NIST Interagency Report
NIST IR 8320D ipd

Hardware Enabled Security:

Hardware-Based Confidential Computing

Initial Public Draft

Michael Bartock
Murugiah Souppaya
Jerry Wheeler
Timothy Knoll
Muthukkumaran Ramalingam
Stefano Righi

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8320D.ipd

NATIONAL INSTITUTE OF
STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Interagency Report
NIST IR 8320D ipd

Hardware Enabled Security:
Hardware-Based Confidential Computing

Initial Public Draft

Michael Bartock
Murugiah Souppaya
Computer Security Division
Information Technology
Laboratory

Jerry Wheeler
Timothy Knoll
Intel Corporation

Muthukkumaran Ramalingam
Stefano Righi
AMI

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8320D.ipd

February 2023

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Certain commercial equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

How to Cite this NIST Technical Series Publication:

Bartock M, Souppaya MP, Wheeler J, Knoll T, Ramalingam M, Righi S (2023) Hardware-Enabled Security: Hardware-Based Confidential Computing. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) NIST IR 8320D ipd. https://doi.org/10.6028/NIST.IR.8320D.ipd

Author ORCID iDs

Michael Bartock: 0000-0003-0875-4555
Murugiah Souppaya: 0000-0002-8055-8527

Public Comment Period
February 23, 2023 - April 9, 2023

Submit Comments
hwsec@nist.gov

National Institute of Standards and Technology
Attn: Applied Cybersecurity Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 2000) Gaithersburg, MD 20899-2000

All comments are subject to release under the Freedom of Information Act (FOIA).

Abstract

Organizations employ a growing volume of machine identities, often numbering in the thousands or millions per organization. Machine identities, such as secret cryptographic keys, can be used to identify which policies need to be enforced for each machine. Centralized management of machine identities helps streamline policy implementation across devices, workloads, and environments. However, the lack of protection for sensitive data in use (e.g., machine identities in memory) puts it at risk. This report presents an effective approach for overcoming security challenges associated with creating, managing, and protecting machine identities throughout their lifecycle. It describes a proof-of-concept implementation, a prototype, that addresses those challenges by using hardware-based confidential computing. The report is intended to be a blueprint or template that the general security community can use to validate and utilize the described implementation.

Keywords

confidential computing; cryptographic key; hardware-enabled security; hardware security module (HSM); machine identity; machine identity management; trusted execution environment (TEE)

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Audience

The primary audiences for this report are security professionals, such as security engineers and architects; system administrators and other information technology (IT) professionals responsible for securing physical or virtual platforms; and hardware, firmware, and software developers who may be able to leverage hardware-enabled security techniques and technologies, particularly hardware-based confidential computing, to improve machine identity management and protection.

Trademark Information

Call for Patent Claims

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

    i. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or

    ii. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: hwsec@nist.gov

Table of Contents

List of Figures

1. Introduction

1.1. Purpose and Scope

The purpose of this report is to describe an effective approach for managing machine identities so that they are protected from malware and other security-related vulnerabilities. This report first explains selected security challenges in creating, managing, and protecting machine identities throughout their lifecycle. It then describes a proof-of-concept implementation, a prototype, that was designed to address those challenges by using hardware-based confidential computing. The report provides sufficient details about the prototype implementation so that organizations can reproduce it if desired. The report is intended to be a blueprint or template that can be used by the general security community to validate and utilize the described implementation.

The prototype implementation presented in this report is only one possible way to solve the security challenges. It is not intended to preclude the use of other products, services, techniques, etc., that can also solve the problem adequately, nor is it intended to preclude the use of any cloud products or services not specifically mentioned in this report.

This report builds upon the terminology and concepts described in NIST Interagency or Internal Report (IR) 8320, *Hardware-Enabled Security: Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases* [IR8320]. Reading that report is a prerequisite for reading this publication because it explains the concepts and defines key terminology used in this publication.

1.2. Terminology

For consistency with related NIST reports, this report uses the following definitions for trust-related terms:

- **Trust**: "The confidence one element has in another that the second element will behave as expected." [Polydys]
- **Trusted**: An element that another element relies upon to fulfill critical requirements on its behalf.

1.3. Document Structure

This document is organized into the following sections and appendices:

- Section 2 discusses security challenges associated with creating, managing, and protecting machine identities.
- Sections 3, 4, and 5 describe the stages of the prototype implementation:
    - Stage 0: performing enterprise machine identity management
    - Stage 1: protecting secret keys in-use by utilizing hardware-based confidential computing

  - Stage 2: bringing together machine identity management and protection of secret keys in-use

- Appendix A provides an overview of the high-level hardware architecture of the prototype implementation.

