Contents

1 Introduction .................................................................................................................. 9

2 Audience ..................................................................................................................... 10
   2.1 Trusted Computing................................................................................................. 10
   2.1.1 The Chain of Trust ......................................................................................... 10
   2.1.2 Hardware Root of Trust .................................................................................. 10
   2.1.3 Supported Trusted Boot Options ..................................................................... 11
   2.1.4 Remote Attestation ......................................................................................... 12
   2.2 Intel® Security Libraries for Datacenter Features ............................................. 13
       2.2.1 Platform Integrity ......................................................................................... 13
       2.2.2 Data Sovereignty ......................................................................................... 13
       2.2.3 Application Integrity .................................................................................... 13
       2.2.4 Workload Confidentiality for Virtual Machines and Containers ............. 13
       2.2.5 Signed Flavors ............................................................................................. 14

3 Deployment Architecture ............................................................................................ 15

4 Platform Configuration ............................................................................................... 16

5 Use Case 1a: Platform Attestation (Linux*) ............................................................. 18
   5.1 Scope ..................................................................................................................... 18
   5.2 Requirements ......................................................................................................... 18
       5.2.1 Hardware ....................................................................................................... 18
       5.2.2 Software ........................................................................................................ 19
       5.2.3 Operating System ......................................................................................... 19
       5.2.4 Networking .................................................................................................... 19
   5.3 Installation .............................................................................................................. 20
       5.3.1 Installing the Database Server ....................................................................... 20
       5.3.2 Installing the Certificate Management Service .......................................... 20
       5.3.3 Installing the Authentication and Authorization Service ............................ 22
       5.3.4 Installing the Verification Service .................................................................. 25
       5.3.5 Installing the Trust Agent for Linux ............................................................. 26
   5.4 Demonstrating the Platform Integrity Use Case .................................................. 28
       5.4.1 Verify Services ............................................................................................... 28
       5.4.2 Register Hosts ............................................................................................... 28
       5.4.3 List Hosts ....................................................................................................... 29
       5.4.4 Retrieve Reports (Untrusted – No Flavors) .................................................... 29
       5.4.5 Import Flavors ............................................................................................... 29
       5.4.6 Import the default SOFTWARE Flavor ......................................................... 29
       5.4.7 Retrieve Reports (Trusted) ............................................................................. 30
       5.4.8 Demonstrate Untrusted State ........................................................................ 30
       5.4.9 Returning the hosts to a Trusted state ......................................................... 33

6 Use Case 1b: Platform Attestation (Windows*) ......................................................... 35
   6.1 Scope ..................................................................................................................... 35
   6.2 Requirements ......................................................................................................... 35
       6.2.1 Hardware ....................................................................................................... 35
       6.2.2 Software ........................................................................................................ 36
6.2.3 Operating System ............................................................ 36
6.2.4 Networking ..................................................................... 36
6.3 Installation ........................................................................... 36
   6.3.1 Installing the Database Server ..................................... 36
   6.3.2 Installing the Certificate Management Service .......... 37
   6.3.3 Installing the Authentication and Authorization Service 38
   6.3.4 Installing the Verification Service .............................. 42
   6.3.5 Installing the Trust Agent for Windows ..................... 43
6.4 Demonstrating the Platform Attestation Use Case ............... 44
   6.4.1 Verify Services ......................................................... 44
   6.4.2 Register Hosts .......................................................... 44
   6.4.3 List Hosts ................................................................ 45
   6.4.4 Retrieve Reports (Untrusted – No Flavors) .................. 45
   6.4.5 Import Flavors ......................................................... 45
   6.4.6 Import the default SOFTWARE Flavor .................... 45
   6.4.7 Retrieve Reports (Trusted) ....................................... 46
   6.4.8 Demonstrate Untrusted State ................................. 46
   6.4.9 Returning the hosts to a Trusted state ....................... 49

7 Use Case 1c: Platform Attestation (VMware*) ....................... 50
   7.1 Scope ........................................................................... 50
   7.2 Requirements ............................................................. 50
      7.2.1 Hardware ............................................................... 50
      7.2.2 Software ............................................................... 51
      7.2.3 Operating System ................................................ 51
   7.3 Installation ..................................................................... 51
      7.3.1 Installing the Database Server ............................. 51
      7.3.2 Installing the Certificate Management Service .... 52
      7.3.3 Installing the Authentication and Authorization Service 53
      7.3.4 Installing the Verification Service ....................... 57
   7.4 Demonstrating the Platform Attestation Use Case .......... 58
      7.4.1 Verify Services ..................................................... 58
      7.4.2 Register Hosts ....................................................... 58
      7.4.3 List Hosts ............................................................ 59
      7.4.4 Retrieve Reports (Untrusted – No Flavors) ............. 59
      7.4.5 Import Flavors ....................................................... 59
      7.4.6 Retrieve Reports (Trusted) ................................. 60
      7.4.7 Demonstrate Untrusted State ............................. 60
      7.4.8 Returning the hosts to a Trusted state ................. 63

8 Use Case 2a: Asset Tags (Linux) ........................................... 64
   8.1 Scope ........................................................................... 64
   8.2 Requirements ............................................................. 64
   8.3 Demonstrating the Asset Tag Use Case for Linux .......... 64
      8.3.1 Verify that hosts are registered and Trusted ........ 64
      8.3.2 Create Asset Tag Certificates for each host .......... 64
      8.3.3 Deploy Asset Tags ............................................... 65
      8.3.4 Create new Attestation Reports for each host ....... 66

9 Use Case 2b: Asset Tags (Windows) .................................... 67
   9.1 Scope ........................................................................... 67
   9.2 Requirements ............................................................. 67
9.3 Demonstrating the Asset Tag Use Case for Windows.................................67
  9.3.1 Verify that hosts are registered and Trusted ..................................67
  9.3.2 Create Asset Tag Certificates for each host ..................................67
  9.3.3 Deploy Asset Tags .......................................................................68
  9.3.4 Create new Attestation Reports for each host ..............................69

10 Use Case 2c: Asset Tags (VMware) ............................................................70
  10.1 Scope ..............................................................................................70
  10.2 Requirements ..................................................................................70
  10.3 Demonstrating the Asset Tag Use Case for VMware .......................70
       10.3.1 Verify that hosts are registered and Trusted ...........................70
       10.3.2 Create Asset Tag Certificates for each host .........................70
       10.3.3 Deploy Asset Tags ..................................................................71
       10.3.4 Create Asset Tag Flavor .......................................................72
       10.3.5 Create new Attestation Reports for each host .......................72

11 Use Case 3: Application Integrity .............................................................73
  11.1 Scope ..............................................................................................73
  11.2 Requirements ..................................................................................73
  11.3 Demonstrating the Application integrity Use Case for Linux ............73
       11.3.1 Verify that hosts are registered and Trusted ...........................73
       11.3.2 Create a New SOFTWARE Flavor ...........................................73
       11.3.3 Deploy the Application Manifest ..........................................74
       11.3.4 Create new Attestation Reports for each host .......................74

12 Use Case 4: OpenStack Orchestration* ....................................................76
  12.1 Scope ..............................................................................................76
  12.2 Requirements ..................................................................................76
       12.2.1 Hardware ...............................................................................76
       12.2.2 Software ...............................................................................77
       12.2.3 Operating System .................................................................77
       12.2.4 Networking ...........................................................................77
       12.2.5 OpenStack ............................................................................77
       12.2.6 Installation ............................................................................78
  12.3 Demonstrating the OpenStack Orchestration* Use Case ....................79
       12.3.1 Configuring a Tenant in the Integration Hub ...........................79
       12.3.2 Setting Launch Requirements in Image Traits .......................80
       12.3.3 Launching an Instance on a Compliant Node ..........................81
       12.3.4 Launching an Instance where No Nodes are Compliant ..........82

13 Use Case 5: Kubernetes Orchestration ......................................................83
  13.1 Scope ..............................................................................................83
  13.2 Requirements ..................................................................................84
       13.2.1 Hardware ...............................................................................84
       13.2.2 Operating System .................................................................84
       13.2.3 Networking ...........................................................................84
       13.2.4 Kubernetes ............................................................................84
       13.2.5 Installation ............................................................................85
  13.3 Demonstrating the Kubernetes Scheduling Use Case .........................87
       13.3.1 Configuring a Tenant in the Integration Hub .........................87
       13.3.2 Configuring Pods to Require Intel® ScL Attributes ..................89
       13.3.3 Launching a Pod on a Compliant Worker Node .....................92
13.3.4 Launching a Pod where No Worker Nodes are Compliant ...... 92

14 Use Case 6a: Workload Confidentiality with VMs .............................. 94
14.1 Scope ......................................................................................................... 94
14.2 Requirements ....................................................................................... 94
  14.2.1 Hardware .................................................................................. 94
  14.2.2 Software ................................................................................ 94
  14.2.3 Operating System ................................................................. 95
  14.2.4 Networking .......................................................................... 95
  14.2.5 OpenStack ........................................................................... 95
14.3 Installation ............................................................................................. 95
  14.3.1 Installing the Database Server ................................................ 95
  14.3.2 Installing the Certificate Management Service ......................... 96
  14.3.3 Installing the Authentication and Authorization Service ........... 97
  14.3.4 Installing the Verification Service ........................................... 101
  14.3.5 Installing the Workload Service ............................................. 102
  14.3.6 Installing the Integration Hub ................................................. 103
  14.3.7 Installing the Key Broker Service ........................................... 104
  14.3.8 Installing the Workload Policy Manager .................................. 106
  14.3.9 Installing the Trust Agent and Workload Agent for Linux ...... 107
14.4 Preparing Platform Integrity ............................................................... 109
14.5 OpenStack Orchestration ..................................................................... 110
14.6 Demonstrating Workload Confidentiality with VMs ............................ 111
  14.6.1 Encrypting an Image .................................................................. 111
  14.6.2 Uploading the Image Flavor ...................................................... 111
  14.6.3 Creating the Image Flavor to Image ID Association .................... 111
  14.6.4 Demonstrate Launching Encrypted VMs on Trusted Hosts ....... 112
  14.6.5 Demonstrate Launching Encrypted VMs on Untrusted Hosts ..... 112

15 Use Case 6b: Workload Confidentiality with Docker Containers ...... 114
15.1 Scope ........................................................................................................ 114
15.2 Requirements .......................................................................................... 114
  15.2.1 Hardware ................................................................................ 114
  15.2.2 Software ................................................................................. 114
  15.2.3 Operating System ................................................................. 115
  15.2.4 Networking .......................................................................... 115
  15.2.5 Kubernetes ............................................................................. 115
15.3 Installation ............................................................................................... 115
  15.3.1 Installing the Database Server ................................................ 115
  15.3.2 Installing the Certificate Management Service ......................... 116
  15.3.3 Installing the Authentication and Authorization Service ........... 117
  15.3.4 Installing the Verification Service ........................................... 121
  15.3.5 Installing the Workload Service ............................................. 122
  15.3.6 Installing the Intel® SecL-DC Custom Resource Definitions for
      Kubernetes ............................................................................... 123
  15.3.7 Installing the Integration Hub ................................................. 124
  15.3.8 Installing the Key Broker Service ........................................... 125
  15.3.9 Installing the Workload Policy Manager .................................. 127
  15.3.10 Installing the Trust Agent and Workload Agent for Linux ..... 128
  15.3.11 Preparing Platform Integrity .................................................. 130
15.4 Kubernetes Integration ........................................................................... 132
15.4.1 Configuring a Tenant in the Integration Hub ...................... 132
15.4.2 Configuring Pods to Require Intel® Secl Attributes .......... 134
15.4.3 Launching a Pod (unencrypted) on a Compliant Worker Node .......................................................... 136
15.5 Demonstrating Workload Confidentiality with Docker Containers .... 136
15.5.1 Encrypting an Image ...................................................... 136
15.5.2 Uploading the Image Flavor ........................................... 136
15.5.3 Creating the Image Flavor to Image ID Association ............ 137
15.5.4 Demonstrate Launching Encrypted Docker Containers on Trusted Hosts ....................................................... 137
15.5.5 Demonstrate Launching Encrypted VMs on Untrusted Hosts. 137
<table>
<thead>
<tr>
<th>Document Number</th>
<th>Revision Number</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td></td>
<td>• Initial release of the document.</td>
<td>January 2019</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
<td>Updated as per Intel SecL-DC v1.5</td>
<td>Aug 2019</td>
</tr>
<tr>
<td>1.6</td>
<td>1.6</td>
<td>Updated as per Intel SecL-DC v1.6</td>
<td>Dec 2019</td>
</tr>
</tbody>
</table>
1 Introduction

This document defines the key prerequisites, requirements, scope and process for enabling Intel® Security Libraries for Data Center (Intel® SecL - DC) use cases in a Proof of Concept setting.
2 Audience

This document is intended for Systems Engineers who intend to complete a POC for one or more use cases for Intel® SecL – DC.

When using this document, be sure to read the sections for each use case you intend to enable as part of the POC to ensure that all requirements are met. Many use cases build upon prerequisite use cases, but add additional requirements that must be considered before starting a deployment.

2.1 Trusted Computing

Trusted Computing consists of a set of industry standards defined by the Trusted Computing Group to harden systems and data against attack. These standards include verifying platform integrity, establishing identity, protection of keys and secrets, and more. One of the functions of Intel Security Libraries is to provide a “Trusted Platform,” using Intel security technologies to add visibility, auditability, and control to server platforms.

2.1.1 The Chain of Trust

In a Trusted Computing environment, a key concept is verification of the integrity of the underlying platform. Verifying platform integrity typically means cryptographic measurement and/or verification of firmware and software components. The process by which this measurement and verification takes place affects the overall strength of the assertion that the measured and verified components have not been altered. Intel refers to this process as the “Chain of Trust,” whereby at boot time, a sequence of cryptographic measurements and signature verification events happen in a defined order, such that measurement/verification happens before execution, and each entity responsible for performing a measurement or verification is measured by another step earlier in the process. Any break in this chain leads to an opportunity for an attacker to modify code and evade detection.

2.1.2 Hardware Root of Trust

The Root of Trust, the first link in the chain, can be one of several different options. Anything that happens in the boot process before the Root of Trust must be considered to be within the “trust boundary,” signifying components whose trustworthiness cannot be assessed. For this reason, it’s best to use a Root of Trust that starts as early in the system boot process as possible, so that the Chain of Trust during the boot process can cover as much as possible.

Multiple Root of Trust options exist, ranging from firmware to hardware. In general, a hardware Root of Trust will have a smaller “trust boundary” than a firmware Root of Trust. A hardware Root of Trust will also have the benefit of immutability – where firmware can easily be flashed and modified, hardware is much more difficult to tamper with.
2.1.2.1 **Intel® Trusted Execution Technology (Intel® TXT)**

Intel® Trusted Execution Technology is a hardware Root of Trust feature available on Intel® server platforms starting with the Grantley generation. Intel® TXT is enabled in the system BIOS (typically under the Processor > Advanced tab), and requires Intel® VT-d and Intel VT-x features to be enabled as prerequisites (otherwise the option will be grayed out). Intel® TXT will ship "disabled" by default.

2.1.2.2 **Intel® BootGuard (Intel® BtG)**

Intel® BootGuard is a hardware Root of Trust feature available on Intel® server platforms starting with the Purley-Refresh generation. Unlike Intel® TXT, Intel® BtG is configured in platform fuses, not in the system BIOS. Intel® BtG is fused into several “profiles” that determine the behavior of the feature. Intel® BtG supports both “verify” and “measure” profiles; in “verify” profiles, Intel® BtG will verify the signature of the platform Initial Boot Block (IBB). In “measure” profiles, Intel® BtG will hash the IBB and extend that measurement to a TPM PCR. It is recommended that Intel® BtG be fused into the “measure and verify” profile for maximum protection and auditability.

Because the Intel® BtG profile is configured using fuses, the server OEM/ODM will determine the profile used at manufacturing time. Please contact your server vendor to determine what Intel® BtG profiles are available in their product line.

Because Intel® BtG only measures/verifies the integrity of the IBB, it’s important to have an additional technology handle measurements later in the boot process. Intel® TXT can provide this function using tboot to invoke SINIT, and UEFI SecureBoot can alternatively provide similar functionality (note that Linux users should properly configure Shim and use a signed kernel for UEFI SecureBoot).

2.1.3 **Supported Trusted Boot Options**

Intel® SecL-DC supports several options for Trusted Computing, depending on the features available on the platform.
Note: Intel® recommends using UEFI boot mode, and either TXT + tboot, or TXT + BtG + UEFI SecureBoot, or BtG + UEFI SecureBoot. These options will avoid any incompatibilities and provide the best measured boot coverage. Take note that tboot is not currently compatible with UEFI SecureBoot.

2.1.4 Remote Attestation

Trusted computing consists primarily of two activities – measurement, and attestation. Measurement is the act of obtaining cryptographic representations for the system state. Attestation is the act of comparing those cryptographic measurements against expected values to determine whether the system booted into an acceptable state.

Attestation can be performed either locally, on the same host that is to be attested, or remotely, by an external authority. The trusted boot process can optionally include a local attestation involving the evaluation of a TPM-stored Launch Control Policy (LCP). In this case, the host’s TPM will compare the measurements that have been taken so far to a set of expected PCR values stored in the LCP; if there is a mismatch, the boot process is halted entirely.

Intel® SecL utilizes remote attestation, providing a remote Verification Service that maintains a database of expected measurements (or “flavors”), and compares the actual boot-time measurements from any number of hosts against its database to provide an assertion that the host booted into a “trusted” or “untrusted” state. Remote attestation is typically easier to centrally manage (as opposed to creating an LCP for each host and entering the policy into the host’s TPM), does not halt the boot process allowing for easier remediation, and separates the attack surface into separate components that must both be compromised to bypass security controls.

Both local and remote attestation can be used concurrently. However, Intel® SecL, and this document, will focus only on remote attestation. For more

2.2 **Intel® Security Libraries for Datacenter Features**

2.2.1 **Platform Integrity**

Platform Integrity is the use case enabled by the specific implementation of the Chain of Trust and Remote Attestation concepts. This involves the use of a Root of Trust to begin an unbroken chain of platform measurements at server boot time, with measurements extended to the Trusted Platform Module and compared against expected values to verify the integrity of measured components. This use case is foundational for other Intel® SecL use cases.

2.2.2 **Data Sovereignty**

Data Sovereignty builds on the Platform Integrity use case to allow physical TPMs to be written with Asset Tags containing any number of key/value pairs. This use case is typically used to identify the geographic location of the physical server, but can also be used to identify other attributes. For example, the Asset Tags provided by the Data Sovereignty use case could be used to identify hosts that meet specific compliance requirements and can run controlled workloads.

2.2.3 **Application Integrity**

Added in the Intel® SecL-DC 1.5 release, Application Integrity allows any files and folders on a Linux host system to be included in the Chain of Trust integrity measurements. These measurements are attested by the Verification Service along with the other platform measurements, and are included in determining the host’s overall Trust status. The measurements are performed by a measurement agent called tbootXM, which is built into initrd during Trust Agent installation. Because initrd is included in other Trusted Computing measurements, this allows Intel® SecL-DC to carry the Chain of Trust all the way to the Linux filesystem.

2.2.4 **Workload Confidentiality for Virtual Machines and Containers**

Added in the Intel® SecL-DC 1.6 release, Workload Confidentiality allows virtual machine and Docker container images to be encrypted at rest, with key access tied to platform integrity attestation. Because security attributes contained in the platform integrity attestation report are used to control access to the decryption keys, this feature provides both protection for at-rest data, IP, code, etc in Docker container or virtual machine images, and also
enforcement of image-owner-controlled placement policies. When decryption keys are released, they are sealed to the physical TPM of the host that was attested, meaning that only a server that has successfully met the policy requirements for the image can actually gain access.

Workload Confidentiality begins with the Workload Policy Manager (WPM) and a qcow2 or Docker image that needs to be protected. The WPM is a lightweight application that will request a new key from the Key Broker, use that key to encrypt the image, and generate an Image Flavor. The image owner will then upload the encrypted image to their desired image storage service (for example, OpenStack Glance or a local Docker Registry), and the image ID from the image storage will be uploaded along with the Image Flavor to the Intel® SecL Workload Service. When that image is used to launch a new VM or container, the Workload Agent will intercept the VM or container start and request the decryption key for that image from the Workload Service. The Workload Service will use the image ID and the Image Flavor to find the key transfer URL for the appropriate Key Broker, and will query the Verification Service for the latest Platform Integrity trust attestation report for the host. The Key Broker will use the attestation report to determine whether the host meets the policy requirements for the key transfer, and to verify that the report is signed by a Verification Service known to the Broker. If the report is genuine and meets the policy requirements, the image decryption key is sealed using an asymmetric key from that host’s TPM, and sent back to the Workload Service. The Workload Service then caches the key for 5 minutes (to avoid performance issues for multiple rapid launch requests; note that these keys are still wrapped using a sealing key unique to the host’s TPM, so multiple hosts would require multiple keys even for an identical image) and return the wrapped key to the Workload Agent on the host, which then uses the host TPM to unseal the image decryption key. The key is then used to create a new LUKS volume, and the image is decrypted into this volume.

This functionality means that a physical host must pass policy requirements in order to gain access to the image key, and the image will be encrypted at rest both in image storage and on the compute host.

2.2.5 Signed Flavors

Added in the Intel® SecL-DC 1.6 release, Flavor signing is an improvement to the existing handling of expected attestation measurements, called “Flavors.” This feature adds the ability to digitally sign Flavors so that the integrity of the expected measurements themselves can be verified when attestations occur. This also means that Flavors can be more securely transferred between different Verification Service instances.

Flavor signing is seamlessly added to the existing Flavor creation process (both importing from a sample host and “manually” creating a Flavor using the POST method to the /v2/flavors resource). When a Flavor is created, the Verification Service will sign it using a signing certificate signed by the Certificate Management Service (this is created during Verification Service setup). Each time that the Verification Service evaluates a Flavor, it will first verify the signature on that Flavor to ensure the integrity of the Flavor contents before it is used to attest the integrity of any host.
3 Deployment Architecture

For simplicity, Intel® SecL – DC POC deployments should include one to three physical server platforms with TPMs installed and Intel® TXT activated, as well as one additional server (physical or virtual) for deployment of the services other than the Trust Agent.

This includes (as applicable – the list differs based on the use cases being enabled in the POC) the Intel® SecL – DC Verification Service, Integration Hub, Authentication and Authorization Service, Certificate Management Service, Key Broker Service, Workload Service, and any applicable OpenStack* (Nova*, Glance*, Horizon*, Keystone*, and so forth) or Kubernetes* services. This document assumes that, for this POC, all of the Intel® SecL-DC management services will be installed on a single physical or virtual machine.

Each physical server with a TPM should be installed with (as applicable for the appropriate use case) the Trust Agent, the Workload Agent, and any compute node components for OpenStack or Kubernetes.
Each of the Intel® SecL-DC use cases covered in this document require that all TPM-enabled physical servers must have a supported combination of hardware technologies enabled for Trusted Boot. Generally this means using one or both of Intel® TXT and or Boot Guard (Boot Guard must be in a “measurement” profile). Note that Boot Guard is enabled by the server OEM and is configured in fuses, and is not an option in the BIOS that can be enabled or disabled manually. TXT is enabled in the server BIOS.

Different configurations are needed depending on whether the physical server is installed using Legacy BIOS mode or UEFI mode, whether TXT is enabled, and whether UEFI Secure Boot will be enabled.

