repo sync -c -j$(nproc) --force-sync
```

1.2.2 Build Trusted OS Image for target

Once the cloning of the repositories is completed, use following instructions to build trusted OS image.

Set up Build Environment

```
$ sed -i 's/meta/meta-tep-trusted-os/g' ./openembedded-core/.templateconf
$ source ./openembedded-core/oe-init-build-env
```

Copy multiconfig files from meta-tep-trusted-os layer

```
$ mkdir -p conf/multiconfig/
$ cp -r ../openembedded-core/meta-tep-trusted-os/conf/multiconfig conf/
```

Build image for tgl-u target

```
$ bitbake mc:x86-tep-trusted-os-tgl-initramfs:core-image-trusted-os
```

Trusted-OS image:

build/tmp-x86-tep-trusted-os-tgl-initramfs-glibc/deploy/images/intel-corei7-64/core-image-trusted-os-intel-corei7-64.wic

Trusted-OS Container Image:

build/tmp-x86-tep-trusted-os-tgl-initramfs-glibc/deploy/images/intel-corei7-64/core-image-trusted-os-intel-corei7-64.tar.bz2

1.3 Build Trusted OS with ACRN hypervisor as reference unified build

This is sample unified build approach using yocto multi-configuration where one can build acrn hypervisor and trusted-os as one target image. Alternatively, you can build trusted os as standalone and then deploy build image in acrn build path to package it.

Note: Make sure that changes are done as per your requirements and common yocto recipes are properly masked in respective multi-configuration files to avoid applying changes in both SOS and Trusted-OS.

Following are some recommended configuration for pre-launched trusted VM in ACRN hybrid_rt scenario config file. Refer xmls-2.3.tar.xz sample changes.

1. Disable the ivshmem which is enabled by default.

2. Increased the size of memory allocated to pre-launch VM.

"<size desc="The memory size in Bytes for the VM">0x80000000</size>"

3. Changed the boot arguments to ensure booting of TEP secure OS as pre-launch VM.

4. Increased the number of vUARTS for interVM communication from 2 to 6.

5. Passthrough a USB storage device to pre-launch VM for storage.

6. Enable the TPM device passthrough to pre-launch VM by enabling mmio passthrough.

1.3.1 Clone acrn and Trusted OS meta-layers for unified build.

1. ACRN Unified build system with Trusted OS are given as follow, these instructions are based on intel gitlab repositories.
2. Refer to the latest build guide from:

   https://gitlab.devtools.intel.com/OWR/IoTG/SMIE/Security/secure-computing/tep-docs/-/blob/tep_2.2_release/docs/BUILDING.md#acrn-unified-build

Make a new directory.

```
  $ mkdir <work_dir>
  $ cd <work_dir>
```

Git clone the repo manifest.

```
 $ repo init -u
ssh://git@gitlab.devtools.intel.com:29418/OWR/IoTG/SMIE/Security/secure
-computing/meta-tep-manifests.git -b refs/heads/acrn_2.2 -g all
```

Pull meta-layers

```
   $ repo sync -c -j$(nproc) --force-sync
```

1.3.2 Build ACRN unified image with Trusted OS for target.

Once the cloning of the repositories is completed, use following instructions to build trusted OS image.

Set up Build Environment

```
$ sed -i 's/meta/meta-tep-acrn/g' ./openembedded-core/.templateconf
$ source ./openembedded-core/oe-init-build-env
$ mkdir -p conf/multiconfig/
$ cp -r ../openembedded-core/meta-tep-trusted-os/conf/multiconfig conf/
$ cp -r ../openembedded-core/meta-tep-acrn/conf/multiconfig conf/
```

Build the Image

```
$ bitbake mc:x86-tep-trusted-os-tgl-initramfs:core-image-trusted-os
```

Note: ***Following step is important if you don't want to use any changes from TEP into SOS and GuestVMs. Alternatively, you can use proper BBMASKS.***

```
$ bitbake-layers remove-layer meta-tep-trusted-os


$ bitbake mc:x86-tep-acrn-tgl:acrn-image-minimal
```

Final image will be located at:

Trusted-OS Initramfs Image:
build/tmp-x86-tep-trusted-os-tgl-initramfs-glibc/deploy/images/intel-corei7-64/bzImage-initramfs-intel-corei7-64.bin

ACRN unified image with trusted-os:

build/tmp-x86-tep-acrn-tgl/deploy/images/intel-corei7-64/acrn-image-minimal-intel-corei7-64.wic

1.4 Build SELinux based yocto host for container:

Below are the instructions to build a yocto based host with SELinux feature enabled as bare metal host for TEP docker container. One can also use other Linux based host for container (i.e Fedora, ubuntu, redhat, etc). we only validated TEP container on yocto based docker enabled Linux host. This yocto SELinux based build is for reference integration example, customer may change it as per their requirements. For container host build we are referring meta-selinux from yocto opensource layer in our sample integrations. We have TEP specific SELinux rules and changes in intel specific meta-tep-container layer. This host build system demonstrates SELinux container enablement in yocto system.

1.4.1 clone meta layers:

Make a new directory

```
$ mkdir -p tep-docker
$ cd tep-docker
```

Clone the repo manifest for TEP 2.2 release.

```
$ "repo init -u
ssh://git@gitlab.devtools.intel.com:29418/OWR/IoTG/SMIE/Security/secure-computing/meta-tep-manifests.git -b refs/heads/docker_2.2 -g all"
```

Pull the meta-layers

```
 $ repo sync -c -j$(nproc) --force-sync
```

1.4.2 Build SELinux yocto based bare metal host OS for docker containers:

Use meta-tep-container bblayers.conf.sample and local.conf.sample

```
$ sed -i 's/meta/meta-tep-container/g' ./openembedded-core/.templateconf

$ source ./openembedded-core/oe-init-build-env
```

Copy multiconfig files from meta-tep-container and meta-tep-trusted-os layer

```
$ mkdir -p conf/multiconfig/
$ cp -r ../openembedded-core/meta-tep-trusted-os/conf/multiconfig conf/
$ cp -r ../openembedded-core/meta-tep-container/conf/multiconfig conf/
```

Build the image with docker