- Appendix B contains supplementary information provided by AMI describing the components and the steps needed to set up the prototype for managing machine identities.

- Appendix C contains supplementary information provided by Intel describing the components and the steps needed to set up the prototype for enabling hardware components for confidential computing with trusted execution enclaves.

- Appendix D contains supplementary information explaining how the components are integrated with each other to provide runtime protection of machine identities.

- Appendix E lists and defines acronyms and other abbreviations used in the document.

2. Challenges with Creating, Managing, and Protecting Machine Identities

Organizations employ a growing volume of machine identities, often numbering in the thousands or millions per organization. This demands centralized management. The centralized management of machine identities helps streamline policy implementation across devices, workloads, and environments. Proper policy management helps machine identities do their job of securing communication and preventing unauthorized access effectively.

NIST IR 8320C, *Hardware-Enabled Security: Machine Identity Management and Protection* [IR8320C] provides an overview of challenges organizations may face when using machine identities, as well as techniques to improve the security of cloud computing and accelerate the adoption of cloud computing technologies by establishing a hardware-based trusted boundary for confidential computing enclaves. Refer to Sec. 2 of IR 8320C for additional details on challenges with protecting machine identities.

The ultimate goal is to be able to use “trust” as a boundary for hardware-based confidential computing to protect in-use machine identities. This goal is dependent on smaller prerequisite goals described as *stages*, which can be thought of as requirements that the solution must meet.

- **Stage 0: Enterprise Machine Identity Management.** Security and automation for all machine identities in the organization should be a priority. A proper, enterprise-wide machine identity management strategy enables security teams to keep up with the rapid growth of machine identities, while also allowing the organization to keep scaling securely. The key components of a typical enterprise-grade machine identity management solution are described in Sec. 3.

- **Stage 1: Secret Key In-Use Protection with Hardware-Based Confidential Computing**. The confidential computing paradigm can be used to protect secret keys in-use in dynamic environments. Section 4 describes the primary components of a hardware-based confidential computing environment and illustrates a reference architecture demonstrating how its components interact.

- **Stage 2: Machine Identity Management and End-to-End Protection**. Stage 0 discusses how a machine identity can be managed and Stage 1 describes how sensitive information is protected in use in conjunction with hardware-based confidential computing. Stage 2 is about the integration of the two so that machine identity management enables the prerequisites for confidential computing to be leveraged when the secret key is used at runtime. Section 5 describes how these components can be composed together to provide end-to-end protection for machine identities.

Utilizing hardware-enabled security features, the prototype in this document strives to provide the following capabilities:

- Centralized control and visibility of all machine identities
- Machine identities as secure as possible in all major states: at rest, in transit, and in use in random access memory (RAM)
- Strong access control for different types of machine identities in the software development lifecycle and DevOps pipeline
- Machine identity deployment and use in DevOps processes, striving to be as secure as possible

3. Stage 0: Enterprise Machine Identity Management

This section describes stage 0 of the prototype implementation: enterprise machine identity management.

The foundation of machine identity management is built around the ability to achieve three important capabilities: visibility, intelligence, and automation. These capabilities must be available across all machine identities used by organizations today, and they should also be architected to support capabilities that organizations may use in the future. Managing machine identities in modern organizations is an extremely complex task that involves multiple teams, software products, and platforms with highly efficient coordination between them. An effective and efficient machine identity management platform should be architected to integrate with many other software and systems that are part of machine identities' lifecycles.

Figure 1 details a Stage-0 implementation of a typical enterprise-grade machine identity management solution. The major functional components include the following, with the numbers corresponding to those shown in Fig. 1:

1. Inventory/Discovery
2. Reporting/Analysis
3. Enforce Policy
4. Assign Roles
5. Automate Lifecycle

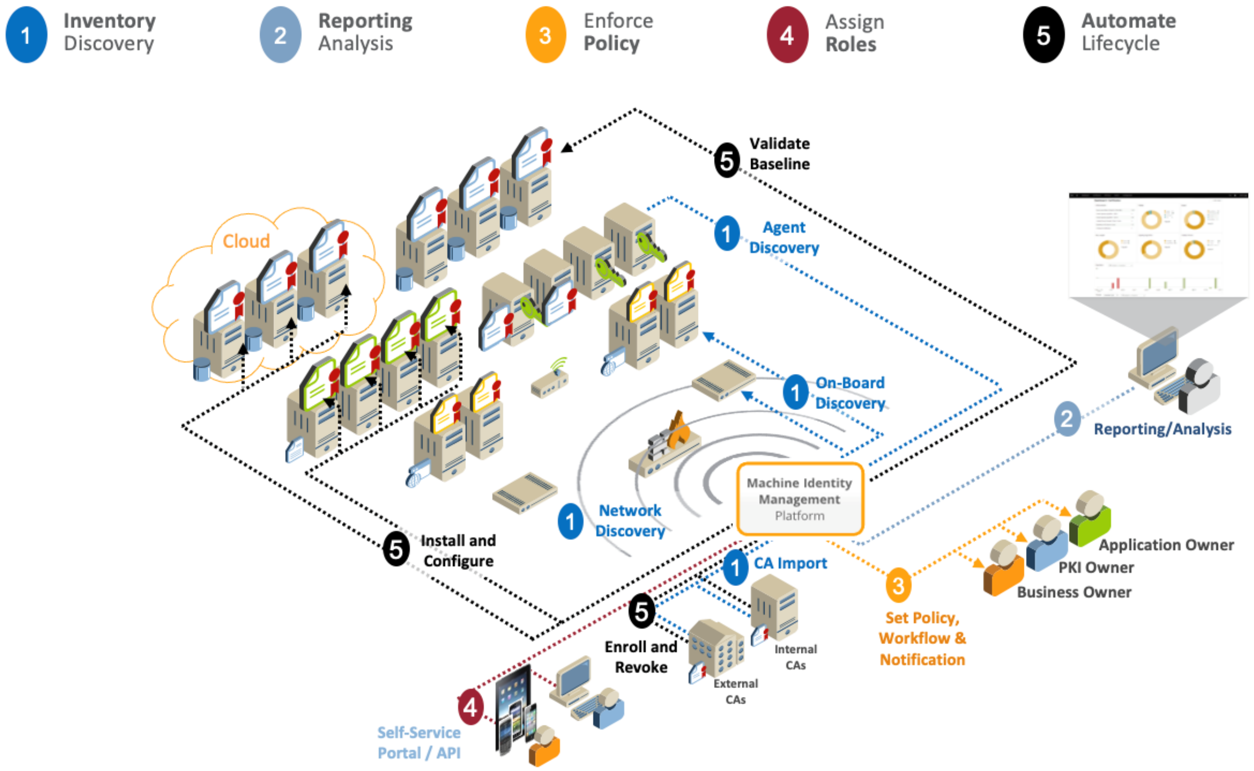


Fig. 1. Stage 0 Implementation: Typical Enterprise-Grade Machine Identity Management

For more detailed information and the solution architecture for Stage 0, please refer to Sec. 3 of IR 8320C [IR8320C].

4. Stage 1: Secret Key In-Use Protection with Hardware-Based Confidential Computing

This section describes Stage 1 of the prototype implementation: protecting secret keys in-use with hardware-based confidential computing.

Mechanisms to protect secret keys in-use exist. An attached or network-based hardware security module (HSM) performs cryptographic processing inside the HSM[1] where the private key is stored. Therefore, loading the key into RAM is not necessary. However, while this works in some deployments, it's not suited for dynamic and multi-tenant environments such as public or private cloud and edge. In these environments, workloads can get scheduled on any host and using an HSM has additional operational and performance costs. A solution that works in these environments is desirable. This means a solution that does not require additional hardware, can scale if needed and, ideally, uses software configuration and deployment paradigms. The solution described in this document uses confidential computing to protect keys in-use. Confidential computing uses trusted execution environments (TEEs) to protect secrets from other software running on the host, including privileged software like the operating system (OS), hypervisor, and firmware. Software that operates on the secrets also runs in the TEE so that secrets never need to get loaded into regular RAM. TEEs provide isolated areas of execution.

---

[1] See Sec. 7.5, "Protecting Keys and Secrets" in NIST IR 8320 [IR8320].

