## Contents

1 **Introduction** ................................................................. 10
   1.1 Overview................................................................. 10
      1.1.1 Trusted Computing............................................. 10
      1.1.2 Intel® Security Libraries for Datacenter Features ......... 13

2 **Intel® Security Libraries Components** .............................. 16
   2.1 Certificate Management Service .................................... 16
   2.2 Authentication and Authorization Service ...................... 16
   2.3 Verification Service................................................... 16
   2.4 Workload Service ...................................................... 16
   2.5 Trust Agent .............................................................. 17
   2.6 Workload Agent ......................................................... 17
   2.7 Integration Hub ........................................................ 17
   2.8 Workload Policy Manager ............................................ 18
   2.9 Key Broker Service ................................................... 18

3 **Intel® Security Libraries Installation** ................................. 19
   3.1 Building from Source ................................................ 19
   3.2 Hardware Considerations............................................ 19
   3.3 Recommended Service Layout ...................................... 20
      3.3.1 Platform Integrity ............................................. 21
      3.3.2 Workload Confidentiality .................................... 21
   3.4 Installing/Configuring the Database ............................... 22
      3.4.1 Using the Provided Database Installation Script .......... 22
      3.4.2 Provisioning the Database .................................... 23
      3.4.3 Database Server TLS Certificate ............................ 24
   3.5 Installing the Certificate Management Service ................ 24
      3.5.1 Required For..................................................... 24
      3.5.2 Supported Operating Systems ............................... 24
      3.5.3 Recommended Hardware ..................................... 24
      3.5.4 Installation ....................................................... 25
   3.6 Installing the Authentication and Authorization Service ....... 26
      3.6.1 Required For..................................................... 26
      3.6.2 Prerequisites ..................................................... 26
      3.6.3 Package Dependencies ....................................... 26
      3.6.4 Supported Operating Systems ............................... 26
      3.6.5 Recommended Hardware ..................................... 26
      3.6.6 Installation ....................................................... 27
      3.6.7 Creating Users ................................................ 27
   3.7 Installing the Verification Service ................................... 30
      3.7.1 Required For..................................................... 30
      3.7.2 Prerequisites ..................................................... 30
      3.7.3 Package Dependencies ....................................... 30
      3.7.4 Supported Operating Systems ............................... 31
      3.7.5 Recommended Hardware ..................................... 31
      3.7.6 Installation ....................................................... 31
3.8 Installing the Workload Service ................................................... 32
  3.8.1 Required For .................................................................. 32
  3.8.2 Prerequisites ................................................................ 32
  3.8.3 Supported Operating Systems ...................................... 33
  3.8.4 Recommended Hardware ............................................. 33
  3.8.5 Installation .................................................................. 33
3.9 Installing the Trust Agent for Linux ............................................. 33
  3.9.1 Required For .................................................................. 33
  3.9.2 Package Dependencies .................................................. 34
  3.9.3 Supported Operating Systems ...................................... 34
  3.9.4 Prerequisites ................................................................ 34
  3.9.5 Installation .................................................................. 38
3.10 Installing the Workload Agent ..................................................... 39
  3.10.1 Required For .............................................................. 39
  3.10.2 Supported Operating Systems ...................................... 40
  3.10.3 Prerequisites ................................................................ 40
  3.10.4 Installation .................................................................. 40
3.11 Installing the Trust Agent for Windows ......................................... 40
  3.11.1 Required For .............................................................. 41
  3.11.2 Supported Operating Systems ...................................... 41
  3.11.3 Prerequisites ................................................................ 41
  3.11.4 Installation .................................................................. 42
3.12 Trust Agent Provisioning ............................................................ 43
3.13 Trust Agent Registration ............................................................ 44
3.14 Importing the HOST_UNIQUE Flavor ........................................... 45
3.15 Installing the Integration Hub ..................................................... 46
  3.15.1 Required For .............................................................. 46
  3.15.2 Prerequisites ................................................................ 46
  3.15.3 Package Dependencies .................................................. 47
  3.15.4 Supported Operating Systems ...................................... 47
  3.15.5 Recommended Hardware ............................................. 47
  3.15.6 Installation .................................................................. 47
3.16 Installing the Key Broker Service ................................................ 49
  3.16.1 Required For .............................................................. 49
  3.16.2 Prerequisites ................................................................ 49
  3.16.3 Package Dependencies .................................................. 50
  3.16.4 Supported Operating Systems ...................................... 50
  3.16.5 Recommended Hardware ............................................. 50
  3.16.6 Installation .................................................................. 50
  3.16.7 Importing Verification Service Certificates .................... 51
3.17 Installing the Workload Policy Manager ...................................... 53
  3.17.1 Required For .............................................................. 53
  3.17.2 Package Dependencies .................................................. 53
  3.17.3 Supported Operating Systems ...................................... 53
  3.17.4 Recommended Hardware ............................................. 53
  3.17.5 Installation .................................................................. 53
4 Authentication .................................................................................. 55
  4.1 Create Token ........................................................................ 55
4.2 User Management ................................................................. 55
   4.2.1 Username and Password requirements .................... 56
   4.2.2 Create User ............................................................. 56
   4.2.3 Search User ............................................................ 56
   4.2.4 Change User Password ............................................. 56
   4.2.5 Delete User ............................................................. 57
4.3 Roles and Permissions ....................................................... 57
   4.3.1 Create Role ............................................................ 57
   4.3.2 Search Roles .......................................................... 58
   4.3.3 Delete Role ............................................................ 58
   4.3.4 Assign Role to User .................................................. 58
   4.3.5 List Roles Assigned to User ...................................... 58
   4.3.6 Remove Role from User ............................................ 58
   4.3.7 Role Definitions ..................................................... 59
5 Connection Strings ................................................................... 62
   5.1 Trust Agent (Windows and Linux) ................................. 62
   5.2 VMware ESXi .............................................................. 62
6 Platform Integrity Attestation .................................................. 63
   6.1 Host Registration .......................................................... 64
   6.1.1 Trust Agent ............................................................ 64
   6.1.2 Registration via Verification Service API .................. 64
   6.2 Flavor Creation for Automatic Flavor Matching ............. 65
   6.2.1 Importing a Flavor from a Sample Host ..................... 66
   6.2.2 Creating a Flavor Manually ...................................... 66
   6.3 Creating the Default SOFTWARE Flavor (Linux Only) .... 67
   6.4 Creating and Provisioning Asset Tags ......................... 68
   6.4.1 Creating Asset Tag Certificates ............................... 68
   6.4.2 Deploying Asset Tags ............................................. 69
   6.5 Retrieving Current Attestation Reports ....................... 72
   6.6 Retrieving Current Host State Information ................... 72
   6.7 Upgrading Hosts in the Datacenter to a New BIOS or OS Version 0000000000 72
   6.8 Removing Hosts From the Verification Service .............. 73
   6.9 Removing Flavors ....................................................... 73
   6.10 Invalidating Asset Tags .............................................. 73
   6.11 Remediating an Untrusted attestation ......................... 73
   6.12 Attestation Reporting ............................................... 74
   6.12.1 Sample Call – Generating a New Attestation Report .... 75
   6.12.2 Sample Call – Retrieving an Existing Attestation Report .... 75
   6.13 Integration ............................................................... 76
   6.13.1 Integration Hub .................................................. 76
   6.13.2 Integration with OpenStack ..................................... 78
   6.13.3 Integration with Kubernetes ................................. 83
7 Workload Confidentiality ........................................................ 89
   7.1 Virtual Machine Confidentiality .................................... 91
   7.1.1 Prerequisites ........................................................ 91
7.1.2 Workflow ................................................................. 92
7.2 Docker Container Confidentiality ............................... 93
   7.2.1 Docker Container Integrity ................................ 93
   7.2.2 Prerequisites ...................................................... 93
   7.2.3 Workflow .......................................................... 95

8 Trusted Virtual Kubernetes Worker Nodes .......................... 97
   8.1 Prerequisites .......................................................... 98
   8.2 Workflow ............................................................. 99
   8.3 Sample VM Trust Report ......................................... 99

9 Flavor Management .......................................................... 102
   9.1 Flavor Format Definitions ........................................ 102
      9.1.1 Meta ............................................................. 102
      9.1.2 Hardware ...................................................... 102
      9.1.3 PCRs ............................................................. 103
      9.1.4 Sample PLATFORM Flavor ............................... 104
      9.1.5 Sample OS Flavor .......................................... 107
      9.1.6 Sample HOST_UNIQUE Flavor .......................... 113
      9.1.7 Sample ASSET_TAG Flavor .............................. 115
   9.2 Flavor Matching ..................................................... 116
      9.2.1 When Does Flavor Matching Happen? .................. 116
      9.2.2 Flavor Matching Performance ............................ 117
      9.2.3 Flavor Groups ................................................ 118
      9.2.4 Default Flavor Group ...................................... 118
      9.2.5 Flavor Match Policies ...................................... 118
      9.2.6 Flavor Match Event Triggers .............................. 120
      9.2.7 Sample Flavorgroup API Calls ........................... 121
   9.3 SOFTWARE Flavor Management ................................ 121
      9.3.1 What is a SOFTWARE Flavor? ............................ 121
      9.3.2 Creating a SOFTWARE Flavor part ..................... 123
      9.3.3 Sample SOFTWARE Flavor Creation Call .............. 123
      9.3.4 Deploying a SOFTWARE Flavor Manifest to a Host 124
      9.3.5 SOFTWARE Flavor Matching ............................. 125
      9.3.6 Kernel Upgrades ............................................. 125

10 Scalability and Sizing ..................................................... 127
   10.1 Configuration Maximums ....................................... 127
      10.1.1 Registered Hosts ........................................... 127
      10.1.2 HDD Space .................................................. 127
   10.2 Database Rotation Settings ..................................... 127
   10.3 Log Rotation ....................................................... 127

11 Intel Security Libraries Configuration Settings .................... 128
   11.1 Verification Service .............................................. 128
      11.1.1 Installation Answer File Options ....................... 128
      11.1.2 Configuration Options .................................... 131
      11.1.3 Command-Line Options .................................. 134
      11.1.4 Directory Layout .......................................... 136
11.2 Trust Agent ................................................................. 138
11.2.1 Installation Answer File Options .......................... 138
11.2.2 Configuration Options .............................................. 140
11.2.3 Command-Line Options .......................................... 142
11.2.4 Directory Layout ...................................................... 145
11.3 Integration Hub ............................................................ 146
11.3.1 Installation Answer File ............................................ 146
11.3.2 Configuration Options .............................................. 147
11.3.3 Command-Line Options .......................................... 149
11.3.4 Directory Layout ...................................................... 151
11.4 Certificate Management Service ................................. 151
11.4.1 Installation Answer File Options ......................... 151
11.4.2 Configuration Options .............................................. 152
11.4.3 Command-Line Options .......................................... 153
11.4.4 Directory Layout ...................................................... 155
11.5 Authentication and Authorization Service .................. 155
11.5.1 Installation Answer File Options ......................... 155
11.5.2 Configuration Options .............................................. 156
11.5.3 Command-Line Options .......................................... 156
11.5.4 Directory Layout ...................................................... 159
11.6 Workload Service .......................................................... 159
11.6.1 Installation Answer File Options ......................... 159
11.6.2 Configuration Options .............................................. 161
11.6.3 Command-Line Options .......................................... 161
11.6.4 Directory Layout ...................................................... 165
11.7 Key Broker Service ...................................................... 165
11.7.1 Installation Answer File Options ......................... 165
11.7.2 Configuration Options .............................................. 167
11.7.3 Command-Line Options .......................................... 167
11.7.4 Directory Layout ...................................................... 169
11.8 Workload Agent ............................................................. 170
11.8.1 Installation Answer File Options ......................... 170
11.8.2 Configuration Options .............................................. 170
11.8.3 Command-Line Options .......................................... 170
11.8.4 Directory Layout ...................................................... 171
11.9 Workload Policy Manager ............................................ 172
11.9.1 Installation Answer File Options ......................... 172
11.9.2 Configuration Options .............................................. 172
11.9.3 Command-Line Options .......................................... 172
12 Certificate and Key Management ........................................... 175
12.1 Host Verification Service Certificates and Keys ............. 175
12.1.1 SAML ................................................................. 175
12.1.2 Asset Tag .............................................................. 176
12.1.3 Privacy CA ........................................................... 176
12.1.4 Endorsement CA .................................................... 177
12.2 TLS Certificates ............................................................ 178
13 TLS Policies ........................................................................ 179
## Revision History

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Revision Number</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Updated for all GA Failures</td>
<td>May 2019</td>
</tr>
<tr>
<td>1.5</td>
<td></td>
<td>Updated for version 1.5 release</td>
<td>July 2019</td>
</tr>
<tr>
<td>1.6 BETA</td>
<td></td>
<td>Updated for 1.6 BETA release</td>
<td>November 2019</td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td>Updated for version 1.6 release</td>
<td>December 2019</td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td>Updated for version 2.0 release</td>
<td>February 2020</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
<td>Updated for version 2.1 release</td>
<td>April 2020</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Overview

Intel Security Libraries for Datacenter is a collection of software applications and development libraries intended to help turn Intel platform security features into real-world security use cases.

1.1.1 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.

1.1.1.1 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.

1.1.2 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.
1.1.1.2.1 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.

1.1.1.2.2 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).

1.1.1.3 Supported Trusted Boot Options

Intel® SecL-DC supports several options for Trusted Computing, depending on the features available on the platform.
Note: A security bug related to UEFI Secure Boot and Grub2 modules has resulted in some modules required by tboot to not be available on RedHat 8 UEFI systems. Tboot therefore cannot be used currently on RedHat 8. A future tboot release is expected to resolve this dependency issue and restore support for UEFI mode.

1.1.1.4 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 information on TPM Launch Control Policies, consult the Intel Trusted Execution Technology (Intel TXT) Software Development Guide (https://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf).

1.1.2 Intel® Security Libraries for Datacenter Features

1.1.2.1 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.

1.1.2.2 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.

1.1.2.3 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.

1.1.2.4 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 hosts 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.

Beginning with the Intel® SecL-DC version 2.1 release, the Key Broker now supports 3rd-party key managers that are KMIP-compliant. The Key Broker has been updated to use the “libkmip” client.

### 1.1.2.5 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.

1.1.2.6 Trusted Virtual Kubernetes Worker Nodes

Added in the Intel® SecL-DC version 2.1 release, this feature provides a Chain of Trust solution extending to Kubernetes Worker Nodes deployed as Virtual Machines. This feature addresses Kubernetes deployments that use Virtual Machines as Worker Nodes, rather than using bare-metal servers.

When libvirt initiates a VM Start, the Intel® SecL-DC Workload Agent will create a report for the VM that associates the VM’s trust status with the trust status of the host launching the VM. This VM report will be retrievable via the Workload Service, and contains the hardware UUID of the physical server hosting the VM. This UUID can be correlated to the Trust Report of that server at the time of VM launch, creating an audit trail validating that the VM launched on a trusted platform. A new report is created for every VM Start, which includes actions like VM migrations, so that each time a VM is launched or moved a new report is generated ensuring an accurate trust status.

By using Platform Integrity and Data Sovereignty-based orchestration (or Workload Confidentiality with encrypted worker VMs) for the Virtual Machines to ensure that the virtual Kubernetes Worker nodes only launch on trusted hardware, these VM trust reports provide an auditing capability to extend the Chain of Trust to the virtual Worker Nodes.
2 Intel® Security Libraries Components

2.1 Certificate Management Service

Starting with Intel® SecL-DC 1.6, most non-TPM-related certificates used by Intel® SecL-DC applications will be issued by the new Certificate Management Service. This includes acting as a root CA and issuing TLS certificates for all of the various web services.

2.2 Authentication and Authorization Service

Starting with Intel® SecL-DC 1.6, authentication and authorization for all Intel® SecL applications will be centrally managed by the new Authentication and Authorization Service (AAS). Previously, each application would manage its own users and permissions independently; this change allows authentication and authorization management to be centralized.

2.3 Verification Service

The Verification Service component of Intel® Security Libraries performs the core Platform Integrity and Data Sovereignty functionality by acting as a remote attestation authority.

Platform security technologies like Intel® TXT, Intel® BootGuard, and UEFI SecureBoot extend measurements of platform components (such as the system BIOS/UEFI, OS kernel, etc) to a Trusted Platform module as the server boots. Known-good measurements for each of these components can be directly imported from a sample server. These expected measurements can then be compared against actual measurements from registered servers, allowing the Verification Service to attest to the “trustiness” of the platform, meaning whether the platform booted into a “known-good” state.

2.4 Workload Service

The Workload Service acts as a management service for handling Workload Flavors (Flavors used for Virtual Machines and Containers). In the Intel® SecL-DC 1.6 release, the Workload Service uses Flavors to map decryption key IDs to image IDs. When a launch request for an encrypted workload image is intercepted by the Workload Agent, the Workload Service will handle mapping the image ID to the appropriate key ID and key request URL, and will initiate the key transfer request to the Key Broker.
2.5 **Trust Agent**

The Trust Agent resides on physical servers and enables both remote attestation and the extended chain of trust capabilities. The Agent maintains ownership of the server's Trusted Platform Module, allowing secure attestation quotes to be sent to the Verification Service. Incorporating the Intel® SecL HostInfo and TpmProvider libraries, the Trust Agent serves to report on platform security capabilities and platform integrity measurements.

The Trust Agent is supported for Windows* Server 2016 Datacenter and Red Hat Enterprise Linux* (RHEL) 7.4 and later.

2.6 **Workload Agent**

The Workload Agent is the component responsible for handling all of the functions needed for Workload Confidentiality for virtual machines and Docker containers on a physical server. The Workload Agent uses libvirt hooks to identify VM lifecycle events (VM start, stop, hibernate, etc), and intercepts those events to perform needed functions like requesting decryption keys, creation and deletion of encrypted LUKS volumes, using the TPM to unseal decryption keys, etc. The WLA also includes the Docker SecureOverlay Driver that performs analogous functionality for Docker containers.

2.7 **Integration Hub**

The Integration Hub acts as a middle-man between the Verification Service and one or more scheduler services (such as OpenStack* Nova), and "pushes" attestation information retrieved from the Verification Service to one or more scheduler services according to an assignment of hosts to specific tenants. In this way, Tenant A can receive attestation information for hosts that belong to Tenant A, but receive no information about hosts belonging to Tenant B.

The Integration Hub serves to disassociate the process of retrieving attestations from actual scheduler queries, so that scheduler services can adhere to best practices and retain better performance at scale. The Integration Hub will regularly query the Intel® SecL Verification Service for SAML attestations for each host. The Integration Hub maintains only the most recent currently valid attestation for each host, and will refresh attestations when they would expire. The Integration Hub will verify the signature of the SAML attestation for each host assigned to a tenant, then parse the attestation status and asset tag information, and then will securely push the parsed key/value pairs to the plugin endpoints enabled.

The Integration Hub features a plugin design for adding new scheduler endpoint types. Currently the Integration Hub supports OpenStack Nova and Kubernetes endpoint plugins. Other integration plugins may be added.
2.8 **Workload Policy Manager**

The Workload Policy Manager is a Linux command line utility used by an image owner to encrypt VM (qcow2) or container (Docker) images, and to create an Image Flavor used to provide the encryption key transfer URL during launch requests. The WPM utility will use an existing or request a new key from the Key Broker Service, use that key to encrypt the image, and output the Image Flavor in JSON format. The encrypted image can then be uploaded to the image store of choice (like OpenStack Glance), and the Image Flavor can be uploaded to the Workload Service. The ID of the image on the image storage system is then mapped to the Image Flavor in the WLS; when the image is used to launch a new instance, the WLS will find the Image Flavor associated with that image ID, and use the Image Flavor to determine the key transfer URL.

2.9 **Key Broker Service**

The Key Broker Service is effectively a policy compliance engine. Its job is to manage key transfer requests for encrypted images, releasing keys only to servers that meet policy requirements. The Key Broker registers one or more SAML signing certificates from any Verification Services that it will trust. When a key transfer request is received, the request includes a trust attestation report signed by the Verification Service. If the signature matches a registered SAML key, the Broker will then look at the actual report to ensure the server requesting the key matches the image policy (currently only overall system trust is supported as a policy requirement). If the report indicates the policy requirements are met, the image decryption key is wrapped using a public key unique to the TPM of the host that was attested in the report, such that only the host that was attested can unseal the decryption key and gain access to the image.
3 Intel® Security Libraries Installation

3.1 Building from Source

Intel® Security Libraries is distributed as open source code, and must be compiled into installation binaries before installation.

Instructions and sample scripts for building the Intel® SecL-DC components can be found here:

https://01.org/intel-secl/documentation/build-installation-scripts

After the components have been built, the installation binaries can be found in the directories created by the build scripts. Intel® SecL-DC is migrating from being based on Java to being written in GO, so there is a slightly different build path based on the language the component uses:


<servicename>/out/<servicename>.bin

For components written in Java (Verification Service, Integration Hub, Key Broker):

<servicename>/packages/<servicename>/target/<servicename>-%<version>-%SNAPSHOT.bin

In addition, the build script will produce some sample database creation scripts that can be used during installation to configure database requirements (instructions are given in the installation sections):

Create_db: authservice/out/create_db.sh

Install_pgdb: authservice/out/install_pgdb.sh

3.2 Hardware Considerations

Intel® SecL-DC supports and uses a variety of Intel security features, but there are some key requirements to consider before beginning an installation. Most important among these is the Root of Trust configuration. This involves
deciding what combination of TXT, Boot Guard, tboot, and UEFI Secure Boot to enable on platforms that will be attested using Intel® SecL.

Key points:
- At least one “Static Root of Trust” mechanism must be used (TXT and/or BtG)
- For Legacy BIOS systems, tboot must be used
- For UEFI mode systems, UEFI SecureBoot must be used*

Use the chart below for a guide to acceptable configuration options.

![Trusted Boot Options](chart.png)

*Note: A security bug related to UEFI Secure Boot and Grub2 modules has resulted in some modules required by tboot to not be available on RedHat 8 UEFI systems. Tboot therefore cannot be used currently on RedHat 8. A future tboot release is expected to resolve this dependency issue and restore support for UEFI mode.

### 3.3 Recommended Service Layout

The Intel® SecL-DC services can be installed in a variety of layouts, partially depending on the use cases desired and the OS of the server(s) to be protected. In general, the Intel® SecL-DC applications can be divided into management services that are deployed on the network on the management plane, and host or node components that must be installed on each protected server.
Management services can typically be deployed anywhere with network access to all of the protected servers. This could be a set of individual VMs per service; containers; or all installed on a single physical or virtual machine.

Node components must be installed on each protected physical server. Typically this is needed for Windows and Linux deployments.

### 3.3.1 Platform Integrity

The most basic use case enabled by Intel® SecL-DC, Platform Integrity requires only the Verification Service and, to protect Windows or Linux hosts, the Trust Agent. This also enables the Application Integrity use case by default for Linux systems.

The Integration Hub may be added to provide integration support for OpenStack or Kubernetes. The Hub is often installed on the same machine as the Verification Service, but optionally can be installed separately.

### 3.3.2 Workload Confidentiality

Workload Confidentiality introduces a number of additional services and agents. For a POC environment, all of the management services can be installed on a single machine or VM. This includes:

- Certificate Management Service (CMS)
- Authorization and Authentication Service (AAS)
- Host Verification Service (HVS)
- Workload Service (WLS)
- Integration Hub (HUB)
- Key Broker Service (KBS) with backend key management
- Workload Policy Manager (WPM)

In a production environment, it is strongly suggested that the WPM and KBS be deployed (with their own CMS and AAS) separately for each image owner. For a Cloud Service Provider, this would mean that each customer/tenant who will use the Workload Confidentiality feature would have their own dedicated AAS/CMS/KBS/WPM operated on their own networks, not controlled by the CSP. This is because the Key Broker and WPM are the tools used to define the policies that will allow images to launch, and these policies and their enforcement should remain entirely under the control of the image owner.

The node components must be installed on each protected physical server:

- Trust Agent (TA)
- Workload Agent (WLA)
3.4 Installing/Configuring the Database

The Intel® SecL-DC Authentication and Authorization Service (AAS) requires a Postgresql 11 database. Scripts (install_pgdb.sh, create_db.sh) are provided with the AAS that will automatically add the Postgresql repositories and install/configure a sample database. If this script will not be used, a Postgresql 11 database must be installed by the user before executing the AAS installation.

3.4.1 Using the Provided Database Installation Script

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-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Create the iseclpgdb.env answer file:
ISECL_PKGDB_IP_INTERFACES=localhost
ISECL_PKGDB_PORT=5432
ISECL_PKGDB_SAVE_DB_INSTALL_LOG=true
ISECL_PGDB_CERT_DNS=localhost
ISECL_PGDB_CERT_IP=127.0.0.1

Note that the values above assume that the database will be accessed locally. If the database server will be external to the Intel® SecL services, change these values to the hostname or FQDN and IP address where the client will access the database server.

Run the following command:

dnf module disable postgresql -y

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 needs to be repeated if the database server will be shared.

### 3.4.2 Provisioning the Database

Each Intel® SecL service that uses a database (the Authentication and Authorization Service, the Verification Service, the Integration Hub, the Workload Service) requires its own schema and access. After installation, the database must be created initialized and tables created. Execute the create_db.sh script to configure the database.

If a single shared database server will be used for each Intel® SecL service (for example, if all management plane services will be installed on a single VM), run the script multiple times, once for each service that requires a database.

If separate database servers will be used (for example, if the management plane services will reside on separate systems and will use their own local database servers), execute the script on each server hosting a database.

```
./create_db.sh <database name> <database_username> <database_password>
```

For example:

```
./create_db.sh isecl_hvs_db hvs_db_username hvs_db_password
./create_db.sh isecl_aas_db aas_db_username aas_db_password
./create_db.sh isecl_wls_db wls_db_username wls_db_password
./create_db.sh isecl_hub_db hub_db_username hub_db_password
```

Note that the database name, username, and password details for each service must be used in the corresponding installation answer file for that service.
3.4.3 **Database Server TLS Certificate**

The database client for Intel® SecL services requires the database TLS certificate to authenticate communication with the database server.

If the database server for a service is located on the same server that the service will run on, only the path to this certificate is needed. If the provided Postgres scripts are used, the certificate will be located in /usr/local/pgsql/data/server.crt.

If the database server will be run separately from the Intel® SecL service(s), the certificate will need to be copied from the database server to the service machine before installing the Intel® SecL services.

The database client for Intel® SecL services will validate that the Subject Alternative Names in the database server’s TLS certificate contain the hostname(s)/IP address(es) that the clients will use to access the database server. If configuring a database without using the provided scripts, ensure that these attributes are present in the database TLS certificate.

3.5 **Installing the Certificate Management Service**

3.5.1 **Required For**

The CMS is REQUIRED for all use cases.

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

3.5.2 **Supported Operating Systems**


3.5.3 **Recommended Hardware**

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

3.5.4 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
   SAN_LIST=<Comma-separated list of IP addresses and hostnames for the CMS>,127.0.0.1,localhost
   ```

   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-2.1.0.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:
3.6 Installing the Authentication and Authorization Service

3.6.1 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)

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

- The Certificate Management Service must be installed and available
- The Authentication and Authorization Service database must be available

3.6.3 Package Dependencies
The Intel® SecL-DC Authentication and Authorization Service (AAS) requires a Postgresql 11 database. A script (install_pgdb.sh) is provided with the AAS that will automatically add the Postgresql repositories and install/configure a sample database. If this script will not be used, a Postgresql 11 database must be installed by the user before executing the AAS installation.

3.6.4 Supported Operating Systems

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

### 3.6.6 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:

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

Create the authservice.env installation answer file:

```bash
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=<database port number; default is 5432>
AAS_DB_NAME=<database name>
AAS_DB_USERNAME=<database username>
AAS_DB_PASSWORD=<database password>
AAS_DB_SSLCERTSRC=<path to database TLS certificate; the default location is typically /usr/local/pgsql/data/server.crt >
AAS_ADMIN_USERNAME=<username for AAS administrative user>
AAS_ADMIN_PASSWORD=<password for AAS administrative user>
SAN_LIST=<comma-separated list of IPs and hostnames for the AAS; this should match the value for the AAS_TLS_SAN in the cms.env file from the CMS installation>
BEARER_TOKEN=<bearer token from CMS installation>
```

Execute the AAS installer:

```bash
./authservice-v2.1.0.bin
```

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.

### 3.6.7 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 automate the process.
Create the populate-users.env file:

ISECL_INSTALL_COMPONENTS=KBS,TA,WLS,WPM,AH,VS,WLA,AAS

AAS_API_URL=https://<AAS IP address or hostname>:8444/aas
AAS_ADMIN_USERNAME=<AAS username>
AAS_ADMIN_PASSWORD=<AAS password>

VS_CERT_SAN_LIST=<comma-separated list of IPs and hostnames for the Verification Service>
AH_CERT_SAN_LIST=<comma-separated list of IPs and hostnames for the Integration Hub>
WLS_CERT_SAN_LIST=<comma-separated list of IPs and hostnames for the Workload Service>
KBS_CERT_SAN_LIST=<comma-separated list of IPs and hostnames for the Key Broker Service>
TA_CERT_SAN_LIST=<comma-separated list of IPs and hostnames for the Trust Agent>

VS_SERVICE_USERNAME=<Username for the VS service user>
VS_SERVICE_PASSWORD=<Password for the VS service user>

AH_SERVICE_USERNAME=<Username for the Hub service user>
AH_SERVICE_PASSWORD=<Password for the Hub service user>

WPM_SERVICE_USERNAME=<Username for the WPM service user>
WPM_SERVICE_PASSWORD=<Password for the WPM service user>

WLS_SERVICE_USERNAME=<Username for the WLS service user>
WLS_SERVICE_PASSWORD=<Password for the WLS service user>

WLA_SERVICE_USERNAME=<Username for the WLA service user>
WLA_SERVICE_PASSWORD=<Password for the WLA service user>

GLOBAL_ADMIN_USERNAME=<Username for the global Administrator user>
GLOBAL_ADMIN_PASSWORD=<Password for the global Administrator user>

INSTALL_ADMIN_USERNAME=<Username for the installation user>
INSTALL_ADMIN_PASSWORD=<Password for the global installation user>

**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 SAN list specified here must exactly match the SAN list for the applicable service in that service’s env installation file.
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
Workload Policy Manager Service User
Workload Service User Name
Workload Service User
Global Admin User
Installation 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-users.env file.

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

The populate-users script will also output an installation token. This token has all privileges needed for installation of the Intel® SecL services, and uses the credentials provided with the INSTALLATION_ADMIN_USERNAME and password. 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, or the INSTALLATION_ADMIN_USERNAME and password can be used to generate an authentication token.
3.7 Installing the Verification Service

This section details how to install the Intel® SecL-DC services. For instructions on running these services as containers, see the following section.

3.7.1 Required For

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

3.7.2 Prerequisites

The following must be completed before installing the Verification Service:

- The Certificate Management Service must be installed and available
- The Authentication and Authorization Service must be installed and available
- The Verification Service database must be available

3.7.3 Package Dependencies

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

- Logback (optional)
- Java* 8 JDK
- OpenSSL
- Postgres* client and server 11.6 (server component optional if an external Postgres database is used)
- Unzip
- Zip
- Openssl
- Wget
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.

### 3.7.4 Supported Operating Systems


### 3.7.5 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).

### 3.7.6 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.
A sample minimal mtwilson.env file is provided below. For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Verification Service.