```
$ bitbake mc:x86-tep-docker-selinux:core-image-selinux
```

SELinux enabled yocto image:

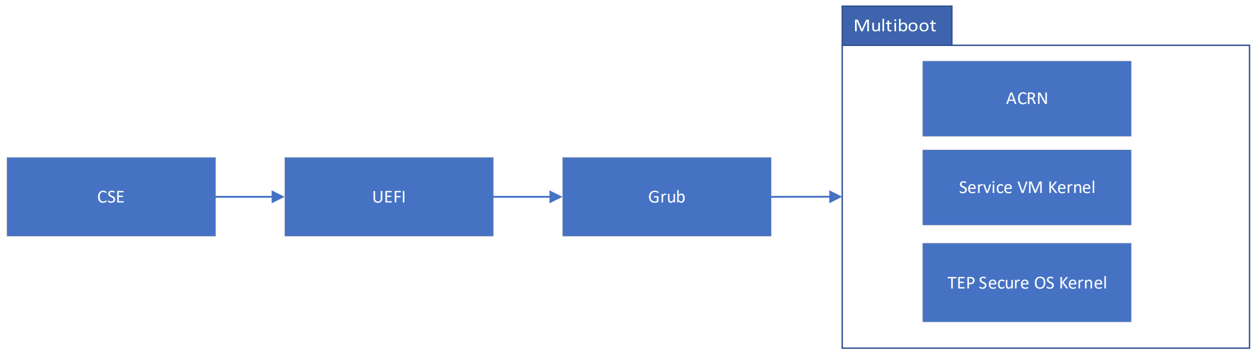
<tep-docker>/build/tmp-x86-tep-docker-selinux-glibc/deploy/images/intel-corei7-64/core-image-selinux-intel-corei7-64.wic

2 Platform Configuration

Platform with Trusted OS as recommends some configuration to achieve desired security goals. Some of these configurations are described here. One shall make sure that these changes are done at platform level to get system configurations right.

2.1 Secure Boot configuration

This section describes the secure booting of ACRN based Trusted Edge Platform (TEP) solution with UEFI FW. The method uses GRUB to securely boot the ACRN and TEP Secure OS. The flow diagram for secure boot have been mentioned below.



On booting the platform, UEFI verifies the GRUB.

GRUB verifies and launches the ACRN hypervisor, TEP Secure OS and Service VM.

We have followed the wiki https://projectacrn.github.io/latest/tutorials/acrn-secure-boot-with-grub.html to implement the secure boot.

2.1.1 Secure Boot Steps for booting the grub binary, acrn.bin and kernel images of pre-launch VM and Service OS VM

Follow the ACRN wiki link to securely boot the TEP on ACRN.

https://projectacrn.github.io/latest/tutorials/acrn-secure-boot-with-grub.html

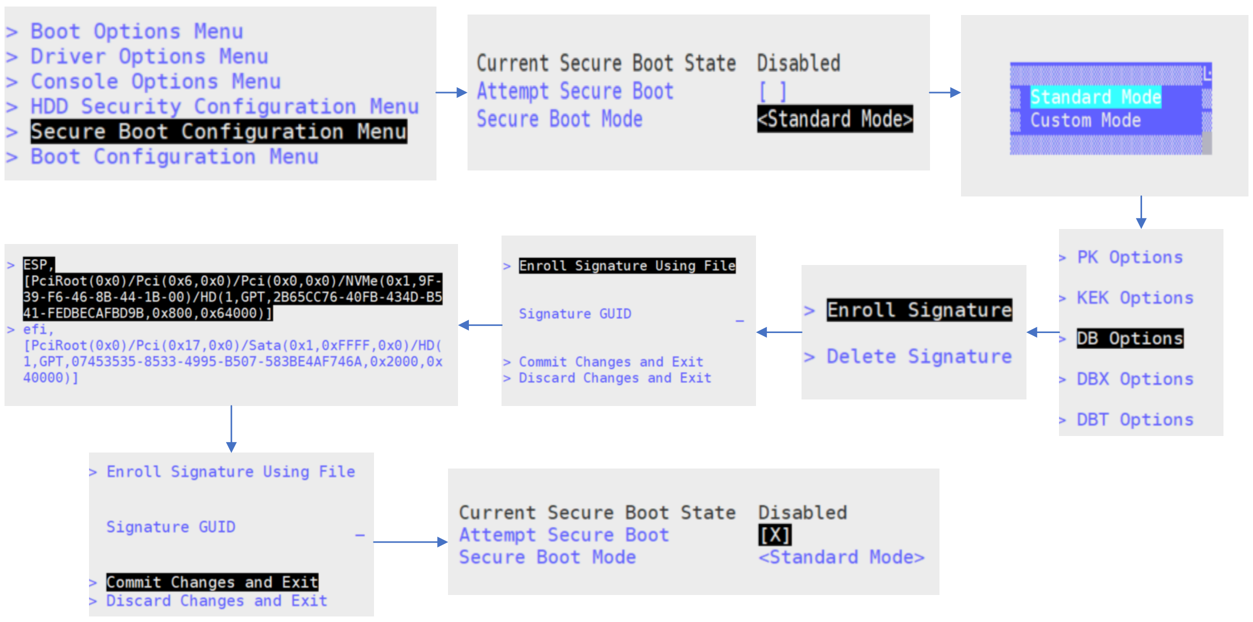
Along with following the above given wiki link one must perform few extra steps as given below:

1. Enabling grub authentication with password is optional.
2. Creating of grub.init.cfg is a must even if grub authentication is being enabled or not.
3. While creating standalone grub efi binary using the script provided in ACRN wiki, we must add an argument "**--disable-shim-lock**" in grub-mkstandalone functionality. This is important, as not including this argument will lead to error while booting with grub2.06.

***grub-mkstandalone ***
***--directory /usr/lib/grub/x86_64-efi ***
***--format x86_64-efi ***
***--disable-shim-lock ***
***--modules "$MODULES" ***
***--pubkey /mnt/ngs/boot.key ***
***--output ./bootx64.efi ***
***"boot/grub/grub.cfg=/boot/grub.init.cfg" ***
"boot/grub/grub.cfg.sig=/boot/grub.init.cfg.sig"

4. While signing the grub.cfg, acrn.bin, sos kernel bzImage (as mentioned in the ACRN wiki) one must also sign the following files:
    a. gpg --homedir keys --detach-sign path/to/grub.init.cfg
    b. gpg –homedir keys --detach-sign path/to/TEP Secure OS bzImage
    c. gpg –homedir keys --detach-sign path/to/ACPI_VM0.bin
5. Enabling Secure boot in UEFI bios:
   Note: Make sure that you have bootguard enabled BIOS in order to get HW root of trust and chain of trust extended from FW/HW to OS.
    a. Copy the **db.auth**(created as described in ACRN wiki) in a pendrive and connect it to the target board.
    b. Re-start the target board and enter the UEFI FW. Goto secure boot settings.
    c. Select Secure Boot Mode and select **Custom Mode**.
    d. Select **Custom Secure Boot Options** to enroll the db.auth key.
    e. Select **DB Options**. Further select Enroll Signature.
    f. Select **Enroll Signature Using file**. It lists the partitions.
    g. Select the partition which contains db.auth key and select the **db.auth** file.
    h. After selecting the **db.auth** file, select the setting Commit Changes and Exit.

    Below figure shows the flow of deploying the db.auth key in BIOS.



```
> Boot Options Menu
> Driver Options Menu
> Console Options Menu
> HDD Security Configuration Menu
> Secure Boot Configuration Menu
> Boot Configuration Menu
```

```
Current Secure Boot State   Disabled
Attempt Secure Boot         [ ]
Secure Boot Mode            <Standard Mode>
```

```
Standard Mode
Custom Mode
```

```
> PK Options
> KEK Options
> DB Options
> DBX Options
> DBT Options
```

```
> Enroll Signature
> Delete Signature
```

```
> Enroll Signature Using File

  Signature GUID                 _

> Commit Changes and Exit
> Discard Changes and Exit
```

```
> ESP,
  [PciRoot(0x0)/Pci(0x6,0x0)/Pci(0x0,0x0)/NVMe(0x1,9F-
  39-F6-46-8B-44-1B-00)/HD(1,GPT,2B65CC76-40FB-434D-B5
  41-FEDBECAFBD9B,0x800,0x64000)]
> efi,
  [PciRoot(0x0)/Pci(0x17,0x0)/Sata(0x1,0xFFFF,0x0)/HD(
  1,GPT,07453535-8533-4995-B507-583BE4AF746A,0x2000,0x
  40000)]
```

```
> Enroll Signature Using File

  Signature GUID                 _

> Commit Changes and Exit
> Discard Changes and Exit
```

```
Current Secure Boot State   Disabled
Attempt Secure Boot         [X]
Secure Boot Mode            <Standard Mode>
```

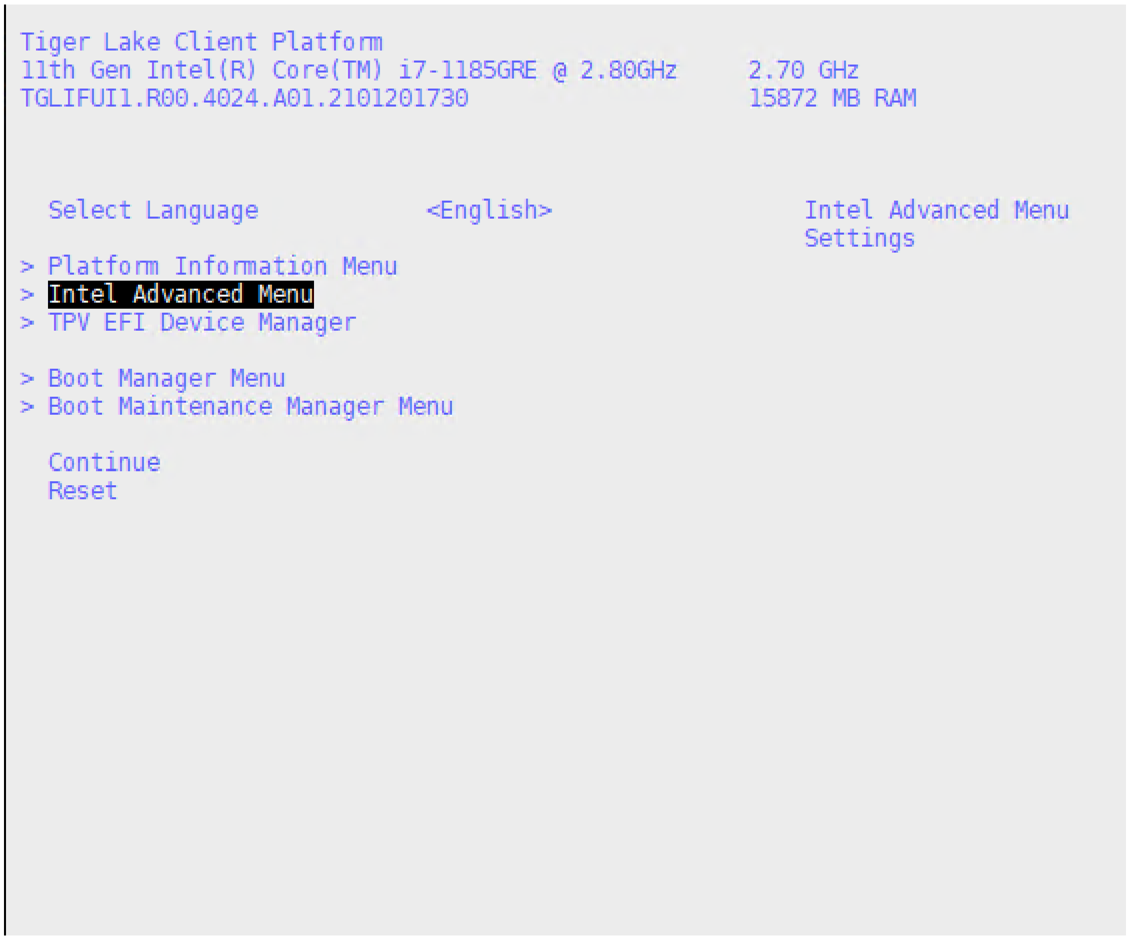
6. Enable the secure boot setting and restart the system. Select the drive on which ACRN image is flashed to securely boot.

2.2 TPM configuration

To use TPM as trusted execution environment in the platform which will act as hardware root of trust for trusted os user configurations, disk encryption and measured boot, platform shall have TPM enabled. following instructions are specific to Intel PTT enabled platform. These steps will help to check if TPM is enabled in TGL-U platform.

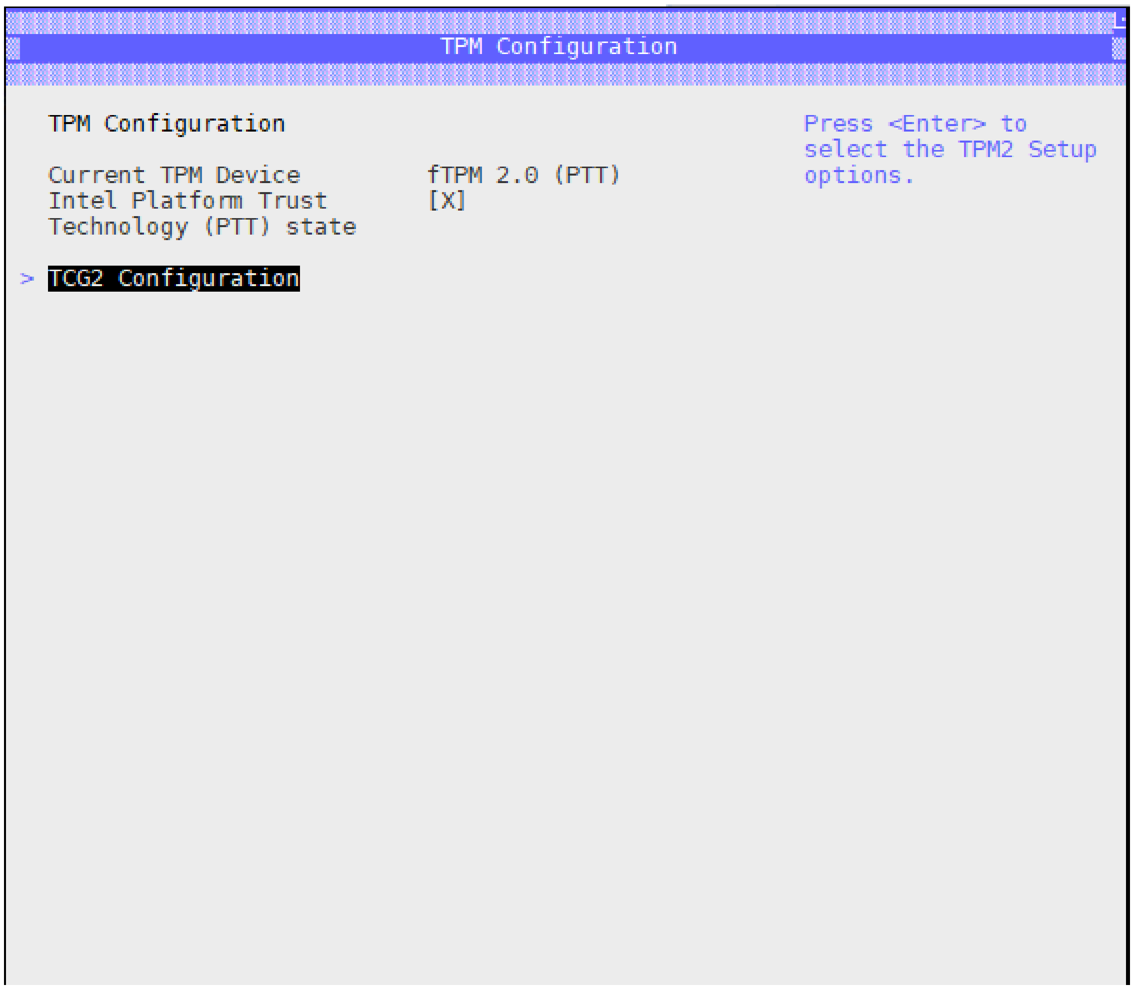
Note: make sure that we have PTT enabled bios/fw image. One can verifies it using following steps on TGL-U bios.

1. Go to bios menu->Intel Advanced Menu



