USER GUIDE

Intel Corporation

Intel® Software Guard Extensions (Intel® SGX) - Key Management Reference Application (KMRA) on 3rd and 4th Gen Intel® Xeon® Scalable Processors

Authors

Veronika Karpenko

Radoslaw Jablonski

David Lu

Jon Strang

Kapil Sood

Seosamh O'Riordain

Darragh Coen

Kamil Lorek

Abhijit Sinha

Chenxi Wang

1 Introduction

Key Management Reference Application (KMRA) is a proof-of-concept software created to demonstrate the integration of Intel® Software Guard Extensions (Intel® SGX) asymmetric key capability with a hardware security model (HSM) on a centralized key server. The goal of this document is to outline the steps to set up a NGINX* workload to access the private key in an Intel® SGX enclave on 3rd and 4th Gen Intel® Xeon® Scalable processors, using the Public-Key Cryptography Standard (PKCS) #11 interface and OpenSSL.

Intel® SGX provides a more secure environment for application owners to run their applications' sensitive code and data inside an Intel SGX enclave, enhancing protection of their enclave code and data from privileged software and applications.

The Crypto API Toolkit for Intel® Software Guard Extensions (Crypto API Toolkit for Intel® SGX) is an SDK for using the cryptographic capabilities within an Intel SGX. It aims to enhance the security of ISVs' and OEMs' data protection applications by exposing enhanced and optimized interfaces that run the cryptographic operations more securely within Intel SGX.

RSA keypairs can be generated into tokens where each token is stored in an Intel SGX enclave. The private key object can only be used to perform cryptographic operations with the correct credentials, without leaving the Intel SGX enclave. This provides more security for the private key and prevents it from being exposed and compromised.

Quote generation and verification libraries from Intel® Software Guard Extensions Data Center Attestation Primitives (Intel® SGX DCAP) are used to attest an Intel SGX platform. The KMRA client generates an Intel SGX quote using the Crypto API Toolkit for Intel SGX. The KMRA server verifies the quote before wrapping and extracting the encrypted keys from the HSM for use inside the compute server's Intel SGX enclave.

This document details the setup of each component, such as Intel SGX, NGINX, Crypto API Toolkit for Intel SGX, PKCS#11 engine, and OpenSSL. Each section contains the necessary commands and instructions to configure and install the component using Ansible scripts. Reference documents and links to source code are provided for more information outside the scope of this document.

KMRA can be deployed in multiple ways by ansible scripts, Docker containers, and VMware vSphere. To install the KMRA application with the ansible scripts, follow the instructions provided in Section 7. To deploy KMRA containers using Dockerfiles, follow the instructions provided in Section 5.7. To deploy KMRA on virtual machine using VMware vSphere, ensure that ESXi 7.0 or 8.0 is used and follow the instructions provided in Section 9.

This document is targeted at development engineers, validation teams, benchmarking teams, and application engineers who are interested in configuring NGINX on a platform to use Intel SGX enclaves to help secure the private key.

KMRA is not meant to be used in a production environment.

This document is part of the Network & Edge Platform Experience Kits.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	February 2021	Initial release.
002	April 2021	Revised the document for public release to Intel® Network Builders.
003	July 2021	Added support for Red Hat Enterprise Linux (RHEL), Intel SGX 2.13.3, Intel SGX DCAP 1.10.3, SGX upstream kernel, containers, and other improvements. Added sections for Intel Provisioning Certificate Caching Service (PCCS) installation and for KMRA Docker container deployment.
004	August 2021	Added new steps to deploy the Dockerfiles and run the KMRA Ansible scripts. Added new information for installing and deploying the PCCS and Ctk_loadkey containers. Also added a Common Issues section for the containers. Updated Linux drivers, NGINX, and component versions.
005	March 2022	Upgraded to SGX 2.15.1 and DCAP 1.12.1; Docker container improvements.
006	July 2022	Updates in deployment via Dockerfiles and Platform Registration and Attestation. Added new NGINX Key Management with Intel SGX Enclave Architecture diagram. Added VM deployment and testing section and Appendix. Upgraded to Intel SGX 2.16 and DCAP 1.13.
007	March 2023	Updates include: Ubuntu 22.04 host support, Upgrade to SGX 2.18.1 and DCAP 1.15, CTK restart bug fix

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
Ansible	Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration.
Ansible-playbook	Playbooks are the files where Ansible code is written.
BIOS	Basic Input/Output System is a set of computer instructions in firmware that controls input and output operations.
CA	Certificate authority
CDN	Content Delivery Network is a system of distributed servers (network). It delivers pages and other web content to a user, based on the geographic location of the user, the origin of the webpage, and the content delivery server.
DCAP	Data Center Attestation Primitives. Intel® Software Guard Extensions Data Center Attestation Primitives (Intel® SGX DCAP) provides SGX attestation support targeted for data centers, cloud services providers, and enterprises.
ECDSA	Elliptic curve digital signature algorithm
FLC	Flexible launch control
GID	Group ID
GPL	General public license
HSM	Hardware security module
KMRA	Key Management Reference Application (KMRA)
LGPL	Lesser general public license
mTLS	Mutual transport layer security
OS	Operating system
PCCS	Provisioning Certificate Caching Service
PKCS	Public-Key Cryptography Standard
PKCS#11	Public-Key Cryptography Standard. The PKCS#11 standard defines a platform-independent API to cryptographic tokens, such as hardware security modules (HSM) and smart cards.
PSW	Platform software
RHEL	Red Hat Enterprise Linux
RSA	RSA is a public-key cryptosystem that is widely used for secure data transmission. The acronym RSA comes from the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who publicly described the algorithm in 1977. (Wikipedia)
SGX	Intel® Software Guard Extensions (Intel® SGX) is a set of instructions that increases the security of application code and data, giving them more protection from disclosure or modification.
SSL	Secure Sockets Layer is a networking protocol designed for securing connections between web clients and web servers over an insecure network, such as the internet.
TLS	Transport Layer Security
VMware	Virtualization and cloud computing software
vSphere	VMware's cloud computing virtualization platform. It includes an updated vCenter Configuration Manager, as well as vCenter Application Discovery Manager, and the ability of vMotion to move more than one virtual machine at a time from one host server to another.
ESXi	VMware ESXi is a bare metal hypervisor that installs easily on to a server and partitions it into multiple virtual machines.

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
Intel® SGX Binaries and Installation Instructions	https://01.org/intel-software-guard-extensions/downloads

REFERENCE	SOURCE
Intel® SGX Programming Reference and SDK for Linux	https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html#combined https://download.01.org/intel-sgx/latest/linux-latest/docs/ https://github.com/intel/linux-sgx
PKCS#11 Specification	http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html
ETSI NFV Security Standards (SEC001, SEC012, SEC013, others)	http://www.etsi.org/technologies-clusters/technologies/nfv
Intel® SGX Resources	https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html https://www.intel.com/content/www/us/en/develop/download/intel-software-guard-extensions-intel-sgx-developer-guide.html https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
Intel® SGX Crypto - Toolkit Open Source	https://github.com/intel/crypto-api-toolkit
Intel® SGX ECDSA Attestation DCAP and APIs	https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/ https://github.com/cloud-security-research/sgx-ra-tls https://github.com/intel/SGXDataCenterAttestationPrimitives
Intel® SGX Flexible Launch Control (FLC)	https://github.com/intel/linux-sgx/blob/master/psw/ae/ref_le/ref_le.md https://www.intel.com/content/www/us/en/developer/articles/technical/an-update-on-3rd-party-attestation.html
Intel® SGX Open Source Projects	https://github.com/intel/intel-sgx-ssl https://github.com/intel/sgx-ra-sample https://github.com/oscarlab/graphene
Intel® SGX Security Analysis	https://www.intel.com/content/www/us/en/security-center/default.html https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/overview.html
Intel® Software Guard Extensions (Intel® SGX) Platform Enablement and Validation Requirements for Intel® Xeon® SP	https://cdrdv2.intel.com/v1/dl/getContent/611589
Ubiquitous Availability of Crypto Technologies Solution Brief	https://networkbuilders.intel.com/solutionslibrary/crypto-ubiquitous-availability-of-crypto-technologies-solution-brief
Intel® Software Guard Extensions (Intel® SGX) – Key Management Reference Application (KMRA) on Intel® Xeon® Processors Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-sgx-kmra-on-intel-xeon-processors-technology-guide
Red Hat Enterprise Linux Download for Development Use	https://developers.redhat.com/products/rhel/download
VMware vSphere	https://docs.vmware.com/en/VMware-vSphere/index.html
VMware ESXi	https://www.vmware.com/products/esxi-and-esx.html

2 Solution Overview

2.1 NGINX Key Management Architecture Flow with Intel® SGX

The key management architecture shown in Figure 1 enables NGINX applications to help protect the private key inside an Intel SGX enclave. This architecture demonstrates the integration of Intel SGX with the NGINX application, with SGX Enclave Attestation and Key Server. Readers may refer to the *Intel® Software Guard Extensions (Intel® SGX) – Key Management Reference Application (KMRA) on Intel® Xeon® Processors Technology Guide* (see Reference Documentation) for architecture and software design details.

