Quick Start Guide

Network and Edge Reference System Architectures – 5G vRAN Security

5G vRAN security software based on 4th Gen Intel® Xeon® Scalable processor platform.

Authors

Abhijit Sinha

Alex Du

John Logan

Introduction

The Reference System Architectures (Reference System[1]) are forward-looking Kubernetes*-cluster cloud-native reference platforms aiming to ease the development and deployment of network and edge solutions.

This guide enables 5G vRAN security software using the Access Profile of the Container Bare Metal Reference System Architecture (BMRA) on 4th Gen Intel® Xeon® Scalable processors on a Kubernetes (K8s) cluster.

Some of the key highlights of the implementation are mentioned below:

- **Service Management Orchestrator (SMO) and vRAN communication via NETCONF:** A client/server implementation abstracting communication between SMO and either the Centralized Unit (CU), Distributed Unit (DU), or RAN Intelligent Controller (RIC) secured by Intel's Key Management Reference Application (KMRA)
- **NETCONF security implementation:** RSA/ECDSA (SHA384 length) keys are secured using KMRA
- **Secured container certification:** Cosign/sigstore Kubernetes features configuration for container certification
- **Secure UEFI boot:** Steps to enable secure boot chain using UEFI Secure Boot (Appendix A)

Architecture and Setup

Figure 1 shows the architecture diagram of the Access Profile used for vRAN deployment. The profile enables the Intel® Software Guard Extensions (Intel® SGX), KMRA, and SR-IOV network plugins, and Intel® oneAPI libraries on a real-time OS for deploying 5G vRAN security use case.

[1] In this document, "Reference System" refers to the Network and Edge Reference System Architecture.

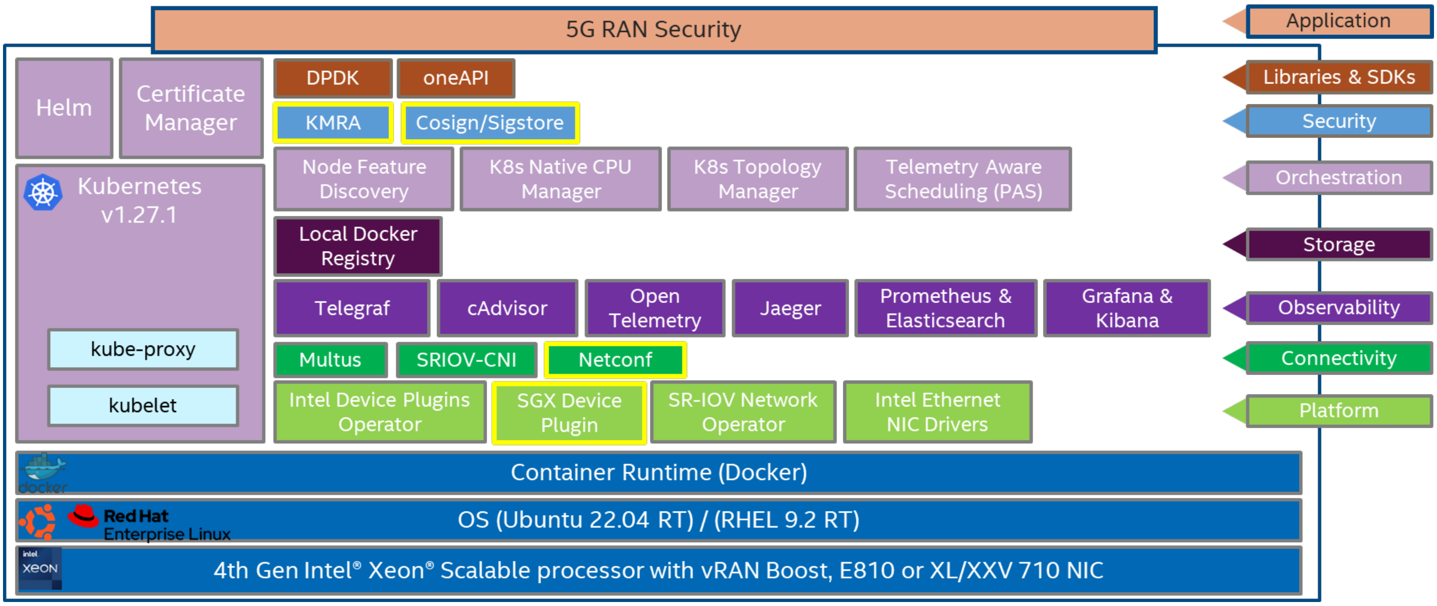


Figure 1: BMRA Access Configuration Profile Architecture for 5G vRAN Security

Hardware BOM

Following is the list of the hardware components that are required for 5G vRAN setup:

Ansible Host	Laptop or server running a UNIX base distribution with an internet connection
Controller Node	(Optional) Any 3rd or 4th Gen Intel® Xeon® Scalable processor-based server
Worker Node	1x 4th Gen Intel® Xeon® Scalable processor on Intel® Software Development Platform (Intel® SDP) S2EG4SEQ5Q OR 1x 4th Gen Intel® Xeon® Scalable processor with Intel® vRAN Boost on Intel SDP S2EG4SEQ5Q
FEC Accelerator	(Optional for this use case) Intel® vRAN Accelerator ACC100 Adapter on the target BBU server Note: The above is not required for 4th Gen Intel® Xeon® Scalable processor with Intel® vRAN Boost
Ethernet Adapter	Intel® Ethernet Network Adapter E810-CQDA2 or Intel® Ethernet Controller XXV/XL710 on the worker node
Recommended BIOS	Low Latency BIOS configuration is recommended with Intel® SGX turned on (refer to Chapter 3.8 of BMRA User Guide)

Software BOM

Following is the list of the software components for 5G vRAN deployment, including security:

Security	OpenSSL, Intel SGX, KMRA, Cosign/Sigstore container certification
Observability	Telegraf, Open Telemetry, Prometheus, Jaeger, cAdvisor, Grafana, Kibana
Acceleration/ Data Plane Library	DPDK, oneAPI
Connectivity	Multus, SR-IOV CNI, NETCONF

Operators and Device Plugins	SR-IOV device plugin, Intel SGX device plugin
Ethernet Drivers	i40e, ice, iavf
Container Runtime	Containerd
Orchestration	K8s v1.27.1, Node Feature Discovery, CPU Manager
OS	Ubuntu 22.04 LTS with real-time (kernel: 5.15.0.1036-realtime)* RHEL 9.2 with real-time (kernel: 5.14.0-284.11.1.rt14.296.el9_2.x86_64) *When using SGX functionality with Ubuntu realtime kernel, update kernel to version 5.15.0-1045.50 or later.

For details about the software versions for the **Access Edge** Configuration Profile, refer to Chapter 4 of the BMRA User Guide listed in the Reference Documentation section.

Getting Started

Prerequisites

Before starting the deployment, verify the following steps:

- A fresh OS installation is expected on the controller and target nodes to avoid a conflict between the RA deployment process with the existing software packages. To deploy RA on the existing OS, ensure that there are no prior Docker or Kubernetes* (K8s) installations on the servers.
- The hostname must be in lowercase, numerals, and hyphen' - ' format only for the target server.
  - For example: wrk-8 is acceptable; wrk_8, WRK8, Wrk^8 are not accepted as hostnames.
- The servers in the cluster are Network Time Protocol (NTP) synced, i.e., they must have the same date and time.
- The BIOS on the target server is set as per the recommended settings and Intel® SGX is enabled.

Deployment Setup

Figure 2 shows the deployment model for 5G vRAN security using BMRA. The Ansible host is only used for configuring and deploying BMRA on a set of target servers and is not a part of the final deployment cluster.

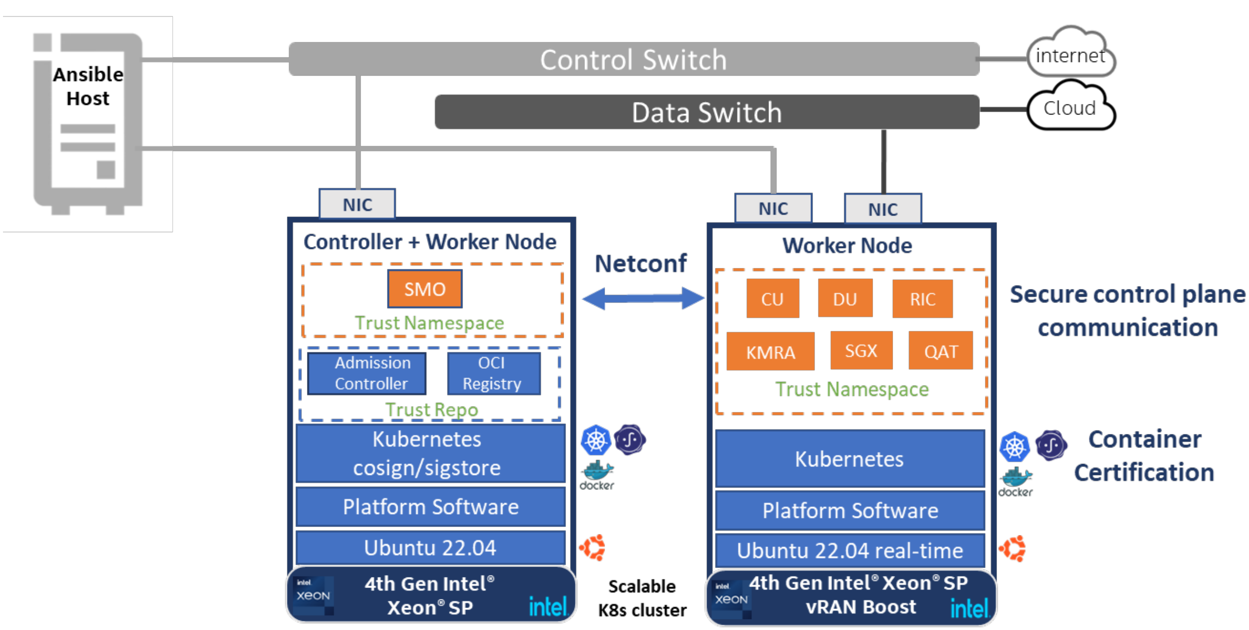


Figure 2: BMRA deployment set-up for 5G vRAN Security

Note: The software can also be deployed using a single node deployment (SNO) where the controller and worker nodes are in the same 4th Gen Intel® Xeon® Scalable processor server. **This document demonstrates the single-node cluster implementation.**

Installation Flow for RA Deployment

Ansible playbooks are used to deploy the FlexRAN™ software and the necessary software packages using the Access Profile. Before the playbooks can be run, there are a few steps to prepare the environment and change relevant configuration options.

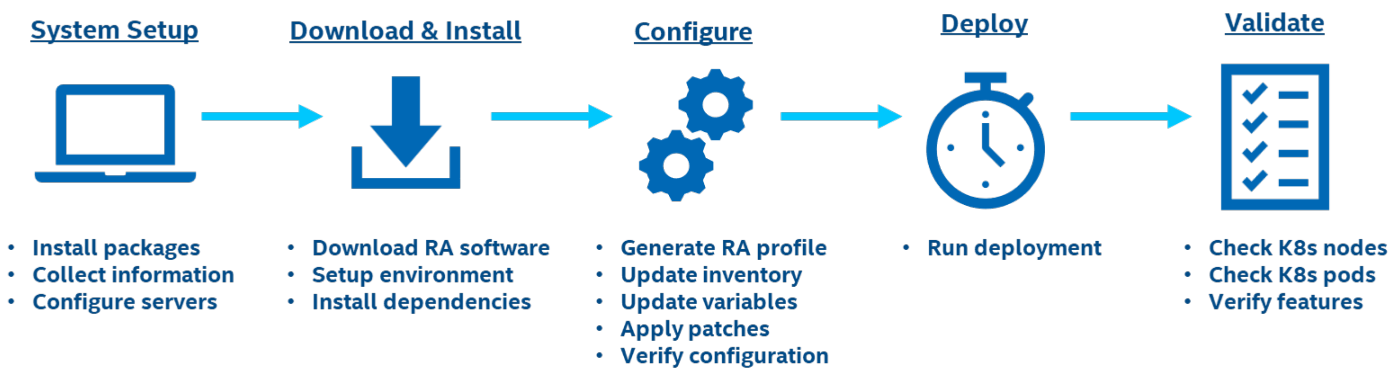


Figure 3: RA Deployment using Ansible Playbooks

Step 1 – Set Up the System

The following steps assume that both the Ansible host and target server are running Ubuntu as the operating system. For RHEL, use 'yum' or 'dnf' as the package manager instead of 'apt'.

Ansible Host

1. Install necessary packages (some might already be installed):

```
# sudo apt update
# sudo apt install -y python3 python3-pip openssh-client git build-essential
# pip3 install -upgrade pip
```

2. Generate an SSH keypair if needed (check /root/.ssh/):

```
# ssh-keygen -t rsa -b 4096 -N „“ -f ~/.ssh/id_rsa
```

3. Copy the public key to the target server(s):

```
# ssh-copy-id root@<target IP>
```

4. Verify password-less connectivity to the target server(s):

```
# ssh root@<target IP>
```

Target Server(s)

The following steps are required for all the target nodes:

1. (Optional) Install Ubuntu 22.04 with Real-Time (RT) kernel. You can follow the steps here as a reference.
2. (Optional) Verify that the kernel is tagged as a real-time kernel:

```
# uname -ri
5.15.0-1036-realtime x86_64
```

Note: Steps 1 and 2 are optional as we are not deploying the FlexRAN™ software application on the worker node in this release.

3. Install necessary packages (some might already be installed):

```
# sudo apt install -y python3 openssh-server lshw
```

4. As part of the configuration in Step 3, information about PCI devices for SR-IOV must be specified. Find the relevant Network PCI IDs (bus:device.function) using 'lspci' and note down the IDs for later when configuring host_vars on the Ansible host:

```
# lspci | grep Eth
```

```
18:00.0 Ethernet controller: Intel Corporation Ethernet Controller E810-C for QSFP (rev 01)
18:00.1 Ethernet controller: Intel Corporation Ethernet Controller E810-C for QSFP (rev 01)
```

5. Verify if the Intel QAT device is visible in the OS on the target server:

```
QAT Devices
# lspci -nnD | grep 494*
0000:76:00.0 Co-processor [0b40]: Intel Corporation Device [8086:4942] (rev 40)
0000:7a:00.0 Co-processor [0b40]: Intel Corporation Device [8086:4942] (rev 40)
```

6. Verify if Intel SGX is enabled on the target server:

```
sudo dmesg | grep -i sgx
[    4.554704] sgx: EPC section 0x1070180000-0x107f3fefff
[    4.555636] sgx: EPC section 0x2070180000-0x207fffffff
```

7. (Optional) Find the FEC accelerator card's PCI IDs (domain:bus:device.function) using 'lspci' and confirm that the device ID is '0d5c' and note it down for later when configuring host_vars on the Ansible host:

```
# lspci -D | grep -i acc
0000:31:00.0 Processing accelerators: Intel Corporation Device 0d5c
```

Step 2 - Download and Install

Ansible Host

Download & Install

1. Download the source code from the GitHub repository for the Reference System server:

```
# git clone https://github.com/intel/container-experience-kits/
# cd container-experience-kits
# git checkout v23.10
```

2. Set up Python* virtual environment and install dependencies:

```
# python3 -m venv venv
# source venv/bin/activate
# pip3 install -r requirements.txt
```

3. Install Ansible dependencies for the Reference System:

```
ansible-galaxy install -r collections/requirements.yml
```

Step 3 – Configure

The **Access Edge** configuration profile is used for **5G vRAN Security** software deployment.

Configure

Configuring BMRA for 5G vRAN Software

Ansible Host

1. Generate the configuration files.

```
# export PROFILE=access
# make k8s-profile PROFILE=${PROFILE} ARCH=spr
```

2. Update the *inventory.ini* file to match the target server's hostname. The values for `<target hostname>` and `<target IP>` must be updated to match the target systems in the BMRA cluster.

```
# vim inventory.ini

[all]
<target hostname>   ansible_host=<target IP> ip=<target IP> ansible_user=root
localhost           ansible_connection=local ansible_python_interpreter=/usr/bin/python3

[vm_host]

[kube_control_plane]
<target hostname>

[etcd]
<target hostname>

[kube_node]
<target hostname>
```

```
[oru]
<target hostname>

[k8s_cluster:children]
kube_control_plane
kube_node

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

3. Update the host_vars filename with the target machine's hostname.

```
# cp host_vars/node1.yml host_vars/<target hostname>.yml
```

To utilize features depending on SR-IOV, host_vars must be updated with information about the PCI devices on the target server. The example below can be used as a reference for the configuration but should be updated to match the correct PCI IDs of the target server.

4. Update *host_vars/<target hostname>.yml* with PCI device information specific to the target server.

```
# vim host_vars/<target hostname>.yml
#dataplane_interfaces:[]
dataplane_interfaces:
  - bus_info: "18:00.0"
    pf_driver: "iavf"
    default_vf_driver: "vfio-pci"
    sriov_numvfs: 4
```

5. Update *host_vars/<target hostname>.yml* to enable Intel® SGX to the target server.

```
# vim host_vars/<target hostname>.yml
configure_sgx: true # Intel Software Guard Extensions (SGX)
```

6. Make sure the following parameters are set correctly in *group_vars/all.yml.*

```
# vim group_vars/all.yml

#kube_proxy_nodeport_addresses_cidr: 127.0.0.0/8  #(comment the line to expose registry
service)
```

7. Enable the Intel® SGX device plugin in *group_vars/all.yml*.

```
# vim group_vars/all.yml
sgx_dp_enabled: true
```

8. Disable the FlexRAN™ POD and FEC operator in *group_vars/all.yml* for the vRAN security use case as we will not deploy the FlexRAN™ software container for this use case.

```
# vim group_vars/all.yml
intel_flexran_enabled: false
intel_sriov_fec_operator_enabled: false
```

9. Enable the Key Management Reference Application (KMRA) and vRAN container and disable ctk_loadkey_demo in *group_vars/all.yml.* The KMRA, which is set as false by default, can be enabled by setting the pccs, apphsm, and ctk_loadkey containers to true. To use KMRA, an API key must be requested from https://api.portal.trustedservices.intel.com/provisioning-certification (click on "Subscribe").

```
# vim group_vars/all.yml
kmra:
  sbx: false                # Enable pre-PRQ SGX platform
  oran:
    enabled: true           # Put KMRA into ORAN mode
    local_build: true       # Build oran container by default
  oran_netopeer2_server:
    enabled: true           # Enable netopeer2 server
  oran_netopeer2_client:
    enabled: true           # Enable netopeer2 client
  pccs:
    enabled: true           # Enable PCCS application
    api_key: "ffffffffffffffffffffffffffffffff" # Replace with your PCCS API key
  apphsm:
    enabled: true           # Enable AppHSM application
  ctk_loadkey_demo:
```

```
    enabled: false         # Enable CTK demo application
```

10. Enable the sigstore policy controller to enforce cosign container image security in the *group_vars/all.yml.*

```
# vim group_vars/all.yml
sigstore_policy_controller_install: true
```

11. Prepare the Docker* registry address by updating the target servers IP in the *group_vars/all.yml*

```
# vim group_vars/all.yml
registry_local_address: "<target IP>:{{ registry_nodeport }}"
```

12. If the server is behind a proxy, update *group_vars/all.yml* by updating and uncommenting the lines for `http_proxy`, `https_proxy`, and `additional_no_proxy`.

```
# vim group_vars/all.yml
## Proxy configuration ##
http_proxy: "http://proxy.example.com:port"
https_proxy: "http://proxy.example.com:port"
additional_no_proxy: ".example.com,mirror_ip"
```

13. (Required) Apply required patches for Kubespray:

```
# ansible-playbook -i inventory.ini playbooks/k8s/patch_kubespray.yml
```

14. (Optional) It is recommended that you check dependencies of components enabled in group_vars and host_vars with the package dependency checker:

```
# ansible-playbook -i inventory.ini playbooks/preflight.yml
```

15. (Optional) Verify that Ansible can connect to the target server, by running the below command and checking the output generated in the **all_system_facts.txt** file:

```
# ansible -i inventory.ini -m setup all > all_system_facts.txt
```

Step 4 – Deploy

Ansible Host

Now the Reference System can be deployed by using the following command:

```
# ansible-playbook -i inventory.ini playbooks/access.yml
```

Deploy

(Optional) If the playbook fails or if you want to clean up the environment to run a new deployment, you can optionally use the provided Cluster Removal Playbook to remove any previously installed Kubernetes and related plugins.