Programmable TEE implementations may support *attestability*, the ability for a TEE to "provide *evidence* or *measurements* of its origin and current state, so that the evidence can be verified by another party and—programmatically or manually—it can decide whether to trust code running in the TEE. It is typically important that such evidence is signed by hardware that can be vouched for by a manufacturer, so that the party checking the evidence has strong assurances that it was not generated by malware or other unauthorized parties." [ConfCC] The evidence can contain the public key part of an ephemeral public/private key pair generated inside the TEE.[2] The *relying party* can wrap secrets with the TEE public key[3] before sharing them with the TEE. Considerations such as the freshness of the evidence and protection against replay attacks are TEE technology-dependent. For more detailed information on this solution and the use of TEE, please refer to Sec. 4 of IR 8320C [IR8320C].

Fig. 2 shows a detailed view of the interactions between the workload on the host and the TEE. It also shows the transfer of the private key from the network HSM. Components in Fig. 2 include the client, workload, TEE adapter, TEE, and TEE attestation and network HSM proxy.

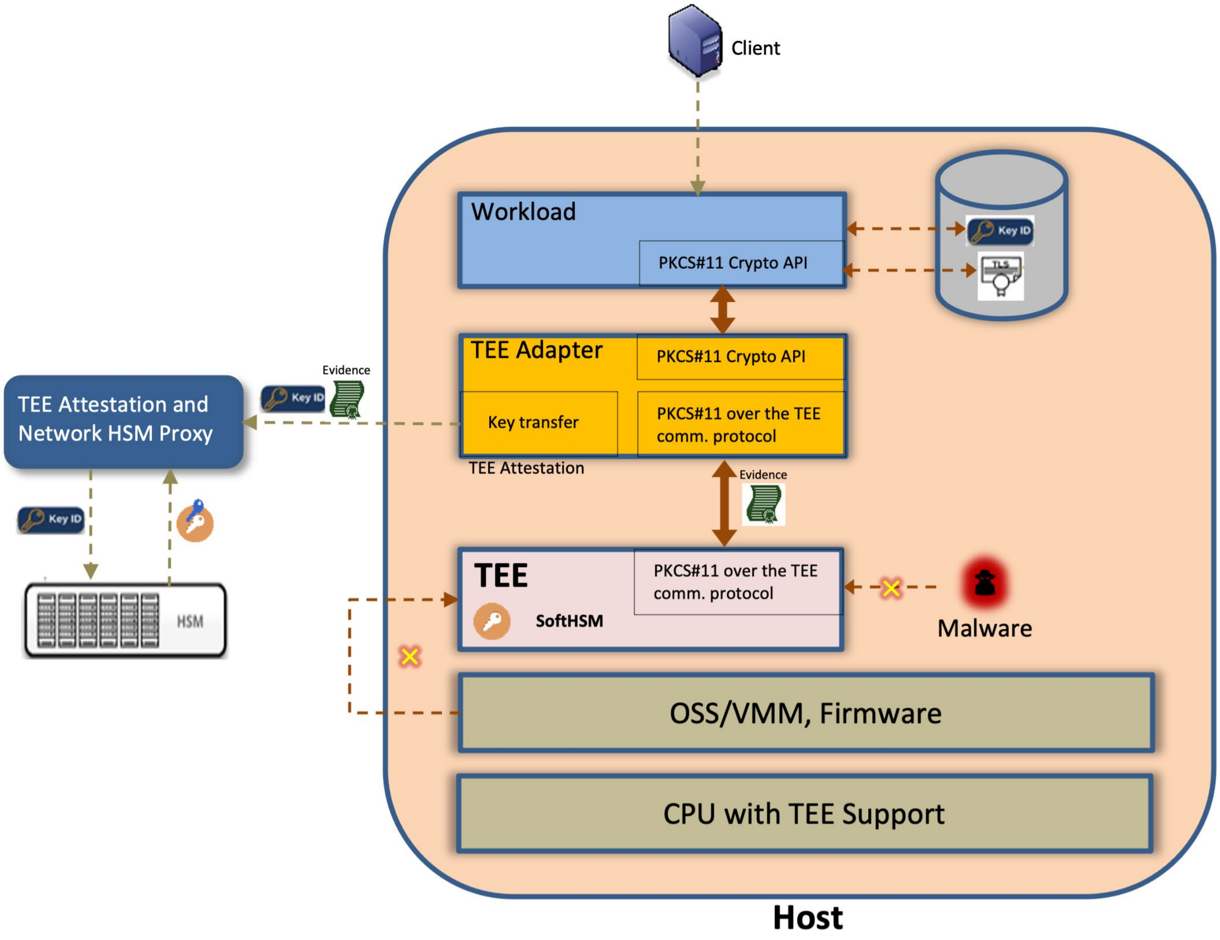


Fig. 2. Private Key Protection Flows

---

[2] The public key could also be communicated to the relying party separately and its hash included in the evidence. By checking that the hash of the public key and the hash in the evidence match, the relying party ensures that the public key has been generated inside a TEE.