TXT and Boot Guard represent the “static Root of Trust” measurements. Tboot and UEFI Secure Boot represent the “dynamic Root of Trust” measurements. Both Roots of Trust are required for Intel® SecL-DC.

Tboot is the “normal” Dynamic RoT utility for TXT. However, tboot is currently incompatible with UEFI Secure Boot. For servers with TXT enabled, if you enable UEFI Secure Boot, tboot will not be used. If UEFI Secure Boot is not enabled, the Trust Agent will automatically install and configure tboot.

Tboot will not perform any function unless TXT is enabled. If Boot Guard will be the only Static RoT with TXT disabled, you must enable UEFI Secure Boot to act as the Dynamic RoT.

If both TXT and Boot Guard are enabled, you may choose whether to use UEFI Secure Boot or tboot as the Dynamic RoT. Due to incompatibility, both may not be used.

See the chart below:
**Note:** Intel® recommends using UEFI boot mode, and either TXT + tboot, or TXT + BtG + UEFI SecureBoot, or BtG + UEFI SecureBoot. These options will avoid any incompatibilities and provide the best measured boot coverage. Take note that tboot is not currently compatible with UEFI SecureBoot.
The Platform Attestation use case will enable external platform integrity verification using TCG-standard practices. This feature is the foundation for other Intel® SecL – DC use cases.

Platform Attestation involves taking measurements of system components during system boot and then cryptographically verifying that the actual measurements taken match a set of expected or approved values, ensuring that the measured components were in an acceptable or "trusted" state at the time of the last system boot.

Intel® SecL – DC leverages the Trusted Compute Group specification for a trusted boot process, extending measurements of platform components to registers in a Trusted Platform Module (TPM) and securely generating quotes of those measurements from the TPM for remote comparison to expected values (attestation).

The deployment will include the Verification Service and one or more Trust Agents. The Verification Service is typically installed on a Virtual Machine, while the Trust Agent must be installed on each physical server.

5.1 Scope

This use case will include the installation of the Verification Service and one or more Trust Agent compute nodes. After installation, the Platform Attestation use case will be demonstrated by registering the Trust Agent hosts, importing Flavors from the registered hosts, retrieving Attestation Reports for the registered hosts and performing a negative test to demonstrate an “Untrusted” result.

Note: The Workload Confidentiality use cases (Use Cases 6a and 6b) require a slightly different installation flow. If the Workload Confidentiality use cases are desired, skip ahead to Use Case 6a or 6b. After walking through the Workload Confidentiality use case, you can return to this section and proceed immediately after the installation (pick back up at section 3.4).

5.2 Requirements

5.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration
- The system must either have Intel® TXT supported and enabled in the system BIOS, OR must have Intel® Boot Guard supported in a “measurement” profile
- If Intel® Boot Guard is used but Intel® TXT is disabled, UEFI Secure Boot must be enabled

- Each physical server must have a physical or firmware Trusted Platform Module installed and active in the system BIOS. TPM 2.0 is supported for Linux*. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS).
- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require a TPM or Intel® TXT or Intel® Boot Guard, but does require network access to the other POC servers.

### 5.2.2 Software
- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC* driver (for example, SQL Workbench*) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL* or a REST* API utility like Postman* to execute API requests.

### 5.2.3 Operating System
- The Intel® SecL – DC Verification Service supports Red Hat Enterprise* Linux (RHEL*) 7.6.
- The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.6.

### 5.2.4 Networking
- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC installers will require access to package repositories; either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.
5.3  **Installation**

The Platform Attestation use case requires four Intel® SecL – DC services: the Certificate Management Service, the Authentication and Authorization Service, the Verification Service and the Trust Agent.

5.3.1  **Installing the Database Server**

For the purposes of POCs, a single shared database server will be used for all of the services that require database access. Each service will use its own database schema.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Install a sample Postgresql 11 database using the install_pgdb.sh script. This script will automatically install the Postgresql database and client packages required.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Create the iseclpgdb.env answer file:

ISECL_PGDB_IP_INTERFACES=localhost
ISECL_PGDB_PORT=5432
ISECL_PGDB_SAVE_DB_INSTALL_LOG=true

Execute the installation script:

./install_pgdb.sh

**Note**: the database installation only needs to be performed once if the same database server will be used for all services that require a database. Only the “create_db” step need to be repeated if the database server will be shared.

After installation, the database schemas must be created initialized and tables created for each service.

./create_db.sh aas_db aas_db_user password
./create_db.sh mw_as vs_db_user password

5.3.2  **Installing the Certificate Management Service**

5.3.2.1  **Supported Operating Systems**

5.3.2.2 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

5.3.2.3 Installation

To install the Intel® SecL-DC Certificate Management Service:

2. Create the cms.env installation answer file for an unattended installation:

```
AAS_TLS_SAN=<comma-separated list of IPs and hostnames for the AAS>,127.0.0.1,localhost
SAN_LIST=<comma-separated list of IPs and hostnames for the CMS>,127.0.0.1,localhost
AAS_API_URL=https://<Authentication and Authorization Service IP or Hostname>:8444/aas
```

The SAN list will be used to authenticate the Certificate Signing Request from the AAS to the CMS. Only a CSR originating from a host matching the SAN list will be honored. Later, in the AAS authservice.env installation answer file, this same SAN list will be provided for the AAS installation. These lists must match, and must be valid for IPs and/or hostnames used by the AAS system. If both the AAS and CMS will be installed on the same system, “127.0.0.1,localhost” may be used. The SAN list variables also accept the wildcards “?” (for single-character wildcards) and “*” (for multiple-character wildcards) to allow address ranges or multiple FQDNs. The AAS_API_URL represents the URL for the AAS that will exist after the AAS is installed.

For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Certificate Management Service.

3. Execute the installer binary.

```
./certificate-management-service-1.0.bin
```

When the installation completes, the Certificate Management Service is available. The services can be verified by running `cms status` from the command line.

```
# cms status
```

After installation is complete, the CMS will output a bearer token to the console. This token will be used with the AAS during installation to authenticate certificate requests to the CMS. If this token expires or otherwise needs to be recreated, use the following command:

```
cms setup cms_auth_token --force
```
In addition, the SHA384 digest of the CMS TLS certificate will be needed for installation of the remaining Intel® SecL services. The digest can be obtained using the following command:

cms tlscertsha384

5.3.3 Installing the Authentication and Authorization Service

5.3.3.1 Required For

The AAS is REQUIRED for all use cases.
- Platform Integrity with Data Sovereignty and Signed Flavors
- Application Integrity
- Workload Confidentiality (both VMs and Docker Containers)

5.3.3.2 Prerequisites

The following must be completed before installing the Authentication and Authorization Service:

- The CMS must be installed and available

5.3.3.3 Supported Operating Systems


5.3.3.4 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

5.3.3.5 Installation

To install the AAS, a bearer token from the CMS is required. This bearer token is output at the end of the CMS installation. However, if a new token is needed, simply use the following command from the CMS command line:

cms setup cms_auth_token --force

Create the authservice.env installation answer file:
Note: the AAS_ADMIN credentials specified in this answer file will have administrator rights for the AAS and can be used to create other users, create new roles, and assign roles to users.

Execute the AAS installer:

./authservice-v1.6.bin

5.3.3.5.1 Creating Users

After installation is complete, a number of roles and user accounts must be generated. Most of these accounts will be service users, used by the various Intel® SeCL services to work together. Another set of users will be used for installation permissions, and a final administrative user will be created to provide the initial authentication interface for the actual human user. The administrative user can be used to create additional users with appropriately restricted roles based on organizational needs.

Creating these required users and roles is facilitated by a script that will accept credentials and some configuration settings from an answer file, and will automate the process.

Create the populate-users.env file:

```
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1/
CMS_TLS_CERT_SHA384=<CMS_TLS certificate sha384>
AAS_DB_HOSTNAME=localhost
AAS_DB_PORT=5432
AAS_DB_NAME=aas_db
AAS_DB_USERNAME=aas_db_user
AAS_DB_PASSWORD=password
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password
SAN_LIST=<comma-separated list of IPs and hostnames for the AAS>,127.0.0.1,localhost
BEARER_TOKEN=<bearer token from CMS installation>

Note: the AAS_ADMIN credentials specified in this answer file will have administrator rights for the AAS and can be used to create other users, create new roles, and assign roles to users.

Execute the AAS installer:

./authservice-v1.6.bin

5.3.3.5.1 Creating Users

After installation is complete, a number of roles and user accounts must be generated. Most of these accounts will be service users, used by the various Intel® SeCL services to work together. Another set of users will be used for installation permissions, and a final administrative user will be created to provide the initial authentication interface for the actual human user. The administrative user can be used to create additional users with appropriately restricted roles based on organizational needs.

Creating these required users and roles is facilitated by a script that will accept credentials and some configuration settings from an answer file, and will automate the process.

Create the populate-users.env file:

```
ISECL_INSTALL_COMPONENTS=TA,AH,VS,AAS,HVS
AAS_API_URL=https://127.0.0.1:8444/aas
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password
VS_CERT_SAN_LIST=* AH_CERT_SAN_LIST=* TA_CERT_SAN_LIST=* VS_SERVICE_USERNAME=verification_service
VS_SERVICE_PASSWORD=password
AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password
GLOBAL_ADMIN_USERNAME=admin GLOBAL_ADMIN_PASSWORD=password
```
**Note:** The **ISECL_INSTALL_COMPONENTS** variable is a comma-separated list of the components that will be used in your environment. Not all services are required for every use case. If a given service will not be used in your deployment, simply delete the unnecessary service abbreviation from the **ISECL_INSTALL_COMPONENTS** list, and leave the SAN and credential variables for that service blank.

**NOTE:** The SAN list variables each support wildcards ("*" and "?"), In particular, without wildcards the Trust Agent SAN list would need to explicitly list each hostname or IP address for all Trust Agents that will be installed, which is not generally feasible. Using wildcards, domain names and entire IP ranges can be included in the SAN list, which will allow any host matching those ranges to install the relevant service.

The **GLOBAL_ADMIN** credentials will be used to create an administrative user with all permissions.

Execute the populate-users script:

```
./populate-users
```

**Note:** The script can be executed with the --output_json argument to create the populate-user.json. This json output file will contain all of the users created by the script, along with usernames, passwords, and role assignments. This file can be used both as a record of the service and administrator accounts, and can be used as alternative inputs to recreate the same users with the same credentials in the future if needed. Be sure to protect this file if the --output_json argument is used.

The script will automatically generate the following users:

- Verification Service User
- Attestation Hub Service User
- Global Admin User

These user accounts will be used during installation of several of the Intel® SecL-DC applications. In general, whenever credentials are required by an installation answer file, the variable name should match the name of the corresponding variable used in the populate-user.env file.

The populate-users script will also output an installation token. The remaining Intel® SecL-DC services require this token (set as the “BEARER_TOKEN” variable in the installation env files) to grant the appropriate privileges for installation. By default this token will be valid for two hours; the populate-users script can be rerun with the same populate-users.env file to regenerate the token if more time is required.
5.3.3.6 Creating an Authentication Token

Intel® SecL-DC uses a Bearer Token authentication schema. Tokens are issued by the AAS and can be used as authentication for other services. By default, tokens are valid for 2 hours, after which a new token will be needed.

To issue a new token, use the following API call:

```plaintext
POST https://<AAS IP or hostname>:8444/aas/token
{
    "username" : "admin",
    "password" : "password"
}
```

For the purposes of this document, it is easiest to simply use the default Administrator password for all API requests. In a production environment it would be strongly recommended to create users with more restrictive roles based on the access needed.

Use this token as authentication for all API requests in this document. For example, use a header that looks like the following:

```
"Authorization: Bearer <token content>"
```

5.3.4 Installing the Verification Service

5.3.4.1 Package Dependencies

The Intel® Security Libraries Verification Service requires the following packages and their dependencies:

- Monit
- Logback (optional)
- Java* 8 JDK
- OpenSSL
- Postgres* client and server 11 (server component optional if an external Postgres database is used)
- Unzip

If they are not already installed, the Verification Service installer attempts to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

5.3.4.2 Supported Operating Systems

5.3.4.3 **Recommended Hardware**

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers
- (Optional) One network interface for Asset Tag provisioning (only required for "pull" tag provisioning; required to provision Asset Tags to VMware ESXi servers).

5.3.4.4 **Installation**

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file for an unattended installation.

   ```
   MTWILSON_SERVER=<IP address or hostname of the Verification Service>
   MTWILSON_API_URL="https://<Verification Service IP or hostname>:8443/mtwilson/v2"
   DATABASE_HOSTNAME=localhost
   DATABASE_USERNAME=vs_db_user
   DATABASE_PORTNUM="5432"
   DATABASE_PASSWORD=password
   DATABASE_SCHEMA="mw_as"
   VS_SERVICE_USERNAME=verification_service
   VS_SERVICE_PASSWORD=password
   CMS_TLS_CERT_SHA384=<Certificate Management Service TLS digest>
   BEARER_TOKEN=<Installation user token>
   AAS_API_URL=https://127.0.0.1:8444/aas
   CMS_BASE_URL=https://127.0.0.1:8445/cms/v1
   VS_TLS_CERT_DNS=<comma separated list of hostnames for the Verification Service>
   VS_TLS_CERT_IP=<comma separated list of IP addresses for the Verification Service>
   ```

   Execute the installer binary.

   ```
   ./host-verification-service-linux-4.6.bin
   ```

   When the installation completes, the Verification Service is available. The services can be verified by running `mtwilson status` from the Verification Service command line.

   ```
   # mtwilson status
   ```

5.3.5 **Installing the Trust Agent for Linux**

The Intel® SecL – DC Trust Agent must be installed on each physical TPM/Intel® TXT-enabled server that will be registered and attested.
5.3.5.1 Package Dependencies

The Trust Agent requires the following packages and their dependencies:

- trousers
- tboot
- tpm-quote-tools
- tpm2-tool
- openssl

If they are not already installed, the Trust Agent installer will attempt to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection.

If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

5.3.5.2 Supported Operating Systems

The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.4 and above.

5.3.5.3 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with Intel® TXT activated in the system BIOS
- TPM (version 1.2 or 2.0) installed and activated in the system BIOS, with cleared ownership status
- System must be booted to a tboot boot option (Trust Agent installation will automatically install tboot if not present, and then require a reboot before proceeding)
- (Provisioning step only) Intel® SecL – DC Verification Service server installed and active

5.3.5.4 Installation

To install the Trust Agent for Linux:

3. Create the trustagent.env answer file in the /root/ directory:
4. Copy the Trust Agent installer binary to the /root/ directory.

5. Execute the Trust Agent installer and wait for the installation to complete.
   - The Trust Agent will install tboot and other prerequisites if not already present. Tboot will **not** be installed if the server is booted using UEFI SecureBoot, due to incompatibility.
   - If tboot is installed by the Trust Agent installer, the installation will abort and reboot the host. This is because the Trust Agent requires the host to be booted into a tboot boot option, which populates the OS-level measurements in the host TPM.
   - After the host reboots, re-run the Trust Agent installation binary to resume the installation.

6. Reboot the server to the “TCB” boot option (should automatically be selected as the default) to populate the TPM with the Trust Agent component measurements.

### 5.4 Demonstrating the Platform Integrity Use Case

#### 5.4.1 Verify Services

The Version API can be used to confirm that all services are up and running before proceeding to the actual use case demonstration.

GET https://verification.server.com:8443/mtwilson/v2/version

GET https://trustagent.server.com:1443/v2/version

#### 5.4.2 Register Hosts

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

Each Trust Agent host will need to be registered with a separate call.

POST https://verification.service.com:8443/mtwilson/v2/hosts

```json
{
    "host_name": "<hostname of host to be registered>"
}
```
5.4.3 **List Hosts**

After registration, the `/hosts` API can be used to list all registered hosts and confirm that the registrations were all successful.


5.4.4 **Retrieve Reports (Untrusted – No Flavors)**

You can now retrieve the latest Attestation Report for each host to demonstrate the “faults” shown when match Flavors do not exist in the Verification Service database. Currently no Flavors should be present, so all of these Reports will show an Untrusted status, with Faults showing that matching Flavors were required but not found.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

5.4.5 **Import Flavors**

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (BIOS, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host; the BIOS and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors

```json
{
    "connection_string": "https://trustagent.server.com:1443",
    "partial_flavor_types": ["PLATFORM" , "OS", "HOST_UNIQUE"],
    "flavorgroup_name": "",
    "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

5.4.6 **Import the default SOFTWARE Flavor**

The SOFTWARE Flavor is required for Linux and Windows servers. By default, a manifest for the Trust Agent components is pushed to the Trust Agent during installation; after rebooting to the TCB boot mode, this Flavor can now be imported. Importantly, this Flavor part should only be imported once – the
default SOFTWARE Flavor has a unique label for each version, and requests to import the SOFTWARE Flavor for a version of the Trust Agent that already exists will throw an error. For example, if you have 3 Trust Agent servers, you may import the PLATFORM, OS, and HOST_UNIQUE Flavors from each of them, but the SOFTWARE Flavor should only be imported once.

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
    "connection_string": "https://trustagent.server.com:1443; ",
    "partial_flavor_types": ["SOFTWARE"],
    "flavorgroup_name": "",
    "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

5.4.7 Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, you can retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

This is the same request used previously, but this time all of the Reports should show that the “Overall” trust is “True,” and we should no longer see any Faults.
GET
https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json

5.4.8 Demonstrate Untrusted State

Finally, it is possible to demonstrate what an Untrusted attestation looks like and what can cause it. There are a number of ways to force a host to attest as Untrusted.

In production, a host can become Untrusted if the BIOS or OS kernel are upgraded/downgraded to use a version where no Flavor exists, if a malicious attacker has inserted malicious code into a measured component, or so forth.

For testing purposes, the easiest ways to force a host to become Untrusted are to either boot the host to a non-tboot boot option (simply reboot, and at the boot menu option when RHEL boots, select the option without tboot).

Attestation Reports are valid for 24 hours by default (this is configurable); you need to force a new Report to be generated now, after the previous change:
POST https://verification.server.com:8443/mtwilson/v2/reports
Headers:
Accept: application/json
input: {"host_name": "<Trust Agent host>"}
The new Report will show that the BIOS Flavor part is still Trusted, but that the HOST_UNIQUE and OS Flavor Parts have mismatched measurements. This is because the platform performs the BIOS-related measurements before the OS boots, but tboot performs the other measurements.

Because the system booted without tboot, the TPM memory registers that store the tboot measurements will all be 0’s or F’s, and will mismatch the expected measurements in the Flavors.

5.4.8.1 (Optional) Database Modification

While slightly more complicated, it is also possible to edit one or more Flavors in the database and change the expected measurement results. Because of the size of the Flavor objects in the database, this is best performed with a remote GUI database client like SQL Workbench, as opposed to using the command line. This step is optional, and will show a different Fault more consistent with a platform that has been tampered with by a malicious attack.

First you need to enable remote access to the Verification Service database, which is disabled by default.

In the /usr/local/pgsql/data/postgresql.conf file, add the following line:

```
listen_addresses='*' 
```

Next, edit /usr/local/pgsql/data/pg_hba.conf and change the last line to the following:

```
host all all 127.0.0.1/0 password 
```

Finally, restart the Postgresql database:

```
systemctl restart postgresql-11 
```

Now you should be able to access the database using a remote database client, like SQL Workbench. By default, the database name is mw_as. The username and password were configured in the mtwilson.env answer file during the installation; the example used “root” and “password”.

After connecting to the mw_as database, look at the mw_flavor table. You should see a number of Flavors, including BIOS, OS, and HOST_UNIQUE Flavors from when they were imported in an earlier step.

In the “content” column is the actual Flavor object. Find a “BIOS” Flavor:

```
{"id":006b9c2b-4ab3-496f-8b8d-0f5c32f4393f,"content":null} 
```

Find the “pcr_0” measurements (depending on the TPM version and your platform OEM, you may see only SHA1, only SHA256, or both measurements).
Change a digit in each of these entries. Do not change the length, simply change the last digit of each “pcr_0” hash to a different hexadecimal digit.

Save the changes to the database.

Next, we need to change the Verification Service settings to skip Flavor signature validation. By default, the HVS will verify the signature on each Flavor used in an attestation; since we are directly modifying a Flavor, that signature validation will fail, and create a different fault from the one we’re trying to demonstrate (a Flavor signature validation fault would say: “Signature is not trusted for flavor with id <Flavor ID>”).

```
mtwilson config skip.flavor.signature.verification true
```

```
mtwilson restart
```

Now force the creation of a new Report.

```
POST https://verification.server.com:8443/mtwilson/v2/reports
{
    "host_name": "<Trust Agent hostname>",
}
```

Do this for each host in the POC environment. When you get the Report for the host that was using the modified BIOS flavor, the OS and HOST_UNIQUE attestations will remain Trusted.

The BIOS Flavor part, however, will show that it is Untrusted, with a “PcrValueMismatch” fault indicating that the actual measurement for PCR 0 does not match the expected value.

Because PCR 0 includes the hash of the core system BIOS, this replicates an event where the BIOS has been maliciously modified.

```
<BIOS>
    <trust>false</trust>
    <rules>
        <rules>
            <rule
                rule_name="com.intel.mtwilson.core.verifier.policy.rule.PcrMatchesConstant">
                <markers>
                    <markers>BIOS</markers>
                </markers>
                <expectedPcr
                    digest_type="com.intel.mtwilson.core.common.model.PcrSha256">
                    <index>pcr_0</index>
                    <value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</value>
                </expectedPcr>
            </rule>
        </rules>
    </rules>
</BIOS>
```
5.4.9 Returning the hosts to a Trusted state

Now that you demonstrated an Untrusted state, you need to recover the hosts back to a Trusted state.

The easiest way to do this is to simply import new Flavors for each host:

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
   "connection_string": "https://trustagent.server.com:1443",
   "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
   "flavorgroup_name": "",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

Repeat this call for each registered host to import new Flavors. This will not overwrite the existing Flavors, but will add new Flavors imported directly from each host.

When new Flavors are imported, the Verification Service will automatically try to see which hosts (if any) should be matched to those Flavors. Because the new Flavors will actually match the PCR values seen on the host TPM, the Verification Service will match the new Flavors instead of the Flavors that were modified with bad values.

After new Flavors have been imported from each host, retrieve the Reports for each host again to verify that each host is Trusted:

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json
The “Overall” value should be “true” for each host.
Use Case 1b: Platform Attestation (Windows*)

The Platform Attestation use case will enable external platform integrity verification using TCG-standard practices. This feature is the foundation for other Intel® SecL – DC use cases.

Platform Attestation involves taking measurements of system components during system boot, and then cryptographically verifying that the actual measurements taken match a set of expected or approved values, ensuring that the measured components were in an acceptable or "trusted" state at the time of the last system boot.

Intel® SecL – DC leverages the Trusted Compute Group specification for a trusted boot process, extending measurements of platform components to registers in a TPM, and securely generating quotes of those measurements from the TPM for remote comparison to expected values (attestation).

Deployment will include the Verification Service and one or more Trust Agents. The Verification Service is typically installed on a Virtual Machine, while the Trust Agent must be installed on each physical server.

6.1 Scope

This use case will include the installation of the Verification Service and one or more Trust Agent compute nodes. After installation, the Platform Attestation use case will be demonstrated by registering the Trust Agent hosts; importing Flavors from the registered hosts; retrieving Attestation Reports for the registered hosts; and performing a negative test to demonstrate an "Untrusted" result.