```
# ansible-playbook -i inventory.ini playbooks/redeploy_cleanup.yml
```

Step 5 – Validate

Ansible Host

To interact with the Kubernetes CLI (kubectl), start by connecting to the target node in the cluster, which can be done using the following command:

```
# ssh root@<target ip>
```

Validate

Once connected, the status of the Kubernetes cluster can be checked:

```
# kubectl get nodes -o wide
# kubectl get pods -A
```

Target Server

a) Validate Secure NETCONF communication

1. The below commands can be used to check the pod names in cosign namespace:

```
# kubectl get pods -n my-cosign-namespace
```

2. You can verify the logs in the kmra-oran-netopeer2-server and kmra-oran-netopeer2-client to validate the secure NETCONF communication:

```
# kubectl logs -f <kmra-oran-netopeer2-server_POD> -c kmra-oran-netopeer2-server-oran -n
my-cosign-namespace
```

```
# kubectl logs -f <kmra-oran-netopeer2-client_POD> -c kmra-oran-netopeer2-client-oran -n
my-cosign-namespace
```

Note: The user needs to use the specific kmra-oran-netopeer2 server and client POD names based on the deployment.

b) Manually run NETCONF command on Netopeer2 client

1. On the target server, run the below commands to enter the netopeer2-client pod:

```
# kubectl get pod -A | grep netopeer2-client

# kubectl exec -it <kmra-oran-netopeer2-client_POD> -c kmra-oran-netopeer2-client-oran -n
my-cosign-namespace bash
```

2. On a different terminal on the target server, run the following commands to see logs on netopeer2-server:

```
# kubectl get pod -A | grep netopeer2-server

#kubectl logs -f <kmra-oran-netopeer2-server_POD> -c kmra-oran-netopeer2-server-oran -n my-
cosign-namespace
```

3. Go back to the netopeer2-client terminal and run the test:

```
# export MODULE=/usr/local/lib/libpkcs11-proxy.so

# netopeer2-cli <<EOF
cert add /tmp/ca.pem
connect --tls --host localhost --port 6513 --cert /tmp/oran_cert.pem --key
pkcs11:token=token_client;object=client_key_priv;pin-value=1234
get-config --source running -o /tmp/config.xml
EOF
```

You can monitor the logs on the netopeer2-server side after the above command is executed.

c) Monitor Netopeer2 client POD traffic using tcpdump

1. On the target server find the netopeer2-server pod IP address:

```
# kubectl describe pod kmra-oran-netopeer2-server -n my-cosign-namespace | grep -i podIP
                  cni.projectcalico.org/podIP: 10.244.229.210/32
                  cni.projectcalico.org/podIPs: 10.244.229.210/32

# netstat -r | grep -i 10.244.229.210
10-244-229-210. 0.0.0.0         255.255.255.255 UH        0 0          0 cali8c4659c7205
```

2. Run the following command to start monitoring the netopeer2-server pod traffic:

```
# tcpdump -i cali8c4659c7205

Note: Replace the calico interface with the netopeer2-server pod interface
```

3. Go back to the netopeer2-client terminal and run the test:

```
# export MODULE=/usr/local/lib/libpkcs11-proxy.so

# netopeer2-cli <<EOF
cert add /tmp/ca.pem
connect --tls --host localhost --port 6513 --cert /tmp/oran_cert.pem --key
pkcs11:token=token_client;object=client_key_priv;pin-value=1234
get-config --source running -o /tmp/config.xml
EOF
```

d) Verify the O-RAN key type and length

1. Obtain the key certificate from O-RAN sysrepo configmap:

```
kubectl get configmap kmra-oran-netopeer2-server-cu-oran-sysrepo-config -n my-cosign-
namespace -o yaml
```

You will get a key certificate like this:

```
# cat ca.crt

<name>cacerts</name>
            <certificate>
                <name>cacert</name>
```

```
<cert>MIIB+zCCAYCgAwIBAgIUfpr1mbSNp0uckfjSyuOTa3wYW68wCgYIKoZIzj0EAwMwNDEPMA0GA1UECgwGQXBwS
FNNMQ0wCwYDVQQLDARyb290MRIwEAYDVQQDDAlsb2NhbGhvc3QwHhcNMjMxMDEyMTQxODIyWhcNMjMxMTExMTQxODIy
WjA0MQ8wDQYDVQQKDAZBcHBIU00xDTALBgNVBAsMBHJvb3QxEjAQBgNVBAMMCWxvY2FsaG9zdDB2MBAGByqGSM49AgE
GBSuBBAAiA2IABBgvYxsqB1lBM7XVTAt4AR0Xdoc+r4+NXZQX9loz4pvOHMsX2czCbVAlPbD7gRA9C45wv38Z6zQeKG
MyeSrrfOG432MNwJk0hFmV3MOq0uaYQ41TNgQqVKxXTlyDh7rIgKNTMFEwHQYDVR0OBBYEFHO1K4wDqgnPrMPVpbf/f
uQS+5u/MB8GA1UdIwQYMBaAFHO1K4wDqgnPrMPVpbf/fuQS+5u/MA8GA1UdEwEB/wQFMAMBAf8wCgYIKoZIzj0EAwMD
aQAwZgIxAOYE0eCuaj4PTV90XBBRncEwo66TIaT0IdF8JAv+eyuQeTw6w3hhvC9wZLZbe8tdJQIxALNKkTgc9vDqtEw
TMbd+ggRM/De8/6NYG0rijPVjoEAflnUc0TRb/3Cm8rlF28LDuA==</cert>
            </certificate>
```

2. Add the prefix (BEGIN CERTIFICATE) and suffix (END CERTIFICATE) lines to the certificate into it as shown:

```
-----BEGIN CERTIFICATE-----
MIIB+zCCAYCgAwIBAgIUfpr1mbSNp0uckfjSyuOTa3wYW68wCgYIKoZIzj0EAwMwNDEPMA0GA1UECgwGQXBwSFNNMQ0
wCwYDVQQLDARyb290MRIwEAYDVQQDDAlsb2NhbGhvc3QwHhcNMjMxMDEyMTQxODIyWhcNMjMxMTExMTQxODIyWjA0MQ
8wDQYDVQQKDAZBcHBIU00xDTALBgNVBAsMBHJvb3QxEjAQBgNVBAMMCWxvY2FsaG9zdDB2MBAGByqGSM49AgEGBSuBB
AAiA2IABBgvYxsqB1lBM7XVTAt4AR0Xdoc+r4+NXZQX9loz4pvOHMsX2czCbVAlPbD7gRA9C45wv38Z6zQeKGMyeSrr
fOG432MNwJk0hFmV3MOq0uaYQ41TNgQqVKxXTlyDh7rIgKNTMFEwHQYDVR0OBBYEFHO1K4wDqgnPrMPVpbf/fuQS+5u
/MB8GA1UdIwQYMBaAFHO1K4wDqgnPrMPVpbf/fuQS+5u/MA8GA1UdEwEB/wQFMAMBAf8wCgYIKoZIzj0EAwMDaQAwZg
IxAOYE0eCuaj4PTV90XBBRncEwo66TIaT0IdF8JAv+eyuQeTw6w3hhvC9wZLZbe8tdJQIxALNKkTgc9vDqtEwTMbd+g
gRM/De8/6NYG0rijPVjoEAflnUc0TRb/3Cm8rlF28LDuA==
-----END CERTIFICATE-----
```

3. Check the key length type and length by using the command:

```
openssl x509 -in ca.crt -text -noout
```

You will find the signature algorithm section in the output, which confirms that the ECDSA keys were used as a signature algorithm for the key: `Signature Algorithm: ecdsa-with-SHA384`

Additional feature verification tests can be found here.

Appendix A Enabling UEFI Secure Boot on Ubuntu

A.1 Introduction

This document describes the steps to enable a secure boot chain using Intel® Boot Guard and UEFI Secure Boot on a 4th Gen Intel® Xeon® 'Archer City' evaluation board. The system is running Ubuntu 22.04.03 OS, using Intel reference BIOS from the Eagle Stream Refresh BKC 2023 WW39 release.

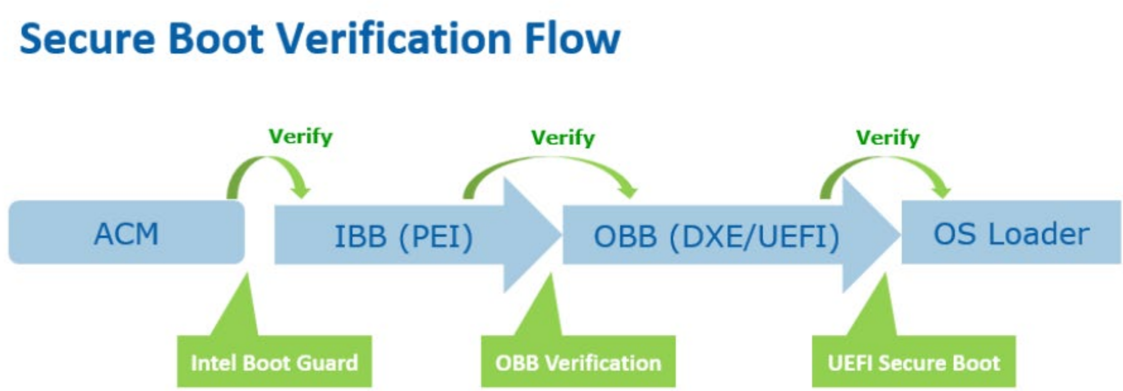
The various steps listed in the doc describe how to modify a system to enable Intel® Boot Guard and UEFI Secure Boot. The author's system was configured to boot to Ubuntu 22.03.04 prior to any changes.

These steps require access to the Archer City serial console to view BIOS log messages, and also require the ability to use the Linux command line. Archer City has a BMC connection which allows access to a serial console. Linux command line is also accessible via the serial console, or can be accessed via ssh, etc.

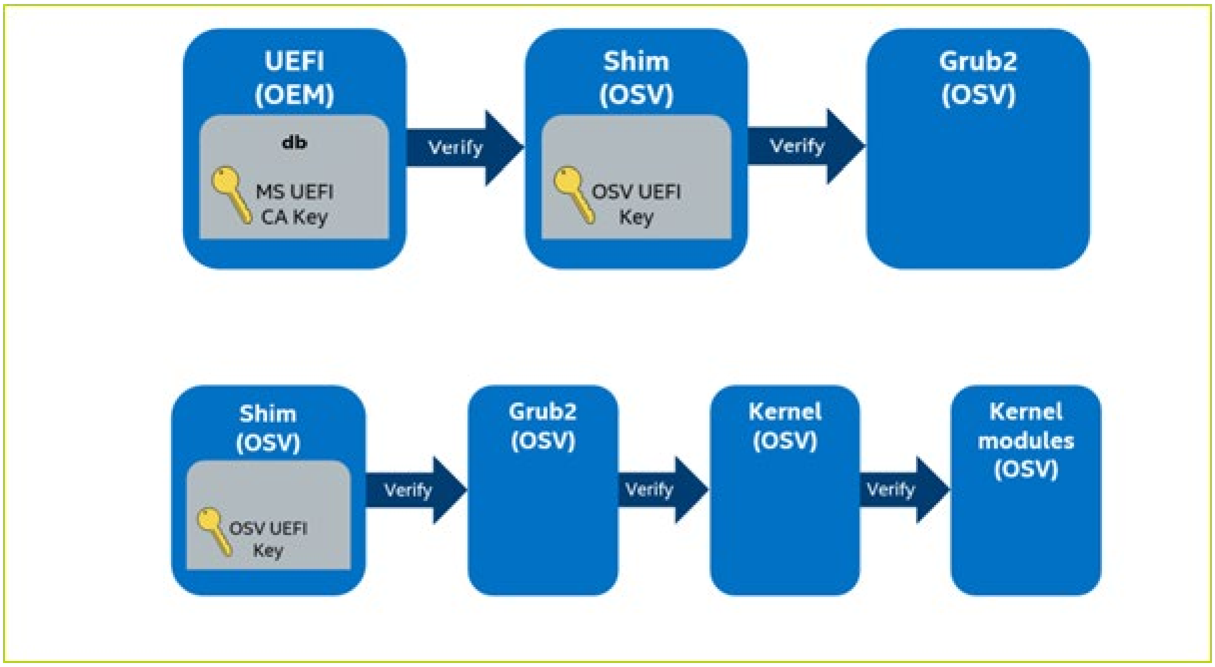
A.2 Intel® Boot Guard and UEFI Secure Boot Overview

Intel® Boot Guard and UEFI Secure Boot are 2 security technologies that protect the boot flow on Intel systems.

Intel® Boot Guard is an Intel technology that allows verification of the BIOS's Initial Boot Block (IBB), the first piece of the BIOS to execute. The verification of the IBB is performed by an Intel supplied firmware binary called 'Authenticated Code Module', ACM. The ACM executes from the processor cache prior to execution of the IBB. The IBB must then verify the OEM Boot Block (OBB) the main piece of the BIOS which executes after IBB. The OBB implements UEFI Secure Boot to verify subsequent images, such as the OS loader. The basic flow is shown below:



UEFI Secure Boot is a technology to verify code launched by UEFI firmware. It uses keys stored in BIOS to verify the digital signature of any image it loads. This allows it to verify the OS loader/bootloader, Linux kernel, and other UEFI programs. Typically this verification uses a Microsoft key pre-provisioned in the BIOS. This can cause an issue on Linux systems, where OS kernel and module images may change many times. This would require re-signing the images using Microsoft keys. Therefore, on Linux systems, it is typical for the distribution to include an app called 'shim', which acts as a first-stage bootloader. Shim can store an OS vendor key which can be used to verify subsequent images (grub, Linux OS kernel, Linux kernel modules). Shim can also store user-created keys to allow the signing of Linux OS and kernel modules. The flow is shown below:



In this document, a secure boot flow will be created which enables Boot Guard with a user-defined key and enables UEFI secure boot with a Microsoft key to verify shim, a Canonical key to verify grub bootloader, Ubuntu Linux kernel, and Canonical signed kernel modules, and a user-defined key to sign user-created kernel modules.

For more detailed information on Boot Guard and UEFI Secure Boot, see the linked document below:

https://networkbuilders.intel.com/solutionslibrary/secure-the-network-infrastructure-secure-boot-methodologies?wapkw=boot%20guard

A.3 Enabling Boot Guard

This section describes the steps to enable Intel® Boot Guard Profile 5 to secure the loading of the Initial Boot Block of the BIOS.

A.3.1 Software and Tools Required

To configure Intel® Boot Guard on an 4th Gen Intel® Xeon® system, the user requires the following software packages from Intel:

- 4th Gen Intel® Xeon® Best Known Configuration (BKC) package

  Intel regularly releases BKC packages for each processor. The BKC contains all the firmware required to build an IFWI for the processor, along with links to required tools, OS drivers and support packages, etc.

  For this app note, Eagle Stream Platform Sapphire Rapids Edge Enhanced (EE) LCC Mainline Server BKC#19 was used. Release Notes for this BKC can be found at the following link:

  https://www.intel.com/content/www/us/en/secure/content-details/788146/eagle-stream-platform-sapphire-rapids-edge-enhanced-ee-lcc-mainline-server-bkc-19-release-notes.html?wapkw=spr%20ee%20%20bkc%20%2319&DocID=788146

  The BKC consists of an updated package and associated report for the Archer City reference platform. The BKC package includes UEFI Firmware, SPS Firmware, and onboard device drivers. BKC testing is designed to verify the interoperability of the various firmware and driver components, which have already undergone extensive individual validation.

  For this app note, a prebuilt IFWI will be used as a starting point. The following link downloads a binaries.zip file which contains prebuilt IFWIs for the Archer City hardware platform.

  https://cdrdv2.intel.com/v1/dl/getContent/789801/789154?filename=binaries.zip

- BpmGen2

  BpmGen2 is a tool used to create and insert the Boot Policy Manifest and Key Manifest files required for Intel® Boot Guard. BpmGen2 can be downloaded from the Intel website at the following link:

  https://www.intel.com/content/www/us/en/secure/content-details/573188/intel-boot-policy-manifest-generatorversion-2-7-toolkit.html?wapkw=bpmgen2

(Note: The above link requires access permissions on the Intel website)

At the time of writing, the latest version was bpmgen2release2023-09-20.zip

- FITm

Modular FIT (FITm) is a tool used to create IFWIs. It is supplied in the Intel Server Platform Services package. For this app note, version 06.01.04.003.0 of Intel SPS was used. This package can be downloaded from:

https://cdrdv2.intel.com/v1/dl/getContent/789173/789172?filename=SPS_E5_06.01.04.003.0.zip

A.3.2 Setup Software

Download the BKC, BpmGen2, and SPS files given in the links in the previous section. On the author's system, these were all saved to folder **c:\share\secureboot**.

The rest of this document will use c:\share\secureboot as the root directory for all steps. Please adjust any path info to match your system if you use a different setup.

Unzip the binaries.zip, bpmgen2, and SPS packages in **c:\share\secureboot**

The above tools/packages are all used in a Windows 11 environment.

A.3.3 Create key pairs for OEM and BPM

Intel® Boot Guard requires two 3K RSA public/private key pairs to sign the OEM KM and BPM. These can be created using standard tools such as OpenSSL, or the BpmGen2 tool includes a support function to create the required keys. The steps using the BpmGen2GUI tool are shown below.

Double-click *BpmGen2GUI.exe* in bpmgen2release2023-09-20 folder to start BpmGen2GUI. On the first launch, the tool will ask for a working directory to be set. Set it to **c:\share\secureboot**.

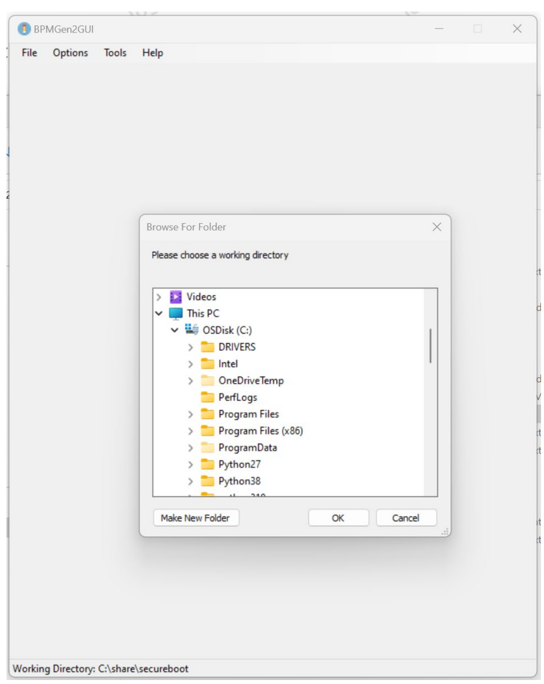


Figure 4. BpmGen2GUI Working Directory

Click on the 'Tools' menu at the top of the BpmGen2 main window. Select option 'Generate Signing Key -> RSA'. In the signing key window, set the Key Size to 3072, the Private Key Filename to 'OEM_3K_priv.pem', and the Public Key Filename to 'OEM_3K_pub.pem' as shown below. Click OK. The tool will ask if you wish to store the keys in the working directory. Select 'Yes'

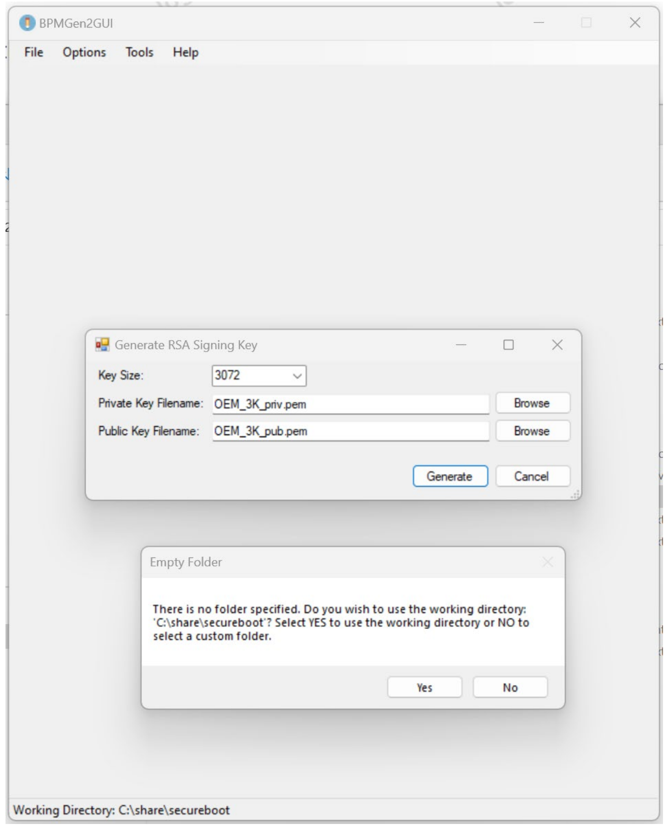


Figure 5. OEM KM Key Settings

Repeat the above step to create a 2nd keypair with filenames **BPM_3K_priv.pem and BPM_3K_pub.pem**

A.3.4 Create Key Manifest using BPMGen2

The OEM Key Manifest contains a hash of the BPM public key. The OEM KM is signed using the OEM private key. The OEM KM signature is authenticated during system boot using the OEM public key, which is part of the signature structure in the KM. A hash of the OEM public key is stored in the IFWI, and then in on-chip fuses after the device has been EOM'd. During system boot, firmware reads the OEM KM, calculates the hash of the OEM public key in the OEM KM, and then compares this hash to the value in the IFWI or in fuses. If they both match, the OEM public key is deemed to be correct, and it is used to check the OEM KM signature. If the signature is authenticated, then the OEM KM is good, and the hash of the BPM key within the OEM KM is deemed good.

To create an OEM KM compatible with the IFIW being used, open the BpmGen2GUI application and select the option 'Create Key Manifest'. Configure the GUI as shown below:

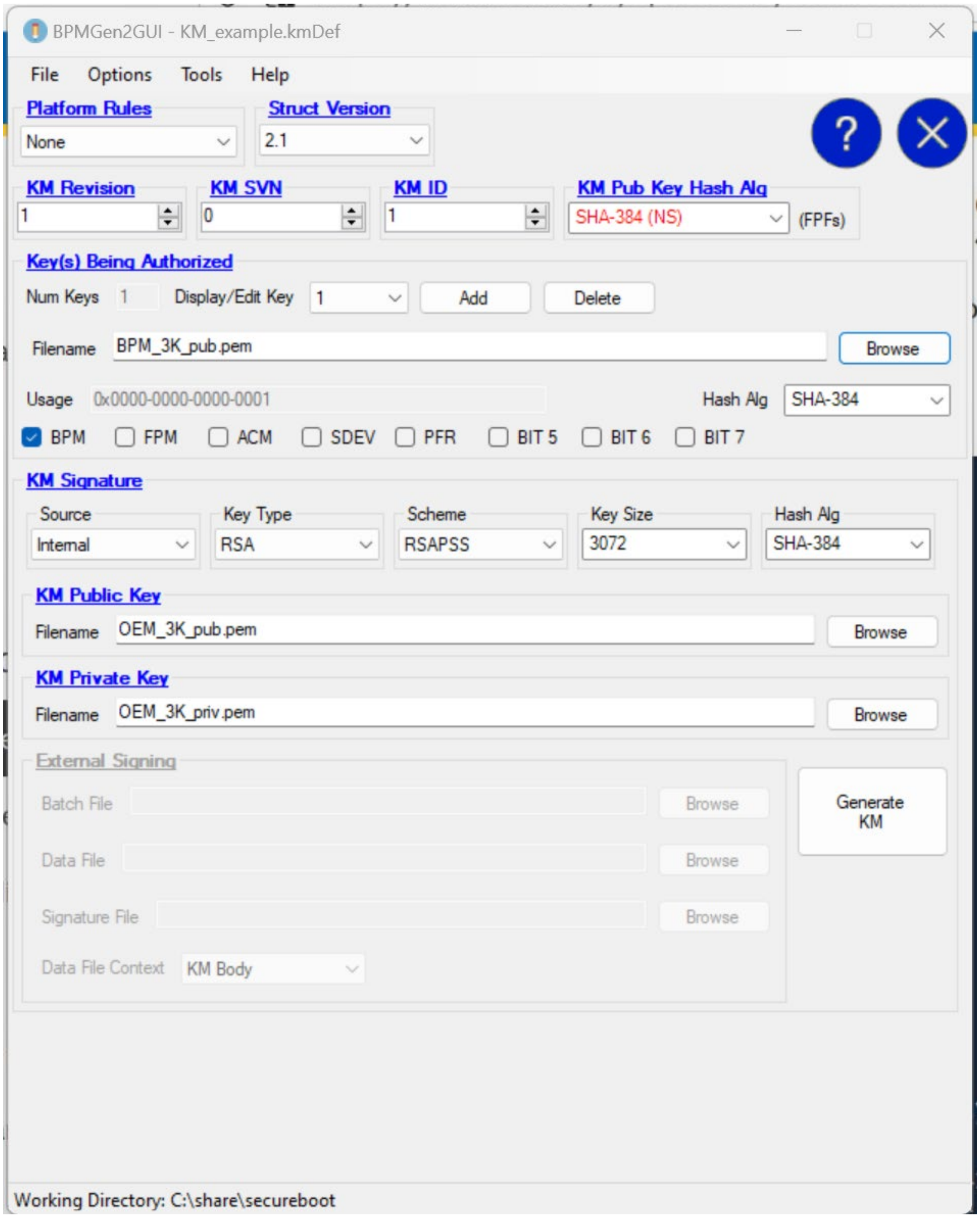


Figure 6. BpmGen2GUI OEM KM Settings

Select File -> Save, save file with name 'KM_example.kmDef'

Click 'Generate KM'. Save output file with name 'KM_example.bin'

A.3.5 Create a BPM Definition File using BPMGen2

A Boot Policy Manifest contains various boot related settings used by the ACM and BIOS during system boot. This includes hash values of the IBB and OBB sections of the BIOS. The BPM is signed with the BPM private key and is authenticated during system boot using the BPM public key. A hash of the BPM public key is stored in the OEM Key Manifest.

Creating a BPM is a two-step process. First, a BPM definition file is created using BpmGen2GUI. Later the actual BPM binary file is created using a bpmgen2 command line tool. The CLI tool is also used to insert the BPM and OEM KM binaries into the BIOS region of the IFWI.

An example BPM Definition file – BPM_example.bpDef – is listed in Appendix B of this document. It contains the settings required for the IFWI being used in this example. Copy contents of Appendix B into a file called 'BPM_example.bpDef'. This file can be opened by BPMGen2 GUI.

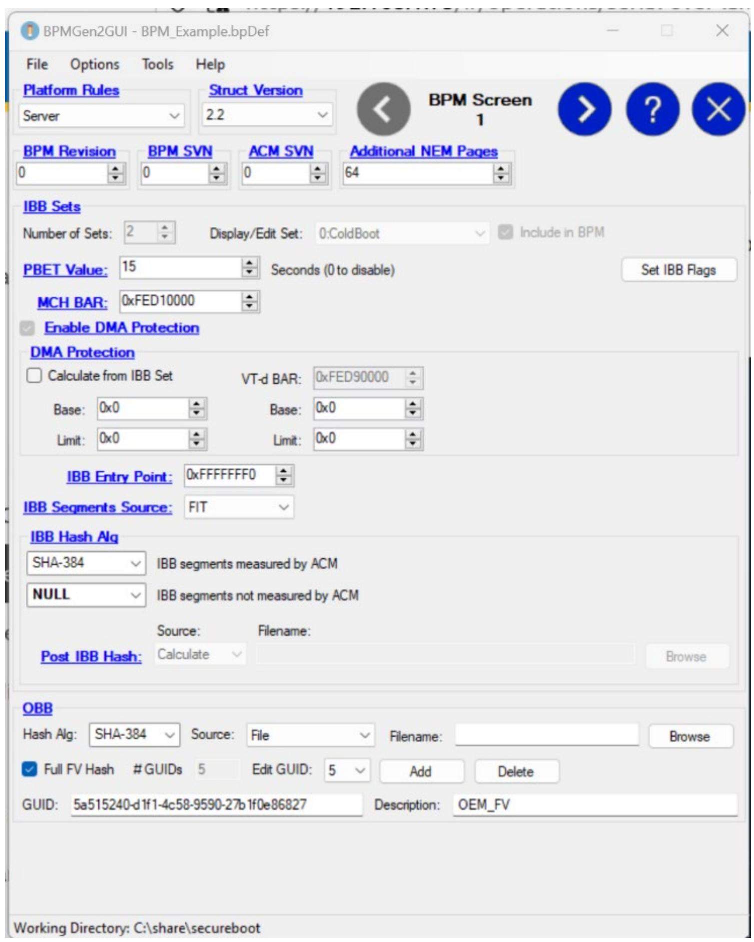


Figure 7. BpmGen2GUI BPM Screen 1

Note: the example file assumes the user has used the same working directory and key pair names as the author. If you are using different directory or different filenames, please modify the public and private key names/paths on the 'BPM Screen 3' tab in BPMGen2GUI.

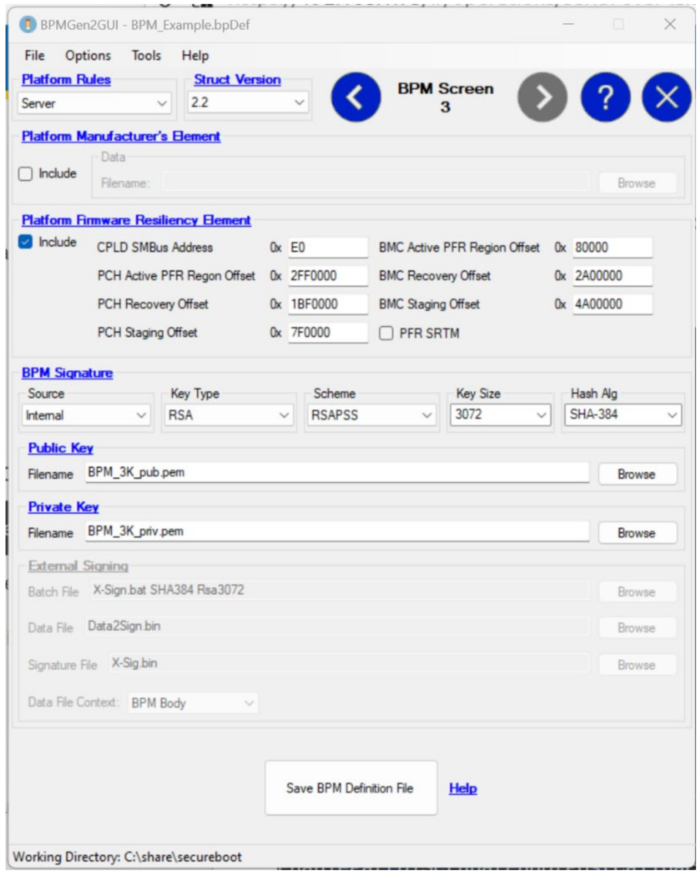


Figure 8. BpmGen2GUI BPM Screen 3

A.3.6 Insert KM and BPM Using the bpmgen2 Command Line Tool

In this section, a prebuilt IFWI will be modified by injecting the OEM Key Manifest and BPM files, using bpmgen2.

The following IFWI will be used as a starting point:

**C:\share\secureboot\binaries\production\
EGSDCRB.SYS.OR.64.2023.37.2.02.1938.1_EMR_EBG_SPS_IPClean_External_Production.bin**

Note: the user should check that the above IFWI boots on their system before making any modifications.

Open a command window and move to the BpmGen2 directory.

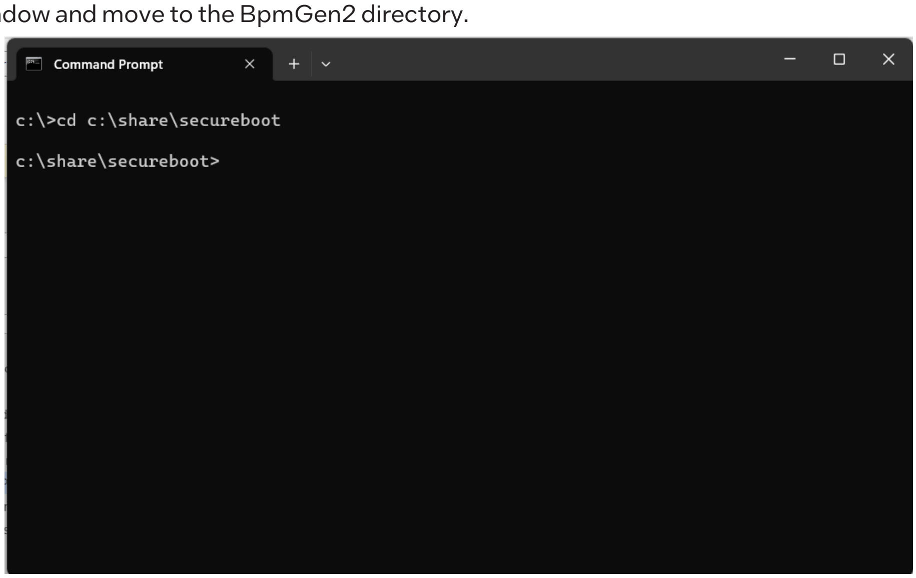
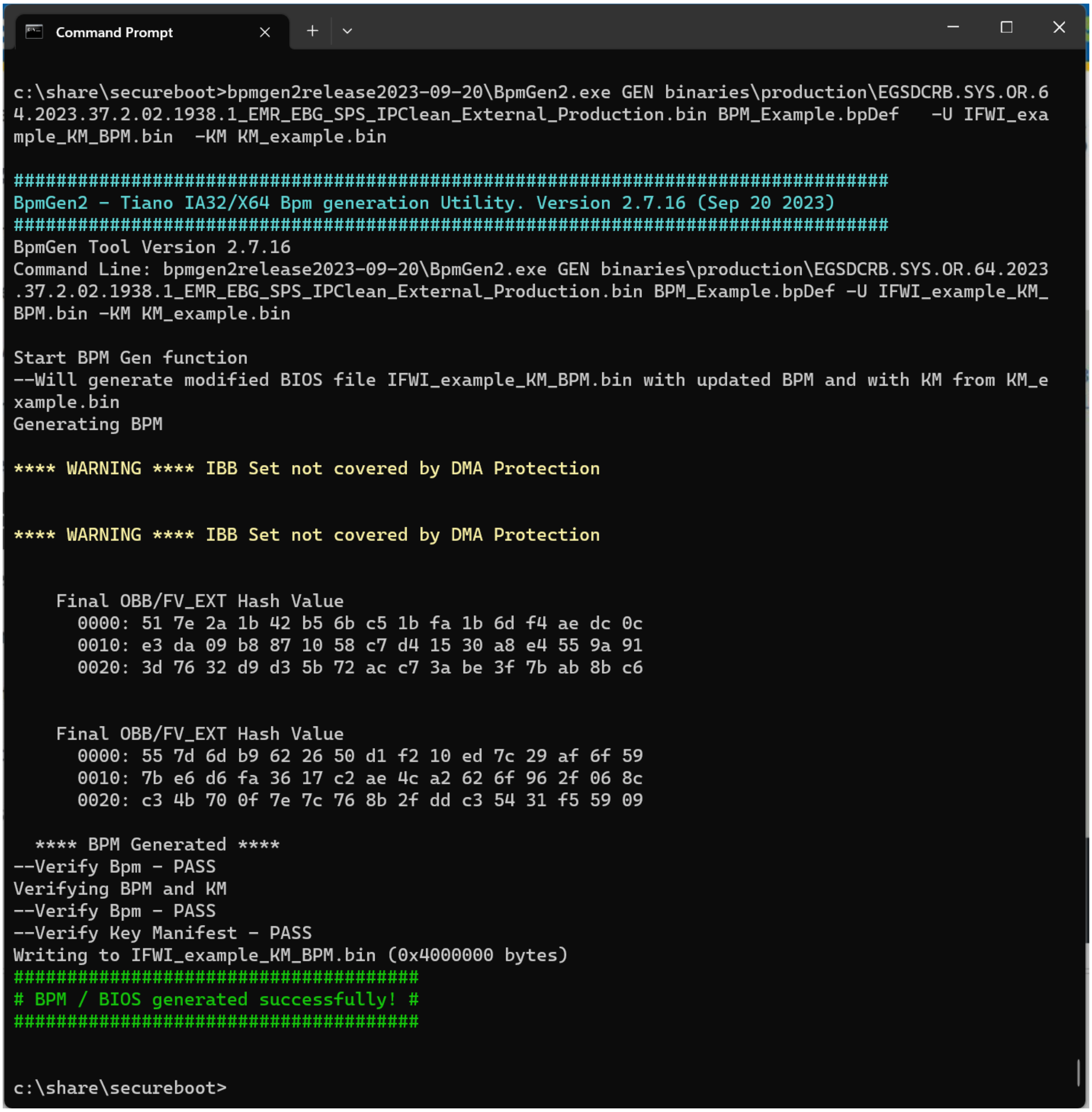


Figure 9. Windows Command Prompt

Execute the following command to create a BPM binary, then insert the BPM binary and the KM_example.bin into the IFWI. The modified IFWI is stored in file IFWI_example_KM_BPM.bin.

c:\share\secureboot>bpmgen2release2023-09-20\BpmGen2.exe GEN binaries\production\EGSDCRB.SYS.OR.64.2023.37.2.02.1938.1_EMR_EBG_SPS_IPClean_External_Production.bin BPM_Example.bpDef -U IFWI_example_KM_BPM.bin -KM KM_example.bin