[3] This can be done in two steps. First, a Symmetric Wrapping Key (SWK) is generated by the relying party. The SWK is then wrapped with the TEE public key and sent to the TEE. The relying party can then share secrets with the TEE after wrapping them with the SWK.

For detailed information about the solution overview and the interaction of its components, please refer to Sec. 4 of IR 8320C [IR8320C].

5. Stage 2: Machine Identity Management and End-to-End Protection

This section describes Stage 2 of the prototype implementation, which brings together the Stage 0 and Stage 1 prototypes.

In-use secret key protection with hardware-based confidential computing provides a level of protection that is not available from traditional machine identity management solutions. In dynamic and multi-tenant environments such as public or private cloud and edge, secret key protection typically relies on software controls. Software controls can be circumvented by malicious agents because of vulnerabilities in the software, a malicious administrator, or poor operational procedures. On the other hand, confidential computing protects sensitive data such as secret keys with hardware-based mechanisms that are supported by the CPU. This allows the hardware-based protection of secret keys.

Fig. 3 shows the high-level architecture of the prototype. There are two distinct workflows in the figure: the configuration and provisioning flows are depicted by the gray dashed lines, and the runtime flows are depicted by the green dotted lines.

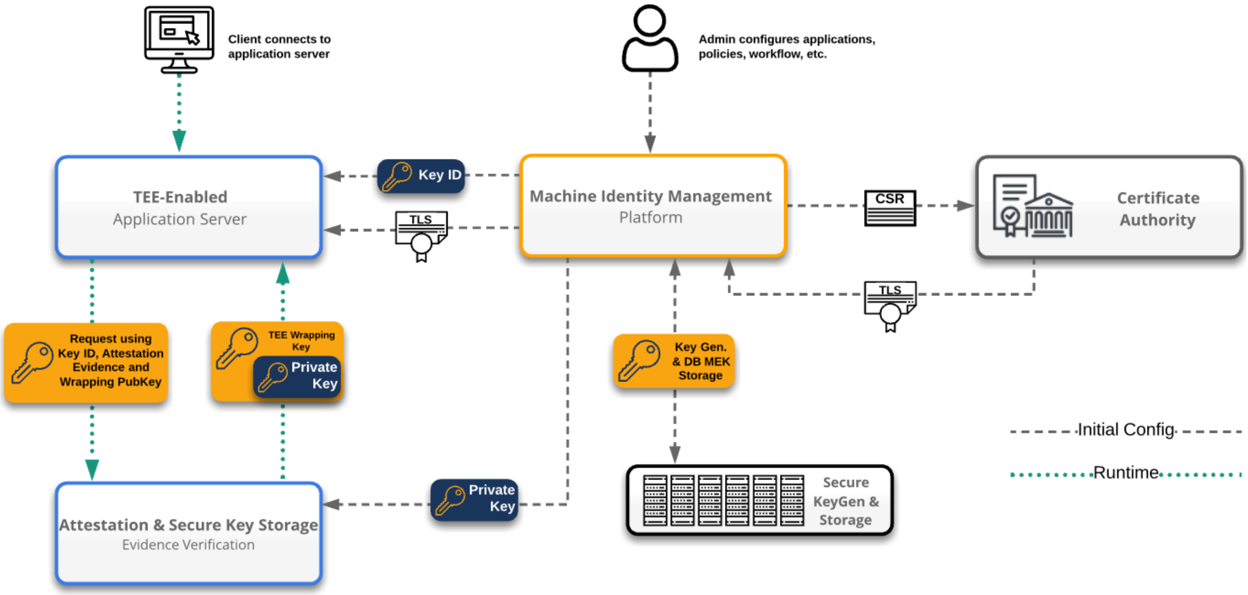


Fig. 3. High-Level Prototype Architecture

Please refer to Sec. 5 of IR 8320C [IR8320C] for the detailed steps for the configuration and provisioning flows and the runtime flows.