```
MTWILSON_SERVER=<IP address or hostname of the Verification Service>
MTWILSON_API_URL=https://$MTWILSON_SERVER:8443/mtwilson/v2
DATABASE_HOSTNAME=<hostname or IP address to database server>
DATABASE_USERNAME=<Database administrative username>
DATABASE_PORTNUM=5432
DATABASE_PASSWORD=<Database password>
DATABASE_SCHEMA=<Database schema>
VS_SERVICE_USERNAME=<Verification Service username>
VS_SERVICE_PASSWORD=<Verification Service password>
CMS_TLS_CERT_SHA384=<Certificate Management Service TLS digest>
BEARER_TOKEN=<Installation token from populate-users script>
AAS_API_URL=https://<Authentication and Authorization Service IP or Hostname>:8444/aas
CMS_BASE_URL=https://<Certificate Management Service IP or Hostname>:8445/cms/v1
VS_TLS_CERT_DNS=<comma separated list of hostnames for the Verification Service; must match hostnames in VS_CERT_SAN_LIST from populate-users script>
VS_TLS_CERT_IP=<comma separated list of IP addresses for the Verification Service; must match IPs in VS_CERT_SAN_LIST from populate-users script>
```

Execute the installer binary.

```
./host-verification-service-linux-5.1-SNAPSHOT.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
```

### 3.8 Installing the Workload Service

#### 3.8.1 Required For

The WLS is REQUIRED for the following use cases.

- Workload Confidentiality (both VMs and Docker Containers)

#### 3.8.2 Prerequisites

The following must be completed before installing the Workload Service:

- The Certificate Management Service must be installed and available
- The Authentication and Authorization Service must be installed and available
- The Verification Service must be installed and available
- The Workload Service database must be available
3.8.3 Supported Operating Systems


3.8.4 Recommended Hardware

3.8.5 Installation

1. Copy the Workload Service installation binary to the /root/ directory.
2. Create the workload-service.env installation answer file:

   WLS_DB_USERNAME=<database username>
   WLS_DB_PASSWORD=<database password>
   WLS_DB_HOSTNAME=<IP or hostname of database server>
   WLS_DB_PORT=<Database port; 5432 by default>
   WLS_DB=<name of the WLS database>
   WLS_DB_SSLCERTSRC=<path to database TLS certificate; the default location is typically /usr/local/pgsql/data/server.crt>
   HVS_URL=https://<IP address or hostname of the Host verification Service>:8443/mtwilson/v2/
   WLS_SERVICE_USERNAME=<username for WLS service account>
   WLS_SERVICE_PASSWORD=<password for WLS service account>
   CMS_BASE_URL=https://<IP or hostname to CMS>:8445/cms/v1/
   CMS_TLS_CERT_SHA384=<sha384 of CMS TLS certificate>
   AAS_API_URL=https://<IP or hostname to AAS>:8444/aas/
   SAN_LIST=<comma-separated list of IPs and hostnames for the WLS>

Execute the WLS installer binary:

./workload-service-2.1.0.bin

3.9 Installing the Trust Agent for Linux

3.9.1 Required For

The Trust Agent for Linux is REQUIRED for all use cases.

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

The Trust Agent requires the following packages and their dependencies:

- **Tboot** (Optional, for TXT-based deployments **without** UEFI SecureBoot only)
- Openssl
- tar
- redhat-lsb

If they are not already installed, the Trust Agent 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.

Tboot will not be installed automatically. Instructions for installing and configuring tboot are documented later in this section.

3.9.3 Supported Operating Systems

The Intel® Security Libraries Trust Agent for Linux supports Red Hat Enterprise Linux 8.1. Windows support is described in the section “Installing the Trust Agent for Windows.”

3.9.4 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with Intel Trusted Execution Technology activated in the system BIOS.
- Trusted Platform Module (version 2.0) installed and activated in the system BIOS, with cleared ownership status.

**Note:** For Linux systems, TPM 1.2 and TPM resource sharing to applications other than the Trust Agent are not supported at this time. Do not install trousers or another TSS stack application after installing the Trust Agent on Linux systems.

- System must be booted to a tboot boot option OR use UEFI SecureBoot.

**Note:** A security bug related to UEFI Secure Boot and Grub2 modules has resulted in some modules required by tboot to not be available on RedHat 8 UEFI systems. Tboot therefore cannot be used currently on RedHat 8. A future tboot release is expected to resolve this dependency issue and restore support for UEFI mode.
• (Provisioning step only) Intel® SecL Verification Service server installed and active.

• (REQUIRED for servers configured with TXT and tboot only) If the server is installed using an LVM, the LVM name must be identical for all Trust Agent systems. The Grub bootloader line that calls the Linux kernel will contain the LVM name of the root volume, and this line with all arguments is part of what is measured in the TXT/Tboot boot process. This will cause the OS Flavor measurements to differ between two otherwise identical hosts if their LVM names are different. Simply using a uniform name for the LVM during OS installation will resolve this possible discrepancy.

• (Optional, REQUIRED for Virtual Machine Confidentiality only):
  • QEMU/KVM must be installed
  • Libvirt must be installed

• (Optional, REQUIRED for Docker Container Confidentiality only): Docker CE 19.03.5 must be installed

  Note: The specific Docker-CE version 19.03.5 is required for Docker Container Confidentiality. Only this version is supported for this use case.

### 3.9.4.1 Tboot Installation

  Note: A solution to a security bug has resulted in some modules required by tboot to not be available on RedHat 8 UEFI systems. Tboot therefore cannot be used currently on RedHat 8. A future tboot release is expected to resolve this dependency issue and restore support for UEFI mode.

Tboot is required to build a complete Chain of Trust for Intel® TXT systems that are not using UEFI Secure Boot. Tboot acts to initiate the Intel® TXT SINIT ACM (Authenticated Code Module), which populates several TPM measurements including measurement of the kernel, grub command line, and initrd. Without either tboot or UEFI Secure Boot, the Chain of Trust will be broken because the OS-related components will be neither measured nor signature-verified prior to execution. Because tboot acts to initiate the Intel® TXT SINIT ACM, tboot is only required for platforms using Intel® TXT, and is not required for platforms using another hardware Root of Trust technology like Intel® Boot Guard.

Intel® SecL-DC requires tboot 1.9.7 or greater. For most platforms, the version of tboot available from the RedHat software repository will meet all requirements. Some newer platforms and platform firmware versions may require a later version of tboot, including later versions than are available on
the RedHat software repositories. This is due to updates that can be made to
the Intel® TXT SINIT ACM behavior, and the SINIT ACM is contained in the
BIOS firmware.

If a newer version of tboot is required than is available from the repository,
the most current version can be found here:

https://sourceforge.net/projects/tboot/files/tboot/

Tboot requires configuration of the grub boot loader after installation. To
install and configure tboot:

1) Install tboot

   # yum install tboot

2) Make a backup of your current grub.cfg file

   The below examples assume RedHat has been installed on a platform
   using Legacy boot mode. The grub path will be slightly different for
   platforms using Legacy BIOS.

   # cp /boot/grub2/grub.cfg /boot/grub2/grub.bak

3) Generate a new grub.cfg with the tboot boot option

   # grub2-mkconfig -o /boot/grub2/grub.cfg

4) Update the default boot option

   Ensure that the GRUB_DEFAULT value is set to the tboot option. The tboot
   boot option can be found my looking in the
   /boot/redhat/grub.cfg file. For
   example (the precise menu entry may be different, but should say
   “tboot”):

   menuentry 'Red Hat Enterprise Linux GNU/Linux, with tboot 1.9.7 and Linux
   4.18.0-147.el8.x86_64' --class red --class gnu-linux --class gnu --class os --
   class tboot {

   # vi /etc/default/grub

   GRUB_DEFAULT='Red Hat Enterprise Linux GNU/Linux, with tboot 1.9.7 and Linux
   4.18.0-147.el8.x86_64'

5) Reboot the system

   Because measurement happens at system boot, a reboot is needed to boot
to the tboot boot option and populate measurements in the TPM.

6) Verify a successful trusted boot with tboot

   Tboot provides the “txt-stat” command to show the tboot log. The first
part of the output of this command can be used to verify a successful
trusted launch. In the output below, note the “TXT measured launch” and
“secrets flag set” at the bottom. Both of these should show “TRUE” if the tboot measured launch was successful. If either of these show “FALSE,” the measured launch has failed. This usually simply indicates that the tboot boot option was not selected during boot.

If the measured launch was successful, proceed to install the Trust Agent.

```bash
# txt-stat
Intel(r) TXT Configuration Registers:
STS:   0x0001c091
       senter_done: TRUE
       sexit_done: FALSE
       mem_config_lock: FALSE
       private_open: TRUE
       locality_1_open: TRUE
       locality_2_open: TRUE
ESTS:  0x00
       txt_reset: FALSE
E2STS: 0x0000000000000006
       secrets: TRUE
ERRORCODE: 0x00000000
DIDVID: 0x0000000001b0078086
       vendor_id: 0x8086
       device_id: 0xb007
       revision_id: 0x1
FSBIF: 0xffffffffffffffff
QPIIF: 0x00000009d0003000
SINIT.BASE: 0x6fec0000
SINIT.SIZE: 262144B (0x40000)
HEAP.BASE: 0x6ff00000
HEAP.SIZE: 1048576B (0x100000)
DPR: 0x0000000070000051
       lock: TRUE
       top: 0x70000000
       size: 5MB (5242880B)
PUBLIC.KEY:
   9c 78 f0 d8 53 de 85 4a 2f 47 76 1c 72 b8 6a 11
   16 4a 66 a9 84 c1 aa d7 92 e3 14 4f b7 1c 2d 11

*******************************************************************
TXT measured launch: TRUE
secrets flag set: TRUE
*******************************************************************
```
3.9.5 Installation

Installation of the Trust Agent is split into two major steps: Installation, which covers the creation of system files and folders, and Provisioning, which involves the creation of keys and secrets and links the Trust Agent to a specific Verification Service. Both operations can be performed at the same time using an installation answer file. Without the answer file, the Trust Agent can be installed and left in an un-provisioned state regardless of whether a Verification Service is up and running, until such time as the datacenter administrator is ready to run the provisioning step and link the Trust Agent to a Verification Service.

To install the Trust Agent for Linux:

1. Copy the Trust Agent installation binary to the /root/ directory.

2. (Optional; required to perform Provisioning and Installation at the same time.) Create the trustagent.env answer file in the /root/ directory (for full configuration options, see section 9.2). The minimum configuration options for installation are provided below.

For Platform Attestation only, provide the following in trustagent.env:

```
MTWILSON_API_URL=https://<Verification Service IP or Hostname>:8443/mtwilson/v2
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 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
SAN_LIST=<Comma-separated list of IP addresses and hostnames for the TAgent matching the SAN list specified in the populate-users script; may include wildcards>
```

For Workload Confidentiality with VM Encryption, add the following (in addtion to the basic Platform Attestation sample):

```
WLA_SERVICE_USERNAME=<Username for the WLA service user>
WLA_SERVICE_PASSWORD=<Username for the WLA service user>
WLS_API_URL=https://<WLS IP address or hostname>:5000/wls/
```

For Workload Confidentiality with Docker Container Encryption, add the following (in addition to the basic Platform Attestation sample):

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

If the trustagent.env answer file was provided with the minimum required options, the Trust Agent will be installed and also Provisioned to the Verification Service specified in the answer file.

If no answer file was provided, the Trust Agent will be installed, but will not be Provisioned. TPM-related functionality will not be available from the Trust Agent until the Provisioning step is completed.

The Trust Agent will add a new grub menu entry for application measurement. This new entry will include tboot if the existing grub contains tboot as the default boot option.

**Note:** If the Linux Trust Agent is installed without being Provisioned, the Trust Agent process will not actually run until the Provisioning step has been completed.

4. (Legacy BIOS systems using tboot ONLY) Update the grub boot loader:

   `grub2-mkconfig -o /boot/grub2/grub.cfg`

5. After Provisioning is completed, the Linux Trust Agent must be rebooted so that the default SOFTWARE Flavor manifest can be measured and extended to the TPM. If the Workload Agent will also be installed on the system (see the next section), wait to reboot the server until after the Workload Agent has been installed, as this modifies the default SOFTWARE Flavor manifest.

### 3.10 Installing the Workload Agent

#### 3.10.1 Required For

- Workload Confidentiality (both VMs and Docker Containers)
3.10.2 **Supported Operating Systems**


3.10.3 **Prerequisites**

The following must be completed before installing the Workload Agent:

- Intel® SecL Trust Agent installed and active.
- cryptsetup
- *(REQUIRED for Virtual Machine Confidentiality only):*
  - QEMU/KVM must be installed
  - Libvirt must be installed
- *(REQUIRED for Docker Container Confidentiality only):* Docker CE 19.03.5 must be installed

**Note:** The specific Docker-CE version 19.03.5 is required for Docker Container Confidentiality. Only this version is supported for this use case.

3.10.4 **Installation**

1) Copy the Workload Agent installation binary to the `/root/` directory

2) Verify that the trustagent.env answer file is present. This file was necessary for installing/provisioning the Trust Agent. Note that the additional content required for Workload Confidentiality with either VM Encryption or Docker Container Encryption must be included in the trustagent.env file (samples provided in the previous section) for use by the Workload Agent.

3) Execute the Workload Agent installer binary.

   `workload-agent-v2.1.0.bin`

4) *(Legacy BIOS systems using tboot ONLY)* Update the grub boot loader:

   `grub2-mkconfig -o /boot/grub2/grub.cfg`

5) Reboot the server. The Workload Agent populates files that are needed for the default SOFTWARE Flavor, and a reboot is required for those measurements to happen.

3.11 **Installing the Trust Agent for Windows**
3.11.1 Required For

The Trust Agent for Windows is REQUIRED for the following use cases:
- Platform Integrity with Data Sovereignty and Signed Flavors

Other use cases are currently not supported for Windows.

3.11.2 Supported Operating Systems

The Trust Agent for Windows supports Windows Server 2016 Datacenter.

3.11.3 Prerequisites

The following must be completed before installing the Trust Agent:
- Supported server hardware including an Intel® Xeon processor.
- Trusted Platform Module (version 1.2 or 2.0) installed and activated in the system BIOS, with cleared ownership status.
- Coreinfo (https://docs.microsoft.com/en-us/sysinternals/downloads/coreinfo) must be installed
- (Provisioning step only) Intel® SecL Verification Service server installed and active.
- The Authentication and Authorization Service must be installed and available
- The Certificate Management Service must be installed and available

3.11.3.1 TPM Ownership

The Intel® SecL-DC Trust Agent for Windows requires the TPM ownership secret to be stored in the local system registry. To confirm that the secret is populated in the registry:

6) Open a Command Prompt as Administrator
7) Run the following command:
   
   REG QUERY hklm\system\controlset001\services\tpm\wmi\admin

8) If the output contains the OwnerAuthFull key and a corresponding value, the ownership secret is present in the registry and no further action is needed.

If the output does not contain the secret, system must be configured to store the secret in the registry.
To configure GPO to store the ownership secret in the local registry:

1) Open a Command Prompt as an Administrator
2) Run gpedit.msc
3) In the GP Editor, browse to Computer Configuration\Administrative Templates\System\Trusted Platform Module Services\
4) Set the Operating System Managed TPM Authentication Level to “Full”
5) Clear the TPM ownership and reboot

To clear TPM ownership from within Windows:

a) Open a Command Prompt as Administrator
b) Run tpm.msc
c) From the TPM Management Console that appears, click “Clear TPM”
d) After the process is complete, reboot

3.11.4 Installation

Installation of the Trust Agent is split into two major steps: Installation, which covers the creation of system files and folders, and Provisioning, which involves the creation of keys and secrets and links the Trust Agent to a specific Verification Service. Both operations can be performed at the same time using an installation answer file. Without the answer file, the Trust Agent can be installed and left in an un-provisioned state regardless of whether a Verification Service is up and running, until such time as the datacenter administrator is ready to run the provisioning step and link the Trust Agent to a Verification Service.

To install the Trust Agent for Windows:

1. (Optional; required to perform Provisioning and Installation at the same time.) Create the trustagent.ini answer file in the C:\Temp directory (for full configuration options, see section 9.2). The minimum configuration options for installation are provided below.
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.

If the trustagent.ini answer file was provided with the minimum required options, the Trust Agent will be installed and also Provisioned to the Verification Service specified in the answer file.

If no answer file was provided, the Trust Agent will be installed, but will not be Provisioned. TPM-related functionality will not be available from the Trust Agent until the Provisioning step is completed.

### 3.12 Trust Agent Provisioning

“Provisioning” the Trust Agent involves connecting to a Verification Service to download the Verification Service PrivacyCA certificate, create a new Attestation Identity Keypair in the TPM, and verify or create the TPM Endorsement Certificate and Endorsement Key. The Verification Service PrivacyCA root certificate is used to sign the EC, and the EC is used to generate the Attestation Identity Keypair. The AIK is used by the Verification Service to verify the integrity of quotes from the host’s TPM.

Provisioning can be performed separately from installation (meaning you can install the Trust Agent without Provisioning, and then Provision later). If the trustagent.env answer file is present and has the required Verification Service information during installation, the Agent will automatically run the Provisioning steps.

**Note:** The trustagent.env answer file must contain user credentials for a user with sufficient privileges. The minimum role required for performing provisioning is the “trustagent_provisioner” role.

**Note:** If the Linux Trust Agent is installed without being Provisioned, the Trust Agent process will not actually run until the Provisioning step has been completed.
If the answer file is not present during installation, the Agent can be provisioned later by adding the trustagent.env file and running the following command:

```
tagent provision-attestation <trustagent.env or trustagent.ini file path>
```

### 3.13 Trust Agent Registration

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

The Trust Agent can register the host with a Verification Service by running the following command (the trustagent.env or trustagent.ini answer file must be present in the current working directory):

```
tagent create-host
```

Hosts can also be registered using a REST API request to the Verification Service:

```
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": "intel:https://<hostname or IP address>:1443",
"flavorgroup_name": "",
"description": "<description>"
```

**Note:** When a new host is registered, the Verification Service will automatically attempt to match the host to appropriate Flavors. If appropriate Flavors are not found, the host will still be registered, but will be in an Untrusted state until/unless appropriate Flavors are added to the Verification Service.
### 3.14 Importing the HOST_UNIQUE Flavor

RHEL and VMWare ESXi hosts have measured components that are unique to each host. This means that a special HOST_UNIQUE flavor part needs to be imported for each RHEL and ESXi host, in addition to any other OS or Platform Flavors.

**Note:** Importing a Flavor requires user credentials for a user with sufficient privileges. The minimum role required for creating the HOST_UNIQUE Flavor part is the "host_unique_flavor_creator" role. This role can only create HOST_UNIQUE Flavor parts, and cannot create any other Flavors.

On Red Hat Enterprise Linux hosts with the Trust Agent, this can be performed from the Trust Agent command line (this requires the `trustagent.env` answer file to be present in the current working directory):

```
tagent create-host-unique-flavor
```
This can also be performed using a REST API (required for VMWare ESXi hosts):

```json
POST https://verification.service.com:8443/mtwilson/v2/flavors
{
    "connection_string": "<Connection string>",
    "partial_flavor_types": ["HOST_UNIQUE"],
    "tls_policy_id":"TRUST_FIRST_CERTIFICATE"
}
```

### 3.15 Installing the Integration Hub

**Note:** The Integration Hub is only required to integrate Intel® SecL with third-party scheduler services, such as OpenStack Nova or Kubernetes. The Hub is not required for usage models that do not require Intel® SecL security attributes to be pushed to an integration endpoint.

#### 3.15.1 Required For

The Hub is REQUIRED for the following use cases.

- Workload Confidentiality (both VMs and Docker Containers)

The Hub is OPTIONAL for the following use cases (used only if orchestration or other integration support is needed):

- Platform Integrity with Data Sovereignty and Signed Flavors
- Application Integrity

#### 3.15.2 Prerequisites

The Intel® Security Libraries Integration Hub can be run as a VM or as a bare-metal server. The Hub may be installed on the same server (physical or VM) as the Verification Service.

- The Verification Service must be installed and available
- The Authentication and Authorization Service must be installed and available
- The Certificate Management Service must be installed and available
- The Integration Hub database must be available
3.15.3 Package Dependencies

The Intel® SecL Integration Hub requires a number of packages and their dependencies:

If these are not already installed, the Integration Hub installer attempts to install these packages 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 your repository package lists before installation.

3.15.4 Supported Operating Systems


3.15.5 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB
- 1 GB free space to install the Verification Service services. Additional free space is needed for the Attestation Hub database and logs (database and log space requirements are dependent on the number of managed servers).
- One network interface with network access to the Verification Service.
- One network interface with network access to any integration endpoints (for example, OpenStack Nova).

3.15.6 Installation

3.15.6.1 Installing/Configuring the Database

The Intel® SecL-DC Integration Hub requires a Postgresql 11 database. Scripts (install_pgdb.sh, create_db.sh) are provided with the Hub that will automatically add the Postgresql repositories and install/configure a sample database. If this script will not be used, a Postgresql 11 database must be installed by the user before executing the Hub installation.

3.15.6.1.1 Using the Provided Database Installation Script

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 must be created initialized and tables created. Execute the create_db.sh script to configure the database.

./create_db.sh <database name> <database_username> <database_password>

### 3.15.6.2 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=<AH service user username>
AH_SERVICE_PASSWORD=<AH service user password>
MTWILSON_SERVER=<IP address or hostname of the Verification Service>

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=<database name>
ATTESTATION_HUB_DB_HOSTNAME=<database hostname or IP>
ATTESTATION_HUB_DB_PORTNUM=<database port; default is 5432>
ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
ATTESTATION_HUB_DB_USERNAME=<Database username>
ATTESTATION_HUB_DB_PASSWORD=<Database password>

AH_TLS_CERT DNS=<comma separated list of hostnames for the Hub; must match hostnames in AH_CERT_SAN_LIST from populate-users script>
AH_TLS_CERT_IP=<comma separated list of IP addresses for the Hub; must match IPs in AH_CERT_SAN_LIST from populate-users script>
```

3. Execute the installer binary.

```
./attestation-hub-5.1-SNAPSHOT.bin
```

### 3.16 Installing the Key Broker Service

#### 3.16.1 Required For

The KBS is REQUIRED for the following use cases:
- Workload Confidentiality (both VMs and Docker Containers)

#### 3.16.2 Prerequisites

The following must be completed before installing the Key Broker:
- The Verification Service must be installed and available
• The Authentication and Authorization Service must be installed and available
• The Certificate Management Service must be installed and available
• (Recommended; Required if a 3rd-party Key Management Server will be used) A KMIP 2.0-compliant 3rd-party Key management Server must be available.
  o The Key Broker will require the KMIP server’s client certificate, client key and root ca certificate.
  o The Key Broker uses the libkmip client to connect to a KMIP server
  o The Key Broker has been validated using the pykmip 0.9.1 KMIP server as a 3rd-party Key Management Server. While any general KMIP 2.0-compliant Key Management Server should work, implementation differences among KMIP providers may prevent functionality with specific providers.

3.16.3 Package Dependencies

3.16.4 Supported Operating Systems


3.16.5 Recommended Hardware

3.16.6 Installation

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

2) Create the installation answer file:

AAS_API_URL=https://<AAS IP or hostname>:8444/aas
CMS_BASE_URL=https://<CMS IP or hostname>:8445/cms/v1/
KMS_TLS_CERT_IP=<comma-separated list of IP addresses for the Key Broker>
KMS_TLS_CERT_DNS=<comma-separated list of hostnames for the Key Broker>
CMS_TLS_CERT_SHA384=<SHA384 hash of CMS TLS certificate>
BEARER_TOKEN=<installation token from populate-users script>

3) Execute the KBS installer.

./kms-5.1-SNAPSHOT.bin
3.16.6.1 Configure the Key Broker to use a KMIP-compliant Key Management Server

The Key Broker immediately after installation will be configured to use a filesystem key management solution. This should be used only for testing and POC purposes; using a secure 3rd-party Key management Server should be used for production deployments. To configure the Key Broker to point to a 3rd-party KMIP-compliant Key Management Server:

4) Copy the KMIP server’s client certificate, client key and root ca certificate into /opt/kms/configuration/ on the Key Broker

5) Change the ownership of these files to "kms:kms"

6) chown kms:kms /opt/kms/configuration/*

7) Configure the variables for kmip support as below
   1) kms config key.manager.provider com.intel.kms.keystore.kmip.KMIPKeyManager
   2) kms config kmip.server.address <IP>
   3) kms config kmip.server.port <PORT>
   4) kms config kmip.ca.certificates.path <path to kmip ca certificate>
   5) kms config kmip.client.certificate.path <path to kmip client certificate>
   6) kms config kmip.client.key.path <path to kmip client key>

8) Restart the Key Broker for the settings to take effect
kms restart

3.16.7 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.

3.16.7.1 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-----
MIID9TCCAl2gAwIBAgIBCTANBgkqhkiG9w0BAQwFADBQMgQwCQYDVQQGEwJVUzEL
3.16.7.2 Importing a PrivacyCA Certificate

Use OpenSSL to display the PrivacyCA certificate content:

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

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

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

-----END CERTIFICATE-----
3.17 Installing the Workload Policy Manager

3.17.1 Required For

The WPM is REQUIRED for the following use cases.

- Workload Confidentiality (both VMs and Docker Containers)

3.17.2 Package Dependencies

- (Required only if Docker Container encryption is needed) Docker-ce 19.03 must be installed. This is needed only if the option "WPM_WITH_CONTAINER_SECURITY=Yes" is set in the wpm.env answer file.

3.17.3 Supported Operating Systems


3.17.4 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

3.17.5 Installation

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

For Docker Container Encryption only, add the following line to the wpm.env installation answer file:

```
WPM_WITH_CONTAINER_SECURITY=yes
```

3) Execute the WPM installer:

```
./workload-policy-manager-2.1.0.bin
```
4 Authentication

Beginning in the Intel® SecL-DC 1.6 release, authentication is centrally managed by the Authentication and Authorization Service (AAS). This service uses a Bearer Token authentication method, which replaces the previous HTTP BASIC authentication. This service also centralizes the creation of roles and users, allowing much easier management of users, passwords, and permissions across all Intel® SecL-DC services.

To make an API request to an Intel® SecL-DC service, an authentication token is now required. API requests must now include an Authorization header with an appropriate token:

"Authorization: Bearer $TOKEN"

The token is issued by the AAS and will expire after a set amount of time. This token may be used with any Intel® SecL-DC service, and will carry the appropriate permissions for the role(s) assigned to the account the token was generated for.

4.1 Create Token

To request a new token from the AAS:

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

The response will be a token that can be used in the Authorization header for other requests. The length of time for which the token will be valid is configured on the AAS using the key AAS_JWT_TOKEN_DURATION_MINS (in the installation answer file during installation) or aas.jwt.token.duration.mins (configured on the AAS after installation). In both cases the value is the length of time in minutes that issued tokens will remain valid before expiring.

4.2 User Management

Users in Intel® SecL-DC are no longer restrained to a specific service, as they are now centrally managed by the Authentication and Authorization Service. Any user may now be assigned roles for any service, allowing user accounts to be fully defined by the tasks needed.
4.2.1 Username and Password requirements

Passwords have the following constraints:

- cannot be empty - ie must at least have one character
- maximum length of 255 characters

Usernames have the following requirements:

- Format: username[@host_name[domain]]
- [@host_name[domain]] is optional
- username shall be minimum of 2 and maximum of 255 characters
- username allowed characters are alphanumeric, ., -, _ - but cannot start with -.
- Domain name must meet requirements of a host name or fully qualified internet host name
- Examples
  - admin, admin_wls, admin@wls, admin@wls.intel.com, wls-admin@intel.com

4.2.2 Create User

POST https://<IP or hostname of AAS>:8444/aas/users
Authorization: Bearer <token>

```json
{
    "username" : "<username>",
    "password" : "<password>"
}
```

4.2.3 Search User

GET https://<IP or hostname of AAS>:8444/aas/users?<parameter>=<value>
Authorization: Bearer <token>

4.2.4 Change User Password

PATCH https://<IP or hostname of AAS>:8444/aas/users/changepassword

```json
{
    "username" : "<username>",
    "old_password" : "<old_password>",
    "new_password" : "<new_password>",
    "password_confirm" : "<new_password>"
}
```
4.2.5 **Delete User**

DELETE https://<IP or hostname of AAS>:8444/aas/users/<User ID>
Authorization: Bearer <token>

4.3 **Roles and Permissions**

Permissions in Intel® SecL-DC are managed by Roles. Roles are a set of predefined permissions applicable to a specific service. Any number of Roles may be applied to a User. While new Roles can be created, each Intel® SecL service defines permissions that are applicable to specific predetermined Roles. This means that only pre-defined Roles will actually have any permissions. Role creation is intended to allow Intel® SecL-DC services to define their permissions while allowing role and user management to be centrally managed on the AAS. When a new service is installed, it will use the Role creation functions to define roles applicable for that service in the AAS.

4.3.1 **Create Role**

POST https://<AAS IP or Hostname>:8444/aas/roles
Authorization: Bearer <token>

```
{
    "service": "<Service name>",
    "name": "<Role Name>",
    "permissions": [<array of permissions>]
}
```

- **Service** field contains a minimum of 1 and maximum of 20 characters. Allowed characters are alphanumeric plus the special characters -, _, @, . , ,
- **Name** field contains a minimum of 1 and maximum of 40 characters. Allowed characters are alphanumeric plus the special characters -, _, @, . , ,
- **Service** and **Name** fields are mandatory
- **Context** field is optional and can contain up to 512 characters. Allowed characters are alphanumeric plus the special characters -, _, @, . , ,=
- **Permissions** field is optional and allow up to a maximum of 512 characters.

The Permissions array must a comma-separated list of permissions formatted as resource:action:

Permissions required to execute specific API requests are listed with the API resource and method definitions in the API documentation.
4.3.2 Search Roles

GET https://<AAS IP or Hostname>:8444/aas/roles?<parameter>=<value>
Authorization: Bearer <token>

Search parameters supported:

- Service=<name of service>
- Name=<role name>
- Context=<context>
- contextContains=<partial “context” string>
- allContexts=<true or false>
- filter=false

4.3.3 Delete Role

DELETE https://<AAS IP or Hostname>:8444/aas/roles/<role ID>
Authorization: Bearer <token>

4.3.4 Assign Role to User

POST https://<AAS IP or Hostname>:8444/aas/users/<user ID>/roles
Authorization: Bearer <token>

{  
  "role_ids": ["<comma-separated list of role IDs>"
}

4.3.5 List Roles Assigned to User

GET https://<AAS IP or Hostname>:8444/aas/users/<user ID>/roles
Authorization: Bearer <token>

4.3.6 Remove Role from User

DELETE https://<AAS IP or Hostname>:8444/aas/users/<user ID>/roles/<role ID>
Authorization: Bearer <token>
## 4.3.7 Role Definitions

The following roles are created during installation (or by the CreateUsers script) and exist by default.