```
Command Prompt

c:\share\secureboot>bpmgen2release2023-09-20\BpmGen2.exe GEN binaries\production\EGSDCRB.SYS.OR.6
4.2023.37.2.02.1938.1_EMR_EBG_SPS_IPClean_External_Production.bin BPM_Example.bpDef   -U IFWI_exa
mple_KM_BPM.bin  -KM KM_example.bin

#################################################################################
BpmGen2 - Tiano IA32/X64 Bpm generation Utility. Version 2.7.16 (Sep 20 2023)
#################################################################################
BpmGen Tool Version 2.7.16
Command Line: bpmgen2release2023-09-20\BpmGen2.exe GEN binaries\production\EGSDCRB.SYS.OR.64.2023
.37.2.02.1938.1_EMR_EBG_SPS_IPClean_External_Production.bin BPM_Example.bpDef -U IFWI_example_KM_
BPM.bin -KM KM_example.bin

Start BPM Gen function
--Will generate modified BIOS file IFWI_example_KM_BPM.bin with updated BPM and with KM from KM_e
xample.bin
Generating BPM

**** WARNING **** IBB Set not covered by DMA Protection


**** WARNING **** IBB Set not covered by DMA Protection


    Final OBB/FV_EXT Hash Value
      0000: 51 7e 2a 1b 42 b5 6b c5 1b fa 1b 6d f4 ae dc 0c
      0010: e3 da 09 b8 87 10 58 c7 d4 15 30 a8 e4 55 9a 91
      0020: 3d 76 32 d9 d3 5b 72 ac c7 3a be 3f 7b ab 8b c6


    Final OBB/FV_EXT Hash Value
      0000: 55 7d 6d b9 62 26 50 d1 f2 10 ed 7c 29 af 6f 59
      0010: 7b e6 d6 fa 36 17 c2 ae 4c a2 62 6f 96 2f 06 8c
      0020: c3 4b 70 0f 7e 7c 76 8b 2f dd c3 54 31 f5 59 09

  **** BPM Generated ****
--Verify Bpm - PASS
Verifying BPM and KM
--Verify Bpm - PASS
--Verify Key Manifest - PASS
Writing to IFWI_example_KM_BPM.bin (0x4000000 bytes)
######################################
# BPM / BIOS generated successfully! #
######################################


c:\share\secureboot>
```

Figure 10. bpmgen2.exe GEN Output

Execute the following command to dump information on the KM and BPM that have been inserted:

c:\share\secureboot>bpmgen2release2023-09-20\BpmGen2.exe INFO IFWI_example_KM_BPM.bin

```
Command Prompt

      KeySize:    0x0c00
      HashAlg:    0x000c 0x0C:SHA384
      Signature:
      0000: 62 1d e2 62 cc 41 9f 64 7e b9 4f dc 74 e8 96 96
      0010: dc 15 41 da 96 30 ab 31 9d 25 d2 c6 65 cc 41 de
      0020: 53 96 82 e5 ff b3 0a a3 c4 7b 53 b7 c0 10 b2 d4
      0030: be 17 0a ca 32 d0 7d 8d 3f 87 2f ff ba 13 2a b0
      0040: 4c bb f0 26 72 0e 06 46 f0 9d 24 49 b9 1b ef 22
      0050: ab 0b a1 5c 70 11 e8 58 9b 1b d3 e5 0d 3c fe cd
      0060: 27 d5 08 ce 7f 75 2f 8a 1a fa 8f bb 12 f2 b6 3a
      0070: a3 c5 1d 85 d4 52 72 af 93 c4 e1 03 d8 3b 66 08
      0080: a7 92 da 81 50 3e cf 7e 06 37 71 3f 5a ed f5 ef
      0090: 84 7f 47 26 a8 5d cd ab 47 9b c3 64 fd 96 58 75
      00a0: 68 e4 1a 04 3c 3d bd 0a eb 12 b7 13 a9 3e b3 48
      00b0: f9 c5 16 88 0f b3 7c cd ed 1f a5 3a f2 8d 1f a2
      00c0: 71 08 fb 1d 50 9f d9 60 f4 bb a3 61 57 5f a1 51
      00d0: b9 c2 fa 04 81 b9 85 dd c4 4b e3 7c ae 7a 46 4a
      00e0: a7 7e 93 01 a2 ad c5 1d 5d 30 9b 3a e8 32 46 b6
      00f0: 8c 12 a7 be 8c 22 45 b1 3a 8b d1 b0 80 a3 8d 80
      0100: b3 42 92 4d 19 07 81 8d 2d 49 c0 61 e4 fc 9a fa
      0110: af 3f 10 bb 04 06 a3 03 d0 45 f8 82 1d bb bc 2a
      0120: 82 36 a7 fd fd 64 15 35 5d f6 b0 29 29 f5 2e 54
      0130: 54 19 48 59 05 0c a3 74 af 84 8d db a3 69 d9 11
      0140: f5 ed cc 3d 19 ee 94 06 cd e9 93 95 2b d3 d6 1d
      0150: 0a 04 27 9d 3b 3e 1b ce 23 2a f4 6c 16 ed 22 5e
      0160: 6d e6 ed ce dc b4 66 7f f6 86 6b 93 7c d5 0a b8
      0170: 5c 9c bc 9e b6 b1 b2 05 2d 28 fc ef 02 a9 a1 33

Key Manifest Size: 0x365

# FYI: KM Public Key Hash Digest (Modulus+Exponent)#
  f1 ca a5 0b a2 a7 b9 4f 6d 98 9e 87 4d 35 b3 b5
  8a a5 bc 1c 3c 4f f3 10 68 5a 74 f9 85 cb 77 c9
  79 da d6 3c 82 67 5c 2d 6a 4d 2f c1 93 bf 51 f0

# FYI: KM Public Key Hash Digest (Modulus Only)#
  af 3c fa 66 df 99 02 1a 41 93 d4 cb d0 a9 64 2f
  4a 82 d0 06 8f e3 33 5a 62 5f 2f 61 89 20 fa 48
  7d 02 00 22 0b 03 56 1f 8f 08 04 12 fc b6 ee 86