References

[ConfCC]	Confidential Computing Consortium (2021) A Technical Analysis of Confidential Computing. https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/CCC-Tech-Analysis-Confidential-Computing-V1.pdf
[IR8320]	Bartock M, Souppaya M, Savino R, Knoll T, Shetty U, Cherfaoui M, Yeluri R, Malhotra A, Banks D, Jordan M, Pendarakis D, Rao JR, Romness P, Scarfone KA (2022) Hardware-Enabled Security: Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) 8320. https://doi.org/10.6028/NIST.IR.8320
[IR8320C]	Bartock M, Souppaya M, Cherfaoui M, Xie J, Cleary P (2022) Hardware-Enabled Security: Machine Identity Management and Protection. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) 8320C. https://doi.org/10.6028/NIST.IR.8320C.ipd
[Polydys]	Polydys ML, Wisseman S (2009) Software Assurance in Acquisition: Mitigating Risks to the Enterprise. https://apps.dtic.mil/dtic/tr/fulltext/u2/a495389.pdf

Appendix A. Hardware Architecture

This appendix provides an overview of the high-level hardware architecture of the prototype implementation.

The prototype implementation is comprised of three servers that reside in geographically separate locations. Two of the servers, the administration and lifecycle management components, are in a NIST lab connected to an Intel lab via an IPsec virtual private network (VPN). The administration and lifecycle management servers deployed as virtual machines (VMs) in the NIST lab are:

1. Red Hat Enterprise Linux (RHEL) 8.5 as the Kubernetes control plane node
2. RHEL 8.5 as the AMI® Trusted Environment (TruE®) services node

The third server is in the Intel lab. It is running RHEL and it has an Intel® Software Guard Extension (SGX®) enabled chipset to protect key material while running as an AMI TruE managed node.

The prototype implementation network is a flat management network for the AMI components, Kubernetes control plane node, and Intel compute server. Fig. 4 shows the high-level architecture of how the three servers in the prototype are connected.

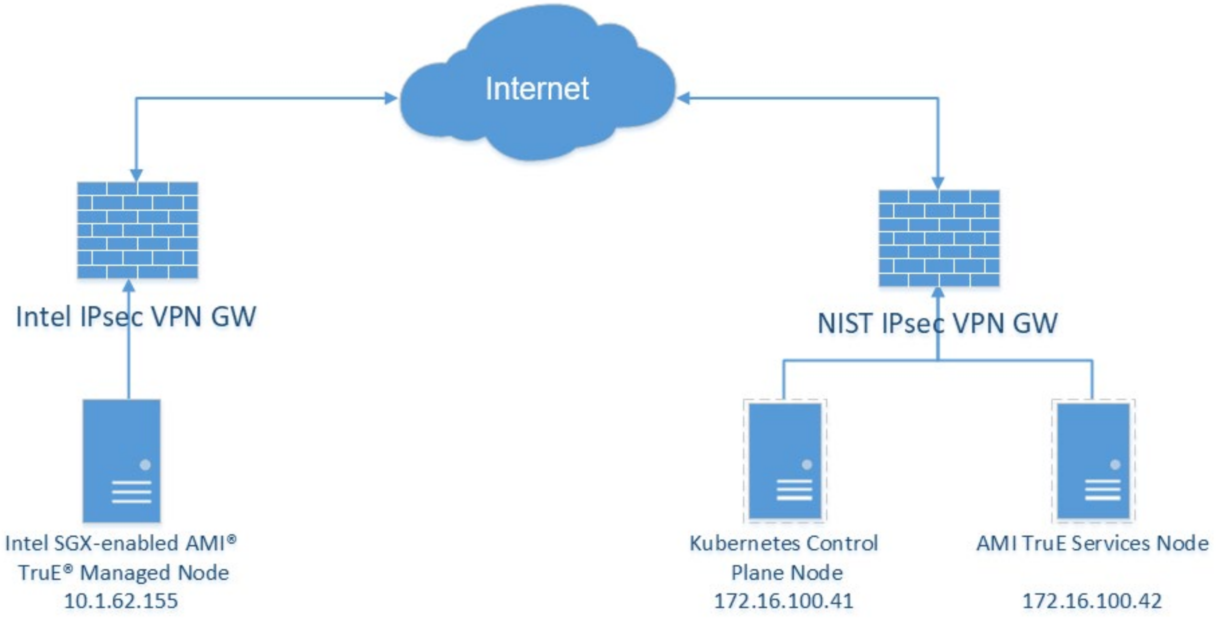


Fig. 4. Prototype Architecture

Appendix B provides additional details for installing and configuring the AMI TruE components of this prototype. Appendix C explains how to enable the Intel SGX feature and describes how it provides protection for sensitive information.