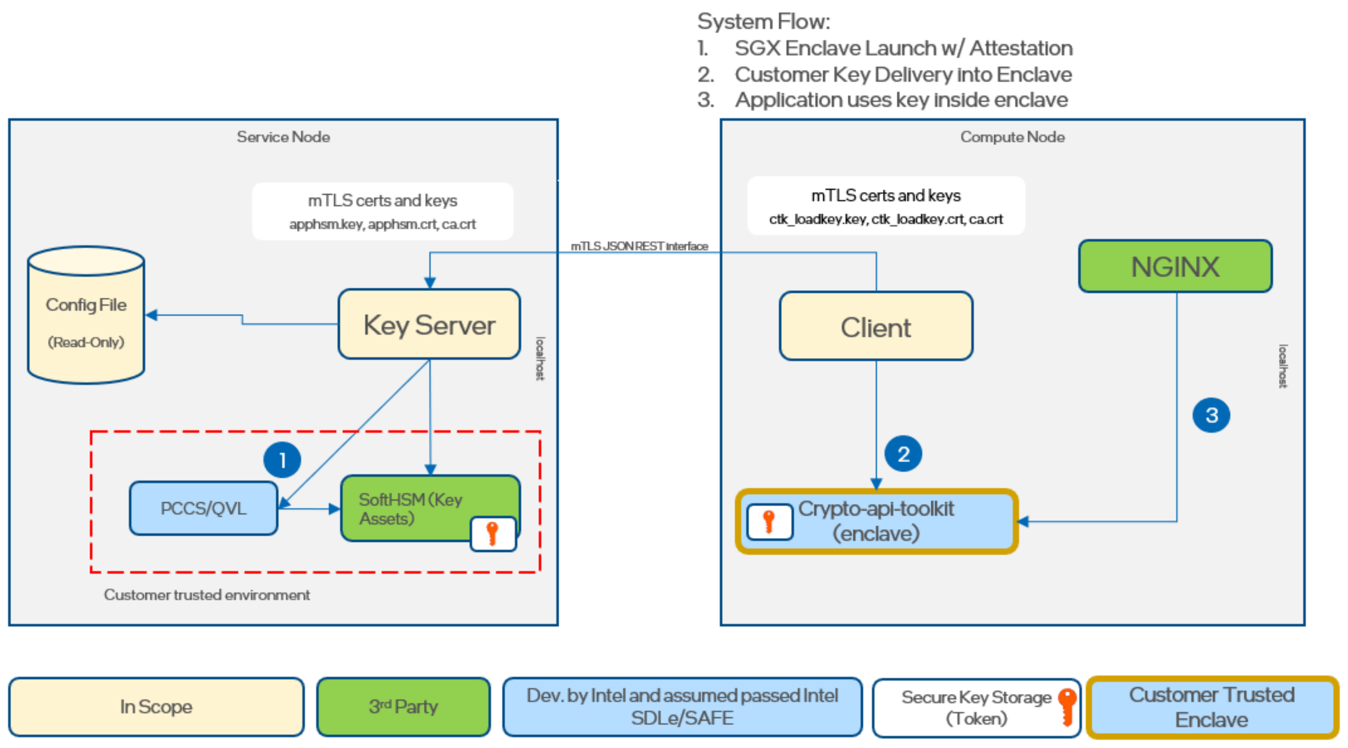

System Flow:
1. SGX Enclave Launch w/ Attestation
2. Customer Key Delivery into Enclave
3. Application uses key inside enclave
Service Node
mTLS certs and keys
apphsm.key, apphsm.crt, ca.crt
Config File (Read-Only)
Key Server
localhost
1
PCCS/QVL
SoftHSM (Key Assets)
Customer trusted environment
mTLS JSON REST interface
Compute Node
mTLS certs and keys
ctk_loadkey.key, ctk_loadkey.crt, ca.crt
NGINX
Client
localhost
2
3
Crypto-api-toolkit (enclave)
In Scope
3rd Party
Dev. by Intel and assumed passed Intel SDLe/SAFE
Secure Key Storage (Token)
Customer Trusted Enclave

Figure 1. NGINX Key Management with Intel SGX Enclave Architecture

As shown in Figure 1, the key management flow of NGINX with Intel SGX has three main steps.

2.1.1 Step 1 - SGX Enclave Launch with DCAP Attestation

A compute node has Intel SGX enabled and Crypto API Toolkit for Intel SGX installed. An Intel SGX quote is generated inside the Crypto API Toolkit for Intel SGX Enclave for DCAP attestation. The Intel SGX quote is attested on the key server side.

2.1.2 Step 2 - Customer Key Delivery into Enclave

The wrapped private key is provisioned by the key server into the Crypto API Toolkit for Intel® SGX enclave.

2.1.3 Step 3 - NGINX Application Uses the Key Protected Inside the Enclave

The NGINX workload can more securely access the private key through the PKCS#11 interface using the libp11 engine configured with OpenSSL. NGINX can establish a transport layer security (TLS) connection using the private key from the Crypto API Toolkit for Intel SGX enclave.

2.2 KMRA Software Design

Key Management Reference Application (KMRA) is a proof-of-concept software created to demonstrate the integration of the asymmetric key capability of Intel® SGX with a third-party hardware security model (HSM) on a centralized key server. Figure 2 shows the KMRA NGINX/Intel SGX key management software design. For a complete list of components, see Table 3.

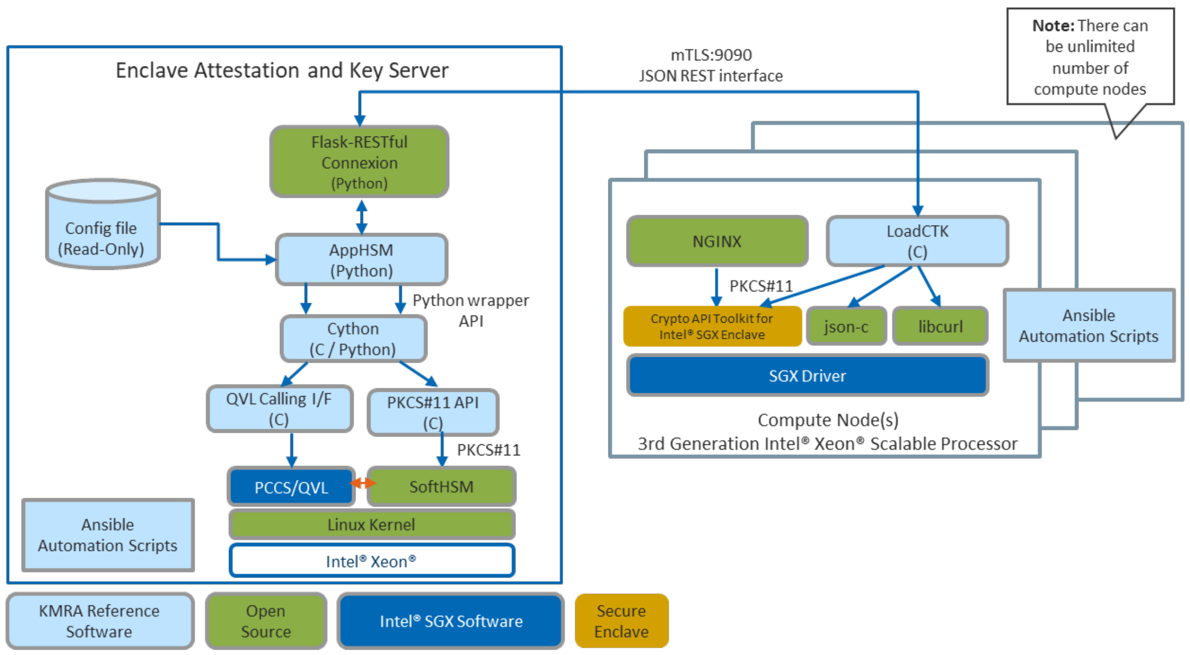


Figure 2. KMRA NGINX/Intel SGX Key Management Software Design

KMRA service node is a centralized HSM that provisions wrapped keys to the compute node. A Flask REST API runs on the service node with Connexion verifying all incoming requests. The REST API uses a Cython wrapper for C to interact with SoftHSMv2 to wrap and extract keys through the PKCS#11 interface. The operations are exposed and supported via the PKCS#11 interface for Linux. This universal interface helps security applications to access and work with the key servers and HSM. In this case, the PKCS#11 interface is used by the NGINX application to access keys in the Crypto API Toolkit for Intel SGX enclave.

To validate the client, mutual transport layer security (TLS) is implemented on the service node where each client certificate is verified. The client certificate must be generated by a mutual certificate authority (CA). The Subject OUN field is extracted from the certificate and mapped to permissions and keys in the configuration file. The Intel SGX quote of a client is validated by the Quote Verification Library before the keys are extracted and provisioned.

KMRA compute node is a client running on an Intel SGX-enabled platform. The client sends a request to a service node. The request contains an Intel SGX quote, a public key from Crypto API Toolkit for Intel SGX, and a unique ID to identify the keypair to extract. The client constructs the requests by using json-c and sends requests with libcurl.

When the client receives a response containing wrapped keys, the server certificate is validated, and the keys are imported into Crypto API Toolkit for Intel SGX. For NGINX to access the secured private key provisioned by the service node, a libp11 engine is configured with OpenSSL. The libp11 engine is an interface for NGINX to access keys secured by Crypto API Toolkit for Intel SGX.

2.3 KMRA Software Bill of Materials

Table 3. KMRA Software Bill of Materials

COMPONENT NAME	SOURCE
Connexion	https://github.com/zalando/connexion
Flask-RESTful	https://github.com/flask-restful/flask-restful/
Flask	https://github.com/pallets/flask/
Cython	https://github.com/cython/cython/
Glib	https://tracker.debian.org/pkg/glib2.0
Json-c	https://tracker.debian.org/pkg/json-c
libcurl	https://tracker.debian.org/pkg/curl
crypto-api-toolkit	https://github.com/intel/crypto-api-toolkit
SoftHSMv2	https://github.com/opendnssec/SoftHSMv2/
Intel SGX PSW, SDK	https://github.com/intel/linux-sgx
Intel SGX driver	https://github.com/intel/linux-sgx-driver

COMPONENT NAME	SOURCE
Ansible	https://github.com/ansible/ansible/
DCAP	https://github.com/intel/SGXDataCenterAttestationPrimitives
Libp11	https://github.com/OpenSC/libp11/
NGINX	https://github.com/nginx/nginx/
OpenSSL	https://www.openssl.org/
requests	https://tracker.debian.org/pkg/requests
Intel SGX SSL	https://github.com/intel/intel-sgx-ssl
pytest	https://github.com/pytest-dev/pytest/
pylint	https://github.com/PyCQA/pylint
cmocka	https://github.com/clibs/cmocka/
Ansible-lint	https://github.com/ansible-community/ansible-lint
Autoconf	https://www.gnu.org/software/autoconf/
Build-essentials	https://packages.debian.org/jessie/build-essential
Docker	https://www.docker.com/
Git	https://tracker.debian.org/pkg/git
Automake	https://www.gnu.org/software/automake/
Libtool	https://www.gnu.org/software/libtool/
Wget	https://www.gnu.org/software/wget/wget.html
Make	http://savannah.gnu.org/projects/make/
Pip	https://github.com/pypa/pip/
Python-apt	https://sourceforge.net/projects/python-apt/
Sudo	https://www.sudo.ws/
G++	https://gcc.gnu.org/
Dkms	https://github.com/dell/dkms
NodeJS	https://nodejs.org/en/
J2cli	https://github.com/kolypto/j2cli
GnuPG	https://gnupg.org/
PKCS11-Proxy	https://github.com/SUNET/pkcs11-proxy

2.4 KMRA Releases

Refer to the following table for Intel SGX and DCAP versions in KMRA releases. These versions are required in the Prerequisites section.