################
# BiosDataArea #
################
-- End of Info --

c:\share\secureboot>
```

Figure 11. bpmgen2.exe INFO Output

Note the value for 'KM Public Key hash Digest (Modulus + Exponent). You will have to enter this value into the FITm tool in the next step.

A.3.7 Enable Intel® Boot Guard In IFWI Using FITm

In this step, Intel® Boot Guard will be enabled in the IFWI. FITm will be used to decompose the IFWI created in the previous step, Intel® Boot Guard settings will be changed to enabled Intel® Boot Guard profile 5, then the IFWI will be rebuilt.

Start the FITm tool by double-clicking:

C:\share\secureboot\SPS_E5_06.01.04.003.0\SPS_Tools_4.2.97.709\ModularFit_exe_gui_SPS_E5_06.01.04.003.0\FITm.exe

This will open the FITm GUI, as shown below:

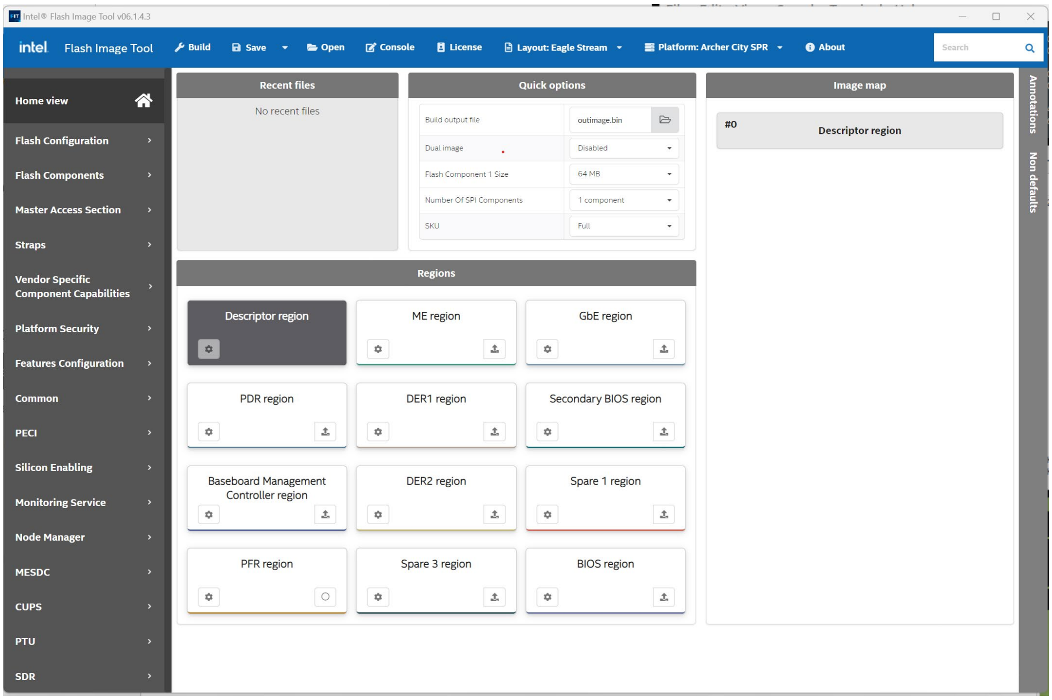


Figure 12. FITm Home View

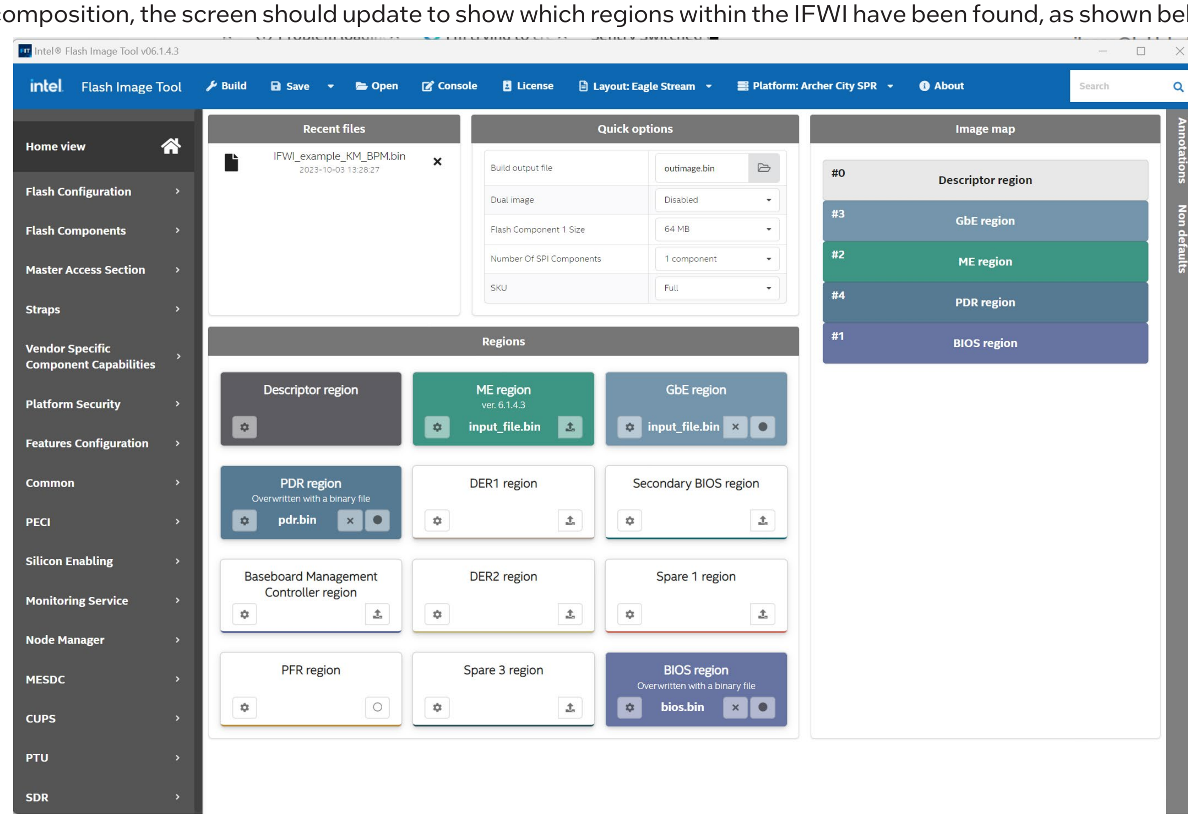
Click on the 'Open' option at the top of the FITm window. Open the following file:

C:\share\secureboot\IFWI_example_KM_BPM.bin

FITm will now import and decompose the above IFWI.

Note: During decomposition, FITm may display some warning about errors in the ME region and some other regions. These can be ignored. If the tool reports 'Full decomposition completed successfully' at the end of the run, it has succeeded.

After decomposition, the screen should update to show which regions within the IFWI have been found, as shown below.



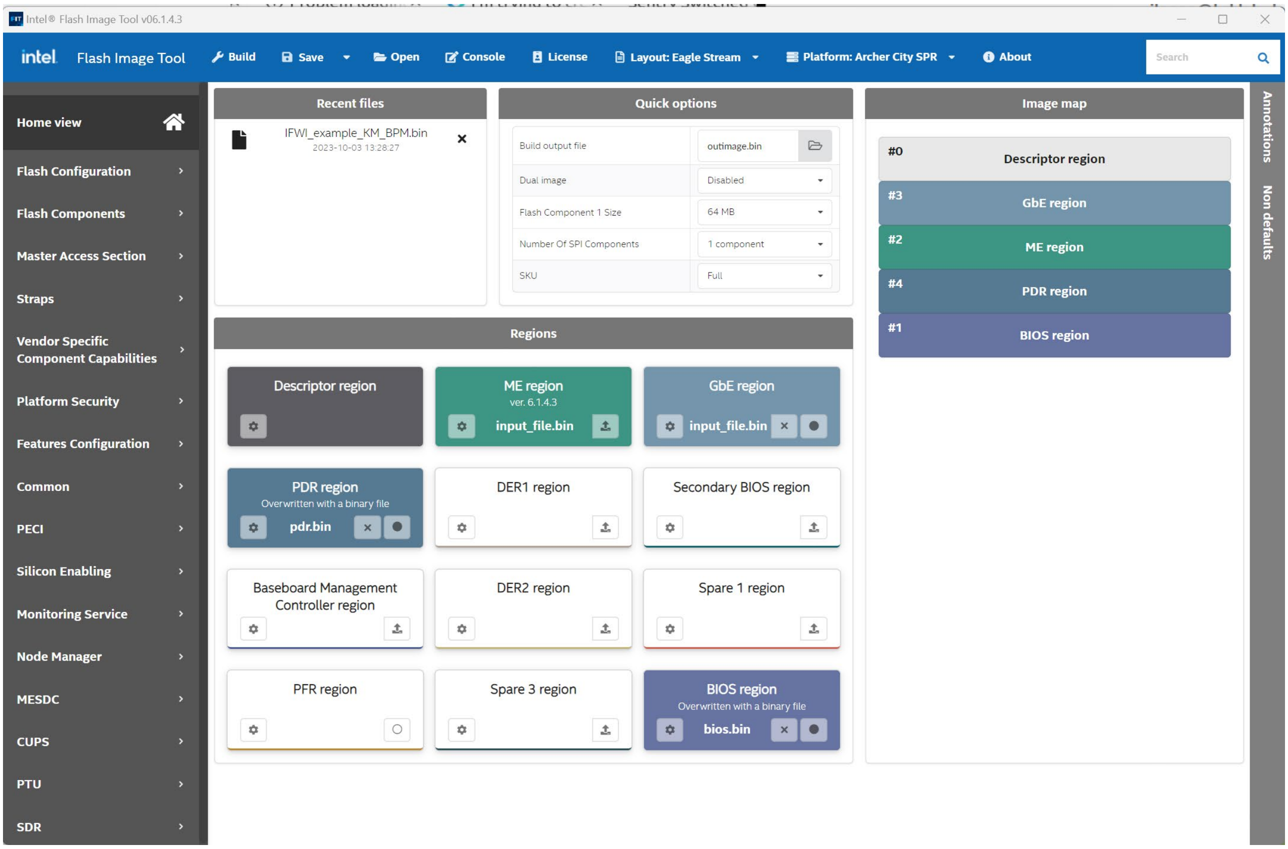


Figure 13. FITm After Decomposition

Click on the 'Platform Security' tab on the left-hand side of the FITm window. Select the 'OEM data' option. Change the screen options to match below.

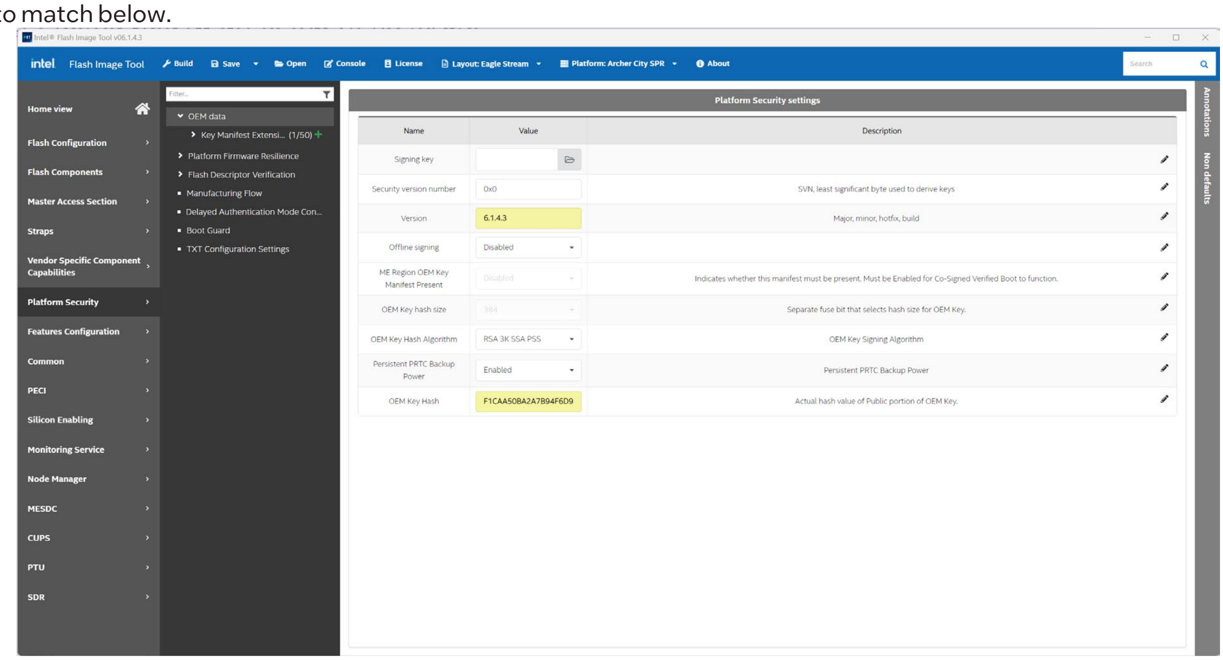


Figure 14. FITm Platform Security -> OEM Data

Note: The OEM Key Hash value will be different on each user's system because the OEM key is different on each system. Use the value of 'KM Public Key Hash Digest (Modulus + Exponent) you noted in the previous step. When entering the value, it must have no white space between the byte values. For example, on the author's system the value displayed by bpmgen2 INFO command was:

```
# FYI: KM Public Key Hash Digest (Modulus+Exponent)#
  f1 ca a5 0b a2 a7 b9 4f 6d 98 9e 87 4d 35 b3 b5
  8a a5 bc 1c 3c 4f f3 10 68 5a 74 f9 85 cb 77 c9
  79 da d6 3c 82 67 5c 2d 6a 4d 2f c1 93 bf 51 f0
```

This was reformatted to:

f1caa50ba2a7b94f6d989e874d35b3b58aa5bc1c3c4ff310685a74f985cb77c979dad63c82675c2d6a4d2fc193bf51f0

Next, select option 'Boot Guard' Change settings to match below:

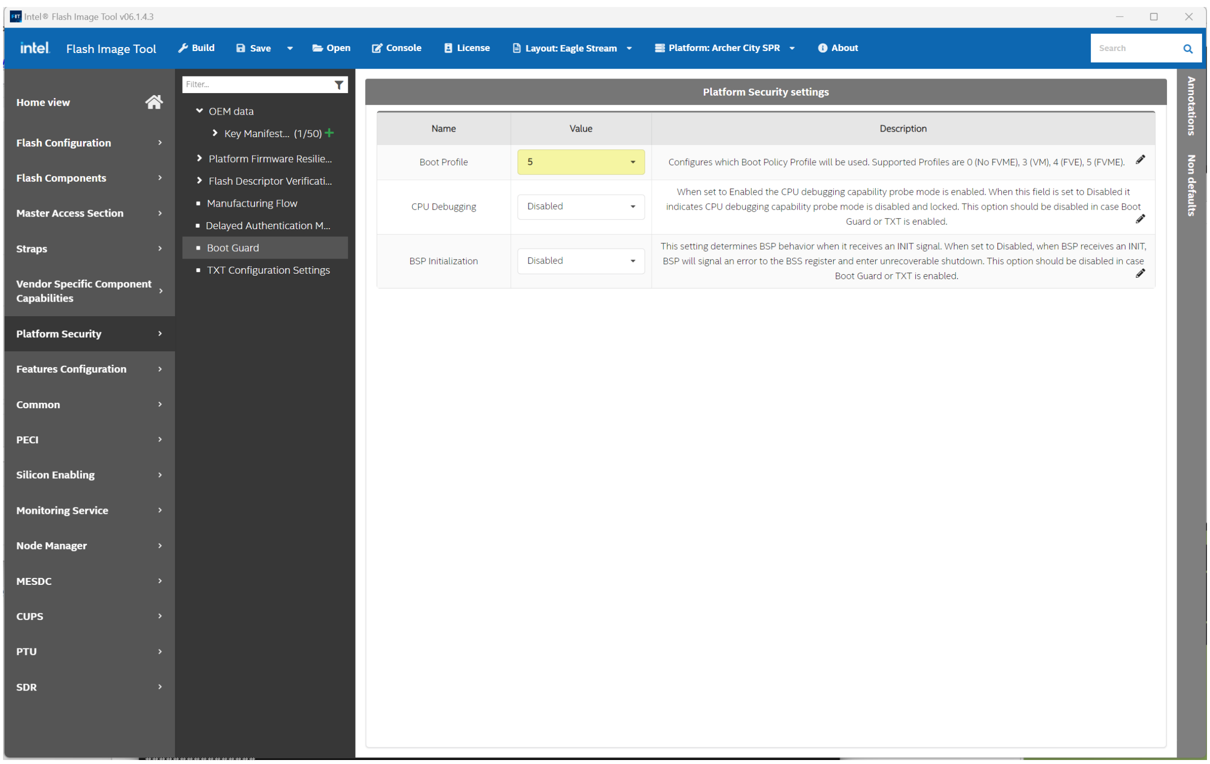


Figure 15. Boot Guard Settings

Click on the 'Home View' option on lefthand side of the main window. In the 'Build output file' box, set the filename to 'c:\share\secureboot\IFWI_example_KM_BPM_BTG5.bin'

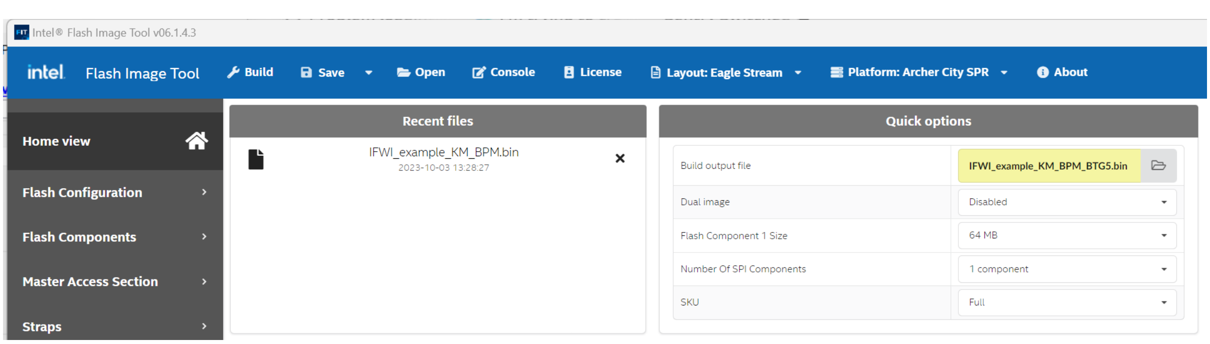


Figure 16. FITm Build Output File Setting

Click the 'Save' option at the top of main window, then set the filename to IFWI_example_fitm.xml. This will create a FITm configuration file with all the settings made in earlier steps. This configuration can load with the 'Open' option if you need to.

Now click 'Build' to create an updated IFWI with Intel® Boot Guard Profile 5 enabled.

A.3.8 Test New IFWI on the System

Burn IFWI_Example_BPM_KM BTG5.bin to serial flash on the target system, then boot the system. On the author's system, a Dediprog flash programmer was used to update the serial flash device on an Archer City evaluation board.

Boot the system with the new IFWI. If BPM and KM creation/insertion and Intel® Boot Guard enabling were successful, the system should boot to BIOS, and depending on the setup before updating IFWI, it may boot to OS.

To confirm that Intel® Boot Guard is active, Intel provides a s/w tool 'spsInfo' which can dump various boot and security-related settings, including the Intel® Boot Guard configuration.

spsInfo can be found in the following folder.

C:\share\secureboot\SPS_E5_06.01.04.003.0\SPS_Tools_4.2.97.709

There are Windows, Linux, and EFI versions of spsInfo. To run the app, place it in a location where it can be seen by the BIOS or OS, then execute the app.

For example, to run the EFI version in the BIOS, place the spsInfo.efi app in the system's FAT32 boot partition, or mount on a USB stick, enter the BIOS shell, and execute the app. The output should look like below:

```
  SB_ACM_SVN_EN:                          Enabled (1)
  SB_KM_SVN_EN:                           Enabled (1)
  SB_BSMM_SVN_EN:                         Enabled (1)
  TXT Supported:                          Yes (1)
  Error Enforcement Policy 0:             01
  Error Enforcement Policy 1:             01
  Persistent PRTC Backup Power:           Yes (0)
  VLN_EN:                                 Disabled (0)
  OEM Public Key Hash 0:                  F1 CA A5 0B A2 A7 B9 4F 6D 98 9E 87 4D 35 B3 B5
                                          8A A5 BC 1C 3C 4F F3 10 68 5A 74 F9 85 CB 77 C9
                                          79 DA D6 3C 82 67 5C 2D 6A 4D 2F C1 93 BF 51 F0
  RBE SVN:                                00
  IDLM SVN:                               00
  OEM KM SVN:                             00
  ROT KM SVN:                             00
  Secure boot ACM SVN:                    00
  Secure boot KM SVN:                     00
  Secure boot BSMM SVN:                   00
  PMC SVN:                                00
  OEM Secure Boot Policy:                    Boot Guard Profile 5 - FVME (3)

No PTU Option ROM detected in DER region.
FS0:\EFI\>
FS0:\EFI\>
```

Figure 17. spsInfo Output

Note in above that Boot Guard profile 5 is enabled and the OEM public key hash matches the value set in FITm in the previous step.

To perform the same test in Linux, copy the Linux version of spsInfo (called 'spsInfoLinux64') into your Linux filesystem, then execute with the command 'sudo ./spsInfoLinux64'.

A.4 Enabling UEFI Secure Boot

This section describes the steps to enable UEFI Secure Boot to secure the loading of the shim pre-bootloader, the grub2 bootloader used by Ubuntu, the Ubuntu kernel, and kernel modules.

The following tools/packages are required on the target system for the creation of keys/certificates and signing support:

- OpenSSL
- mokutil package
- sbsigntool package

Most Ubuntu systems have OpenSSL installed. If it is not present on your system, use the following command to install:

- sudo apt install openssl

Instructions for installing mokutil and sbsigntool are given in later sections of this document.

A.4.1 Check Shim Pre-bootloader Installed

A typical Ubuntu 22.04 install should have shim pre-installed. Shim is a small pre-bootloader that is called by the BIOS before the grub2 bootloader. Shim is signed by Microsoft certificate which is stored in db in BIOS.

There are various ways to check if shim is installed, two examples are given below:

To check if shim is installed from the Linux command line, use the command 'efibootmgr -v' to display the system's boot entries. On the author's system, the list of boot options is shown below:

```
root@brklab-legolas:~# efibootmgr -v
BootCurrent: 0004
Timeout: 10 seconds
BootOrder: 0004,0003,0002,0000,0001
Boot0000* Enter Setup   FvVol(cdbb7b35-6833-4ed6-9ab2-57d2acddf6f0)/FvFile(7b257abf-b5ec-42d5-a4cd-8e291e1f7b39)
Boot0001* UEFI Internal Shell   FvVol(cdbb7b35-6833-4ed6-9ab2-57d2acddf6f0)/FvFile(7c04a583-9e3e-4f1c-ad65-e05268d0b4d1)-.d.e.l.a.y. .5. .-.n.o.n.e.s.t..
.
Boot0002* Boot Device List      FvVol(cdbb7b35-6833-4ed6-9ab2-57d2acddf6f0)/FvFile(eec25bdc-67f2-4d95-b1d5-f81b2039d11d)
Boot0003* UEFI INTEL SSDSC2KG019T8 PHYG048401E51P9DGN   PciRoot(0x0)/Pci(0x17,0x0)/Sata(0,65535,0)N.....YM....R,Y.
Boot0004* ubuntu        HD(1,GPT,6be43297-fca2-40be-9ed0-932f809a9c0c,0x800,0x219800)/File(\EFI\ubuntu\shimx64.efi)
MirroredPercentageAbove4G: 0.00
MirrorMemoryBelow4GB: false
root@brklab-legolas:~#
```

Figure 18. Examine Boot Options Using efibootmgr

Boot entry Boot0004 is the option to boot Ubuntu OS, and it is using shimx64.efi pre-bootloader

To check if shim is installed from BIOS, open a BIOS shell, then execute the command:

- bcfg boot dump