Appendix B. AMI TruE Machine Identity Management Implementation

This appendix contains supplementary information describing the components and the steps needed to set up the prototype implementation for AMI TruE.

B.1. Hardware and Software Requirements

This section explains the hardware and software requirements for AMI TruE installation. AMI TruE services are released as docker containers and require a Kubernetes control plane node and one or more Kubernetes worker nodes.

Deployment Model

Typically, AMI TruE uses a three-node deployment model:

1. Kubernetes control plane node. It can be a system or VM with these hardware and software components:
    - Kubernetes control plane
    - Docker local registry
    - Ansible controller
    - Network File System (NFS) server for Kubernetes
2. AMI TruE services node. It can be a system or VM with the given hardware and software requirements. The services node is configured as a Kubernetes worker node and runs all AMI TruE services workloads. It includes the following components:
    - AMI TruE core services
    - AMI TruE platform security services
3. AMI TruE managed node(s). These are the systems that are deployed in data center or edge infrastructure. AMI TruE requires at least one managed system in the cluster. Note: The RHEL version should be the same across all the nodes connected to the AMI TruE cluster.

General System Requirements

The following are general requirements for all nodes used for AMI TruE deployment:

- Internet connectivity is required for installation.
- All nodes have their clocks synchronized.
- Each node has a unique hostname.
- RHEL systems should have a valid subscription. If not, create a free account from this link and run the command below it:

  ```
  # subscription-manager register
  ```

  Input your username and password when prompted.

  ```
  # subscription-manager attach --auto
  ```

The minimum hardware and software requirements for all types of nodes are given below. Note that worker nodes and managed nodes may require additional hardware based on the number of workloads they handle.

- Processor: 4-core 2.66 GHz CPU
- Memory: 16 GB
- Disk space: 200 GB
- Single network interface with IPv4 network configured
- Operating system: RHEL 8.5, 64-bit
- Latest updates installed

BIOS Prerequisites

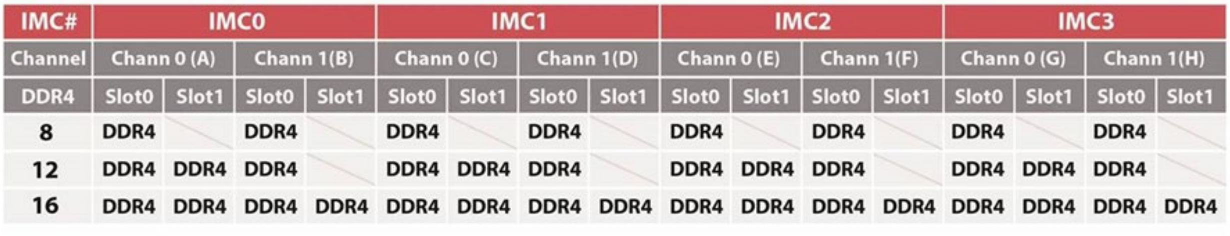
The Basic Input/Output System (BIOS) prerequisites for Intel SGX agents are:

- Intel SGX enabled
- Data Center Attestation Primitives (DCAP) driver signing required

If an SGX agent is installed on the same system with Intel Trusted Execution Technology (TXT) and Unified Extensible Firmware Interface (UEFI) SecureBoot enabled, DCAP driver signing is required.

Memory DIMM Population Requirements

3rd Generation Intel Xeon Scalable processors have four Integrated Memory Controllers (iMCs). Each iMC has two Double Data Rate (DDR) channels and each channel supports two DDR4 Dual In-Line Memory Modules (DIMMs), so one processor can have a maximum of 16 DDR4 DIMMs. These processors only support the SGX feature for the specific DIMM configurations (that is, the exact DDR channels and slots of each processor) shown in Fig. 5. If different DIMMs are populated in the system, the populated DIMMs must be symmetric between {iMC0, iMC1} and {iMC2, iMC3}, and the populated DIMMs must be identical between socket 1 and socket 2 if two processors are installed. Memory mirroring is not supported and must be disabled.