Table 4. Intel® SGX and DCAP Versions for KMRA

KMRA VERSION	SGX VERSION	DCAP VERSION
2.3	**2.18.1**	**1.15**
2.2.2	2.17	1.14
2.2	2.16	1.13
2.0, 2.0.1, 2.1	2.15.1	1.12.1
1.4	2.15	1.12
1.3	2.14	1.11
1.2, 1.2.1	2.13.3	1.10.3
1.1	2.12	1.9
1.0	2.11	1.8

3 Step by Step Installation Overview

The following table summarizes the installation steps and includes links to the details elsewhere in this document.

Table 5. Installation Steps

STEP	DESCRIPTION	LINK TO WHERE DISCUSSED
Step 1	Install prerequisites	Prerequisites
Step 2	Install Intel Provisioning Certificate Caching Service (PCCS)	Platform Registration and Attestation
Step 3	Deploy KMRA Docker containers	Deploy the KMRA demo using containers
Step 4	Use Ansible to install Intel SGX components	Installation of SGX Components Using Ansible
Step 5	Use Ansible scripts to set up and install KMRA	KMRA Setup and Installation Using Ansible Scripts
Step 6	Remove components installed by the Ansible playbooks	Removal of Components Installed by Ansible Scripts

4 Prerequisites

4.1 Software

It is assumed Linux is being used, specifically Ubuntu 18.04, 20.04, 22.04, or RHEL 8.2. For any other operating system, see the Reference Documentation for source code and more information about each component.

4.2 Hardware

This document is specific to the setup of NGINX with the Crypto API Toolkit for Intel SGX on a production-fused 3rd and 4th Gen Intel® Xeon® Scalable processor.

4.2.1 SGX BIOS Option on a 3rd and 4th Gen Intel® Xeon® Scalable Processor

Intel SGX must be enabled in the BIOS. Without that, the Intel SGX kernel module cannot be loaded. The process requires the correct hardware (installing memory modules in certain way) and software configuration (microcode updates, BIOS options).

4.2.1.1 BIOS Hardware Configuration

To enable Intel SGX in BIOS on a 3rd and 4th Gen Intel® Xeon® Scalable processor, correct memory installation is required. See the Reference Documentation for platform enablement and validation requirements. Not every memory configuration is supported by Intel SGX. If your memory configuration is not correct, the Intel SGX option is grayed in the BIOS settings.

The reference setup was tested on a platform where every slot 0 of each memory bank contained an 8 GB DDR4 module.

4.2.1.2 BIOS Software Configuration

Note: We recommend installing the latest microcode updates before enabling SGX in the BIOS.

The following lists the options needed for Intel SGX enablement.

Socket Configuration->Common RefCode Configuration->UMA-Based Clustering = [Disable]

Socket Configuration->Processor Configuration->Total Memory Encryption (TME) = [Enable]

Socket Configuration->Processor Configuration->SW Guard Extensions (Intel® SGX) = [Enable]

Socket Configuration->Processor Configuration->SGXLEPUBKEYHASHx Write Enable = [Enable]

Socket Configuration->Processor Configuration->Enable/Disable SGX Debug = [Disable]

Socket Configuration->Processor Configuration->Enable/Disable SGX Auto MP Registration Agent = [Enable]

Advanced->HW Validation Test Only->Delayed Authentication Mode (DAM) Override->Enable

Advanced->HW Validation Test Only->Delayed Authentication Mode (DAM)→Disable

4.3 Ansible

Ansible is an open-source project for managing software installations and configurations. With Ansible you can automate many installation steps in 'playbooks' that are easy to run and maintain.

An Ansible package can be installed using the pip3 Python package manager. Always use the latest version of the Ansible package.

For Ubuntu, use the following:

```
# sudo apt install python3-pip
# sudo -H python3 -m pip install ansible
```

For RHEL, use the following:

```
$ sudo yum install python3-pip
$ sudo -H python3 -m pip install ansible
```

4.4 Configuration of sudo

Ansible playbooks for installing Intel SGX components must be executed as a non-root user with sudo password-less access. To enable 'sudo' without password' for a target user, use the following command:

```
# sudo visudo
```

A configuration screen is displayed.

Enter the following for password-less sudo access for the target user.

```
your_username ALL=(ALL) NOPASSWD: ALL
```

5 Platform Registration and Attestation

5.1 Provisioning Certification Service

Log in and subscribe to get a key for the Provisioning Certification Service at the following site:
https://api.portal.trustedservices.intel.com/provisioning-certification

The primary key from the subscription is used in the next steps.

This key is accessible at any time on that website.

5.2 Intel® SGX Multi-Package Registration Software Installation

This section is designed to provide a brief set of instructions to aid a user in installing and configuring the multi-package libraries and tools. Packages for multi-package registration service are available at the following sites.

Note: The following commands install Intel SGX components for KMRA v2.3 with SGX 2.18 and DCAP 1.15 versions

For Ubuntu:

Packages (in an archive) for Ubuntu 22.04 are located here:

https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/ubuntu22.04-server/sgx_debian_local_repo.tgz

Packages (in an archive) for Ubuntu 20.04 are located here:

https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/ubuntu20.04-server/sgx_debian_local_repo.tgz

Packages (in an archive) for Ubuntu 18.04 are located here:

https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/ubuntu18.04-server/sgx_debian_local_repo.tgz

Download and extract the archive:

```
(non-root user) $ wget <ARCHIVE_URL>
(non-root user) $ tar zxvf sgx_debian_local_repo.tgz
```

Examples of Ubuntu 20.04, please change the .deb name in the following commands based on your Ubuntu version.

libsgx-ra-network

```
(non-root user) $ sudo dpkg -i ./sgx_debian_local_repo/pool/main/libs/libsgx-ra-network/libsgx-ra-
network_1.15.100.3-focal1_amd64.deb
```

libsgx-ra-uefi

```
(non-root user) $ sudo dpkg -i ./sgx_debian_local_repo/pool/main/libs/libsgx-ra-uefi/libsgx-ra-
uefi_1.15.100.3-focal1_amd64.deb
```

sgx-ra-service

```
(non-root user) $ sudo dpkg -i ./sgx_debian_local_repo/pool/main/s/sgx-ra-service/sgx-ra-
service_1.15.100.3-focal1_amd64.deb
```

To query the installed packages and their versions:

```
(non-root user) $ sudo dpkg-query --list libsgx-ra-network libsgx-ra-uefi sgx-ra-service
```

For RHEL:

Packages (in an archive) for RHEL 8.6 are here: https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/rhel8.6-server/sgx_rpm_local_repo.tgz

Download and extract archive:

```
(non-root user) $ wget https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/rhel8.6-
server/sgx_rpm_local_repo.tgz
(non-root user) $ tar zxvf sgx_rpm_local_repo.tgz
```

Install the libsgx-ra-network, libsgx-ra-uefi, and sgx-ra-service packages:

```
(non-root user) $ sudo rpm -i ./sgx_rpm_local_repo/libsgx-ra-network-1.15.100.3-1.el8.x86_64.rpm
./sgx_rpm_local_repo/libsgx-ra-uefi-1.15.100.3-1.el8.x86_64.rpm ./sgx_rpm_local_repo/sgx-ra-
service-1.15.100.3-1.el8.x86_64.rpm
```

To query the installed packages and their versions:

```
(non-root user) $ sudo rpm --query libsgx-ra-network sgx_rpm_local_repo sgx-ra-service
```

Be sure to set correct proxy variables in the /etc/ environment. The primary key needs to be added to the Intel SGX Multi-Package Registration configuration file at /etc/mpa_registration.conf. After rebooting, observe the log at /var/log/mpa_registration.log for successful registration.

5.3 Intel Provisioning Certificate Caching Service (PCCS) Installation

Before running Ansible playbooks for KMRA installation, install the Intel Provisioning Certificate Caching Service (PCCS).

The installation steps and the procedure are described in more detail in the DCAP PCCS readme: https://github.com/intel/SGXDataCenterAttestationPrimitives/blob/DCAP_1.15/QuoteGeneration/pccs/README.md

Before running the install_pccs Ansible script, you must first add your "Intel API key" in `/kmra/ansible/sgx_infra_setup/group_vars/all`. To do this, on line 75, replace the "X" api_key: XXXX with your Intel API primary key from Section 5.1.

To install PCCS with KMRA ansible-playbook, run:

```
(non-root user) $ ansible-playbook -i inventory install_pccs.yml
```

To install PCCS manually, follow the OS-specific steps below.

Note: In the following steps, it is assumed that PCCS is installed on every host, is used in setup, and listens on 'localhost' using port 8081. If PCCS is installed on a separate machine, set its hostname in the `group_vars/all` file by updating the variable named 'pccs_hostname'.

For Ubuntu:

PCCS package (in an archive) for Ubuntu 22.04 is located here: https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/ubuntu22.04-server/sgx_debian_local_repo.tgz

PCCS package (in an archive) for Ubuntu 20.04 is located here: https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/ubuntu20.04-server/sgx_debian_local_repo.tgz

PCCS package (in an archive) for Ubuntu 18.04 is located here: https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/ubuntu18.04-server/sgx_debian_local_repo.tgz

As root user, update the NodeJS version to 16 or later:

```
(root user) # curl -sL https://deb.nodesource.com/setup_16.x | bash -
(root user) # apt-get install -y nodejs
(root user) # node --version
```

Download and install the PCCS Debian package:

```
(root user) # wget <ARCHIVE_URL>
(root user) $ tar zxvf sgx_debian_local_repo.tgz
(root user) # dpkg -i ./sgx_debian_local_repo/pool/main/s/sgx-dcap-pccs/sgx-dcap-pccs_1.15.100.3-
focal1_amd64.deb
```

For RHEL:

As root user, install NodeJS version 10.20 or later:

```
(root user) # yum install -y nodejs
```

Download, extract, and install the PCCS RHEL package:

```
(root user) # wget https://download.01.org/intel-sgx/sgx-dcap/1.15/linux/distro/rhel8.6-
server/sgx_rpm_local_repo.tgz
(root user) # tar zxvf sgx_rpm_local_repo.tgz
(root user) # rpm -i ./sgx_rpm_local_repo/sgx-dcap-pccs-1.15.100.4-1.el8.x86_64.rpm
```

Go to /opt/intel/sgx-dcap-pccs directory and run the following command:

```
(root user) # sudo -u pccs ./install.sh
```

Answer all questions when prompted and use the primary key when asked to "Set your Intel PCS API key".