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Permissions</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA:Administrator</td>
<td>TA:<em>::</em></td>
<td>Used by the Verification Service to access Trust Agent APIs, including retrieval of TPM quotes, provisioning Asset Tags and SOFTWARE Flavors, etc.</td>
</tr>
<tr>
<td>VS:ReportRetriever</td>
<td>VS: [&quot;reports:retrieve:<em>&quot;, &quot;reports:search:</em>&quot;, &quot;hosts:search:<em>&quot;, &quot;hosts:retrieve:</em>&quot;]</td>
<td>Used by the Integration Hub to retrieve attestation reports from the Verification Service</td>
</tr>
<tr>
<td>KMS:Keymanager</td>
<td>KMS: [&quot;keys:create:<em>&quot;, &quot;keys:transfer:</em>&quot;]</td>
<td>Used by the WPM to create and retrieve symmetric encryption keys to encrypt workload images</td>
</tr>
<tr>
<td>WLS:FlavorsImageRetrieval</td>
<td>WLS: image_flavors:retrieve:*</td>
<td>Used by the Workload Agent during Workload Confidentiality flows to retrieve the image Flavor</td>
</tr>
<tr>
<td>VS: ReportCreator</td>
<td>VS: [&quot;reports:create:*&quot;]</td>
<td>Used by the Workload Service to create new attestation</td>
</tr>
</tbody>
</table>
reports on the Verification Service as part of Workload Confidentiality key retrievals.

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Permissions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td><em>:</em>:*</td>
<td>Global administrator role used for the initial administrator account. This role has all permissions across all services, including permissions to create new roles and users.</td>
</tr>
<tr>
<td>AAS: Administrator</td>
<td><em>:</em>:*</td>
<td>Administrator role for the AAS only. Has all permissions for AAS resources, including the ability to create or delete users and roles.</td>
</tr>
<tr>
<td>AAS: RoleManager</td>
<td><code>AAS: [roles:create:*, roles:retrieve:*, roles:search:*, roles:delete:*]</code></td>
<td>AAS role that allows all actions for Roles, but cannot create or delete Users or assign Roles to Users.</td>
</tr>
<tr>
<td>AAS: UserManager</td>
<td><code>AAS: [users:create:*, users:retrieve:*, users:store:*, users:search:*, users:delete:*]</code></td>
<td>AAS role with all permissions for Users, but has no ability to create Roles or assign Roles to Users.</td>
</tr>
<tr>
<td>AAS: UserRoleManager</td>
<td><code>AAS: [user_roles:create:*, user_roles:retrieve:*,</code></td>
<td>AAS role with permissions to assign Roles to Users.</td>
</tr>
<tr>
<td>Role</td>
<td>Permissions</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>user_roles:search:*</td>
<td></td>
<td>Users, but cannot create delete or modify Users or Roles.</td>
</tr>
<tr>
<td>user_roles:delete:*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VS: AttestatioNRegister</td>
<td>VS: [host_tls_policies:create:<em>, hosts:create:</em>, hosts:store:<em>, hosts:search:</em>, host_unique_flavors:create:<em>, flavors:search:</em>, tpm_passwords:retrieve:<em>, tpm_passwords:create:</em>, host_aiks:certify:*]</td>
<td>Role used for Trust Agent provisioning. Used to create the installation token provided during installation.</td>
</tr>
<tr>
<td>VS: Certifier</td>
<td>VS: host_signing_key_certificates:create:*</td>
<td>Used for installation of the Workload Agent</td>
</tr>
</tbody>
</table>
5  Connection Strings

Connection Strings define a remote API resource endpoint that will be used to communicate with the registered host for retrieving TPM quotes and other host information. Connection Strings differ based on the type of host.

5.1  Trust Agent (Windows and Linux)

The Trust Agent connection string connects directly to the Trust Agent on a given host. The Verification Service will use a service account with the needed Trust Agent permissions to connect to the Trust Agent. In previous Intel® SecL versions, each Trust Agent had its own unique user access controls. Starting in the 1.6 release, all authentication has been centralized with the new Authentication and Authorization Service, eliminating the need for credentials to be provided for connection strings connecting to Trust Agent resources.

intel:https://<HostNameOrIp>:1443

5.2  VMware ESXi

The VMware ESXi connection string is actually directed to vCenter, not the actual ESXi host. Many ESXi hosts managed by the same vCenter server will use the same connection string. The username and password specified are vCenter credentials, and the vCenter “Validate Session” privilege is required for access.

vmware:https://<vCenterHostNameOrIp>:443/sdk;h=<hostname of ESXi host>;u=<username>;p=<password>
6 Platform Integrity Attestation

Platform attestation is the cornerstone use case for ISecL. Platform attestation involves taking measurements of system components during system boot, and then cryptographically verifying that the actual measurements taken matched 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.

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

This section includes basic REST API examples for these workflows. See the Javadoc for more detailed documentation on REST APIs supported by ISecL.

Typical workflows in the datacenter might include:

- Creating a set of acceptable flavors for attestation with automatic flavor matching that represent the known-good measurements for acceptable BIOS and OS versions in the datacenter
- Registering hosts for attestation with automatic flavor matching
- Upgrading hosts in the datacenter to a new BIOS or OS version
- Removing hosts from the Verification Service
- Removing flavors
- Provisioning asset tags to hosts
- Invalidating asset tags
- Retrieving current attestation reports
- Retrieving current host state information
- Remediating an untrusted attestation
6.1 **Host Registration**

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

6.1.1 **Trust Agent**

6.1.1.1 **Registration via Trust Agent Command Line**

The Trust Agent can register the host with a Verification Service by running the following command:

```bash
tagent create-host <Verification Service base URL> <username> <password>
```

*Note: Because VMWare ESXi hosts do not use a Trust Agent, this method is not applicable for registration of ESXi hosts.*

6.1.2 **Registration via Verification Service API**

Any Trust Agent or VMware ESXi host can be registered using a Verification Service API request. Registration can be performed with or without a set of existing Flavors. Rules for Flavor matching can be set by using the Flavor Group in the request; if no Flavor Group is specified, the “mtwilson_automatic” Flavor Group will be used. See the Flavor Management section for additional details on Flavors, Flavor Groups, and Flavor matching.

6.1.2.1 **Sample Call**

```plaintext
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": "<connection string>",
   "flavorgroup_name": "",
   "description": "<description>"
}
```

Requires the permission “hosts:create”
6.2 Flavor Creation for Automatic Flavor Matching

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

Flavors can be created manually, or can be imported from an example host.

Flavors are automatically matched to hosts based on the Flavorgroup used by the host and the Flavors, and the Flavor Match Policies of the Flavorgroup. The ISecL Verification Service creates a default Flavorgroups during installation called “automatic.” This Flavorgroup is configured to be used as a pool of all acceptable Flavors in a given environment, and will automatically match the appropriate Flavor parts to the correct host. This Flavorgroup is used by default and is expected to be useful for the majority of deployments. If no Flavorgroup is specified when creating a Flavor, it will be placed in the “automatic” Flavorgroup.

Flavors are also divided into Flavor parts, which correspond to the PLATFORM, OS, HOST_UNIQUE, SOFTWARE, and ASSET_TAG measurements. These can be created and maintained separately (so that users can manage acceptable OS and BIOS versions, rather than entire host configurations). By default, if not specified, the Verification Service will import Flavors as separate Flavor parts, as appropriate for the host type.

By using individual Flavor parts, individual versions of OS or PLATFORM measurements can be managed and automatically mapped. Whenever a host changes states (Untrusted, Connected, etc.) the Verification Service will attempt to match appropriate Flavors to that host. If a Flavor is removed or added, all appropriate hosts will be updated to use the new Flavor, or to no longer use the deleted Flavor. Hosts that are currently using a BIOS where that BIOS versions’ PLATFORM Flavor was deleted will now appear Untrusted, for example. This can be used to easily flag as Untrusted hosts that are using software that has been End-Of-Lifed, or perhaps an OS kernel with a known security vulnerability.

**Note:** See the Flavor Management section for additional details on how flavors can be managed, and how the Flavor matching engine works. The sample workflow provided here is intended to be an introduction only.
6.2.1 Importing a Flavor from a Sample Host

POST https://server.com:8443/mtwilson/v2/flavors
input:
{ "connection_string": "<connection string>",
"partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
"flavorgroup_name": "",
"tls_policy_id":"TRUST_FIRST_CERTIFICATE"}

Requires the permission "flavors:create"

Note: The HOST_UNIQUE Flavor parts, used by Red Hat Enterprise Linux and VMWare ESXi host types, MUST be created for each registered host of that type, and should in general be imported from that host. This means that importing the HOST_UNIQUE flavor should always be done for each host registered (except for Windows hosts, which do not have HOST_UNIQUE measurements).

To import ONLY the HOST_UNIQUE Flavor part from a host:

POST https://server.com:8443/mtwilson/v2/flavors
input:
{ "connection_string": "<connection string>",
"partial_flavor_types": ["HOST_UNIQUE"],
"flavorgroup_name": "",
"tls_policy_id":"TRUST_FIRST_CERTIFICATE"}

Requires the permission "flavors:create"

6.2.2 Creating a Flavor Manually

Flavors can be directly created (rather than importing from a sample host) if the required information is known. If no Flavorgroup is specified, the Flavor will be placed in the “automatic” group. Note that the “label” is a required field and must be unique.

POST https://server.com:8443/mtwilson/v2/flavors
{ "connection_string": "",
"tls_policy_id": "",
"flavor_collection": {
  "flavors": [{
    "meta": {
      "vendor": "INTEL",
      "description": {
        "flavor_part": "PLATFORM",
      }
    }
  }
}
"label" : "Intel Corporation_SE5C610.86B.01.01.1008.031920151331_TPM1.2",
"bios_name": "Intel Corporation",
"bios_version": "SE5C620.86B.00.01.0004.071220170215",
"tpm_version": "2.0"
},

"hardware":
  {
    "processor_info": "...",
    "processor_flags": "...",
    "feature":
      {
        "tpm":
          {
            "enabled": true,
            "pcr_banks":
              {
                "SHA1",
                "SHA256"
              }
          },
        "txt":
          {"enabled": true}
      }
  },

"pcrs":
  {
    "SHA1":
      {
        "pcr_0":
          {"value": "d2ed125942726641a7260c4f92beb67d531a0def"},
        "pcr_17":
          {"value": "1ec12004b371e3afd43d04155abde7476a3794fa", "event": ...
      }
  }

Requires the permission "flavors:create"

6.3 Creating the Default SOFTWARE Flavor (Linux Only)

As part of the new Application Integrity feature added in Intel® SecL-DC version 1.5, a new default SOFTWARE Flavor part is provided so that the Linux Trust Agent itself can be measured and included in the attestation process. The default SOFTWARE Flavor includes a manifest for the static files and folders in the Trust Agent. The manifest is automatically deployed to each Linux Trust Agent during the provisioning step.

Note: The Linux Trust Agent must be rebooted after the Provisioning step is completed (typically Provisioning happens during installation, based on whether all of the required variables are set in the trustagent.env file). Rebooting allows the default SOFTWARE Flavor manifest to be measured and extended to the TPM PCRs. If the reboot is not performed, the system will require a SOFTWARE Flavor, but the
measurements will not exist, and the system will appear Untrusted. If an un-rebooted host is used to create the SOFTWARE Flavor, the Flavor will be created based on measurements that do not exist, and will fail.

The SOFTWARE Flavor part should be created separately from the other Flavor parts. Only one default SOFTWARE Flavor needs to be created for each version of the Linux Trust Agent. If the SOFTWARE Flavor for the same Trust Agent version is imported multiple times, subsequent imports will fail as the Flavor already exists.

To import the SOFTWARE Flavor part from a host:

```
POST https://server.com:8443/mtwilson/v2/flavors
input:
  { "connection_string": "<connection string>",
    "partial_flavor_types": ["SOFTWARE"],
    "flavorgroup_name": "",
    "tls_policy_id":"TRUST_FIRST_CERTIFICATE"
}
```

Requires the permission "flavors:create"

6.4 Creating and Provisioning Asset Tags

Asset Tags represent a set of key/value pairs that can be associated with a host in hardware. This enables usages around restricting workflows to specific hosts based on tags, which could include location information, compliance tags, etc.

ISecL creates Asset Tags by creating a certificate containing the list of key/value pairs to be tagged to the host, with the host’s hardware UUID as the certificate subject. A hash of this certificate is then written to an NVRAM index in the host’s TPM. This value is included in TPM quotes, and can be attested using an Asset Tag flavor that matches up the expected value and the actual key/value pairs.

6.4.1 Creating Asset Tag Certificates

Asset Tag certificates can be created with a single REST API call, with any number of key/value pairs. Note that one certificate must be created for each host to be tagged, even if they will all be tagged with identical key/value pairs.

```
POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
```
6.4.2 Deploying Asset Tags

6.4.2.1 Windows and Red Hat Enterprise Linux

Asset Tags can be provisioned to a Windows or RHEL host via a REST API request on the Verification Service that will in turn make a request to the Trust Agent on the host to be tagged.

```plaintext
POST https://verification.server.com:8443/mtwilson/v2/rpc/deploy-tag-certificate
{
   "certificate_id": "<certificate ID>"
}
```

6.4.2.2 VMWare

Since VMWare ESXi hosts do not use a Trust Agent, the process for writing Asset Tags to a VMWare host is different from RHEL or Windows. A new interface has been added to ESXi via a new "esxcli" command starting in vSphere 6.5 Update 2 that allows the Asset Tag information to be written to the TPM via a command-line command. The older process is also described below.

The high-level workflow for using Asset Tags with VMWare ESXi is:

1. Create the Asset Tag Certificate for the host.
2. Calculate the Certificate Hash value.
3. Provision the Certificate Hash value to the host TPM and reboot
4. Create the Asset Tag Flavor.
Note: Asset Tag is currently not supported for VMWare hosts using TPM 2.0.

6.4.2.2.1 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 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.

6.4.2.2.2 Provision the Certificate Hash to the Host TPM

Due to a new feature added in vSphere 6.5 Update 2, the process for provisioning Asset Tags on VMWare ESXi hosts has been significantly improved. Both the old and new process for provisioning Asset Tags is documented below. Intel recommends using vSphere 6.5 Update 2 or later due to the significant difference in the process.

vSphere 6.5 Update 2 or Later

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 command:

   1. 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

   4. Reboot the host. After rebooting, the TPM PCR 22 will have the measured value of the hash.
vSphere 6.5 Update 1 or Older

There is no direct interface from vCenter or ESXi previous to vSphere 6.5 Update 2 that will write the Tag information to the host TPM.

Writing Asset Tag information to a TPM requires TPM ownership; vCenter takes TPM ownership with a secret password at boot time. This means that the process for writing Asset Tags to a VMWare host requires:

1. Clear TPM ownership.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

2. Reactivate TPM/TXT.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

3. Booting to an OS that has the ability to issue TPM commands
   a. Typically the provisioning OS used is Ubuntu or RHEL, booted temporarily using PXE.

4. Writing the Tag information
   a. The TPM index 0x40000010 must be defined, and the hash of the Asset Tag certificate must be written to that index.

5. Clear TPM ownership.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

6. Reactivate TPM/TXT.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

7. Boot back to VMWare ESXi.

When the system is rebooted to ESXi, the Trusted Boot process will extend the value to PCR22, and this value can be used during attestation.

6.4.2.2.3 Creating the Asset Tag Flavor (VMWare ESXi Only)

While for RHEL and Windows hosts the Asset Tag Flavor is automatically created during the Tag Provisioning step, for VMWare ESXi hosts the 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 vCenter connection string>",
Once the Asset Tag Flavor is imported, the host can be attested including Asset Tags as normal.

### 6.5 Retrieving Current Attestation Reports

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

### 6.6 Retrieving Current Host State Information

GET https://verification.service.com:8443/mtwilson/v2/host-status?latestPerHost=true

### 6.7 Upgrading Hosts in the Datacenter to a New BIOS or OS Version

Software and firmware updates are a common occurrence in the datacenter. Automatic Flavor matching makes this process relatively simple:

1. Create a new Flavor for the new version. This may be manually created or imported directly from a sample host that has already received the upgrade. Be sure to create new Flavors for each TPM version represented in your datacenter.

   POST https://server.com:8443/mtwilson/v2/flavors
   input:
   { "connection_string": "<connection string>" ,
   "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
   "flavorgroup_name": "",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE"}

2. Update the hosts to the new software or firmware version as normal. On the next attestation attempt, the Verification Service will automatically match the updated hosts to the new Flavor.

3. (Optional) If desired, delete the Flavor for the older version after the update is completed. This will cause any hosts that are still using the old version to attest as Untrusted. Which can easily flag hosts that missed the upgrade for remediation.

DELETE https://verification.service.com:8443/mtwilson/v2/flavors/<Flavor ID>
6.8 Removing Hosts From the Verification Service

Hosts can be deleted at any time. Reports for that host will remain in the Verification Service database for audit purposes.

DELETE https://verification.service.com:8443/mtwilson/v2/hosts/<host ID>

The Host ID can be retrieved either at the time the host is created, or by searching hosts using the host’s hostname.

6.9 Removing Flavors

Flavors can be deleted; this will cause any hosts that match the deleted Flavor to evaluate as Untrusted. This can be done if, for example, an old BIOS version needs to be retired and should no longer exist in the datacenter. By deleting the PLATFORM Flavor, hosts with the old BIOS version will attest as Untrusted, flagging them for easy remediation.

DELETE https://verification.service.com:8443/mtwilson/v2/flavors/<Flavor ID>

6.10 Invalidating Asset Tags

Asset Tags can be deleted in two ways.

Deleting the ASSET_TAG Flavor part will retain the Asset Tag certificate in the database, but will cause the host using this Tag to no longer use the Asset Tag for attestation (the Tag result will be disregarded and no tags will be exposed in the attestation Reports).

DELETE https://verification.service.com:8443/mtwilson/v2/flavors/<Asset Tag Flavor ID>

Deleting the actual Asset Tag certificate will remove the certificate from the database, but will not actually affect attestation results (the authority for attestation results is the Flavor).

DELETE https://verification.service.com:8443/mtwilson/v2/tag-certificates/<Asset Tag Certificate ID>

6.11 Remediating an Untrusted attestation

Hosts can become Untrusted for a wide variety of causes. The first clue to finding the root cause for an Untrusted attestation is the attestation Report itself – the Report will show Trust results for the PLATFORM, OS, HOST_UNIQUE, and ASSET_TAG Flavor parts individually, along with the OVERALL trust. If the Report shows that the PLATFORM Flavor part trust is
“false” for example, it means that the PLATFORM measurements did not match any Flavors in the host’s Flavorgroup.

Untrusted attestation Reports will contain “faults” that describe the specific attestation rules that were not satisfied. This often shows enough information to describe the cause of the Untrusted status. A fault like “RequiredButNotDefined” means that a Flavor part is required by the Flavorgroup policy, but no Flavors for that Flavor part exist in the Flavorgroup (for example, generally Flavorgroups should always require a PLATFORM Flavor part; if no PLATFORM Flavors are in the Flavorgroup, hosts in the Flavorgroup will attest with this fault).

Other faults include:

“PcrMatchesConstant” - describes a rule that evaluates whether a TPM PCR has a specific value

“PcrEventLogIntegrity” - the module event log is replayed during attestation to verify that the resulting measurement matches the actual value in the module PCR. If the replay does not match, it indicates the event log cannot itself be trusted.

“AikCertificateTrusted” – This rule evaluates whether the TPM quote was signed by the TPM associated with this host. As part of host registration, the public half of the Attestation Identity Keypair is captured, and this public key is used to verify the signature on TPM quotes from that host.

See the Appendix for a full list of the rules evaluated during Attestation.

The Flavor matching engine will use the most-similar Flavor for the attestation Report in the case of an Untrusted result.

The fault will explain in a general sense what rule the host attestation violated. To remediate, the rule will need to be satisfied. This could mean creating a new Flavor to match the actual observed values, or it could mean that the host has been tampered with and should have its BIOS flashed or OS reloaded.

### 6.12 Attestation Reporting

Attestation results are delivered in the form of Host Reports. A Report can delivered in several different formats, which can change the type of data returned.

The preferred format for Host Reports is a SAML attestation. A SAML-formatted report includes a chain or signatures that provides auditability for the Report. The SAML attestation will include the base trust status of the host, as well as the overall trust for each individual Flavor used in the attestation. The Report will also contain host information, such as TPM version, Operating System name and version, BIOS version, etc. The SAML Report will not,
however, contain individual measurements and comparisons of values. This format of the Report is ideal for securely communicating the trust status of a host and for audit history.

Attestation Reports can also be retrieved in JSON or XML format. These formats will not include the signature chain provided in the SAML format, but will contain the actual measurement values and expected Flavor values used for comparison. These reports are typically used for remediation, because they will show specifically why a given Host attested as Untrusted.

The format for a Report is determined by the “Accept” header in the request.

Attestations are automatically generated in the Verification Service by a repeating scheduled background process. This process looks for Attestation Reports that are close to expiration, and triggers a new Attestation Report. By default, Attestation Reports are valid for 90 minutes, and the background refresh process will trigger a new attestation when a Report is found to be within 3 minutes of expiration.

A user can either retrieve the most recent currently valid Attestation Report for a given host, or may trigger a new Attestation Report to be generated. Typically, it is best to retrieve an existing Report for performance reasons. Generating a new Attestation Report requires the generation of a new TPM quote from the TPM of the host being attested; TPM performance differs greatly between vendors, and a quote can take anywhere between 2-7 seconds to generate.

**6.12.1 Sample Call – Generating a New Attestation Report**

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

input: {"host_name":"host-1"}

Requires the permission “reports:create”

**6.12.2 Sample Call – Retrieving an Existing Attestation Report**

GET https://server.com:8443/mtwilson/v2/reports?hostName=HostName.server.com

Below are the supported criteria options in order of precedence. If no host filter criteria is specified, then results are returned for all active hosts.

- id - unique UUID of the report entry in the database
- hostId - unique UUID of the host entry in the database
- hostName - name of the host
- hostHardwareId - hardware UUID of the host
• hostStatus - current state of the host, which supports the following options:
  — CONNECTED - host is in connected state
  — QUEUE - host is in queue to be processed
  — CONNECTION_FAILURE - connection failure
  — UNAUTHORIZED - unauthorized
  — AIK_NOT_PROVISIONED - AIK certificate is not provisioned
  — EC_NOT_PRESENT - endorsement certificate is not present
  — MEASURED_LAUNCH_FAILURE - TXT measured launch failure
  — TPM_OWNERSHIP_FAILURE - TPM ownership failure
  — TPM_NOT_PRESENT - TPM is not present
  — UNSUPPORTED_TPM - unsupported TPM version
  — UNKNOWN - unknown host state

Requires the permissions “reports:search”

Other search criteria may also be used. By default, the most recent currently valid attestation is returned. However, different query parameters can be used to retrieve all attestations for a specific host over the last 30 days, for example.

6.13 Integration

Intel® SecL can be integrated with scheduler services (or potentially other services) to provide additional security controls. For example, by integrating Intel® SecL with the OpenStack scheduler service, the OpenStack placement service can incorporate the Intel® SecL security attributes into VM scheduling.

6.13.1 Integration Hub

The Integration Hub acts as the central integration point between the Verification Service and any number of third party services. The primary purpose of the Hub is to collect and maintain up-to-date attestation information, and to “push” that information to the external services. The secondary purpose is to allow for multitenancy. The Verification Service does not allow for permissions to be applied for specific hosts; a user with the “attestation” role can access all attestations. The Integration Hub allows hosts to be associated with specific tenants; the tenants do not have direct access to the Verification Service, and the Hub will push attestations only for the associated hosts to a given tenant’s integration endpoints.

For example, Tenant A is using hosts 1-10 for an OpenStack environment. Tenant B is using hosts 11-15 for a Docker environment. The Attestation Hub can push the information for hosts 1-10 to Tenant A’s OpenStack endpoint, and hosts 11-15 to Tenant B’s Docker endpoint. Neither tenant will have
access to the Verification Service, and will not be able to see attestation or other host details regarding infrastructure used by other tenants.

Different integration endpoints can be added to the Integration Hub through a plugin architecture. By default, the Attestation Hub includes plugins for OpenStack and Kubernetes (Kubernetes deployments require the additional installation of two Intel® SecL-DC Custom Resource Definitions on the Kube Master).
6.13.2 Integration with OpenStack

Starting in the Rocky release, OpenStack can now use “Traits” to provide qualitative data about Nova Compute hosts, and to establish Trait requirements for VM instances. The updated scheduler will place VMs requiring a given Trait on Nova Compute nodes that meet the Trait requirements.

Intel SecL-DC uses the Integration Hub to continually push platform integrity and Asset Tag information to the OpenStack Traits resources. This means the OpenStack scheduler natively supports workload scheduling incorporating Intel SecL-DC security attributes, including attestation report Trust status and Asset Tags. The OpenStack Placement Service will automatically attempt to place images with Trait requirements on compute nodes that have those Traits.

**NOTE:** This control only applies to instances launched using the OpenStack scheduler, and the Traits functions will not
affect manually-launched instances where a specific Compute Node is defined (since this does not use the scheduler at all). Intel SecL-DC uses existing OpenStack interfaces and does not modify OpenStack code. The datacenter owner or OpenStack administrator is responsible for the security of the OpenStack workload scheduling process in general, and Intel recommends following published OpenStack security best practices.

6.13.2.1 Prerequisites

- Verification Service must be installed and running.
- OpenStack* Rocky Nova, Glance, Horizon, and Keystone services must be installed and running
- The Integration Hub must be installed and running.

6.13.2.2 Setting 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 set the Image Traits for Intel SecL-DC, a specific naming convention is used. This naming convention will match the Traits that the Integration Hub will automatically push to OpenStack. Two types of Traits are currently supported – one Trait is used to require that the Compute Node be Trusted in the Attestation Report, and the other Trait is used to require specific Asset Tag key/value pairs.

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

For example, to define a Trait that will require an Asset Tag where “State = CA,” use the following:

CUSTOM_ISECL_AT_TAG__STATE_CA=required

These Traits can be set using CLI commands for OpenStack Glance:
openstack image set --property trait:CUSTOM_ISECL_AT_STATE__CA=required <image_name>

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

To remove a Trait so that it is no longer required for an Image:

openstack image unset --property trait:CUSTOM_ISECL_AT_STATE__CA <image_name>

openstack image unset --property trait:CUSTOM_ISECL_TRUSTED <image_name>

### 6.13.2.3 Configuring the Integration Hub for Use with OpenStack

After installation, the Integration Hub will automatically start retrieving Reports from the Verification Service. However, the Hub needs to be configured with endpoint information defining credentials and URLs for the OpenStack environment, and needs to have Hosts assigned to specific Tenants so that the right host info can be pushed to the correct OpenStack.

### 6.13.2.4 Integration Hub Tenant Creation

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. Below is an example using OpenStack where the api.endpoint is Nova and the auth.endpoint is Keystone.

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

{
  "name": "<Tenant name>",
  "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",
```
6.13.2.5 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

6.13.2.6 Assign Hosts to Tenants

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.

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

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.
6.13.2.7 Scheduling Instances

Once Trait requirements are set for Images and the Integration Hub is configured to push attributes to OpenStack, instances can be launched in OpenStack as normal. As long as the OpenStack Nova scheduler is used to schedule the workloads, only compliant Compute Nodes will be scheduled to run instances of controlled Images.

**NOTE**: This control only applies to instances launched using the OpenStack scheduler, and the Traits functions will not affect manually-launched instances where a specific Compute Node is defined (since this does not use the scheduler at all). Intel SecL-DC uses existing OpenStack interfaces and does not modify OpenStack code. The datacenter owner or OpenStack administrator is responsible for the security of the OpenStack workload scheduling process in general, and Intel recommends following published OpenStack security best practices.
6.13.3 Integration with Kubernetes

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.

6.13.3.1 Prerequisites

- Verification Service must be installed and running.
- Kubernetes Master Node must be installed and running
- Kubernetes Worker Nodes must be configured as physical hosts and attached to the Master Node
- The Integration Hub must be installed and running.

6.13.3.2 Installing the Intel® SecL Custom Resource Definitions

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.

Two CRDs are required for integration with Intel® SecL – an extension for the Master nodes, and a scheduler extension. A single installer will deploy both of these CRDs, but they are deployed in different ways. The Master node extension is deployed as a Kubernetes pod using the “secl” namespace. The scheduler extension is installed as a binary service. In a future release, the scheduler extension will be migrated to be deployed as a Kubernetes pod like the Master node extension.

To deploy the Kubernetes integration CRDs for Intel® SecL:
1) Add a mount path to the /etc/kubernetes/manifests/kube-scheduler.yaml file for the Intel SecL scheduler extension:

```yaml
- mountPath: /opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/
  name: extended sched
  readOnly: true
```

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

```yaml
- hostPath:
  path: /opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/
  type: ""
  name: extended sched
```

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

```yaml
- 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

```sh
./isecl-k8s-extensions.bin-v2.1.0
```

5) The installer will output a set of files 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 files will be copied:

- 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) Restart the Integration Hub so that the keystore changes take effect:

```
attestation-hub restart
```

7) On the Integration Hub system, set the permissions on the copied files:
chown attestation-hub:attestation-hub *p12

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

```
scp /opt/attestation-hub/configuration/hub_public_key.pem root@K8s.server.com:/opt/isecl-k8s-extensions/isecl-k8s-scheduler/config/
```

9) Restart the Master node CRD by deleting the deployment and relaunching it:

```
kubectl apply -f /opt/isecl-k8s-extensions/yamls/secl-controller.yaml
```

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

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

To verify the Scheduler CRD has been deployed:

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

To verify the Master CRD pod has been deployed:

```
skubectl get pods -n secl
```

### 6.13.3.3 Integration Hub Tenant Creation

Generate an authentication token from Kubernetes. This token will be used by the Integration Hub to access Kubernetes:

```
kubectl describe secrets -n secl | grep 'token:'
```

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller or Kubernetes API endpoint. Below is an example using OpenStack where the api.endpoint is the Kubernetes Master and the auth.endpoint is a client token and server keystore. Note that the `kubernetes.api.bearer.token` value is the authentication token retrieved in the command above.