```
Shell> bcfg boot dump
Option: 00. Variable: Boot0004
  Desc    - ubuntu
  DevPath - HD(1,GPT,6BE43297-FCA2-40BE-9ED0-932F809A9C0C,0x800,0x219800)/\EFI\ubuntu\sh
imx64.efi
  Optional- N
Option: 01. Variable: Boot0003
  Desc    - UEFI INTEL SSDSC2KG019T8 PHYG048401E51P9DGN
  DevPath - PciRoot(0x0)/Pci(0x17,0x0)/Sata(0x0,0xFFFF,0x0)
  Optional- Y
Option: 02. Variable: Boot0002
  Desc    - Boot Device List
  DevPath - Fv(CDBB7B35-6833-4ED6-9AB2-57D2ACDDF6F0)/FvFile(EEC25BDC-67F2-4D95-B1D5-F81B
2039D11D)
  Optional- N
Option: 03. Variable: Boot0000
  Desc    - Enter Setup
  DevPath - Fv(CDBB7B35-6833-4ED6-9AB2-57D2ACDDF6F0)/FvFile(7B257ABF-B5EC-42D5-A4CD-8E29
1E1F7B39)
  Optional- N
Option: 04. Variable: Boot0001
  Desc    - UEFI Internal Shell
  DevPath - Fv(CDBB7B35-6833-4ED6-9AB2-57D2ACDDF6F0)/FvFile(7C04A583-9E3E-4F1C-AD65-E052
68D0B4D1)
```

Figure 19. Examine Boot Options Using bcfg

Again, boot option Boot0004 is shown with the shim pre-bootloader present. If shim is not installed, please refer to Ubuntu documentation to install/setup.

A.4.2 Create/Download Secure Boot keys/Certificates For BIOS

UEFI Secure Boot requires provisioning of the following keys/certificates within the BIOS:

- Platform key (PK): This is a key created/supplied by the platform owner. It establishes a trust relationship between the platform owner and the platform firmware. It is used to enroll a Key Exchange Key (KEK)
- Key Exchange Key (KEK): This is key establishes a trust relationship between the operating system and platform firmware. The KEK is used to sign updates to the db and dbx databases.
- DB (Signature database): This is a list of public keys and hashes that can be used to validate UEFI boot binaries (shim, bootloader, OS kernel, etc)

When using standard prebuilt Ubuntu kernels, the PK, KEK, and DB should be populated as follows:

- PK
  - The platform owner must create this key. Typically, is an RSA 2K keypair. The public key is stored in the BIOS. The openssl commands to generate a PK and convert it into the required format are:

  - openssl req -new -x509 -newkey rsa:2048 -keyout PK.key -nodes -days 3650 -subj "/CN=YourOrgName/" -out PK.crt
  - openssl x509 -in PK.crt -out PK.cer -outform DER

- KEK
  - Microsoft KEK CA. This can be downloaded from:

  http://www.microsoft.com/pkiops/certs/MicCorKEKCA2011_2011-06-24.crt
- DB (signature database) – contains Microsoft UEFI CA, used to authenticate shim

  This can be downloaded from:

  http://www.microsoft.com/pkiops/certs/MicCorUEFCA2011_2011-06-27.crt

  Create the PK using the commands given above. Download the certificates listed for KEK and DB. Store MicCorKEKCA2011_2011-06-24.crt, MicCorUEFCA2011_2011-06-27.crt, and PK.cer in a location that will be visible to the BIOS, for example on a USB stick or a directory in the system boot partition.

  For example, on the author's system, they were placed in boot partition at /boot/ef/EFI/Certs.

Figure 20. PK, KEK and DB Certificates/Keys

A.4.3 Enrolling KEK, DB, and PK In BIOS To Enable UEFI Secure Boot

To enroll the keys/certificates from the previous section, boot the system to the BIOS setup screen. Navigate to EDKII Menu -> Secure Boot Configuration

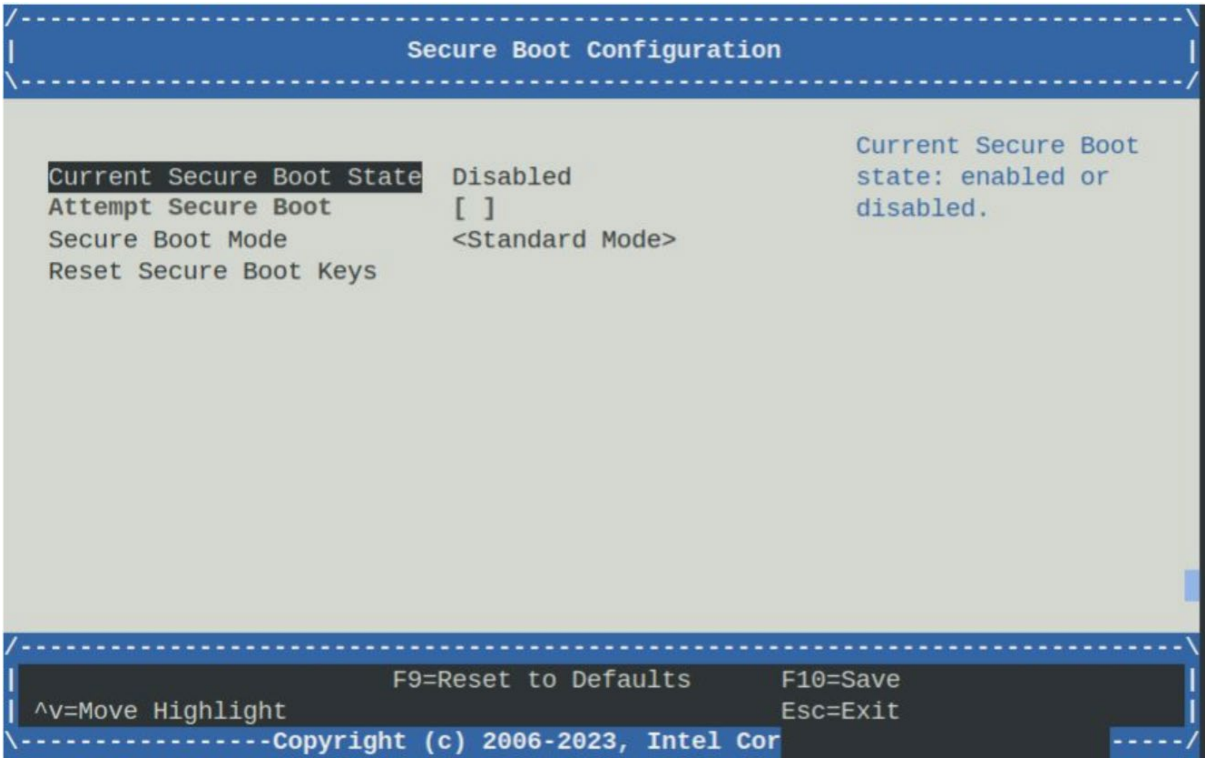


Figure 21. Secure Boot Configuration Screen

Set 'Secure Boot Mode' to 'Custom Mode' This will Add a new option 'Custom Secure Boot Options' to the screen.

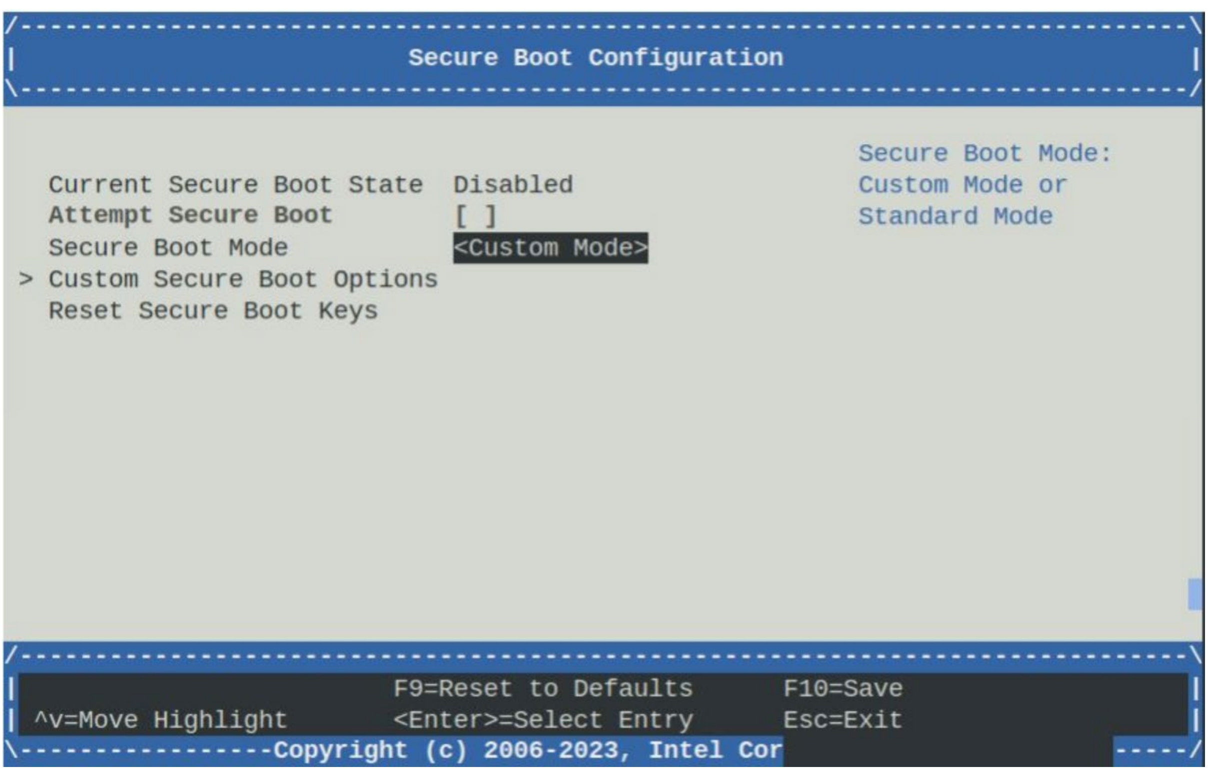


Figure 22. Secure Boot Custom Mode

Select 'Custom Secure Boot Options'. This will open a screen listing PK, KEK, DB, DBX and DBT options.

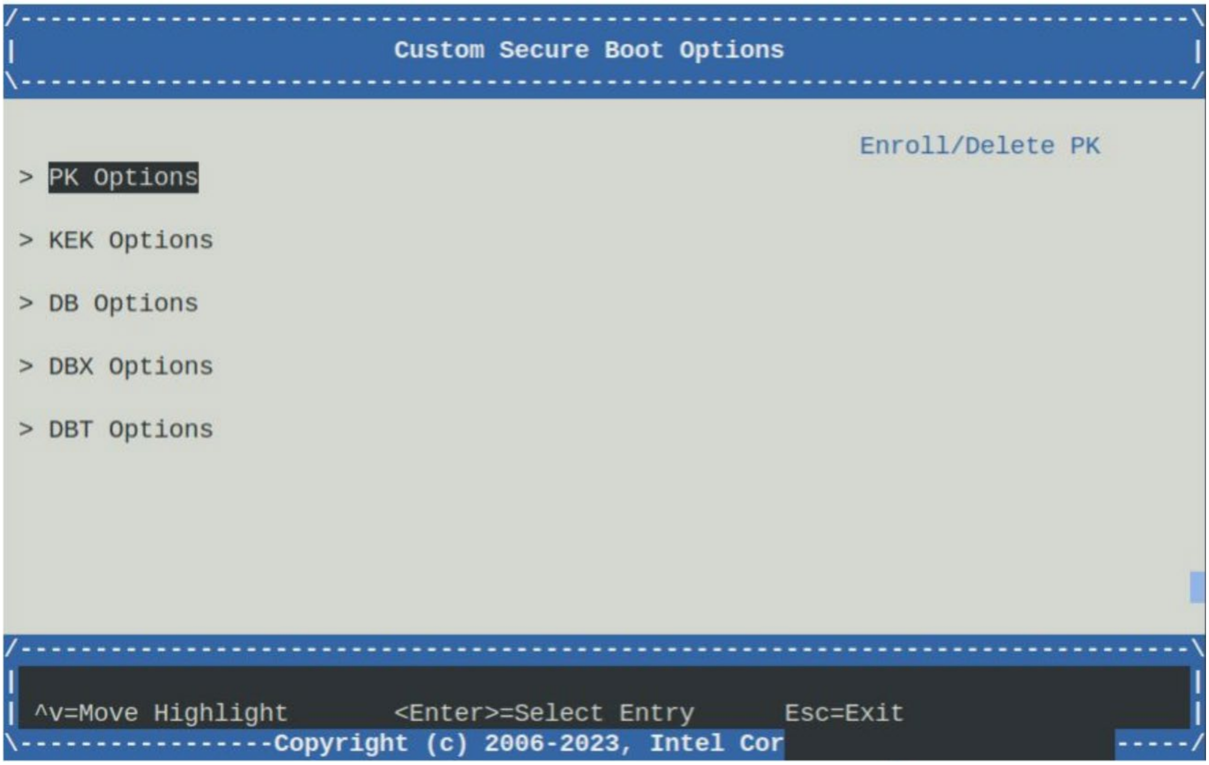


Figure 23. Custom Secure Boot Options

Select KEK Options -> Enroll KEK->Enroll KEK using File. This will list volumes on the system which can be opened to locate files. On the author's system, there is 1 Sata hard drive.

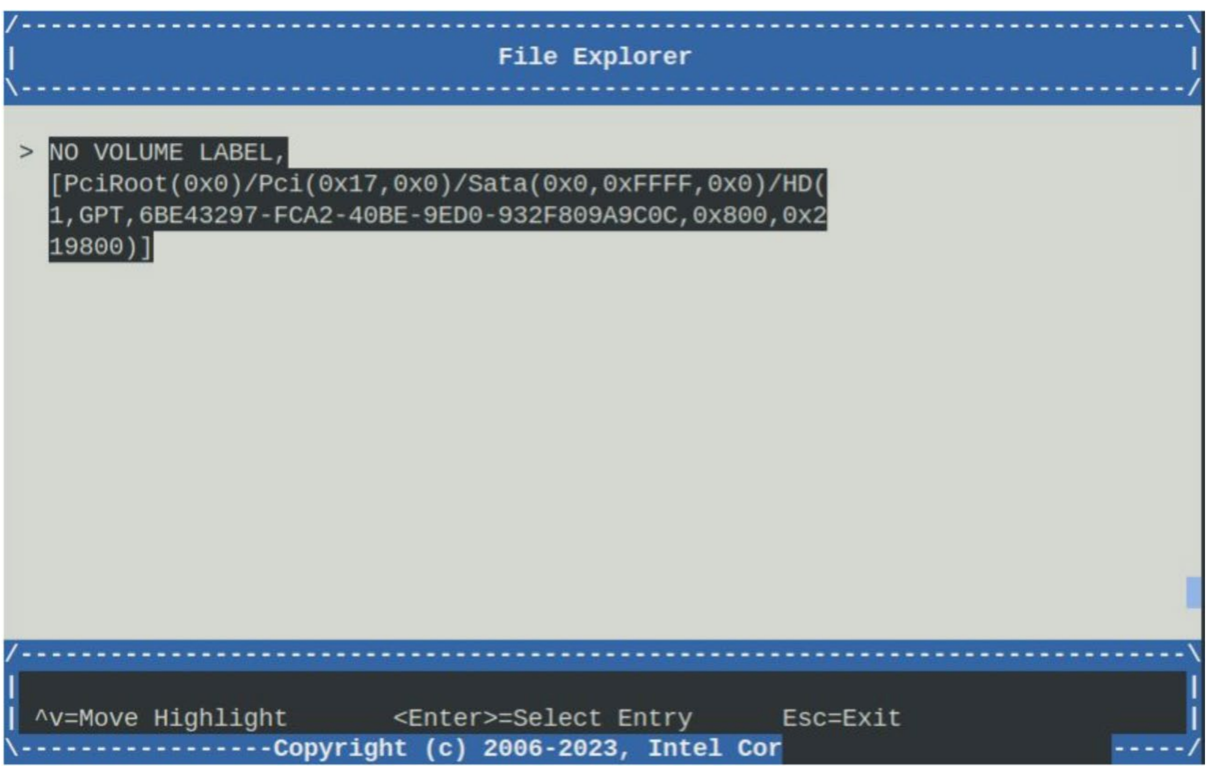


Figure 24. Secure Boot File Explorer

Open the volume where the certificate files were stored earlier.

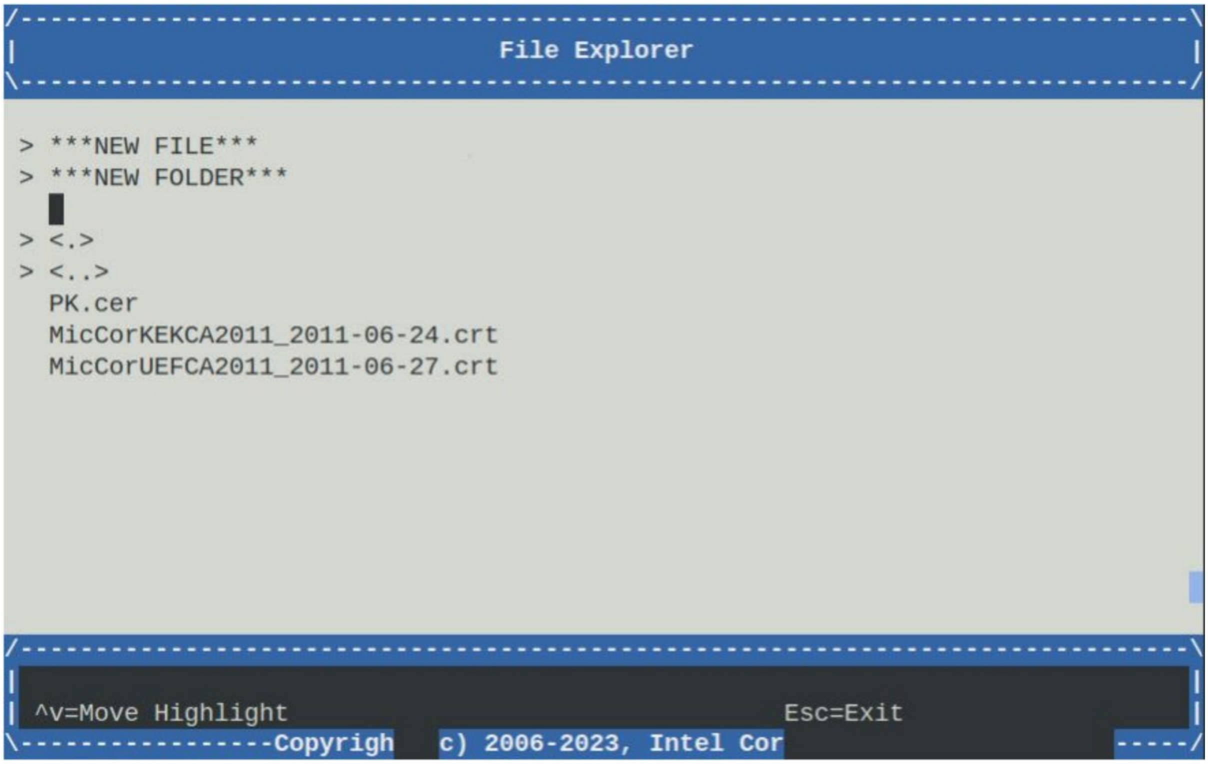


Figure 25. Secure Boot FIle Selection

On the author's system, select the MicCorKEKCA2011_2011-06-24.crt certificate. This will select the certificate and BIOS will use the public key in the certificate for KEK.

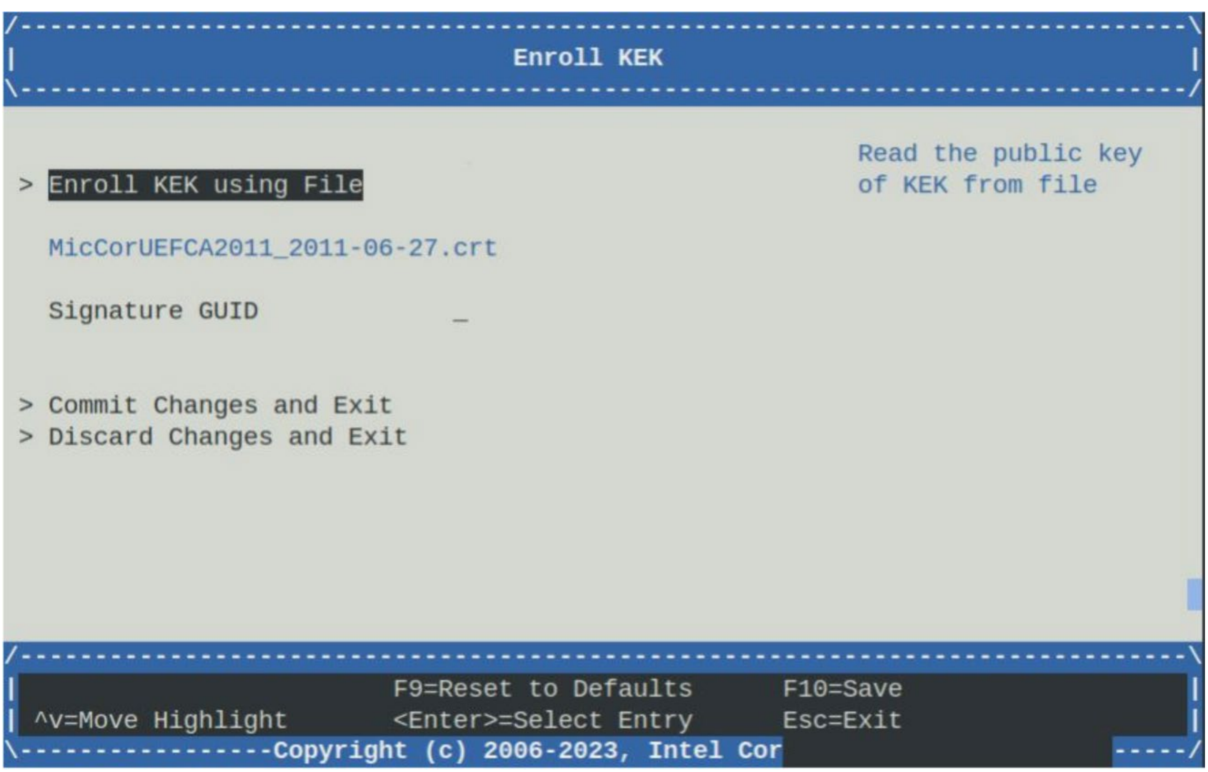


Figure 26. Enroll KEK

Select 'Commit Changes and Exit'. This will complete enrollment of the KEK, then drop back to the Custom Secure Boot Options Screen

Repeat the above steps for DB Options, enrolling MicCorUEFCA2011_2011-06-27.crt

Then repeat the above steps for PK Options, enrolling PK.cer.

Note: PK must be the last certificate to be enrolled. Do not attempt to enroll PK before KEK or DB

After PK has been enrolled, use the Esc key to drop back to the Secure Boot Configuration screen. This will now show that Secure Boot is enabled, and Secure Boot will be attempted on the next system boot:

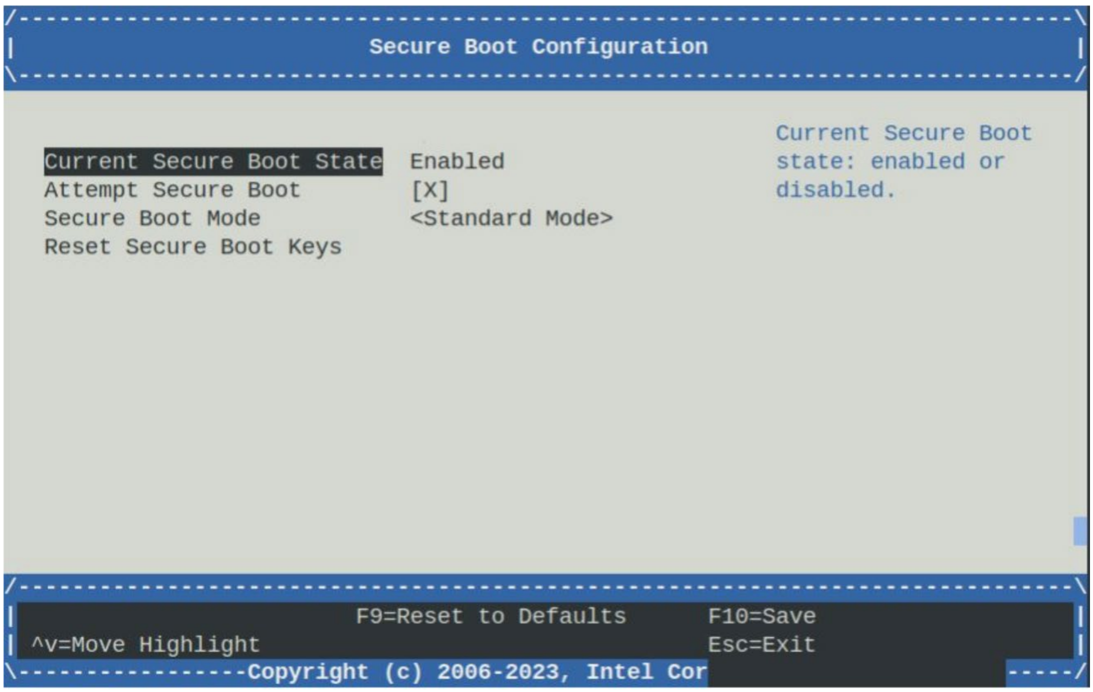


Figure 27. Secure Boot Enabled

Use the Esc key to return to top-level BIOS screen, then select 'Continue'.

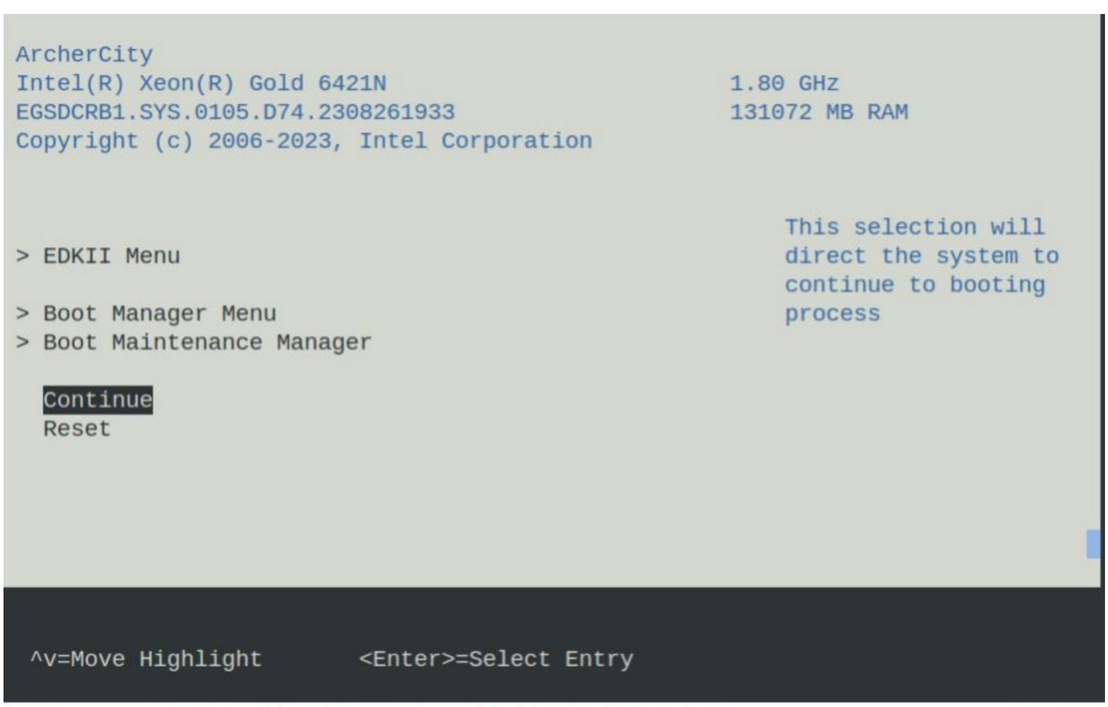


Figure 28.BIOS Top Level Screen

The BIOS will detect changes have been made and will trigger a reset and reboot.

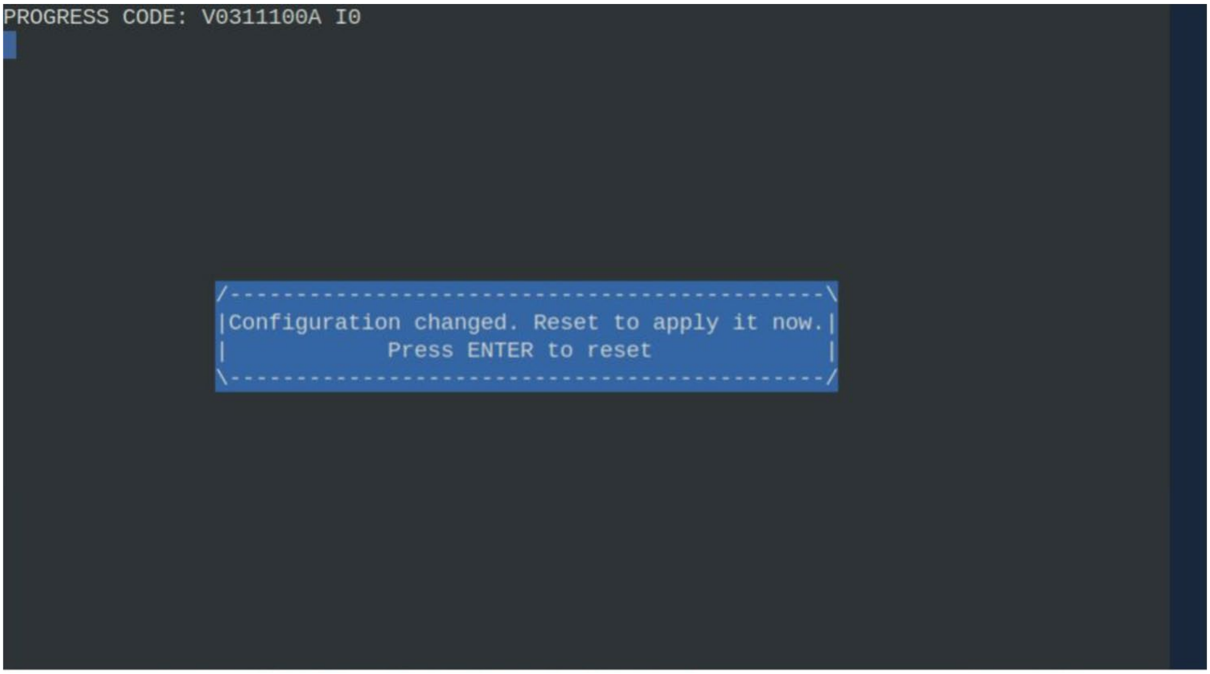


Figure 29. BIOS Reboot

Press enter and allow the system to reboot to the Linux prompt.

Figure 30. Linux prompt

A.4.4 Check UEFI Secure Boot Is Enabled

Boot the system to the Linux prompt. To check if Secure Boot was enabled using the system boot, execute the following command:

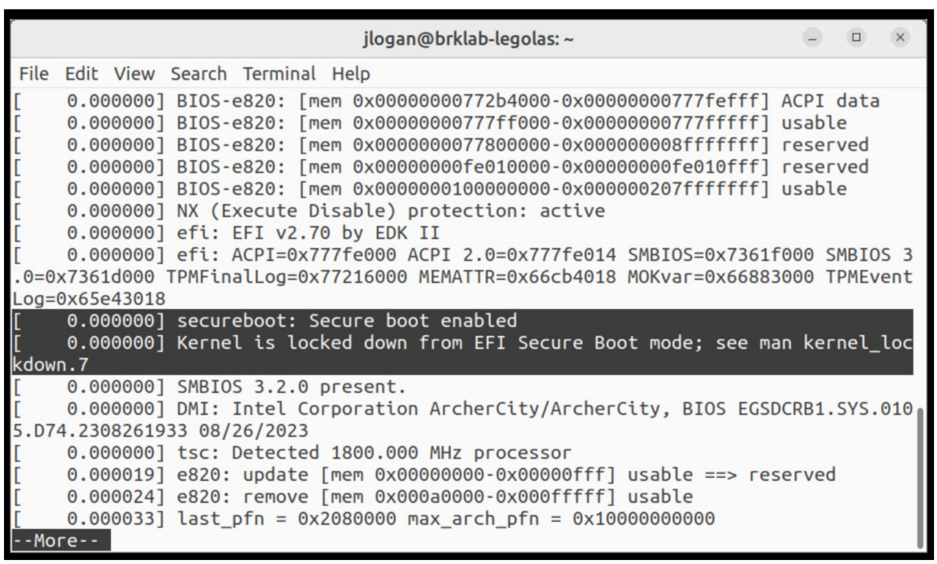
- sudo dmesg | grep "secureboot"

This should find secure boot indications in the dmesg log:

```
jlogan@brklab-legolas: ~
File  Edit  View  Search  Terminal  Help
jlogan@brklab-legolas:~$ sudo dmesg | grep "secureboot"
[    0.000000] secureboot: Secure boot enabled
[    0.018295] secureboot: Secure boot enabled
jlogan@brklab-legolas:~$
```

Figure 31. Secure Boot Enabled Messages

Searching in the dmesg log will show more info on secure boot. Use the command 'sudo dmesg | more' to examine the dmesg log:



```
jlogan@brklab-legolas: ~
File  Edit  View  Search  Terminal  Help
[    0.000000] BIOS-e820: [mem 0x00000000772b4000-0x00000000777fefff] ACPI data
[    0.000000] BIOS-e820: [mem 0x00000000777ff000-0x00000000777fffff] usable
[    0.000000] BIOS-e820: [mem 0x0000000077800000-0x000000008fffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fe010000-0x00000000fe010fff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000207fffffff] usable
[    0.000000] NX (Execute Disable) protection: active
[    0.000000] efi: EFI v2.70 by EDK II
[    0.000000] efi: ACPI=0x777fe000 ACPI 2.0=0x777fe014 SMBIOS=0x7361f000 SMBIOS 3
.0=0x7361d000 TPMFinalLog=0x77216000 MEMATTR=0x66cb4018 MOKvar=0x66883000 TPMEvent
Log=0x65e43018
[    0.000000] secureboot: Secure boot enabled
[    0.000000] Kernel is locked down from EFI Secure Boot mode; see man kernel_loc
kdown.7
[    0.000000] SMBIOS 3.2.0 present.
[    0.000000] DMI: Intel Corporation ArcherCity/ArcherCity, BIOS EGSDCRB1.SYS.010
5.D74.2308261933 08/26/2023
[    0.000000] tsc: Detected 1800.000 MHz processor
[    0.000019] e820: update [mem 0x00000000-0x00000fff] usable ==> reserved
[    0.000024] e820: remove [mem 0x000a0000-0x000fffff] usable
[    0.000033] last_pfn = 0x2080000 max_arch_pfn = 0x10000000000
--More--
```

Figure 32. dmesg Log

A.4.5 Install mokutil Linux utility

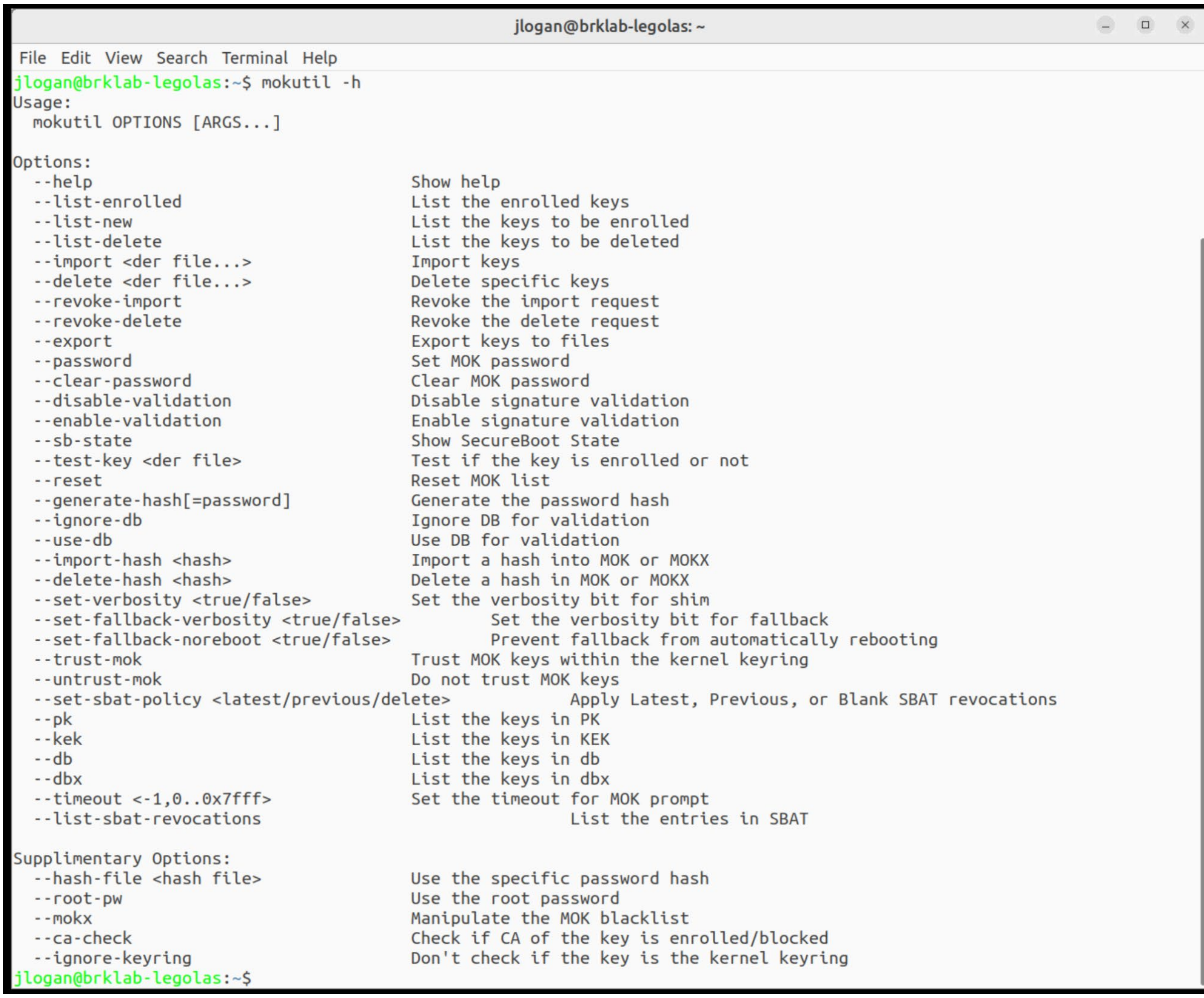
At this point, UEFI Secure Boot is enabled and will authenticate shim, grub bootloader, Linux OS kernel and OS kernel models during boot. The kernel and kernel modules must be signed by Canonical and will be authenticated by the Canonical Ltd Secure Boot Signing key, which is embedded in shim. In the following sections, the user will add their own key into shim which can be used to sign user created/supplied kernel modules.

The keys stored in shim database are known as 'Machine Owner Keys' (MOK). mokutil is a tool that allows the user to import or delete MOKs.

Use the following command to install mokutil package on the target system:

- sudo apt install mokutil

Command 'mokutil -h' will display help info:



```
jlogan@brklab-legolas: ~
File  Edit  View  Search  Terminal  Help
jlogan@brklab-legolas:~$ mokutil -h
Usage:
  mokutil OPTIONS [ARGS...]

Options:
  --help                                Show help
  --list-enrolled                       List the enrolled keys
  --list-new                            List the keys to be enrolled
  --list-delete                         List the keys to be deleted
  --import <der file...>                Import keys
  --delete <der file...>                Delete specific keys
  --revoke-import                       Revoke the import request
  --revoke-delete                       Revoke the delete request
  --export                              Export keys to files
  --password                            Set MOK password
  --clear-password                      Clear MOK password
  --disable-validation                  Disable signature validation
  --enable-validation                   Enable signature validation
  --sb-state                            Show SecureBoot State
  --test-key <der file>                 Test if the key is enrolled or not
  --reset                               Reset MOK list
  --generate-hash[=password]            Generate the password hash
  --ignore-db                           Ignore DB for validation
  --use-db                              Use DB for validation
  --import-hash <hash>                  Import a hash into MOK or MOKX
  --delete-hash <hash>                  Delete a hash in MOK or MOKX
  --set-verbosity <true/false>          Set the verbosity bit for shim
  --set-fallback-verbosity <true/false>         Set the verbosity bit for fallback
  --set-fallback-noreboot <true/false>          Prevent fallback from automatically rebooting
  --trust-mok                           Trust MOK keys within the kernel keyring
  --untrust-mok                         Do not trust MOK keys
  --set-sbat-policy <latest/previous/delete>            Apply Latest, Previous, or Blank SBAT revocations
  --pk                                  List the keys in PK
  --kek                                 List the keys in KEK
  --db                                  List the keys in db
  --dbx                                 List the keys in dbx
  --timeout <-1,0..0x7fff>              Set the timeout for MOK prompt
  --list-sbat-revocations                               List the entries in SBAT

Supplimentary Options:
  --hash-file <hash file>               Use the specific password hash
  --root-pw                             Use the root password
  --mokx                                Manipulate the MOK blacklist
  --ca-check                            Check if CA of the key is enrolled/blocked
  --ignore-keyring                      Don't check if the key is the kernel keyring
jlogan@brklab-legolas:~$
```

Figure 33 mokutil Help Info

Use the following command to display the MOKs currently enrolled in shim:

- mokutil --list-enrolled

This will display the Canonical certificate:

```
jlogan@brklab-legolas: ~
File  Edit  View  Search  Terminal  Help
jlogan@brklab-legolas:~$ mokutil --list-enrolled
[key 1]
SHA1 Fingerprint: 76:a0:92:06:58:00:bf:37:69:01:c3:72:cd:55:a9:0e:1f:de:d2:e0
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b9:41:24:a0:18:2c:92:67
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=GB, ST=Isle of Man, L=Douglas, O=Canonical Ltd., CN=Canonical Ltd. Master Certificate Authority
        Validity
            Not Before: Apr 12 11:12:51 2012 GMT
            Not After : Apr 11 11:12:51 2042 GMT
        Subject: C=GB, ST=Isle of Man, L=Douglas, O=Canonical Ltd., CN=Canonical Ltd. Master Certificate Authority
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:bf:5b:3a:16:74:ee:21:5d:ae:61:ed:9d:56:ac:
                    bd:de:de:72:f3:dd:7e:2d:4c:62:0f:ac:c0:6d:48:
                    08:11:cf:8d:8b:fb:61:1f:27:cc:11:6e:d9:55:3d:
                    39:54:eb:40:3b:b1:bb:e2:85:34:79:ca:f7:7b:bf:
                    ba:7a:c8:10:2d:19:7d:ad:59:cf:a6:d4:e9:4e:0f:
                    da:ae:52:ea:4c:9e:90:ce:c6:99:0d:4e:67:65:78:
                    5d:f9:d1:d5:38:4a:4a:7a:8f:93:9c:7f:1a:a3:85:
                    db:ce:fa:8b:f7:c2:a2:21:2d:9b:54:41:35:10:57:
                    13:8d:6c:bc:29:06:50:4a:7e:ea:99:a9:68:a7:3b:
                    c7:07:1b:32:9e:a0:19:87:0e:79:bb:68:99:2d:7e:
                    93:52:e5:f6:eb:c9:9b:f9:2b:ed:b8:68:49:bc:d9:
                    95:50:40:5b:c5:b2:71:aa:eb:5c:57:de:71:f9:40:
                    0a:dd:5b:ac:1e:84:2d:50:1a:52:d6:e1:f3:6b:6e:
                    90:64:4f:5b:b4:eb:20:e4:61:10:da:5a:f0:ea:e4:
                    42:d7:01:c4:fe:21:1f:d9:b9:c0:54:95:42:81:52:
                    72:1f:49:64:7a:c8:6c:24:f1:08:70:0b:4d:a5:a0:
                    32:d1:a0:1c:57:a8:4d:e3:af:a5:8e:05:05:3e:10:
                    43:a1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                AD:91:99:0B:C2:2A:B1:F5:17:04:8C:23:B6:65:5A:26:8E:34:5A:63
            X509v3 Authority Key Identifier:
                AD:91:99:0B:C2:2A:B1:F5:17:04:8C:23:B6:65:5A:26:8E:34:5A:63
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage:
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://www.canonical.com/secure-boot-master-ca.crl
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        3f:7d:f6:76:a5:b3:83:b4:2b:7a:d0:6d:52:1a:03:83:c4:12:
        a7:50:9c:47:92:cc:c0:94:77:82:d2:ae:57:b3:99:04:f5:32:
        3a:c6:55:1d:07:db:12:a9:56:fa:d8:d4:76:20:eb:e4:c3:51:
        db:9a:5c:9c:92:3f:18:73:da:94:6a:a1:99:38:8c:a4:88:6d:
        c1:fc:39:71:d0:74:76:16:03:3e:56:23:35:d5:55:47:5b:1a:
        1d:41:c2:d3:12:4c:dc:ff:ae:0a:92:9c:62:0a:17:01:9c:73:
        e0:5e:b1:fd:bc:d6:b5:19:11:7a:7e:cd:3e:03:7e:66:db:5b:
        a8:c9:39:48:51:ff:53:e1:9c:31:53:91:1b:3b:10:75:03:17:
        ba:e6:81:02:80:94:70:4c:46:b7:94:b0:3d:15:cd:1f:8e:02:
        e0:68:02:8f:fb:f9:47:1d:7d:a2:01:c6:07:51:c4:9a:cc:ed:
        dd:cf:a3:5d:ed:92:bb:be:d1:fd:e6:ec:1f:33:51:73:04:be:
        3c:72:b0:7d:08:f8:01:ff:98:7d:cb:9c:e0:69:39:77:25:47:
        71:88:b1:8d:27:a5:2e:a8:f7:3f:5f:80:69:97:3e:a9:f4:99:
        14:db:ce:03:0e:0b:66:c4:1c:6d:bd:b8:27:77:c1:42:94:bd:
        fc:6a:0a:bc
jlogan@brklab-legolas:~$
```

Figure 34. Canonical Certificate

In the next section, the user will create an additional MOK and enroll into shim.

A.4.6 Create MOK Key and Enroll in Shim

In this section, the user will create a certificate containing a public MOK using OpenSSL. This MOK will be used to sign a user-created kernel module.

First, create a config file containing the info shown below. Save this file with the name 'testMOK.cnf'

```
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd
```

```
[req]
distinguished_name  = req_distinguished_name
x509_extensions           = v3
string_mask               = utf8only
prompt              = no

[req_distinguished_name]
countryName         =  <Your country code>
stateOrProvinceName =  <Your Province>
0.organizationName  = <Your Org name >
commonName                = Secure Boot Signing
emailAddress          = <You email address>


[v3]
subjectKeyIdentifier      = hash
authorityKeyIdentifier    = keyid:always,issuer
basicConstraints          = critical,CA:FALSE
extendedKeyUsage    = codeSigning, 1.3.6.1.4.1.311.10.3.6, 1.3.6.1.4.1.2312.16.1.2
nsComment           = "OpenSSL Generated Certificate"
```

Change the entries with <YOUR*> to the user's details.

Use the following command to create a certificate containing the MOK certificate containing the public key , and a file containing the private key:

```
openssl req -config ./testMOK.cnf  \
      -new -x509 -newkey rsa:2048 \
      -nodes -days 3650 -outform DER \
      -keyout MOK.priv  \
      -out  MOK.der
```

Figure 35 Openssl MOK Certificate Creation

To display the contents of the MOK certificate, use command:

- openssl x509 -in MOK.der -noout -text