```
Tiger Lake Client Platform
11th Gen Intel(R) Core(TM) i7-1185GRE @ 2.80GHz     2.70 GHz
TGLIFUI1.R00.4024.A01.2101201730                    15872 MB RAM



  Select Language           <English>              Intel Advanced Menu
                                                   Settings
> Platform Information Menu
> Intel Advanced Menu
> TPV EFI Device Manager

> Boot Manager Menu
> Boot Maintenance Manager Menu

  Continue
  Reset
```

2. TPM Configuration



```
                         TPM Configuration

  TPM Configuration                                         Press <Enter> to
                                                            select the TPM2 Setup
  Current TPM Device           fTPM 2.0 (PTT)               options.
  Intel Platform Trust         [X]
  Technology (PTT) state

> TCG2 Configuration
```

Make sure that PTT is enabled.

2.3 TME configuration

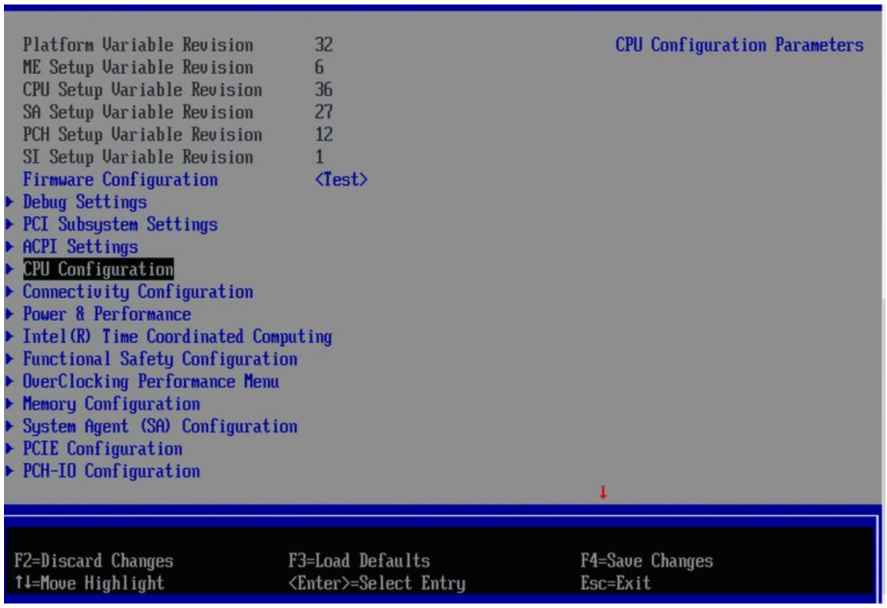
Total Memory Encryption (TME) is used to protect DRAM data from physical attacks. Such attacks include, moving DRAM module to another system, probing the DDR to read the cycles, etc. System memory is encrypted by the TME block attached to the memory controller. All cycles through TME block will be encrypted 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 an 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.

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

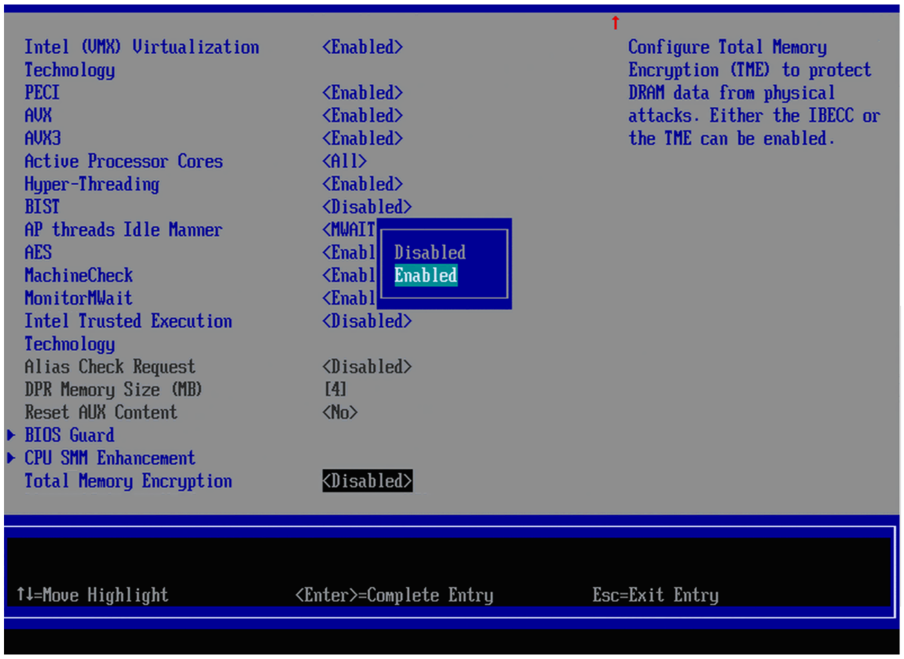
To enable the TME capabilities in system,

1. Go to bios menu->Intel Advanced Menu
2. Select “CPU Configuration”



```
Platform Variable Revision          32              CPU Configuration Parameters
ME Setup Variable Revision          6
CPU Setup Variable Revision         36
SA Setup Variable Revision          27
PCH Setup Variable Revision         12
SI Setup Variable Revision          1
Firmware Configuration              <Test>
▸ Debug Settings
▸ PCI Subsystem Settings
▸ ACPI Settings
▸ CPU Configuration
▸ Connectivity Configuration
▸ Power & Performance
▸ Intel(R) Time Coordinated Computing
▸ Functional Safety Configuration
▸ OverClocking Performance Menu
▸ Memory Configuration
▸ System Agent (SA) Configuration
▸ PCIE Configuration
▸ PCH-IO Configuration

F2=Discard Changes          F3=Load Defaults          F4=Save Changes
↑↓=Move Highlight           <Enter>=Select Entry      Esc=Exit
```

3. Select “Enabled” for Total Memory Encryption.