Use the following command to check the status of the PCCS service:
```
(root user) # systemctl status pccs
```

Use the following command to start/stop/restart the PCCS service:
```
(root user) # systemctl start/stop/restart pccs
```

5.4 Create group and add user to group for SGX components

To provision crypto-api-toolkit token for nginx the user needs to be added to 'sgx' and 'sgx_prv' group. The group is not created on "sgx-aesm-service" package installation. Use the following instructions to create and configure this group manually:

GID - Next available group ID in /etc/group for 'sgx_prv' group eg. 1004

USER - User to add to 'sgx_prv' group for access to SGX
```
(non-root user) $ sudo groupadd --gid <GID> sgx_prv
(non-root user) $ sudo usermod -a -G sgx_prv <USER>
```

Repeat same steps for 'sgx' group:
```
(non-root user) $ sudo groupadd --gid <GID> sgx
(non-root user) $ sudo usermod -a -G sgx <USER>
```

Verify that group and user in /etc/group is created correctly.

To refresh permissions for the current user, perform one of the following actions:

- Reload the session by logging out and in.

OR

- Start a new shell session by logging to a new group:
```
(non-root user) $ newgrp sgx_prv
(non-root user) $ newgrp sgx
```

To check that the current user is in group, run:
```
(non-root user) $ groups
```

To check that the root folder /kmra under the group ‘sgx’, run:
```
(non-root user) $ ls -lrt
```

5.5 Overview

This section outlines the steps to deploy the KMRA demo using containers.

Note: KMRA Docker images are available on the Docker Hub.

https://hub.docker.com/r/intel/apphsm

https://hub.docker.com/r/intel/ctk_loadkey

https://hub.docker.com/r/intel/pccs

https://hub.docker.com/r/intel/nginx

The source code for GPL/LGPL licensed components distributed in KMRA Docker images can be found in the Docker images here: `/sources`

KMRA v1.4 - v1

KMRA v2.0.1 - v2

KMRA v2.1 - v2.1

KMRA v2.2 – v2.2.2

KMRA v2.3 – v2.3

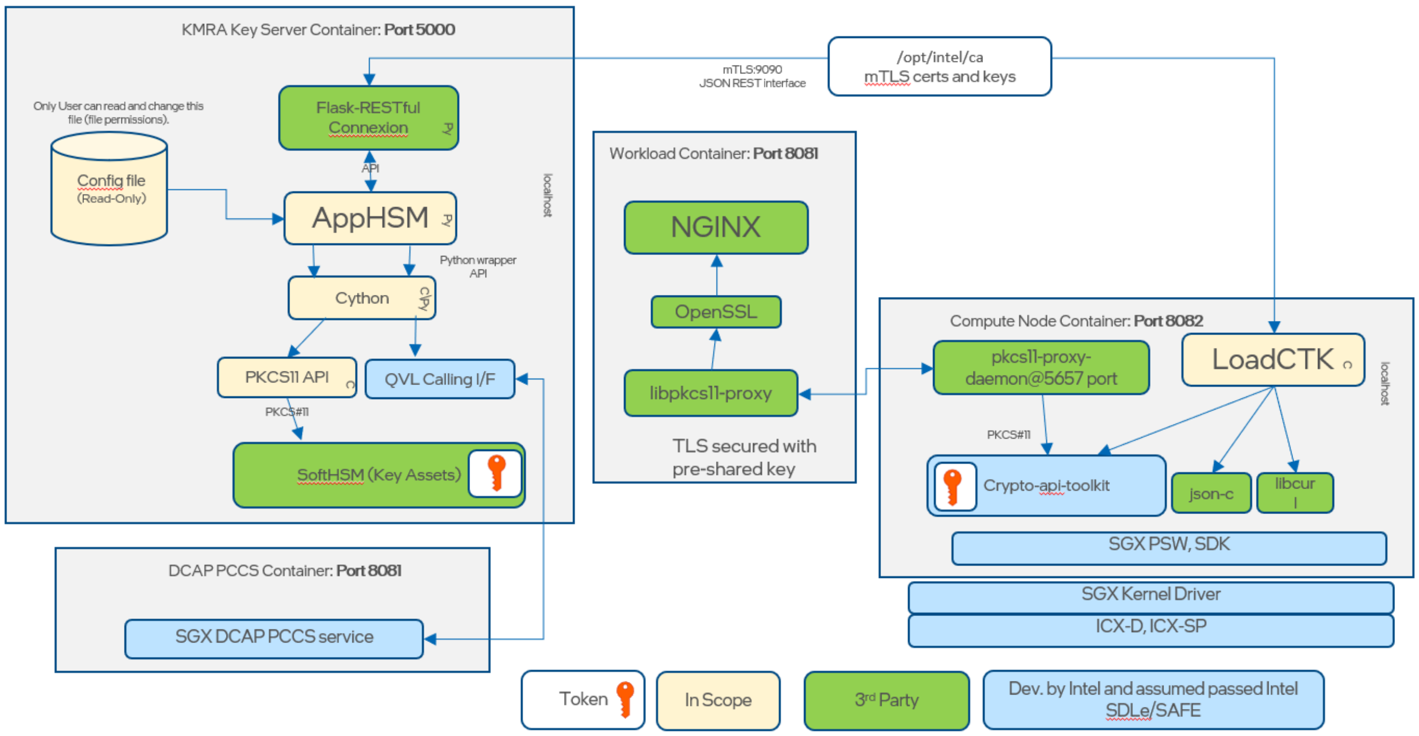

KMRA Key Server Container: Port 5000
Only User can read and change this file (file permissions).
Config file (Read-Only)
Flask-RESTful Connexion
API
AppHSM
Python wrapper API
Cython
PKCS11 API
QVL Calling I/F
PKCS#11
SoftHSM (Key Assets)
localhost
mTLS:9090 JSON REST interface
/opt/intel/ca mTLS certs and keys
Workload Container: Port 8081
NGINX
OpenSSL
libpkcs11-proxy
TLS secured with pre-shared key
Compute Node Container: Port 8082
pkcs11-proxy-daemon@5657 port
LoadCTK
PKCS#11
Crypto-api-toolkit
json-c
libcurl
SGX PSW, SDK
SGX Kernel Driver
ICX-D, ICX-SP
DCAP PCCS Container: Port 8081
SGX DCAP PCCS service
Token
In Scope
3rd Party
Dev. by Intel and assumed passed Intel SDLe/SAFE

Figure 3. KMRA SW Design and Deployment Using Docker Images

5.6 Prerequisites

To build KMRA Docker images from Dockerfiles, follow these steps:

1. Add the user in docker group
2. In case you are behind a network proxy add the proxy variables in the ~/.docker/config.json file
3. Install the Intel SGX kernel driver. The out-of-tree Intel SGX kernel driver can be installed using the KMRA Ansible scripts in the ansible/sgx-infra-setup directory, as described in Section 6.4.
4. If deploying containers behind a proxy, update no_proxy, http_proxy, and https_proxy in Dockerfile.apphsm and Dockerfile.ctk with the correct proxy settings. Also, update the no_proxy variable in *.sh files.

5.7 Deployment via Dockerfiles

Use the following steps to deploy the containers via Dockerfiles.

1. Create custom bridge network:
```
$ docker network create kmra-net
$ docker network ls | grep kmra
```
2. Build and run the Dockerfiles from the main directory as mentioned in the next sections:
```
$ cd kmra/
```
   Note: Update <PATH-TO-REPO> in the Docker run commands below as Docker requires absolute paths when mounting volumes (-v option).

5.8 PCCS Container

1. If needed, set proxy environment variables (no_proxy, http_proxy, https_proxy) before building PCCS container:
```
$ export no_proxy="..."
$ export http_proxy="..."
$ export https_proxy="..."
```
2. Build the PCCS container:
```
$ docker build -t pccs -f containers/pccs/Dockerfile .
```
3. Prepare certificate and configuration file for the PCCS container (on host):
```
$ cd kmra/containers/pccs/
$ rm -rf certs/*
$ mkdir -p certs
$ cd certs
$ bash ../scripts/pccs_generate_certificates.sh
```

4. Set PCCS_* environment variables before generating config file using export (on host). Log in and subscribe to get a key for the Provisioning Certification Service at the following site: https://api.portal.trustedservices.intel.com/provisioning-certification. The primary key from the subscription will be used in the next steps. This key is accessible at any time on this website.

```
$ (Optional) export PCCS_ADMIN_PASS="example-admin-pass"
$ export PCCS_API_KEY="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

PCCS_API_KEY environment is required by the script. Other settings are optional, and default values are used for values not given. See Environment Variables for a list of available variables.
Note: Remove the quotation marks after adding the values.

5. Generate configuration for PCCS (on host):

```
$ cd kmra/containers/pccs/config/
$ bash ../scripts/pccs_generate_config.sh
```

6. Run PCCS container with mounted generated certificates and config file:

```
$ cd kmra/containers/pccs/
$ bash run_pccs.sh
```

7. Build arguments
   - USER - the name of the user inside the container (`kmra` by default)
   - UID - UID for the above user (`1000` by default)

8. Environment Variables
   - PCCS_API_KEY - value of 'primary key' from the subscription from site: https://api.portal.trustedservices.intel.com/provisioning-certification
   - PCCS_ADMIN_PASS - password for PCCS admin (not used in this demo but value can be provided)
   - PCCS_USER_PASS - password for PCCS user (not used in this demo but value can be provided)
   - http_proxy, https_proxy - must be set if using PCCS behind corporate proxy: '-e https_proxy="http://proxy:port"' (to be set at image build time)
   - PCCS_PORT - port on which PCCS service will listen (8081 by default)
   - PCCS_LOCAL_ONLY - flag (Y/N) indicating whether PCCS service should listen on local network interface only (N by default)

5.9 AppHSM Container

1. Generate mTLS certificates and keys to share with the containers. The keys and certificates generated by the following script are valid for one day only.