6.2 Requirements

6.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration
  - The system must either have Intel® TXT supported and enabled in the system BIOS, OR must have Intel® Boot Guard supported in a "measurement" profile
  - If Intel® Boot Guard is used but Intel® TXT is disabled, UEFI Secure Boot must be enabled
- Each physical server must have a physical or firmware Trusted Platform Module installed and active in the system BIOS. Both TPM 2.0 is supported for Windows. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS).
6.2.2 Software
- (Optional, required for database modification for optional Untrusted state
demonstration) A GUI-based remote database client that supports the
PostgreSQL JDBC driver (for example, SQL Workbench) installed on a
laptop or other system outside of the POC environment, but with network
connectivity.
- cURL or a REST API utility like Postman to execute API requests

6.2.3 Operating System
- The Intel® SecL – DC Verification Service supports RHEL 7.6.
- The Intel® SecL – DC Trust Agent for Windows supports Microsoft Windows
Server* 2016 and above.

6.2.4 Networking
- The Intel® SecL – DC Verification Service must be able to reach each
physical server over the network.
- The Intel® SecL – DC installers will require access to package repositories;
either internet access or access to suitable repository mirrors must be
provided (as well as any applicable subscriptions to access the
repositories).
- Hostname resolution is not necessary but is very helpful. IP addresses may
be used in place of hostnames. Ensure that IP addresses and hostnames
are resolvable from all Trust Agent hosts to the Verification Service and
vice versa. Be consistent – use either all IP addresses, or all hostnames, do
not mix-and-match.

6.3 Installation
The Platform Attestation use case requires four Intel® SecL – DC services: the
Certificate Management Service, the Authentication and Authorization Service,
the Verification Service and the Trust Agent.

6.3.1 Installing the Database Server
For the purposes of POCs, a single shared database server will be used for all
of the services that require database access. Each service will use its own
database schema.

Add the Postgresql 11 repository:
Install a sample Postgresql 11 database using the install_pgdb.sh script. This script will automatically install the Postgresql database and client packages required.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Create the iseclpgdb.env answer file:
ISECL_PGDB_IP_INTERFACES=localhost
ISECL_PGDB_PORT=5432
ISECL_PGDB_SAVE_DB_INSTALL_LOG=true

Execute the installation script:

./install_pgdb.sh

Note: the database installation only needs to be performed once if the same database server will be used for all services that require a database. Only the “create_db” step need to be repeated if the database server will be shared.

After installation, the database schemas must be created initialized and tables created for each service.

./create_db.sh aas_db aas_db_user password
./create_db.sh mw_as vs_db_user password

6.3.2 Installing the Certificate Management Service

6.3.2.1 Supported Operating Systems


6.3.2.2 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

6.3.2.3 Installation

To install the Intel® SecL-DC Certificate Management Service:

2. Create the cms.env installation answer file for an unattended installation:

   ```
   AAS_TLS_SAN=<comma-separated list of IPs and hostnames for the AAS>
   SAN_LIST=<comma-separated list of IPs and hostnames for the CMS>,127.0.0.1,localhost
   AAS_API_URL=https://<Authentication and Authorization Service IP or Hostname>:8444/aas
   ```

   The SAN list will be used to authenticate the Certificate Signing Request from the AAS to the CMS. Only a CSR originating from a host matching the SAN list will be honored. Later, in the AAS authservice.env installation answer file, this same SAN list will be provided for the AAS installation. These lists must match, and must be valid for IPs and/or hostnames used by the AAS system. If both the AAS and CMS will be installed on the same system, “127.0.0.1,localhost” may be used. The SAN list variables also accept the wildcards “?” (for single-character wildcards) and “*” (for multiple-character wildcards) to allow address ranges or multiple FQDNs.

   The AAS_API_URL represents the URL for the AAS that will exist after the AAS is installed.

   For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Certificate Management Service.

3. Execute the installer binary.

   ```
   ./certificate-management-service-1.0.bin
   ```

   When the installation completes, the Certificate Management Service is available. The services can be verified by running `cms status` from the command line.

   ```
   # cms status
   ```

   After installation is complete, the CMS will output a bearer token to the console. This token will be used with the AAS during installation to authenticate certificate requests to the CMS. If this token expires or otherwise needs to be recreated, use the following command:

   ```
   cms setup cms_auth_token --force
   ```

   In addition, the SHA384 digest of the CMS TLS certificate will be needed for installation of the remaining Intel® SecL services. The digest can be obtained using the following command:

   ```
   cms tlscertsha384
   ```

### 6.3.3 Installing the Authentication and Authorization Service

#### 6.3.3.1 Required For

The AAS is REQUIRED for all use cases.

- Platform Integrity with Data Sovereignty and Signed Flavors
- Application Integrity
- Workload Confidentiality (both VMs and Docker Containers)

6.3.3.2 Prerequisites

The following must be completed before installing the Authentication and Authorization Service:

- The CMS must be installed and available

6.3.3.3 Supported Operating Systems


6.3.3.4 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

6.3.3.5 Installation

To install the AAS, a bearer token from the CMS is required. This bearer token is output at the end of the CMS installation. However, if a new token is needed, simply use the following command from the CMS command line:

```
cms setup cms_auth_token --force
```

Create the authservice.env installation answer file:

```bash
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1/
CMS_TLS_CERT_SHA384=<CMS_TLS_certificate_sha384>
AAS_DB_HOSTNAME=localhost
AAS_DB_PORT=5432
AAS_DB_NAME=aas_db
AAS_DB_USERNAME=aas_db_user
AAS_DB_PASSWORD=password
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password
SAN_LIST=<comma-separated list of IPs and hostnames for the AAS>,127.0.0.1=localhost
BEARER_TOKEN=<bearer token from CMS installation>
```

Note: the AAS_ADMIN credentials specified in this answer file will have administrator rights for the AAS and can be used to create other users, create new roles, and assign roles to users.
Execute the AAS installer:

```
./authservice-v1.6.bin
```

### 6.3.3.5.1 Creating Users

After installation is complete, a number of roles and user accounts must be generated. Most of these accounts will be service users, used by the various Intel® SeC-L services to work together. Another set of users will be used for installation permissions, and a final administrative user will be created to provide the initial authentication interface for the actual human user. The administrative user can be used to create additional users with appropriately restricted roles based on organizational needs.

Creating these required users and roles is facilitated by a script that will accept credentials and some configuration settings from an answer file, and will automate the process.

Create the populate-users.env file:

```ini
ISECL_INSTALL_COMPONENTS=TA,AN,VS,AAS,HVS
AAS_API_URL=https://127.0.0.1:8444/aas
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password

VS_CERT_SAN_LIST=*  
AH_CERT_SAN_LIST=*  
TA_CERT_SAN_LIST=*  

VS_SERVICE_USERNAME=verification_service
VS_SERVICE_PASSWORD=password

AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password

GLOBAL_ADMIN_USERNAME=admin
GLOBAL_ADMIN_PASSWORD=password
```

**Note:** The `ISECL_INSTALL_COMPONENTS` variable is a comma-separated list of the components that will be used in your environment. Not all services are required for every use case. If a given service will not be used in your deployment, simply delete the unnecessary service abbreviation from the `ISECL_INSTALL_COMPONENTS` list, and leave the SAN and credential variables for that service blank.

**NOTE:** The SAN list variables each support wildcards ("*" and "?"). In particular, without wildcards the Trust Agent SAN list would need to explicitly list each hostname or IP address for all Trust Agents that will be installed, which is not generally feasible. Using wildcards, domain names
and entire IP ranges can be included in the SAN list, which will allow any host matching those ranges to install the relevant service.

The GLOBAL_ADMIN credentials will be used to create an administrative user with all permissions.

Execute the populate-users script:

```
./populate-users
```

**Note:** The script can be executed with the `–output_json` argument to create the populate-user.json. This JSON output file will contain all of the users created by the script, along with usernames, passwords, and role assignments. This file can be used both as a record of the service and administrator accounts, and can be used as alternative inputs to recreate the same users with the same credentials in the future if needed. Be sure to protect this file if the `–output_json` argument is used.

The script will automatically generate the following users:

- Verification Service User
- Attestation Hub Service User
- Global Admin User

These user accounts will be used during installation of several of the Intel® SecL-DC applications. In general, whenever credentials are required by an installation answer file, the variable name should match the name of the corresponding variable used in the populate-user.env file.

In addition, the populate-users script will also output a “Bearer Token.” This token will be used to authenticate during installation of other services.

### 6.3.3.6 Creating an Authentication Token

Intel® SecL-DC uses a Bearer Token authentication schema. Tokens are issued by the AAS and can be used as authentication for other services. By default, tokens are valid for 2 hours, after which a new token will be needed.

To issue a new token, use the following API call:

```json
POST https://<AAS IP or hostname>:8444/aas/token
{
   "username" : "admin",
   "password" : "password"
}
```

For the purposes of this document, it is easiest to simply use the default Administrator password for all API requests. In a production environment it would be strongly recommended to create users with more restrictive roles based on the access needed.

Use this token as authentication for all API requests in this document. For example, use a header that looks like the following:

```
6.3.4 Installing the Verification Service

6.3.4.1 Package Dependencies

The Intel® Security Libraries Verification Service requires the following packages and their dependencies:

- Monit
- Logback (optional)
- Java® 8 JDK
- OpenSSL
- Postgres* client and server 11 (server component optional if an external Postgres database is used)
- Unzip

If they are not already installed, the Verification Service installer attempts to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

6.3.4.2 Supported Operating Systems


6.3.4.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers
- (Optional) One network interface for Asset Tag provisioning (only required for “pull” tag provisioning; required to provision Asset Tags to VMware ESXi servers).

6.3.4.4 Installation

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file for an unattended installation.
Execute the installer binary.

```
./host-verification-service-linux-4.6.bin
```

When the installation completes, the Verification Service is available. The services can be verified by running `mtwilson status` from the Verification Service command line.

```
# mtwilson status
```

### 6.3.5 Installing the Trust Agent for Windows

The Intel® SecL – DC Trust Agent must be installed on each physical TPM/Intel® TXT-enabled server that will be registered and attested.

#### 6.3.5.1 Supported Operating Systems

The Intel® SecL – DC Trust Agent for Windows supports Microsoft Windows Server 2016 and above.

#### 6.3.5.2 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with Intel® TXT activated in the system BIOS.
- TPM (version 1.2 or 2.0) installed and activated in the system BIOS, with cleared ownership status.
- Intel® SecL – DC Verification Service server installed and active.

#### 6.3.5.3 Installation

To install the Trust Agent for Windows:

1. Create the `trustagent.ini` answer file in the `C:\Temp` directory.

MTWILSON_SERVER=<IP address or hostname of the Verification Service>
MTWILSON_API_URL="https://<Verification Service IP or hostname>:8443/mtwilson/v2"
DATABASE_HOSTNAME=localhost
DATABASE_USERNAME=vs_db_user
DATABASE_PORTNUM="5432"
DATABASE_PASSWORD=password
DATABASE_SCHEMA="mw_as"
VS_SERVICE_USERNAME=verification_service
VS_SERVICE_PASSWORD=password
CMS_TLS_CERT_SHA384=<Certificate Management Service TLS digest>
BEARER_TOKEN=<Installation user token>
AAS_API_URL=https://127.0.0.1:8444/aas
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1
VS_TLS_CERT_DNS=<comma separated list of hostnames for the Verification Service>,localhost
VS_TLS_CERT_IP=<comma separated list of IP addresses for the Verification Service>,127.0.0.1
[TRUST_AGENT]
MTWILSON_API_URL=https://<Verification Service IP or Hostname>:8443/mtwilson/v2
REGISTER_TPM_PASSWORD=y
TRUSTAGENT_LOGIN_REGISTER=true
PROVISION_ATTESTATION=y
CURRENT_IP=<Trust Agent IP address>
CMS_TLS_CERT_SHA384=<Sha384 digest of CMS TLS certificate>
BEARER_TOKEN=<Installation token from populate-users script>
CMS_BASE_URL=https://<IP address or hostname of CMS>:8445/cms/v1/
AAS_API_URL=https://<IP address or hostname of AAS>:8444/aas

The MTWILSON_TLS_CERT_SHA384 value can be retrieved from the Verification Service using the following command from the Verification Service command line:

cat /opt/mtwilson/configuration/https.properties

2. Copy the Trust Agent installer executable to the C:\Temp directory.
3. Execute the Trust Agent installer, and wait for the installation to complete.

6.4 Demonstrating the Platform Attestation Use Case

6.4.1 Verify Services

The Version API can be used to confirm that all services are up and running, before proceeding to the actual use case demonstration.

GET https://verification.server.com:8443/mtwilson/v2/version
GET https://trustagent.server.com:1443/v2/version

6.4.2 Register Hosts

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

Each Trust Agent host will need to be registered with a separate call.

POST https://verification.service.com:8443/mtwilson/v2/hosts

```json
{
   "host_name": "<hostname of host to be registered>",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE",
   "connection_string": "https://trustagent.server.com:1443",
   "flavorgroup_name": "",
   "description": "<description>"
}
```
6.4.3 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.


6.4.4 Retrieve Reports (Untrusted – No Flavors)

Now it is possible to retrieve the latest Attestation Report for each host to demonstrate the “faults” shown when match Flavors do not exist in the Verification Service database. Currently no Flavors should be present, so all of these Reports will show an Untrusted status, with Faults showing that matching Flavors were required but not found.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

6.4.5 Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (BIOS, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host; the BIOS and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors

```json
{
  "connection_string": "https://trustagent.server.com:1443",
  "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
  "flavorgroup_name": "",
  "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

6.4.6 Import the default SOFTWARE Flavor

The SOFTWARE Flavor is required for Linux and Windows servers. By default, a manifest for the Trust Agent components is pushed to the Trust Agent during installation; after rebooting to the TCB boot mode, this Flavor can now be imported. Importantly, this Flavor part should only be imported once – the default SOFTWARE Flavor has a unique label for each version, and requests to import the SOFTWARE Flavor for a version of the Trust Agent that already exists will throw an error. For example, if you have 3 Trust Agent servers, you may import the PLATFORM, OS, and HOST_UNIQUE Flavors from each of them, but the SOFTWARE Flavor should only be imported once.

POST https://verification.server.com:8443/mtwilson/v2/flavors
6.4.7 Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

This is the same request used previously, but this time all of the Reports should show that the “Overall” trust is “True”, and you should no longer see any Faults.

GET
https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

6.4.8 Demonstrate Untrusted State

Finally, you need to demonstrate what an Untrusted attestation looks like and what can cause it. There are a number of ways to force a host to attest as Untrusted.

In production, a host can become Untrusted if the BIOS or OS kernel are upgraded/downgraded to use a version where no Flavor exists, or if a malicious attacker has inserted malicious code into a measured component, and so forth.

For testing purposes, the easiest ways to force a host to become Untrusted are to either boot the host to a non-tboot boot option (simply reboot, and at the boot menu option when RHEL boots, select the option without tboot).

Attestation Reports are valid for 24 hours by default (this is configurable); you need to force a new Report to be generated now, after the change.

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
input: {"host_name":"<Trust Agent host>"}

The new Report will show that the BIOS Flavor part is still Trusted, but that the HOST_UNIQUE and OS Flavor Parts have mismatched measurements. This is because the platform performs the BIOS-related measurements before the OS boots, but tboot performs the other measurements.
Because the system booted without tboot, the TPM memory registers that store the tboot measurements will all be 0’s or F’s, and will mismatch the expected measurements in the Flavors.

6.4.8.1 (Optional) Database Modification

While slightly more complicated, it is also possible to edit one or more Flavors in the database and change the expected measurement results. Because of the size of the Flavor objects in the database, this is best performed with a remote GUI database client like SQL Workbench, as opposed to using the command line. This step is optional, and will show a different Fault more consistent with a platform that has been tampered with by a malicious attack.

First you need to enable remote access to the Verification Service database, which is disabled by default.

In the /usr/local/pgsql/data/postgresql.conf file, add the following line:
listen_addresses='***'

Next, edit /usr/local/pgsql/data/pg_hba.conf and change the last line to the following:
host all all 127.0.0.1/0 password

Finally, restart the Postgresql database:
systemctl restart postgresql-11

Now you should be able to access the database using a remote database client, like SQL Workbench. By default, the database name is mw_as. The username and password were configured in the mtwilson.env answer file during the installation; the example used "root" and "password".

After connecting to the mw_as database, look at the mw_flavor table. You should see a number of Flavors, including BIOS, OS, and HOST_UNIQUE Flavors from when they were imported in an earlier step.

In the "content" column is the actual Flavor object. Find a "BIOS" Flavor:

Find the "pcr_0" measurements (depending on the TPM version and your platform OEM, you may see only SHA1, only SHA256, or both measurements).

{"SHA1":{"pcr_0":{"value":"d2ed125942726641a7260c392beb67d531a0def"}}
"SHA256":{"pcr_0":{"value":"db83f0e8a1773c21164c17986037cdf8afclbbdclb815772c6dalbebf1a7f8a3"}

Change a digit in each of these entries. Do not change the length, simply change the last digit of each "pcr_0" hash to a different hexadecimal digit.

Save the changes to the database.

Next, we need to change the Verification Service settings to skip Flavor signature validation. By default, the HVS will verify the signature on each Flavor used in an attestation; since we are directly modifying a Flavor, that signature validation will fail, and create a different fault from the one we’re trying to demonstrate (a Flavor signature validation fault would say: “Signature is not trusted for flavor with id <Flavor ID>”).

mtwilson config skip.flavor.signature.verification true
mtwilson restart

Now force the creation of a new Report.
POST https://verification.server.com:8443/mtwilson/v2/reports
{
   "host_name":"

Do this for each host in the POC environment. When you get the Report for the host that was using the modified BIOS flavor, the OS and HOST_UNIQUE attestations will remain Trusted.

The BIOS Flavor part, however, will show that it is Untrusted, with a “PcrValueMismatch” fault indicating that the actual measurement for PCR 0 does not match the expected value.

Because PCR 0 includes the hash of the core system BIOS, this replicates an event where the BIOS has been maliciously modified.

<BIOS>
   <trust>false</trust>
   <rules>
      <rules>
         <rule
rule_name="com.intel.mtwilson.core.verifier.policy.rule.PcrMatchesConstant">
            <markers>
               <markers>BIOS</markers>
            </markers>
            <expectedPcr
digest_type="com.intel.mtwilson.core.common.model.PcrSha256">
               <index>pcr_0</index>

               <value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</value>

               <pcrBank>SHA256</pcrBank>
            </expectedPcr>
         </rule>
      </rules>
   </rules>
</BIOS>
6.4.9 Returning the hosts to a Trusted state

Now that you demonstrated an Untrusted state, you need to recover the hosts back to a Trusted state.

The easiest way to do this is to simply import new Flavors for each host:

```
POST https://verification.server.com:8443/mtwilson/v2/flavors
{
  "connection_string": "https://trustagent.server.com:1443;u=tagentadmin;p=password",
  "partial_flavor_types": ["BIOS", "OS", "HOST_UNIQUE"],
  "flavorgroup_name": "mtwilson_automatic",
  "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

Repeat this call for each registered host to import new Flavors. This will not overwrite the existing Flavors, but will add new Flavors imported directly from each host.

When new Flavors are imported, the Verification Service will automatically try to see which hosts (if any) should be matched to those Flavors. Because the new Flavors will actually match the PCR values seen on the host TPM, the Verification Service will match the new Flavors instead of the Flavors that were modified with bad values.

After new Flavors have been imported from each host, retrieve the Reports for each host again to verify that each host is Trusted:

```
GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
```

Headers:
Accept: application/json

The "Overall" value should be "true" for each host.
7 Use Case 1c: Platform Attestation (VMware*)

The Platform Attestation use case will enable external platform integrity verification using TCG-standard practices. This feature is the foundation for other Intel® SecL – DC use cases.

Platform Attestation involves taking measurements of system components during system boot, and then cryptographically verifying that the actual measurements taken match a set of expected or approved values, ensuring that the measured components were in an acceptable or “trusted” state at the time of the last system boot.

Intel® SecL – DC leverages the Trusted Compute Group specification for a trusted boot process, extending measurements of platform components to registers in a TPM, and securely generating quotes of those measurements from the TPM for remote comparison to expected values (attestation).

Deployment will include only the Verification Service, which is typically installed on a Virtual Machine.

Platform Attestation for VMware ESXi* servers does not require a Trust Agent. Instead, it requires communication with vCenter Server*; vCenter* and ESXi will handle the TPM-specific functions, and the Intel® SecL – DC Verification Service will retrieve the TPM quote for a given host from a vCenter API.

7.1 Scope

This use case will include the installation of the Verification Service. After installation, the Platform Attestation use case will be demonstrated by registering the VMware ESXi hosts, importing Flavors from the registered hosts, retrieving Attestation Reports for the registered hosts and performing a negative test to demonstrate an “Untrusted” result.

7.2 Requirements

7.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration
  - The system must either have Intel® TXT supported and enabled in the system BIOS, OR must have Intel® Boot Guard supported in a "measurement" profile
  - If Intel® Boot Guard is used but Intel® TXT is disabled, UEFI Secure Boot must be enabled
- Each physical server must have a physical or firmware TPM installed and active in the system BIOS. Both TPM versions 1.2 and 2.0 are supported
for VMware. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS) and must be “Enabled” in the system BIOS (TPM 1.2 only; TPM 2.0 is always “Enabled”)

- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require a TPM or Intel® TXT, but does require network access to the other POC servers

7.2.2 Software
- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity
- cURL or a REST API utility like Postman to execute API requests

7.2.3 Operating System
- The Intel® SecL – DC Verification Service supports RHEL 7.6
- The ESXi version required depends on the TPM version used. For TPM 1.2, ESXi 6.5 Update 2 or later is required; for TPM 2.0, vSphere 6.7 Update 1 or later is required. However, Asset Tags (described in the Asset Tags for VMware section) are not currently supported for VMware TPM 2.0 servers.
- An instance of VMware vCenter Server* is required. The specific version of vCenter required depends on the ESXi version that will be used; see VMware documentation for vCenter and ESXi interoperability requirements. Both the vCenter Virtual Appliance* and the Windows application are supported, but for ease of deployment the Virtual Appliance is recommended. Each ESXi host must be managed by vCenter and be added to a vCenter Cluster* object.

7.3 Installation
This section details how to install the Intel® SecL – DC services.

7.3.1 Installing the Database Server
For the purposes of POCs, a single shared database server will be used for all of the services that require database access. Each service will use its own database schema.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Install a sample Postgresql 11 database using the install_pgdb.sh script. This script will automatically install the Postgresql database and client packages required.
Add the Postgresql 11 repository:
https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Create the iseclpgdb.env answer file:
ISECL_PGDB_IP_INTERFACES=localhost
ISECL_PGDB_PORT=5432
ISECL_PGDB_SAVE_DB_INSTALL_LOG=true

Execute the installation script:
./install_pgdb.sh

Note: the database installation only needs to be performed once if the same database server will be used for all services that require a database. Only the “create_db” step need to be repeated if the database server will be shared.

After installation, the database schemas must be created initialized and tables created for each service.

./create_db.sh aas_db aas_db_user password
./create_db.sh mw_as vs_db_user password

7.3.2 Installing the Certificate Management Service

7.3.2.1 Supported Operating Systems

7.3.2.2 Recommended Hardware
- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

7.3.2.3 Installation
To install the Intel® SecL-DC Certificate Management Service:
2. Create the cms.env installation answer file for an unattended installation:

The SAN list will be used to authenticate the Certificate Signing Request from the AAS to the CMS. Only a CSR originating from a host matching the SAN list will be honored. Later, in the AAS authservice.env installation answer file, this same SAN list will be provided for the AAS installation. These lists must match, and must be valid for IPs and/or hostnames used by the AAS system. If both the AAS and CMS will be installed on the same system, "127.0.0.1,localhost" may be used. The SAN list variables also accept the wildcards "?" (for single-character wildcards) and "*" (for multiple-character wildcards) to allow address ranges or multiple FQDNs.

The AAS_API_URL represents the URL for the AAS that will exist after the AAS is installed.

For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Certificate Management Service.

3. Execute the installer binary.

`.certificate-management-service-1.0.bin`

When the installation completes, the Certificate Management Service is available. The services can be verified by running `cms status` from the command line.