```
Intel (VMX) Virtualization          <Enabled>       Configure Total Memory
Technology                                          Encryption (TME) to protect
PECI                                <Enabled>       DRAM data from physical
AVX                                 <Enabled>       attacks. Either the IBECC or
AVX3                                <Enabled>       the TME can be enabled.
Active Processor Cores              <All>
Hyper-Threading                     <Enabled>
BIST                                <Disabled>
AP threads Idle Manner              <MWAIT
AES                                 <Enabl   Disabled
MachineCheck                        <Enabl   Enabled
MonitorMWait                        <Enabl
Intel Trusted Execution             <Disabled>
Technology
Alias Check Request                 <Disabled>
DPR Memory Size (MB)                [4]
Reset AUX Content                   <No>
▸ BIOS Guard
▸ CPU SMM Enhancement
Total Memory Encryption             <Disabled>

↑↓=Move Highlight           <Enter>=Complete Entry       Esc=Exit Entry
```

Note:

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

is enabled.

3 Device Provisioning

Trusted OS uses PTT as default TPM for root cryptographic keys and root of trust for OS secure operations. TPM device on platform need to be provisioned with AES-256 user key for confidentiality and ECC-384 public key for verification. These two keys shall be provisioned in secure environment before device get ready for trusted OS. Following are sample steps one can follow to provision these attributes into TPM.

Note: To support dTPM which may have lower strength cryptographic AES and ECC algorithms, customer shall change encryption tool and provision those keys to get it working. We done limited testing for Infineon dTPM with AES-128 and ECC-256 bits only.

Pre-Requisite:

- Admin should choose between dTPM or PTT according to requirement as below as supported by TPM.
  1. Discrete TPM – AES-128 user key for confidentiality and ECC-256 public key for verification.
  2. PTT – AES-256 user key for confidentiality and ECC-384 public key for verification.
- Admin machine requirement – Creation of signed PCR for LUKS needs TPM2 operation. Admin machine requires physical TPM or Virtual TPM for performing admin steps.

3.1 TPM Device Provisioning

Refer this sample script for Linux platform sample device provisioning script for Linux commands for following operations. These sample operations will provide device key creation and provisioning in TPM. Intel PTT is POR TPM device for our validation, but one can use compatible dTPM as well. Only limited dTPM testing was done.

1. Generate a sample ECC key. this step can be done on host machine.
   a. $: `openssl ecparam -genkey -name secp384r1 -out pcr_pol_signing_key_priv.pem`
   b. $: `openssl ec -in pcr_pol_signing_key_priv.pem -out pcr_pol_signing_key_pub.pem -pubout`
2. Generate a 256 bit 'test' AES key or use one from the host system which you are using for encryption in the above script. Make sure you change 'key' in the sample host script accordingly.
   a. $: `tpm2_getrandom -o tep_config_data_aes_key.bin 32`

   Note: If setting up LUKS for a TEP platform with **discrete TPM**, use a 128-bit AES key instead of 256 bits. Therefore, use '16' in above command.

   Note: you can use different random number generator as well. i.e. /dev/urandom

3. Use above keys (ECC public key and AES key) from step #1 and #2 and from command-line (i.e Serial port) on TEP machine use tep_device_provision_sample.py to perform device provisioning.
   a. `$: python3 tep_device_provision_sample.py -pol_pub_key pcr_pol_signing_key_pub.pem -enc_key tep_config_data_aes_key.bin`

Note : you may need to clear TPM before provisioning using command $: `tpm2_clear`
Note: you may need to 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. (Future TODO)

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

3.2 Trusted VM/Container provisioning for user config.

Once Device provision is performed one can proceed into Trusted VM provisioning steps. This shall be the first step required when system first time boots with trusted OS and ready for configuration. The pre-requisites for this step are to have device provisioned with user keys.

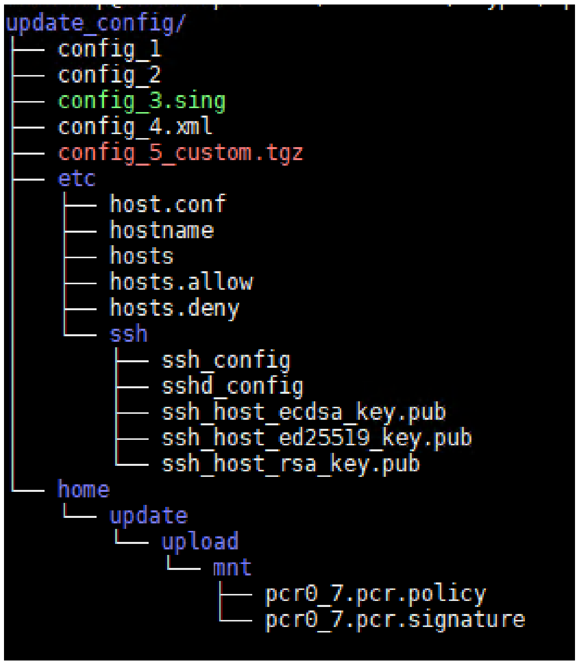
3.2.1 Creating user configuration signing and encryption.

Trusted VM will accept encrypted and signed user configuration data. Once data is transfer to trusted VM, 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. user config provisioning flow diagram show how the system works. Followings are steps to be followed to create encrypted and signed user config data and then verify it on system.

3.2.1.1 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 one example.