```
$ cd kmra/apphsm/ca/
$ bash -c "APPHSM_HOSTNAME=apphsm ./gen_all.sh"
```

Note: APPHSM_HOSTNAME must match the name of AppHSM container. The key(s), certificate(s) and csr files might need to be regenerated as they have an expiry time on them. Also, the user might have to change permissions to make those file readable.

2. Build and run Apphsm Key Server container:

```
  $ cd kmra/
  $ docker build -t apphsm -f containers/Dockerfile.apphsm .
  $ docker run -it --rm \
  --cpu-shares 512 --pids-limit 100 --memory=2048m --security-opt=no-new-privileges \
  --read-only --tmpfs /var/lib/softhsm/tokens --tmpfs /tmp \
  -v `pwd`/containers/apphsm/sgx_default_qcnl.conf:/etc/sgx_default_qcnl.conf:ro \
  -v `pwd`/apphsm/ca/:/opt/intel/ca:ro \
  --name apphsm --env no_proxy="pccs" \
  --network kmra-net --health-cmd=`pgrep -f "python3.8 /opt/intel/apphsm/apphsm.py" || exit 1`
\
  apphsm:latest
```

3. (Optional) Run AppHSM with custom configuration:
   - Create custom key and certificate. They can be generated by the script:

```
$ cd containers/apphsm/custom_config
$ bash ./gen_key_cert.sh
```

   - Edit the configuration file "apphsm.conf" and add the required key(s). The "token_name" name field must be different for every key. The "key_name" and "certificate_file" fields must correspond to file names with key and certificate placed in "custom_config" directory. The provided default configuration matches the default key and certificate file names generated by "gen_key_cert.sh" script. Mount the directory with custom config file, keys, and certificates:

```
$ cd kmra/
```

```
  $ docker run -it --rm \
  --cpu-shares 512 --pids-limit 100 --memory=2048m --security-opt=no-new-privileges \
  --read-only --tmpfs /var/lib/softhsm/tokens --tmpfs /tmp \
  -v `pwd`/containers/apphsm/sgx_default_qcnl.conf:/etc/sgx_default_qcnl.conf:ro \
  -v `pwd`/apphsm/ca/:/opt/intel/ca:ro \
  -v `pwd`/containers/apphsm/custom_config:/opt/apphsm_config:ro \
  --name apphsm --env no_proxy="pccs" --cap-drop=all \
  --network kmra-net --health-cmd=`pgrep -f "python3 /opt/intel/apphsm/apphsm.py" || exit 1` \
apphsm:latest
```

4. Build arguments
   - DCAP_VERSION - 1.13 by default
   - DCAP_LIB_VERSION - 1.13.100.4 by default
   - SGX_LIB_VERSION - 2.16.100.4 by default
   - USER - a name of the user inside the container (kmra by default)
   - UID - UID for the above user (1000 by default)

5. Environment variables
   - APPHSM_PORT - port on which AppHSM service listens (5000 by default)
   - no_proxy - a list of host names, IP addresses, IP subnets for which the proxy is not used
   - APPHSM_KEY_IN_TOKEN_NAME - label of the key in the softhsm for apphsm (key_1 by default)
   - APPHSM_KEY_IN_TOKEN_CERT_PATH - path to a sample certificate served with key from softhsm
   - APPHSM_TOKEN_NAME - label of the token in the softhsm for apphsm (token_1 by default)
   - TEST_CTK_LOADKEY_CERT_USER_ID - name of the client that is visible in the 'OU' field of the generated certificate for the client side. This name must be defined in apphsm.conf in the 'clients' section. (ctk_loadkey_user_id_01234 by default)
   - TEST_UNIQUE_UID - unique id of the key definition located on AppHSM side in apphsm.conf. You can obtain this key by requesting this unique id using ctk_loadkey
   - DEFAULT_USER_PIN - softhsm user pin. Valid length is 4-16 chars. (1234 is default)
   - DEFAULT_SO_PIN - softhsm security officer pin. Valid length is 4-16 chars. Consult your security officer. (12345678 is default)
   - APPHSM_CUSTOM_CONFIG_DIR - location of custom configuration inside container (/opt/apphsm_config by default)

   PCCS configuration file `containers/apphsm/sgx_default_qcnl.conf`:
   - PCCS_URL - valid URL address of PCCS service (https://pccs:8081/sgx/certification/v3/ by default)
   - USE_SECURE_CERT - to accept insecure HTTPS cert for PCCS URL, set this option to FALSE (FALSE by default)

   gen_key_cert.sh script:
   - DEFAULT_KEY_NAME - file name to store the generated key
   - DEFAULT_CERTIFICATE_NAME - file name to store the generated certificate

5.10 Ctk_loadkey Container

1. Build and run ctk_loadkey container:

```
  $ cd kmra/
  $ docker build -t ctk_loadkey -f containers/Dockerfile.ctk .
  $ docker run -it --rm --device /dev/sgx/enclave \
  --cpu-shares 512 --pids-limit 100 --memory=2048m --security-opt=no-new-privileges \
  --device /dev/sgx/provision \
  --env PCCS_HOSTNAME=pccs --env APPHSM_HOSTNAME=apphsm --env no_proxy="apphsm,pccs" \
  --name ctk_loadkey --health-cmd='curl -s --insecure https://localhost:8082 || exit 1' --
network kmra-net -p 8082:8082 \
   --read-only --tmpfs /opt/intel/cryptoapitoolkit/tokens --tmpfs /tmp \
   -v `pwd`/containers/ctk/sgx_default_qcnl.conf:/etc/sgx_default_qcnl.conf:ro \
   -v `pwd`/containers/nginx/p11_proxy_tls.psk:/etc/p11_proxy_tls.psk:ro \
   -v `pwd`apphsm/ca/:/opt/intel/ca:ro \
   --user kmra:$(getent group sgx_prv | cut -d: -f3) ctk_loadkey:latest
```

NOTE:
On some host systems (e.g Ubuntu 22) additional argument may be required to mount /dev/sgx_enclave inside container:

```
   --group-add $(getent group sgx | cut -d: -f3)
```

You can check if sgx group is needed by executing command below on host:

```
   $ ls -lg /dev/sgx_enclave
     crw-rw---- 1 sgx 10, 125 Dec 13 10:54 /dev/sgx_enclave
```

2. Build arguments
   - DCAP_VERSION - 1.13 by default

- DCAP_LIB_VERSION - 1.13.100.4 by default
- SGX_VERSION - 2.16 by default
- SGX_LIB_VERSION - 2.16.100.4 by default
- USER - a name of the user inside the container (kmra by default)
- UID - UID for the above user (1000 by default)

3. Environment variables
    - APPHSM_HOSTNAME - host on which AppHSM service listens (localhost by default). Usually, it is set to the name of the AppHSM container - "apphsm" (the last argument in Docker run … command starting AppHSM container)
    - APPHSM_PORT - port on which AppHSM service listens (5000 by default)
    - NGINX_HOSTNAME - the host name on which NGINX listens (0.0.0.0 by default)
    - NGINX_PORT - port on which NGINX listens (8082 by default)
    - no_proxy - a list of host names, IP addresses, IP subnets for which the proxy is not used
    - CLIENT_TOKEN - private key for the certificate for NGINX is stored by ctk_loadkey in this token label (client_token by default)
    - CLIENT_KEY_LABEL - label of the key which NGINX uses to find its key (client_key_priv by default)
    - TEST_UNIQUE_UID - unique id of the key definition located on AppHSM side in apphsm.conf. You can obtain this key by requesting this unique id using ctk_loadkey
    - TEST_UNIQUE_UID - unique id of the key definition located on AppHSM side in apphsm.conf. Client can obtain this key by requesting this unique id using ctk_loadkey
    - DEFAULT_USER_PIN - Crypto-Api-Toolkit user pin. Valid length is 4-16 chars. (1234 is default)
    - DEFAULT_SO_PIN - Crypto-Api-Toolkit security officer pin. Valid length is 4-16 chars. Consult your security officer. (12345678 is default)
    - DEFAULT_CLIENT_TOKEN_ID - ID of the key pair. The ID is in hexadecimal with a variable length, used by pkcs11-tool as argument to write certificate into token
    - KEEP_TOKENS - Do not cleanup crypto api toolkit tokens on startup

    PCCS configuration file containers/ctk/sgx_default_qcnl.conf:

    - PCCS_URL - valid URL address of PCCS service (https://pccs:8081/sgx/certification/v3/ by default)
    - USE_SECURE_CERT - to accept insecure HTTPS cert for PCCS URL, set this option to FALSE (FALSE by default)

5.11 NGINX Container

Build and run NGINX container:

```
$ cd kmra/
$ docker build -t nginx -f containers/nginx/Dockerfile.nginx .
$ docker run -it --rm --cpu-shares 512 --pids-limit 100 --memory=2048m \
  --read-only --tmpfs /tmp \
  --env PKCS11_PROXY_SOCKET=tls://ctk_loadkey:5657 \
  --env no_proxy="ctk_loadkey" \
  -v `pwd`/containers/nginx/p11_proxy_tls.psk:/etc/p11_proxy_tls.psk:ro \
  --name nginx --network kmra-net -p 8082:8082 nginx:latest
```

- Environment variables:`PKCS11_PROXY_SOCKET - ctk_loadkey container IP address and exposed port number of pkcs11-proxy`
- NGINX_HOSTNAME - the host name on which NGINX will listen (0.0.0.0 by default)
- NGINX_PORT - port on which NGINX will listen (8082 by default)
- CLIENT_TOKEN - token name with private key for NGINX (client_token by default)
- CLIENT_KEY_LABEL - label of the key which NGINX will use to finds its key (client_key_priv by default)
- DEFAULT_USER_PIN - Crypto-Api-Toolkit user pin, valid length is 4-16 chars (1234 is default)

REMARKS:

- TLS pre-shared key file format: key_name:16 bytes of key in hexadecimal format. The key is common for both ctk_loadkey and NGINX container. See sample p11_proxy_tls.psk file in the source code.

5.12 Common Issues

5.12.1 Failure in task '[create_empty_token_in_hsm: Create token ...]'

Error log:
```
TASK [create_empty_token_in_hsm : Create token with name 'client_token'] ******* fatal:
[localhost]: FAILED! => {"changed": true, "cmd": ["softhsm2-util", "--module",
"/usr/local/lib/libp11sgx.so.0.0.0", "--init-token", "--free", "--label", "client_token", "--
pin", "1234", "--so-pin", "12345 678"], ... Could not initialize the PKCS#11 library/module:
usr/local/lib/libp11sgx.so.0.0.0\nERROR: Please check log files for additional information.",
"stderr_lines": ["[get_driver_type /home/sgx/jenkins/ubuntuServer2004-release-build-tr unk-
213.3/build_target/PROD/label/Builder-UbuntuSrv20/label_exp/ubuntu64/linux-trunk-
opensource/psw/urts/linux/edmm_utility.cpp:111] Failed to open Intel SGX device.", "ERROR:
Could not initialize the PKCS#11 library/module: /usr/local/lib/libp11sgx.so.0.0.0", "ERROR:
Please check log files for additional information."], "stdout": "", "stdout_lines": []}
```

Root cause:
There is a problem with sharing SGX services (e.g., lack of `--device /dev/sgx/enclave` passed to the container during runtime)

5.12.2 Failure in task [install_ctk_loadkey: Copy ca cert and ctk_loadkey keys ...]

Error log:
```
TASK [install_ctk_loadkey : Copy ca cert and ctk_loadkey keys to /opt/intel/ctk_loadkey] ***
changed: [localhost] => (item=ctk_loadkey.crt) fatal: [localhost]: FAILED! => {"msg": "an error
occurred while trying to read the file /opt/intel/ca/ctk_loadkey.key': [Errno 13] Permission
denied: b'/opt/intel/ca/ctk_loadkey.key'. [Errno 13] Permission denied:
b'/opt/intel/ca/ctk_loadkey.key'"}
```

Root cause:
There is mismatch between user id in the container and the user that owns shared certificate/key files. Make sure that certificate/key files for ctk_loadkey are accessible for 'kmra' user inside container.

5.12.3 Enclave not authorized to run in task [provision_ctk_with_key_from_apphsm ...]

Error log:
```
TASK [provision_ctk_with_key_from_apphsm : Provision token client_token with key
client_key_priv from AppHSM] *** fatal: [localhost]: FAILED! => {"changed": true, "cmd": "cd
/opt/intel/ctk_loadkey; https_proxy=\"\" ./ctk_loadkey -t client_token -p 1234 -u
unique_id_1234 -P 5000 -H silpixa00400537", "delta": "0:00:00.478512", "end": "2021-07-19
13:33:45.166066", "msg": "non-zero return code", "rc": 5, "start": "2021-07-19
13:33:44.687554", "stderr": "[error_driver2api sgx_enclave_common.cpp:247] Enclave not
authorized to run, .e.g. provisioning enclave hosted in app without access rights to
/dev/sgx_provision. You need add the user id to group sgx_prv or run the app as
root.\n[load_pce ../pce_wrapper.cpp:175] Error, call sgx_create_enclave for PCE fail
[load_pce], SGXError:4004.", "stderr_lines": ["[error_driver2api sgx_enclave_common.cpp:247]
Enclave not authorized to run, .e.g. provisioning enclave hosted in app without access rights
to /dev/sgx_provision. You need add the user id to group sgx_prv or run the app as root.",
"[load_pce ../pce_wrapper.cpp:175] Error, call sgx_create_enclave for PCE fail [load_pce],
SGXError:4004."], "stdout": "Error during C_WrapKey-size: CKR_GENERAL_ERROR\nError during
creating ecdsa_quote: CKR_GENERAL_ERROR\nError during ctk_quote generation", "stdout_lines":
["Error during C_WrapKey-size: CKR_GENERAL_ERROR", "Error during creating ecdsa_quote:
CKR_GENERAL_ERROR", "Error during ctk_quote generation"]}
```

Root cause:
There is a mismatch between group id for 'sgx_prv' group in the container and the same group on the host. Make sure that both group ids are the same.

5.12.4 SSL peer certificate error in task [provision_ctk_with_key_from_apphsm..]

Error log:
```
fatal: [localhost]: FAILED! => {"changed": true, "cmd": "cd /opt/intel/ctk_loadkey;
https_proxy=\"\" ./ctk_loadkey -t client_token -p 1234 -u unique_id_1234 -P 5000 -H apphsm",
"delta": "0:00:01.702477", "end": "2021-07-20 12:15:58.111058", "msg": "non-zero return code",
"rc": 5, "start": "2021-07-20 12:15:56.408581", "stderr": "", "stderr_lines": [], "stdout":
```

```
"rest_api_check_version: Supports AppHSM v0.1 (or newer) API v0.1.\nrest_api_perform_request:
REST API request failed 'SSL peer certificate or SSH remote key was not
OK'!\nrest_api_check_version: Failed to get AppHSM version!\nFAILED REST API initialization for
host 'apphsm' on port 5000: -93\nFailed to send export key request: CKR_GENERAL_ERROR",
"stdout_lines": ["rest_api_check_version: Supports AppHSM v0.1 (or newer) API v0.1.",
"rest_api_perform_request: REST API request failed 'SSL peer certificate or SSH remote key was
not OK'!", "rest_api_check_version: Failed to get AppHSM version!", "FAILED REST API
initialization for host 'apphsm' on port 5000: -93", "Failed to send export key request:
CKR_GENERAL_ERROR"]}
```

Root cause:
AppHSM common name in the AppHSM certificate does not match domain name used for connecting to AppHSM from ctk_loadkey container (e.g., in the error above, certificate was generated for 'localhost,' but it should be generated for 'apphsm' instead). Generate certificates again with proper APPHSM_HOSTNAME variable set and restart apphsm and ctk_loadkey containers.

6 Installation of SGX Components Using Ansible

6.1 Overview

This section outlines the steps to set up the ecosystem for Intel SGX components. Installation is done using Ansible scripts. All components are installed automatically in the correct sequence and configured.

At the end of the installation process, your system is ready to run Intel SGX enclaves.

6.2 Intel SGX Ingredients

The following contains short descriptions of the components that are installed by the Ansible scripts.

6.2.1 Intel SGX DCAP Kernel Driver

The Intel SGX DCAP kernel driver can be used on a machine supporting flexible launch control (FLC) capability. The module is named intel_sgx. Before installation, the in-tree kernel driver for intel_sgx is checked. If intel_sgx is not detected, it is installed. If it is detected, then this step is skipped.

6.2.2 Intel SGX Runtime Libraries – SGX PSW (Platform Software)

Intel SGX PSW libraries are needed for running Intel SGX enclaves.

6.2.3 Intel SGX SDK

SGX SDK contains tools and libraries needed for building Intel SGX applications, including the Crypto API Toolkit for Intel SGX.

6.2.4 Intel® SGX DCAP Libraries

Intel® Software Guard Extensions Data Center Attestation Primitives (Intel® SGX DCAP) provides SGX attestation support targeted for data centers, cloud services providers, and enterprises. This attestation model uses the elliptic curve digital signature algorithm (ECDSA). These libraries are needed for third-party Intel SGX attestation.

6.2.5 Intel® SGX SSL

The Intel® Software Guard Extensions SSL (Intel® SGX SSL) cryptographic library provides cryptographic services for Intel® Software Guard Extensions (Intel® SGX) enclave applications. The Intel® SGX SSL cryptographic library is based on the underlying OpenSSL open-source project, providing a general-purpose cryptography library.

6.2.6 Crypto API Toolkit for Intel® SGX

The Crypto API Toolkit for Intel SGX provides cryptographic functionality, PKCS#11 API, and the SGX enclave to store the NGINX private key and perform crypto operation with the private key. This is based on the open-source SoftHSMv2 project.

6.2.7 Other Components

Additional libraries and tools are installed that are required by some parts of the setup, for example, autotools, compilers, zlib.

6.3 Installed Intel SGX Component Versions

The table below lists the versions of the installed SGX components.

Table 6. Versions of Installed Intel SGX Components

COMPONENT NAME	VERSION
Intel SGX Linux Driver for Intel DCAP	1.41
Intel SGX SDK	2.18
Intel SGX PSW runtime libraries	2.18
Intel SGX SSL	lin_2.18_1.1.1q
Crypto-Api-Toolkit	91ee49
DCAP libraries	1.15.100.3

6.4 Using Ansible Scripts for Intel SGX Ingredient Installation

KMRA source code is at this link: https://01.org/key-management-reference-application-kmra

The Ansible script that is responsible for installing and configuring Intel SGX ingredients is named 'install_sgx_dcap_ingredients.yml'. It is in the ansible/sgx_infra_setup directory.

```
$ cd ansible/sgx_infra_setup/
```

Run the following command as a non-root user with password-less sudo authentication.