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

{
    "name": "<tenant_name>",
    "plugins": [
    {
        "name": "kubernetes",
        "properties": [ ]
    }
```
"key": "api.endpoint",
"value": "https://<k8s_master>:6443"
},
{
"key": "tenant.name",
"value": "<tenant_name>"
},
{
"key": "plugin.provider",
"value": "com.intel.attestationhub.plugin.kubernetes.KubernetesPluginImpl"
},
{
"key": "kubernetes.server.keystore",
"value": "/opt/attestation-hub/configuration/k8s_trust.p12"
},
{
"key": "kubernetes.server.keystore.password",
"value": "<server_keystore_password>"
},
{
"key": "kubernetes.api.bearer.token",
"value": "<client_token>"
}
]
}
]

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 and `kubernetes.server.keystore.password` must be the keystore password output by the scheduler extensions installer. To retrieve the client keystore password:

cat /opt/attestation-hub/configuration/root_keystore.properties

To retrieve the `kubernetes.api.bearer.token`:

kubectl describe secrets -n secl | grep 'token:'

### 6.13.3.4 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

6.13.3.5 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-e411-904d-e017a4403562"
  ]
}

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.

6.13.3.6 Configuring Pods to Require Intel® SecL Attributes

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

```bash
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) Add the following to any Pod creation files:

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: isecl.trusted
                operator: In
                values:
                  - "true"
              - key: isecl.TAG_Country
                operator: In
                values:
                  - CA
                  - US
```
key: isecl.TAG_Customer
operator: In
values:
  - Coke
  - Pepsi

key: isecl.TAG_State
operator: In
values:
  - CA

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

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.

6.13.3.7 Taint Untrusted Worker Nodes

Optionally, the Intel® SecL Kubernetes CRDs can be configured to flag worker nodes as “tainted” to prevent any pods from launching on them. This restriction is applied regardless of whether the pod has a specific trust policy – if a worker node is flagged as “tainted,” no pods will be launched on that worker.

This setting is disabled by default. To enable this setting:

1) Edit the secl-controller.yaml file under /opt/isecl-k8s-extensions/yaml/secl-controller.yaml and set “TAINT_UNTRUSTED_NODES=true”

2) Run: kubectl apply -f /opt/isecl-k8s-extensions/yaml/secl-controller.yaml

Worker nodes that attest as untrusted will be “tainted” with the NoExecute flag and unable to launch pods.

If a worker was previously considered tainted and the untrusted state is resolved, the Intel® SecL CRDs will remove the tainted flag and the worker will be able to launch pods again.
Workload Confidentiality builds upon Platform Attestation to protect data in virtual machine and container images. At its core, this feature is about allowing an image owner to set policies that define the conditions under which their image will be allowed to run; if the policy conditions are met, the decryption key will be provided, and if the conditions are not met, the image will remain encrypted and inaccessible. This provides a level of enforcement beyond integration with orchestrators, and protects sensitive data when the image is at rest.

Workload Encryption relies on Platform Attestation to define the security attributes of hosts. When a protected image is launched, the Workload Agent on the host launching the VM or container image will detect the attempt (using either Libvirt hooks for VMs, or as a function of the Docker Secure Overlay Driver in the case of containers) and use the Image ID to find the Image Flavor on the Workload Service. The Workload Service will retrieve the current trust report for the host launching the image, and use that report to make a key retrieval request to the key transfer URL retrieved from the image flavor. The key transfer URL refers to the URL to the image owner’s Key Broker Service, along with the ID of the key needed.

In a typical production deployment, a Cloud Service Provider would enable Intel® SecL-DC security controls by installing the Intel® SecL-DC applications (with the exception of the Key Broker and Workload Policy Manager), and configuring each workload host to be Trusted (as per the Platform Integrity Attestation use case).

The owner of the workload image(s) to be protected (for example, the end customer of the CSP) must install a Key Broker Service (which must be available for network communication from the Workload Service hosted on the CSP), the Workload Policy Manager, and their own Authentication and Authorization Service and Certificate Management Service (these will manage authentication and certificates for the KBS and WPM).

Any number of image owner customers with their own unique KBS/WPM/AAS/CMS deployments may protect images that can be run by a single CSP deployment.

The image owner will use the WPM to encrypt any image(s) to be protected; the WPM will automatically create a new image encryption key using the KBS, and will output the encrypted image and an Image Flavor. The image owner can then upload the encrypted image to the CSP’s image storage service, and then upload the Image Flavor to the CSP-hosted WLS.
When a compute host at the CSP attempts to launch a protected image, the WLA on the host will detect the launch request, and will issue a key transfer request to the WLS. The WLS will use the image ID to retrieve the Image Flavor, which contains the key retrieval URL for that image. This URL is hosted on the KBS of the image owner (which is why the KBS must be available to network requests from the WLS). The WLS will access the HVS to retrieve the current Platform Integrity Attestation report for the host, and will use this report to make a key transfer request to the KBS at the key transfer URL.

The KBS will receive the request, verify that the Platform Integrity Attestation report is signed using a known SAML signing key (verifying that the report comes from a known and trusted HVS), and will then verify that the report shows that the host is trusted.

If these requirements are met, the KBS will use the host’s Binding Key (the public half of an asymmetric keypair generated by the host’s TPM and included in the attestation report) as a Key Encryption Key to seal the Image Encryption Key to the TPM of the host that was attested.

When the host receives the response to the key request, it will unseal the Image Encryption Key using its TPM. Because the Key Encryption Key is unique to this host’s TPM, only the actual host that was attested will be able to gain access to the image.

With the Image Encryption Key, the host’s WLA will create the appropriate encrypted volume(s) for the image and begin the launch as normal.

The WLA does not retain the key on disk; if/when the host is rebooted or the WLA is restarted, restarting the workloads based on protected images will
trigger new key requests based on new Platform Integrity Attestation reports. In this way, if a host is compromised in a method detectable by the Platform Integrity feature, protected images will be unable to launch on this server.

7.1 Virtual Machine Confidentiality

7.1.1 Prerequisites

To enable Virtual Machine Confidentiality, the following Intel® SecL-DC components must be installed and available:

- Authentication and Authorization Service
- Certificate Management Service
- Key Broker Service
- Host Verification Service
- Workload Service
- Trust Agent + Workload Agent (on each virtualization host)
Workload Policy Manager

See the Installation subsection on Recommended Service Layout for recommendations on how/where to install each service.

It is strongly recommended to use a VM orchestration solution (for example, OpenStack) with the Intel® SecL-DC Integration Hub to schedule encrypted workloads on compute hosts that have already been pre-checked for their Platform Integrity status. See the Platform Integrity Attestation subsection on Integration with OpenStack for an example.

You will need at least one QCOW2-format virtual machine image (for quick testing purposes, a very small minimal premade image like CirrOS is recommended; a good place to look for testing images is the OpenStack Image Guide found here: https://docs.openstack.org/image-guide/obtain-images.html).

One or more hypervisor compute nodes running QEMU/KVM is required. Each of these nodes must have the Intel® SecL-DC Trust Agent and Workload Agent installed, and they must be registered with the Verification Service. Each of these servers should show as “trusted;” see the Platform Integrity Attestation section for details. You should have Flavors that match the system configuration for these hosts, and attestation reports should show all Flavor parts as “trusted=true.” Hosts that are not trusted (including servers where there is no trust status, like hosts with no Trust Agent) will fail to launch any encrypted workloads.

7.1.2 Workflow

7.1.2.1 Encrypting Images

wpm create-image-flavor -l <user-friendly unique label> -i <path to image file> -e <output path and filename for encrypted image> -o <output path for JSON image flavor>

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

7.1.2.2 Uploading the Image Flavor

POST https://<Workload Service IP or Hostname>:5000/wls/flavors
{<Image Flavor content from NPM 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.
7.1.2.3 Creating the Image Flavor to Image ID Association

The WLS needs to know the ID of the image as it exists in the image storage service used by the CSP (for example, OpenStack Glance). Use the below API request to create an association between the Image Flavor created in the previous step and the image ID.

```
POST https://<Workload Service IP or Hostname>:5000/wls/images
{
  "id": "<image ID on image storage>",
  "flavor_ids": ["<Image Flavor ID>"
}
```

7.1.2.4 Launching Encrypted VMs

Instances of the protected images can now be launched as normal. Encrypted images will only be accessible on hosts with a Platform Integrity Attestation report showing the host is trusted.

If the VM is launched on a host that is not trusted, the launch will fail, as the decryption key will not be provided.

7.2 Docker Container Confidentiality

7.2.1 Docker Container Integrity

Intel® recommends using Docker Notary to verify the integrity of Docker container images at launch.

https://docs.docker.com/notary/getting_started/

7.2.2 Prerequisites

To enable Docker Container Confidentiality, the following Intel® SecL-DC components must be installed and available:

- Authentication and Authorization Service
- Certificate Management Service
- Key Broker Service
- Host Verification Service
- Workload Service
- Trust Agent + Workload Agent (on each Docker host)
- Workload Policy Manager

See the Installation subsection on Recommended Service Layout for recommendations on how/where to install each service.

It is strongly recommended to use a container orchestration solution (for example, Kubernetes) with the Intel® SecL-DC Integration Hub to schedule encrypted Docker containers on compute hosts that have already been pre-checked for their Platform Integrity status. See the Platform Integrity Attestation subsection on Integration with Kubernetes for an example.

You will need at least one Docker container image. For quick testing purposes, Intel recommends one or more of the following:

https://github.com/jessfraz/dockerfiles/

Image names:

1. Openvpn
2. k8scan
3. postfix

One or more Docker container worker nodes running Docker 19.03 is required. Each of these nodes must have the Intel® SecL-DC Trust Agent and Workload Agent installed, and they must be registered with the Verification Service. Each of these servers should show as “trusted;” see the Platform Integrity Attestation section for details. You should have Flavors that match the system configuration for these hosts, and attestation reports should show all Flavor parts as “trusted=true.” Hosts that are not trusted (including servers where there is no trust status, like hosts with no Trust Agent) will fail to launch any encrypted workloads.

**Important Note:** Docker version 19.03 is specifically required, and other versions are not supported. Installation of the Workload Agent for Docker Container Confidentiality will replace the existing Docker binaries (the client and daemon, in /usr/bin/dockerd and /usr/bin/docker) with a recompiled Docker engine that includes the Secure Overlay Driver. This is what allows the launch of encrypted containers to be intercepted and decrypted. The Docker runtime must not be upgraded or downgraded to any other version; doing so will cause encrypted Docker Containers to fail to launch.

In the future, the Container Encryption feature will be modified to use OCI-standard container encryption without the need for recompilation or file replacement.
7.2.3 Workflow

7.2.3.1 Encrypting Docker Container Images

The first step is encryption of a Docker Container image. The WPM is a command line utility that will perform the actual image encryption and allow the resulting encrypted image to be uploaded to a Docker Registry.

The commands needed are slightly different depending on whether Notary is being used to validate container integrity.

If Notary is not being used:

```
wpm create-container-image-flavor -i <container image name> -t <tag-name> -e -f <Dockerfile Path> -d <dirPath> -o <output path for JSON image flavor>
```

If Notary is being used:

```
wpm create-container-image-flavor -i <imageName> -t <TagName> -e -s -n https://<notaryIP>:<notaryPort> -f <Dockerfile Path> -d <dirPath>
```

Also, if Notary is being used, set the following environment variable before uploading the image to the Registry:

```
export DOCKER_CONTENT_TRUST=1
```

After generating the encrypted image with the WPM, the encrypted image can be uploaded to a local Docker Registry.

7.2.3.2 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.

7.2.3.3 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:

```
docker images --no-trunc
```

Next, convert this to a UUID that can be used by Intel® SecL:

```
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.

```
POST https://<Workload Service IP or Hostname>:5000/wls/images
{
  "id": "<image ID on image storage>",
  "flavor_ids": ["<Image Flavor ID>"]
}
```

### 7.2.3.4 Launching Encrypted Docker Containers

Containers of the protected images can now be launched as normal using Kubernetes pods and deployments. Encrypted images will only be accessible on hosts with a Platform Integrity Attestation report showing the host is trusted.

If the Docker Container is launched on a host that is not trusted, the launch will fail, as the decryption key will not be provided.
Trusted Virtual Kubernetes Worker Nodes

While the existing Platform Integrity Attestation functions support bare-metal Kubernetes Worker Nodes, using Virtual Machines to host the Worker Nodes is a common deployment architecture. This feature aims to help extend the Chain of Trust to protect the integrity of Virtual Machines, including virtual Kubernetes Worker Nodes. This feature requires the foundational Platform integrity Attestation feature as a prerequisite for the bare-metal servers hosting the virtual Worker Nodes.

**Note:** This feature requires a degree of separation between the VM and Kubernetes infrastructure. All physical, bare-metal servers should be virtualization hosts, and all Kubernetes Worker Nodes should be Virtual Machines running on those physical virtualization hosts. Kubernetes clusters should not use a mixture of both virtual and bare-metal Workers. The physical virtualization clusters should not include a mixture of hosts protected by Intel® SecL Platform integrity Attestation and hosts that are not protected. VM trust reports can only be generated for VM instances launched on hosts with Intel® SecL services enabled.

**Also important to note is that this feature alone will not prevent any VMs from launching.** VMs will still be launched on Untrusted platforms unless additional steps are taken (for example, using OpenStack orchestration integration with Intel® SecL, or using the Workload Confidentiality feature to encrypt the Kubernetes Worker Node VM image). This feature generates VM attestation reports that can be used to audit compliance and extend the Chain of Trust, and relies on other datacenter policies and/or Intel® SecL features to enforce compliance.

When libvirt initiates a VM Start, the Intel® SecL-DC Workload Agent will create a report for the VM that associates the VM’s trust status with the trust status of the host launching the VM. This VM report will be retrievable via the Workload Service, and contains the hardware UUID of the physical server hosting the VM. This UUID can be correlated to the Trust Report of that server at the time of VM launch, creating an audit trail validating that the VM launched on a trusted platform. A new report is created for every VM Start, which includes actions like VM migrations, so that each time a VM is launched or moved a new report is generated ensuring an accurate trust status.
By using Platform Integrity and Data Sovereignty-based orchestration (or Workload Confidentiality with encrypted worker VMs) for the Virtual Machines to ensure that the virtual Kubernetes Worker nodes only launch on trusted hardware, these VM trust reports provide an auditing capability to extend the Chain of Trust to the virtual Worker Nodes.

Optionally, the Kubernetes Worker Node VM images can be encrypted and protected as per the Workload Confidentiality feature of Intel® SecL. This adds a layer of enforcement – rather than simply reporting whether the VM started on a Trusted platform (and is therefore Trusted), Workload Confidentiality ensures that the Worker Node VM image can only be decrypted on compliant platforms.

In both cases (with VM image encryption and without), the VM Trust Reports are accessed through the Workload Service:

GET https://<Workload Service>:5000/wls/reports?instance_id=<instance ID>

This query will return the latest VM trust report for the provided Instance ID (the Instance ID is the VM’s ID as it is identified by Libvirt; in OpenStack this would correspond directly to the OpenStack Instance ID).

As a best practice, Intel® recommends using an orchestration layer (such as OpenStack) integrated with Intel® SecL to launch VMs only on Trusted platforms. See the previous section, “Integration” under the “Platform Integrity Attestation” feature for details.

As an additional layer of protection, the Kubernetes Worker Node VM images can be encrypted using the Workload Confidentiality feature. This adds cryptographic enforcement to the workload orchestration and ensures instances of the Worker Node images will only be launched on Trusted platforms.

8.1 Prerequisites

- All physical, bare-metal servers should be virtualization hosts. Virtualization hosts must be Linux platforms using Libvirt.

- All Kubernetes Worker Nodes should be Virtual Machines running on those physical virtualization hosts.

- Kubernetes clusters must not use a mixture of both virtual and bare-metal Workers.

- The physical virtualization clusters must not include a mixture of hosts protected by Intel® SecL Platform integrity Attestation and hosts that are not protected. VM trust reports can only be generated for VM instances launched on hosts with Intel® SecL services enabled.

- The Intel® SecL Platform integrity Attestation feature must be used to protect all physical virtualization hosts. These platforms must all be
registered with the Verification Service, must have the Trust Agent installed and running, and must be Trusted. See the Platform integrity Attestation section for details.

- In addition to the services required by Platform Integrity Attestation, the Workload Agent must be installed on each physical virtualization host, and the Workload Service must be installed on the management plane.

- (Optional; recommended) Virtual Machines should be orchestrated using an Intel® SecL-supported orchestrator, such as OpenStack. This will help launch the VMs only on compliant platforms.

- (Optional) Virtual Machine Images may be encrypted using the Workload Confidentiality feature. This adds a layer of cryptographic enforcement to the orchestration of virtual worker VMs, ensuring that the VMs can only be launched on compliant platforms.

8.2 Workflow

There are no additional steps required to enable this feature; if the Workload Agent is running on the physical virtualization host, VM trust reports will automatically be generated at every VM Start. Intel® strongly recommends using an orchestration integration for the VM management layer (for example, the provided Integration Hub integration with OpenStack) to help ensure that the worker node VMs only launch on Trusted physical hosts. If no orchestration is used, the platform service provider should ensure that all physical hosts are always in a Trusted state and take action to ensure Untrusted platforms cannot launch VMs.

The primary benefit of the Trusted Virtual Kubernetes Worker Node feature is auditability of the Chain of Trust. By retrieving the VM Trust Report from the Workload Service for a given Worker Node instance, auditors can verify that the VM launched on a Trusted platform. The VM trust report also includes the hardware UUID of the physical host. This UUID, along with the time that the VM instance was launched, can be used to pull the correlating physical host trust report from the Verification Service to provide proof of compliance.

To retrieve a VM trust report from the Workload Service:

GET https://<Workload Service>:5000/wls/reports?instance_id=<instance ID>

This will return the latest report for the specified instance ID.

8.3 Sample VM Trust Report

A sample VM Trust Report from the Workload Service is below. The report is generated by the Workload Agent and signed using the host’s TPM, then stored in the Workload Service. The report contains some key attributes:
**instance_id**: This is the ID of the instance. In OpenStack, this would correlate directly to the Instance ID for the VM.

**image_id**: This is the ID for the source image used to launch the instance. In OpenStack, this correlates directly to the Image ID for the VM.

**host_hardware_uuid**: The hardware UUID of the physical host that started the VM. This attribute identifies which host performed the VM start and attestet the VM. This UUID can be used to query the Verification Service to retrieve attestations of the host. By correlating the VM Trust Report with the Host Trust Report, we can verify that this instance was started on a Trusted platform.

**image_encrypted**: True or False based on whether the source image was protected using the Workload Confidentiality feature.

**trusted**: True or False, based on whether the VM instance was started on a Trusted platform. Because the report is generated at every "vm start" through Libvirt, a new report will be generated whenever the VM is turned on or migrated, reflecting the state of the VM and its host at every opportunity for the state to change.

```xml
<Response xmlns="http://wls.server.com/wls/reports">
  <instance_manifest>
    <instance_info>
      <instance_id>bd06385a-5530-4644-a510-e384b8c3323a</instance_id>
      <host_hardware_uuid>00964993-89c1-e711-906e-00163566263e</host_hardware_uuid>
      <image_id>773e22da-f687-47ca-89e7-5df655c60b7b</image_id>
    </instance_info>
    <image_encrypted>true</image_encrypted>
  </instance_manifest>
  <policy_name>Intel VM Policy</policy_name>
  <results>
    <e>
      <rule>
        <rule_name>EncryptionMatches</rule_name>
        <markers>
          <e>IMAGE</e>
        </markers>
        <expected>
          <name>encryption_required</name>
          <value>true</value>
        </expected>
      </rule>
    </e>
    <flavor_id>3a3e1ccf-2618-4a0d-8426-fb7ac8b7abc</flavor_id>
    <trusted>true</trusted>
  </results>
</Response>
```
9 Flavor Management

9.1 Flavor Format Definitions

A Flavor is a standardized set of expectations that determines what platform measurements will be considered “trusted.” Flavors are constructed in a specific format, containing a metadata section describing the Flavor, and then various other sections depending on the Flavor type or Flavor part.

9.1.1 Meta

The first part of a Flavor is the “meta” section:

```
"meta": {
    "vendor": "INTEL",
    "description": {
        "flavor_part": "PLATFORM",
        "bios_name": "Intel Corporation",
        "bios_version": "SE5C620.86B.00.01.0004.071220170215",
        "tpm_version": "2.0"
    }
}
```

This section defines the Flavor part and any versioning information.

**Note:** Even when the BIOS or OS version remains the same, the actual measurements in the measured boot process will be different between TPM 1.2 and TPM 2.0, and so the TPM version is captured here as well. The attributes in the Meta section are used by the Flavor matching engine when matching Flavors to Hosts.

9.1.2 Hardware

The “hardware” section is unique to PLATFORM flavor parts:

```
"hardware": {
    "processor_info": "54 06 05 00 FF FB EB BF",
    "processor_flags": "fpu vme de ...",
    "feature": {
        "tpm": {
            "enabled": true,
            "pcr_banks": [
                "SHA1",
                "SHA256"
            ]
        }
    }
}
```
This part of the Flavor defines expected hardware attributes of the host, and contains processor and TPM-related attributes.

### 9.1.3 PCRs

The last section of a Flavor is the “PCRs” section, which contains the actual expected measurements for any PCRs. This section will contain PCR measurements for each applicable algorithm supported by the TPM (SHA1 only for TPM 1.2, SHA256 and SHA1 sections for TPM 2.0).

Some PCRs simply have a value and nothing else. Other PCRs, however, contain different “event” measurements. This indicates that separate individual platform or OS components are independently measured and extended to the same PCR. PCRs with event measurements will contain an “Event” array that lists, in the correct order, all of the events in the measurement event log that are extended to this PCR. When the Verification Service attests a host against a given Flavor, each measurement event is compared to the Flavor value, and all of the events are replayed to confirm that a replay of all of the measurement extensions do in fact result in the hash seen in the PCR value. In this way, the Verification Service can ensure that the measurement event log contents are secure, and the individual measurements can be attested so that the cause for an Untrusted attestation can easily be seen.

The full PCRs section is not shown here due to length; see the sample Flavor sections for a full sample.
9.1.4 Sample PLATFORM Flavor

The PLATFORM Flavor part encompasses measurements that are unique to a specific platform, including the server OEM, BIOS version, etc. A PLATFORM Flavor can be "shared" across all hosts of the same model that have the same BIOS version.

```json
{
  "flavor_collection": {
    "flavors": [
      "meta": {
        "vendor": "INTEL",
        "description": {
          "flavor_part": "PLATFORM",
          "bios_name": "Intel Corporation",
          "bios_version": "SE5C620.86B.00.01.0004.071220170215",
          "tpm_version": "2.0"
        }
      },
      "hardware": {
        "processor_info": "54 06 05 00 FF FB EB BF",
        "processor_flags": "fpu vme de _",
        "feature": {
          "tpm": {
            "enabled": true,
            "pcr_banks": [
              "SHA1",
              "SHA256"
            ]
          },
          "txt": {
            "enabled": true
          }
        }
      },
      "pcrs": {
        "SHA1": {
          "pcr_0": {"value": "d2ed125942726641a7260c4f92beb67d531a0def"},
          "pcr_17": {
            "value": "1ec12004b371e3af4d43d04155abde7476a3794fa",
            "event": {
              "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
              "value": "2fb7d57dcc5455af9ac08d82bdf315dbc59a044",
              "label": "HASH_START",
              "info": {
                "ComponentName": "HASH_START",
                "EventName": "OpenSource.EventName"
              }
            }
          }
        }
      }
    ]
  }
}
```
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "ffb1806465d2e1b7531fd5a2a6eafaa7c5a047",
"label": "BIOSAC_REG_DATA",
"info": {
    "ComponentName": "BIOSAC_REG_DATA",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973731f83e21f9392d2fc",
"label": "CPU_SCRTM_STAT",
"info": {
    "ComponentName": "CPU_SCRTM_STAT",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "9069ca78e7450a285173431b3e52c5c25299e473",
"label": "LCP_CONTROL_HASH",
"info": {
    "ComponentName": "LCP_CONTROL_HASH",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "5ba93c9db0cff93f52b521d7420e43f6eda2784f",
"label": "LCP_DETAILS_HASH",
"info": {
    "ComponentName": "LCP_DETAILS_HASH",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "5ba93c9db0cff93f52b521d7420e43f6eda2784f",
"label": "STM_HASH",
"info": {
    "ComponentName": "STM_HASH",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973731f83e21f9392d2fc",
"label": "OSSINITDATA_CAP_HASH",
"info": {
    "ComponentName": "OSSINITDATA_CAP_HASH",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973731f83e21f9392d2fc",
"label": "OSSINITDATA_CAP_HASH",
"info": {
    "ComponentName": "OSSINITDATA_CAP_HASH",
    "EventName": "OpenSource.EventName"
}
},

"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973731f83e21f9392d2fc",
"label": "OSSINITDATA_CAP_HASH",
"info": {
    "ComponentName": "OSSINITDATA_CAP_HASH",
    "EventName": "OpenSource.EventName"
}
}


9.1.5 Sample OS Flavor

An OS Flavor encompasses all of the measurements unique to a given OS. This includes the OS kernel and other measurements.

```json
{
    "flavor_collection": {
        "flavors": [{
            "meta": {
                "vendor": "INTEL",
                "description": {
                    "flavor_part": "OS",
                    "os_name": "RedHatEnterpriseServer"
                }
            }
        }
    }
}
```
"os_version": "7.3",
"vmm_name": "",
"vmm_version": "",
"tpm_version": "2.0"
},
"pcrs": {
"SHA1": {
"pcr_17": {
"value": "1ec12004b371e3af4d43d04155abde7476a3794fa",
"event": ["event": {
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "2fb7d57d5c5f5af9a0c8d82bdf315dbc59a044",
"label": "HASH_START",
"info": {
"ComponentName": "HASH_START",
"EventName": "OpenSource.EventName"
}
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "ffb1806465d2de1b7531fd5a2a6e4a07c5a047",
"label": "BIOSAC_REG_DATA",
"info": {
"ComponentName": "BIOSAC_REG_DATA",
"EventName": "OpenSource.EventName"
}
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973731f83e21fab9392d2fc",
"label": "CPU_SCRTM_STAT",
"info": {
"ComponentName": "CPU_SCRTM_STAT",
"EventName": "OpenSource.EventName"
}
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "9069ca78e7450a2852c55973431b33e21fab9392d2fc",
"label": "LCP_CONTROL_HASH",
"info": {
"ComponentName": "LCP_CONTROL_HASH",
"EventName": "OpenSource.EventName"
}
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "5ba93c9db0c6f93f52b521d7420e43f6eda2784f",
"label": "LCP_DETAILS_HASH",
"info":                   {
  "ComponentName": "LCP_DETAILS_HASH",
  "EventName": "OpenSource.EventName"
}
}

{
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha1",
  "value": "5ba93c9db0cffe93f52b521d7420e43f6eda2784f",
  "label": "STM_HASH",
  "info":                   {
    "ComponentName": "STM_HASH",
    "EventName": "OpenSource.EventName"
  }
}

{
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha1",
  "value": "3c585604e87f855973731feaa83e21fab9392d2fc",
  "label": "OSSINITDATA_CAP_HASH",
  "info":                   {
    "ComponentName": "OSSINITDATA_CAP_HASH",
    "EventName": "OpenSource.EventName"
  }
}

{
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha1",
  "value": "3d42560dcf165a5557b3156a21583f2c6dbef10e",
  "label": "MLE_HASH",
  "info":                   {
    "ComponentName": "MLE_HASH",
    "EventName": "OpenSource.EventName"
  }
}

{
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha1",
  "value": "274f929dabdb8b98a7031bbcd9ea5613c2a8e5e6",
  "label": "NV_INFO_HASH",
  "info":                   {
    "ComponentName": "NV_INFO_HASH",
    "EventName": "OpenSource.EventName"
  }
}

{
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha1",
  "value": "ca96de412b4e8c062a570d3013d2fccb4b20250a",
  "label": "tb_policy",
  "info":                   {
    "ComponentName": "tb_policy",
    "EventName": "OpenSource.EventName"
  }
}
}

{
    "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
    "value": "d123e2f2b30f1effa8d9522f667af0dac4f48cfb",
    "label": "vmlinuz",
    "info": {
        "ComponentName": "vmlinuz",
        "EventName": "OpenSource.EventName"
    }
},
{
    "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
    "value": "f3742133e1a0deb48177a74ed225418e5cf73fd1",
    "label": "initrd",
    "info": {
        "ComponentName": "initrd",
        "EventName": "OpenSource.EventName"
    }
}
],
"SHA256": {
    "pcr_17": {
        "value": "50bd58407a1893056a5ff493245cfe785f045b2c0e1cc3e6e9eb5812d8d91bd",
        "event": [
            {
                "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
                "value": "9301981c093654d5aa3430ba05c880a52eb22b9e18248f5f93e1f6d1acbb947",
                "label": "HASH_START",
                "info": {
                    "ComponentName": "HASH_START",
                    "EventName": "OpenSource.EventName"
                }
            },
            {
                "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
                "value": "67abdd721024f0ff4e0b3f4c2fc13bc5bad42d0b785d456d88d203d15aa450",
                "label": "CPU_SCRTM_STAT",
                "info": {

```
"ComponentName": "CPU_SCRTM_STAT",
"EventName": "OpenSource.EventName"
},

{"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"df3f619804a92fda04057192dc43dd748ea778adc52bc49865ce8050ac014b812119",
"label": "LCP_CONTROL_HASH",
"info":
{ "ComponentName": "LCP_CONTROL_HASH",
"EventName": "OpenSource.EventName"
}
},

{"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"6e340b9c0bb37a989ca544e6bb780a2c78901d3f93b33738768511a30617afa01d",
"label": "LCPDETAILS_HASH",
"info":
{ "ComponentName": "LCPDETAILS_HASH",
"EventName": "OpenSource.EventName"
}
},

{"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"6e340b9c0bb37a989ca544e6bb780a2c78901d3f93b33738768511a30617afa01d",
"label": "STM_HASH",
"info":
{ "ComponentName": "STM_HASH",
"EventName": "OpenSource.EventName"
}
},

{"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"67abdd721024f0ff4e0b3f4c2fc13bc5bad42d0b7851d4568d88d203d15aaa450",
"label": "OSSINITDATA_CAP_HASH",
"info":
{ "ComponentName": "OSSINITDATA_CAP_HASH",
"EventName": "OpenSource.EventName"
}
},

{"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"26e1d98742f79c950dc637f8c067b072a1b0e8ff75db4e609c7e7321acf3f4",
"label": "MLE_HASH",
"info":
{
"ComponentName": "MLE_HASH",
    "EventName": "OpenSource.EventName"
  }
},
  
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha256",
    "value":
  "0f6e0c7a5944963d7081ea494ddff1e9afa689e148e39f684db06578869ea38b",
    "label": "NV_INFO_HASH",
    "info":
  
  "ComponentName": "NV_INFO_HASH",
    "EventName": "OpenSource.EventName"
  }
},
  
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha256",
    "value":
  "27808f64e6383982cd3b6c10cfch3457c0b65f465f779d89b668839eaf263a67",
    "label": "tb_policy",
    "info":
  
  "ComponentName": "tb_policy",
    "EventName": "OpenSource.EventName"
  }
},
  
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha256",
    "value":
  "c89ad1d1e9adaa7ecfee2abc376b92472685f7d1b9f3799b49974b66ed9638",
    "label": "vmlinuz",
    "info":
  
  "ComponentName": "vmlinuz",
    "EventName": "OpenSource.EventName"
  }
},
  
  "digest_type":
  "com.intel.mtwilson.lib.common.model.MeasurementSha256",
    "value":
  "81b88e268e697ccf1790d41b9de748a8f395acfb47aa67c9845479d4e8456f77",
    "label": "initrd",
    "info":
  
  "ComponentName": "initrd",
    "EventName": "OpenSource.EventName"
  }
  
  
  "flavorgroup_name": "mtwilson_automatic"
9.1.6 Sample HOST_UNIQUE Flavor

Host-Unique flavors define measurements for a specific host. This can be either a single large flavor that incorporates all of the host measurements into a single flavor document used only to attest a single host, or can be a small subset of measurements that are specific to a single host. For example, some VMWare module measurements will change from one host to the next, while most others will be shared assuming the same ESXi build is used. The full Flavor requirement for such a host would include Host-Unique flavors to cover the measurements that are unique to only this one host, and would still use a generic PLATFORM and OS flavor for the other measurements that would be identical for other similarly configured hosts.

Note: The HOST_UNIQUE Flavors are unique to a specific host, and should always be imported directly from the specific host. Windows hosts do not require a HOST_UNIQUE flavor part.