```
update_config/
├── config_1
├── config_2
├── config_3.sing
├── config_4.xml
├── config_5_custom.tgz
├── etc
│   ├── host.conf
│   ├── hostname
│   ├── hosts
│   ├── hosts.allow
│   ├── hosts.deny
│   └── ssh
│       ├── ssh_config
│       ├── sshd_config
│       ├── ssh_host_ecdsa_key.pub
│       ├── ssh_host_ed25519_key.pub
│       └── ssh_host_rsa_key.pub
└── home
    └── update
        └── upload
            └── mnt
                ├── pcr0_7.pcr.policy
                └── pcr0_7.pcr.signature
```

Figure 1: sample config files tree structure

2. Create a .tgz file of it. (this will reduce the size)
   a. `$: tar -cvzf update_config.tgz <update_config>`
3. Encrypt and signed this .tgz file with given sample host tool. (tep_encrypt_signed _user_config.py)
   a. `$: python3 tep_encrypt_signed _user_config.py <update_config.tgz> <ecc_key>  <aes_key>`

   Output - *tep_user_config_data.bin*

   Note: make sure that you use correct keys (ECC and AES) refer section 3.1.

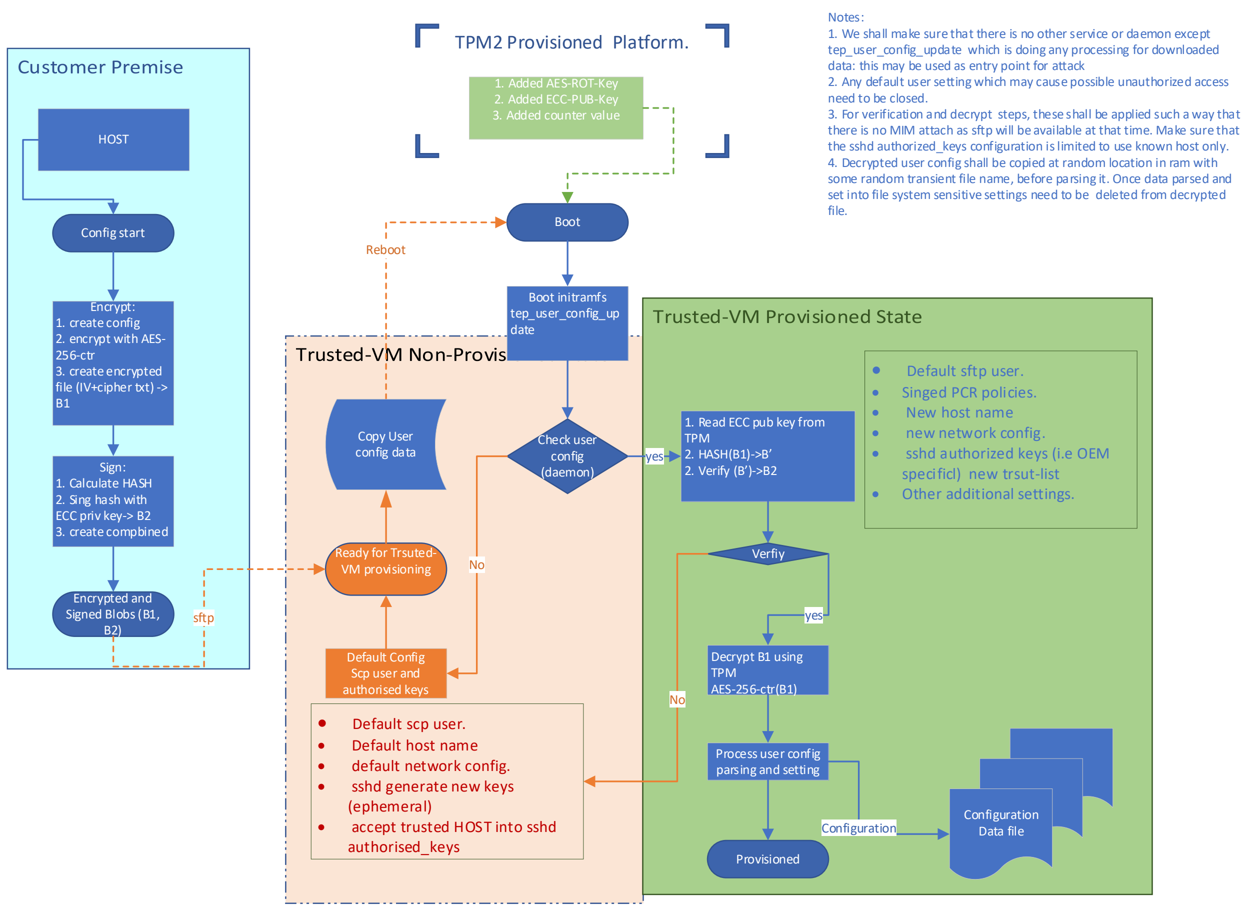
      i. ECC private keys which is associated with the public key provisioned in the device shall be used.

      ii. Same Aes-256-bit keys shall be used which is provisioned in the device.

   b. 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: change permission of file update_user_key_for_dev.pem as below - chmod 0444 update_user_key_for_dev.pem.



Customer Premise
HOST
Config start
Encrypt:
1. create config
2. encrypt with AES-256-ctr
3. create encrypted file (IV+cipher txt) -> B1
Sign:
1. Calculate HASH
2. Sing hash with ECC priv key-> B2
3. create compbined
Encrypted and Signed Blobs (B1, B2)
sftp
TPM2 Provisioned Platform.
1. Added AES-ROT-Key
2. Added ECC-PUB-Key
3. Added counter value
Boot
Reboot
Boot initramfs tep_user_config_update
Trusted-VM Non-Provis
Copy User config data
Check user config (daemon)
Ready for Trsuted-VM provisioning
No
Default Config Scp user and authorised keys
Default scp user.
Default host name
default network config.
sshd generate new keys (ephemeral)
accept trusted HOST into sshd authorised_keys
Trusted-VM Provisioned State
yes
1. Read ECC pub key from TPM
2. HASH(B1)->B'
2. Verify (B')->B2
Default sftp user.
Singed PCR policies.
New host name
new network config.
sshd authorized keys (i.e OEM specificl) new trsut-list
Other additional settings.
Verfiy
yes
No
Decrypt B1 using TPM AES-256-ctr(B1)
Process user config parsing and setting
Configuration
Configuration Data file
Provisioned

Notes:
1. We shall make sure that there is no other service or daemon except tep_user_config_update which is doing any processing for downloaded data: this may be used as entry point for attack
2. Any default user setting which may cause possible unauthorized access need to be closed.
3. For verification and decrypt steps, these shall be applied such a way that there is no MIM attach as sftp will be available at that time. Make sure that the sshd authorized_keys configuration is limited to use known host only.
4. Decrypted user config shall be copied at random location in ram with some random transient file name, before parsing it. Once data parsed and set into file system sensitive settings need to be deleted from decrypted file.

Figure 2: user config provisioning

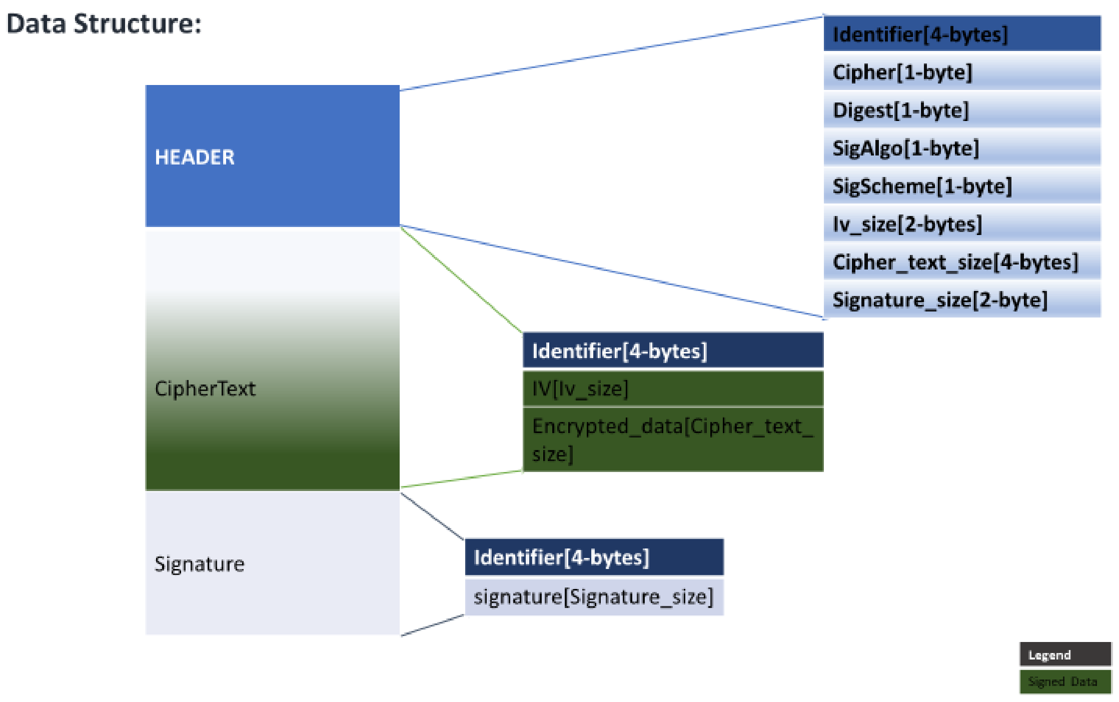


Figure 3: Signed Encrypted data format

3.2.1.2 Authentication and decryption of user config data at target machine have trusted OS.

Trusted OS have tep_user_config_update.service which have sample implementation to perform following operations 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.
    a. **Note:** the update of the respective config is end user dependent. We have sample implementation here at `verify decrypt service`

3.3 LUKS configuration.

1. Retrieve TEP platform PCR values (TEP machine):

   a. LUKS partition passphrase is sealed to TPM PCR's 0, 7 and 10. Value in these PCR is a function of BIOS, GRUB, VMM and trusted-OS code. Follow below steps to gather the PCR values:

   b. Open Serial port on the TEP platform :
      i. `Boot to ACRN shell.`
      ii. `vm_console 0`
      iii. `Userid : root , Password : 123456*18`

   c. ON TEP : Retrieve PCR values

      trusted-os: $: tpm2_pcrread -o pcr0_7_10.dat "sha256:0,7,10"

   Note : Instead of SHA256, if SHA384 PCR bank is enabled on TEP machine, use "sha384:0,7,10" in above command.

   d. ON TEP : Copy PCR values to Admin machine

      trusted-os: `scp pcr0_7_10.dat admin@admin-machine:/path/to/policy/location`

2. Generate TEP Luks Config Data file(**Admin** machine)

   a. Use create_luks_pcr_policy.py to generate various ingredients needed for LUKS passphrase setup.

      i. Authorized PCR Policy : This is a digest of PCR policy which contains info regarding the PCR signing public key and the PCR's included (i.e. 0, 7 and 10)

      ii. 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.

      iii. $: `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`

         Note: on above command type can be changed to “dtpm” instead of “ptt” for discrete TPM.

   b. Use create_luks_config.py to generate a YAML configuration file which will store all relevant luks configuration data.

      i. $: `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 incased enabled in grub.

   c. Copy luks_config_file.yaml (from above step) into update_config/home/update/upload/mnt directory.

   d. Follow section 3.2.1.1 step 3 for signed configuration.

Trusted-os ships with a few scripts in the filesystem to aid in the initial setup of the LUKS functionality. The following scripts exist in /opt/tep-luks/: tep_luks_module.py

a. 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.
b. ACRN systemd init service will invoke tep_luks_module.py to setup luks initialization and in terms of container based luks - entrypoint.sh will take care of running tep_luks_module.py to setup luks initialization.
c. tep_luks_module.py will read luks_config_file.yaml and encrypt partition first time.
d. In subsequent boots, systemd init(ACRN)or entrypoint.sh(Container) 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 mount one shall remove old sshd host keys and regenerate new keys in mounted partition as below. This only need 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
      a. /home/root/tep_luks_dev/ssh_host_rsa_key
      b. /home/root/tep_luks_dev/ssh_host_ecdsa_key
      c. /home/root/tep_luks_dev/ssh_host_ed25519_key
   2. `systemctl restart sshdgenkeys.service`

Following NVIndex used for the sealed luks passphrase:

luks_passphrase_handle=0x8100_A001

Debugging LUKS Failure –

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

3.4 TrustAgent configuration.

Platform Integrity in TEP is enabled by the implementation of the Chain of Trust and Remote Attestation. This use case is for foundational security. Attestation refers to the process of authenticating and attesting to the state of a remote platform and its operating system.

TEP OS uses the iSecL framework for remote attestation use cases.

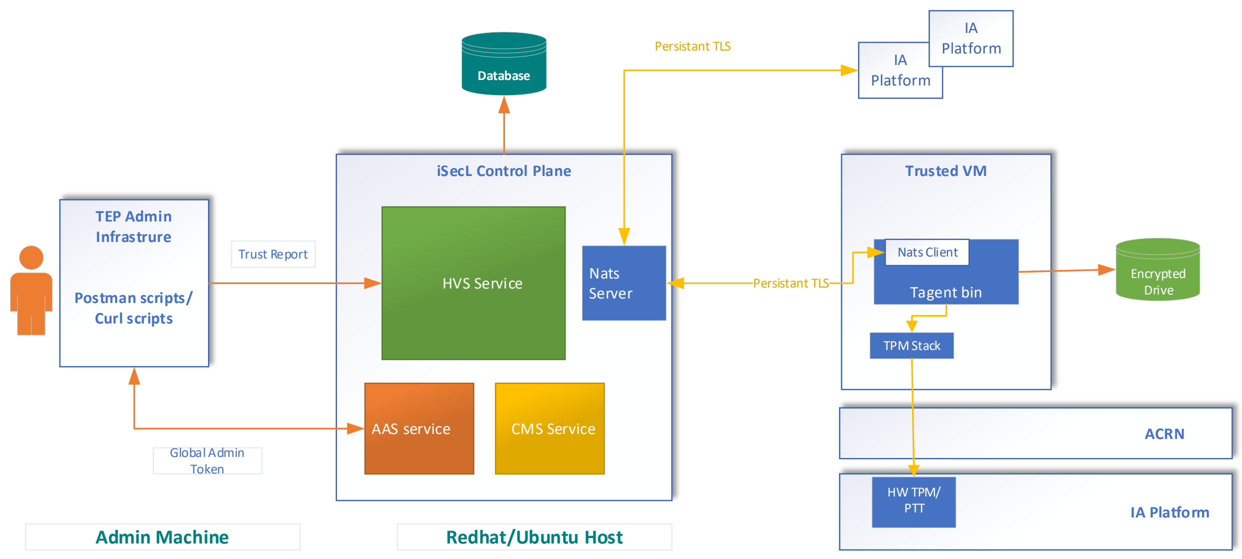
3.4.1 Attestation components

Attestation usecases requires the following to be setup,

- Attestation Server a.k.a. iSecL control plane
- TEP Admin infrastructure setup
- Trustagent component integrated with TEP OS.

Below is the picture for attestation components for trusted VM. Same components hold good for TEP container.

Attestation Usecase



3.4.2 TEP Trustagent

Trust Agent resides on TEP trusted VM/container 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

TrustAgent Setup configuration:

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

- Create an Trustagent answer file `trustagent.env`
- Follow section 4.2.6, for getting Bearer token details.