```
(non-root user) $ ansible-playbook -i inventory install_sgx_dcap_ingredients.yml
```

The install_sgx_dcap_ingredients.yml command installs the following components:

- Intel SGX DCAP kernel driver
- Intel SGX SDK
- Intel SGX runtime libraries
- Intel SGX DCAP quote generation and quote verification libraries
- Intel SGX-SSL
- Crypto-Api-Toolkit
- SoftHSMv2

During the installation, the Ansible playbook command reports elements as they are installed.

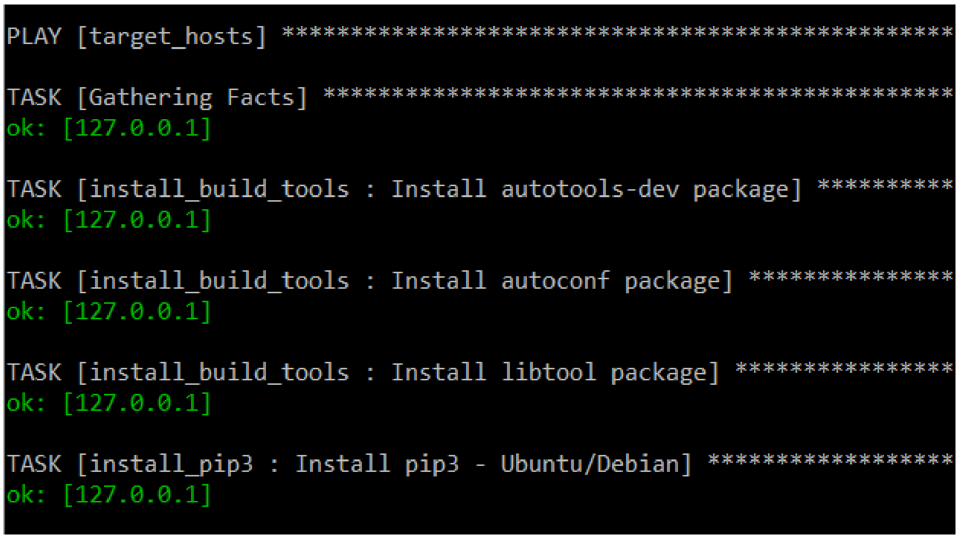
Note: The SGX 2.16 sgx_prv group is not created on "sgx-aesm-service" package installation. Create and configure this group manually with the following instructions.

GID – Next available group ID in /etc/group for sgx-prv group, for example, 1004

USER – User to add to SGX_prv group for access to Intel SGX

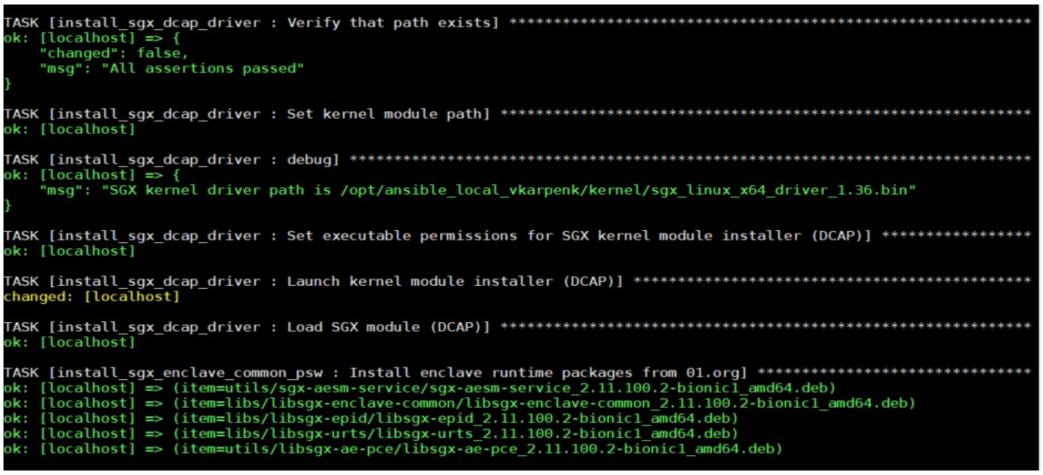
```
(non-root user) $ sudo groupadd --gid <GID> sgx_prv
(non-root user) $ sudo usermod -a -G sgx_prv <USER>
```

Verify group and user in /etc/group are created correctly, log off, and log in to refresh permissions.



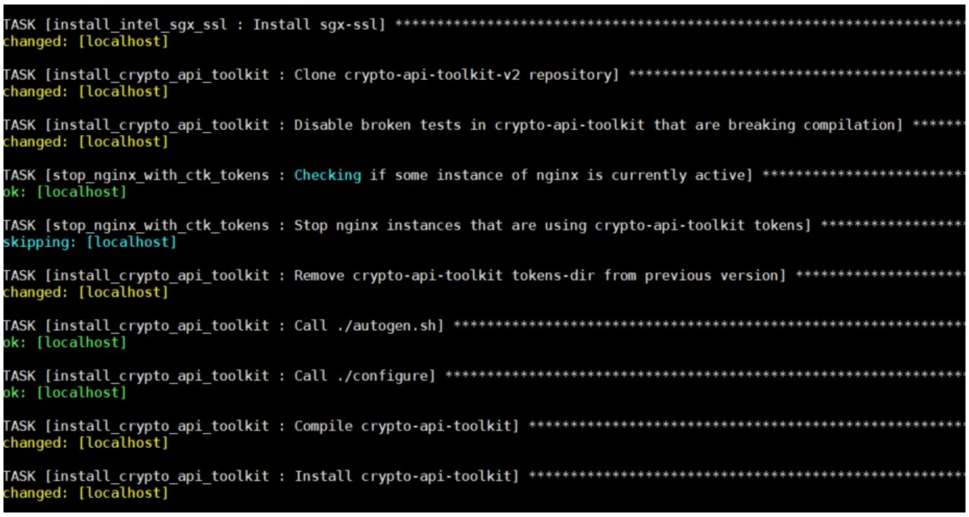
```
PLAY [target_hosts] ************************************************

TASK [Gathering Facts] *********************************************
ok: [127.0.0.1]

TASK [install_build_tools : Install autotools-dev package] **********
ok: [127.0.0.1]

TASK [install_build_tools : Install autoconf package] ****************
ok: [127.0.0.1]

TASK [install_build_tools : Install libtool package] *****************
ok: [127.0.0.1]

TASK [install_pip3 : Install pip3 - Ubuntu/Debian] *******************
ok: [127.0.0.1]
```

Figure 4. Sample Output of Ansible Playbook Command



```
TASK [install_sgx_dcap_driver : Verify that path exists] ***********************************************************
ok: [localhost] => {
    "changed": false,
    "msg": "All assertions passed"
}

TASK [install_sgx_dcap_driver : Set kernel module path] ************************************************************
ok: [localhost]

TASK [install_sgx_dcap_driver : debug] *****************************************************************************
ok: [localhost] => {
    "msg": "SGX kernel driver path is /opt/ansible_local_vkarpenk/kernel/sgx_linux_x64_driver_1.36.bin"
}

TASK [install_sgx_dcap_driver : Set executable permissions for SGX kernel module installer (DCAP)] *****************
ok: [localhost]

TASK [install_sgx_dcap_driver : Launch kernel module installer (DCAP)] *********************************************
changed: [localhost]

TASK [install_sgx_dcap_driver : Load SGX module (DCAP)] ************************************************************
ok: [localhost]

TASK [install_sgx_enclave_common_psw : Install enclave runtime packages from 01.org] *******************************
ok: [localhost] => (item=utils/sgx-aesm-service/sgx-aesm-service_2.11.100.2-bionic1_amd64.deb)
ok: [localhost] => (item=libs/libsgx-enclave-common/libsgx-enclave-common_2.11.100.2-bionic1_amd64.deb)
ok: [localhost] => (item=libs/libsgx-epid/libsgx-epid_2.11.100.2-bionic1_amd64.deb)
ok: [localhost] => (item=libs/libsgx-urts/libsgx-urts_2.11.100.2-bionic1_amd64.deb)
ok: [localhost] => (item=utils/libsgx-ae-pce/libsgx-ae-pce_2.11.100.2-bionic1_amd64.deb)
```

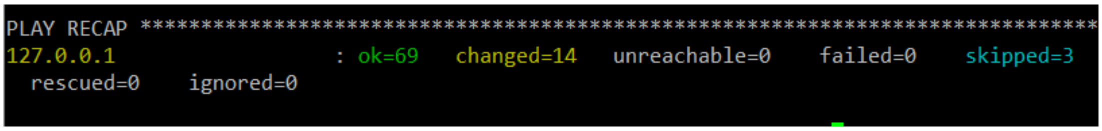
Figure 5. Sample Output of Ansible Playbook Command Installing Intel SGX DCAP Driver and SGX PSW



```
TASK [install_intel_sgx_ssl : Install sgx-ssl] *********************************************************************
changed: [localhost]

TASK [install_crypto_api_toolkit : Clone crypto-api-toolkit-v2 repository] *****************************************
changed: [localhost]

TASK [install_crypto_api_toolkit : Disable broken tests in crypto-api-toolkit that are breaking compilation] *******
changed: [localhost]

TASK [stop_nginx_with_ctk_tokens : Checking if some instance of nginx is currently active] *************************
ok: [localhost]

TASK [stop_nginx_with_ctk_tokens : Stop nginx instances that are using crypto-api-toolkit tokens] ******************
skipping: [localhost]

TASK [install_crypto_api_toolkit : Remove crypto-api-toolkit tokens-dir from previous version] *********************
changed: [localhost]

TASK [install_crypto_api_toolkit : Call ./autogen.sh] **************************************************************
ok: [localhost]

TASK [install_crypto_api_toolkit : Call ./configure] ***************************************************************
ok: [localhost]

TASK [install_crypto_api_toolkit : Compile crypto-api-toolkit] *****************************************************
changed: [localhost]

TASK [install_crypto_api_toolkit : Install crypto-api-toolkit] *****************************************************
changed: [localhost]
```

Figure 6. Sample Output of Ansible Playbook Command Installing crypto-api-toolkit

At the end of the installation process, a summary of tasks is displayed. Installation is successful if there are no 'failed' entries reported.



```
PLAY RECAP *************************************************************************
127.0.0.1                  : ok=69    changed=14    unreachable=0    failed=0    skipped=3
  rescued=0    ignored=0
```

Figure 7. Sample Output of Successful Installation

7 KMRA Setup and Installation Using Ansible Scripts

7.1 Overview

AppHSM is a server application that provides a REST API for delivering cryptographic keys from the SoftHSMv2 key server to the authorized compute servers. This flow is shown in Figure 2. Ctk_loadkey is a client for AppHSM service. The AppHSM REST API server provisions wrapped keys to the Crypto API Toolkit for Intel® SGX token, secured in the enclave. Both applications are installed using Ansible scripts provided by KMRA.

7.2 Installed Components

The following lists the main activities that are completed as a part of the KMRA setup and installation using Ansible scripts:

- Install Intel SGX ingredients (if not installed earlier)
- Install NGINX
- Install local instance of OpenSSL for NGINX

- Install libp11 library support for local OpenSSL
- Create token with RSA2K keypair in Crypto API Toolkit for Intel® SGX
- Create certificate for NGINX using the Crypto API Toolkit for Intel® SGX token
- Start NGINX with the created certificate to help secure the mutually authenticated TLS connection between the KMRA Client (CTK-LoadKey) and KMRA Server (AppHSM) applications
- Run KMRA client application
- Start KMRA server application

Table 7. Versions of Components Installed by KMRA

COMPONENT NAME	VERSION
NGINX	1.21.1
OpenSSL	1.1.1q
Libp11 for OpenSSL	1d93ed

7.3 Using Ansible Scripts for KMRA Setup

Run the following command to start the Ansible scripts to set up KMRA.