```
# cms status
```

After installation is complete, the CMS will output a bearer token to the console. This token will be used with the AAS during installation to authenticate certificate requests to the CMS. If this token expires or otherwise needs to be recreated, use the following command:

```
cms setup cms_auth_token --force
```

In addition, the SHA384 digest of the CMS TLS certificate will be needed for installation of the remaining Intel® SecL services. The digest can be obtained using the following command:

```
cms tlscertsha384
```

## 7.3.3 Installing the Authentication and Authorization Service

### 7.3.3.1 Required For

The AAS is REQUIRED for all use cases.

- Platform Integrity with Data Sovereignty and Signed Flavors
- Application Integrity
- Workload Confidentiality (both VMs and Docker Containers)
7.3.3.2 Prerequisites

The following must be completed before installing the Authentication and Authorization Service:

- The CMS must be installed and available

7.3.3.3 Supported Operating Systems


7.3.3.4 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

7.3.3.5 Installation

To install the AAS, a bearer token from the CMS is required. This bearer token is output at the end of the CMS installation. However, if a new token is needed, simply use the following command from the CMS command line:

cms setup cms_auth_token --force

Create the authservice.env installation answer file:

```
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1/
CMS_TLS_CERT_SHA384=<CMS_TLS_certificate_sha384>
AAS_DB_HOSTNAME=localhost
AAS_DB_PORT=5432
AAS_DB_NAME=aas_db
AAS_DB_USERNAME=aas_db_user
AAS_DB_PASSWORD=password
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password
SAN_LIST=<comma-separated list of IPs and hostnames for the AAS>,127.0.0.1,localhost
BEARER_TOKEN=<bearer_token_from_CMS_installation>
```

**Note:** the AAS_ADMIN credentials specified in this answer file will have administrator rights for the AAS and can be used to create other users, create new roles, and assign roles to users.

Execute the AAS installer:

```
./authservice-v1.6.bin
```
7.3.3.5.1 Creating Users

After installation is complete, a number of roles and user accounts must be generated. Most of these accounts will be service users, used by the various Intel® SecL services to work together. Another set of users will be used for installation permissions, and a final administrative user will be created to provide the initial authentication interface for the actual human user. The administrative user can be used to create additional users with appropriately restricted roles based on organizational needs.

Creating these required users and roles is facilitated by a script that will accept credentials and some configuration settings from an answer file, and will automate the process.

Create the populate-users.env file:

```
ISECL_INSTALL_COMPONENTS=VS,AAS
AAS_API_URL=https://<AAS IP address or hostname>:8444/aas
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password

VS_CERT_SAN_LIST=*  
AH_CERT_SAN_LIST=* 

VS_SERVICE_USERNAME=verification_service
VS_SERVICE_PASSWORD=password
GLOBAL_ADMIN_USERNAME=admin
GLOBAL_ADMIN_PASSWORD=password
```

**Note:** The `ISECL_INSTALL_COMPONENTS` variable is a comma-separated list of the components that will be used in your environment. Not all services are required for every use case. If a given service will not be used in your deployment, simply delete the unnecessary service abbreviation from the `ISECL_INSTALL_COMPONENTS` list, and leave the SAN and credential variables for that service blank.

**NOTE:** The SAN list variables each support wildcards( "*" and "?"). In particular, without wildcards the Trust Agent SAN list would need to explicitly list each hostname or IP address for all Trust Agents that will be installed, which is not generally feasible. Using wildcards, domain names and entire IP ranges can be included in the SAN list, which will allow any host matching those ranges to install the relevant service.

The `GLOBAL_ADMIN` credentials will be used to create an administrative user with all permissions.

Execute the populate-users script:

```
./populate-users
```
Note: The script can be executed with the –output_json argument to create the populate-user.json This json output file will contain all of the users created by the script, along with usernames, passwords, and role assignments. This file can be used both as a record of the service and administrator accounts, and can be used as alternative inputs to recreate the same users with the same credentials in the future if needed. Be sure to protect this file if the –output_json argument is used.

The script will automatically generate the following users:

Verification Service User
Attestation Hub Service User
Global Admin User

These user accounts will be used during installation of several of the Intel® SecL-DC applications. In general, whenever credentials are required by an installation answer file, the variable name should match the name of the corresponding variable used in the populate-user.env file.

In addition, the populate-users script will also output a “Bearer Token.” This token will be used to authenticate during installation of other services.

7.3.3.6 Creating an Authentication Token

Intel® SecL-DC uses a Bearer Token authentication schema. Tokens are issued by the AAS and can be used as authentication for other services. By default, tokens are valid for 2 hours, after which a new token will be needed.

To issue a new token, use the following API call:

```
POST https://<AAS IP or hostname>:8444/aas/token
{
    "username" : "admin",
    "password" : "password"
}
```

For the purposes of this document, it is easiest to simply use the default Administrator password for all API requests. In a production environment it would be strongly recommended to create users with more restrictive roles based on the access needed.

Use this token as authentication for all API requests in this document. For example, use a header that looks like the following:

```
“Authorization: Bearer <token content>”
```
7.3.4 Installing the Verification Service

7.3.4.1 Package Dependencies

The Intel® Security Libraries Verification Service requires the following packages and their dependencies:

- Monit
- Logback (optional)
- Java* 8 JDK
- OpenSSL
- Postgres* client and server 11 (server component optional if an external Postgres database is used)
- Unzip

If they are not already installed, the Verification Service installer attempts to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

7.3.4.2 Supported Operating Systems


7.3.4.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers
- (Optional) One network interface for Asset Tag provisioning (only required for “pull” tag provisioning; required to provision Asset Tags to VMware ESXi servers).

7.3.4.4 Installation

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file for an unattended installation.
 Execute the installer binary.

 ./host-verification-service-linux-4.6.bin

When the installation completes, the Verification Service is available. The services can be verified by running mtwilson status from the Verification Service command line.

```
# mtwilson status
```

### 7.4 Demonstrating the Platform Attestation Use Case

#### 7.4.1 Verify Services

The Version API can be used to confirm that all services are up and running, before proceeding to the actual use case demonstration.


#### 7.4.2 Register Hosts

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

Each ESXi host will need to be registered with a separate call. Note that the Connection String requires access to vCenter. Provide credentials for a vCenter user with at minimum the "Validate Session" permission (using an Administrator account is acceptable)

POST [https://verification.service.com:8443/mtwilson/v2.getHosts](https://verification.service.com:8443/mtwilson/v2/host)

```json
{
    "host_name": "<hostname of ESXi host to be registered, as it appears in vCenter>",
    "tls_policy_id" : "TRUST_FIRST_CERTIFICATE",
}
7.4.3 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.


7.4.4 Retrieve Reports (Untrusted – No Flavors)

You can now retrieve the latest Attestation Report for each host to demonstrate the “faults” shown when match Flavors do not exist in the Verification Service database.

Currently no Flavors should be present, so all of these Reports will show an Untrusted status, with Faults showing that matching Flavors were required but not found.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

7.4.5 Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (BIOS, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host, the BIOS and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors

{  
  "connection_string": "vmware:https://vCenter.server.com/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>;u=username@vsphere.local;p=password",
  "flavorgroup_name": "mtwilson_automatic",
  "description": "<description>"
}
7.4.6  Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, you can retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

This is the same request used previously, but this time all of the Reports should show that the “Overall” trust is “True,” and we should no longer see any Faults.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json

7.4.7  Demonstrate Untrusted State

Finally, it is possible to demonstrate what an Untrusted attestation looks like and what can cause it. There are a number of ways to force a host to attest as Untrusted. In production, a host can become Untrusted if the BIOS or OS kernel are upgraded/downgraded to use a version where no Flavor exists, or if a malicious attacker has inserted malicious code into a measured component, and so forth.

For testing purposes, the easiest ways to force a host to become Untrusted are to either boot the host to a non-tboot boot option (simply reboot, and at the boot menu option when RHEL boots, select the option without tboot).

Attestation Reports are valid for 24 hours by default (this is configurable); you need to force a new Report to be generated now, after our change.

POST https://verification.server.com:8443/mtwilson/v2/reports
Headers:
Accept: application/json
input: {"host_name":"<ESXi host name>"}

The new Report will show that the BIOS Flavor part is still Trusted, but that the HOST_UNIQUE and OS Flavor Parts have mismatched measurements. This is because the platform performs the BIOS-related measurements before the OS boots, but tboot performs the other measurements.

Because the system booted without tboot, the TPM memory registers that store the tboot measurements will all be 0’s or F’s, and will mismatch the expected measurements in the Flavors.

7.4.7.1  (Optional) Database Modification

While slightly more complicated, it is also possible to edit one or more Flavors in the database and change the expected measurement results. Because of the size of the Flavor objects in the database, this is best performed with a remote GUI database client like SQL Workbench, as opposed to using the command
line. This step is optional, and will show a different Fault more consistent with a platform that has been tampered with by a malicious attack.

First you need to enable remote access to the Verification Service database, which is disabled by default.

In the `/usr/local/pgsql/data/postgresql.conf` file, add the following line:

```
listen_addresses='*' 
```

Next, edit `/usr/local/pgsql/data/pg_hba.conf` and change the last line to the following:

```
host all all 127.0.0.1/0 password
```

Finally, restart the Postgresql database:

```
systemctl restart postgresql-11
```

Now you should be able to access the database using a remote database client, like SQL Workbench. By default, the database name is `mw_as`. The username and password were configured in the `mtwilson.env` answer file during the installation; the example used “root” and “password”.

After connecting to the `mw_as` database, look at the `mw_flavor` table. You should see a number of Flavors, including BIOS, OS, and HOST_UNIQUE Flavors from when they were imported in an earlier step.

In the “content” column is the actual Flavor object. Find a “BIOS” Flavor:

```
{"content": }
```

Find the "pcr_0" measurements (depending on the TPM version and your platform OEM, you may see only SHA1, only SHA256, or both measurements).

```
"SHA1":{"pcr_0":{"value":"d2ed125942726641a7260c4f92be67d531a0def"}

"SHA256":{"pcr_0":{"value":"db83f0e0a17b3c21164c17986037cdf8af1bbd1b815772c6dalbemba7f8a3"}
```

Change a digit in each of these entries. Do not change the length, simply change the last digit of each "pcr_0" hash to a different hexadecimal digit.

Save the changes to the database.

Next, we need to change the Verification Service settings to skip Flavor signature validation. By default, the HVS will verify the signature on each Flavor used in an attestation; since we are directly modifying a Flavor, that signature validation will fail, and create a different fault from the one we're
trying to demonstrate (a Flavor signature validation fault would say: “Signature is not trusted for flavor with id <Flavor ID>”).

mtwilson config skip.flavor.signature.verification true
mtwilson restart

Now force the creation of a new Report.
POST https://verification.server.com:8443/mtwilson/v2/reports
{
   "host_name": "<Trust Agent hostname>
}

Do this for each host in the POC environment. When you get the Report for the host that was using the modified BIOS flavor, the OS and HOST_UNIQUE attestations will remain Trusted.

The BIOS Flavor part, however, will show that it is Untrusted, with a “PcrValueMismatch” fault indicating that the actual measurement for PCR 0 does not match the expected value.

Because PCR 0 includes the hash of the core system BIOS, this replicates an event where the BIOS has been maliciously modified.

```xml
<BIOS>
  <trust>false</trust>
  <rules>
    <rule
      rule_name="com.intel.mtwilson.core.verifier.policy.rule.PcrMatchesConstant">
      <markers>
        <markers>BIOS</markers>
      </markers>
      <expectedPcr
        digest_type="com.intel.mtwilson.core.common.model.PcrSha256">
        <index>pcr_0</index>
        <value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</value>
        <pcrBank>SHA256</pcrBank>
      </expectedPcr>
    <faults>
      <fault_name="com.intel.mtwilson.core.verifier.policy.fault.PcrValueMismatchSha256">
        <description>Host PCR 0 with value 987c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9 does not match expected value 187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</description>
        <pcrIndex>pcr_0</pcrIndex>
      </faults>
```
7.4.8 Returning the hosts to a Trusted state

Now that you demonstrated an Untrusted state, you need to recover the hosts back to a Trusted state.

The easiest way to do this is to simply import new Flavors for each host:

```
POST https://verification.server.com:8443/mtwilson/v2/flavors
{
    "connection_string": "vmware:https://vCenter.server.com/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>;u=<username@vSphere.local>;p=<password>",
    "partial_flavor_types": ["BIOS", "OS", "HOST_UNIQUE"],
    "flavorgroup_name": "mtwilson_automatic",
    "tls_policy_id":"TRUST_FIRST_CERTIFICATE"
}
```

Repeat this call for each registered host to import new Flavors. This will not overwrite the existing Flavors, but will add new Flavors imported directly from each host.

When new Flavors are imported, the Verification Service will automatically try to see which hosts (if any) should be matched to those Flavors. Because the new Flavors will actually match the PCR values seen on the host TPM, the Verification Service will match the new Flavors instead of the Flavors that were modified with bad values.

After new Flavors have been imported from each host, retrieve the Reports for each host again to verify that each host is Trusted:

```
GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
```

Headers:
Accept: application/json

The “Overall” value should be “true” for each host.
8 Use Case 2a: Asset Tags (Linux)

8.1 Scope

This use case will expand on the Platform Attestation use cases to add attestation of hardware-based Asset Tags. This use case will not include any new software installations, and will instead use the existing software with new API requests to provision Asset Tags to hosts and then generate new Attestation Reports to see the results.

8.2 Requirements

Use Case 1a, Platform Attestation (Linux) must have been completed. The same hardware and software resources will be used to complete this use case.

No additional hardware or software is required beyond what was used for the previous use case.

8.3 Demonstrating the Asset Tag Use Case for Linux

8.3.1 Verify that hosts are registered and Trusted

To begin, you need to verify that all prerequisite steps were completed.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers: Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows "OVERALL=true".

8.3.2 Create Asset Tag Certificates for each host

Asset Tags are a combination of an Asset Tag Certificate, and an Asset Tag Flavor based on that certificate that is associated with a host.

The Asset Tag Certificate is unique for each host, even if all of the key/value pairs are identical. The Subject of the certificate is the Hardware UUID of the server; this is used as part of the later Asset Tag Flavor matching, and to ensure that each Tag must be applied to a specific server.
The Asset Tag Certificate creation call consists of the Hardware UUID of the host, and then a list of any number of key/value pairs. The key/value pairs are the actual Tags that will be reflected in the Attestation Reports for the host, and can represent locations or any other information. For example, a server could be tagged with “Country=USA; Department=Finance; Compliance=PCI”.

Repeat this call for each host.

POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{"hardware_uuid": "<hardware UUID of host to be tagged>",
 "selection_content": [
   {
     "name": "Country",
     "value": "USA"
   },
   {
     "name": "Department",
     "value": "Finance"
   },
   {
     "name": "Compliance",
     "value": "PCI"
   }
 ]
}

8.3.3 Deploy Asset Tags

After the Asset Tag Certificates are all created, they need to actually be deployed to the physical servers. On the backend, this process involves generating a cryptographic hash of the Certificate and writing that hash to a specific NVRAM index on the TPM of the server to be tagged. Making the Tag deployment request will also automatically generate the Asset Tag Flavor.

The deployment request is very simple, and only requires the ID of the Asset Tag Certificate to be deployed and the hostname or IP address (however the host is registered in the “host_name” field when the registration call is made).

The Verification Service will search for the host using the hostname or IP address and use the connection string provided at the time of registration to make a request to the Trust Agent running on that host. The Trust Agent will then perform the actual TPM commands to write the hash to the TPM.

POST https://verification.server.com:8443/mtwilson/v2/rpc/deploy-tag-certificate
{
   "certificate_id": "<certificate ID>",
   "host": "<Hostname of host to be tagged>"
}

This request will also cause a new Asset Tag Flavor to be created for the actual attestation process. Repeat this step for each host to be tagged.
8.3.4 **Create new Attestation Reports for each host**

Deploying the Asset Tags does not actually create a new Attestation Report, so you will need to create a new Report for each tagged host.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host. POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
{"host_name":"<hostname or IP address>"}

Repeat this step for each registered host.

Note that each new Report will now contain an additional Flavor section for the Asset Tag; this will show whether the Tag is Trusted (meaning the most recent currently valid Asset Tag certificate for that host is actually reflected in the host TPM), and will expose all of the key/value pairs used in the Asset Tag Certificate that was deployed.
9  **Use Case 2b: Asset Tags (Windows)**

9.1  **Scope**

This use case will expand on the Platform Attestation use cases to add attestation of hardware-based Asset Tags. This use case will not include any new software installations, and will instead use the existing software with new API requests to provision Asset Tags to hosts and then generate new Attestation Reports to see the results.

9.2  **Requirements**

Use Case 1b, Platform Attestation (Windows) must have been completed. The same hardware and software resources will be used to complete this use case. No additional hardware or software is required beyond what was used for the previous use case.

9.3  **Demonstrating the Asset Tag Use Case for Windows**

9.3.1  **Verify that hosts are registered and Trusted**

To begin, you need to verify that all prerequisite steps were completed.

GET  
https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:  
Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows "OVERALL=true".

9.3.2  **Create Asset Tag Certificates for each host**

Asset Tags are a combination of an Asset Tag Certificate, and an Asset Tag Flavor based on that certificate that is associated with a host.

The Asset Tag Certificate is unique for each host, even if all of the key/value pairs are identical. The Subject of the certificate is the Hardware UUID of the server; this is used as part of the later Asset Tag Flavor matching, and to ensure that each Tag must be applied to a specific server.
The Asset Tag Certificate creation call consists of the Hardware UUID of the host, and then a list of any number of key/value pairs. The key/value pairs are the actual Tags that will be reflected in the Attestation Reports for the host, and can represent locations or any other information. For example, a server could be tagged with “Country=USA; Department=Finance; Compliance=PCI”.

Repeat this call for each host.

```json
POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{
  "hardware_uuid": "<hardware UUID of host to be tagged>",
  "selection_content": [
    {
      "name": "Country",
      "value": "USA"
    },
    {
      "name": "Department",
      "value": "Finance"
    },
    {
      "name": "Compliance",
      "value": "PCI"
    }
  ]
}
```

### 9.3.3 Deploy Asset Tags

After the Asset Tag Certificates are all created, they need to actually be deployed to the physical servers. On the backend, this process involves generating a cryptographic hash of the Certificate and writing that hash to a specific NVRAM index on the TPM of the server to be tagged. Making the Tag deployment request will also automatically generate the Asset Tag Flavor.

The deployment request is very simple, and only requires the ID of the Asset Tag Certificate to be deployed, and the hostname or IP address (however the host is registered in the "host_name” field when the registration call is made).

The Verification Service will search for the host using the hostname or IP address and use the connection string provided at the time of registration to make a request to the Trust Agent running on that host. The Trust Agent will then perform the actual TPM commands to write the hash to the TPM.

```json
POST https://verification.server.com:8443/mtwilson/v2/rpc/deploy-tag-certificate
{
  "certificate_id": "<certificate ID>",
  "host": "<Hostname of host to be tagged>"
}
```

This request will also cause a new Asset Tag Flavor to be created for the actual attestation process. Repeat this step for each host to be tagged.
9.3.4 Create new Attestation Reports for each host

Deploying the Asset Tags does not actually create a new Attestation Report, so you will need to create a new Report for each tagged host.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.

```plaintext
POST https://verification.server.com:8443/mtwilson/v2/reports
Headers:
Accept: application/json
{"host_name":"<hostname or IP address>"}
```

Repeat this step for each registered host.

Note that each new Report will now contain an additional Flavor section for the Asset Tag. This will show whether the Tag is Trusted (meaning the most recent currently valid Asset Tag certificate for that host is actually reflected in the host TPM), and will expose all of the key/value pairs used in the Asset Tag Certificate that was deployed.
10 Use Case 2c: Asset Tags (VMware)

10.1 Scope

This use case will expand on the Platform Attestation use cases to add attestation of hardware-based Asset Tags. This use case will not include any new software installations, and will instead use the existing software with new API requests to provision Asset Tags to hosts and then generate new Attestation Reports to see the results.

10.2 Requirements

Use Case 1c, Platform Attestation (VMware) must have been completed. The same hardware and software resources will be used to complete this use case. No additional hardware or software is required beyond what was used for the previous use case. However, SSH must be enabled for the ESXi hosts to be tagged. See VMware documentation for instruction on enabling the SSH service on an ESXi host from vCenter. SSH can be safely disabled after the tags are written.

10.3 Demonstrating the Asset Tag Use Case for VMware

10.3.1 Verify that hosts are registered and Trusted

To begin, you need to verify that all prerequisite steps were completed.

```
GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
```

Headers:
Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows “OVERALL=true”.

10.3.2 Create Asset Tag Certificates for each host

Asset Tags are a combination of an Asset Tag Certificate, and an Asset Tag Flavor based on that certificate that is associated with a host.

The Asset Tag Certificate is unique for each host, even if all of the key/value pairs are identical. The Subject of the certificate is the Hardware UUID of the
server; this is used as part of the later Asset Tag Flavor matching, and to ensure that each Tag must be applied to a specific server.

The Asset Tag Certificate creation call consists of the Hardware UUID of the host and then a list of any number of key/value pairs. The key/value pairs are the actual Tags that will be reflected in the Attestation Reports for the host, and can represent locations or any other information. For example, a server could be tagged with “Country=USA; Department=Finance; Compliance=PCI”.

Repeat this call for each host.

POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{"hardware_uuid": "<hardware UUID of host to be tagged>",
"selection_content": [
{
    "name": "Country",
    "value": "USA"
},
{
    "name": "Department",
    "value": "Finance"
},
{
    "name": "Compliance",
    "value": "PCI"
}
]}

10.3.3 Deploy Asset Tags

10.3.3.1 Calculate the Certificate Hash Value

Only the hash value of the Asset Tag Certificate can be provisioned to the TPM, due to the low size of the NVRAM.

1. Retrieve the Asset Tag Certificate. The Asset Tag Certificate can be retrieved either from the response when the Asset Tag certificate is created or by using a GET API request to retrieve the certificate:

GET https://verification.server.com:8443/mtwilson/v2/tag-certificates?subjectEqualTo=<HardwareUUID>

2. Copy only the “certificate” value (this will be the certificate in encoded format) and write the data to a file on a Linux system. Remove any line breaks and save the file. Assuming the filename used is “tag-cert”, use the following string to generate the correct hash:

```
cat tag-cert | base64 --decode | openssl dgst -sha1 | awk -F " " '{print $2)'
```

This hash value will be what is actually written to the TPM NVRAM.
10.3.3.2 **Provision the Certificate Hash to the Host TPM**

Starting in ESXi 6.5u2, you can now use SSH to write Asset Tags directly with no need for TPM clears, reboots, PXE or BIOS access. SSH to the ESXi host using root credentials, then use the next command:

```
esxcli hardware tpm tag set -d <hash>
```

You can use the following command to verify that the tag was written:

```
esxcli hardware tpm tag get
```

Reboot the host. After rebooting, the TPM PCR 22 will have the measured value of the hash.