```
TA_TLS_CERT_CN=Trust Agent TLS Certificate
HVS_URL=https://<Ip address or hostname of HVS>:8443/hvs/v2
AAS_API_URL=https://<Ip address or hostname of AAS>:8444/aas/v1
CMS_BASE_URL=https://<Ip address or hostname of CMS>:8445/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>
TPM_OWNER_SECRET=<KEEP IT EMPTY>

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.
  - Using steps mentioned at section 3.2 update TEP user config blob and reboot the TEP trsutedVM/TEP Container. Create updated 'tep_user_config_data.bin'.
- Set system time which aligns to TEP attestation server and save using hwclock.
- Boot TGL platform to TEP OS
  - Init service will initially invoke tep_ta_config_update service which will verify and decrypt 'tep_user_config_data.bin' and update /home/update/upload/mnt folder with all the necessary files for trustagent.env execution.
  - Init service will invoke to setup luks initialization & decrypt the storage drive.
  - Init service will invoke tep_ta_setup.sh for trust agent provision and starting the tagent. Tagent provision is one time step.
    - Following would be created as part of trustagent provision.
      - /home/root/tep_luks_dev/trustagent - Stores the keys, certs and configuration files for trustagent
      - /home/root/tep_luks_dev/log/trustagent – Stores the trustagent logs.
      - Note: This location must be configured to Luks drive.
  - For subsequent boots, Trustagent service will check whether tagent is provisioned and it would start tagent for next boot.
- Trust agent status checks
  - Check TA status. "`tagent status`". Tagent should be active.

- Check NATS server connection is established with server
  - `netstat -t`

```
root@tep-trusted-os-tgl-initramfs-intel-corei7-64:/usr/bin/trustagent# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 10.34.130.178:57880 10.34.128.92:4222 ESTABLISHED
```

4 Attestation Admin Tasks

4.1 Intel-Secl Control Plane

iSecL control plane is the server 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

It recommended to follow iSecl github documentation & product guides for building their control plane. Here are the details about the intel-isecl.

Git repo: https://github.com/intel-secl/intel-secl/tree/v4.0.0

Documentation & Product user Guides:

https://github.com/intel-secl/docs/tree/v4.0/develop

Build Setup:

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

Deployment of iSecl control plane for foundational security can be done various methods,

- Bare metal method and
- Using Kubernetes

Here are the details of components required for deployment using bare metal method. Follow the intel-isecl product guide for creating answer files and deployment instructions.

Component	**Binary/Script**
Postgresql 11.0	Postgresql 11.0 – available for Redhat/Linux distributions create_db.sh – creates users for aas & hvs database. Available from build of control-plane server iseclpgdb.env – DB env file
CMS	cms-v4.0.0.bin - installer for Certificate management system. Available from build of control-plane server

	cms.env - Create answer file for CMS installation
AAS	authservice-v4.0.0.bin - Installer for Auth Service system. Available from build of control-plane server authservice.env - Create answer file for AAS installation
Populate users	populate-users.sh – Creates the users and tokens required for installation populate-users.env - Create answer file for creating required users
HVS	hvs-v4.0.0.bin – Installer for Verification system. Available from build of control-plane server hvs.env - Create answer file for HVS installation
Nats Server	nats-server – Install NATS distribution provided. download-tls-certs.sh - available form build of control-plane server at intel-secl/tools/

4.2 TEP Admin attestation infrastructure

TEP admin have the following tasks for setting up the attestation usecases.

- Get global admin token using the userid and password
- 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 groups required and post to HVS.
    - Boot a golden host with Trust agent provisioned. Import flavors from golden host.
- Host registration and generation of reports
    - Register hosts required. Use the host names TA_HOST_ID used in with provisioning Trustagent on TEP devices.
- Here 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 regsited
    - SAML Report
    - All Hosts - Provides status of all TEP devices connected.
- Creating trustagent.env required for TEP device trustagent provisioning.

4.2.1 Postman scripts

Intel-isecl provides post man scripts and API collection's for iSecL control plane to be used in postman environment. Postman collection would provide majority of the functionality. Admin user can customize or add these scripts as per the usecase requirements.

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

4.2.2 Admin Token

The Global Admin user account has all roles for all services. This is a default administrator account that can be used to perform any task, including creating any other users. In general, this account is useful for POC installations, but in production it should be used only to create user accounts with more restrictive roles. The administrator credentials should be protected and not shared.

Use below as body for postman scripts in getting the global admin token,

POST https://{{isecl-server}}:8444/aas/v1/token

```
{
    "username": <admin-user-id>,
    "password": <admin-password>
}
```

4.2.3 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,

a. Flavor templates
b. Flavor Groups
c. Flavor import

Wiki for flavor configuration - https://github.com/intel-secl/docs/blob/v4.0/develop/product-guides/Foundational%20&%20Workload%20Security.md#flavor-management

Flavor templates:

Flavor Templates are conditional rules that apply to a Flavor part cumulatively based on defined conditions. Here is the sample flavor templates.

Post the template mentioned below to https://{{isecl-server}}:8443/hvs/v2/flavor-templates and this would generate id. Save the id generated.

```
{
       "flavorgroup_names": null,
       "flavor_template": {
           "label": "default-tep",
           "condition": [
           "//host_info/os_name//*[text()='meta-intel-ese Reference Distro']",
           "//host_info/hardware_features/TPM/meta/tpm_version//*[text()='2.0']",
            "//host_info/hardware_features/UEFI/enabled//*[text()!='true'] or //host_info/hardware_feature
s/UEFI/meta/secure_boot_enabled//*[text()!='true']"
          ],
        "flavor_parts": {
            "PLATFORM": {
                "meta": {
                    "tpm_version": "2.0",
                    "vendor": "Linux"
                },
                "pcr_rules": [
                    {
                        "pcr": {
                            "index": 0,
                            "bank": [
                                "SHA384",
                                "SHA256"
                            ]
                        },
                        "pcr_matches": true
                    },
                    {
                        "pcr": {
                            "index": 7,
                            "bank": [
                                "SHA384",
                                "SHA256"
                            ]
                        },
                        "pcr_matches": true,
                        "eventlog_equals": {}
                    }
                ]
            },
            "OS": {
                "meta": {
                    "tpm_version": "2.0",
                    "vendor": "Linux"
                },
                "pcr_rules": [
                    {
                        "pcr": {
                            "index": 8,
                            "bank": [
                                "SHA384",
                                "SHA256"
                            ]
                        },
                        "pcr_matches": true
                    },
                    {
                        "pcr": {
                            "index": 9,
                            "bank": [
                                "SHA384",
                                "SHA256"
                            ]
                        },
                        "pcr_matches": true,
                        "eventlog_includes": [
                            "/acrn.bin",
                            "/bzImage-trustedVM",
                            "/bzImage",
                            "/ACPI_VM0.bin"
                        ]
                    }
                ]
            }
        }
    }
}
```

Flavor Groups:

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 Platform & OS flavor types.

POST https://{{isecl-server}}:8443/mtwilson/v2/flavorgroups

Here is the sample flavor group, use the id generated while posting flavor templates while creating flavor groups.

```
  {
           "name": "tep_2.2",
           "flavorTemplateIds": ["34cadbff-6c83-4f41-ad5b-97fe052397dc"],
           "flavor_match_policy_collection": {
               "flavor_match_policies": [
                   {
                       "flavor_part": "PLATFORM",
                       "match_policy": {
                           "match_type": "ANY_OF",
                           "required": "REQUIRED"
                       }
                   },
                    {
                        "flavor_part": "OS",
                        "match_policy": {
                            "match_type": "ANY_OF",
                            "required": "REQUIRED"
                        }
                    }
                ]
            }
        }
```

Import Flavors:

Flavor creation is the process of adding one or more sets of acceptable measurements to the Verification Service database. These measurements correspond to specific system components and are used as the basis of comparison to generate trust attestations.