```json
{"flavors": [ {
    "meta": {
        "id": "4d387cbd-f72b-4742-b4e5-c5b0ffed59e0",
        "vendor": "INTEL",
        "description": {
            "flavor_part": "HOST_UNIQUE",
            "source": "Purley11",
            "bios_name": "Intel Corporation",
            "bios_version": "SE5C620.86B.00.01.0004.071220170215",
            "os_name": "RedHatEnterpriseServer",
            "os_version": "7.4",
            "tpm_version": "2.0",
            "hardware_uuid": "00448C61-46F2-E711-906E-001560A04062"
        }
    },
    "pcrs": {
        "SHA256": {
            "pcr_17": {
                "value": "f9ef8c53ddf8096d36eda5506436c52b4bfa2bd451a89aa102f0318722176",
                "event": {
                    "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
                    "value": "df3f619804a92f6b4057192dc43d748ea778adc52bc498ce80524c014b81119",
                    "label": "LCP_CONTROL_HASH",
                    "info": {
                        "ComponentName": "LCP_CONTROL_HASH",
                        "EventName": "OpenSource.EventName"
                    }
                }
            }
        }
    }
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value": "f9ef8c53ddf8096d36eda5506436c52b4bfa2bd451a89aa102f0318722176"
]
```
"value": "09f468dfc1d98a1fee86eb7297a56b0e097d57be66db4eae539061332da2e723",
"label": "initrd",
"info":
  "ComponentName": "initrd",
  "EventName": "OpenSource.EventName"
}]
},
"pcr_18": {
  "value": "c1f7bfdae5f270d9f13aa9620b8977951d6b759f1131fe9f9289317f3a56efa1",
  "event": [
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
      "value": "df3f619804a92f4d4057192dc43dd748ea778ad52bc498ce86524c014b81119",
      "label": "LCP_CONTROL_HASH",
      "info":
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
    }
  ]
},
"SHA1": {
  "pcr_17": {
    "value": "48695f747a3d494710bd14d20cb0a93c78a485cc",
    "event": [
      {
        "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
        "value": "9069ca78e7450a285173431b3e52c5c25299e473",
        "label": "LCP_CONTROL_HASH",
        "info":
          "ComponentName": "LCP_CONTROL_HASH",
          "EventName": "OpenSource.EventName"
      },
      {
        "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
        "value": "b1f8db372e396bb12b8280821b7e0ac54a5e2791",
        "label": "initrd",
        "info":
          "ComponentName": "initrd",
          "EventName": "OpenSource.EventName"
      }
    ]
  },
  "pcr_18": {
    "value": "983ec7db975ed31e2c85ef8e375c038d6d307efb",
    "event": [
      }"
9.1.7 Sample ASSET_TAG Flavor

Asset Tag flavor parts are unique to Asset Tag attestation. These flavors verify that the Asset Tag data in the host's TPM correctly matches the most recently created, currently valid Asset Tag certificate that has been deployed to that host.

```json
{
    "meta": {
        "id": "b3e0c056-5b6c-4b6b-95c4-de5f1473cac0",
        "description": {
            "flavor_part": "ASSET_TAG",
            "hardware_uuid": "<Hardware UUID of the server to be tagged>"
        }
    },
    "external": {
        "asset_tag": {
            "tag_certificate": {
                "encoded": "<Tag certificate in base64 encoded format>",
                "issuer": "CN=assetTagService",
                "serial_number": 1519153544161,
                "subject": "<Hardware UUID of the server to be tagged>",
                "not_before": "2018-02-20T11:05:41-0800",
                "not_after": "2019-02-20T11:05:41-0800",
                "fingerprint_sha256": "2YjKiugQN4kgQ2Np34vydzXurfBBpTMfweF1pY1U1zc=",
                "attribute": [
                    {
                        "attr_type": {"id": "2.5.4.789.2"},
                        "attribute_values": [{"objects": {}}]
                    },
                    {
                        "attr_type": {"id": "2.5.4.789.2"},
                        "attribute_values": [{"objects": {}}]
                    },
                    {
                        "attr_type": {"id": "2.5.4.789.2"},
                        "attribute_values": [{"objects": {}}]
                    }
                ]
            }
        }
    }
}
```
9.2 Flavor Matching

Flavors are matched to host by objects called "Flavor Groups." A Flavor Group represents a set of rules to satisfy for a set of flavors to be matched to a host for attestation. For example, a Flavor Group can require that a PLATFORM Flavor and an OS Flavor be used for attestation. Without this level of association, a host that matches measurements for only a PLATFORM flavor, for example, can be attested as Trusted, even though the OS Flavor would attest the host as Untrusted.

Flavor matching can be automatic (the default), or can explicitly specify a host to which the Flavor Group must apply.

Automatic flavor matching allows for more ease in datacenter lifecycle management with updates and patches that may cause the appropriate flavors to change over time. Automatic flavor matching will trigger a new matching action when a new flavor is added, when an existing flavor is deleted, or when a host is initially attested as Untrusted. The system will automatically attempt to find a new set of flavors that match the Flavor Group rules that will attest the host as Trusted. For example, if a host in your datacenter has recently had a BIOS update, the next attestation will cause the host to appear Untrusted (because the PLATFORM measurements will now differ). Using automatic flavor matching, the Verification Service will automatically search for a new PLATFORM flavor that matches the actual BIOS version and measurement seen on the host. If a new BIOS version is successfully found, the Verification Service will use the new version for attestation, and the host will appear Trusted. If no matching PLATFORM flavor is found, the host will appear Untrusted. When automatic flavor matching is used, think of the various flavors in the Verification Service as a collection of valid configurations, and an attested host matching any combination of those configurations (within the confines of the Flavor Group requirements for which flavor types must be present) will be attested as Trusted.

Host-based flavor matching explicitly maps a specific host to a flavor. Host-based attestation requires that a host saves its entire configuration in a composite flavor document in the system, and then later validates against this flavor to detect any changes. In this case, if a host received a BIOS upgrade, the host will attest as Untrusted, and no attempt will be made to re-match a new flavor. An administrator will need to explicitly specify a new flavor to be used for that host.

9.2.1 When Does Flavor Matching Happen?

Generally speaking, a new Flavor match operation is triggered whenever a host is registered, whenever a host is attested and would be untrusted, and whenever a Flavor is added to or removed from a Flavor group.
When a new host is registered, the Verification Service will retrieve the Host Report and derive the platform information needed for Flavor matching (BIOS version, server OEM, OS type and version, TPM version, etc.). The Verification Service then searches through the Flavors in the same Flavor group that the host is in, and finds any Flavors that match the platform information.

If a Flavor is deleted, the Verification Service finds any hosts that are currently associated with that Flavor, and attempts to match them to alternative Flavors.

If a Flavor is added, the Verification Service looks for any hosts in the same Flavor group that are not currently matched to a Flavor of the appropriate Flavor part, and checks to see whether those hosts should be mapped to the new Flavor.

If a new Report is generated for a host and would not result in a Trusted attestation, the Verification Service will first repeat the Flavor matching process to be sure that no matching Flavors exist in the host's Flavor group that would result in a Trusted attestation. If the Service still finds no matching Flavors, the host will appear as Untrusted.

### 9.2.2 Flavor Matching Performance

Flavor matching causes affected hosts to be moved into the “QUEUE” state while the host and Flavor are evaluated to determine whether the host and Flavor should be linked. Hosts can remain in the QUEUE state for varying amounts of time based on the extent of the Flavor match required. This means that the trust status of a host will not be actually updated to reflect a new Flavor until after the process finishes, which may take a few seconds or minutes depending on the number of registered hosts, Flavors in the same Flavorgroup, etc.

If a new host is registered, only that host will be added to the queue, and other hosts will be unaffected. The Verification Service will look for only the HOST_UNIQUE flavor part applicable to that specific host, and then will look at all PLATFORM and OS Flavors in the same Flavorgroup has the host, using the Flavor metadata and host info to narrow the results. The Service will match the new host to the most similar Flavors, and then move the host to the “CONNECTED” state and generate a new trust report.

When a new PLATFORM or OS Flavor is created, the Service will instead add all hosts in the same Flavorgroup as the new Flavors to the queue. Each host in the queue will then be re-evaluated against every PLATFORM and OS Flavor in the Flavorgroup to determine the closest match.

This means that adding a new Flavor can cause more hosts to each spend more time in the QUEUE state, as compared to adding a new host. For this reason, as a best practice for initial population of Flavors and hosts for a new deployment, it is suggested that Flavors be created before registering hosts. This is not a concern after the initial population of Flavors and hosts.
9.2.3 **Flavor Groups**

Flavor Groups represent a collection of one or more Flavors that are possible matches for a collection of one or more hosts. Flavor Groups link to both Flavors and hosts – a host in Flavor Group “ABC” will only be matched to Flavors in Flavor Group “ABC.”

9.2.4 **Default Flavor Group**

By default the Verification Service includes a Flavor Group named “automatic” and another named “unique.” During host registration, the “automatic” Flavor Group is used as a default selection if no other Flavor Group is specified.

9.2.4.1 **automatic**

The automatic Flavor Group is used as the default Flavor Group for all hosts and all Flavor parts. If no other Flavor Groups are specified when creating Flavors or Hosts, all Hosts and Flavors will be added to this group. This is useful for datacenters that want to manage a single set of acceptable configurations for all hosts.

9.2.4.2 **unique**

The unique Flavor Group is used to contain HOST_UNIQUE Flavors. This Flavor Group is used by the backend software and should not be managed manually.

9.2.5 **Flavor Match Policies**

Flavor Match Policies are used to define how the Flavor Match engine will match Flavors to hosts for attestation for a given Flavor Group. Each Flavor part can have defined Flavor Match Policies within a given Flavor Group.

i.e.,

```
"PLATFORM": { "any_of", "required" }
"OS": { "all_of", "required_if_defined" }
"HOST_UNIQUE": {"latest","required_if_defined"}
"ASSET_TAG":{"latest","required_if_defined"}
"SOFTWARE": {"all_of","required_if_defined"}
```

The sample Policy above would require that a PLATFORM Flavor part be matched, but any PLATFORM Flavor part in the Flavor Group may be matched. The OS Flavor Part will only be required if there is an OS Flavor part in the Flavor Group; if there are no OS Flavor parts in the Group, the match will not be required. If more than one OS Flavor part exists in the Group, all of those OS parts will be required to match for a host to be Trusted.
9.2.5.1 Default Flavor Match Policy

The “automatic” Flavor Group, and any Flavor Group created without explicitly defining a Flavor Match Policy, will be created using the following Flavor Match Policy. This is the default behavior for Flavor Matching:

```
"PLATFORM": { "any_of", "required" }
"OS": { "any_of", "required" }
"HOST_UNIQUE": {"latest","required_if_defined"}
"ASSET_TAG":{"latest","required_if_defined"}
"SOFTWARE": {"all_of","required_if_defined"}
```

9.2.5.2 ANY_OF

The ANY_OF Policy allows any Flavor of the specified Flavor part to be matched. If the Flavor Group contains OS Flavor 1 and OS Flavor 2, a host will be Trusted if it matches either OS Flavor 1 or OS Flavor 2.

9.2.5.3 ALL_OF

The ALL_OF Policy requires all Flavors of the specified Flavor Part in the Flavor Group to be matched. For example, if Flavor Group X contains PLATFORM Flavor Part 1 and PLATFORM Flavor Part 2, a host in Flavor Group X will need to match both PLATFORM Flavor 1 and PLATFORM Flavor 2 to attest as Trusted. If the host matches only one of the Flavors, or neither of them, the host will be attested as Untrusted.

9.2.5.4 LATEST

The LATEST Policy requires that the most recently created Flavor of the specified Flavor part be used when matching to a host. For example:

```
"ASSET_TAG": { "latest", "required_if_defined" }
```

ASSET_TAG Flavor parts by default use the above Policy. This means that if Asset Tag Flavors are in the Flavor Group, the most recently created Asset Tag Flavor will be used. If no Asset Tag Flavors are present in the Flavor Group, then this Flavor part will be ignored.
9.2.5.5  **REQUIRED**

The REQUIRED Policy requires a Flavor of the specified part to be matched. For example:

```
"PLATFORM": { "any_of", "required" }
```

This policy means that a PLATFORM Flavor part must be used; if the Flavor Group contains no PLATFORM Flavor parts, hosts in this Flavor Group will always count as Untrusted.

9.2.5.6  **REQUIRED_IF_DEFINED**

The REQUIRED_IF_DEFINED Policy requires that a Flavor part be used if a Flavor of that part exists. If no Flavor part of this type exists in the Flavor Group, the Flavor part will not be required.

```
"ASSET_TAG": { "latest", "required_if_defined" }
```

ASSET_TAG Flavor parts by default use the above Policy. This means that if Asset Tag Flavors are in the Flavor Group, the most recently created Asset Tag Flavor will be used. If no Asset Tag Flavors are present in the Flavor Group, then this Flavor part will be ignored.

9.2.6  **Flavor Match Event Triggers**

Several events will cause the background queue service to attempt to re-match Flavors and hosts:

1. **Host registration**
   
   This event is the first time a host will be attempted to be matched to appropriate Flavors in the same Flavor Group, and affects only the host that was added (other hosts will not be re-matched to Flavors when you add a new host).

2. **Flavor creation**
   
   When a new Flavor is added to a Flavor Group, the queue system will repeat the Flavor match operation for all hosts in the same Flavor Group as the new Flavor.

3. **Flavor deletion**
   
   When a Flavor is deleted, the queue system will repeat the Flavor match operation for all hosts in the same Flavor Group as the deleted Flavor.

4. **Creation of a new Attestation Report**
   
   When a new Attestation Report is generated, if the host would attest as Untrusted with the currently-matched Flavors, the host being attested will be re-matched as part of the Report generation process. This ensures that Reports are always generated using the best possible Flavor matches available in the database.
9.2.7 Sample Flavorgroup API Calls

9.2.7.1 Create a New Flavorgroup

POST https://server.com:8443/mtwilson/v2/flavorgroups
Input:
{
   "flavorgroup_name":"firstTest",
   "flavor_match_policy_collection":{
       "flavor_match_policies": [
       {
         "flavor_part": "PLATFORM",
         "match_policy": {
            "match_type": "ANY_OF",
            "required": "REQUIRED"
         }
       }
     ]
   }
}

Output:
"id": "a0950923-596b-41f7-b9ad-09f525929ba1",
"flavorgroup_name": "firstTest",
"flavor_match_policy_collection":{
   "flavor_match_policies": [
   {
     "flavor_part": "PLATFORM",
     "match_policy":{
       "match_type": "ANY_OF",
       "required": "REQUIRED"
     }
   }
  ]
}

9.3 SOFTWARE Flavor Management

9.3.1 What is a SOFTWARE Flavor?

A SOFTWARE Flavor part defines the measurements expected for a specific application, or a specific set of files and folders on the physical host. SOFTWARE Flavors can be used to attest the boot-time integrity of any static files or folders on a physical server.

A single server can have multiple SOFTWARE Flavors associated. Intel® SecL-DC provides a “default” SOFTWARE Flavor that is deployed to each Trust
Agent server during the provisioning step. This default Flavor includes the static files and folders of the Trust Agent itself, so that the Trust Agent is measured during the server boot process, and its integrity is included in the attestation of the other server measurements.

Using SOFTWARE Flavors consists of two parts – creating the actual SOFTWARE Flavor, and deploying the SOFTWARE Flavor manifest to the host.
9.3.2 Creating a SOFTWARE Flavor part

Creating a new SOFTWARE Flavor requires creating a manifest of the files and folders that need to be measured.

There are three different types of entries for the manifest: Directories, Symlinks, and Files.

9.3.2.1 Directories

A Directory defines measurement rules for measuring a directory. Effectively this involves listing the contents of the directory and hashing the results; in this way, a Directory measurement can verify that no files have been added or removed from the directory specified, but will not measure the integrity of individual files (i.e., files can change within the directory, but cannot be renamed, added, or removed).

Directory entries can use regular expressions to define explicit Include and Exclude filters. For example, “Exclude=*.log” would exclude all files ending with .log from the measurement, meaning files with the .log extension can be added or removed from the directory.

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

9.3.2.2 Symlinks

A Symlink entry defines a symbolic link that will be measured. The actual symbolic link is hashed, not the file or folder the symlink points to. In this way, the measurement will detect the symbolic link being modified to point to a different location, but the actual file or folder pointed to can have its contents change.

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

9.3.2.3 Files

Individual files can be explicitly specified for measurement as well. Each file listed will be hashed and extended separately. This means that if any file explicitly listed this way changes its contents or is deleted or moved, the measurement will change, and the host will become Untrusted.

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

9.3.3 Sample SOFTWARE Flavor Creation Call

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 Connection String must point to the sample Trust Agent host. The Label defines the name of the new Flavor (ideally this should be the name of the application being measured for easier management).

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="Tomcat" 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>

9.3.4 Deploying a SOFTWARE Flavor Manifest to a Host

Once the SOFTWARE Flavor has been created, it can be deployed to any number of Trust Agent servers. This requires the Flavor ID (returned from Flavor creation) and the Host ID (returned from host registration). The Verification Service will send a request to the appropriate Trust Agent and create the manifest.

**Note:** After the SOFTWARE Flavor manifest is deployed to a host, the host must be rebooted. This will allow the measurements specified in the Flavor to be taken and extended to the TPM. Until the host is rebooted, the host will now appear Untrusted, as it now requires measurements from a SOFTWARE Flavor that have not yet been extended to the TPM.

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

Input:
{
  "flavor_id": "a6544ff4-6dc7-4c74-82be-578592e7e3ba",
  "host_id": "a6544ff4-6dc7-4c74-82be-578592e7e3ba"
}
9.3.5 SOFTWARE Flavor Matching

The default Flavor Match Policy for SOFTWARE Flavor parts is "ALL_OF","REQUIRED_IF_DEFINED". This means that all Software Flavors defined in a Flavorgroup must match to all hosts in that Flavorgroup. If no SOFTWARE Flavors are in the Flavorgroup, then hosts can still be considered Trusted.

Because the default uses the “ALL_OF” Policy, it’s recommended to use Flavorgroups dedicated to specific software loadouts. For example, if a number of hosts will act as virtualization hosts and will have SOFTWARE Flavors for the hypervisor and VM management applications, those hosts should be placed in their own Flavorgroup as they will all run similar or identical application loadouts. If another group of servers in the datacenter will act as container hosts, these hosts might need SOFTWARE Flavors that include attestation of container runtimes and management applications, and will have a very different application loadout from the VM-based hosts. These should be placed in their own Flavorgroup, so that the VM hosts are attested using the hypervisor-related SOFTWARE Flavors, and the container hosts are attested using the container-related SOFTWARE Flavors.

As with other Flavor parts, hosts will be matched to Flavors in the same Flavorgroup that the host is added to, and will not be matched to Flavors in different Flavorgroups. Flavor matching will happen on the same events as for other Flavor parts.

9.3.6 Kernel Upgrades

Because the Application Integrity functionality involves adding a measurement agent (tbootXM) to initrd, an additional process must be followed when updating the OS kernel to ensure the new initrd also contains the measurement agent. This is not required if Application Integrity will not be used.

1) Update grub to have the boot menu-entry created for the new kernel version in grub.cfg
   (grub2-mkconfig -o <path to grub file>)
2) Reboot the host and boot into new kernel menu-entry.
3) Generate a new initrd with tbootXM. (/opt/tbootxm/bin/generate_initrd.sh)
4) Copy the generated initrd to the boot directory. (cp /var/tbootxm/<generated initrd file name> /boot/)
5) Update the “TCB protection” menu-entry with the new kernel version.
   a) Source trustagent.env, or
      export GRUB_FILE=/boot/efi/EFI/redhat/grub.cfg
   b) Run the configure_host script:
      cd /opt/tbootxm/bin
      ./configure_host.sh
6) Update the default boot menu-entry to have new kernel version. (edit /etc/default/grub)
7) Update the grub to reflect the updates. (grub2-mkconfig -o <path to grub file>)
8) Reboot the host and boot into TCB protection menu-entry.

After updating the system with the new initrd, the Software Flavor should attest as Trusted. Note that changing grub and initrd does result in a new OS Flavor measurements, so an updated OS Flavor should be imported after updating the kernel and regenerating initrd.
10  Scalability and Sizing

10.1  Configuration Maximums

10.1.1  Registered Hosts

The Intel® SecL Verification Service can support a maximum of 2000 registered hosts with a single Verification Service instance with default settings.

10.1.2  HDD Space

The HDD space recommendations below represent expected log and database growth using default settings. Altering the database or log rotation settings, or the SAML expiration setting, may change the amount of disk space required. For default settings, 100 GB of disk space is recommended.

10.2  Database Rotation Settings

The Intel® SecL Verification Service database will automatically rotate the audit log table after one million records, and will retain up to ten total rotations. These settings are user-configurable if a longer retention period is needed.

```
mtwilson.audit.log.num.rotations
```
- defines the maximum number of rotations before the oldest rotation is deleted to make space for a new rotation.

```
mtwilson.audit.log.max.row.count
```
- defines the maximum number of rows in the audit log table before a rotation will occur.

10.3  Log Rotation

The Intel® SecL services (the Verification Service, Trust Agent, and Integration Hub) use Logrotate to rotate logs automatically during a daily cron job.

By default, logs are rotated once per month or when they exceed 1 GB in size, whichever comes first, and 12 total rotations will be retained.
# Intel Security Libraries
## Configuration Settings

### 11.1 Verification Service

#### 11.1.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTWILSON_SERVER</td>
<td>192.168.1.1</td>
<td>Hostname or IP address of the Verification Service</td>
</tr>
<tr>
<td>MTWILSON_API_BASEURL</td>
<td>https://&amp;{MTWILSON_SERVER}:8443/mtwilson/v1</td>
<td>v1 baseurl for the Verification Service. Generally this should not be changed.</td>
</tr>
<tr>
<td>MC_FIRST_USERNAME</td>
<td>administrator</td>
<td>Username for a new administrator-level user to be created during installation.</td>
</tr>
<tr>
<td>MC_FIRST_PASSWORD</td>
<td>mypassword123</td>
<td>Password for the new administrator-level user to be created during installation.</td>
</tr>
<tr>
<td>INSTALL_PKGS</td>
<td>&quot;logrotate&quot;</td>
<td>Defines the optional Verification Service components that will be installed. logrotate: the installer will automatically install and configure logrotate. This is optional.</td>
</tr>
<tr>
<td>LOG_SIZE</td>
<td>50M</td>
<td>Defines the log rotation size threshold for the Verification Service log. This is required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_OLD</td>
<td>3</td>
<td>Defines the number of rotated logs to be retained by logrotate before deletion. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_ROTATION_PERIOD</td>
<td>daily</td>
<td>Defines time interval. Log files are rotated when they grow bigger than size bytes, but not before the additionally specified time interval. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_COMPRESS</td>
<td>compress</td>
<td>Defines to compress log files. Old versions of log files are compressed with gzip(1) by default. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_DELAYCOMPRESS</td>
<td>delaycompress</td>
<td>Defines to postpone compression of the previous log file to the next rotation cycle. This is used with compress. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_COPYTRUNCATE</td>
<td>copytruncate</td>
<td>Defines to truncate the original log file in place after creating a copy, instead of moving the old log file and optionally creating a new one. Required if logrotate is installed.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MTWILSON_TLS_POLICY.Allow</td>
<td>certificate,certificate-digest,public-key,public-key-digest,TRUST_FIRST_CERTIFICATE,</td>
<td>Defines the TLS policies that will be allowed by the Verification Services. Policies not included in this list will not be created. See the TLS Policy Management section for details on TLS Policy types.</td>
</tr>
<tr>
<td>MTWILSON_DEFAULT_TLS_POLICY_ID</td>
<td>TRUST_FIRST_CERTIFICATE</td>
<td>Defines the default TLS policy to be used if no TLS policy is specified. Note that only TRUST_FIRST_CERTIFICATE can be specified here, because no other TLS policies exist at the time of installation. This setting is optional; if not used, there will be no default TLS Policy, and all API calls that require a TLS Policy will require the desired TLS Policy to be explicitly specified in the call.</td>
</tr>
<tr>
<td>DATABASE_HOSTNAME</td>
<td>127.0.0.1</td>
<td>Defines the database server IP address or hostname. This should be the loopback address for local database server installations, but should be the IP address or hostname of the database server if a remote database will be used.</td>
</tr>
<tr>
<td>DATABASE_PORTNUM</td>
<td>5432</td>
<td>Defines the port number for communication with the database server. By default with a local database server installation, this port will be set to 5432.</td>
</tr>
<tr>
<td>DATABASE_SCHEMA</td>
<td>mw_as</td>
<td>Defines the schema name of the database. If a remote database connection will be used, this schema must be created in the remote database before installing the Verification Service.</td>
</tr>
<tr>
<td>DATABASE_USERNAME</td>
<td>root</td>
<td>Username for accessing the database. If a remote database connection will be used, this user/password must be created and granted all permissions for the database schema before installing the Verification Service.</td>
</tr>
<tr>
<td>DATABASE_PASSWORD</td>
<td>dbpassword</td>
<td>Password for accessing the database. If a remote database connection will be used, this user/password must be created and granted all permissions for the database schema before installing the Verification Service.</td>
</tr>
<tr>
<td>MTWILSON_AUDIT_LOG_MAX_ROW_COUNT</td>
<td>1000000</td>
<td>Optional; uses a default of 1000000 if not specified during installation. Defines the maximum number of rows for a single rotation of the audit log table in the database. After reaching this number of records, the table will rotate.</td>
</tr>
<tr>
<td>MTWILSON_AUDIT_LOG_NUM_ROTATIONS</td>
<td>10</td>
<td>Optional; uses a default of 10 if not specified during installation. Defines the maximum number of rotations for the database audit table. After this number of rotations have occurred, subsequent rotations will result in the deletion of the oldest rotation to make room for the newest one.</td>
</tr>
<tr>
<td>POSTGRESQL_KEEP_PGPASS</td>
<td>TRUE</td>
<td>If set to &quot;TRUE,&quot; the Postgres database connectivity information will be stored in plaintext in the hidden file /opt/mtwilson/configuration/.pgpass. This prevents the user from being prompted for database credentials whenever the Hoist Verification Service starts.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ADD_POSTGRESQL_REPO</td>
<td>yes</td>
<td>This setting instructs the installer to add the Postgresql repository to the repository list.</td>
</tr>
<tr>
<td>TAG_VALIDITY_SECONDS</td>
<td>31536000</td>
<td>This value defines in seconds the length of time Asset Tag Certificates will remain valid.</td>
</tr>
<tr>
<td>JETTY_TLS_CERT_DN</td>
<td>CN=Mt Wilson</td>
<td>Defines the Distinguished Name for the TLS Certificate</td>
</tr>
<tr>
<td>JETTY_PORT</td>
<td>8442</td>
<td>Insecure (http) Jetty port. This port must match the port defined in the endpoint.url value.Insecure (http) Jetty port. This port must match the port defined in the endpoint.url value.</td>
</tr>
<tr>
<td>JETTY_SECURE_PORT</td>
<td>8443</td>
<td>Secure (https) Jetty port. This port must match the port defined in the mtwilson.api.url value.</td>
</tr>
<tr>
<td>JAVAX_NET_SSL_KEYSTOR E</td>
<td>/opt/mtwilson/configuration/keystore.jks</td>
<td>Defines the location of the Jetty webserver SSL keystore.</td>
</tr>
<tr>
<td>VS_TLS_CERT_DNS</td>
<td>devops5,localhost</td>
<td>Comma-separated list of hostnames to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>VS_TLS_CERT_IP</td>
<td>192.168.1.1,127.0.0.1</td>
<td>Comma-separated list of IP addresses to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>QUEUE_EXECUTION_INTERVAL</td>
<td>3</td>
<td>Defines the frequency in seconds at which the background queue process is executed</td>
</tr>
<tr>
<td>ESXIHOSTS_AUTOUPDATE_INTERVAL</td>
<td>120</td>
<td>Defines the frequency in seconds at which ESXi host information is retrieved from configured vCenters and updated in the database</td>
</tr>
<tr>
<td>MTWILSON_TELEMETRY_INTERVAL</td>
<td>86400</td>
<td>Defines the frequency in seconds at which telemetry data is collected. Telemetry data includes a simple count of all hosts currently registered.</td>
</tr>
<tr>
<td>CMS_TLS_CERT_SHA384</td>
<td></td>
<td>Certificate Management Service TLS digest</td>
</tr>
<tr>
<td>BEARER_TOKEN</td>
<td></td>
<td>Token with roles to download tls, saml and related setup activities</td>
</tr>
<tr>
<td>AAS_API_URL</td>
<td>https://&lt;Authentication and Authorization Service IP or Hostname&gt;:8444/aas</td>
<td>Base url of the AAS</td>
</tr>
<tr>
<td>CMS_BASE_URL</td>
<td>https://&lt;Certificate Management Service IP or Hostname&gt;:8445/cms/v1</td>
<td>Base url of the CMS</td>
</tr>
<tr>
<td>SKIP_FLAVOR_SIGNATURE_VERIFICATION</td>
<td>false</td>
<td>If set to true, this will cause the Verification Service to skip signature validation on flavors. This is not recommended. Default value is false.</td>
</tr>
</tbody>
</table>
11.1.2 Configuration Options

The Verification Service configuration is encrypted and stored in the file /opt/mtwilson/configuration/mtwilson.properties. To view or change any configuration settings, use the following commands:

**View Configuration:**

```bash
mtwilson export-config --stdout
```

To change the value of any configuration setting in mtwilson.properties, use the following command:

```bash
mtwilson config <key> <value>
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtwilson.queue.execution.interval</td>
<td>3</td>
<td>Defines in seconds the length of time between background queue operations. The background queue is the process that automatically matches Flavors to Hosts that share the same Flavorgroup.</td>
</tr>
<tr>
<td>mtwilson.queue.max.threads</td>
<td>32</td>
<td>The number of threads that the background queue will allocate for concurrent task execution.</td>
</tr>
<tr>
<td>mtwilson.queue.execution.timeout</td>
<td>60</td>
<td>The amount of time in seconds an individual background queue task is allowed to take before throwing a timeout exception.</td>
</tr>
<tr>
<td>mtwilson.telemetry.interval</td>
<td>86400</td>
<td>Defines in seconds the length of time between checks for the number of hosts registered in the Verification Service. By default this checks once every 24 hours. A report of the last 90 days of host counts can be retrieved through a REST API (see the Javadoc for details).</td>
</tr>
<tr>
<td>mtwilson.esxihosts.autoupdate.interval</td>
<td>120</td>
<td>Defines in seconds the length of time between checks to see if any changes have occurred in vCenter for any VMware vCenter Cluster objects that have been registered with the Verification Service. If a new host has been added or removed from the Cluster in vCenter, the host will be added or removed respectively in the Verification Service as well.</td>
</tr>
<tr>
<td>saml.validity.seconds</td>
<td>3600</td>
<td>Defines in seconds the length of time a Report will remain valid. The Verification Service automatically checks for Reports that are nearing expiration and refreshes them; lowering this value will increase the frequency of automatic background Report generation.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mtwilson.tls.policy.allow</td>
<td>certificate,certificate-digest,public-key,certificate-digest,TRUST_FIRST_CERTIFICATE</td>
<td>Defines the TLS policies that will be allowed. TLS policies not in this list will be denied. This list is based off of the MTWILSON_TLS_POLICY_ALLOW list provided at installation time in the mtwilson.env answer file. See the TLS Policy section for information on TLS Policies.</td>
</tr>
<tr>
<td>mtwilson.default.tls.policy.id</td>
<td>TRUST_FIRST_CERTIFICATE</td>
<td>Defines the default TLS policy to be used if no TLS policy is specified. This value is based off of the MTWILSON_DEFAULT_TLS_POLICY_ID value provided at installation time in the mtwilson.env answer file. Note that only TRUST_FIRST_CERTIFICATE and INSECURE can be specified during installation, because no other TLS policies exist at the time of installation. This can be changed to a new TLS policy, however, by setting the ID of the TLS policy to be used as a default. This setting is optional.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageIncludeFilter.startsWith</td>
<td>com.intel,org.glassfish.jersey.media.multipart</td>
<td>Do not change this value. Java code package name string to include for classpath jar loading optimization.</td>
</tr>
<tr>
<td>mtwilson.extensions.fileIncludeFilter.contains</td>
<td>mtwilson, jersey-media-multipart</td>
<td>Do not change this value. Java jar filename string identifier to include for classpath jar loading optimization.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageExcludeFilter.startsWith</td>
<td>java,javax</td>
<td>Do not change this value. Java code package name string to exclude for classpath jar loading optimization.</td>
</tr>
<tr>
<td>mtwilson.host</td>
<td>192.168.1.1</td>
<td>The IP address or hostname of the Verification Service. This is configured at installation time by the MTWILSON_SERVER value specified in the mtwilson.env answer file at installation time.</td>
</tr>
<tr>
<td>mtwilson.api.url</td>
<td>https://192.168.1.1:8443/mtwilson/v1</td>
<td>Defines the baseurl the Verification Service v1 APIs. Note that this will need to be updated if the Jetty secure port is changed.</td>
</tr>
<tr>
<td>dbcp.validation.query</td>
<td>select 1</td>
<td>Query used to verify that the database is accessible.</td>
</tr>
<tr>
<td>dbcp.validation.on.return</td>
<td>false</td>
<td>This property determines whether or not the pool will validate objects before they are borrowed from the pool.</td>
</tr>
<tr>
<td>dbcp.validation.on.borrow</td>
<td>true</td>
<td>This property determines whether or not the pool will validate objects before they are borrowed from the pool.</td>
</tr>
<tr>
<td>mtwilson.locales</td>
<td>en-US</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.db.user</td>
<td>root</td>
<td>Defines the database user.</td>
</tr>
<tr>
<td>mtwilson.db.password</td>
<td>dbpassword</td>
<td>Defines the database password.</td>
</tr>
<tr>
<td>mtwilson.db.driver</td>
<td>org.postgresql.Driver</td>
<td>Defines the database driver to be used. Do not change this value.</td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td><strong>Sample Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>mtwilson.db.schema</td>
<td>mw_as</td>
<td>Defines the database schema name.</td>
</tr>
<tr>
<td>mtwilson.db.port</td>
<td>5432</td>
<td>Defines the database connection port. The default port for a locally installed Postgresql database server is 5432.</td>
</tr>
<tr>
<td>mtwilson.db.host</td>
<td>127.0.0.1</td>
<td>Defines the IP or hostname of the database server. By default for a locally installed Postgresql database server this will be 127.0.0.1.</td>
</tr>
<tr>
<td>mtwilson.audit.log.max.row.count</td>
<td>1000000</td>
<td>Defines the maximum number of rows for a single rotation of the audit log table in the database. After reaching this number of records, the table will rotate.</td>
</tr>
<tr>
<td>mtwilson.audit.log.num.rotations</td>
<td>10</td>
<td>Defines the maximum number of rotations for the database audit table. After this number of rotations have occurred, subsequent rotations will result in the deletion of the oldest rotation to make room for the newest one.</td>
</tr>
<tr>
<td>mtwilson.privacyca.ek.p12.password</td>
<td></td>
<td>Defines Endorsement CA password</td>
</tr>
<tr>
<td>mtwilson.privacyca.aik.p12.password</td>
<td></td>
<td>Defines Privacy CA password</td>
</tr>
<tr>
<td>mtwilson.as.dek</td>
<td></td>
<td>Decryption key used for sensitive data encrypted in the database.</td>
</tr>
<tr>
<td>saml.key.alias</td>
<td>samlkey1</td>
<td>Alias for the SAML signing certificate. Do not change this value.</td>
</tr>
<tr>
<td>saml.keystore.file</td>
<td>SAML.jks</td>
<td>Keystore that contains the SAML signing key.</td>
</tr>
<tr>
<td>jetty.tls.cert.dn</td>
<td>CN=Mt Wilson</td>
<td>Defines the Distinguished Name for the TLS Certificate</td>
</tr>
<tr>
<td>jetty.port</td>
<td>8442</td>
<td>Insecure (http) Jetty port. This port must match the port defined in the endpoint.url value.</td>
</tr>
<tr>
<td>jetty.secure.port</td>
<td>8443</td>
<td>Secure (https) Jetty port. This port must match the port defined in the mtwilson.api.url value.</td>
</tr>
<tr>
<td>javax.net.ssl.keyStore</td>
<td>/opt/mtwilson/configuration/keystore.jks</td>
<td>Defines the location of the Jetty webserver SSL keystore.</td>
</tr>
<tr>
<td>jetty.tls.cert.dns</td>
<td>devops5,localhost</td>
<td>Comma-separated list of hostnames to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>jetty.tls.cert.ip</td>
<td>192.168.1.1,127.0.0.1</td>
<td>Comma-separated list of IP addresses to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>mtwilson.ca.dn</td>
<td>CN=mtwilson-ca,OU=mtwilson</td>
<td>CA distinguishable name</td>
</tr>
<tr>
<td>SKIP_FLAVOR_SIGNATURE_VERIFICATION</td>
<td>false</td>
<td>If set to true, the signature verification of Flavors will be skipped. Defaults to False.</td>
</tr>
</tbody>
</table>

### 11.1.3 Command-Line Options

The Verification Service supports several command-line commands that can be executed only as the Root user:

Syntax:

mtwilson <command>

#### 11.1.3.1 Help

mtwilson help

Displays the list of available CLI commands.

#### 11.1.3.2 Start

mtwilson start

Starts the services.

#### 11.1.3.3 Stop

mtwilson stop

Stops the services.

#### 11.1.3.4 Restart

mtwilson restart

Restarts the services.

#### 11.1.3.5 Status

mtwilson status

Reports whether the service is currently running.

#### 11.1.3.6 Uninstall

mtwilson uninstall
Uninstalls the service, including the deletion of all files and folders. Database content is not removed. See section 14.1 for additional details.

11.1.3.7 Version

mtwilson version

Reports the version of the service.

11.1.3.8 Fingerprint

mtwilson fingerprint

Displays the TLS certificate information.

11.1.3.9 Java-detect

mtwilson java-detect

Displays the detected path and installed version of Java.

11.1.3.10 Erase-data

mtwilson erase-data

Deletes all non-user information from the database.

11.1.3.11 Erase-users

mtwilson erase-users [--all]

Deletes all users from the database, except for the default administrative user. If the “—all” option is used, the administrative user will be deleted as well.

11.1.3.12 Zeroize

mtwilson zeroize

Shreds all secrets, keys, and configurations.

11.1.3.13 Login-password

mtwilson login-password [username] [password] [--permissions] [permission1] [permission2]...

Creates a new user with the specified username and password. If the --permissions option is used, the user can be directly assigned permissions.
Note: This command only allows the assignment of individual permissions, not roles. Permissions are defined in a domain:permission format. For example, hosts:create would allow the user to create new hosts, but not modify or delete existing hosts. Any number of permissions may be applied to a single user. Wildcards are also acceptable; the permissions *:* grants all permissions on all domains, effectively creating an administrative user.

11.1.3.14 Export-config

mtwilson export-config <outfile|--in=infile|--out=outfile|--stdout>

Exports the current configuration. Configuration settings are stored in the encrypted file /opt/mtwilson/configuration/mtwilson.properties; this command allows the configuration to be decrypted or output to the console.

11.1.3.15 Config

mtwilson config [key] [value]

Configures a specified configuration setting to a specified value. Changing settings may require a service restart to take effect.

11.1.3.16 Setup

mtwilson setup [--force|--noexec] [task1 task2 ...]

Re-runs the installation setup tasks, or the specific tasks listed.

11.1.3.17 Replace-tls-key-pair

mtwilson replace-root-key-pair [--private-key=newprivatekey.pem] [--cert-chain=]

Replaces the TLS key pair and certificate. See the Certificate and Key Management section for more details.

11.1.4 Directory Layout

The Verification Service installs by default to /opt/mtwilson with the following folders.

11.1.4.1 Backup

This folder contains backup copies of the Service configuration files, generated at installation with a timestamp.
11.1.4.2 Bin
This folder contains executable scripts.

11.1.4.3 Configuration
This folder contains certificates, keys, and configuration files.

11.1.4.4 Env
This folder contains environment variable files.

11.1.4.5 Features
This folder contains utility scripts and files for specific features.

11.1.4.6 Java
This folder contains application Java libraries.

11.1.4.7 Logs
This folder contains log files.

11.1.4.8 Monit
This folder contains the configuration files for the Monit process monitoring application.

11.1.4.9 Repository

11.1.4.10 Share
## 11.2 Trust Agent

### 11.2.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAS_API_URL</td>
<td>AAS_API_URL=https://{host}:{port}/aas/v1</td>
<td>API URL for Authentication Authorization Service (AAS).</td>
</tr>
<tr>
<td>AUTOMATIC_PULL_MANIFEST</td>
<td>AUTOMATIC_PULL_MANIFEST=Y</td>
<td>Instructs the installer to automatically pull application-manifests from HVS similar to tagent setup get-configured-manifest</td>
</tr>
<tr>
<td>AUTOMATIC_REGISTRATION</td>
<td>AUTOMATIC_REGISTRATION=Y</td>
<td>Instructs the installer to automatically register the host with HVS similar to running tagent setup create-host and tagent setup create-host-unique-flavor.</td>
</tr>
<tr>
<td>BEARER_TOKEN</td>
<td>BEARER_TOKEN=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJzdWIiLCJuYW1lIjoiVXNlcCIiLCJpZ...</td>
<td>JWT from AAS that contains &quot;install&quot; permissions needed to access ISECL services during provisioning and registration</td>
</tr>
<tr>
<td>CMS_BASE_URL</td>
<td>CMS_BASE_URL=https://{host}:{port}/cms/v1</td>
<td>API URL for Certificate Management Service (CMS).</td>
</tr>
<tr>
<td>CMS_TLS_CERT_SHA384</td>
<td>CMS_TLS_CERT_SHA384=bd8af5091289958b5765da4...</td>
<td>SHA384 Hash sum for verifying the CMS TLS certificate.</td>
</tr>
<tr>
<td>MTWILSON_API_URL</td>
<td>MTWILSON_API_URL=http://{host}:{port}/mtwilson/v2</td>
<td>The url used during setup to request information from HVS.</td>
</tr>
<tr>
<td>PROVISION_ATTESTATION</td>
<td>PROVISION_ATTESTATION=N=Y</td>
<td>When present, enables/disables whether tagent setup is called during installation. If trustagent.env is not present, the value defaults to no ('N').</td>
</tr>
<tr>
<td>SAN_LIST</td>
<td>SAN_LIST=10.123.100.1, 201.102.10.22,mya.example.com</td>
<td>CSV list that sets the value for SAN list in the TA TLS certificate. Defaults to 127.0.0.1.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TA_TLS_CERT_CN</td>
<td>TA_TLS_CERT_CN=Acme Trust Agent 007</td>
<td>Sets the value for Common Name in the TA TLS certificate. Defaults to CN=trustagent.</td>
</tr>
<tr>
<td>TPM_OWNER_SECRET</td>
<td>TPM_OWNER_SECRET=625d6...</td>
<td>20 byte hex value to be used as the secret key when taking ownership of the TPM. <em>Note: If this field is not specified, GTA will generate a random secret key.</em></td>
</tr>
<tr>
<td>TPM_QUOTE_IPV4</td>
<td>TPM_QUOTE_IPV4=no</td>
<td>When enabled (=y), uses the local system's ip address as a salt when processing a quote nonce. This field must align with the configuration of HVS.</td>
</tr>
<tr>
<td>TA_SERVER_READ_TIMEOUT</td>
<td>TA_SERVER_READ_TIMEOUT=30</td>
<td>Sets tagent server ReadTimeout. Defaults to 30 seconds.</td>
</tr>
<tr>
<td>TA_SERVER_READ_HEADER_TIMEOUT</td>
<td>TA_SERVER_READ_HEADER_TIMEOUT=10</td>
<td>Sets tagent server ReadHeaderTimeout. Defaults to 30 seconds.</td>
</tr>
<tr>
<td>TA_SERVER_WRITE_TIMEOUT</td>
<td>TA_SERVER_WRITE_TIMEOUT=10</td>
<td>Sets tagent server WriteTimeout. Defaults to 10 seconds.</td>
</tr>
<tr>
<td>TA_SERVER_IDLE_TIMEOUT</td>
<td>TA_SERVER_IDLE_TIMEOUT=10</td>
<td>Sets tagent server IdleTimeout. Defaults to 10 seconds.</td>
</tr>
<tr>
<td>TA_SERVER_MAX_HEADER_BYTES</td>
<td>TA_SERVER_MAX_HEADER_BYTES=1048576</td>
<td>Sets tagent server MaxHeaderBytes. Defaults to 1MB(1048576)</td>
</tr>
<tr>
<td>TA_ENABLE_CONSOLE_LOG</td>
<td>TA_ENABLE_CONSOLE_LOG=true</td>
<td>When set true, tagent logs are redirected to stdout. Defaults to false</td>
</tr>
</tbody>
</table>
### 11.2.2 Configuration Options

The Trust Agent configuration settings are managed in

/opt/trustagent/configuration/config.yml

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpmquoteipv4: true</td>
<td>When enabled, the Trust Agent will perform an additional hash of the nonce using the bytes from the Trust Agent server IP when returning TPM quotes. This should always be set to True.</td>
</tr>
<tr>
<td>logging:</td>
<td></td>
</tr>
<tr>
<td>loglevel: info</td>
<td>Defines the Trust Agent logging level</td>
</tr>
<tr>
<td>logenablestdout: false</td>
<td>If set to True, the Trust Agent will log to stdout. By default this is False and the logs are sent to /var/log/trustagent/trustagent.log</td>
</tr>
<tr>
<td>logentrymaxlength: 300</td>
<td>Defines the maximum length of a single log entry</td>
</tr>
<tr>
<td>webservice:</td>
<td></td>
</tr>
<tr>
<td>port: 1443</td>
<td>Defines the port on which the Trust Agent API server will listen</td>
</tr>
<tr>
<td>readtimeout: 30s</td>
<td></td>
</tr>
<tr>
<td>configuration</td>
<td>description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>readheadertimeout: 10s</td>
<td></td>
</tr>
<tr>
<td>writetimeout: 10s</td>
<td></td>
</tr>
<tr>
<td>idletimeout: 10s</td>
<td></td>
</tr>
<tr>
<td>maxheaderbytes: 1048576</td>
<td></td>
</tr>
<tr>
<td>hvs:</td>
<td></td>
</tr>
<tr>
<td>url:</td>
<td><a href="https://0.0.0.0:8443/mtwilson/v2">https://0.0.0.0:8443/mtwilson/v2</a></td>
</tr>
<tr>
<td></td>
<td>Defines the baseurl for the Verification Service</td>
</tr>
<tr>
<td>tpm:</td>
<td></td>
</tr>
<tr>
<td>ownersecretkey: 625d6d8...1be0b4e957</td>
<td>Defines the TPM ownership secret. This is randomly generated unless manually specified during installation in the trustagent.env file. Note that changing this value may require clearing the TPM ownership in the server BIOS.</td>
</tr>
<tr>
<td>aiksecretkey: 59acd1367...edcbede60c</td>
<td>Defines the AIK secret. Randomly generated. If this is changed, a new AIK will need to be provisioned.</td>
</tr>
<tr>
<td>aas:</td>
<td></td>
</tr>
<tr>
<td>baseurl:</td>
<td><a href="https://0.0.0.0:8444/aas/">https://0.0.0.0:8444/aas/</a></td>
</tr>
<tr>
<td></td>
<td>Defines the base URL for the AAS</td>
</tr>
<tr>
<td>cms:</td>
<td></td>
</tr>
<tr>
<td>baseurl:</td>
<td><a href="https://0.0.0.0:8445/cms/v1">https://0.0.0.0:8445/cms/v1</a></td>
</tr>
<tr>
<td></td>
<td>Defines the base URL for the CMS</td>
</tr>
<tr>
<td>tlscertdigest: 330086b3...ae477c8502</td>
<td>Defines the SHA383 hash of the CMS TLS certificate</td>
</tr>
<tr>
<td>tls:</td>
<td></td>
</tr>
<tr>
<td>certsan: 10.1.2.3,server.domain.com,localhost</td>
<td>Comma-separated list of hostnames and IP addresses for the Trust Agent. Used in the Agent TLS certificate.</td>
</tr>
<tr>
<td>certcn: Trust Agent TLS Certificate</td>
<td>Common Name for the Trust Agent TLS certificate</td>
</tr>
</tbody>
</table>
11.2.3 Command-Line Options

11.2.3.1 Available Commands

11.2.3.1.1 help

Show the help message.

11.2.3.1.2 setup [task]

Run setup task. Available Tasks for 'setup':

**tagent setup (all)**
- Runs all setup tasks to provision the trust agent.
- Required environment variables: AAS_API_URL, CMS_BASE_URL, CMS_TLS_CERT_SHA384, BEARER_TOKEN, MTWILSON_API_URL

**tagent setup trustagent.env**
- Runs all setup tasks to provision the trust agent using trustagent.env file for environment variables (the file must contain all of the required environment variables listed in 'tagent setup (all)'. See "Environment variables" below).

**tagent setup download-ca-cert**
- Fetches the latest CMS Root CA Certificates, overwriting existing files.
- Required environment variables: BEARER_TOKEN, CMS_BASE_URL

**tagent setup download-cert**
- Fetches a signed TLS Certificate from CMS, overwriting existing files.
- Required environment variables: CMS_BASE_URL, CMS_TLS_CERT_SHA384

**tagent setup update-certificates**
- Runs 'download-ca-cert' and 'download-cert'
- Required environment variables: CMS_BASE_URL, CMS_TLS_CERT_SHA384, BEARER_TOKEN

**tagent setup provision-attestation**
- Runs setup tasks associated with HVS/TPM provisioning.
- Required environment variables: BEARER_TOKEN, MTWILSON_API_URL

**tagent setup create-host**
- Registers the trust agent with the verification service.
- Required environment variables: Bearer_Token, MTWilson_API_URL

**Tadget setup create-host-unique-flavor**
- Populates the verification service with the host unique flavor
- Required environment variables: Bearer_Token, MTWilson_API_URL

**Tadget setup get-configured-manifest**
- Uses environment variables to pull application-integrity manifests from the verification service.
- Required Environment variables: Bearer_Token, MTWilson_API_URL, Flavor_UUIDs or Flavor_Lables

**Environment variables used by tadget setup:**

* Indicates the environment variable is optional.

**AAS_API_URL**
- Used by the trust agent service to validate jwt/bearer tokens.
- Ex. AAS_API_URL=https://{host}:{port}/aas/v1

**AUTOMATIC_PULL_MANIFEST**
- When 'Y', instructs the installer to download application manifests using the Flavor_UUIDs or Flavor_Lables environment variables by running 'get-configured-manifest'. Defaults to 'N'.
- Ex. AUTOMATIC_PULL_MANIFEST=Y

**AUTOMATIC_REGISTRATION**
- When 'Y', instructs the installer to register the host with HVS by running 'create-host' and 'create-host-unique-flavor'. Defaults to 'N'.
- Ex. AUTOMATIC_REGISTRATION=Y

**Bearer_Token**
- 'jwt' token used during setup to communicate to CMS and HVS.
- Ex. Bearer_Token=eyJhbGciOiJSUzM4NCIsjdkMTdiNmUz...

**CMS_BASE_URL**
- URL used by setup to download root-ca and tls certificates from CMS.
- Ex. CMS_BASE_URL=https://{host}:{port}/cms/v1

**CMS_TLS_CERT_SHA384**
- SHA384 sum used during setup to secure communications with CMS.
- Ex. CMS_TLS_CERT_SHA384=bd8ebf5091289958b5765da4...

**MTWilson_API_URL**
- The URL used during setup to collect information from HVS.
- MTWILSON_API_URL=https://{host}:{port}/mtwilson/v2

PROVISION_ATTESTATION*
- When 'Y', instructs the installer to provision the host with HVS by calling 'tagent setup'. Defaults to 'N'.
- Ex. AUTOMATIC_REGISTRATION=Y

SAN_LIST*
- CSV list that sets the value for SAN list in the TA TLS certificate. Defaults to "127.0.0.1,localhost".
- Ex. SAN_LIST=10.123.100.1,201.102.10.22,my.example.com

TA_ENABLE_CONSOLE_LOG*
- When set to 'true', trust agent logs are redirected to stdout. Defaults to false.
- Ex. TA_ENABLE_CONSOLE_LOG=true

TA_SERVER_IDLE_TIMEOUT*
- Sets the trust agent service's idle timeout. Defaults to 10 seconds.
- Ex. TA_SERVER_IDLE_TIMEOUT=10

TA_SERVER_MAX_HEADER_BYTES*
- Sets trust agent service's maximum header bytes. Defaults to 1MB.
- Ex. TA_SERVER_MAX_HEADER_BYTES=1048576

TA_SERVER_READ_TIMEOUT*
- Sets trust agent service's read timeout. Defaults to 30 seconds.
- Ex. TA_SERVER_READ_TIMEOUT=30

TA_SERVER_READ_HEADER_TIMEOUT*
- Sets trust agent service's read header timeout. Defaults to 30 seconds.
- Ex. TA_SERVER_READ_HEADER_TIMEOUT=10

TA_SERVER_WRITE_TIMEOUT*
- Sets trust agent service's write timeout. Defaults to 10 seconds.
- Ex. TA_SERVER_WRITE_TIMEOUT=10

TA_TLS_CERT_CN*
- Sets the value for Common Name in the TA TLS certificate. Defaults to "Trust Agent TLS Certificate".
- Ex. TA_TLS_CERT_CN=Acme Trust Agent 007

TPM_OWNER_SECRET*
- When provided, setup uses the 40 character hex string for the TPM owner password. The TPM owner secret is generated when not provided.
- Ex.
TPM_OWNER_SECRET=625d6d8a18f98bf764760fa392b8c01be0b4e959

TPM_QUOTE_IPV4*
- When 'Y', used the local system's ip address a salt when processing
TPM quotes. Defaults to 'N'.
- Ex. TPM_QUOTE_IPV4=Y

TRUSTAGENT_LOG_LEVEL*
- Sets the verbosity level of logging (trace|debug|info|error). Defaults to 'info'.
- Ex. TRUSTAGENT_LOG_LEVEL=debug

TRUSTAGENT_PORT*
- The port on which the trust agent service will listen.
- Ex. TRUSTAGENT_PORT=10433

11.2.3.1.3 uninstall
Uninstall trust agent.

11.2.3.1.4 version
Print build version info.

11.2.3.1.5 start
Start the trust agent service.

11.2.3.1.6 stop
Stop the trust agent service.

11.2.3.1.7 status
Get the status of the trust agent service.

11.2.4 Directory Layout

11.2.4.1 Windows

11.2.4.2 Linux
The Linux Trust Agent installs by default to /opt/trustagent, with the following subfolders:

11.2.4.2.1 Bin
Contains executables and scripts.
11.2.4.2.2 Configuration

Contains the config.yml configuration file, as well as certificates and keystores. This includes the AIK public key blob after provisioning.

11.2.4.2.3 Var

Contains information gathered from the platform and SOFTWARE Flavor manifests. All files with the name “manifest_*.xml” will be parsed to define measurements during boot. Generally these should be automatically provisioned from the Verification Service when creating/deploying SOFTWARE Flavors.

11.3 Integration Hub

11.3.1 Installation Answer File

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong></td>
<td></td>
</tr>
<tr>
<td>MTWILSON_SERVER</td>
<td>IP address or hostname of the HVS</td>
</tr>
<tr>
<td>MTWILSON_API_URL</td>
<td>https://&lt;Verification Service IP or hostname&gt;:8443/mtwilson/v2</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_NAME</td>
<td>attestation_hub_pu</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_HOSTNAME</td>
<td>localhost</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_PORTNUM</td>
<td>5432</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_DRIVER</td>
<td>org.postgresql.Driver</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_USERNAME</td>
<td>&lt;Database administrative username&gt;</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_PASSWORD</td>
<td>&lt;Database password&gt;</td>
</tr>
<tr>
<td>CMS_TLS_CERT_SHA384</td>
<td>&lt;Certificate Management Service TLS digest&gt;</td>
</tr>
<tr>
<td>Bearer_TOKEN</td>
<td>Installation token</td>
</tr>
<tr>
<td>AAS_API_URL</td>
<td>https://&lt;Authentication and Authorization Service IP or Hostname&gt;:8444/aas</td>
</tr>
<tr>
<td>CMS_BASE_URL</td>
<td>https://&lt;Certificate Management Service IP or Hostname&gt;:8445/cms/v1</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td>ATTESTATION_HUB_PORT_HTTP</td>
<td>19082</td>
</tr>
<tr>
<td>ATTESTATION_HUB_PORT_HTTPS</td>
<td>19445</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ATTESTATION_HUB_POLL_INTERVAL</td>
<td>2</td>
</tr>
<tr>
<td>ATTESTATION_HUB_SAML_TIMEOUT</td>
<td>1440</td>
</tr>
<tr>
<td>AH_TLS_CERT_IP</td>
<td>127.0.0.1,192.168.1.1</td>
</tr>
<tr>
<td>AH_TLS_CERT_DNS</td>
<td>Localhost, server.sample.com</td>
</tr>
<tr>
<td>MTWILSON_SERVER_PORT</td>
<td>8443</td>
</tr>
<tr>
<td>LOG_ROTATION_PERIOD</td>
<td>daily, weekly, monthly, yearly</td>
</tr>
<tr>
<td>LOG_COMPRESS</td>
<td>compress</td>
</tr>
<tr>
<td>LOG_DELAYCOMPRESS</td>
<td>delaycompress</td>
</tr>
<tr>
<td>LOG_COPYTRUNCATE</td>
<td>copytruncate</td>
</tr>
<tr>
<td>LOG_SIZE</td>
<td>1K</td>
</tr>
<tr>
<td>LOG_OLD</td>
<td>12</td>
</tr>
</tbody>
</table>

11.3.2 Configuration Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpoint.url</td>
<td>http://server.com:19082</td>
<td>Defines the baseurl for the Hub web server API using the http port.</td>
</tr>
<tr>
<td>jetty.tls.cert.dns</td>
<td>server.com,localhost</td>
<td>Subject Alternative Names</td>
</tr>
<tr>
<td>attestation-hub.db.schema</td>
<td>attestation_hub_pu</td>
<td>Defines the database schema name.</td>
</tr>
<tr>
<td>attestation-hub.poll.interval</td>
<td>2</td>
<td>Defines in minutes how long the Hub will wait before sending a new polling request to the Verification Service to retrieve new reports.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mtwilson.api.password</td>
<td>HubPassword</td>
<td>Defines the Verification Service user that will be used by the Integration Hub for API requests. This user must be created with at minimum the reports:search permission or the reports_manager or auditor roles.</td>
</tr>
<tr>
<td>mtwilson.api.url</td>
<td>https://server.com:8443/mtwilson/v2</td>
<td>Defines the base URL for the Verification Service API.</td>
</tr>
<tr>
<td>javax.net.ssl.keyStore</td>
<td>/opt/attestation-hub/configuration/keystore.jks</td>
<td>Defines the SSL keystore file path.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageExcludeFilter.startsWith</td>
<td>java, javax</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.username</td>
<td>hubadmin</td>
<td>Defines the Verification Service user that will be used by the Integration Hub for API requests. This user must be created with at minimum the reports:search permission or the reports_manager or auditor roles.</td>
</tr>
<tr>
<td>jetty.tls.cert.ip</td>
<td>192.168.1.1,127.0.0.1</td>
<td>Defines the IP addresses that are listed in the Integration Hub TLS Certificate as Subject Alternative Names. These addresses are used to validate connections to the Hub.</td>
</tr>
<tr>
<td>attestation-hub.db.password</td>
<td>dbpassword</td>
<td>Defines the database connection password.</td>
</tr>
<tr>
<td>attestation-hub.db.driver</td>
<td>org.postgresql.Driver</td>
<td>Defines the database connection driver. Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.extensions.fileIncludeFilter.contains</td>
<td>mtwilson, attestation-hub</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.api.tls.policy.certificate.sha384</td>
<td></td>
<td>20 hex-encoded bytes. Obtain this from the Verification Service server, in /opt/mtwilson/configuration/https.properties</td>
</tr>
<tr>
<td>jetty.port</td>
<td>19082</td>
<td>Defines the webserver insecure (http) port.</td>
</tr>
<tr>
<td>mtwilson.server</td>
<td>hvs.server.com</td>
<td>Defines the IP address or hostname of the Verification Service from which the Hub will poll new attestation reports.</td>
</tr>
<tr>
<td>jetty.secure.port</td>
<td>19445</td>
<td>Defines the webserver secure (https) port.</td>
</tr>
<tr>
<td>attestation-hub.db.name</td>
<td>attestation_hub_pu</td>
<td>Defines the database schema name.</td>
</tr>
<tr>
<td>password.vault.file</td>
<td>/opt/attestation-hub/configuration/password-vault.jck</td>
<td>Information about keystore path, default would be configuration.</td>
</tr>
<tr>
<td>attestation-hub.db.username</td>
<td>root</td>
<td>Defines the database connection credentials.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageIncludeFilter.startsWith</td>
<td>com.intel.org.glassfish.jersey.media.multipart</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tenant.configuration.path</td>
<td>/opt/tenantconfig</td>
<td>Defines the path where tenant configurations will be stored. These are also stored in the database.</td>
</tr>
<tr>
<td>attestation-hub.db.portnum</td>
<td>5432</td>
<td>Defines the database connection port.</td>
</tr>
<tr>
<td>mtwilson.api.username</td>
<td>hubadmin</td>
<td>Defines the Verification Service credentials that the Hub will use to access the VS API.</td>
</tr>
<tr>
<td>jetty.tls.cert.dn</td>
<td>CN=Attestation Hub</td>
<td>Defines the Distinguished Name of the Hub TLS certificate.</td>
</tr>
<tr>
<td>mtwilson.password</td>
<td>HubPassword</td>
<td>Defines the Verification Service credentials that the Hub will use to access the VS API.</td>
</tr>
<tr>
<td>attestation-hub.db.hostname</td>
<td>localhost</td>
<td>Defines the database connection IP address or hostname.</td>
</tr>
<tr>
<td>password.vault.type</td>
<td>JCEKS</td>
<td>Defines password for keystore vault.</td>
</tr>
<tr>
<td>attestation-hub.db.url</td>
<td>jdbc:postgresql://localhost:5432/attestation_hub_pu</td>
<td>Defines the complete database connection URL.</td>
</tr>
<tr>
<td>mtwilson.server.port</td>
<td>8443</td>
<td>Defines the Verification Service API port.</td>
</tr>
</tbody>
</table>

### 11.3.3 Command-Line Options

#### 11.3.3.1 Available Commands

**11.3.3.1.1 Help**

attestation-hub help

Displays the list of available CLI commands.