---

10.3.4 **Create Asset Tag Flavor**

For VMware ESXi hosts, the Asset Tag Flavor must be created by importing it from the host after the Tag has been provisioned.

**POST** https://verification.server.com:8443/mtwilson/v2/flavors

```
{ "connection_string": "vmware:https://<vCenter.server.com>/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>;u=<username@vSphere.local>;p=<password>",
 "tls_policy_id":"TRUST_FIRST_CERTIFICATE",
 "partial_flavor_types": ["ASSET_TAG"]
}
```

Once the Asset Tag Flavor is imported, the host can be attested including Asset Tags as normal.

---

10.3.5 **Create new Attestation Reports for each host**

Deploying the Asset Tags does not actually create a new Attestation Report, so you will need to create a new Report for each tagged host.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.

**POST** https://verification.server.com:8443/mtwilson/v2/reports

Headers:

```
Accept: application/json
```

```
{"host_name":"<hostname or IP address>"}
```

Repeat this step for each registered host.

Note that each new Report will now contain an additional Flavor section for the Asset Tag; this will show whether the Tag is Trusted (meaning the most recent currently valid Asset Tag certificate for that host is actually reflected in the host TPM), and will expose all of the key/value pairs used in the Asset Tag Certificate that was deployed.
11 Use Case 3: Application Integrity

11.1 Scope

This use case will expand on the Platform Attestation use cases to add attestation of user-defined applications. This use case will not include any new software installations, and will instead use the existing software with new API requests to specify new files and folders to be measured and attested.

11.2 Requirements

Use Case 1a, Platform Attestation (Linux) and Use Case 2a, Asset tags (Linux) must have been completed. The same hardware and software resources will be used to complete this use case.

No additional hardware or Intel® SecL software is required beyond what was used for the previous use cases.

11.3 Demonstrating the Application integrity Use Case for Linux

11.3.1 Verify that hosts are registered and Trusted

To begin verify that all prerequisite steps were completed.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows “OVERALL=true”.

11.3.2 Create a New SOFTWARE Flavor

Creating a new SOFTWARE Flavor requires specifying a sample host where the application, files or folders that will be measured are currently present. The measurements specified in the manifest will be captures when this call is executed, and the Verification Service will communicate with the Trust Agent and create a SOFTWARE Flavor based on the file measurements.
The software manifest below includes files related to the Trust Agent, but these can be defined for any application on a bare-metal server by specifying the static files and folders that comprise that application.

POST https://server.com:8443/mtwilson/v2/flavor-from-app-manifest

Input:
<ManifestRequest xmlns="lib:wml:manifests-req:1.0">
<connectionString>intel:https://trustagent.server.com:1443;u=trustagentUsername;p=trustagentPassword</connectionString>
<Manifest xmlns="lib:wml:manifests:1.0" DigestAlg="SHA384" Label="Samplev1" Uuid="">

  <Dir Type="dir" Include=".*" Exclude="" Path="/opt/trustagent/hypertext WEB-INF" />

  <Symlink Path="/opt/trustagent/bin/tpm_nvinfo" />

  <File Path="/opt/trustagent/bin/module_analysis_da.sh" />
</Manifest>
</ManifestRequest>

11.3.3 Deploy the Application Manifest

Once the new SOFTWARE Flavor is created, its manifest can be deployed to any number of Trust Agent servers. Deployment can be performed with an API request to the Verification Service, and requires the ID of the host, and the ID of the SOFTWARE Flavor.

POST https://server.com:8443/mtwilson/v2/rpc/deploy-software-manifest

Input:
{
  "flavor_id":"a2345ff4-6dc7-4c74-82be-578592e7e3ba",
  "host_id":"45874ff4-6d37-5875-82be-12392e7e123"
}

After the manifest has been deployed, the Trust Agent server must be rebooted. This will allow tbootXM to measure the files and folders specified in the manifest, and extend them to the TPM for attestation.

11.3.4 Create new Attestation Reports for each host

After rebooting each server after deploying the SOFTWARE Flavor manifest, a new Attestation Report must be generated to reflect the new Flavor.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
{
  "host_name":"<hostname or IP address>"
}

Repeat this step for each registered host.
Note that each new Report will now contain an additional SOFTWARE Flavor, in addition to the default Trust Agent SOFTWARE Flavor. Because the default Flavor Match Policy for SOFTWARE Flavors is “all_of”, both of the SOFTWARE Flavors must match the host attestation for the host to be Trusted.
12 **Use Case 4: OpenStack Orchestration**

OpenStack Orchestration* with Intel® SecL – DC involves using security attributes (“Trust” and Asset Tag key/value pairs) to define image launch requirements and populate the compute nodes with traits reflecting their status in the Intel® SecL – DC Verification Service.

In this way, an image owner can define requirements for the image so that instances of that image are always launched on Trusted compute nodes with specific Asset Tag key/value pairs.

12.1 **Scope**

This use case will build on Use Cases 1a and 2a and add OpenStack integration to demonstrate using Intel® SecL – DC to control where Virtual Machines (VMs) are allowed to launch.

The installation of OpenStack will not be described here. For OpenStack installation instructions, see OpenStack documentation. Also see Section 3.2 for the list of OpenStack services required for this use case.

In addition to the Intel® SecL – DC components previously installed in Use Cases 1a and 2a, Integration Hub* will also be installed. The Integration Hub is the Intel® SecL – DC component that assigns OpenStack environments to specific tenants and handles pushing the required attributes to OpenStack.

After installation, the Use Case will be demonstrated by setting Image Traits to represent the required attributes for hosts to launch instances of protected images. A successful launch on a Trusted host whose Traits match the requirements in the Image Traits will be demonstrated, followed by a failed launch where the Asset Tags or Trust Status are not met by any available compute node.

12.2 **Requirements**

Use Cases 1a and 2a must have been completed prior to starting the OpenStack Orchestration use case. The same software and hardware from those previous use cases will be used and expanded upon (no new hardware is required, but additional software will need to be installed).

12.2.1 **Hardware**

- One or more server platforms with Intel® TXT supported and enabled in the system BIOS.
- Each physical server must have a physical or firmware TPM installed and active in the system BIOS. Both TPM versions 1.2 and 2.0 are supported.
for Linux. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS) and must be “Enabled” in the system BIOS (TPM 1.2 only; TPM 2.0 is always “Enabled”).

- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require a TPM or Intel® TXT, but does require networking access to the other POC servers.
- One server (which may be physical or virtual) for the Intel® SecL – DC Integration Hub. This server does not require a TPM or Intel® TXT, but does require networking access to the other POC servers. This server can be the same server used for the Verification Service.

12.2.2 Software

- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL or a REST API utility like Postman to execute API requests.

12.2.3 Operating System

- The Intel® SecL – DC Verification Service supports RHEL 7.6.
- The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.6.
- The Intel® SecL – DC Integration Hub supports RHEL 7.6.

12.2.4 Networking

- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC Integration Hub must be able to reach the Intel® SecL – DC Verification Service and OpenStack services over the network.
- The Intel® SecL – DC installers will require access to package repositories. Either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful, particularly with OpenStack. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

12.2.5 OpenStack

- OpenStack Rocky*: Nova*, Glance*, Horizon*, Neutron* and Keystone* services must be installed and running. Note that specifically OpenStack Rocky or later is required.
A minimum of one OpenStack compute node with the Intel® SecL – DC Trust Agent deployed as per Use Case 1a. Generally this means installing OpenStack Nova Compute* on the same server(s) used as Trust Agent servers in the previous use cases.

12.2.6 Installation

12.2.6.1 Installing/Configuring the Database

The Integration Hub requires its own database schema. On the same database server configured for Use Case 1, use the create_db script to create the Hub database schema:

```
./create_db.sh attestation_hub_pu hub_db_user password
```

12.2.6.2 Installing the Integration Hub

To install the Integration Hub, follow these steps:

1. Copy the Integration Hub installation binary to the `/root/` directory.
2. Create the `attestation-hub.env` installation answer file. See the sample file below.

```
ATTESTATION_HUB_PORT_HTTP=19082
ATTESTATION_HUB_PORT_HTTPS=19445

AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password
MTWILSON_SERVER=<IP address or hostname of the Verification Service>
MTWILSON_API_URL=https://<Verification Service IP or hostname>:8443/mtwilson/v2

CMS_TLS_CERT_SHA384=<CMS TLS digest>
BEARER_TOKEN=<Installation token from populate-users script>
AAS_API_URL=https://<AAS IP or Hostname>:8444/aas
CMS_BASE_URL=https://<CMS IP or Hostname>:8445/cms/v1

ATTESTATION_HUB_DB_NAME="attestation_hub_pu"
ATTESTATION_HUB_DB_HOSTNAME="localhost"
ATTESTATION_HUB_DB_PORTNUM="5432"
ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
ATTESTATION_HUB_DB_USERNAME=hub_db_user
ATTESTATION_HUB_DB_PASSWORD=password
```

3. Execute the installer binary.
12.3 Demonstrating the OpenStack Orchestration* Use Case

12.3.1 Configuring a Tenant in the Integration Hub

12.3.1.1 Create the Tenant

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. A Tenant defines the connection and authentication details to reach the OpenStack services.

POST https://hub.server.com:19445/v1/tenants

{
    "name": "DemoTenant",
    "plugins": [
    {
        "name": "nova",
        "properties": [
        {
            "key": "api.endpoint",
            "value": "http://<Nova API endpoint>/compute/v2.1"
        },
        {"key": "auth.endpoint",
        "value": "http://<Keystone API endpoint>:5000/identity"
        },
        {"key": "auth.version",
        "value": "v3"
        },
        {"key": "user.name",
        "value": "<Username for Nova API>"
        },
        {"key": "user.password",
        "value": "<Password for Nova API>"
        },
        {"key": "tenant.name",
        "value": "<Name of tenant in OpenStack>"
        },
        {"key": "domain.name",
        "value": "<Name of Domain in OpenStack>"
        }
    ]
    }
}
12.3.1.2 Assign Hosts to the Tenant

Hosts must be assigned to a tenant before Intel® SecL – DC security attributes will be pushed to the OpenStack Traits. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of hardware_uuids.

Hosts are assigned using the Tenant ID (returned in the Create Tenant step) and the Hardware UUID of one or more hosts. List each OpenStack Compute Node's Hardware UUID in the array.

POST https://hub.server.com:19445/v1/host-assignments

```
{
    "tenant_id": "<Tenant ID>",
    "hardware_uuids": [
        "<Host 1 Hardware UUID>",
        "<Host 2 Hardware UUID>"
    ]
}
```

12.3.1.3 Verify that the Hub is Retrieving Reports

Next we want to list the hosts as seen in the Hub to ensure the Hub is communicating with the Verification Service and retrieving Reports.

GET https://hub.server.com:19445/v1/hosts

This will return a list of all of the hosts seen by the Hub with their most recent Report status. By default, the Hub will poll the Verification Service for new Reports every 2 minutes, refresh this list and then send updates to all Tenant endpoints according to which hosts were assigned to which Tenant.

12.3.2 Setting Launch Requirements in Image Traits

Image Traits define the policy for which Traits are required for that Image to be launched on a Nova Compute node. By setting these Traits to “required,” the OpenStack scheduler will require these same Traits to be present on a Nova Compute node in order to launch instances of the image.

To require a “Trusted” Attestation Report:

CUSTOM_ISECL_TRUSTED=required

The naming convention for Asset Tags is more flexible, and any number of these Traits can be used simultaneously.
**Note:** All of the Traits must be present on the Compute Node for the scheduler to allow instances to land, so be sure not to set mutually exclusive Asset Tag values.

CUSTOM_ISECL_AT_TAG_<key>_<value>=required

In Use Case 2a, we applied Asset Tags to the hosts using the following key/value pairs:

Country=USA; Department=Finance; Compliance=PCI

We can then make an Image Trait that will require hosts to be tagged with “Country=USA” to meet the placement requirements.

CUSTOM_ISECL_AT_TAG_COUNTRY_USA=required

These Traits can be set using CLI commands for OpenStack Glance*, using the name of your VM image:

openstack image set --property trait:CUSTOM_ISECL_AT_TAG_COUNTRY_USA=required <image_name>

openstack image set --property trait:CUSTOM_ISECL_TRUSTED=required <image_name>

This image will now require hosts that are currently Trusted, and that have the “Country=USA” tag. Other tags will be disregarded. Additional Traits can be set to require more than one Asset Tag key/value pair, in which case all of the required Traits must be present – partial matches will fail.

### 12.3.3 Launching an Instance on a Compliant Node

Simply launch an instance of the image using the OpenStack Placement service. This can be done using the OpenStack Horizon UI*, the commandline, or using a REST API.

Next is an example of the REST API for launching a VM. See OpenStack documentation for other examples:

```python
POST http://openstack.compute.com:port/v2.1/servers
{
    "server" : {
        "accessIPv4": "1.2.3.4",
        "name" : "LaunchOnTrustedWithTags",
        "imageRef" : "<Image ID or path to image>",
        "flavorRef" : <flavor>",
        "metadata" : {
            "My Server Name" : "LaunchOnTrustedWithTags"
        }
    }
}
```

Because the Trust Status of the compute node(s) is currently Trusted, and because the tag “Country=USA” is actually present on the host(s), the new VM instance should launch successfully.
12.3.4 Launching an Instance where No Nodes are Compliant

The OpenStack Placement Service filters the compute nodes and will skip those that do not meet the requirements of the VM (like available CPU, memory, or custom attributes like Intel® SecL – DC Trust and Asset tags).

This means that if the CUSTOM_CIT_TRUSTED=required Trait is present in the Image Properties, any “Untrusted” host (where “Overall=false” in the Trust Report, or if no Report is available, or if the Report is expired), will be left out of the schedule.

This is easily demonstrated by applying an Asset Tag requirement for a key/value pair that does not exist on any host.

To do this, apply the Trait CUSTOM_CIT_AT_TAG_EXIST_FALSE=required to the Image Properties:

```
openstack image set --property
trait:CUSTOM_CIT_AT_TAG_EXIST_FALSE=required <image_name>
```

This can be applied in addition to the Trait requirements that already exist. In this way we’ll have a set of requirements, some of which can be met by our compute node(s) and one of which cannot.

Once the Trait is set on the Image Properties, launch a new instance of the Image.

The launch will fail, with an error saying that no suitable host was found.

In the OpenStack Nova API log, an error will show that the Image Traits requirements were not met on any available host.
13 Use Case 5: Kubernetes Orchestration

Through the use of Custom Resource Definitions for the Kubernetes Master, Intel® Security Libraries can make Kubernetes aware of Intel® SecL security attributes and make them available for pod orchestration. In this way, a security-sensitive pod can be launched only on “Trusted” physical worker nodes, or on physical worker nodes that match specified Asset Tag values.

**NOTE**: This control only applies to pods launched using the Kubernetes scheduler, and these scheduling controls will not affect manually-launched instances where a specific worker node is defined (since this does not use the scheduler at all). Intel SecL-DC uses existing Kubernetes interfaces and does not modify Kubernetes code, using only the standard Custom Resource Definition mechanism to add this functionality to the Kubernetes Master. The datacenter owner or Kubernetes administrator is responsible for the security of the Kubernetes workload scheduling process in general, and Intel recommends following published Kubernetes security best practices.

13.1 Scope

This use case will build on Use Cases 1a and 2a and add Kubernetes integration to demonstrate using Intel® SecL – DC to control where Kubernetes pods are allowed to launch.

The installation of Kubernetes will not be described here. For Kubernetes installation instructions, see Kubernetes documentation.

This document does not specify a particular pod or specific container images to use.

In addition to the Intel® SecL – DC components previously installed in Use Cases 1a and 2a, Integration Hub* will also be installed. The Integration Hub is the Intel® SecL – DC component that assigns Kubernetes environments to specific tenants and handles pushing the required attributes to Kubernetes.

After installation, the Use Case will be demonstrated by configuring pods with matchExpressions to represent the required attributes for hosts to launch instances of protected pods. A successful launch on a Trusted host whose security attributes match the requirements will be demonstrated, followed by a failed launch where the Asset Tags or Trust Status are not met by any available compute node.
13.2 Requirements

13.2.1 Hardware

- One or more server platforms with in a supported Root of Trust configuration.
- Each physical server must have a physical or firmware Trusted Platform Module version 2.0 installed and active in the system BIOS. The TPM must have "cleared" ownership (the TPM ownership can be cleared in the system BIOS).
- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require any Root of Trust configuration, but does require network access to the other POC servers.
- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL or a REST API utility like Postman to execute API requests.

13.2.2 Operating System

- The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.
- The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.4 and above.
- The Intel® SecL – DC Integration Hub supports RHEL 7.4 and above.

13.2.3 Networking

- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC Integration Hub must be able to reach the Intel® SecL – DC Verification Service and OpenStack services over the network.
- The Intel® SecL – DC installers will require access to package repositories. Either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful, particularly with Kubernetes. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

13.2.4 Kubernetes

- Kubernetes Master Node must be installed and running
A minimum of one Kubernetes Worker Node with the Intel® SecL – DC Trust Agent deployed as per Use Case 1a. Generally this means installing the Kubernetes Worker Node on the same server(s) used as Trust Agent servers in the previous use cases.

13.2.5 Installation

13.2.5.1 Installing/Configuring the Database

The Integration Hub requires its own database schema. On the same database server configured for Use Case 1, use the create_db script to create the Hub database schema:

```
./create_db.sh attestation_hub_pu hub_db_user password
```

13.2.5.2 Installing the Integration Hub

To install the Integration Hub, follow these steps:

1. Copy the Integration Hub installation binary to the `/root/` directory.
2. Create the `attestation-hub.env` installation answer file. See the sample file below.

```
ATTESTATION_HUB_PORT_HTTP=19082
ATTESTATION_HUB_PORT_HTTPS=19445

AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password
MTWILSON_SERVER=<IP address or hostname of the Verification Service>
MTWILSON_API_URL=https://<Verification Service IP or hostname>:8443/mtwilson/v2

CMS_TLS_CERT_SHA384=<CMS TLS digest>
BEARER_TOKEN=<Installation token from populate-users script>
AAS_API_URL=https://<AAS IP or Hostname>:8444/aas
CMS_BASE_URL=https://<CMS IP or Hostname>:8445/cms/v1

ATTESTATION_HUB_DB_NAME="attestation_hub_pu"
ATTESTATION_HUB_DB_HOSTNAME="localhost"
ATTESTATION_HUB_DB_PORTNUM="5432"
ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
ATTESTATION_HUB_DB_USERNAME=hub_db_user
ATTESTATION_HUB_DB_PASSWORD=password
```

3. Execute the installer binary.

13.2.5.3 Installing the Intel® SecL-DC Custom Resource Definitions for Kubernetes

Intel® SecL uses Custom Resource Definitions to add the ability to base orchestration decisions on Intel® SecL security attributes to Kubernetes.
These CRDs allow Kubernetes administrators to configure pods to require specific security attributes so that the Kubernetes Master Node will schedule those pods only on Worker Nodes that match the specified attributes.

Perform the following steps on the Kubernetes Master Node:

1) Add a mount path to the `/etc/Kubernetes/manifests/kube-scheduler.yaml` file for the Intel SecL scheduler extension:

   - mountPath: /opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/
     name: extendedsched
     readOnly: true

2) Add a volume path to the `/etc/Kubernetes/manifests/kube-scheduler.yaml` file for the Intel SecL scheduler extension:

   - hostPath:
     path: /opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/
     type: ""
     name: extendedsched

3) Add “policy-config-file” path in the “/etc/Kubernetes/manifests/kube-scheduler.yaml” file under ‘command’ section:

   - command:
     - kube-scheduler
     - --policy-config-file=/opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/scheduler-policy.json
     - --bind-address=127.0.0.1
     - --kubeconfig=/etc/kubernetes/scheduler.conf
     - --leader-elect=true

4) Copy the isecl-k8s-extensions.bin installer to the Kubernetes Master and execute the installer

   ./isecl-k8s-extensions.bin

5) The installer will output a set of keystores upon completion into `/opt/isecl-k8s-extensions/attestation-hub-keystores/`. These contain keys that will be used by the Integration Hub to communicate with this Kubernetes Master. Copy the contents of this directory to the Integration Hub:

   scp -r /opt/isecl-k8s-extensions/attestation-hub-keystores/* root@integration-hub.server.com:/opt/attestation-hub/configuration/

   The following keystores will be copied:
   - root_k8s_client.p12
   - root_k8s_trust.p12
   - root_keystore.properties

   Note that the Integration Hub can manage multiple Kubernetes Master environments at the same time, but the keystores must be kept separate. To do this, create subfolders in the Hub configuration directory for each separate Kubernetes environment, and copy the appropriate keystores to the matching subfolder.
6) On the Integration Hub system, set the permissions on the copied files:

```
chown attestation-hub:attestation-hub *p12
```

7) Copy the Integration Hub public key to the Kubernetes Master:

```
scp attestation-hub.server.com:/opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/hub_public_key.pem /etc/kubernetes/pki/
```

8) Run the command `systemctl restart kubelet` to restart all the control plane container services, including the base scheduler.

9) (Optional) Verify that the Intel® SecL Custom Resource Definitions have been started:

```
kubectl get crds
kubectl get -o json hostattributes.isecl.intel.com
```

---

13.3 Demonstrating the Kubernetes Scheduling Use Case

13.3.1 Configuring a Tenant in the Integration Hub

13.3.1.1 Create the Tenant

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. A Tenant defines the connection and authentication details to reach the OpenStack services.

```
POST https://hub.server.com:19445/v1/tenants
```

```json
{
   "name": "DemoTenant",
   "plugins": [
   {
      "name": "kubernetes",
      "properties": [
      {
         "key": "api.endpoint",
         "value": "https://kubernetes-master.server.com:6443"
      },
      {
         "key": "tenant.name",
         "value": "DemoTenant"
      },
      {
         "key": "plugin.provider",
         "value": "com.intel.attestationhub.plugin.kubernetes.KubernetesPluginImpl"
      }
   ]
   }
}
```
"key": "kubernetes.client.keystore",
"value": "/opt/attestation-
hub/configuration/root_k8s_client.jks"
},
{
"key": "kubernetes.server.keystore",
"value": "/opt/attestation-
hub/configuration/root_k8s_trust.jks"
},
{
"key": "kubernetes.server.keystore.password",
"value": "<Keystore password>"
},
{
"key": "kubernetes.client.keystore.password",
"value": "<Keystore Password>"
}

NOTE: the value of kubernetes.client.keystore and kubernetes.server.keystore must be the filesystem path on the Integration Hub that contains the Kubernetes Master keystores output from the scheduler extensions for this tenant. The value of kubernetes.server.keystore.password and kubernetes.server.keystore.password must be the keystore passwords output by the scheduler extensions installer.

13.3.1.2 List Hosts

The Integration Hub periodically queries the Verification Service for the list of all new Reports; only Reports generated after the timestamp of the most recent query are returned. Because host registration will trigger the generation of a new Report, any new hosts added to the Verification Service will be seen in the Hub on the next refresh (determined by the value of the POLL_INTERVAL variable during install).

The list of hosts known to the Integration Hub can be retrieved using the below API sample.

GET https://server.com:19445/v1/hosts

13.3.1.3 Assign Hosts to Tenants

Hosts must be assigned to a tenant before Intel SecL-DC security attributes will be pushed to Kubernetes. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of hardware_uuids.

POST https://server.com:19445/v1/host-assignments
{
"tenant_id": "DC02284A-F525-4094-BA01-E317FE28E15F",

The Hub will “push” Intel SecL-DC attributes as OpenStack Traits to the
tenant’s configured endpoints (in this case, Nova) every time it looks for new
attestations.