POST https://{{isecl-server}}:8443/mtwilson/v2/flavors

```
{
"connection_string": "intel:nats://<TA_HOST_ID>",
"partial_flavor_types": ["PLATFORM", "OS"],
"flavorgroup_names" : [<FLAVOR_GROUP_NAME>]
}
```

4.2.4 Host Registration

Registration creates a host record with connectivity details and other host information in the Verification Service database. This host record will be used by the Verification Service to retrieve TPM attestation quotes from the Trust Agent to generate an attestation report.

POST https://{{isecl-server}}:8443/mtwilson/v2/hosts

```
{
   "host_name": "<TA_HOST_ID>",
   "connection_string": "intel:nats://<TA_HOST_ID>",
   "flavorgroup_names" : [<FLAVOR_GROUP_NAME>]
}
```

4.2.5 Report Creation

There are various reports can retrieve from HVS using the postman scripts.

- Attestation Reports
- Saml Reports
- Host State

Report generation details are captured in intel-secl product guide at https://github.com/intel-secl/docs/blob/master/product-guides/Foundational%20&%20Workload%20Security.md#attestation-reporting

4.2.6 Trustagent Env Creation

Trustagent.env is the environment file used when Trustagent is provisioned on TEP device.

Here are the details for fields in this file

Fields	**Description**
TA_TLS_CERT_CN	*Sets the value for Common Name in the TA TLS certificate. Defaults to "Trust Agent TLS Certificate".*
HVS_URL	*Host Verification service URL* *Ex: https://<Verification Service IP or Hostname>:8443/hvs/v2*
AAS_API_URL	*Auth Service URL* *https://<AAS IP or Hostname>:8444/aas/v1*
CMS_BASE_URL	*Certificate Management URL* *https://<CMS IP or Hostname>:8445/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	*sha384 of CMS TLS certificate. Generated with CMS installation of control plane.*
BEAERER_TOKEN	*Trust agent provision token. Generated using Custom claim token.*
TPM_OWNER_SECRET	*Empty. TEP OS do not use owner password.*
TA_SERVICE_MODE	*Outbound. This means a persistent connection will be established from TEP device to Nats Serve. HVS would communicate with TEP device through this channel.*
NATS_SERVERS	*Nats Server IP* *< nats-server-ip>:4222*
TA_HOST_ID	*Unique Host Id. Same Id have to be used when admin registers the TEP device with HVS.*

BEARER TOKEN:

This is a token to authorize the TEP device while performing the TrustAgent provisioning.

Intel-secl supports install admin token and custom claim token to do provisioning.

Its recommended to use Custom claim token for TA provision which would have only limited permissions for performing the provision steps as download-ca-cert, download privacy ca, EK & AIK provision.

Intel-secl product guide provides steps for “generating Custom Claim token using AAS API”.

Fields required while getting CC Token

- `CCC_ADMIN_USERNAME – Configured in populateuser.sh in control plane`
- `CCC_ADMIN_PASSWORD – Configured in populateuser.sh in control plane`
- `CUSTOM_CLAIMS_TOKEN_VALIDITY_SECS – Duration of token validity.`

5 Image Installations.

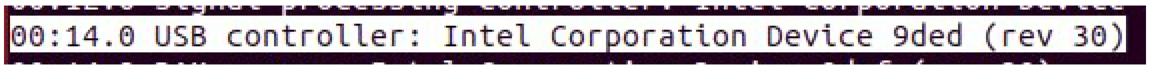
This section will give a brief about how to install ACRN hypervisor profile and bare-metal container images which are built by following build instructions in section 1 above.

4.1 Bring-up Trusted VM as pre-launch VM on Target (ACRN-hybrid configuration):

4.1.1 Dependencies:

a. Dedicated storage device for pre-launch OS. Only dedicated PCI based storage device can be used as pass through in ACRN.

b. For this release, USB storage is used (use the correct port for USB connection so pass-through works). This pass-through information needs to change in acrn build config. Following default USB PCI device on TGL-U rvp platform.



```
00:14.0 USB controller: Intel Corporation Device 9ded (rev 30)
```

c. The storage which will be dedicated to TEP pre-launched vm shall have three primary partitions.

   i. <Partition1> for PKCS11 use cases.

      1. Mounted at /home/root/tmp/

   ii. <Partition2> for update user's upload mount point.

      1. Mounted at /home/update/upload/mnt/

   iii. <Partition3> for LUKS encrypted partition.

      1. Mounted at /home/root/tep_luks_dev/

d. For development environment we are using serial console for controlling and launching of VMs. For production when serial is not enabled acrn could be set-up for auto-launch of trusted-os as pre-launch.

e. PTT/dTPM need to be enabled in BIOS. Make sure that you are using BIOS/FW with PTT enable.

f. TPM is used for cryptographic key store, make sure that user keys are stored in TPM as mentioned in device provisioning steps.

g. For Network access dedicated network interface need to be pass-through to pre-launch VM. Our default configuration uses USB based NIC on above mentioned USB passthrough root device.

4.1.2 Installation and setup of TEP prelaunch VM:

1. Flash the image "**acrn-image-minimal-intel-corei7-64.wic**" which is generated on build host onto target bootable media. NVME is default bootable. This can be performed using an image installer USB or dd from known alternative Linux os environment (preferably linux booting from USB media).
2. dd if=acrn-image-minimal-intel-corei7-64.wic of=/dev/<nvme>
3. Boot using the option "ACRN(Yocto) in grub menu.
4. On serial console ACRN hypervisor shell will come up.
5. Run vm_list in ACRN shell. VMs currently present will show.
6. Command "vm_console 0" takes you to pre-launch VM console. To comeback to ACRN shell press "Ctrl+Space".
7. To go to service OS console do "vm_console 1". Check on acrn shell as follow to bring up trusted VM console.

4.2 Bare metal host installation on target and configuring TEP docker container:

To install SELinux based yocto wic image you shall have following pre-requisites.

4.2.1 Pre-Requisites:

1. TGL-U board with following accessories.
   - nvme drive
   - alternative Linux OS for updating TEP related images, preferably installed on USB drive and attached to TGL board.
   - USB network card.

4.2.2 Installation and setup of TEP container:

Follow following instructions to bring-up SELinux based yocto host as bare metal OS for TEP docker containers.

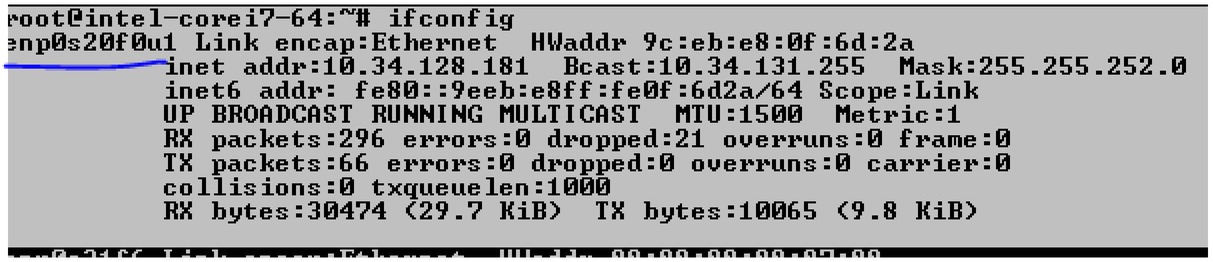
Installation:

- Boot alternative Linux OS from USB media.
- Copy **SELinux enabled yocto image** (core-image-selinux-intel-corei7-64.wic) from your build machine to a above mentioned USB booted OS or other media and attach media to TGL board.
- $dd if=<path of core-image-selinux-intel-corei7-64.wic > of= /dev/nvme0n1 status=progress
- Reboot and select NVME from boot device from UEFI.
- Do ssh setup for accessing TPM simultaneously from container and host.
  - Generate ssh key pair 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

TEP docker container setup:

Once bare metal host operating system is up and running after installation. We can install and setup TEP docker container as below.

1. Macvlan Network Creation:
   - Pre-requisites:
     - Get the subnet, gateway details, ethernet interface



```
root@intel-corei7-64:~# ifconfig
enp0s20f0u1 Link encap:Ethernet  HWaddr 9c:eb:e8:0f:6d:2a
          inet addr:10.34.128.181  Bcast:10.34.131.255  Mask:255.255.252.0
          inet6 addr: fe80::9eeb:e8ff:fe0f:6d2a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:296 errors:0 dropped:21 overruns:0 frame:0
          TX packets:66 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:30474 (29.7 KiB)  TX bytes:10065 (9.8 KiB)
```

   - Create an macvlan network for container network access
     - $ docker network create -d macvlan --subnet=10.34.130.0/24 --gateway=10.34.130.1 -o parent=enp0s20f0u1 my-macvlan-net
     - Check for macvlan with command

       $ docker network ls

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

2. Create an network file with interface to be used for Host macvlan bridge (**only first boot on SELinux host)**
    - For example, "echo enp0s20f0u1 >> /usr/bin/network_container.txt"
3. Run the docker setup for TEP container
    - $ docker import **core-image-trusted-os-intel-corei7-64.tar.bz2** trusted_container:latest
    - $ /usr/bin/docker_setup.sh 3
        - 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.
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.
        - $ docker commit trusted_container trusted_container:latest
        - $ docker stop trusted_container
6. Relaunch container
    - $ /usr/bin/docker_setup.sh 3
    - 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 should be started automatically.