**11.3.3.1.2 Start**

attestation-hub start

Starts the services.

**11.3.3.1.3 Stop**

mtwilson stop

Stops the services.

**11.3.3.1.4 Restart**

attestation-hub restart

Restarts the services.
11.3.3.1.5 Status

attestation-hub status

Reports whether the service is currently running.

11.3.3.1.6 Uninstall

attestation-hub uninstall [--purge]

Uninstalls the service, including the deletion of all files and folders. Database content is not removed. If the --purge option is used, database content will be removed during the uninstallation.

11.3.3.1.7 Version

attestation-hub version

Reports the version of the service.

11.3.3.1.8 Password

attestation-hub password [username] [password] --permissions *:* *

Creates a new user with the specified username and password. Because the Hub does not have granular user permissions, the --permissions *:* is necessary and assigns all permissions to the created user.

11.3.3.1.9 Export-config

attestation-hub export-config <outfile|--in=infile|--out=outfile|--stdout>

Exports the current configuration. Configuration settings are stored in the encrypted file /opt/mtwilson/configuration/mtwilson.properties; this command allows the configuration to be decrypted or output to the console.

11.3.3.1.10 Config

attestation-hub config [key] [value]

Configures a specified configuration setting to a specified value. Changing settings may require a service restart to take effect.

11.3.3.1.11 Setup

attestation-hub setup [--force|--noexec] [task1 task2 ...]

Re-runs the installation setup tasks, or the specific tasks listed.
11.3.4 Directory Layout

11.3.4.1 Logs

11.4 Certificate Management Service

11.4.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS_NOSETUP</td>
<td>false</td>
<td>Determines whether &quot;setup&quot; will be executed after installation. Typically this is set to &quot;false&quot; to install and perform setup in one action. The &quot;true&quot; option is intended for building the service as a container, where the installation would be part of the image build, and setup would be performed when the container starts for the first time to generate any persistent data.</td>
</tr>
<tr>
<td>CMS_PORT</td>
<td>8445</td>
<td>Defines the HTTPS port the service will use.</td>
</tr>
<tr>
<td>AAS_API_URL</td>
<td>https://&lt;Hostname or IP address of the AAS&gt;:8444/aas/</td>
<td>URL to connect to the AAS, used during setup for authentication.</td>
</tr>
<tr>
<td>AAS_TLS_SAN</td>
<td>&lt;Comma-separated list of IPs/hostnames for the AAS&gt;</td>
<td>SAN list populated in special JWT token, this token is used by AAS to get TLS certificate signed from CMS. SAN list in this token and CSR generated by AAS must match.</td>
</tr>
<tr>
<td>LOG_ROTATION_PERIOD</td>
<td>hourly, daily, weekly, monthly, yearly</td>
<td>Log rotation period, for more details refer-<a href="https://linux.die.net/man/8/logrotate">https://linux.die.net/man/8/logrotate</a></td>
</tr>
<tr>
<td>LOG_COMPRESS</td>
<td>Compress</td>
<td>Old versions of log files are compressed with gzip, for more details refer-<a href="https://linux.die.net/man/8/logrotate">https://linux.die.net/man/8/logrotate</a></td>
</tr>
<tr>
<td>LOG_DELAYCOMPRESS</td>
<td>delaycompress</td>
<td>Postpone compression of the previous log file to the next rotation cycle, for more details refer-<a href="https://linux.die.net/man/8/logrotate">https://linux.die.net/man/8/logrotate</a></td>
</tr>
<tr>
<td>LOG_COPYTRUNCATE</td>
<td>Copytruncate</td>
<td>Truncate the original log file in place after creating a copy,'create' creates new one, for more details refer-<a href="https://linux.die.net/man/8/logrotate">https://linux.die.net/man/8/logrotate</a></td>
</tr>
<tr>
<td>LOG_SIZE</td>
<td>1K</td>
<td>Log files are rotated when they grow bigger than size bytes, for more details refer-<a href="https://linux.die.net/man/8/logrotate">https://linux.die.net/man/8/logrotate</a></td>
</tr>
<tr>
<td>LOG_OLD</td>
<td>12</td>
<td>Log files are rotated count times before being removed, for more details refer-<a href="https://linux.die.net/man/8/logrotate">https://linux.die.net/man/8/logrotate</a></td>
</tr>
<tr>
<td>CMS_CA_CERT_VALIDITY</td>
<td>5</td>
<td>CMS Root Certificate Validity in years</td>
</tr>
<tr>
<td>CMS_CA_ORGANIZATION</td>
<td>INTEL</td>
<td>CMS Certificate Organization</td>
</tr>
</tbody>
</table>

151
<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS_CA_LOCALITY</td>
<td>US</td>
<td>CMS Certificate locality</td>
</tr>
<tr>
<td>CMS_CA_PROVINCE</td>
<td>CA</td>
<td>CMS Certificate province</td>
</tr>
<tr>
<td>CMS_CA_COUNTRY</td>
<td>USA</td>
<td>CMS Certificate country</td>
</tr>
<tr>
<td>CMS_TLS_SAN_LIST</td>
<td></td>
<td>Comma-separated list of IP addresses and hostnames to be added to the SAN list of CMS server</td>
</tr>
<tr>
<td>CMS_SERVER_READ_TIMEOUT</td>
<td>30s</td>
<td>MS server - ReadTimeout is the maximum duration for reading the entire request, including the body.</td>
</tr>
<tr>
<td>CMS_SERVER_READ_HEADER_TIMEOUT</td>
<td>10s</td>
<td>CMS server - ReadHeaderTimeout is the amount of time allowed to read request headers</td>
</tr>
<tr>
<td>CMS_SERVER_WRITE_TIMEOUT</td>
<td>10s</td>
<td>CMS server - WriteTimeout is the maximum duration before timing out writes of the response.</td>
</tr>
<tr>
<td>CMS_SERVER_IDLE_TIMEOUT</td>
<td>10s</td>
<td>CMS server - IdleTimeout is the maximum amount of time to wait for the next request when keep-alives are enabled.</td>
</tr>
<tr>
<td>CMS_SERVER_MAX_HEADER_BYTES</td>
<td>1048576</td>
<td>CMS server - MaxHeaderBytes controls the maximum number of bytes the server will read parsing the request header's keys and values, including the request line.</td>
</tr>
<tr>
<td>AAS_JWT_CN</td>
<td>AAS JWT Signing Certificate</td>
<td>CN of AAS JWT certificate, this gets populated in special JWT token. AAS must send JWT certificate CSR with this CN.</td>
</tr>
<tr>
<td>AAS_TLS_CN</td>
<td>AAS TLS Certificate</td>
<td>CN of AAS TLS certificate, this gets populated in special JWT token. AAS must send TLS certificate CSR with this CN.</td>
</tr>
<tr>
<td>AAS_TLS_SAN</td>
<td></td>
<td>SAN list populated in special JWT token, this token is used by AAS to get TLS certificate signed from CMS. SAN list in this token and CSR generated by AAS must match.</td>
</tr>
</tbody>
</table>

### 11.4.2 Configuration Options

The CMS configuration can be found in `/etc/cms/config.yml`. 
11.4.3 Command-Line Options

11.4.3.1 Help

cms help

Displays the list of available CLI commands.

11.4.3.2 Start

cms start

Starts the services.

11.4.3.3 Stop

cms stop

Stops the service.

11.4.3.4 Restart

cms restart

Restarts the services.

<table>
<thead>
<tr>
<th>Port: 8445</th>
</tr>
</thead>
<tbody>
<tr>
<td>loglevel: info</td>
</tr>
<tr>
<td>authserviceurl: https://&lt;AAS IP or hostname&gt;:8444/aas/</td>
</tr>
<tr>
<td>cacertvalidity: 5</td>
</tr>
<tr>
<td>organization: INTEL</td>
</tr>
<tr>
<td>locality: SC</td>
</tr>
<tr>
<td>province: CA</td>
</tr>
<tr>
<td>country: US</td>
</tr>
<tr>
<td>keyalgorithm: rsa</td>
</tr>
<tr>
<td>keyalgorithmlength: 3072</td>
</tr>
<tr>
<td>rootcacertdigest: &lt;sha384&gt;</td>
</tr>
<tr>
<td>tlscertdigest: &lt;sha384&gt;</td>
</tr>
<tr>
<td>tokendurationmins: 20</td>
</tr>
<tr>
<td>aasjwtcn: &quot;&quot;</td>
</tr>
<tr>
<td>aastlscon: &quot;&quot;</td>
</tr>
<tr>
<td>aastlssan: &quot;&quot;</td>
</tr>
<tr>
<td>authdefender:</td>
</tr>
<tr>
<td>maxattempts: 5</td>
</tr>
<tr>
<td>intervalmins: 5</td>
</tr>
<tr>
<td>lockoutdurationmins: 15</td>
</tr>
</tbody>
</table>
11.4.3.5 Status

cms status
Reports whether the service is currently running.

11.4.3.6 Uninstall

cms uninstall
Uninstalls the service, including the deletion of all files and folders.

11.4.3.7 Version

cms version
Reports the version of the service.

11.4.3.8 Tlscertsha384

Shows the SHA384 of the TLS certificate.

11.4.3.9 setup [task]

Runs a specific setup task.

Available Tasks for setup:

11.4.3.9.1 cms setup server [--port=<port>]
- Setup http server on <port>
- Environment variable CMS_PORT=<port> can be set alternatively

11.4.3.9.2 cms setup root_ca [--force]
- Create its own self signed Root CA keypair in /etc/cms for quality of life
- Option [--force] overwrites any existing files, and always generate new Root CA keypair

11.4.3.9.3 cms setup tls [--force] [--host_names=<host_names>]
- Create its own root_ca signed TLS keypair in /etc/cms for quality of life
- Option [--force] overwrites any existing files, and always generate root_ca signed TLS keypair
- Argument <host_names> is a list of host names used by local machine, seperated by comma
- Environment variable CMS_HOST_NAMES=<host_names> can be set alternatively

11.4.3.9.4 cms setup cms_auth_token [--force]
- Create its own self signed JWT keypair in /etc/cms/jwt for quality of life
- Option [--force] overwrites any existing files, and always generate new JWT keypair and token
11.4.4 Directory Layout

The Certificate Management Service installs by default to /opt/cms with the following folders.

11.4.4.1 Bin

This folder contains executable scripts.

11.4.4.2 Cacerts

This folder contains the CMS root CA certificate.

11.5 Authentication and Authorization Service

11.5.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS_BASE_URL</td>
<td>https://&lt;cms IP or hostname&gt;/cms/v1/</td>
<td>Required; Provides the URL for the CMS.</td>
</tr>
<tr>
<td>AAS_NOSETUP</td>
<td>false</td>
<td>Optional. Determines whether &quot;setup&quot; will be executed after installation. Typically this is set to &quot;false&quot; to install and perform setup in one action. The &quot;true&quot; option is intended for building the service as a container, where the installation would be part of the image build, and setup would be performed when the container starts for the first time to generate any persistent data.</td>
</tr>
<tr>
<td>AAS_DB_HOSTNAME</td>
<td>localhost</td>
<td>Required. Hostname or IP address of the AAS database</td>
</tr>
<tr>
<td>AAS_DB_PORT</td>
<td>5432</td>
<td>Required. Database port number</td>
</tr>
<tr>
<td>AAS_DB_NAME</td>
<td>pgdb</td>
<td>Required. Database name</td>
</tr>
<tr>
<td>AAS_DB_USERNAME</td>
<td>dbuser</td>
<td>Required. Database username</td>
</tr>
<tr>
<td>AAS_DB_PASSWORD</td>
<td>dbpassword</td>
<td>Required. Database password</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AAS_DB_SSLMODE</td>
<td>verify-ca</td>
<td>Defines the SSL mode for the connection to the database. If not specified, the database connection will not use certificate verification. If specified, certificate verification will be required for database connections.</td>
</tr>
<tr>
<td>AAS_DB_SSLCERTSRC</td>
<td>/usr/local/pgsql/data/server.crt</td>
<td>Optional, required if the &quot;AAS_DB_SSLMODE&quot; is set to &quot;verify-ca.&quot; Defines the location of the database SSL certificate.</td>
</tr>
<tr>
<td>AAS_DB_SSLCERT</td>
<td>&lt;path_to_cert_file_on_system&gt;</td>
<td>Optional. The AAS_DB_SSLCERTSRC variable defines the source location of the database SSL certificate; this variable determines the local location. If the former option is used without specifying this option, the service will copy the SSL certificate to the default configuration directory.</td>
</tr>
<tr>
<td>AAS_ADMIN_USERNAME</td>
<td>admin@aas</td>
<td>Required. Defines a new AAS administrative user. This user will be able to create new users, new roles, and new role-user mappings. This user will have the AAS:Administrator role.</td>
</tr>
<tr>
<td>AAS_ADMIN_PASSWORD</td>
<td>aasAdminPass</td>
<td>Required. Password for the new AAS admin user.</td>
</tr>
<tr>
<td>AAS_JWT_CERT_SUBJECT</td>
<td>&quot;AAS JWT Signing Certificate&quot;</td>
<td>Optional. Defines the subject of the JWT signing certificate.</td>
</tr>
<tr>
<td>AAS_JWT_TOKEN_DURATI ON_MINS</td>
<td>5</td>
<td>Optional. Defines the amount of time in minutes that an issued token will be valid.</td>
</tr>
<tr>
<td>SAN_LIST</td>
<td>127.0.0.1,localhost,10.x.x.x</td>
<td>Comma-separated list of IP addresses and hostnames that will be valid connection points for the service. Requests sent to the service using an IP or hostname not in this list will be denied, even if it resolves to this service.</td>
</tr>
<tr>
<td>BEARER_TOKEN</td>
<td>&lt;token&gt;</td>
<td>Required. Token from the CMS generated during CMS setup that allows the AAS to perform initial setup tasks.</td>
</tr>
<tr>
<td>LOG_LEVEL</td>
<td>Critical, error, warning, info, debug, trace</td>
<td>Optional. Defaults to INFO. Changes the log level used.</td>
</tr>
</tbody>
</table>

**11.5.2 Configuration Options**

**11.5.3 Command-Line Options**

**11.5.3.1 Help**

Displays the list of available CLI commands.
11.5.3.2 setup <task>

Executes a specific setup task. Can be used to change the current configuration.

Available Tasks for setup:

11.5.3.2.1 authservice setup all
- Runs all setup tasks

11.5.3.2.2 authservice setup database [-force] [--arguments=<argument_value>]
- Available arguments are:
  - db-host alternatively, set environment variable AAS_DB_HOSTNAME
  - db-port alternatively, set environment variable AAS_DB_PORT
  - db-user alternatively, set environment variable AAS_DB_USERNAME
  - db-pass alternatively, set environment variable AAS_DB_PASSWORD
  - db-name alternatively, set environment variable AAS_DB_NAME
  - db-sslmode <disable|allow|prefer|require|verify-ca|verify-full>
    alternatively, set environment variable AAS_DB_SSLMODE
  - db-sslcert path to where the certificate file of database. Only applicable for db-sslmode=<verify-ca|verify-full. If left empty, the cert will be copied to /etc/authservice/tdcertdb.pem alternatively, set environment variable AAS_DB_SSLCERT
  - db-sslcertsrc <path to where the database ssl/tls certificate file>
    mandatory if db-sslcert does not already exist alternatively, set environment variable AAS_DB_SSLCERTSRC
- Run this command with environment variable AAS_DB_REPORT_MAX_ROWS and AAS_DB_REPORT_NUM_ROTATIONS can update db rotation arguments

11.5.3.2.3 authservice setup server [--port=<port>]
- Setup http server on <port>
- Environment variable AAS_PORT=<port> can be set alternatively
  authservice setup tls [-force] [--host_names=<host_names>]
  - Use the key and certificate provided in /etc/threat-detection if files exist
  - Otherwise create its own self-signed TLS keypair in /etc/authservice for quality of life
  - Option [-force] overwrites any existing files, and always generate self-signed keypair
  - Argument <host_names> is a list of host names used by local machine, seperated by coma
  - Environment variable AAS_TLS_HOST_NAMES=<host_names> can be set alternatively

11.5.3.2.4 authservice setup admin [--user=<username>] [--pass=<password>]
- Environment variable AAS_ADMIN_USERNAME=<username> can be set alternatively
- Environment variable AAS_ADMIN_PASSWORD=<password> can be set alternatively
11.5.3.2.5 authservice setup reghost [--user=<username>] [--pass=<password>]
- Environment variable AAS_REG_HOST_USERNAME=<username> can be set alternatively
- Environment variable AAS_REG_HOST_PASSWORD=<password> can be set alternatively

11.5.3.2.6 authservice setup download_ca_cert [--force]
- Download CMS root CA certificate
- Option [--force] overwrites any existing files, and always downloads new root CA cert
- Environment variable CMS_BASE_URL=<url> for CMS API url

11.5.3.2.7 authservice setup download_cert TLS [--force]
- Generates Key pair and CSR, gets it signed from CMS
- Option [--force] overwrites any existing files, and always downloads newly signed TLS cert
- Environment variable CMS_BASE_URL=<url> for CMS API url
- Environment variable BEARER_TOKEN=<token> for authenticating with CMS
- Environment variable KEY_PATH=<key_path> to override default specified in config
- Environment variable CERT_PATH=<cert_path> to override default specified in config
- Environment variable AAS_TLS_CERT_CN=<TLS CERT COMMON NAME> to override default specified in config
- Environment variable AAS_CERT_ORG=<CERTIFICATE ORGANIZATION> to override default specified in config
- Environment variable AAS_CERT_COUNTRY=<CERTIFICATE COUNTRY> to override default specified in config
- Environment variable AAS_CERT_LOCALITY=<CERTIFICATE LOCALITY> to override default specified in config
- Environment variable AAS_CERT_PROVINCE=<CERTIFICATE PROVINCE> to override default specified in config
- Environment variable SAN_LIST=<san> list of hosts which needs access to service

11.5.3.2.8 authservice setup jwt
- Create jwt signing key and jwt certificate signed by CMS
- Environment variable CMS_BASE_URL=<url> for CMS API url
- Environment variable AAS_JWT_CERT_CN=<CERTIFICATE SUBJECT> AAS JWT Certificate Subject
- Environment variable AAS_JWT_INCLUDE_KEYID=<KEY ID> AAS include key id in JWT Token
- Environment variable AAS_JWT_TOKEN_DURATION_MINS=<DURATION> JWT Token validation minutes
- Environment variable BEARER_TOKEN=<token> for authenticating with CMS

11.5.3.3 Start
Starts the service.

11.5.3.4 Status
Displays the current status of the service.
11.5.3.5  **Stop**
Stops the service.

11.5.3.6  **tlscertsha384**
Shows the SHA384 of the TLS certificate.

11.5.3.7  **Uninstall**
Removes the service. Use the "--purge" flag to also delete all data.

11.5.3.8  **Version**
Shows the version of the service.

11.5.4  **Directory Layout**
The Verification Service installs by default to /opt/authservice with the following folders.

11.5.4.1  **Bin**
Contains executable scripts and binaries.

11.5.4.2  **Dbscripts**
Contains database scripts.

11.6  **Workload Service**

11.6.1  **Installation Answer File Options**

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLS_LOGLEVEL</td>
<td>INFO</td>
<td>(Optional) Alternatives include WARN and DEBUG. Sets the log level for the service.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WLS_NOSETUP</td>
<td>false</td>
<td>(Optional) Determines whether &quot;setup&quot; will be executed after installation. Typically this is set to &quot;false&quot; to install and perform setup in one action. The &quot;true&quot; option is intended for building the service as a container, where the installation would be part of the image build, and setup would be performed when the container starts for the first time to generate any persistent data. Defaults to &quot;false&quot; if unset.</td>
</tr>
<tr>
<td>WLS_PORT</td>
<td>5000</td>
<td>(Optional) Defines the HTTPS port used by the service. Defaults to 5000 if unset.</td>
</tr>
<tr>
<td>WLS_DB_HOSTNAME</td>
<td>localhost</td>
<td>(Required) Database hostname</td>
</tr>
<tr>
<td>WLS_DB</td>
<td>wlsdb</td>
<td>(Required) Database name</td>
</tr>
<tr>
<td>WLS_DB_PORT</td>
<td>5432</td>
<td>(Required) Database port number</td>
</tr>
<tr>
<td>WLS_DB_USERNAME</td>
<td>wlsdbuser</td>
<td>(Required) Database username</td>
</tr>
<tr>
<td>WLS_DB_PASSWORD</td>
<td>wlsdbuserpass</td>
<td>(Required) Database password</td>
</tr>
<tr>
<td>HVS_URL</td>
<td>https://&lt;HVS IP address or hostname&gt;:8443/mtwilson/v2/</td>
<td>(Required) Base URL for the HVS</td>
</tr>
<tr>
<td>AAS_API_URL</td>
<td>https://&lt;AAS IP address or hostname&gt;:8444/aas</td>
<td>Base URL for the AAS</td>
</tr>
<tr>
<td>WLS_CERT_SAN_LIST</td>
<td>127.0.0.1,localhost,10.x.x.x</td>
<td>Comma-separated list of IP addresses and hostnames that will be valid connection points for the service. Requests sent to the service using an IP or hostname not in this list will be denied, even if it resolves to this service.</td>
</tr>
<tr>
<td>CMS_BASE_URL</td>
<td></td>
<td>Base URL for the CMS</td>
</tr>
<tr>
<td>BEARER_TOKEN</td>
<td>&lt;token&gt;</td>
<td>(Required) Token from the CMS generated during CMS setup that allows the AAS to perform initial setup tasks.</td>
</tr>
<tr>
<td>WLS_TLS_CERT_CN</td>
<td>'WLS TLS Certificate'</td>
<td>(Optional) Set the Common name for TLS cert to be downloaded from CMS. Default is 'WLS TLS Certificate'.</td>
</tr>
<tr>
<td>WLS_CERT_ORG</td>
<td>'INTEL'</td>
<td>(Optional) Set the Organization in Subject of CSR. Default is 'INTEL'.</td>
</tr>
<tr>
<td>WLS_CERT_COUNTRY</td>
<td>'US'</td>
<td>(Optional) Set the Country in Subject of CSR. Default is 'US'.</td>
</tr>
</tbody>
</table>
### 11.6.2 Configuration Options

The Workload Service configuration can be found in `/etc/workload-service/config.yml`:

```yaml
port: 5000
cmstls_cert_digest: <sha384>
postgres:
  dbname: wlsdb
  user: <database username>
  password: <database password>
  hostname: <database IP or hostname>
  port: 5432
  sslmode: false
hvs_api_url: https://<HVS IP or hostname>:8443/mtwilson/v2/
cms_base_url: https://<CMS IP or hostname>:8445/cms/v1/
aas_api_url: https://<AAS IP or hostname>:8444/aas/
subject:
  tls_cert_commonname: WLS TLS Certificate
  organization: INTEL
  country: US
  province: SF
  locality: SC
wls:
  user: <username of service account used by WLS to access other services>>
  password: <password>
  loglevel: info
  key_cache_seconds: 300
```

### 11.6.3 Command-Line Options

The Workload Service supports several command-line commands that can be executed only as the Root user:

```bash
```

---

161
Syntax:

workload-service <command>

11.6.3.1 Help

Available Commands:

- help|--help    Show this help message
- start    Start workload-service
- stop     Stop workload-service
- status   Determine if workload-service is running
- uninstall  [--purge] Uninstall workload-service. --purge option needs to be applied to remove configuration and data files
- setup     Setup workload-service for use

Setup command usage: workload-service <command> [task...]

Available tasks for setup:

- download_ca_cert
  - Download CMS root CA certificate
  - Environment variable CMS_BASE_URL=<url> for CMS API

- download_cert TLS
  - Generates Key pair and CSR, gets it signed from CMS
  - Environment variable CMS_BASE_URL=<url> for CMS API

- url
  - Environment variable Bearer_TOKEN=<token> for authenticating with CMS
  - Environment variable KEY_PATH=<key_path> to override default specified in config
  - Environment variable CERT_PATH=<cert_path> to override default specified in config
  - Environment variable WLS_TLS_CERT_CN=<COMMON NAME> to override default specified in config
  - Environment variable WLS_CERT_ORG=<CERTIFICATE ORGANIZATION> to override default specified in config
  - Environment variable WLS_CERT_COUNTRY=<CERTIFICATE COUNTRY> to override default specified in config
  - Environment variable WLS_CERT_LOCALITY=<CERTIFICATE LOCALITY> to override default specified in config
  - Environment variable WLS_CERT_PROVINCE=<CERTIFICATE PROVINCE> to override default specified in config

- server
  - Setup http server on given port
    - Environment variable WLS_PORT=<port> should be set

- database
  - Setup workload-service database

Required env variables are:

- WLS_DB_HOSTNAME : database host name
- WLS_DB_PORT    : database port number
- WLS_DB_USERNAME : database user name
- WLS_DB_PASSWORD : database password
- WLS_DB : database schema name

hvsconnection  Setup task for setting up the connection to the Host Verification Service (HVS)
   Required env variables are:
   - HVS_URL : HVS URL
   - HVS_USER : HVS API user name
   - HVS_PASSWORD : HVS API password

aasconnection  Setup to create workload service user roles in AAS
   - AAS_API_URL : AAS API URL
   - BEARER_TOKEN : Bearer Token

logs        Setup workload-service log level
   - Environment variable WLS_LOG_LEVEL=<log level> should be set

11.6.3.2  start
Start workload-service

11.6.3.3  stop
Stop workload-service

11.6.3.4  status
Determine if workload-service is running

11.6.3.5  uninstall
   [--purge] Uninstall workload-service. --purge option needs to be applied to remove configuration and data files

11.6.3.6  setup
Setup workload-service for use
   Setup command usage: workload-service <command> [task...]

11.6.3.6.1  download_ca_cert
   - Download CMS root CA certificate
   - Environment variable CMS_BASE_URL=<url> for CMS API url
11.6.3.6.2  download_cert TLS

- Generates Key pair and CSR, gets it signed from CMS
- Environment variable CMS_BASE_URL=<url> for CMS API url
- Environment variable BEARER_TOKEN=<token> for authenticating with CMS
- Environment variable KEY_PATH=<key_path> to override default specified in config
- Environment variable CERT_PATH=<cert_path> to override default specified in config
- Environment variable WLS_TLS_CERT_CN=<COMMON NAME> to override default specified in config
- Environment variable WLS_CERT_ORG=<CERTIFICATE ORGANIZATION> to override default specified in config
- Environment variable WLS_CERT_COUNTRY=<CERTIFICATE COUNTRY> to override default specified in config
- Environment variable WLS_CERT_LOCALITY=<CERTIFICATE LOCALITY> to override default specified in config
- Environment variable WLS_CERT_PROVINCE=<CERTIFICATE PROVINCE> to override default specified in config

11.6.3.6.3  server

Setup http server on given port
- Environment variable WLS_PORT=<port> should be set

11.6.3.6.4  database  Setup workload-service database

Required env variables are:
- WLS_DB_HOSTNAME  : database host name
- WLS_DB_PORT      : database port number
- WLS_DB_USERNAME  : database user name
- WLS_DB_PASSWORD  : database password
- WLS_DB           : database schema name
11.6.3.6.5  hvsconnection

Setup task for setting up the connection to the Host Verification Service (HVS)

Required env variables are:
- HVS_URL : HVS URL
- HVS_USER : HVS API user name
- HVS_PASSWORD : HVS API password

11.6.3.6.6  aasconnection

Setup to create workload service user roles in AAS

- AAS_API_URL : AAS API URL
- BEARER_TOKEN : Bearer Token

11.6.3.6.7  logs

Setup workload-service log level

- Environment variable WLS_LOG_LEVEL=<log level> should be set

11.6.4  Directory Layout

The Workload Service installs by default to /opt/wls with the following folders.