**13.3.1.4 Assign Hosts to the Tenant**

Hosts must be assigned to a tenant before Intel® SecL – DC security attributes
will be pushed to the OpenStack Traits. Any number of hosts may be assigned
to one tenant. Multiple hosts can be assigned to a tenant in a single request by
using a comma-separated list of hardware_uuids.

Hosts are assigned using the Tenant ID (returned in the Create Tenant step)
and the Hardware UUID of one or more hosts. List each OpenStack Compute
Node’s Hardware UUID in the array.

```
POST https://hub.server.com:19445/v1/host-assignments
{
  "tenant_id": "<Tenant ID>",
  "hardware_uuids": [
    "<Host 1 Hardware UUID>", "<Host 2 Hardware UUID>"
  ]
}
```

**13.3.1.5 Verify that the Hub is Retrieving Reports**

Next we want to list the hosts as seen in the Hub to ensure the Hub is
communicating with the Verification Service and retrieving Reports.

```
GET https://hub.server.com:19445/v1/hosts
```

This will return a list of all of the hosts seen by the Hub with their most recent
Report status. By default, the Hub will poll the Verification Service for new
Reports every 2 minutes, refresh this list and then send updates to all Tenant
endpoints according to which hosts were assigned to which Tenant.

**13.3.2 Configuring Pods to Require Intel® SecL Attributes**

1) (Optional) Verify that the worker nodes have had their Intel® SecL security
attributes populated:

```
kubectl get nodes --show-labels
```

The output should show the Trust status and any Asset Tags applied to all
of the registered Worker Nodes.
2) In Use Case 2a, we applied Asset Tags to the hosts using the following key/value pairs:

    Country=USA; Department=Finance; Compliance=PCI

We can now configure a Pod to require a Trusted status, and the Asset Tags previously set. Add the following to the Pod creation files:

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: isecl.trusted
            operator: In
            values:
            - "true"
          - key: isecl.TAG_Country
            operator: In
            values:
            - USA
          - key: isecl.TAG_Department
            operator: In
            values:
            - Finance
          - key: isecl.TAG_Compliance
            operator: In
            values:
            - PCI
```
Below is a full sample Pod config file:

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: isecl.trusted
            operator: In
            values:
              - "true"
          - key: isecl.TAG_Country
            operator: In
            values:
              - USA
          - key: isecl.TAG_Department
            operator: In
            values:
              - Finance
          - key: isecl.TAG_Compliance
            operator: In
            values:
              - PCI
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: nginx
```

The “isecl.trusted” key defines the requirement for a Trusted host. Only one of these keys should be used. The “TAG_” keys indicate Asset Tags; if the workload should only launch on hosts with the “COUNTRY=USA” Asset Tag, the
pod should be launched with the matchExpression key "TAG_COUNTRY" with the value "USA".

**Note:** All of the matchExpression definitions must be true for a given worker node to launch the pod – in the example above, the host must be attested as Trusted with Asset Tags “Country=US,” “Customer=Customer1,” and “State=CA”. If the worker node has additional Asset Tags beyond the ones required, the pod will still be able to be launched on that node. However, if one of the specified Tags is missing or has a different value, that worker node will not be used for that pod.

### 13.3.3 Launching a Pod on a Compliant Worker Node

Simply launch any Pod using a config file containing the matchExpression policy requirements for Trust and Asset Tags. The Intel® SecL CRDs on the Kubernetes Master will use the Worker Node labels propagated from the Intel® SecL Integration Hub to ensure the Pod is launched according to the policy requirements.

Below is a sample command for the Kubernetes Master to launch a Pod:

```
kubectl create -f pod_config.yaml
```

We can verify that the Master Node CRDs contain attributes successfully pushed from the Integration Hub:

```
kubectl get crds
```

```
kubectl get -o json hostattributes.isecl.intel.com
```

We can list the Worker Nodes and their labels:

```
kubectl get nodes --show-labels
```

Finally, we can list the running Pods to confirm that the Pod launched and is running on a compliant Worker Node:

```
kubectl get pods
```

### 13.3.4 Launching a Pod where No Worker Nodes are Compliant

The Kubernetes Master with the Intel® SecL CRDs installed will require the matchExpressions defined in the Pod config file to be true on a given Worker Node to launch the Pod. All of the conditions must be true for a Worker Node to be selected.

To demonstrate a failure to launch when no compliant Worker Nodes are found, we can simply modify the Pod config file to require a set of Asset Tags that do not exist on any Worker Node:
We can now try to launch this Pod. Since no Worker Nodes have the Asset Tag “NoHostHasThisTag=True”, the Master Node will not be able to find a compliant Worker Node, and the Pod launch will fail.

kubectl create -f pod_config.yaml
14 Use Case 6a: Workload Confidentiality with VMs

14.1 Scope

This use case will build on Use Case 4 to demonstrate using Intel® SecL – DC to control where virtual machine workloads are allowed to be accessed using encryption. The installation of OpenStack will not be described here. For OpenStack installation instructions, see OpenStack documentation. Also see Section 12.2.5 for the list of OpenStack services required for this use case.

A sample qcow2 VM image is required. For the sake of this POC, we generally recommend a small test image like the CirrOS minimal Linux distribution:

https://docs.openstack.org/image-guide/obtain-images.html

14.2 Requirements

14.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration
  - The system must either have Intel® TXT supported and enabled in the system BIOS, OR must have Intel® Boot Guard supported in a "measurement" profile
  - If Intel® Boot Guard is used but Intel® TXT is disabled, UEFI Secure Boot must be enabled.
- Each physical server must have a physical or firmware Trusted Platform Module installed and active in the system BIOS. TPM 2.0 is supported for Linux*. The TPM must have "cleared" ownership (the TPM ownership can be cleared in the system BIOS).
- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service, Workload Service, Integration Hub, Authentication and Authorization Service, Certificate Management Service, Key Broker Service, and Workload Policy Manager (all of these services may reside on a single server). This server does not require a TPM or Intel® TXT, but does require network access to the other POC servers.

14.2.2 Software

- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC* driver (for example, SQL Workbench*) installed on a laptop or other system outside of the POC environment, but with network connectivity.
• cURL* or a REST* API utility like Postman* to execute API requests.

14.2.3 Operating System

• The Intel® SecL – DC services support Red Hat Enterprise* Linux (RHEL*) 7.6.

14.2.4 Networking

• The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
• The Intel® SecL – DC installers will require access to package repositories; either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
• Hostname resolution is not necessary but is very helpful. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

14.2.5 OpenStack

• OpenStack Rocky*: Nova*, Glance*, Horizon*, Neutron* and Keystone* services must be installed and running. Note that specifically OpenStack Rocky or later is required.
• Each physical server must be deployed as an OpenStack compute node. Generally this means installing OpenStack Nova Compute* on the physical TPM-enabled servers.

14.3 Installation

The Workload Integrity use case requires eight Intel® SecL – DC services: the Certificate Management Service, the Authentication and Authorization Service, the Verification Service, the Workload Servicer, the Key Broker Service, the Workload Policy Manager, the Workload Agent, and the Trust Agent.

14.3.1 Installing the Database Server

For the purposes of POCs, a single shared database server will be used for all of the services that require database access. Each service will use its own database schema.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
Install a sample Postgresql 11 database using the install.pgdb.sh script. This script will automatically install the Postgresql database and client packages required.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Create the iseclpgdb.env answer file:

ISECL_PGDB_IP_INTERFACES=localhost
ISECL_PGDB_PORT=5432
ISECL_PGDB_SAVE_DB_INSTALL_LOG=true

Execute the installation script:

./install.pgdb.sh

Note: the database installation only needs to be performed once if the same database server will be used for all services that require a database. Only the “create_db” step need to be repeated if the database server will be shared.

After installation, the database schemas must be created initialized and tables created for each service.

./create_db.sh aas_db aas_db_user password
./create_db.sh mw_as vs_db_user password
./create_db.sh wls_db wls_db_user password
./create_db.sh attestation_hub_pu hub_db_user password

14.3.2 Installing the Certificate Management Service

14.3.2.1 Supported Operating Systems


14.3.2.2 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

14.3.2.3 Installation

To install the Intel® SecL-DC Certificate Management Service:

2. Create the cms.env installation answer file for an unattended installation:

```
AAS_TLS_SAN=<comma-separated list of IPs and hostnames for the AAS>
SAN_LIST=<comma-separated list of IPs and hostnames for the CMS>,127.0.0.1,localhost
AAS_API_URL=https://<Authentication and Authorization Service IP or Hostname>:8444/aas
```

The SAN list will be used to authenticate the Certificate Signing Request from the AAS to the CMS. Only a CSR originating from a host matching the SAN list will be honored. Later, in the AAS authservice.env installation answer file, this same SAN list will be provided for the AAS installation. These lists must match, and must be valid for IPs and/or hostnames used by the AAS system. If both the AAS and CMS will be installed on the same system, “127.0.0.1,localhost” may be used. The SAN list variables also accept the wildcards “?” (for single-character wildcards) and “*” (for multiple-character wildcards) to allow address ranges or multiple FQDNs.

The AAS_API_URL represents the URL for the AAS that will exist after the AAS is installed.

For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Certificate Management Service.

3. Execute the installer binary.

```
./certificate-management-service-1.0.bin
```

When the installation completes, the Certificate Management Service is available. The services can be verified by running `cms status` from the command line.

```
# cms status
```

After installation is complete, the CMS will output a bearer token to the console. This token will be used with the AAS during installation to authenticate certificate requests to the CMS. If this token expires or otherwise needs to be recreated, use the following command:

```
 cms setup cms_auth_token --force
```

In addition, the SHA384 digest of the CMS TLS certificate will be needed for installation of the remaining Intel® SecL services. The digest can be obtained using the following command:

```
 cms tlscertsha384
```

---

### 14.3.3 Installing the Authentication and Authorization Service

#### 14.3.3.1 Required For

The AAS is REQUIRED for all use cases.

- Platform Integrity with Data Sovereignty and Signed Flavors
14.3.3.2 Prerequisites

The following must be completed before installing the Authentication and Authorization Service:

- The CMS must be installed and available

14.3.3.3 Supported Operating Systems


14.3.3.4 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

14.3.3.5 Installation

To install the AAS, a bearer token from the CMS is required. This bearer token is output at the end of the CMS installation. However, if a new token is needed, simply use the following command from the CMS command line:

cms setup cms_auth_token --force

Create the authservice.env installation answer file:

```
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1/
CMS_TLS_CERT_SHA384=<CMS_TLS certificate sha384>
AAS_DB_HOSTNAME=localhost
AAS_DB_PORT=5432
AAS_DB_NAME=aas_db
AAS_DB_USERNAME=aas_db_user
AAS_DB_PASSWORD=password
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password
SAN_LIST=<comma-separated list of IPs and hostnames for the AAS>,127.0.0.1,localhost
BEARER_TOKEN=<bearer token from CMS installation>
```

**Note:** the AAS_ADMIN credentials specified in this answer file will have administrator rights for the AAS and can be used to create other users, create new roles, and assign roles to users.
Execute the AAS installer:

./authservice-v1.6.bin

14.3.3.5.1 Creating Users

After installation is complete, a number of roles and user accounts must be generated. Most of these accounts will be service users, used by the various Intel® SeCL services to work together. Another set of users will be used for installation permissions, and a final administrative user will be created to provide the initial authentication interface for the actual human user. The administrative user can be used to create additional users with appropriately restricted roles based on organizational needs.

Creating these required users and roles is facilitated by a script that will accept credentials and some configuration settings from an answer file, and will automate the process.

Create the populate-users.env file:

```bash
ISECL_INSTALL_COMPONENTS=HVS,KBS,TA,HVS,WLS,WPM,AH,VS,AAS

AAS_API_URL=https://127.0.0.1:8444/aas
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password

VS_CERT_SAN_LIST=*        
AH_CERT_SAN_LIST=*        
TA_CERT_SAN_LIST=*        
WLS_CERT_SAN_LIST=*        
KBS_CERT_SAN_LIST=*        

VS_SERVICE_USERNAME=verification_service 
VS_SERVICE_PASSWORD=password

AH_SERVICE_USERNAME=hub_service 
AH_SERVICE_PASSWORD=password

WPM_SERVICE_USERNAME=wpm_service 
WPM_SERVICE_PASSWORD=password

WLS_SERVICE_USERNAME=wls_service 
WLS_SERVICE_PASSWORD=password

WLA_SERVICE_USERNAME=wla_service 
WLA_SERVICE_PASSWORD=password

GLOBAL_ADMIN_USERNAME=admin 
GLOBAL_ADMIN_PASSWORD=password
```

**Note:** The ISECL_INSTALL_COMPONENTS variable is a comma-separated list of the components that will be used in your environment. Not all services are required for every use case. If a given service will not be used in your deployment, simply delete the unnecessary service abbreviation from the ISECL_INSTALL_COMPONENTS list, and leave the SAN and credential variables for that service blank.
NOTE: The SAN list variables each support wildcards ("*" and "?"). In particular, without wildcards the Trust Agent SAN list would need to explicitly list each hostname or IP address for all Trust Agents that will be installed, which is not generally feasible. Using wildcards, domain names and entire IP ranges can be included in the SAN list, which will allow any host matching those ranges to install the relevant service.

The GLOBAL_ADMIN credentials will be used to create an administrative user with all permissions.

Execute the populate-users script:

./populate-users

Note: The script can be executed with the –output_json argument to create the populate-user.json This json output file will contain all of the users created by the script, along with usernames, passwords, and role assignments. This file can be used both as a record of the service and administrator accounts, and can be used as alternative inputs to recreate the same users with the same credentials in the future if needed. Be sure to protect this file if the –output_json argument is used.

The script will automatically generate the following users:
Verification Service User
Attestation Hub Service User
Global Admin User

These user accounts will be used during installation of several of the Intel® SecL-DC applications. In general, whenever credentials are required by an installation answer file, the variable name should match the name of the corresponding variable used in the populate-user.env file.

In addition, the populate-users script will also output a “Bearer Token.” This token will be used to authenticate during installation of other services.

14.3.3.6 Creating an Authentication Token

Intel® SecL-DC uses a Bearer Token authentication schema. Tokens are issued by the AAS and can be used as authentication for other services. By default, tokens are valid for 2 hours, after which a new token will be needed.

To issue a new token, use the following API call:

```
POST https://<AAS IP or hostname>:8444/aas/token
{
    "username" : "admin",
    "password" : "password"
}
```
For the purposes of this document, it is easiest to simply use the default Administrator password for all API requests. In a production environment it would be strongly recommended to create users with more restrictive roles based on the access needed.

Use this token as authentication for all API requests in this document. For example, use a header that looks like the following:

“Authorization: Bearer <token content>”

14.3.4 Installing the Verification Service

14.3.4.1 Package Dependencies

The Intel® Security Libraries Verification Service requires the following packages and their dependencies:

- Monit
- Logback (optional)
- Java* 8 JDK
- OpenSSL
- Unzip

If they are not already installed, the Verification Service installer attempts to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

14.3.4.2 Supported Operating Systems


14.3.4.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers
- (Optional) One network interface for Asset Tag provisioning (only required for “pull” tag provisioning; required to provision Asset Tags to VMware ESXi servers).
### 14.3.4.4 Installation

#### 14.3.4.4.1

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file for an unattended installation.

Execute the installer binary.

```bash
./host-verification-service-linux-4.6.bin
```

When the installation completes, the Verification Service is available. The services can be verified by running `mtwilson status` from the Verification Service command line.

```
# mtwilson status
```

### 14.3.5 Installing the Workload Service

#### 14.3.5.1 Package Dependencies

#### 14.3.5.2 Supported Operating Systems


#### 14.3.5.3 Recommended Hardware

#### 14.3.5.4 Installation

1. Copy the Workload Service installation binary to the /root/ directory.
2. Create the `workload-service.env` installation answer file:
Execute the WLS installer binary:
./workload-service-1.0.bin

14.3.6 Installing the Integration Hub

To install the Integration Hub, follow these steps:

1. Copy the Integration Hub installation binary to the /root/ directory.
2. Create the `attestation-hub.env` installation answer file. See the sample file below.

```
ATTESTATION_HUB_PORT_HTTP=19082
ATTESTATION_HUB_PORT_HTTPS=19445
AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password
MTWILSON_API_URL=https://127.0.0.1:8443/mtwilson/v2
MTWILSON_SERVER=127.0.0.1

CMS_TLS_CERT_SHA384=<CMS TLS digest>
BEARER_TOKEN=<Installation token from populate-users script>
AAS_API_URL=https://127.0.0.1:8444/aas
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1

ATTESTATION_HUB_DB_NAME="attestation_hub_pu"
ATTESTATION_HUB_DB_HOSTNAME=localhost
ATTESTATION_HUB_DB_PORTNUM="5432"
ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
ATTESTATION_HUB_DB_USERNAME=hub_db_user
ATTESTATION_HUB_DB_PASSWORD=password
```

3. Execute the installer binary.

### 14.3.7 Installing the Key Broker Service

#### 14.3.7.1 Prerequisites

The following must be completed before installing the Key Broker:

- The Verification Service must be installed and available

#### 14.3.7.2 Package Dependencies

#### 14.3.7.3 Supported Operating Systems


#### 14.3.7.4 Recommended Hardware

#### 14.3.7.5 Installation

1) Copy the Key Broker installation binary to the `/root/` directory.

2) Create the `kms.env` installation answer file:
3) Execute the KBS installer.

./kms-4.6.bin

14.3.7.6 Importing Verification Service Certificates

After installation, the Key Broker must import the SAML and PrivacyCA certificates from any Verification Services it will trust. This provides the Key Broker a way to ensure that only attestations that come from a “known” Verification Service. The SAML and PrivacyCA certificates needed can be found on the Verification Service.

14.3.7.6.1 Importing a SAML certificate

Use OpenSSL to display the SAML certificate content:

openssl x509 -in /opt/mtwilson/configuration/saml.crt.pem

Use the SAML certificate output in the following POST call to the Key Broker:

POST https://<Key Broker IP address or hostname>:443/v1/saml-certificates
Content-Type: application/x-pem-file

-----BEGIN CERTIFICATE-----
MIID9TCCAl2gAwIBAgIBCTANBgkqhkiG9w0BAQwFADBQMQswCQYDVQQGEwJVUzELMAkGA1UECBMCU0YxCzAJBgNVBAcTAlNDMQ4wDAYDVQQKEwVJTlRFTDEXMBUGA1UEAxMOQ01TIFNpZ25pbmcgQ0EwHhcNMTkxMjExMTkzOTU1WhcNMjAxMjExMTkzOTU1WjAYMHkwFAYDVQQDEw1cdHdpbHNvbizF1M1BojANBgkqhkiG9w0BAQEFAAOCA
AYAMMIIBigKCAYEArbrDpzR4Ry0MVhSJULHZiVL02GytpyRHtR2N1VXtpJzqmEA
Ep2ufcF8eGmST7DlpGB06KACPCz3pmqj3WzqZNTg7I7F2Z4Fu641fPcA35VW
31xZ1o5LeipjJ0OddT8k3h4hHz7HJhDw4J2fs2R8Gnn8B8GnQbmmGNrfqdxh8tMh
631k8xBNNHXsRbck27FtyN9hDU+z+RFriDvN1SIq9Fyndgqytk/m7ij3aEtkSF
bcCauuU7DFrdZm7B2GCF/7d95751S1Qnvan6uwqDTL674FX2sogVduu/WIyTpo
D2a7A41C3Bmk9945ScEoKZUz2hiphu+eB8E68W1t4rvox/NoI74e0k35AQq
Q3P0D5b+xXaXapz5ChC0PwiczA3A/9uw2f+CS2h2qDx8fWbAbL7i6od1Mn8+TDQZe
1x4650J6/82mRtaAy1EXlx3OT0Vxhl0u1b2J2X4t07+rHAIcoq+TOJ4a0rWGHg
kVCfiCuzzyT/w/RBaqMBAAGjEjAQMA4GA1UdWWEw/wQEAwIGAwEB/wQCCQA/CIGA4GA1Ud
-----END CERTIFICATE-----

14.3.7.6.2 Importing a PrivacyCA Certificate

Use OpenSSL to display the PrivacyCA certificate content:

openssl x509 -in /opt/mtwilson/configuration/PrivacyCA.pem

Use the SAML certificate output in the following POST call to the Key Broker:

POST https://<Key Broker IP address or hostname>:443/v1/privacy-ca-certificates
Content-Type: application/x-pem-file

-----BEGIN CERTIFICATE-----
/cre4K35sawZ3zrAgN8MPk0V/h2pJzJzWLNAd0w4/Pj1c5
YgTQ0RQ9JgerkweYBfrv303e9c22h48sFnhn6F3sbsCdwrs/swwJ0ocrPqygJE266H
DmIPvqjvcqJ131ndDBBkw+R1BFKz1Ycob9rsW16uVqbjBFdJ5QKodXkhqulybo
nJAFS5vmU+1JE
-----END CERTIFICATE-----
Use the PrivacyCA certificate output in the following POST call to the Key Broker:

```
POST https://<Key Broker IP address or hostname>:
[Key Broker Port]
v1/tpm-identity-certificates
Content-Type: application/x-pem-file

-----BEGIN CERTIFICATE-----
MIITaDCCD8gAwIBAgIGAQgQI0eW9MA0GCSqGSIb3DQEBCwUAMAMgMIIHQAIBAgIGAQgQECAwDQ
...
-----END CERTIFICATE-----
```

---

**14.3.8 Installing the Workload Policy Manager**

**14.3.8.1 Supported Operating Systems**


**14.3.8.2 Recommended Hardware**

- 2 vCPUs
- RAM: 8 GB
- 100 GB
• One network interface with network access to the Key Broker and Workload Service
• Additional memory and disk space may be required depending on the size of images to be encrypted

### 14.3.8.3 Installation

1) Copy the WPM installer to the `/root/` directory
2) Create the `wpm.env` answer file:

```bash
KMS_API_URL="https://127.0.0.1:9443/v1/"
WPM_SERVICE_USERNAME=wpm_service
WPM_SERVICE_PASSWORD=password
CMS_TLS_CERT_SHA384=<Sha384 hash of the CMS TLS certificate>
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1/
AAS_API_URL=https://127.0.0.1:8444/aas
BEARER_TOKEN=<Installation token from populate-users script>
```

3) Execute the WPM installer:

```
./workload-policy-manager-1.0.bin
```

### 14.3.9 Installing the Trust Agent and Workload Agent for Linux

The Intel® SecL – DC Trust Agent must be installed on each physical TPM/Intel® TXT-enabled server that will be registered and attested.

#### 14.3.9.1 Package Dependencies

The Trust Agent requires the following packages and their dependencies:

- trousers
- tboot (for TXT-based deployments without UEFI Secure Boot enabled only)
- tpm-quote-tools
- tpm2-tool
- openssl

If they are not already installed, the Trust Agent installer will attempt to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection.