```
jlogan@brklab-legolas: ~/BootSecurity/MOK
File  Edit  View  Search  Terminal  Help
jlogan@brklab-legolas:~/BootSecurity/MOK$ openssl x509 -in MOK.der -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3f:36:0c:47:0f:73:ed:43:3d:bc:8a:6f:f2:66:bc:42:fd:d7:7a:37
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = UK, ST = Scotland, O = Test Org, CN = Secure Boot Signing, emailAddress = example@example.com
        Validity
            Not Before: Oct  5 10:27:38 2023 GMT
            Not After : Oct  2 10:27:38 2033 GMT
        Subject: C = UK, ST = Scotland, O = Test Org, CN = Secure Boot Signing, emailAddress = example@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:87:c6:56:b1:e1:5b:59:37:eb:eb:12:db:5f:7c:
                    2a:f7:7c:71:77:21:ae:d5:36:02:cd:4d:2b:61:6d:
                    26:43:96:b3:79:c0:ba:21:17:4f:7b:e7:e3:3a:22:
                    52:59:36:32:b3:0b:bb:ca:aa:19:b0:0a:21:6e:8e:
                    f7:06:12:e5:38:8b:98:a6:9e:58:1a:e5:5c:76:e0:
                    84:1b:41:0f:06:ac:37:88:02:e3:47:44:6f:03:d4:
                    6e:d9:a7:f8:3b:3d:fa:3d:10:78:47:b5:b9:7f:64:
                    80:95:5f:8c:00:e4:2e:c4:01:63:78:2d:12:67:40:
                    ab:1d:b9:80:bd:b8:5b:02:04:c9:92:08:0c:71:74:
                    fe:ad:e3:e6:47:0b:ed:f4:56:52:4d:89:72:e5:31:
                    2c:1f:90:89:e9:72:2d:c3:b7:45:bf:f0:31:b4:80:
                    f6:99:3e:e4:59:04:76:f0:6b:b0:50:aa:fb:31:b4:
                    10:7f:d1:43:cb:43:d0:6c:04:ce:08:d0:61:e7:83:
                    86:05:a9:d6:89:37:e9:bf:f0:08:41:6d:45:64:ff:
                    95:d9:57:20:5c:64:62:db:70:0e:ef:63:58:b5:b9:
                    f7:d2:c1:d1:40:fd:90:f4:bc:d0:76:64:f0:3c:72:
                    fa:46:2c:a5:13:09:73:64:7b:62:51:cb:ed:e3:eb:
                    3a:ff
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                9A:EE:B2:0D:A4:A9:55:33:01:CE:67:95:E2:A7:F8:13:64:CD:1C:0D
            X509v3 Authority Key Identifier:
                9A:EE:B2:0D:A4:A9:55:33:01:CE:67:95:E2:A7:F8:13:64:CD:1C:0D
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Extended Key Usage:
                Code Signing, 1.3.6.1.4.1.311.10.3.6, 1.3.6.1.4.1.2312.16.1.2
            Netscape Comment:
                OpenSSL Generated Certificate
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        52:e0:76:12:20:aa:93:57:65:84:5a:e1:b5:6b:1d:7b:2b:0c:
        1c:5c:f7:86:5a:58:d2:4f:e4:43:19:e8:83:35:d0:8e:51:5c:
        89:ac:06:b7:a7:8a:de:0a:1e:7c:63:48:8e:8a:3e:63:ca:c5:
        a2:92:36:38:44:75:ed:da:be:59:7d:01:5c:9b:b7:be:6d:90:
        b4:15:d4:1f:d8:a4:f7:a6:08:25:3b:b9:22:e1:3c:d0:c4:93:
        ea:96:13:79:d9:0d:7b:18:a1:76:bc:d7:21:33:8c:51:2f:01:
        af:f5:31:c8:b0:72:de:c5:ef:be:95:53:d5:26:97:37:26:31:
        14:be:86:e4:00:63:76:ab:7a:5a:d1:4f:09:2d:ec:b8:bb:b1:
        9b:38:8b:84:ff:80:86:aa:4a:47:f8:a7:96:c5:eb:24:c2:4b:
        73:bd:3d:f0:16:55:49:e2:0f:7e:a9:5e:ba:62:2e:62:73:8e:
        50:d7:ac:64:19:c6:96:52:7c:97:87:a7:0a:9c:40:f4:5d:32:
        ca:93:52:13:69:cb:b0:44:f0:b9:1e:8e:fe:4a:20:9f:81:02:
        49:ef:d8:f2:c9:93:35:cb:0b:31:41:54:61:59:a8:7a:07:c7:
        0a:9c:3c:35:54:8c:66:14:87:63:c3:52:74:0f:c9:00:4a:16:
        d0:f0:52:8b
jlogan@brklab-legolas:~/BootSecurity/MOK$
```

Figure 36. User Created MOK Certificate

Convert the private key to PEM format with the command:

- openssl x509 -inform DER -in MOK.der -outform PEM -out MOK.pem

Figure 37. MOK Certificate and Key Files

To enroll the certificate in shim using mokutil, use the following command:

- sudo mokutil --import MOK.der

This command will ask the user to input a password, and will then reboot the system. The password will be required after the next system reboot to authorize adding the MOK certificate to shim.

Please enter a password and remember it for use later.

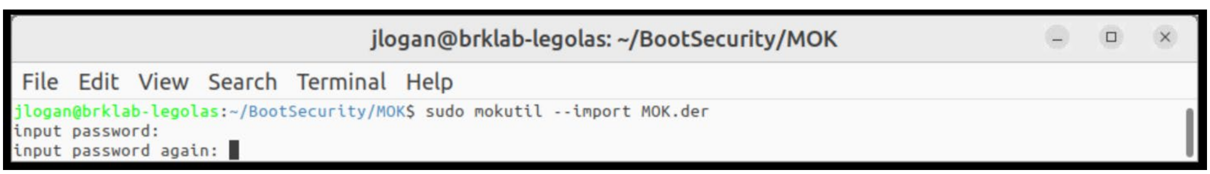


Figure 38. mokutil Password

Now reboot the system with the command 'sudo reboot'

On reboot, the shim will display the MokManager screen:

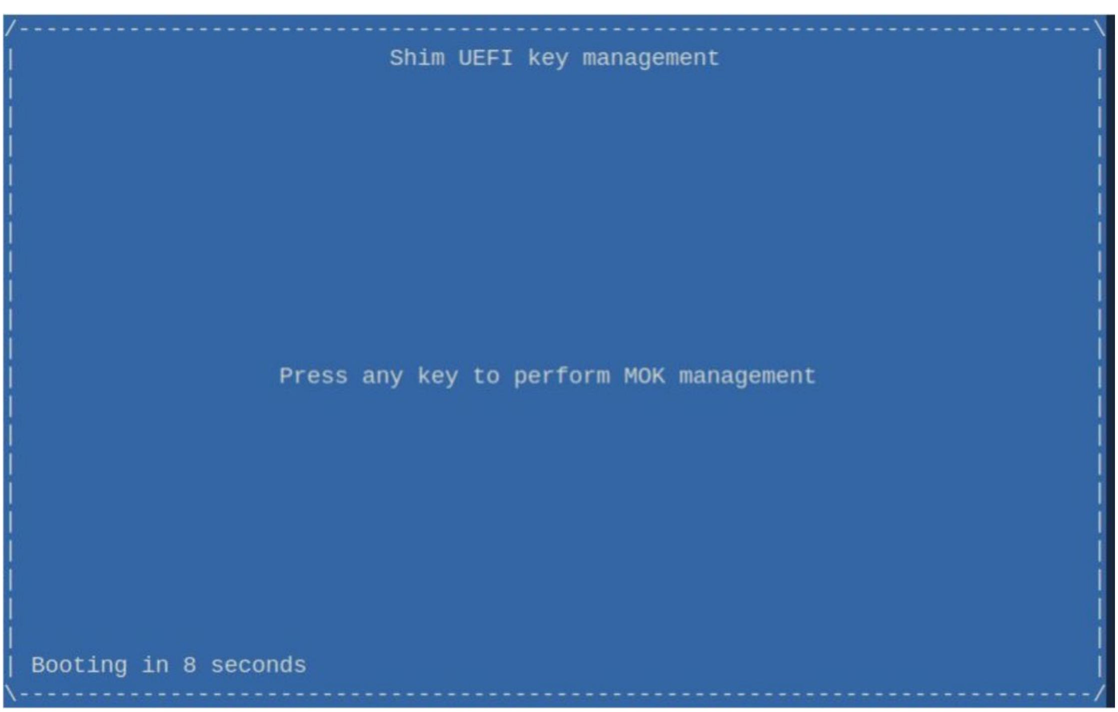


Figure 39 MOKManager

Press any key to enter MokManager. Select option 'Enroll MOK'

Figure 40. Enroll MOK

Selection option 'View key 0' to inspect the certificate. On the author's system, the settings are as shown below:

Figure 41. MOK Info

Hit <return>, then select 'Continue' option. Answer 'Yes' to Enroll the key(s) question:

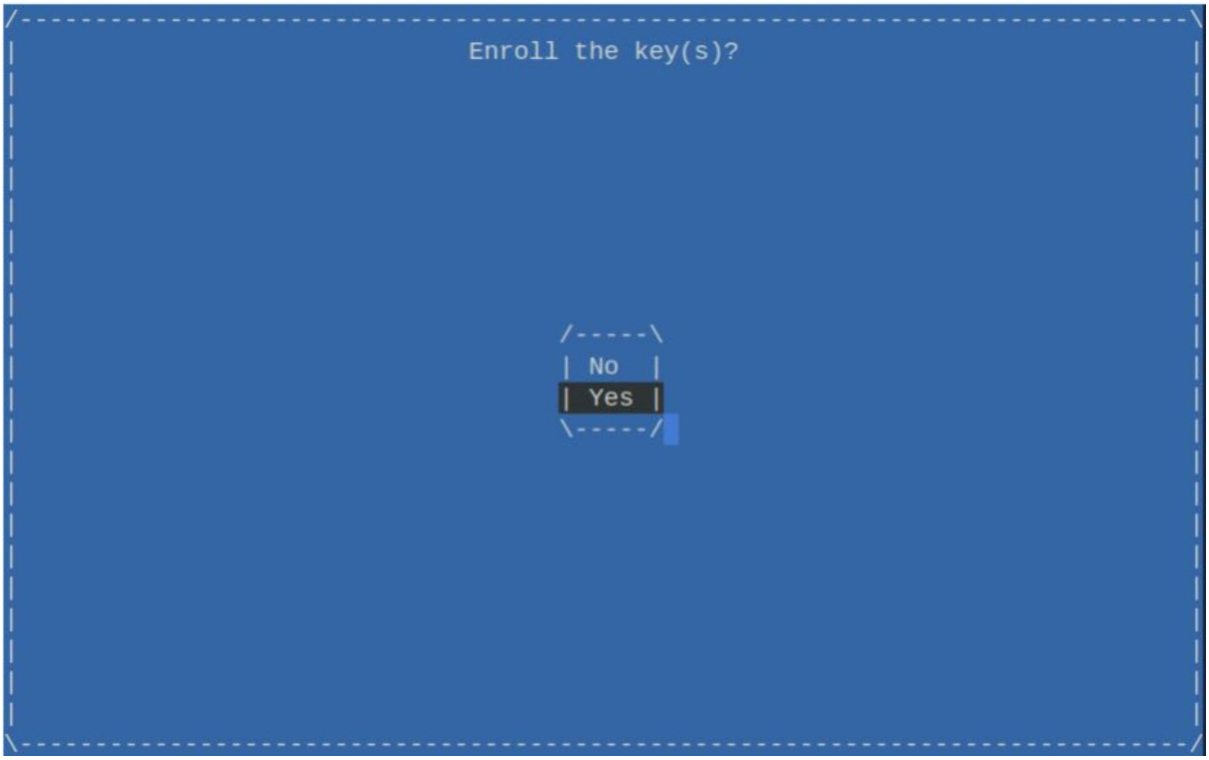


Figure 42. MOK Enrollment

Enter password that was set prior to the reboot:

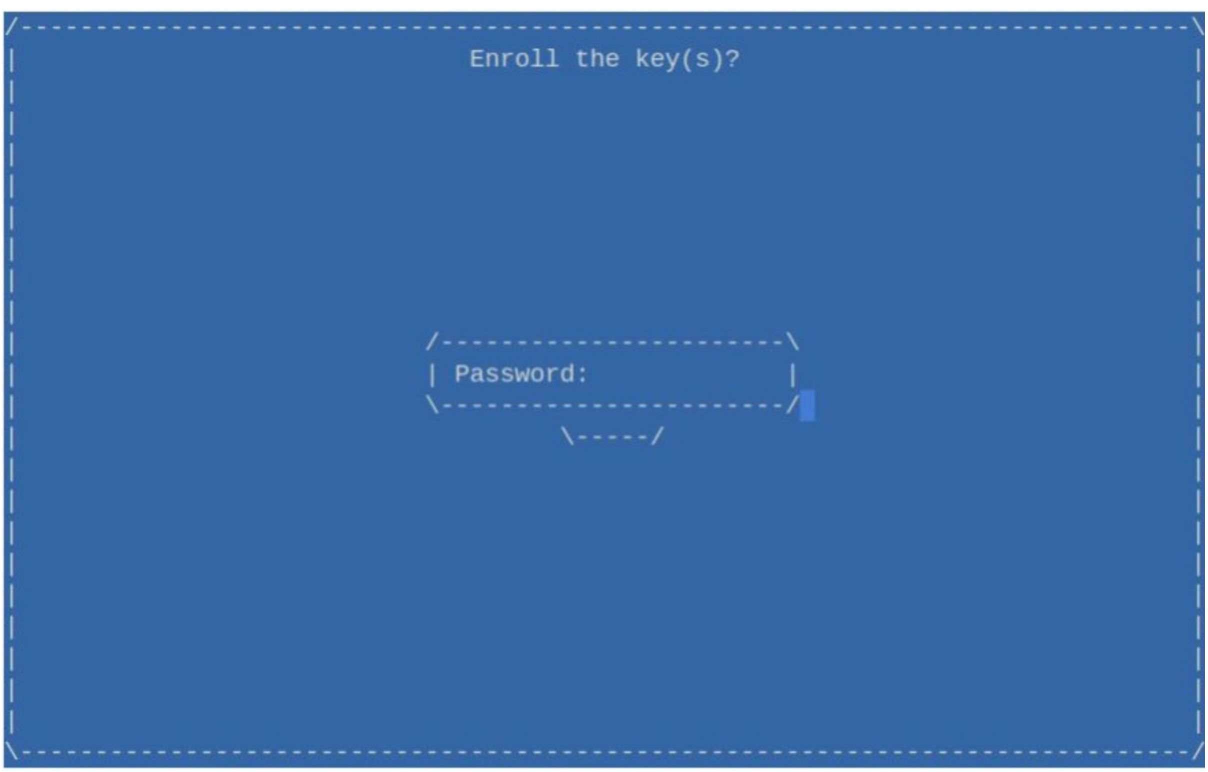


Figure 43. MOK Password Request

Select 'Reboot' to restart system.

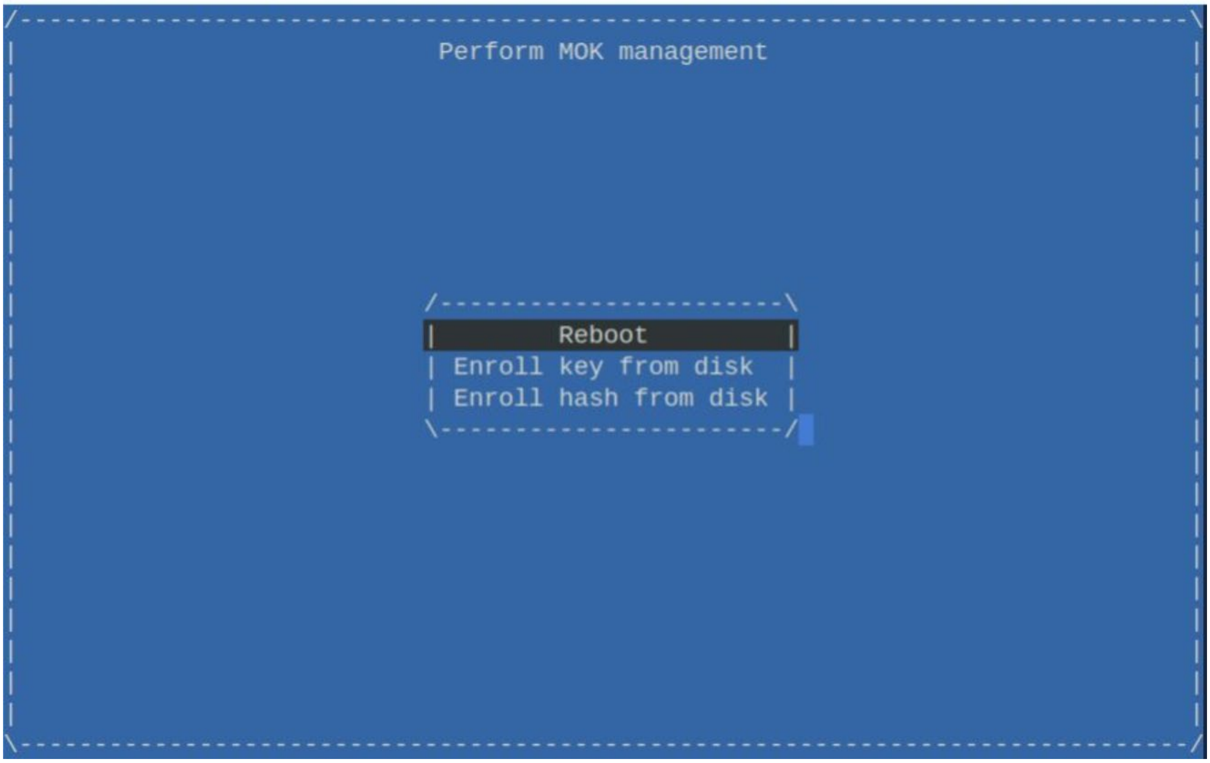


Figure 44. MOK Reboot

Allow the system to reboot to the Linux command line. Use the following command to check if the certificate was enrolled.

- mokutil --list-enrolled

The user created certificate should now be present in addition to the Canonical certificate.

```
jlogan@brklab-legolas: ~
File  Edit  View  Search  Terminal  Help
        71:88:b1:8d:27:a5:2e:a8:f7:3f:5f:80:69:97:3e:a9:f4:99:
        14:db:ce:03:0e:0b:66:c4:1c:6d:bd:b8:27:77:c1:42:94:bd:
        fc:6a:0a:bc

[key 2]
SHA1 Fingerprint: 7b:44:be:7f:10:57:28:d7:14:e6:b2:22:c1:f9:7d:4d:a7:4b:37:f3
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3f:36:0c:47:0f:73:ed:43:3d:bc:8a:6f:f2:66:bc:42:fd:d7:7a:37
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=UK, ST=Scotland, O=Test Org, CN=Secure Boot Signing/emailAddress=example@example.com
        Validity
            Not Before: Oct  5 10:27:38 2023 GMT
            Not After : Oct  2 10:27:38 2033 GMT
        Subject: C=UK, ST=Scotland, O=Test Org, CN=Secure Boot Signing/emailAddress=example@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:87:c6:56:b1:e1:5b:59:37:eb:eb:12:db:5f:7c:
                    2a:f7:7c:71:77:21:ae:d5:36:02:cd:4d:2b:61:6d:
                    26:43:96:b3:79:c0:ba:21:17:4f:7b:e7:e3:3a:22:
                    52:59:36:32:b3:0b:bb:ca:aa:19:b0:0a:21:6e:8e:
                    f7:06:12:e5:38:8b:98:a6:9e:58:1a:e5:5c:76:e0:
                    84:1b:41:0f:06:ac:37:88:02:e3:47:44:6f:03:d4:
                    6e:d9:a7:f8:3b:3d:fa:3d:10:78:47:b5:b9:7f:64:
                    80:95:5f:8c:00:e4:2e:c4:01:63:78:2d:12:67:40:
                    ab:1d:b9:80:bd:b8:5b:02:04:c9:92:08:0c:71:74:
                    fe:ad:e3:e6:47:0b:ed:f4:56:52:4d:89:72:e5:31:
                    2c:1f:90:89:e9:72:2d:c3:b7:45:bf:f0:31:b4:80:
                    f6:99:3e:e4:59:04:76:f0:6b:b0:50:aa:fb:31:b4:
                    10:7f:d1:43:cb:43:d0:6c:04:ce:08:d0:61:e7:83:
                    86:05:a9:d6:89:37:e9:bf:f0:08:41:6d:45:64:ff:
                    95:d9:57:20:5c:64:62:db:70:0e:ef:63:58:b5:b9:
                    f7:d2:c1:d1:40:fd:90:f4:bc:d0:76:64:f0:3c:72:
                    fa:46:2c:a5:13:09:73:64:7b:62:51:cb:ed:e3:eb:
                    3a:ff
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                9A:EE:B2:0D:A4:A9:55:33:01:CE:67:95:E2:A7:F8:13:64:CD:1C:0D
            X509v3 Authority Key Identifier:
                9A:EE:B2:0D:A4:A9:55:33:01:CE:67:95:E2:A7:F8:13:64:CD:1C:0D
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Extended Key Usage:
                Code Signing, 1.3.6.1.4.1.311.10.3.6, 1.3.6.1.4.1.2312.16.1.2
            Netscape Comment:
                OpenSSL Generated Certificate
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        52:e0:76:12:20:aa:93:57:65:84:5a:e1:b5:6b:1d:7b:2b:0c:
        1c:5c:f7:86:5a:58:d2:4f:e4:43:19:e8:83:35:d0:8e:51:5c:
        89:ac:06:b7:a7:8a:de:0a:1e:7c:63:48:8e:8a:3e:63:ca:c5:
        a2:92:36:38:44:75:ed:da:be:59:7d:01:5c:9b:b7:be:6d:90:
        b4:15:d4:1f:d8:a4:f7:a6:08:25:3b:b9:22:e1:3c:d0:c4:93:
        ea:96:13:79:d9:0d:7b:18:a1:76:bc:d7:21:33:8c:51:2f:01:
        af:f5:31:c8:b0:72:de:c5:ef:be:95:53:d5:26:97:37:26:31:
        14:be:86:e4:00:63:76:ab:7a:5a:d1:4f:09:2d:ec:b8:bb:b1:
        9b:38:8b:84:ff:80:86:aa:4a:47:f8:a7:96:c5:eb:24:c2:4b:
        73:bd:3d:f0:16:55:49:e2:0f:7e:a9:5e:ba:62:2e:62:73:8e:
        50:d7:ac:64:19:c6:96:52:7c:97:87:a7:0a:9c:40:f4:5d:32:
        ca:93:52:13:69:cb:b0:44:f0:b9:1e:8e:fe:4a:20:9f:81:02:
        49:ef:d8:f2:c9:93:35:cb:0b:31:41:54:61:59:a8:7a:07:c7:
        0a:9c:3c:35:54:8c:66:14:87:63:c3:52:74:0f:c9:00:4a:16:
        d0:f0:52:8b
jlogan@brklab-legolas:~$
```

Figure 45. Updated MOK List

A.4.7 Creating An Unsigned Kernel Module

In this section, the user will create a simple 'Hello World' kernel module. It will not be signed and will fail to load into the kernel. In the next section, it will be signed using the user-created MOK and will successfully load.

To build a kernel module, install the kernel header files and basic build tools on your system using the following command:

- sudo apt install linux-headers-`uname -r` build-essential

Create a new directory for the kernel module development and enter the folder:

- mkdir test_module
- cd test_module

Create a file called 'hello_world_mod.c' containing the following example code:

```
/* Simple hello world module for secure boot signing test  */

#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("<your name>");
MODULE_DESCRIPTION("Hello World module");


static int __init hello_world_mod_init(void)
{
        printk(KERN_ALERT "Hello World! Loading kernel module \n");
        return 0;
}

static void __exit hello_world_mod_exit(void)
{
        printk(KERN_ALERT "Goodbye, removing module from kernel \n");
}

module_init(hello_world_mod_init);
module_exit(hello_world_mod_exit);
```

Create a file called 'Makefile' containing the following example code:

```
obj-m += hello_world_mod.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Execute the following command to build the kernel module:

- make

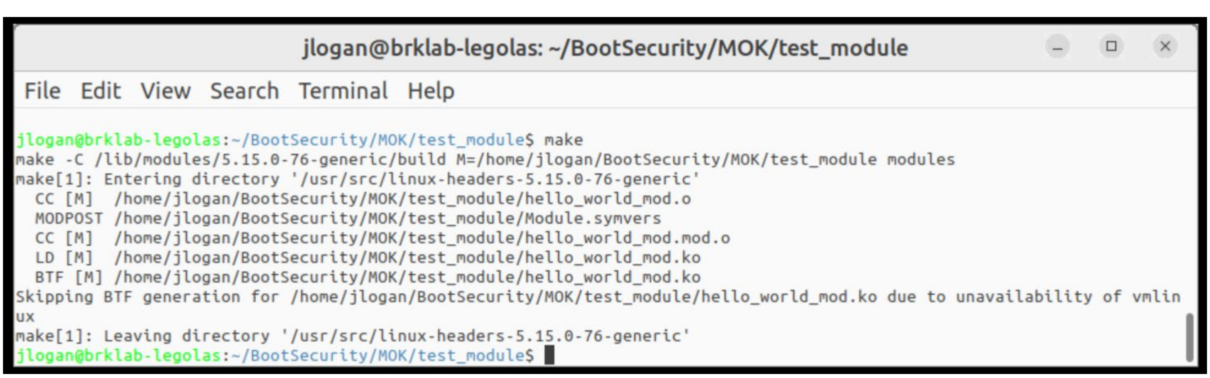


Figure 46. Hello World Module Build

This will create a kernel module 'hello_world_mod.ko' This is a very simple module that will print a message to the kernel log when it is inserted or removed from the kernel.