11.7  Key Broker Service

11.7.1  Installation Answer File Options

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERNAME</td>
<td></td>
<td>KBS admin username</td>
</tr>
<tr>
<td>PASSWORD</td>
<td></td>
<td>KBS admin password</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>CMS_BASE_URL</td>
<td>Required for generating TLS certificate</td>
<td></td>
</tr>
<tr>
<td>CMS_TLS_CERT_SHA384</td>
<td>SHA384 digest of CMS TLS certificate</td>
<td></td>
</tr>
<tr>
<td>AAS_API_URL</td>
<td>AAS baseurl</td>
<td></td>
</tr>
<tr>
<td>BEARER_TOKEN</td>
<td>JWT token for installation user</td>
<td></td>
</tr>
<tr>
<td>KMS_HOME</td>
<td>/opt/kms</td>
<td>Application home directory</td>
</tr>
<tr>
<td>KBS_SERVICE_USERNAME</td>
<td>kms</td>
<td>Non-root user to run KMS</td>
</tr>
<tr>
<td>JETTY_PORT</td>
<td>80</td>
<td>The server will listen for HTTP connections on this port</td>
</tr>
<tr>
<td>JETTY_SECURE_PORT</td>
<td>443</td>
<td>The server will listen for HTTPS connections on this port</td>
</tr>
<tr>
<td>KMS_LOG_LEVEL</td>
<td>INFO</td>
<td>Sets the root log level in logback.xml</td>
</tr>
<tr>
<td>KMS_NOSETUP</td>
<td>false</td>
<td>Skips setup during installation if set to true</td>
</tr>
<tr>
<td>ENDPOINT_URL</td>
<td><a href="http://localhost">http://localhost</a></td>
<td>Endpoint to be used in key transfer url</td>
</tr>
<tr>
<td>KEY_MANAGER_PROVIDER</td>
<td>DirectoryKeyManager</td>
<td>Key manager to be used for key management</td>
</tr>
<tr>
<td>KBS_SERVICE_PASSWORD</td>
<td></td>
<td>The master password protects the configuration file and the password vault. It must be set before</td>
</tr>
</tbody>
</table>
installing and before starting the KBS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMS_TLS_CERT_IP</td>
<td>IP addresses to be included in SAN list</td>
</tr>
<tr>
<td>KMS_TLS_CERT_DNS</td>
<td>DNS addresses to be included in SAN list</td>
</tr>
<tr>
<td>BARBICAN_PROJECT_ID</td>
<td>OpenStack Barbican project id</td>
</tr>
<tr>
<td>BARBICAN_ENDPOINT_URL</td>
<td>OpenStack Barbican endpoint url</td>
</tr>
<tr>
<td>BARBICAN_KEYSTONE_PUBLIC_ENDPOINT</td>
<td>OpenStack Keystone endpoint url</td>
</tr>
<tr>
<td>BARBICAN_TENANTNAME</td>
<td>OpenStack Barbican tenant name</td>
</tr>
<tr>
<td>BARBICAN_USERNAME</td>
<td>OpenStack Barbican admin username</td>
</tr>
<tr>
<td>BARBICAN_PASSWORD</td>
<td>OpenStack Barbican admin password</td>
</tr>
</tbody>
</table>

### 11.7.2 Configuration Options

### 11.7.3 Command-Line Options

The Key Broker Service supports several command-line commands that can be executed only as the Root user:

**Syntax:**

```
kms <command>
```

167
### 11.7.3.1 Start
Starts the service

### 11.7.3.2 Stop
Stops the service

### 11.7.3.3 Uninstall
Removes the service

### 11.7.3.4 Version
Displays the version of the service

### 11.7.3.5 setup
Usage: /usr/local/bin/kms setup [-force|--noexec] [task1 task2 ...]
Available setup tasks:
11.7.3.5.1 kms setup jca-security-providers
11.7.3.5.2 kms setup password-vault
11.7.3.5.3 kms setup jetty-ports
11.7.3.5.4 kms setup jetty-tls-keystore
11.7.3.5.5 kms setup shiro-ssl-port
11.7.3.5.6 kms setup notary-key
11.7.3.5.7 kms setup envelope-key
11.7.3.5.8 kms setup storage-key
11.7.3.5.9 kms setup saml-certificates
11.7.3.5.10 kms setup tpm-identity-certificates

11.7.4 Directory Layout

The Verification Service installs by default to /opt/kms with the following folders.

11.7.4.1 Bin
Contains scripts and executable binaries

11.7.4.2 Configuration
Contains configuration files

11.7.4.3 Env
Contains environment details

11.7.4.4 Features

11.7.4.5 Java
Contains Java artifacts
11.7.4.6 Logs
Contains logs. Primary log file is kms.log

11.7.4.7 Repository
Contains the “keys” subdirectory, which is used for storing image encryption keys.

11.7.4.8 Script
Contains additional scripts

11.8 Workload Agent

11.8.1 Installation Answer File Options

11.8.2 Configuration Options

11.8.3 Command-Line Options
Available Commands:

11.8.3.1 Help
wlagent help|--help|--help Show help message

11.8.3.2 setup
wlagent setup [task] Run setup task

11.8.3.2.1 Available Tasks for setup

  SigningKey
Generate a TPM signing key

  BindingKey
Generate a TPM binding key

  RegisterSigningKey
Register a signing key with the host verification service
- Environment variable BEARER_TOKEN=<token> for authenticating with Verification service

**RegisterBindingKey**

Register a binding key with the host verification service

- Environment variable BEARER_TOKEN=<token> for authenticating with Verification service

### 11.8.3.3 start
Start wlagent

### 11.8.3.4 stop
Stop wlagent

### 11.8.3.5 status
Reports the status of wlagent service

### 11.8.3.6 uninstall
Uninstall wlagent

### 11.8.3.7 uninstall --purge
Uninstalls workload agent and deletes the existing configuration directory

### 11.8.3.8 version
Reports the version of the workload agent

### 11.8.4 Directory Layout

The Workload Agent installs by default to /opt/workload-agent with the following folders.

#### 11.8.4.1 Bin
Contains scripts and executable binaries.
11.9 Workload Policy Manager

11.9.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMS_API_URL</td>
<td>https://&lt;IP address or hostname of the KBS&gt;:443/v1/</td>
<td>Required. Defines the baseurl for the Key Broker Service. The WPM uses this URL to request new encryption keys when encrypting images.</td>
</tr>
<tr>
<td>KMS_TLS_SHA384</td>
<td></td>
<td>Required. SHA384 hash of the Key Broker TLS certificate</td>
</tr>
<tr>
<td>CMS_TLS_CERT_SHA384</td>
<td></td>
<td>Required. SHA384 hash of the CMS TLS certificate</td>
</tr>
<tr>
<td>CMS_BASE_URL</td>
<td>https://&lt;IP address or hostname for CMS&gt;:8445/cms/v1/</td>
<td>Required. Defines the base URL for the CMS owned by the image owner. Note that this CMS may be different from the CMS used for other components.</td>
</tr>
<tr>
<td>AAS_API_URL</td>
<td>https://&lt;IP address or hostname for AAS&gt;:8444/aas</td>
<td>Required. Defines the baseurl for the AAS owned by the image owner. Note that this AAS may be different from the AAS used for other components.</td>
</tr>
<tr>
<td>BEARER_TOKEN</td>
<td>&lt;token&gt;</td>
<td>Required; token from CMS with permissions used for installation.</td>
</tr>
<tr>
<td>WPM_WITH_CONTAINER_SECURITY</td>
<td>“yes” or “no”</td>
<td>Optional, defaults to “no.” Defines whether the WPM will support Docker Container encryption. If this is set to Yes, the appropriate prerequisites for Docker Container encryption will be installed. If this is set to “no,” the WPM will not be able to encrypt Docker Container images, and will only be usable to encrypt Virtual Machine images.</td>
</tr>
<tr>
<td>WPM_LOG_LEVEL</td>
<td>INFO (default), DEBUG</td>
<td>Optional; defines the log level for the WPM. Defaults to INFO.</td>
</tr>
<tr>
<td>WPM_PASSWORD</td>
<td></td>
<td>Defines the credentials for the WPM to use to access the KBS</td>
</tr>
<tr>
<td>WPM_USERNAME</td>
<td></td>
<td>Defines the credentials for the WPM to use to access the KBS</td>
</tr>
</tbody>
</table>

11.9.2 Configuration Options

11.9.3 Command-Line Options

The Workload Policy Manager supports several command-line commands that can be executed only as the Root user:

Syntax:
wpm <command>

11.9.3.1 create-image-flavor

Creates a new image flavor and encrypts a source image. Output is the image flavor in JSON format and the encrypted image.

usage: wpm create-image-flavor [-l label] [-i in] [-o out] [-e encout] [-k key]
  -l, --label     image flavor label
  -i, --in        input image file path
  -o, --out       (optional) output image flavor file path
  -e, --encout    (optional) output encrypted image file path
  -k, --key       (optional) existing key ID
                  if not specified, a new key is generated

11.9.3.2 create-container-image-flavor

Used to encrypt Docker container images and generate a container image flavor.

usage: wpm create-container-image-flavor [-i img-name] [-t tag] [-f dockerFile] [-d build-dir] [-k keyId]
  [-e] [-s] [-n notaryServer] [-o out-file]
  -i, --img-name    container image name
  -t, --tag         (optional) container image tag name
  -f, --docker-file (optional) container file path
                   to build the container image
  -d, --build-dir   (optional) build directory to
                   build the container image
  -k, --key-id      (optional) existing key ID
                   if not specified, a new key is generated
  -e, --encryption-required
                   (optional) boolean parameter specifies
                   if container image needs to be encrypted
  -s, --integrity-enforced
                   (optional) boolean parameter specifies
                   if container image should be signed
  -n, --notary-server (optional) specify notary server url
  -o, --out-file    (optional) specify output file path

11.9.3.3 get-container-image-id

11.9.3.4 create-software-flavor

Not currently supported; intended for future functionality.

11.9.3.5 Uninstall

Removes the WPM.
11.9.3.6 --help
Displays help text

11.9.3.7 --version
Displays the WPM version

11.9.3.8 Setup

usage: wpm setup [<tasklist>]
<tasklist>-space separated list of tasks

11.9.3.8.1 wpm setup

11.9.3.8.2 wpm setup CreateEnvelopeKey

11.9.3.8.3 wpm setup RegisterEnvelopeKey

11.9.3.8.4 wpm setup download_ca_cert [--force]
- Download CMS root CA certificate
- Option [--force] overwrites any existing files, and always downloads new root CA cert
- Environment variable CMS_BASE_URL=<url> for CMS API url

11.9.3.8.5 wpm setup download_cert Flavor-Signing [--force]
- Generates Key pair and CSR, gets it signed from CMS
- Option [--force] overwrites any existing files, and always downloads newly signed Flavor Signing cert
- Environment variable CMS_BASE_URL=<url> for CMS API url
- Environment variable Bearer_TOKEN=<token> for authenticating with CMS
- Environment variable KEY_PATH=<key_path> to override default specified in config
- Environment variable CERT_PATH=<cert_path> to override default specified in config
- Environment variable WPM_FLAVOR_SIGN_CERT_CN=<COMMON NAME> to override default specified in config
- Environment variable WPM_CERT_ORG=<CERTIFICATE ORGANIZATION> to override default specified in config
- Environment variable WPM_CERT_COUNTRY=<CERTIFICATE COUNTRY> to override default specified in config
- Environment variable WPM_CERT_LOCALITY=<CERTIFICATE LOCALITY> to override default specified in config
- Environment variable WPM_CERT_PROVINCE=<CERTIFICATE PROVINCE> to override default specified in config
12 Certificate and Key Management

12.1 Host Verification Service Certificates and Keys

The Host Verification Service has several unique certificates not present on other services.

12.1.1 SAML

The SAML Certificate a is used to sign SAML attestation reports, and is itself signed by the Root Certificate. This certificate is unique to the Verification Service.

/opt/mtwilson/configuration/saml.crt

/opt/mtwilson/configuration/saml.crt.pem

/opt/mtwilson/configuration/SAML.jks

The SAML Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

\texttt{mtwilson replace-saml-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem}

This will:

- Replace key pair in /opt/mtwilson/configuration/SAML.jks, alias samlkey1
- Update /opt/mtwilson/configuration/saml.crt with saml DER public key cert
- Update /opt/mtwilson/configuration/saml.crt.pem with saml PEM public key cert
- Update configuration properties:
  - saml.key.password to null
  - saml.certificate.dn
  - saml.issuer
When the SAML certificate is replaced, all hosts will immediately be added to a queue to generate a new attestation report, since the old signing certificate is no longer valid. No service restart is necessary.

If the Integration Hub is being used, the new SAML certificate will need to be imported to the Hub.

### 12.1.2 Asset Tag

The Asset tag Certificate is used to sign all Asset Tag Certificates. This certificate is unique to the Verification Service.

```
/opt/mtwilson/configuration/tag-cacerts.pem
```

The Asset Tag Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-tag-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:

- Replace key pair in database table mw_file (cakey is private and public key pem formatted, cacerts is cert chain)
- Update `/opt/mtwilson/configuration/tag-cacerts.pem` with cert chain
- Update configuration properties:
  - tag.issuer.dn

No service restart is needed. However, all existing Asset Tags will be considered invalid, and will need to be recreated. It is recommended to delete any existing Asset Tag certificates and Flavors, and then recreate and deploy new Tags.

### 12.1.3 Privacy CA

The Privacy CA certificate is used as part of the certificate chain for creating the Attestation Identity Key (AIK) during Trust Agent provisioning. The Privacy CA must be a self-signed certificate. This certificate is unique to the Verification Service.

The Privacy CA certificate is used by Trust Agent nodes during Trust Agent provisioning; if the Privacy CA certificate is changed, all Trust Agent nodes will need to be re-provisioned.

```
/opt/mtwilson/configuration/PrivacyCA.p12
```

```
/opt/mtwilson/configuration/PrivacyCA.pem
```
The Privacy CA Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-pca-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:
- Replace key pair in `/opt/mtwilson/configuration/PrivacyCA.p12`, alias 1
- Update `/opt/mtwilson/configuration/PrivacyCA.pem` with cert
- Update configuration properties:
  - `mtwilson.privacyca.aik.issuer`
  - `mtwilson.privacyca.aik.validity.days`

After the Privacy CA certificate is replaced, all Trust Agent hosts will need to be re-provisioned with a new AIK:

```
tagent setup download-mtwilson-privacy-ca-certificate --force
tagent setup request-aik-certificate --force
tagent restart
```

### 12.1.4 Endorsement CA

The Endorsement CA is a self-signed certificate used during Trust Agent provisioning.

```
/opt/mtwilson/configuration/EndorsementCA.p12
```

```
/opt/mtwilson/configuration/EndorsementCA.pem
```

The Endorsement CA Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-eca-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:
- Replace key pair in `/opt/mtwilson/configuration/EndorsementCA.p12`, alias 1
- Update `/opt/mtwilson/configuration/EndorsementCA.pem` with accepted ECs
- Update configuration properties:
  - `mtwilson.privacyca.ek.issuer`
  - `mtwilson.privacyca.ek.validity.days`
After the Endorsement CA certificate is replaced, all Trust Agent hosts will need to be re-provisioned with a new Endorsement Certificate:

tagent setup request-endorsement-certificate --force

tagent restart

## 12.2 TLS Certificates

TLS certificates for each service are issued by the Certificate Management Service during installation. If the CMS root certificate is changed, or to regenerate the TLS certificate for a given service, use the following commands (note: environment variables will need to be set; typically these are the same variables set in the service installation .env file):

- `<servicename> download_ca_cert`  
  o Download CMS root CA certificate  
  o Environment variable CMS_BASE_URL=<url> for CMS API url

- `<servicename> download_cert TLS`  
  o Generates Key pair and CSR, gets it signed from CMS  
  o Environment variable CMS_BASE_URL=<url> for CMS API url  
  o Environment variable BEARER_TOKEN=<token> for authenticating with CMS  
  o Environment variable KEY_PATH=<key_path> to override default specified in config  
  o Environment variable CERT_PATH=<cert_path> to override default specified in config
# TLS Policies

The Intel Security Libraries Verification Service validates the authenticity of connections through the use of various TLS verification policies.

## 13.1 TLS Policy Types

Intel Security Libraries Verification Service uses six types of TLS policies.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Behavior</th>
<th>Shared</th>
<th>Per-Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>The certificate policy requires one or more trusted certificates or CA certificates and only connects to a peer whose certificate either is a trusted certificate or is signed by a CA that is trusted. This policy type also performs hostname verification. <strong>Note:</strong> The remote server's hostname must be resolvable from the Mt. Wilson server.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Certificate Fingerprint</td>
<td>This policy stores the SHA384 hash of the certificate for validation rather than the entire certificate itself.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Public Key</td>
<td>The public key policy requires one public key parameter and only connects to a peer using that key. This is similar to SSH public key authentication of clients and hosts. Hostname verification is NOT performed when using Public Key TLS policies.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Public Key Fingerprint</td>
<td>This policy stores the SHA384 hash of the public key for validation rather than the public key itself.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TRUST_FIRST_CERTIFICATE</td>
<td>This policy stores the first certificate encountered when connecting to a host, and uses that certificate for all future TLS validation with that host.</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>INSECURE</td>
<td>This policy disables all TLS validation. All connections are accepted regardless of TLS certificates. This policy should be used for troubleshooting and development only, and should never be used in a production environment.</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

## 13.2 Policy Scope

TLS policies can be per-host or shared across multiple hosts.
13.2.1 Per-Host

A per-host TLS policy is an individual, per-host TLS policy. When the host is deleted, its per-host TLS policy is automatically deleted as well.

13.2.2 Shared

A shared TLS policy may be referenced by multiple host records. When a host that referenced a shared TLS policy is deleted, the shared policy continues to exist regardless if there are any remaining hosts that are referencing it. Shared policies must be explicitly deleted by the user.

The Verification Service requires a TLS policy to be defined for any remote host to which it connects. If no TLS policy is defined, or if the TLS information does not match the TLS policy, the connection fails.

13.3 Default Policy Selection

Any shared-scope policy can be defined as the “default” TLS policy for a given Verification Service environment. For example, if all TLS certificates for all hosts in the attestation environment have been signed by the same CA certificate, that CA certificate can be used to create a shared-scope certificate policy, and this same policy could be used to validate all TLS connections with all attested hosts. By configuring this policy as the default TLS policy, the Verification Service uses this specific policy for all hosts unless another policy is specified.

In the Verification Service UI, this mostly means that the default policy is automatically selected from the drop-down when registering hosts. From an API perspective, it means that, when calling a registration API, if no TLS policy is specifically defined in the call, the default TLS policy is used. Using a shared default policy that is valid across all hosts in the attestation environment can greatly simplify TLS policy and host management.

Note: During installation, the only two shared-scope policies that might be available are TRUST_FIRST_CERTIFICATE and INSECURE, and these only if they have actually been enabled. All other policies must be user-created after installation. To define a default TLS policy, edit the mtwilson.properties file and set the value of mtwilson.default.tls.policy.id to either the UUID or the name of the shared-scope TLS policy to be set as the default. Restart Mt. Wilson to affect the change.

13.4 Default TLS Policies

At the time the Verification Service is installed, two TLS policies are created.
13.4.1 TRUST_FIRST_CERTIFICATE

This policy creates a new TLS policy the first time that a new host is registered to the Verification Service, and uses that policy for all future interactions with that host.

13.4.2 INSECURE

This policy turned off all TLS certificate validation entirely (all connections were trusted, regardless of TLS certificates). This policy should only be used for development or troubleshooting, and should never be used in a production environment.

To configure the Verification Service to use TRUST_FIRST_CERTIFICATE as the default TLS Policy (and disallow the use of INSECURE), use the following settings:

```
mtwilson.tls.policy.allow=TRUST_FIRST_CERTIFICATE
mtwilson.default.tls.policy.id=TRUST_FIRST_CERTIFICATE
```

This can be done automatically during installation by setting the following variables in mtwilson.env:

```
export MTW_TLS_POLICY_ALLOW= TRUST_FIRST_CERTIFICATE
export MTW_DEFAULT_TLS_POLICY_ID=TRUST_FIRST_CERTIFICATE
```
14 Uninstallation

This section describes steps used for uninstalling Intel SecL-DC services.

Note: This section does not apply for containerized deployments. To uninstall a containerized deployment, simply shut down the container and delete the persistence volumes.

14.1 Verification Service

To uninstall the Verification Service, run the following command:

mtwilson uninstall

Removes following directories:
1. usr/local/bin/mtwilson
2. $MTWILSON_HOME/bin
3. $MTWILSON_HOME/java
4. $MTWILSON_HOME/features

mtwilson uninstall --purge

Removes following directories:
1. $MTWILSON_HOME /opt/mtwilson
2. $MTWILSON_CONFIGURATION $MTWILSON_HOME/configuration (/opt/mtwilson/configuration)
3. $MTWILSON_LOGS $MTWILSON_HOME/logs (/opt/mtwilson/logs)

The mtwilson uninstall command will not delete any database content. To completely uninstall and delete all database content and user data, run the following:

mtwilson erase-data

mtwilson erase-users --all

mtwilson uninstall --purge

Note: The uninstall command must be issued last, because the uninstall process removes the scripts that execute the other commands, along with all database connectivity info.
14.2 Trust Agent

To uninstall the Trust Agent, run the following command:

tagent uninstall

Backs up the configuration directory and removes all Trust Agent files, except for configuration files which are saved and restored.

Removes following directories:
1. /usr/local/bin/tagent
2. TRUSTAGENT_HOME : /opt/trustagent
3. /opt/tbootxm
4. /var/log/trustagent/measurement.*

**Note:** TPM ownership can be preserved by retaining the TPM owner secret. If the Operating System will also be cleared, Linux systems will also require the /usr/local/var/lib/tpm/system.data file to be preserved. This file must be preserved from after ownership is taken, and then replaced after the OS reload before the Trust Agent attempts to reassert ownership.

If the ownership secret and/or system.data file are not preserved, reinstallation will require clearing TPM ownership.

14.3 Integration Hub

To uninstall the Integration Hub, run the following command:

attestation hub uninstall

Removes the following directories:
1. /usr/local/bin/attestation-hub
2. /usr/bin/attestation-hub
3. /opt/attestation-hub
4. /etc/logrotate.d/attestation-hub

attestation-hub uninstall --purge

Removes the following directories (in addition to directories removed without the --purge option):
1. Drops the database
2. Drops the user
3. Removes attestation hub tenant configuration path
## 15.1 PCR Definitions

### 15.1.1 Microsoft Windows Server 2016 Datacenter

#### 15.1.1.1 TPM 1.2 and 2.0

<table>
<thead>
<tr>
<th>PCR</th>
<th>Measurement Parameters</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR 0</td>
<td>BIOS ROM and Flash Image</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as the PLATFORM Flavor.</td>
<td>All</td>
</tr>
<tr>
<td>PCR 12</td>
<td>Data events and highly volatile events</td>
<td>This PCR measures some of the modules which has boot counters in it. It changes on every boot and resume (Microsoft Windows ONLY; do not use for attestation as the values change on reboot)</td>
<td>Microsoft Windows Server</td>
</tr>
<tr>
<td>PCR 13</td>
<td>Boot Module Details</td>
<td>This PCR remains static except major changes such as kernel module update, different device driver for different OEM servers, etc. (Microsoft Windows ONLY)</td>
<td>Microsoft Windows Server</td>
</tr>
<tr>
<td>PCR 14</td>
<td>Boot Authorities</td>
<td>Used to record the Public keys of authorities that sign OS components. Expected not to change often. (Microsoft Windows ONLY)</td>
<td>Microsoft Windows Server</td>
</tr>
</tbody>
</table>

### 15.1.2 Red Hat Enterprise Linux

#### 15.1.2.1 TPM 2.0

<table>
<thead>
<tr>
<th>PCR</th>
<th>Measurement Parameters</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR 0</td>
<td>BIOS ROM and Flash Image Initial Boot Block (Intel® BootGuard only)</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as the PLATFORM Flavor. (Intel® BootGuard only): Extends measurements based on the Intel® BootGuard profile configuration and production vs non-production ACM flags; ACM signature; BootGuard key manifest hash; Boot Policy Manifest Signature</td>
<td>All</td>
</tr>
<tr>
<td>PCR 7</td>
<td>Intel® BootGuard configuration and profiles</td>
<td>Describes the success of the IBB measurement event.</td>
<td>All (Intel® BootGuard only)</td>
</tr>
</tbody>
</table>
| PCR 17 | ACM | BIOS AC registration information
|       |     | Digest of Processor S-CRTM
|       |     | Digest of Policycontrol
|       |     | Digest of all matching elements used by the policy
|       |     | Digest of STM
|       |     | Digest of Capability field of OsSinitData
|       |     | Digest of MLE
|       |     | For TA hosts, this PCR includes measurements of the OS, InitRD, and UUID. This changes with every install due to InitRD and UUID change. | VMware ESXi
|       |     | Red Hat Enterprise Linux |

| PCR 18 | MLE [Tboot +VMM] | Digest of public key modulus used to verify SINIT signature
|        |                 | Digest of Processor S-CRTM
|        |                 | Digest of Capability field of OSSinitData table
|        |                 | Digest of PolicyControl field of used policy
|        |                 | Digest of LCP | VMware ESXi
|        |                 | Red Hat Enterprise Linux |

| PCR 19 | OS Specific. | For ESXi and Trust Agent hosts, this PCR contains individual measurements of all of the non-Kernel modules.
|        |        | For Linux hosts, this PCR is a measurement of the OS, InitRD, and UUID. | VMware ESXi
|        |        | Red Hat Enterprise Linux |

### 15.1.3 VMWare ESXi

#### 15.1.3.1 TPM 1.2

<table>
<thead>
<tr>
<th>PCR</th>
<th>Measurement Parameters</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR 0</td>
<td>BIOS ROM and Flash Image</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as the PLATFORM Flavor.</td>
<td>All</td>
</tr>
</tbody>
</table>
| PCR 17 | ACM | This PCR measures the SINIT ACM, and is hardware platform-specific. This PCR is part of the PLATFORM Flavor. | VMware ESXi
|        |     | Red Hat Enterprise Linux |
| PCR 18 | MLE [Tboot +VMM] | This PCR measures the tboot and hypervisor version. In ESXi hosts, only the tboot version is measured. | VMware ESXi
|        |     | Red Hat Enterprise Linux |
| PCR 19 | OS Specific. | For ESXi and Trust Agent hosts, this PCR contains individual measurements of all of the non-Kernel modules.
|        |        | For Citrix Xen hosts, this PCR is a measurement of the OS, InitRD, and UUID. | VMware ESXi
|        |        | Red Hat Enterprise Linux |
## PCR Measurement Parameters

<table>
<thead>
<tr>
<th>PCR</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR 20</td>
<td>For ESXi only. VM Kernel and VMK Boot This PCR is used only by ESXi hosts and is blank for all other host types.</td>
<td>VMware ESXi</td>
</tr>
<tr>
<td>PCR 22</td>
<td>Asset Tag This PCR contains the measurement of the SHA1 of the Asset Tag Certificate provisioned to the TPM, if any.</td>
<td>VMware ESXi</td>
</tr>
</tbody>
</table>

### 15.1.3.2 TPM 2.0

VMware supports TPM 2.0 with Intel TXT starting in vSphere 6.7 Update 1. Earlier versions will support TPM 1.2 only.

<table>
<thead>
<tr>
<th>PCR</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR 0</td>
<td>BIOS ROM and Flash Image This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as part of the PLATFORM flavor.</td>
<td>All</td>
</tr>
<tr>
<td>PCR 17</td>
<td>ACM This PCR measures the SINIT ACM, and is hardware platform-specific. This PCR is part of the PLATFORM Flavor.</td>
<td>VMware ESXi, Red Hat Enterprise Linux</td>
</tr>
<tr>
<td>PCR 18</td>
<td>MLE [Tboot +VMM] This PCR measures the tboot and hypervisor version. In ESXi hosts, only the tboot version is measured. This PCR is part of the PLATFORM Flavor.</td>
<td>VMware ESXi, Red Hat Enterprise Linux</td>
</tr>
</tbody>
</table>
| PCR 19 | OS Specific.  
- ESX and Trust Agent — non Kernel modules  
- Citrix Xen — OS  
- + Init RD + UUID For ESXi this PCR contains individual measurements of all of the non-Kernel modules – this includes all of the VIBs installed on the ESXi host. This is part of the OS flavor. Note that two ESXi hosts with the same version of ESXi installed may require different OS flavors if different VIBs are installed. | VMware ESXi, Red Hat Enterprise Linux |
<p>| PCR 20 | For ESXi only. VM Kernel and VMK Boot This PCR is used only by ESXi hosts for some host-specific measurements, and is part of the host-unique flavor. | VMware ESXi                        |
| PCR 22 | Asset Tag Asset Tag is not currently supported for TPM 2.0 with ESXi. | VMware ESXi                        |</p>
<table>
<thead>
<tr>
<th>Platform</th>
<th>TPM</th>
<th>Flavor Type</th>
<th>Rules to be verified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL 1.2</td>
<td>1.2</td>
<td>HARDWARE</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td>For all flavor evaluations, verification of the AIK certificate is needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 17</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 18</td>
<td>Evaluation of PcrEventLogIncludes would not include the host specific module verification. Even though PCR 19 only contains host specific modules, it is still needed in OS flavor for integrity check.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogIncludes rule for PCR 19</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogIntegrity rule for PCR 19</td>
<td></td>
</tr>
<tr>
<td>ASSET_TAG</td>
<td></td>
<td></td>
<td>AssetTagMatches rule</td>
<td>AssetTagMatches rule needs to be updated to verify the key-value pairs after verifying the tag certificate.</td>
</tr>
<tr>
<td>HOST_SPECIFIC</td>
<td></td>
<td></td>
<td>PcrEventLogIncludes rule for PCR 19</td>
<td>Only for the host specific modules. Only PCR 19 data is needed in this flavor.</td>
</tr>
<tr>
<td>RHEL 2.0</td>
<td>2.0</td>
<td>HARDWARE</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td>Evaluation of PcrEventLogIncludes would not include initrd and vmlinuz modules. They would be handled in host_specific flavor. Evaluation of PcrEventLogIntegrity rule would also include OS modules (initrd &amp; vmlinuz)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogIncludes rule for PCR 17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogIntegrity rule for PCR 17</td>
<td></td>
</tr>
<tr>
<td>ASSET_TAG</td>
<td></td>
<td></td>
<td>AssetTagMatches rule</td>
<td></td>
</tr>
<tr>
<td>HOST_SPECIFIC</td>
<td></td>
<td></td>
<td>PcrEventLogIncludes rule for PCR 17 (initrd &amp; vmlinuz)</td>
<td></td>
</tr>
<tr>
<td>VMware ESXi 1.2</td>
<td>PLATFORM</td>
<td></td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 17</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogEqualsExcluding rule for PCR 19 (excludes dynamic modules based on component name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogIntegrity rule for PCR 19</td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td>TPM</td>
<td>Flavor Type</td>
<td>Rules to be verified</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>-------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>VMware ESXi</td>
<td>2.0</td>
<td>PLATFORM</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td>NOT SUPPORTED</td>
</tr>
<tr>
<td>Windows</td>
<td>1.2</td>
<td>ASSET_TAG</td>
<td>PcrMatchesConstant rule for PCR 13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASSET_TAG</td>
<td>AssetTagMatches rule</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>2.0</td>
<td>PLATFORM</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td>PcrMatchesConstant rule for PCR 13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASSET_TAG</td>
<td>AssetTagMatches rule</td>
<td></td>
</tr>
</tbody>
</table>

A.2 **Intel TXT and the Trusted Boot Process**