If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

#### 14.3.9.2 Supported Operating Systems

The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.6.
14.3.9.3 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with Intel® TXT activated in the system BIOS
- TPM version 2.0 installed and activated in the system BIOS, with cleared ownership status
- System must be booted to a tboot boot option (Trust Agent installation will automatically install tboot if not present, and then require a reboot before proceeding)
- (Provisioning step only) Intel® SecL – DC Verification Service server installed and active

14.3.9.4 Installation

To install the Trust Agent for Linux:

1. Create the trustagent.env answer file in the /root/ directory:

```
MTWILSON_API_URL=https://<Verification Service IP or Hostname>:8443/mtwilson/v2
REGISTER_TPM_PASSWORD=y
PROVISION_ATTESTATION=y
GRUB_FILE=<path to grub.cfg>
CURRENT_IP=<Trust Agent IP address>
CMS_TLS_CERT_SHA384=<CMS TLS digest>
BEARER_TOKEN=<Installation user token from populate-users script>
AAS_API_URL=https://<AAS IP or Hostname>:8444/aas
CMS_BASE_URL=https://<CMS IP or Hostname>:8445/cms/v1
INSTALL_WORKLOAD_AGENT=y
WLA_SERVICE_USERNAME=wla_service
WLA_SERVICE_PASSWORD=passwordssss
TA_TLS_CERT_IP=<Comma-separated list of hostnames for the TAgent>
TA_TLS_CERT_DNS=<Comma-separated list of IP addresses for the TAgent>
WLS_API_URL=https://<Hostname or IP address of WLS>:5000/wls/
```

2. Copy the Trust Agent installer binary to the /root/ directory.

3. Execute the Trust Agent installer and wait for the installation to complete.

- The Trust Agent will install tboot and other prerequisites if not already present. Tboot will not be installed if the server is booted using UEFI SecureBoot, due to incompatibility.
- If tboot is installed by the Trust Agent installer, the installation will abort and reboot the host. This is because the Trust Agent requires the host to be booted into a tboot boot option, which populates the OS-level measurements in the host TPM.
- After the host reboots, re-run the Trust Agent installation binary to resume the installation.
- Installing with these env options will also install the Workload Agent
- After installation is complete, reboot the host to boot to the “TCB” boot option. This will allow the default Software Flavor measurements to be extended to the TPM, measuring the integrity of the Trust Agent and Workload Agent at system boot time.

14.4 Preparing Platform Integrity

Platform Integrity is a prerequisite for the Workload Confidentiality use cases. This section will walk through the steps needed to begin attesting the compute hosts and verify that they are all Trusted.

14.4.1.1 Register Hosts

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

Each Trust Agent host will need to be registered with a separate call.

POST https://verification.service.com:8443/mtwilson/v2/hosts
{
  "host_name": "<hostname of host to be registered>",
  "tls_policy_id" : "TRUST_FIRST_CERTIFICATE",
  "connection_string": "https://trustagent.server.com:1443 ",
  "flavorgroup_name" : "",
  "description" : "<description>"
}

14.4.1.2 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.


14.4.1.3 Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (BIOS, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host; the BIOS and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors
14.4.1.4 Import the Default Software Flavor

The Software Flavor must also be imported, but must only be imported once. Other Flavor parts may be imported from each host. The Software Flavor however can only be imported once for each version of the Trust Agent and Workload Agent, because the automatically created Flavor label must be unique.

POST https://verification.server.com:8443/mtwilson/v2/flavors

```
{
    "connection_string": "https://trustagent.server.com:1443",
    "partial_flavor_types": ["SOFTWARE"],
    "flavorgroup_name": "",
    "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

14.4.1.5 Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, you can retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

Each host should show an Overall trust status of “True”.

14.5 OpenStack Orchestration

In a production environment, it is strongly recommended to integrate with an orchestration service like OpenStack when enabling VM Confidentiality. This allows the orchestrator to attempt to launch workloads on nodes that should be compliant. Without the orchestrator attempting to land workloads on
compliant hosts, the workload may attempt to launch on a noncompliant node, which would fail due to the Workload Encryption protections.

Because demonstrating this use case includes a negative example where we intentionally launch on noncompliant hosts, the OpenStack orchestration feature will be skipped for this use case. See Section 10 for instructions on adding OpenStack orchestration if desired, after completing this walkthrough.

14.6 Demonstrating Workload Confidentiality with VMs

14.6.1 Encrypting an Image

Copy the sample CirrOS image to the WPM system. Use the WPM to encrypt the sample image. Assuming the sample image filename is “cirros-x86.img” and that you are running from the /root directory:

```
wpm create-image-flavor -l cirros-demo-image -i /root/cirros-x86.img -e /root/cirros-encrypted.img -o /root/cirros-demo-flavor
```

After generating the encrypted image with the WPM, the encrypted image must be uploaded to the OpenStack Glance. Note that the ID of the image in this Image Storage service must be retained and used for the next steps.

14.6.2 Uploading the Image Flavor

Generally it is easiest to run these API calls from the WPM system, as the files needed are located on that system from the previous steps. The image Flavor content will be the content of the “/root/cirros-demo-flavor” file created by the WPM in the previous step.

```
POST https://<Workload Service IP or Hostname>:5000/wls/flavors

<Image Flavor content from WPM output>
```

Use the above API request to upload the Image Flavor to the WLS. The Image Flavor will tell other Intel® SecL-DC components the Key Transfer URL for this image.

14.6.3 Creating the Image Flavor to Image ID Association

The WLS needs to know the ID of the image as it exists in OpenStack Glance and link it to the image Flavor. Use the below API request to create an association between the Image Flavor created in the previous step and the image ID. The Image Flavor ID is contained in “/root/cirros-demo-flavor”.

```
POST https://<Workload Service IP or Hostname>:5000/wls/images
```


14.6.4 Demonstrate Launching Encrypted VMs on Trusted Hosts

Instances of the protected images can now be launched as normal. Use the OpenStack APIs or GUI to launch an instance of the protected image. Encrypted images will only be accessible on hosts with a Platform Integrity Attestation report showing the host is trusted.

Because all hosts are currently Trusted, the VM should launch successfully.

14.6.5 Demonstrate Launching Encrypted VMs on Untrusted Hosts

Now that we have demonstrated a successful VM launch on Trusted hosts, we need to show that Untrusted hosts cannot launch protected workloads.

First, delete all running instances of the protected workload (do not delete the image) using OpenStack APIs or the OpenStack GUI.

Next, we will delete Flavors from the Verification Service. Without required Flavors, hosts will attest as Untrusted with the fault “Flavor part missing but required.”

1. Retrieve a list of all PLATFORM Flavors:

GET https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors?key=flavor_part&value=PLATFORM

The response will list one or more PLATFORM Flavors. Each will begin with a Meta section like this:

```
"signed_flavors" : [  
  "flavor" : {  
    "meta" : {  
      "id" : "583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3",  
      "description" : {  
        "flavor_part" : "PLATFORM",  
        "source" : "Q23RU28",  
        "label" : ...,  
        "bios_name" : "Intel Corporation",  
        "bios_version" : "SSE5C620.06B.0X.01.0155.073020181001",  
        "tpm_version" : "2.0",  
        "tboot_installed" : "true"  
      },  
      "bios_name" : "Intel Corporation",  
      "bios_version" : "SSE5C620.06B.0X.01.0155.073020181001",  
      "tpm_version" : "2.0",  
      "tboot_installed" : "true"  
    },  
    "id" : "583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3",  
    "description" : {  
      "flavor_part" : "PLATFORM",  
      "source" : "Q23RU28",  
      "label" : ...,  
      "bios_name" : "Intel Corporation",  
      "bios_version" : "SSE5C620.06B.0X.01.0155.073020181001",  
      "tpm_version" : "2.0",  
      "tboot_installed" : "true"  
    },  
    "id" : "583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3",  
    "description" : {  
      "flavor_part" : "PLATFORM",  
      "source" : "Q23RU28",  
      "label" : ...,  
      "bios_name" : "Intel Corporation",  
      "bios_version" : "SSE5C620.06B.0X.01.0155.073020181001",  
      "tpm_version" : "2.0",  
      "tboot_installed" : "true"  
    }  
  },  
  "id" : "583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3",  
  "description" : {  
    "flavor_part" : "PLATFORM",  
    "source" : "Q23RU28",  
    "label" : ...,  
    "bios_name" : "Intel Corporation",  
    "bios_version" : "SSE5C620.06B.0X.01.0155.073020181001",  
    "tpm_version" : "2.0",  
    "tboot_installed" : "true"  
  }  
];
```

Note the ID in bold.

2. For each Flavor ID returned, delete that Flavor:
DELETE  https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors/<flavor ID>

For example:

DELETE https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors/583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3

The response for each DELETE should be a 204 No Content.

3. List the Flavors again:

GET https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors?key=flavor_part&value=PLATFORM

You should see an empty array indicating no PLATFORM Flavors remain.

4. Generate new reports

For each host, we now need to create a new attestation report. These will automatically generate, but we can speed the process by forcing new reports immediately:

POST https://<Verification Service IP or hostname>:8443/mtwilson/v2/reports

Headers:
Accept: application/json
input: {"host_name":"<Trust Agent host>"}

Repeat this call for each Trust Agent host. The reports returned should all show the Overall trust status is now False.

5. Try to launch the protected image

Using the OpenStack API or GUI, launch a new instance of the protected image.

In this case, the launch should fail.

Behind the scenes, the Workload Agent on the compute host attempted to retrieve a key for the image. The key transfer request was sent to the Key Broker using a report for that host from the Verification Service. Because the host in Untrusted, the key transfer request was denied, and the image remained encrypted and inaccessibly to the hypervisor, causing a failure.
15 Use Case 6b: Workload Confidentiality with Docker Containers

15.1 Scope

This use case will build on Use Case 5 to demonstrate using Intel® SecL – DC to control where Docker container workloads are allowed to be accessed using encryption. The installation of Kubernetes will not be described here. For Kubernetes installation instructions, see Kubernetes documentation.

A sample container image is required. For the sake of this POC, we generally recommend a small test image like the “hello-world” example container from Docker:

https://hub.docker.com/_/hello-world

15.2 Requirements

15.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration
  - The system must either have Intel® TXT supported and enabled in the system BIOS, OR must have Intel® Boot Guard supported in a “measurement” profile
  - If Intel® Boot Guard is used but Intel® TXT is disabled, UEFI Secure Boot must be enabled.

- Each physical server must have a physical or firmware Trusted Platform Module installed and active in the system BIOS. TPM 2.0 is supported for Linux*. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS).

- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service, Workload Service, Integration Hub, Authentication and Authorization Service, Certificate Management Service, Key Broker Service, and Workload Policy Manager (all of these services may reside on a single server). This server does not require a TPM or Intel® TXT, but does require network access to the other POC servers.

15.2.2 Software

- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the
PostgreSQL JDBC* driver (for example, SQL Workbench*) installed on a laptop or other system outside of the POC environment, but with network connectivity.

- cURL* or a REST* API utility like Postman* to execute API requests.

15.2.3 Operating System

- The Intel® SecL – DC services support Red Hat Enterprise* Linux (RHEL*) 7.6.

15.2.4 Networking

- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC installers will require access to package repositories; either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

15.2.5 Kubernetes

- Kubernetes Master Node must be installed and running
- A minimum of one Kubernetes Worker Node which will have the Trust Agent installed (instructions described below).

15.3 Installation

The Workload Integrity use case requires eight Intel® SecL – DC services: the Certificate Management Service, the Authentication and Authorization Service, the Verification Service, the Workload Servicer, the Key Broker Service, the Workload Policy Manager, the Workload Agent, and the Trust Agent.

15.3.1 Installing the Database Server

For the purposes of POCs, a single shared database server will be used for all of the services that require database access. Each service will use its own database schema.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
Install a sample Postgresql 11 database using the install_pgdb.sh script. This script will automatically install the Postgresql database and client packages required.

Add the Postgresql 11 repository:

https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Create the iseclpgdb.env answer file:

ISECL_PGDB_IP_INTERFACES=localhost
ISECL_PGDB_PORT=5432
ISECL_PGDB_SAVE_DB_INSTALL_LOG=true

Execute the installation script:

./install_pgdb.sh

**Note:** the database installation only needs to be performed once if the same database server will be used for all services that require a database. Only the “create_db” step need to be repeated if the database server will be shared.

After installation, the database schemas must be created initialized and tables created for each service.

./create_db.sh aas_db aas_db_user password
./create_db.sh mw_as vs_db_user password
./create_db.sh wls_db wls_db_user password
./create_db.sh attestation_hub_pu hub_db_user password

### 15.3.2 Installing the Certificate Management Service

#### 15.3.2.1 Supported Operating Systems


#### 15.3.2.2 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

#### 15.3.2.3 Installation

To install the Intel® SecL-DC Certificate Management Service:

2. Create the cms.env installation answer file for an unattended installation:

```
AAS_TLS_SAN=<comma-separated list of IPs and hostnames for the AAS>
AAS_API_URL=https://<Authentication and Authorization Service IP or Hostname>:8444/aas
```

The SAN list will be used to authenticate the Certificate Signing Request from the AAS to the CMS. Only a CSR originating from a host matching the SAN list will be honored. Later, in the AAS authservice.env installation answer file, this same SAN list will be provided for the AAS installation. These lists must match, and must be valid for IPs and/or hostnames used by the AAS system. If both the AAS and CMS will be installed on the same system, “127.0.0.1,localhost” may be used. The SAN list variables also accept the wildcards “?” (for single-character wildcards) and “*” (for multiple-character wildcards) to allow address ranges or multiple FQDNs.

The AAS_API_URL represents the URL for the AAS that will exist after the AAS is installed.

For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Certificate Management Service.

3. Execute the installer binary.

```
./certificate-management-service-1.0.bin
```

When the installation completes, the Certificate Management Service is available. The services can be verified by running `cms status` from the command line.

```
# cms status
```

After installation is complete, the CMS will output a bearer token to the console. This token will be used with the AAS during installation to authenticate certificate requests to the CMS. If this token expires or otherwise needs to be recreated, use the following command:

```
cms setup cms_auth_token --force
```

In addition, the SHA384 digest of the CMS TLS certificate will be needed for installation of the remaining Intel® SecL services. The digest can be obtained using the following command:

```
cms tlscertsha384
```

### 15.3.3 Installing the Authentication and Authorization Service

#### 15.3.3.1 Required For

The AAS is REQUIRED for all use cases.

- Platform Integrity with Data Sovereignty and Signed Flavors
15.3.3.2 Prerequisites

The following must be completed before installing the Authentication and Authorization Service:

- The CMS must be installed and available

15.3.3.3 Supported Operating Systems


15.3.3.4 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 10 GB
- One network interface with network access to all Intel® SecL-DC services

15.3.3.5 Installation

To install the AAS, a bearer token from the CMS is required. This bearer token is output at the end of the CMS installation. However, if a new token is needed, simply use the following command from the CMS command line:

```
cms setup cms_auth_token --force
```

Create the authservice.env installation answer file:

```
CMS_BASE_URL=https://<CMS IP or hostname>:8445/cms/v1/
CMS_TLS_CERT_SHA384=<CMS_TLS certificate sha384>
AAS_DB_HOSTNAME=<IP or hostname of database server>
AAS_DB_PORT=5432
AAS_DB_NAME=aas_db
AAS_DB_USERNAME=aas_db_user
AAS_DB_PASSWORD=password
AAS_ADMIN_USERNAME=aas-admin
AAS_ADMIN_PASSWORD=password
SAN_LIST=<comma-separated list of IPs and hostnames for the AAS>
BEARER_TOKEN=<bearer token from CMS installation>
```

**Note:** the AAS_ADMIN credentials specified in this answer file will have administrator rights for the AAS and can be used to create other users, create new roles, and assign roles to users.
Execute the AAS installer:

```
./authservice-v1.6.bin
```

### 15.3.3.5.1 Creating Users

After installation is complete, a number of roles and user accounts must be generated. Most of these accounts will be service users, used by the various Intel® SeL services to work together. Another set of users will be used for installation permissions, and a final administrative user will be created to provide the initial authentication interface for the actual human user. The administrative user can be used to create additional users with appropriately restricted roles based on organizational needs.

Creating these required users and roles is facilitated by a script that will accept credentials and some configuration settings from an answer file, and will automate the process.

Create the populate-users.env file:

```bash
ISECL_INSTALL_COMPONENTS=HVS,KBS,TA,VS,WPM,AH,VS,AAS
AAS_API_URL=https://127.0.0.1:8444/aas
AAS_ADMIN_USERNAME=aas_admin
AAS_ADMIN_PASSWORD=password
VS_CERT_SAN_LIST=* 
AH_CERT_SAN_LIST=* 
TA_CERT_SAN_LIST=* 
WLS_CERT_SAN_LIST=* 
KBS_CERT_SAN_LIST=* 
VS_SERVICE_USERNAME=verification_service
VS_SERVICE_PASSWORD=password 
AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password 
WPM_SERVICE_USERNAME=wpm_service
WPM_SERVICE_PASSWORD=password 
WLS_SERVICE_USERNAME=wls_service
WLS_SERVICE_PASSWORD=password 
WLA_SERVICE_USERNAME=wla_service
WLA_SERVICE_PASSWORD=password 
GLOBAL_ADMIN_USERNAME=admin
GLOBAL_ADMIN_PASSWORD=password
```

**Note:** The `ISECL_INSTALL_COMPONENTS` variable is a comma-separated list of the components that will be used in your environment. Not all services are required for every use case. If a given service will not be used in your deployment, simply delete the unnecessary service abbreviation from the `ISECL_INSTALL_COMPONENTS` list, and leave the SAN and credential variables for that service blank.
NOTE: The SAN list variables each support wildcards ("*") and ("?"). In particular, without wildcards the Trust Agent SAN list would need to explicitly list each hostname or IP address for all Trust Agents that will be installed, which is not generally feasible. Using wildcards, domain names and entire IP ranges can be included in the SAN list, which will allow any host matching those ranges to install the relevant service.

The GLOBAL_ADMIN credentials will be used to create an administrative user with all permissions.

Execute the populate-users script:

```
./populate-users
```

Note: The script can be executed with the –output_json argument to create the populate-user.json. This json output file will contain all of the users created by the script, along with usernames, passwords, and role assignments. This file can be used both as a record of the service and administrator accounts, and can be used as alternative inputs to recreate the same users with the same credentials in the future if needed. Be sure to protect this file if the –output_json argument is used.

The script will automatically generate the following users:
Verification Service User
Attestation Hub Service User
Global Admin User

These user accounts will be used during installation of several of the Intel® SecL-DC applications. In general, whenever credentials are required by an installation answer file, the variable name should match the name of the corresponding variable used in the populate-user.env file.

In addition, the populate-users script will also output a “Bearer Token.” This token will be used to authenticate during installation of other services.

### 15.3.3.6 Creating an Authentication Token

Intel® SecL-DC uses a Bearer Token authentication schema. Tokens are issued by the AAS and can be used as authentication for other services. By default, tokens are valid for 2 hours, after which a new token will be needed.

To issue a new token, use the following API call:

```
POST https://<AAS IP or hostname>:8444/aas/token
{
    "username": "admin",
    "password": "password"
}
```
For the purposes of this document, it is easiest to simply use the default Administrator password for all API requests. In a production environment it would be strongly recommended to create users with more restrictive roles based on the access needed.

Use this token as authentication for all API requests in this document. For example, use a header that looks like the following:

“Authorization: Bearer <token content>”

15.3.4 Installing the Verification Service

15.3.4.1 Package Dependencies

The Intel® Security Libraries Verification Service requires the following packages and their dependencies:

- Monit
- Logback (optional)
- Java* 8 JDK
- OpenSSL
- Unzip

If they are not already installed, the Verification Service installer attempts to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

15.3.4.2 Supported Operating Systems


15.3.4.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers
- (Optional) One network interface for Asset Tag provisioning (only required for “pull” tag provisioning; required to provision Asset Tags to VMware ESXi servers).
15.3.4.4 Installation

15.3.4.4.1

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file for an unattended installation.

Execute the installer binary.

```
MTWILSON_SERVER=<IP address or hostname of the Verification Service>
MTWILSON_API_URL="https://<Verification Service IP or hostname>:8443/mtwilson/v2"
DATABASE_HOSTNAME="127.0.0.1"
DATABASE_USERNAME=vs_db_user
DATABASE_PORTNUM="5432"
DATABASE_PASSWORD=password
DATABASE_SCHEMA="mw_as"
VS_SERVICE_USERNAME=verification_service
VS_SERVICE_PASSWORD=password
CMS_TLS_CERT_SHA384 =<Certificate Management Service TLS digest>
BEARER_TOKEN=<Installation user token>
AAS_API_URL=https://127.0.0.1:8444/aas
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1

./host-verification-service-linux-4.6.bin
```

When the installation completes, the Verification Service is available. The services can be verified by running `mtwilson status` from the Verification Service command line.

`# mtwilson status`

15.3.5 Installing the Workload Service

15.3.5.1 Package Dependencies

15.3.5.2 Supported Operating Systems


15.3.5.3 Recommended Hardware

15.3.5.4 Installation

1. Copy the Workload Service installation binary to the /root/ directory.
2. Create the `workload-service.env` installation answer file:
Execute the WLS installer binary:
./workload-service-1.0.bin

15.3.6 Installing the Intel® SecL-DC Custom Resource Definitions for Kubernetes

Intel® SecL uses Custom Resource Definitions to add the ability to base orchestration decisions on Intel® SecL security attributes to Kubernetes. These CRDs allow Kubernetes administrators to configure pods to require specific security attributes so that the Kubernetes Master Node will schedule those pods only on Worker Nodes that match the specified attributes.

Perform the following steps on the Kubernetes Master Node:

1) Add a mount path to the kube-scheduler.yaml file for the Intel SecL scheduler extension:

   - mountPath: /opt/isecl-k8s-extensions/bin/
     name: extendedsched
     readOnly: true

2) Add a volume path to the kube-scheduler.yaml file for the Intel SecL scheduler extension:

   - hostPath:
     path: /opt/isecl-k8s-extensions/bin/
3) Copy the isecl-k8s-extensions.bin installer to the Kubernetes Master and execute the installer

   ./isecl-k8s-extensions.bin

4) The installer will output a set of keystores upon completion into `attestation-hub-keystores/`. These contain keys that will be used by the Integration Hub to communicate with this Kubernetes Master. Copy the contents of this directory to the Integration Hub:

   scp -r /root/attestation-hub-keystores/* root@integration-hub.server.com:/opt/attestation-hub/configuration/

   Note that the Integration Hub can manage multiple Kubernetes Master environments at the same time, but the keystores must be kept separate. To do this, create subfolders in the Hub configuration directory for each separate Kubernetes environment, and copy the appropriate keystores to the matching subfolder.

5) Copy the Integration Hub public key to the Kubernetes Master:

   scp attestation-hub.server.com:/opt/attestation-hub/configuration/hub_public_key.pem /etc/kubernetes/pki/

6) Run the command `systemctl restart kubelet` to restart all the control plane container services, including the base scheduler.

7) (Optional) Verify that the Intel® SecL Custom Resource Definitions have been started:

   kubectl get crds
   kubectl get -o json hostattributes.isecl.intel.com

**15.3.7 Installing the Integration Hub**

To install the Integration Hub, follow these steps:

3. Copy the Integration Hub installation binary to the `/root/` directory.
4. Create the `attestation-hub.env` installation answer file. See the sample file below.