6 API Interface and Sample applications

6.1 PKCS11 Client and Daemon

Trusted VM consist of standard concept of RPC Server and Client. In order to provide a homogenous application interface to ‘tpm2 pkcs11 module’ from guest VM to Trusted VM. Server-side listener application is called ‘pkcs11_server_daemon’ which will respond on the pkcs11 request from guest VMs. Client-side example application is compiled to ‘demo_pkcs11_app’. This demo application provides uses of tpm2 pkcs11 APIs.

Server-Side daemon ‘pkcs11_server_daemon’ is spawned by Systemd and shall link shared libs that encapsulate the RPC server implementation.

On Guest OS, user applications shall need to link with just one library ‘erpc_client_wrapper.so’ (part of deliverables), which shall expose the required pkcs11.h interface, in order to access PKCS11 APIs. This library implements the RPC client side internally.

In the current release multiple apps can talk to trusted VM. This can be done over multiple TTY ports (limited to max 2 at this point of time), and multiple PKCS11 apps can make use of same TTY port, they shall be in a waited Semaphore queue.

6.2 Features supported in this release

1. Provides tpm2_pkcs11 stack computing infrastructure integrated into the ECS stack in hypervisor where different VM’s are running.
2. Builds *core-image-sec-os* and launches in pre-launch mode
3. Builds a RPC interface library in guest VM.

4. Following object types are supported.
    a. RSA Support.
    b. AES Support.
    c. RNG Support.
    d. Object Management support.
5. Following PKCS11 APIs are now supported with this release.

S.No	PKCS11 API Name	S.No	PKCS11 API Name
1	C_Initialize	26	C_DecryptUpdate
2	C_Finalize	27	C_DecryptFinal
3	C_Getinfo	28	C_FindObjectsInit
4	C_InitToken	29	C_FindObjects
5	C_GetTokenInfo	30	C_FindObjectsFinal
6	C_GetSlotList	31	C_SignUpdate
7	C_GetMechanismInfo	32	C_SignFinal
8	C_OpenSession	33	C_VerifyUpdate
9	C_Login	34	C_VerifyFinal
10	C_InitPIN	35	C_GetFunctionList
11	C_Logout	36	C_DestroyObject
12	C_CloseSession	37	C_CreateObject
13	C_GenerateKeyPair	38	C_GetSessionInfo
14	C_GetAttributeValue	39	C_SetPIN
15	C_SignInit	40	C_CloseAllSessions
16	C_Sign	41	C_GetMechanismList

17	C_VerifyInit	42	C_DigestInit
18	C_Verify	43	C_Digest
19	C_GenerateRandom	44	C_DigestUpdate
20	C_EncryptInit	45	C_DigestFinal
21	C_Encrypt	46	C_SeedRandom
22	C_EncryptUpdate		End of list.
23	C_EncryptFinal		
24	C_DecryptInit		
25	C_Decrypt		

6. Establishes RPC like interface (over UART) communication TrustedVM and GuesVMs. The nature of VMs can be any, depends on ARCN configuration.
7. Supports multiple PKCS11 apps talking to TrustedVM
   a. 6 TTY ports ttyS4 to ttyS9 available.
   b. More than 1 app can access same TTY port.
8. For PKCS11 Objects:

   a) One pre-defined Token has been initialized in the TrustedVM at the time of boot. (This operation typically will be performed at platform manufacturing time. we are simulating it here with pre-defined token)
   b) Sample AES Key is provisioned into TPM by TrustedVM at boot time.

6.3 On Trusted VM side

The 'ipc-pkcs11.target' *systemd* service is launched at boot time. This service will initialize TPM stack and run server-daemon.

In summary there is no need to run anything on TrustedVM side, its ready to accept the PKCS11 calls from Guest OS app.

Optional step for sanity:

Following command could be used to check status of erpc-pkcs11 service at Sec-os console. This service launches 'pkcs11_server_daemon' on TrustedVM.

```
$ systemctl status ipc-pkcs11.target
```

Note: The default **root** password on Trusted OS is 123456*18

Recommendation: End user should change root password as per their requirement.

6.4 On Guest-OS side

On Guest-OS which could be pre or post launch VM. Make sure that you have client binaries and libraries compiled. A sample yocto recipe (ipc-p11-client.bb) is given at which used following code repository, this recipe could be included in IAMGE_INSTALL_append in yocto based user VM.

https://gitlab.devtools.intel.com/OWR/IoTG/SMIE/Security/secure-computing/tep-ipc-stack.git

for non-yocto and Linux based guest VMs, we could use auto-tool based compilation methods.

The source code for client is auto-tools based and could be compiled on any compatible Linux based system having auto tool support. Source code could be found from above shared location.

use following instructions to generate client side library and sample app

```
$ git clone
https://gitlab.devtools.intel.com/OWR/IoTG/SMIE/Security/secur
e-computing/tep-ipc-stack.git
$ git checkout <release commit/tag>
$ cd tep-ipc-stack
$ autogen.sh
$ ./configure --enable-client --enable-debug
$ make
```

Above commands will create:

1) libpkcs11_client_wrapper.so library
2) demo_pkcs11_app binary

In similar fashion more user applications can be built and run.

Refer *Readme*, *makefile.am* and *configure.ac* files for more details.

6.4.1 Running PKCS11 apps on Guest OS:

Note: There is currently an ACRN limitation for TTY ports.

1) A maximum of 6 UARTs can be enumerated per VM.

This can be configured all to one VM or split across other post launch VMs needing the TEP as a service.

2) These vUARTS ports in the ACRN 2.3 is exposed as a PCIe device now. Due to which **we need the following 2 KCONFIG** in the Post launch VM's Kernel.

```
CONFIG_SERIAL_8250_NR_UARTS=32
CONFIG_SERIAL_8250_RUNTIME_UARTS=32
```

6.4.1.1 Running the apps:

The client apps can make use of UART ports starting from ttyS4 to ttyS9 (total of 6) to talk to the Sec-OS. Each of these ports are serialized using a mutex internally, more than 1 process can use them. To choose the require select we need to run the following command:

Selecting the right TTY port.

```
export TEC_COMM_PORT=/dev/ttyS4 <choose S value from 4 onwards, till 9)
```

Run the demo app or your own app from same shell

```
tep_test_app
```

Additional Notes:

- Demo app make use of pre-defined AES objects to demo AES.
- Assumption is that the TrustedVM/Secure OS is launched before the UserVM/GuestOS that way PKCS11 as a service will be available for guest OS.

- In case the configuration of the ports is less than 6, then the vUARTs numbering is handled respectively, always starting from ttyS4
- Multiple apps can use same TTY, they shall be in a semaphore wait.
- Make sure that when we launch guest VM , we set vUART properly for IPC communication. Following sample example in setting uart.

```
acrn-dm -A -m $mem_size -s 0:0,hostbridge -s 1:0,lpc \
      -s 5,virtio-console,@stdio:stdio_port \
   -s 6,virtio-hyper_dmabuf \
      -s 3,virtio-blk,/var/lib/machines/images/vm0.wic \
      -s 4,virtio-net,$tap_name \
      -s 7,virtio-rnd \
      -s 10,uart,vuart_idx:1 \
      -s 11,uart,vuart_idx:2 \
      -s 12,uart,vuart_idx:3 \
      -s 13,uart,vuart_idx:4 \
      -s 14,uart,vuart_idx:5 \
      -s 15,uart,vuart_idx:6 \
      --mac_seed $mac_seed \
      -U ${UUID_POST_STANDARD[0]} \
      --ovmf /usr/share/acrn/bios/OVMF.fd \
      $vm_name
```

7 Intel Recommendations

Followings are intel recommendations for system security.

- Change ‘root’ and ‘update’ user’s password in your yocto build recipe.
    - root_user
    - update-user

- Keep the Trusted OS, Grub and BIOS stacks up to date with patches
- In production system:
    - close all debug interfaces, including JTAG and Serial connections.
    - ACRN VMM dump should be disabled.
    - BIOS Menu lock down with password.
    - Out-Of-band provisioning of UEFI keys should be disabled.
- Recommendation to use MAC system on Service VM to protect the User PIN for PKCS
- 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"
- We recommend use AES-CTR-256, and ECC-384 crypto algorithms for better resistant for near future.
- To protect against an adversary with physical access, the system needs to support TME, VxD with the encrypted disk.
- For TEP docker container it is highly recommended to use SELinux and container-selinux module to provide proper protections.

8 Open Limitations

- VM to VM communication is done in the ACRN configuration XML files. Please refer ACRN configuration webpage for more details at this link.
- PTT is used as default TPM device, only Infineon dTPM was tested.
- While using AES from PTT for encrypt/decrypt, the maximum size of buffer it takes is 64K. if you have large file you shall use it in chunks of 64K.
- Following PKCS11 APIs are not supported by TrustedVM.

S.No	PKCS11 API Name	S.No	PKCS11 API Name
1	C_WaitForSlotEvent	12	C_DigestEncryptUpdate
2	C_GetOperationState	13	C_DecryptDigestUpdate
3	C_SetOperationState	14	C_SignEncryptUpdate
4	C_CopyObject	14	C_DecryptVerifyUpdate
5	C_GetObjectSize	16	C_GenerateKey
6	C_SetAttributeValue	17	C_WrapKey
7	C_DigestKey	18	C_UnwrapKey
8	C_SignRecoverInit	19	C_DeriveKey
9	C_SignRecover	20	C_GetFunctionStatus
10	C_VerifyRecoverInit	21	C_CancelFunction
11	C_VerifyRecover	22	C_GetSlotInfo