Attempt to insert the module into the kernel with the following command:

- sudo insmod hello_world_mod.ko

The above command will fail with a message shown below:

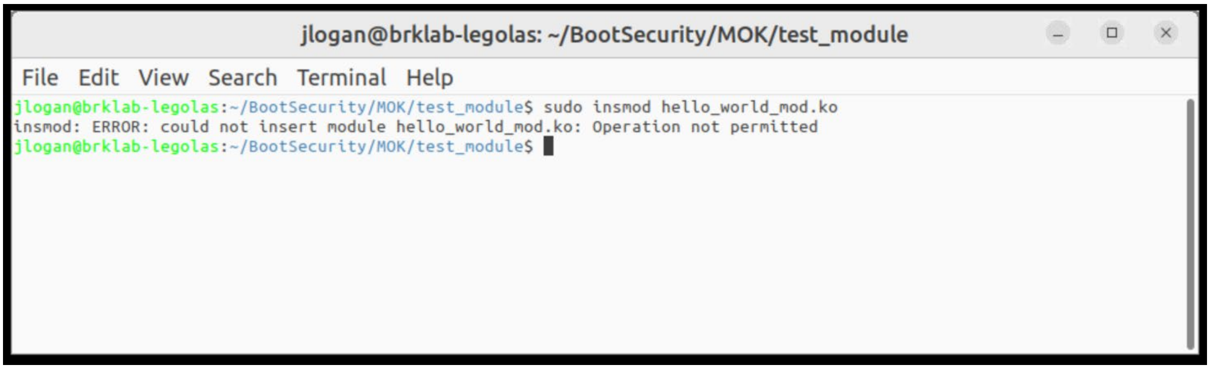


Figure 47. Failed Module Insertion

This error is expected. Secure Boot is enabled, and the module is not signed. Examine the end of the kernel log using command:

- sudo dmesg | tail

It should indicate that an unsecure module load was attempted:

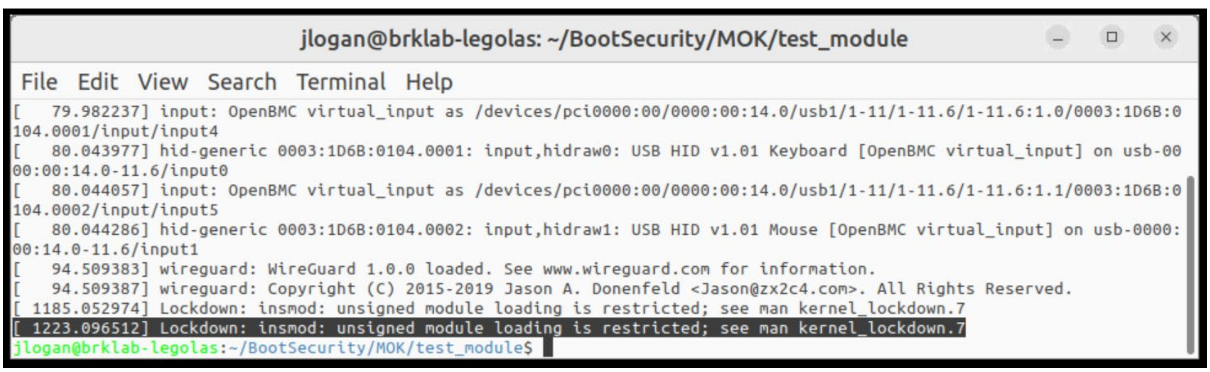


Figure 48. Failed Module Load dmesg Info

A.4.8 Sign Kernel Module With MOK

In this section, the Hello World module created in the previous section will be signed and loaded into the Linux kernel.

To check if the kernel module has been signed, use the following command:

- modinfo ./hello_world_mod.ko

Note that no signature info is shown in the output.

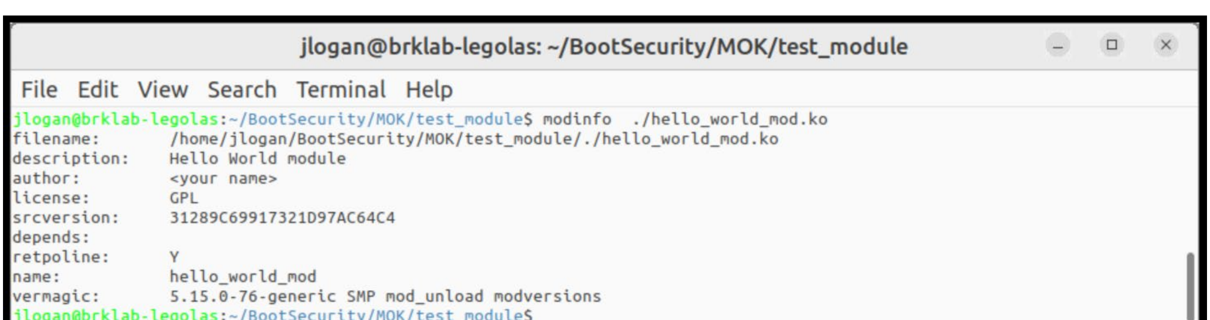


Figure 49. Hello World Modinfo

Also, check the end of the module binary using xxd tool with command:

- xxd hello_world_mod.ko

Note that there is no signature info at the end of the binary.

Figure 50. Hello World xxd Info

To sign the module with the user-created MOK, a tool called ‘kmodsign’ is required. This is part of the sbsigntool package, which includes other signing tools for the kernel, etc.

To install the sbsigntool package, use the command:

- sudo apt install sbsigntool

To sign the kernel module, use the command

- kmodsign sha512 MOK.priv MOK.der hello_world_mod.ko hello_world_mod_signed.ko

(Note: you need to enter paths to your MOK.priv and MOK.der files if they are not in the same directory as the kernel module)

Figure 51. kmodsign Command

The above will produce a signed version of the kernel module. Check this version for signing info using the command:

- modinfo ./hello_world_mod_signed.ko

Note that signature info is now shown:

```
jlogan@brklab-legolas: ~/BootSecurity/MOK/test_module
File  Edit  View  Search  Terminal  Help
jlogan@brklab-legolas:~/BootSecurity/MOK/test_module$ modinfo ./hello_world_mod_signed.ko
filename:       /home/jlogan/BootSecurity/MOK/test_module/./hello_world_mod_signed.ko
description:    Hello World module
author:         <your name>
license:        GPL
srcversion:     31289C69917321D97AC64C4
depends:
retpoline:      Y
name:           hello_world_mod
vermagic:       5.15.0-76-generic SMP mod_unload modversions
sig_id:         PKCS#7
signer:         Secure Boot Signing
sig_key:        3F:36:0C:47:0F:73:ED:43:3D:BC:8A:6F:F2:66:BC:42:FD:D7:7A:37
sig_hashalgo:   sha512
signature:      10:7E:E9:BE:66:78:CE:F9:C2:EB:94:D9:B9:36:05:42:8B:68:00:6B:
                A1:E7:63:9D:12:A4:64:F2:5B:8A:05:88:5A:DE:80:21:90:A3:94:1C:
                2C:4B:FC:10:82:48:A5:CC:0F:BD:27:86:1A:01:FF:0B:28:4A:58:D0:
                34:BD:A1:B1:25:D7:16:38:D4:23:EA:DB:E4:A9:11:A3:1D:B0:AC:B8:
                42:D9:DB:FE:4A:BD:E1:61:E2:02:A5:1E:1F:8B:CD:B0:40:10:BE:AE:
                09:08:69:12:FF:4B:FD:DC:8D:D5:F5:CF:0F:07:5F:DA:A7:2F:68:D4:
                9B:C9:9E:EE:4E:81:8D:74:30:C7:F3:A3:B5:9E:72:DB:0C:E9:3F:31:
                5F:E8:CB:E5:57:06:53:41:6A:FC:33:5C:D1:14:03:18:71:65:7D:B2:
                CC:15:38:E9:BD:1E:30:33:7C:2A:20:3C:8C:62:9A:C0:6A:15:0A:41:
                16:F4:DF:9A:D4:4B:E2:F9:38:7F:3C:90:97:BF:AD:B0:FA:D0:4E:4F:
                A9:54:B1:AC:9C:51:82:2B:0F:C1:B4:CC:CF:3E:A3:56:B5:64:A8:A0:
                03:7C:DC:3A:DC:9E:3E:D8:42:5D:E4:59:D5:1B:DA:BE:FB:31:E0:A3:
                A2:BB:F3:0E:35:DB:4C:77:7F:1C:0A:F2:2F:1B:9C:64
jlogan@brklab-legolas:~/BootSecurity/MOK/test_module$
```

Figure 52. Hello World Signature

Check the end of the hello_world_mod_signed.ko binary for signature info with the command:

xxd ./hello_world_mod_signed.ko

Note that signature info has been appended:

```
jlogan@brklab-legolas: ~/BootSecurity/MOK/test_module
File  Edit  View  Search  Terminal  Help
0000f430: 0854 6573 7420 4f72 6731 1c30 1a06 0355  .Test Org1.0...U
0000f440: 0403 0c13 5365 6375 7265 2042 6f6f 7420  ....Secure Boot
0000f450: 5369 676e 696e 6731 2230 2006 092a 8648  Signing1"0 ..*.H
0000f460: 86f7 0d01 0901 1613 6578 616d 706c 6540  ........example@
0000f470: 6578 616d 706c 652e 636f 6d02 143f 360c  example.com..?6.
0000f480: 470f 73ed 433d bc8a 6ff2 66bc 42fd d77a  G.s.C=..o.f.B..z
0000f490: 3730 0b06 0960 8648 0165 0304 0203 300d  70...`.H.e....0.
0000f4a0: 0609 2a86 4886 f70d 0101 0105 0004 8201  ..*.H...........
0000f4b0: 0010 7ee9 be66 78ce f9c2 eb94 d9b9 3605  ..~..fx.......6.
0000f4c0: 428b 6800 6ba1 e763 9d12 a464 f25b 8a05  B.h.k..c...d.[..
0000f4d0: 885a de80 2190 a394 1c2c 4bfc 1082 48a5  .Z..!....,K...H.
0000f4e0: cc0f bd27 861a 01ff 0b28 4a58 d034 bda1  ...'.....(JX.4..
0000f4f0: b125 d716 38d4 23ea dbe4 a911 a31d b0ac  .%..8.#.........
0000f500: b842 d9db fe4a bde1 61e2 02a5 1e1f 8bcd  .B...J..a.......
0000f510: b040 10be ae09 0869 12ff 4bfd dc8d d5f5  .@.....i..K.....
0000f520: cf0f 075f daa7 2f68 d49b c99e ee4e 818d  ..._../h.....N..
0000f530: 7430 c7f3 a3b5 9e72 db0c e93f 315f e8cb  t0.....r...?1_..
0000f540: e557 0653 416a fc33 5cd1 1403 1871 657d  .W.SAj.3\....qe}
0000f550: b2cc 1538 e9bd 1e30 337c 2a20 3c8c 629a  ...8...03|* <.b.
0000f560: c06a 150a 4116 f4df 9ad4 4be2 f938 7f3c  .j..A.....K..8.<
0000f570: 9097 bfad b0fa d04e 4fa9 54b1 ac9c 5182  .......NO.T...Q.
0000f580: 2b0f c1b4 cccf 3ea3 56b5 64a8 a003 7cdc  +.....>.V.d...|.
0000f590: 3adc 9e3e d842 5de4 59d5 1bda befb 31e0  :..>.B].Y.....1.
0000f5a0: a3a2 bbf3 0e35 db4c 777f 1c0a f22f 1b9c  .....5.Lw..../..
0000f5b0: 6400 0002 0000 0000 0000 0001 f17e 4d6f  d............~Mo
0000f5c0: 6475 6c65 2073 6967 6e61 7475 7265 2061  dule signature a
0000f5d0: 7070 656e 6465 647e 0a                   ppended~.
jlogan@brklab-legolas:~/BootSecurity/MOK/test_module$
```

Figure 53. Signed Hello World xxd Info

Insert the signed kernel module into the Linux kernel with the command:

- sudo insmod ./hello_world_mod_signed.ko

Check the end of the kernel log with the command:

- sudo dmesg | tail

It should now show successful insertion:

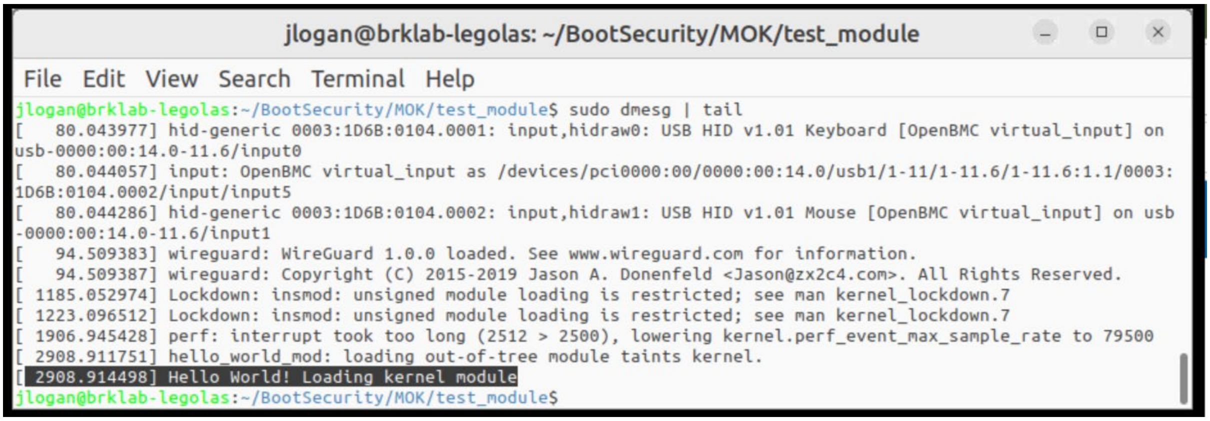


Figure 54. Signed Hello World dmesg Info

Remove the kernel module from the kernel with command:

- sudo rmmod hello_world_mod

Recheck the end of the kernel log. It should show the exit message.

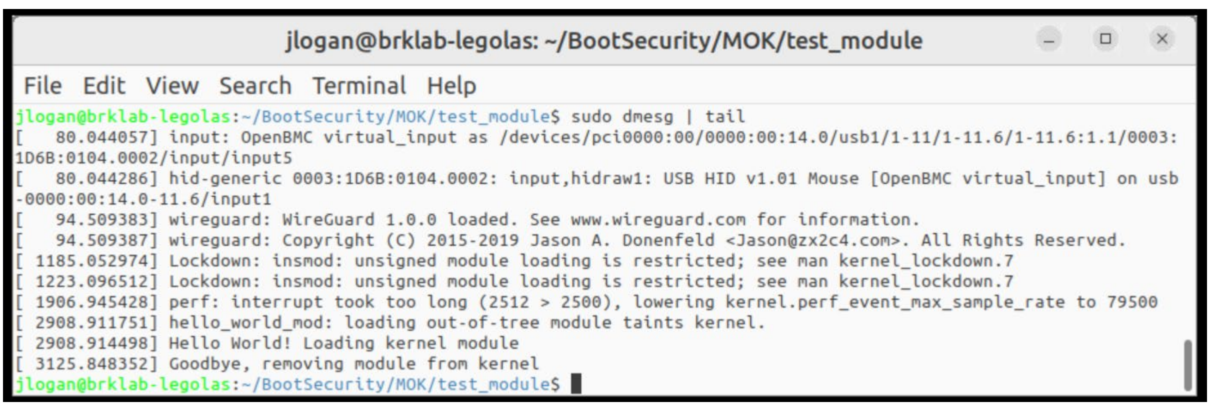


Figure 55. Hello World Module Removal

Appendix B BPM Example Definition File

```
# FILEHEADER
FileID:       _BPMDEF_
FileVersion: 1
ToolVersion: 7
ToolDate:     20191203
FileDate:     20200113
//
# BPM_DEF
PlatformRules:      Server
BpmStrutVersion:    0x22
BpmRevision: 0
BpmRevocation:      0
AcmRevocation:      0
NEMPages:     0x40
IbbSetCount: 2
CurrentIbbSet:      0
//
# IBB_SET
```

```
IbbSetType:  0:ColdBoot
IbbSetInclude:      TRUE
PBETValue:   0xF
MCHBAR:      0x00000000FED10000
VTD_BAR:     0x00000000FED90000
DmaProtBase0:0x0
DmaProtLimit0:      0x0
DmaProtBase1:0x0
DmaProtLimit1:      0x0
IbbFlags:    0x3
//    Bit0 : Enable DMA Protection;
//    Bit1 : Issue TPM Start-up from Locality 3;
//    Bit2 : Extend Authority Measurements into the Authority PCR;
//    Bit3 : On error: Leave TPM Hierarchies enabled. Cap all PCRs;
//    Bit4 : Top Swap Supported;
//    Bit5 : Force (MK)TME;
//    Bit6 : Enforce SPIRAL specification between BIOS/CSE
//    Bit7 : SRTM Attenstation Control
//    Bit8 : Force Communication NEM Buffer
DmaProtAutoCalc:    FALSE
IbbHashAlgID:0x0C
IbbEntry:    0xFFFFFFF0
PostIbbHashAlgID:   0x10
PostIBBHashSource:  Calculate
PostIbbHashFile:
IbbSegSource:FIT
IbbSegFile:
IbbGuid:     4a4ca1c6-871c-45bb-8801-6910a7aa5807
ObbHashAlgID:0x0C
ObbFullFvHash:      TRUE
ObbHashSource:      List
ObbHashFile:
ObbGuid:     27a72e80-3118-4c0c-8673-aa5b4efa9613: FVMAIN_COMPACT
ObbGuid:     013b9639-d6d5-410f-b7a9-f9173c56ecda: PI_SMM_COMPACT
ObbGuid:     b4705b4b-0be6-4bdb-a83a-51cad2345cea: FvPostMemory
ObbGuid:     1638673d-efe6-400b-951f-abac2cb31c60: FSP-S
ObbGuid:     5a515240-d1f1-4c58-9590-27b1f0e86827: OEM_FV
//
# IBB_SET
IbbSetType:  1:S3Resume
IbbSetInclude:      TRUE
PBETValue:   0xF
MCHBAR:      0x00000000FED10000
VTD_BAR:     0x00000000FED90000
DmaProtBase0:0x0
DmaProtLimit0:      0x0
DmaProtBase1:0x0
DmaProtLimit1:      0x0
IbbFlags:    0x3
//    Bit0 : Enable DMA Protection;
//    Bit1 : Issue TPM Start-up from Locality 3;
//    Bit2 : Extend Authority Measurements into the Authority PCR;
//    Bit3 : On error: Leave TPM Hierarchies enabled. Cap all PCRs;
//    Bit4 : Top Swap Supported;
//    Bit5 : Force (MK)TME;
//    Bit6 : Enforce SPIRAL specification between BIOS/CSE
//    Bit7 : SRTM Attenstation Control
//    Bit8 : Force Communication NEM Buffer
```

```
DmaProtAutoCalc:    FALSE
IbbHashAlgID:0x0C
IbbEntry:    0xFFFFFFF0
PostIbbHashAlgID:   0x10
PostIBBHashSource:  Calculate
PostIbbHashFile:
IbbSegSource:FIT
IbbSegFile:
IbbGuid:     4a4ca1c6-871c-45bb-8801-6910a7aa5807
ObbHashAlgID:0x0C
ObbFullFvHash:      TRUE
ObbHashSource:      List
ObbHashFile:
ObbGuid:     27a72e80-3118-4c0c-8673-aa5b4efa9613: FVMAIN_COMPACT
ObbGuid:     013b9639-d6d5-410f-b7a9-f9173c56ecda: PI_SMM_COMPACT
//
# TXT_ELEMENT
TxtInclude:  TRUE
MinSvn:      0x0
TxtFlags:    0x0
// [4:0] = TXT execution profile
//     00000b – Use Default based on HW
//     00001b - Server Profile
//     00010b - Client Profile
// [6:5] = “Memory scrubbing” policy
//     00b - Trust Verified BIOS
//     01b - Trust Any BIOS
//     10b - Trust No BIOS
// [8:7] = Backup Policy
//     00b - Default
//     01b - Power Down
//     10b - Unbreakable Shutdown
//     11b - PFR Recovery
// [31] = Reset AUX control (1=AUX Reset leaf will delete AUX Index)
//MemoryDepletion Power Down
StrideSize:  0x0
AcpiBase:    0x400
PwrmBase:    0xFE000000
PdUseDefault:TRUE
PdMinutes:   5
PdSeconds:   0
PttCmosOffset0:     0x7E
PttCmosOffset1:     0x7F
//TXTE Segments
TxtSegSource:IBB
TxtSegGuid:  4a4ca1c6-871c-45bb-8801-6910a7aa5807
TxtSegHashAlgID:    0x10
//
# PLATFORM_CONFIG_ELEMENT
PcdInclude:  TRUE
PdReqLocation:      TPM
//    Power down request location for CMOS
CmosIndexRegister:  0x70
CmosDataRegister:   0x71
CmosIndexOffset:    125
CmosBitFieldWidth:  3
CmosBitFieldPosition:     0
//
```

```
# TPM1.2_LOCATION
TpmIndexHandle:     0x50000004
TpmByteOffset:      7
TpmBitFieldWidth:   3
TpmBitFieldPosition:      0
//
# TPM2.0_LOCATION
TpmIndexHandle:     0x1C10104
TpmByteOffset:      7
TpmBitFieldWidth:   3
TpmBitFieldPosition:      0
//
# PTT_LOCATION
TpmIndexHandle:     0x1C10104
TpmByteOffset:      7
TpmBitFieldWidth:   3
TpmBitFieldPosition:      0
//
# COMMUNICATION_NEM_BUFFER
CnbsInclude: False
CnbsBase:    0xFF000000
CnbsSize:    0x1000
//
# PLATFORM_MANUFACTURERS_ELEMENT
PmdeInclude: FALSE
PmdeFile:
//
# PLATFORM_FIRMWARE_RESILIENCY_ELEMENT
PfrsInclude: TRUE
PfrsControlFlags:   0x00000000
PfrsCpldSmbusAddr:  0xE0
PfrsPchActiveOffset:      0x2FF0000
PfrsPchRecoveryOffset:    0x1BF0000
PfrsPchStagingOffset:     0x7F0000
PfrsBmcActiveOffset:      0x80000
PfrsBmcRecoveryOffset:    0x2A00000
PfrsBmcStagingOffset:     0x4A00000
//
# BPM_SIGNATURE
BpmSigSource:Internal
BpmSigHashAlgID:    0x0C
BpmSigKeyType:      0x01
BpmSigScheme:0x16
BpmKeySizeBits:     3072
BpmSigPubKey:BPM_3K_pub.pem
BpmSigPrivKey:      BPM_3K_priv.pem
BpmSigBatch: X-Sign.bat SHA384 Rsa3072
BpmSigData:  Data2Sign.bin
BpmSigDataType:     BPM
BpmSigXSig:  X-Sig.bin
//
#EOF
```

Reference Documentation

The *Network and Edge Container Bare Metal Reference System Architecture User Guide* provides information and a full set of installation instructions for a BMRA.

The *Network and Edge Reference System Architectures Portfolio User Manual* provides additional information for the Reference System including a complete list of reference documents.

The Intel FlexRAN™ Docker hub provides additional information on running the FlexRAN™ software in a POD.

Other collaterals, including technical guides and solution briefs that explain in detail the technologies enabled in the Reference Systems are available in the following locations: Network & Edge Platform Experience Kits.

Document Revision History

REVISION	DATE	DESCRIPTION
001	July 2023	Initial release.
002	October 2023	Updated BMRA version to 23.10. Added UEFI Secure Boot process and support for ECDSA keys with SHA384 for 5G RAN security with NETCONF server-client authentication with Intel SGX.

1023/DN/WIT/PDF 784333-002US