```
ATTESTATION_HUB_PORT_HTTP=19082
ATTESTATION_HUB_PORT_HTTPS=19445
AH_SERVICE_USERNAME=hub_service
AH_SERVICE_PASSWORD=password
MTWILSON_SERVER=127.0.0.1
MTWILSON_API_URL=https://127.0.0.1:8443/mtwilson/v2
CMS_TLS_CERT_SHA384=<Sha384 digest of CMS TLS certificate>
BEARER_TOKEN=<Installation token from populate-users script>
AAS_API_URL=https://127.0.0.1:8444/aas
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1
ATTESTATION_HUB_DB_NAME="attestation_hub_pu"
ATTESTATION_HUB_DB_HOSTNAME="localhost"
ATTESTATION_HUB_DB_PORTNUM="5432"
ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
ATTESTATION_HUB_DB_USERNAME=hub_db_user
ATTESTATION_HUB_DB_PASSWORD=password
```

5. Execute the installer binary.

15.3.8 Installing the Key Broker Service

15.3.8.1 Prerequisites

The following must be completed before installing the Key Broker:

- The Verification Service must be installed and available

15.3.8.2 Package Dependencies

15.3.8.3 Supported Operating Systems


15.3.8.4 Recommended Hardware

15.3.8.5 Installation

1) Copy the Key Broker installation binary to the `/root/` directory.

2) Create the `kms.env` installation answer file:

```
AAS_API_URL=https://127.0.0.1:8444/aas
CMS_BASE_URL=https://127.0.0.1:8445/cms/v1/
KMS_TLS_CERT_IP=<comma-separated list of IP addresses for the Key Broker>,127.0.0.1
KMS_TLS_CERT_DNS=<comma-separated list of hostnames for the Key Broker>,localhost
```
CMS_TLS_CERT_SHA384=<SHA384 hash of CMS TLS certificate>
BEARER_TOKEN=<installation token from populate-users script>

3) Execute the KBS installer.

./kms-4.6.bin

15.3.8.6 Importing Verification Service Certificates

After installation, the Key Broker must import the SAML and PrivacyCA certificates from any Verification Services it will trust. This provides a way to ensure that only attestations that come from a "known" Verification Service. The SAML and PrivacyCA certificates needed can be found on the Verification Service.

15.3.8.6.1 Importing a SAML certificate

Use OpenSSL to display the SAML certificate content:

openssl x509 -in /opt/mtwilson/configuration/saml.crt.pem

Use the SAML certificate output in the following POST call to the Key Broker:

POST https://<Key Broker IP address or hostname>:443/v1/saml-certificates
Content-Type: application/x-pem-file

-----BEGIN CERTIFICATE-----
MIID9TCCAl2gAwIBAgIBCTANBgkqhkiG9w0BAQwFADBQMQswCQYDVQQGEwJVUzEL
MAkGA1UECBMCU0YxCzAJBgNVBAcTAlNDMQ4wDAYDVQQKEwVJTlRFTDEXMBUGA1UE
AxMQOQ1TIHFnpz25pbmcgQ0EwHhcNMTkxOTU1WhcnMjAxM1ExMTkzOTU1
WJAVMRYwFAYDVQDDERW1dtdpHNBvbil2YW1iMIIBojANBgkqhkiG9w0BAQEDFAAOCA
AY8AM9IBi8gKCAEArbrDpsHR4y0MVhSJULH0oIvLOU20YqyRh+R2N1VXTpJzgmEA
Ep2utC98+mSCT7DlpGB06KACPCCz3pmqj3WygZNTysG7FZ4Bu641FPCx3A5WVH
31xZe1LSe4j93dF8k4hXHJvJDsW42fs2RGn8bbZG/QbmmNRFydxh08bMh
63ik88BNM6YyvSrBck2Y9h0U+s+xFfIDNv1ISiQ9PyndgOytK/m7ljoAetksF
bCsaauUL7dfDhZmB2GF/C/9d957S1Gnmv6uWqD7TL674F42zq59Vduu/wyToP0
/6D2aA7u1CR38mk9945TSe4DKZn2hKhuu+48ES68Wit4rvox/Noi174e0k3QaCQ
Q3OmpoDxAkAraps5ChCw0w0OpZ3A/8wN2f+CS2HqDs88FaA7hKh718od8N8+TDQe
1/4e/50JE/8sNR/tcAy1EXkm3OTOVXihi0u1Bj84x4Q0T+zHAICOQ+TOJ4A0AtW
kVCI1zCUyTY/7/WBAgBMAACjE+JQAA4MGAAA1U0dWEB/wqEAW1GwDNBqRghkl1G9W0
BAQwFAoCAYEAP/ABH9FqMnMCVU7v7FvL8MjIfFjymA15mCOrmE1KZd7To9NOPEPfU
pFyRA4TuljFjOLClj41jKXnu4Lt6HJZ2bzn8JR7550C1J6GxWbQA664xy30o9IG
otOk1pZ5w5LwP78J7zQ26J+N5HnpdL1W5US/6/vwil3zhJ92KJnaKp45qd52nKZUjdx
YXmKXTEqG2OMVjaLirqomf6Vxbb62OH2ZG8Fgqu3H5yMngqerXn1XMiAv30CyrcMDL1
rJGQR6oQzFzywPC0uWpBw5t2a6tvd4+8T2K3q/dKKO1FURJ/mq70eqVW91y
TOKY1V9mMKeXzIqL0/cre4K35acCW3A2rzGgNM8P0V/hzvP82HrWLNAd04w/Pj1C
Yg7ROQ8jQzerkeWYBfrz303e9c22h48fHnun6F3abcDjws/ss/WJ1OcrFyJQE26HY
DM1Fqrvjgq0jI31nDDBwkk+R1BFkz1Ycobb9rsW16uVqMfBFDJ5QKOoXhquylooba
JAF55vmlh+1JE
-----END CERTIFICATE-----

15.3.8.6.2 Importing a PrivacyCA Certificate

Use OpenSSL to display the PrivacyCA certificate content:

openssl x509 -in /opt/mtwilson/configuration/PrivacyCA.pem

Use the PrivacyCA certificate output in the following POST call to the Key Broker:

POST https://<Key Broker IP address or hostname>:443/v1/privacyca-certificates
Content-Type: application/x-pem-file

-----BEGIN CERTIFICATE-----
MIID4TCCAl2gAwIBAgIBCTANBgkqhkiG9w0BAQwFADBQMQswCQYDVQQGEwJVUzEL
MAkGA1UECBMCU0YxCzAJBgNVBAcTAlNDMQ4wDAYDVQQKEwVJTlRFTDEXMBUGA1UE
AxMQOQ1TIHFnpz25pbmcgQ0EwHhcNMTkxOTU1WhcnMjAxM1ExMTkzOTU1
WJAVMRYwFAYDVQDDERW1dtdpHNBvbil2YW1iMIIBojANBgkqhkiG9w0BAQEDFAAOCA
AY8AM9IBi8gKCAEArbrDpsHR4y0MVhSJULH0oIvLOU20YqyRh+R2N1VXTpJzgmEA
Ep2utC98+mSCT7DlpGB06KACPCCz3pmqj3WygZNTysG7FZ4Bu641FPCx3A5WVH
31xZe1LSe4j93dF8k4hXHJvJDsW42fs2RGn8bbZG/QbmmNRFydxh08bMh
63ik88BNM6YyvSrBck2Y9h0U+s+xFfIDNv1ISiQ9PyndgOytK/m7ljoAetksF
bCsaauUL7dfDhZmB2GF/C/9d957S1Gnmv6uWqD7TL674F42zq59Vduu/wyToP0
/6D2aA7u1CR38mk9945TSe4DKZn2hKhuu+48ES68Wit4rvox/Noi174e0k3QaCQ
Q3OmpoDxAkAraps5ChCw0w0OpZ3A/8wN2f+CS2HqDs88FaA7hKh718od8N8+TDQe
1/4e/50JE/8sNR/tcAy1EXkm3OTOVXihi0u1Bj84x4Q0T+zHAICOQ+TOJ4A0AtW
kVCI1zCUyTY/7/WBAgBMAACjE+JQAA4MGAAA1U0dWEB/wqEAW1GwDNBqRghkl1G9W0
BAQwFAoCAYEAP/ABH9FqMnMCVU7v7FvL8MjIfFjymA15mCOrmE1KZd7To9NOPEPfU
pFyRA4TuljFjOLClj41jKXnu4Lt6HJZ2bzn8JR7550C1J6GxWbQA664xy30o9IG
otOk1pZ5w5LwP78J7zQ26J+N5HnpdL1W5US/6/vwil3zhJ92KJnaKp45qd52nKZUjdx
YXmKXTEqG2OMVjaLirqomf6Vxbb62OH2ZG8Fgqu3H5yMngqerXn1XMiAv30CyrcMDL1
-----END CERTIFICATE-----
15.3.9 Installing the Workload Policy Manager

15.3.9.1 Supported Operating Systems


15.3.9.2 Prerequisites

- Docker-ce 19.03 must be installed.

15.3.9.3 Recommended Hardware

- 2 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to the Key Broker and Workload Service
- Additional memory and disk space may be required depending on the size of images to be encrypted

### 15.3.9.4 Installation

1) Copy the WPM installer to the `/root/` directory
2) Create the `wpm.env` answer file:

```bash
KMS_API_URL=“https://127.0.0.1:9443/v1/"
WPM_SERVICE_USERNAME=wpm_service
WPM_SERVICE_PASSWORD=password
CMS_TLS_CERT_SHA384=<Sha384 hash of the KBS TLS certificate>
CMS_BASE_URL=“https://127.0.0.1:8445/cms/v1/"
AAS_API_URL= https://127.0.0.1:8444/aas
BEARER_TOKEN=<Installation token from populate-users script>
WPM_WITH_CONTAINER_SECURITY="yes"
```

3) Execute the WPM installer:
   ```bash
   ./workload-policy-manager-1.0.bin
   ```

### 15.3.10 Installing the Trust Agent and Workload Agent for Linux

The Intel® SecL – DC Trust Agent must be installed on each physical TPM/Intel® TXT-enabled server that will be registered and attested.

#### 15.3.10.1 Package Dependencies

The Trust Agent requires the following packages and their dependencies:
- trousers
- tboot (for TXT-based deployments without UEFI Secure Boot enabled only)
- tpm-quote-tools
- tpm2-tool
- openssl

If they are not already installed, the Trust Agent installer will attempt to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection.

If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.
15.3.10.2 **Supported Operating Systems**

The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.6.

15.3.10.3 **Prerequisites**

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with Intel® TXT activated in the system BIOS
- TPM (version 1.2 or 2.0) installed and activated in the system BIOS, with cleared ownership status
- System must be booted to a tboot boot option (Trust Agent installation will automatically install tboot if not present, and then require a reboot before proceeding)
- (Provisioning step only) Intel® SecL – DC Verification Service server installed and active

15.3.10.4 **Installation**

To install the Trust Agent for Linux:

1. Create the `trustagent.env` answer file in the `/root/` directory:

   ```
   MTWILSON_API_URL=https://<Verification Service IP or Hostname>:8443/mtwilson/v2
   REGISTER_TPM_PASSWORD=y
   PROVISION_ATTESTATION=y
   GRUB_FILE=<path to grub.cfg>
   CURRENT_IP=<Trust Agent IP address>
   CMS_TLS_CERT_SHA384=<CMS TLS digest>
   BEAKER_TOKEN=<Installation user token from populate-users script>
   AAS_API_URL=https://<AAS IP or Hostname>:8444/aas
   CMS_BASE_URL=https://<CMS IP or Hostname>:8445/cms/v1
   INSTALL_WORKLOAD_AGENT=y
   WLA_SERVICE_USERNAME=wla_service
   WLA_SERVICE_PASSWORD=passwordssss
   WA_WITH_CONTAINER_SECURITY="yes"
   NO_PROXY=<Registry_ip>
   HTTPS_PROXY=<proxy_url>
   REGISTRY_SCHEME_TYPE=https
   TA_TLS_CERT_IP=<Comma-separated list of hostnames for the TAgent matching the SAN list specified in the populate-users script; may include wildcards>
   TA_TLS_CERT_DNS=<Comma-separated list of IP addresses for the TAgent matching the SAN list specified in the populate-users script; may include wildcards>
   WLS_API_URL=https://<Hostname or IP address of WLS>:5000/wls/
   ```

2. Copy the Trust Agent installer binary to the `/root/` directory.
3. Execute the Trust Agent installer and wait for the installation to complete.

   - The Trust Agent will install tboot and other prerequisites if not already present. Tboot will **not** be installed if the server is booted using UEFI SecureBoot, due to incompatibility.
- If tboot is installed by the Trust Agent installer, the installation will abort and reboot the host. This is because the Trust Agent requires the host to be booted into a tboot boot option, which populates the OS-level measurements in the host TPM.

- After the host reboots, re-run the Trust Agent installation binary to resume the installation.

- Installing with these env options will also install the Workload Agent

- After installation is complete, reboot the host to boot to the “TCB” boot option. This will allow the default Software Flavor measurements to be extended to the TPM, measuring the integrity of the Trust Agent and Workload Agent at system boot time.

### 15.3.11 Preparing Platform Integrity

Platform Integrity is a prerequisite for the Workload Confidentiality use cases. This section will walk through the steps needed to begin attesting the compute hosts and verify that they are all Trusted.

#### 15.3.11.1 Register Hosts

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

Each Trust Agent host will need to be registered with a separate call.

```json
POST https://verification.service.com:8443/mtwilson/v2/hosts
{
    "host_name": "<hostname of host to be registered>",
    "tls_policy_id" : "TRUST_FIRST_CERTIFICATE",
    "connection_string": "https://trustagent.server.com:1443 ",
    "flavorgroup_name" : "",
    "description" : "<description>"
}
```

#### 15.3.11.2 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.

```bash
```

#### 15.3.11.3 Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (BIOS, OS, and HOST_UNIQUE) from each Trust Agent host.
Technically, only the HOST_UNIQUE part must come from each host; the BIOS and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
   "connection_string": "https://trustagent.server.com:1443",
   "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
   "flavorgroup_name": "",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

### 15.3.11.4 Import the Default Software Flavor

The Software Flavor must also be imported, but must only be imported once. Other Flavor parts may be imported from each host. The Software Flavor however can only be imported once for each version of the Trust Agent and Workload Agent, because the automatically created Flavor label must be unique.

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
   "connection_string": "https://trustagent.server.com:1443",
   "partial_flavor_types": ["SOFTWARE"],
   "flavorgroup_name": "",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

### 15.3.11.5 Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, you can retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

Each host should show an Overall trust status of “True”.
15.4 Kubernetes Integration

15.4.1 Configuring a Tenant in the Integration Hub

15.4.1.1 Create the Tenant

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. A Tenant defines the connection and authentication details to reach the OpenStack services.

POST https://hub.server.com:19445/v1/tenants

{
    "name": "DemoTenant",
    "plugins": [
        {
            "name": "kubernetes",
            "properties": [
                {
                    "key": "api.endpoint",
                    "value": "https://kubernetes-master.server.com:6443"
                },
                {
                    "key": "tenant.name",
                    "value": "DemoTenant"
                },
                {
                    "key": "plugin.provider",
                    "value": "com.intel.attestationhub.plugin.kubernetes.KubernetesPluginImpl"
                },
                {
                    "key": "kubernetes.client.keystore",
                    "value": "/opt/attestation-hub/configuration/root_k8s_client.jks"
                },
                {
                    "key": "kubernetes.server.keystore",
                    "value": "/opt/attestation-hub/configuration/root_k8s_trust.jks"
                },
                {
                    "key": "kubernetes.server.keystore.password",
                    "value": "<Keystore password>"
                },
                {
                    "key": "kubernetes.client.keystore.password",
                    "value": "<Keystore Password>"
                }
            ]
        }
    ]
}
**NOTE:** the value of `kubernetes.client.keystore` and `kubernetes.server.keystore` must be the filesystem path on the Integration Hub that contains the Kubernetes Master keystores output from the scheduler extensions for this tenant. The value of `kubernetes.server.keystore.password` and `kubernetes.server.keystore.password` must be the keystore passwords output by the scheduler extensions installer.

### 15.4.1.2 List Hosts

The Integration Hub periodically queries the Verification Service for the list of all new Reports; only Reports generated after the timestamp of the most recent query are returned. Because host registration will trigger the generation of a new Report, any new hosts added to the Verification Service will be seen in the Hub on the next refresh (determined by the value of the POLL_INTERVAL variable during install).

The list of hosts known to the Integration Hub can be retrieved using the below API sample.

```
GET https://server.com:19445/v1/hosts
```

### 15.4.1.3 Assign Hosts to Tenants

Hosts must be assigned to a tenant before Intel SecL-DC security attributes will be pushed to Kubernetes. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of `hardware_uuids`.

```
POST https://server.com:19445/v1/host-assignments
{
  "tenant_id": "DC02284A-F525-4094-BA01-E317FE28E15F",
  "hardware_uuids": [ 
    "00886b98-994d-e411-906e-0017a4403562"
  ]
}
```

The Hub will “push” Intel SecL-DC attributes as OpenStack Traits to the tenant’s configured endpoints (in this case, Nova) every time it looks for new attestations.

### 15.4.1.4 Assign Hosts to the Tenant

Hosts must be assigned to a tenant before Intel® SecL – DC security attributes will be pushed to the OpenStack Traits. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of `hardware_uuids`.

Hosts are assigned using the Tenant ID (returned in the Create Tenant step) and the Hardware UUID of one or more hosts. List each OpenStack Compute Node’s Hardware UUID in the array.
POST https://hub.server.com:19445/v1/host-assignments
{
  "tenant_id": "<Tenant ID>",
  "hardware_uuuids": [
    "<Host 1 Hardware UUID>", "<Host 2 Hardware UUID>"
  ]
}

15.4.1.5 Verify that the Hub is Retrieving Reports

Next we want to list the hosts as seen in the Hub to ensure the Hub is communicating with the Verification Service and retrieving Reports.
GET https://hub.server.com:19445/v1/hosts

This will return a list of all of the hosts seen by the Hub with their most recent Report status. By default, the Hub will poll the Verification Service for new Reports every 2 minutes, refresh this list and then send updates to all Tenant endpoints according to which hosts were assigned to which Tenant.

15.4.2 Configuring Pods to Require Intel® SecL Attributes

1) Verify that the worker nodes have had their Intel® SecL security attributes populated:

    kubectl get nodes --show-labels

    The output should show the Trust status and any Asset Tags applied to all of the registered Worker Nodes.

2) We can now configure a Pod to require a Trusted status. Add the following to the Pod creation files:

    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: isecl.trusted
                    operator: In
                    values:
                      - "true"
Below is a full sample Pod config file:

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
            - key: isecl.trusted
              operator: In
              values:
              - "true"
  containers:
  - image: hello-world
    imagePullPolicy: IfNotPresent
    name: hello-world
```

The “isecl.trusted” key defines the requirement for a Trusted host. Only one of these keys should be used. The “TAG_” keys indicate Asset Tags; if the workload should only launch on hosts with the “COUNTRY=USA” Asset Tag, the pod should be launched with the matchExpression key “TAG_COUNTRY” with the value “USA”.

**Note:** All of the matchExpression definitions must be true for a given worker node to launch the pod – in the example above, the host must be attested as Trusted with Asset Tags “Country=US,” “Customer=Customer1,” and “State=CA”. If the worker node has additional Asset Tags beyond the ones required, the pod will still be able to be launched on that node. However, if one of the specified Tags is missing or has a different value, that worker node will not be used for that pod.
15.4.3 Launching a Pod (unencrypted) on a Compliant Worker Node

Simply launch any Pod using a config file containing the matchExpression policy requirements for Trust and Asset Tags. The Intel® SecL CRDs on the Kubernetes Master will use the Worker Node labels propagated from the Intel® SecL Integration Hub to ensure the Pod is launched according to the policy requirements.

Below is a sample command for the Kubernetes Master to launch a Pod:

```
kubectl create -f pod_config.yaml
```

We can verify that the Master Node CRDs contain attributes successfully pushed from the Integration Hub:

```
kubectl get crds
kubectl get -o json hostattributes.isecl.intel.com
```

We can list the Worker Nodes and their labels:

```
kubectl get nodes --show-labels
```

Finally, we can list the running Pods to confirm that the Pod launched and is running on a compliant Worker Node:

```
kubectl get pods
```

15.5 Demonstrating Workload Confidentiality with Docker Containers

15.5.1 Encrypting an Image

Copy the sample “hello—world” image to the WPM system. Use the WPM to encrypt the sample image. Assuming the sample image filename is “hello-world.tgz” and that you are running from the /root directory:

```
```

After generating the encrypted image with the WPM, the encrypted image can be uploaded to a local Docker Registry.

15.5.2 Uploading the Image Flavor

```
POST https://<Workload Service IP or Hostname>:5000/wls/flavors
{<Image Flavor content from WPM output>}
```
Use the above API request to upload the Image Flavor to the WLS. The Image Flavor will tell other Intel® SecL-DC components the Key Transfer URL for this image.

### 15.5.3 Creating the Image Flavor to Image ID Association

For Docker images stored in a Docker Registry, the ID is typically an MD5 hash. This format must be converted for use with the Workload Service. To get the non-truncated ID of the image, use the Docker command:

```bash
docker images --no-trunc
```

Next, convert this to a UUID that can be used by Intel® SecL:

```bash
wpm get-container-image-id <image-full-md5id>
```

The output will be a UUID, which will be considered the ID of the image for the WLS.

Use the below API request to create an association between the Image Flavor created in the previous step and the image ID.

```json
POST https://<Workload Service IP or Hostname>:5000/wls/images
{
"id": "<image ID on image storage>",
"flavor_ids": ["<Image Flavor ID>"
}
```

### 15.5.4 Demonstrate Launching Encrypted Docker Containers on Trusted Hosts

Containers of the protected images can now be launched as normal by launching a pod.

Encrypted images will only be accessible on hosts with a Platform Integrity Attestation report showing the host is trusted.

### 15.5.5 Demonstrate Launching Encrypted VMs on Untrusted Hosts

Now that we have demonstrated a successful container launch on Trusted hosts, we need to show that Untrusted hosts cannot launch protected workloads.

First, delete all running instances of the protected pod (do not delete the image) using Kubernetes APIs.
Next, we will delete Flavors from the Verification Service. Without required Flavors, hosts will attest as Untrusted with the fault “Flavor part missing but required.”

6. Retrieve a list of all PLATFORM Flavors:

GET https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors?key=flavor_part&value=PLATFORM

The response will list one or more PLATFORM Flavors. Each will begin with a Meta section like this:

"signed_flavors" : [
    "flavor" : {
        "meta" : {
            "id" : "583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3",
            "description" : {
                "flavor_part" : "PLATFORM",
                "source" : "Q23RU28",
                "label" : …
                "bios_name" : "Intel Corporation",
                "bios_version" : "SE5C620.86B.0X.01.0155.073020181001",
                "tpm_version" : "2.0",
                "tboot_installed" : "true"
            ...
        }
    }
]

Note the ID in bold.

7. For each Flavor ID returned, delete that Flavor:

DELETE https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors/<flavor ID>

For example:

DELETE https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors/583aa7e9-9cf2-46dd-a3ed-d63e8b8da7d3

The response for each DELETE should be a 204 No Content.

8. List the Flavors again:

GET https://<Verification Service IP or hostname>:8443/mtwilson/v2/flavors?key=flavor_part&value=PLATFORM

You should see an empty array indicating no PLATFORM Flavors remain.

9. Generate new reports

For each host, we now need to create a new attestation report. These will automatically generate, but we can speed the process by forcing new reports immediately:

POST https://<Verification Service IP or hostname>:8443/mtwilson/v2/reports

Headers:
Accept: application/json
input: {"host_name":"<Trust Agent host>"}

Repeat this call for each Trust Agent host. The reports returned should all show the Overall trust status is now False.

10. Try to launch the protected image

Using the Kubernetes API, launch a new instance of the protected pod.

In this case, the launch should fail.

Behind the scenes, the Workload Agent on the compute host attempted to retrieve a key for the image. The key transfer request was sent to the Key Broker using a report for that host from the Verification Service. Because the host is Untrusted, the key transfer request was denied, and the image remained encrypted and inaccessible to the container runtime, causing a failure.