```
(non-root user) $ ansible-playbook -i inventory install_ctk_loadkey_and_apphsm.yml
```

On client hosts:

- SGX components are installed (if not installed already)
- Crypto-Api-Toolkit is installed
- ctk_loadkey application, with keys/certificates for mTLS connection, is installed
- Empty token named 'nginx_token' is created in Crypto-Api-Toolkit

On server host:

- SGX components are installed (if not installed already)
- SoftHSMv2 library is installed
- AppHSM, with keys/certificates for mutual transport layer security (mTLS) connection, is installed and started - it listens on port '5000' by default

7.4 Provisioning Wrapped Keys to Crypto-Api-Toolkit

In the setup step above, an empty token was created in Crypto-Api-Toolkit. The empty token needs to be provisioned with a private key from AppHSM before it can be used more securely by NGINX. Actions needed for token provisioning and starting the NGINX instance that will use the provisioned token are defined in the 'provision_ctk_token_and_start_nginx.yml' Ansible playbook.

Before executing this playbook, do the following:

- add current user to the "sgx_prv" group:

```
(non-root user) $ sudo gpasswd -a current_username sgx_prv
```

and one of the following:
- reload the session by relogging
or
– start a new shell session:

```
(non-root user) $ newgrp sgx_prv
```

For token provisioning and to start the NGINX instance, use the following Ansible command:

```
(non-root user) $ ansible-playbook -i inventory provision_ctk_token_and_start_nginx.yml
```

This playbook only targets client hosts and completes the following steps:

- ctk_loadkey is used to download the private key from AppHSM and to import it into the 'nginx_token' inside the Crypto-Api-Toolkit
- Private openssl-1.1.1.i instance is installed for NGINX
- NGINX installed
- NGINX is configured to use the 'nginx_token' from Crypto-Api-Toolkit for TLS connections
- NGINX is started and it is listening on '8082' port

At the end of the scripts, NGINX is configured and started. The Crypto-Api-Toolkit token is used to help secure the TLS connection. NGINX is installed and listening for connections on 8082 port.

The Ansible scripts automatically test whether NGINX is able to use the secured keys for the TLS connection. This test is done using the OpenSSL_time command.

8 Removal of Components Installed by Ansible Scripts

8.1 Overview

The Ansible playbook named 'uninstall_ctk_loadkey_and_apphsm.yml' automatically removes components installed by the previous playbooks.

8.2 Uninstalled Components

The following components are removed automatically:

- NGINX instance with custom OpenSSL (/opt/intel/nginx and /opt/intel/nginx_openssl)
- ctk_loadkey application (/opt/intel/ctk_loadkey)
- AppHSM application (/opt/intel/apphsm)
- Ansible workspace directory (/opt/ansible_local_$USER)

8.3 Using Ansible Scripts for KMRA Cleanup

To remove the KMRA client and server applications and installed components directories, run:

```
(non-root user) $ ansible-playbook -i inventory uninstall_ctk_loadkey_and_apphsm.yml
```

9 VMware vSphere Deployment

To deploy KMRA on virtual machine using VMware vSphere, ensure that ESXi 7.0 is used. Host MPA registration is required (only once) to use SGX:

1. Create a live Linux distribution (i.e., Linux Ubuntu 20.04) bootable USB key using a tool like Rufus.
2. Download the SGX DCAP driver and the PCK Cert ID Retrieval Tool (PCKCIDRT) and copy them both to the live USB key also.
3. Boot to the live Linux distribution and start a command line terminal.
4. Un-tar PCKCIDRT and use it to grab the Platform Manifest and other data from the UEFI variables.
5. Separate and convert the Platform Manifest to a binary blob.
6. Upload the Platform Manifest to the Intel Registration Service to register the platform.

```
Here are the specific command line instructions to perform the above mentioned steps on the USB key
after booting the live OS:
        chmod +x ./sgx_linux_x64_driver_1.41.bin
        sudo ./sgx_linux_x64_driver_1.41.bin
        sudo mount -o remount,exec /dev
        tar xvzf ./PCKIDRetrievalTool_v 1.13.100.4.tar.gz
        cd PCKIDRetrievalTool_v 1.13.100.4/
        sudo ./PCKIDRetrievalTool -f platform_id.csv
        csvtool col 6 platform_id.csv | xxd -r -p > platformmanifest.bin
        curl -v --data-binary @platformmanifest.bin -X POST
https://api.trustedservices.intel.com/sgx/registration/v1/platform -H "Content-Type:
application/octet-stream"
On successful registration, the reply will be a “201 Created” success status.
```

After MPA host registration, create a Virtual Machine in vSphere with SGX enabled: Under *Customize hardware -> Security devices* select “Enable check box for SGX”. Proceed with setup instructions using Ansible scripts on guest OS.

10 Summary

Intel® SGX provides a more secure environment for application owners to run their applications' sensitive code and data inside an Intel SGX enclave, enhancing protection of their enclave code and data from privileged software and applications. This document outlines the steps needed to set up a NGINX workload to access the private key protected inside an Intel® Software Guard Extensions (Intel® SGX) enclave on a 3rd Gen Intel Xeon Scalable processor with production-fused CPU parts, using the Public-Key Cryptography Standard (PKCS) #11 interface and OpenSSL. This paper focuses on making SGX easy to use and deploy, including automated deployment of all required system components. It is recommended that readers may extend this example and refer to the *Intel® Software Guard Extensions (Intel® SGX) – Key Management Reference Application (KMRA) on Intel® Xeon® Processors Technology Guide* (see Reference Documentation) and associated collateral to assist in the development and deployment of their Intel® SGX systems.

Appendix A REST API Endpoints

The KMRA REST API server allows remote access. It is an interface that allows the client to request keys from the key server to be more securely sent to the Intel SGX-enabled node running ctk_loakdey. The REST API requires the client to send a quote generated in the Intel SGX enclave, a public key from the enclave and a client certificate. The REST API uses mutual TLS to authenticate a client. The certificates for AppHSM and ctk_loadkey are generated by the same certificate authority (CA). The CA certificate is used to authenticate the client. The client certificate also has a Subject OUN field with a specific client ID. Key permissions for this client ID are outlined in the apphsm.conf configuration file. The mutual TLS certificates are not related to the wrapped keys provisioned to the client.

The following list provides a detailed key flow sequence:

- Client generates RSA key pair (rsa_pub, rsa_priv) out-of-band on service node (key server).
- Client generates customer public/private key pair (cust_pub/cust_priv) as a session object on the compute node. An attestation quote sgx_quote_t is generated using Crypto API Toolkit for Intel SGX and attests the hash of cust_pub and the enclave. Client sends REST API request containing a (cust_pub) to AppHSM to trigger the Intel SGX quote verification library.
- The quote provides proof to the service node that the client's enclave is running with Intel SGX protections on a trusted Intel SGX-enabled platform, with a valid TCB. If quote is correct, the hash of cust_pub is verified with sgx_quote_t, cust_pub is imported into the HSM as a session object, and a symmetric wrapping key (aes_swk) is generated as a session object.
- AppHSM creates wrapped_priv_key by wrapping RSA private key (rsa_priv) with SWK (aes_swk) using CKM_AES_KEY_WRAP_PAD. AppHSM also creates wrapped_swk by wrapping SWK (aes_swk) with the imported customer public key (cust_pub) using RSA OAEP. Wrapped keys wrapped_priv_key and wrapped_swk are returned to the client on the compute node.
- Client unwraps keys into Crypto API Toolkit for Intel SGX using cust_priv secured in the enclave. Cust_pub and cust_priv are destroyed after unwrapping when the session ends. OpenSSL on the client node is configured to retrieve protected key using Libp11 engine and provide it to NGINX workload.

The REST API has two URIs implemented.

1. To get the KMRA server version
   ```
   GET /sys/version
   ```
   - HTTPS 201 – OK
   - HTTPS 401 – Unauthorized

   Example command:
   ```
   curl https://localhost:5000/sys/version -X GET -k --cacert ./ca.crt -key
   ./ctk_loadkey.key --cert ./ctk_loadkey.crt
   ```
   Example response:
   ```
   { "version": "0.02", "api_version": "0.02_API"}
   ```
2. To verify quote and hash, import key, and return wrapped keys and the certificate
   ```
   POST /sgx/keys/export
   ```
   - HTTPS 201 – OK
   - HTTPS 401 – Unauthorized
   - HTTPS 400 – Bad Request
   - HTTPS 500 - Internal Server Error

   Example command:
   ```
   curl  https://localhost:5000/sgx/keys/export -X POST -k
   --cacert ./ca.crt --key ./ctk_loadkey.key --cert ./ctk_loadkey.crt
   -H "Content-Type: application/json" -d
   '{'HsmObject': {'h_unique_id': 'xxxx'}, 'SgxObject': {'RsaPublicKey':
   {'ExponentLen': 0, 'Exponent': 'xxxx', 'ModulusLen': 0,
   'Modulus': 'xxxx'}, 'SgxQuote': 'xxxx'}}'
   ```
   Example response:
   ```
   { "wrappedSWK": "L3NneC9rZXlzL2V4cG9ydCB3cmFwcGVkU1dL",
   "wrappedKey": "L3NneC9rZXlzL2V4cG9ydCB3cmFwcGVkS2V5",
   "certificate": "L3NneC9rZXlzL2V4cG9ydCB3cmFwcGVkSCrT"}
   ```

3. To verify quote and hash

```
POST /sgx/attest
```

- HTTPS 201 – OK
- HTTPS 401 – Unauthorized
- HTTPS 400 – Bad Request
- HTTPS 500 - Internal Server Error

Example command:

```
curl  https://localhost:5000/sgx/attest -X POST -k
--cacert ./ca.crt --key ./ctk_loadkey.key --cert ./ctk_loadkey.crt
-H "Content-Type: application/json" -d
'{'HsmObject': {'h_unique_id': 'xxxx'}, 'SgxObject': {'RsaPublicKey':
{'ExponentLen': 0, 'Exponent': 'xxxx', 'ModulusLen': 0,
'Modulus': 'xxxx'}, 'SgxQuote': 'xxxx'}}'
```

Example response:

```
{ "verified": true }
```

0323/DN/WIT/PDF 634677-007US