IMC#	IMC0				IMC1				IMC2				IMC3			
Channel	Chann 0 (A)		Chann 1(B)		Chann 0 (C)		Chann 1(D)		Chann 0 (E)		Chann 1(F)		Chann 0 (G)		Chann 1(H)	
DDR4	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1
8	DDR4		DDR4		DDR4		DDR4		DDR4		DDR4		DDR4		DDR4	
12	DDR4	DDR4	DDR4		DDR4	DDR4	DDR4		DDR4	DDR4	DDR4		DDR4	DDR4	DDR4	
16	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4

Fig. 5. Intel SGX-Required DIMM Configurations

Browser Requirements

AMI TruE provides an HTML5-based intuitive web user interface. It’s recommended to use the latest version of the Chrome, Firefox, Opera, or Safari browser.

B.2. AMI TruE Deployment

AMI TruE core services are deployed as containers in the Kubernetes cluster. Please refer to the AMI TruE Quick Start Guide that comes with the release for the deployment procedures. It provides step-by-step details on the pre-configurations required, installation script configurations, and command-line options for deploying the core services. The same guide also has a troubleshooting section for handling typical deployment issues.

The core services deployment includes the following steps:

- Update deployment configurations that include any prerequisites.
- Set up the NFS share path.
- Update installation configurations.
- Run the setup scripts and wait for the deployment to complete.

AMI TruE Platform Security services are deployed as containers in the Kubernetes cluster. Please refer to the AMI TruE Quick Start Guide for the detailed deployment procedures.

The steps to be followed include the following:

- Extract the platform security artifacts.
- Update the install configurations.
- Update the cloud service provider (CSP) environment configurations.
- Update the enterprise environment configurations (optional).
- Run the setup scripts and wait for the deployment to complete.

The **AMI TruE platform security agent** needs to be installed on the servers to be managed. Please refer to the AMI TruE Quick Start Guide for detailed deployment procedures.

The steps to be followed include the following:

- Update the server role configurations.
- Update the install configurations.
- Set up the Kubernetes workers for the appropriate server role.

The Kubernetes control plane node will then launch the appropriate security agents on the target system.

B.3. Platform Security Services Configuration

The web user interface (UI) is launched with a compatible browser by accessing

https://<host>:30567/WEBAPPS/True

where <host> is the IP address or host name of the installation. Upon a successful connection, the login dialog is launched in the browser window.

1. Type the user credentials in the Username and Password textboxes in the Login Window and click the **Log In** button. The default user credentials are Administrator/superuser.

Users with Administrator privileges will have access to all pages in the web UI, whereas other users will only be able to view the Dashboard. Attempts to navigate to other pages or bookmarks without Administrator privileges will result in an error indicating that the user may not have permission to view them.

2. After a successful login, the page displays the Dashboard by default, which displays telemetry of major component resource collections and their status.
3. Click **Log Out** in the top right corner of the UI. Click **Yes** in the confirmation box to log out, or click **Cancel** to remain logged in.

Configuration is essential after installing platform security services. Ensure that the settings correctly reflect the details of the platform services installed and running. Use the web UI (**Security > Configurations > General Configuration**) for configuration. Platform services installed on a single machine with the default environment file require these steps to be performed:

1. Click **Configure IP Address.**
2. Set the IP address of the single system with all platform security services installed. This will set the given IP address for all services.
3. If any default configuration values need changed:
    a. Select an entry to be modified.
    b. Click **Edit**.
    c. A pop-up dialog listing all settings related to the given service/module is listed. Input the details to be modified.
    d. Click **Save.**

Refer to the AMI TruE Quick Start Guide for any additional security configurations required.

B.4. Uninstallation

Refer to the AMI TruE Quick Start Guide for detailed steps on the uninstallation and cleanup of all components and services installed for AMI TruE.

Appendix C. Intel In-Use Secret Key Protection Implementation

This appendix contains supplementary information describing the components and the steps needed to set up the prototype implementation for enabling hardware components for Intel-based confidential computing.

The prototype uses Intel SGX as the confidential computing technology to help protect secret keys in-use. Intel SGX uses hardware-based memory encryption to isolate specific application code and data in memory. Intel SGX allows user-level code and data to run in private regions of memory, called *enclaves* (Intel SGX enclaves are TEEs). Enclaves are designed to be protected from other workloads, including those running at higher privilege levels. Intel SGX enclaves are loaded by workloads as shared libraries. The communication between a workload and an Intel SGX enclave uses dedicated Intel instructions called eCalls. The Intel SGX enclave can invoke external code using dedicated Intel instructions called oCalls. Fig. 6 shows the isolation of Intel SGX enclaves in a host.

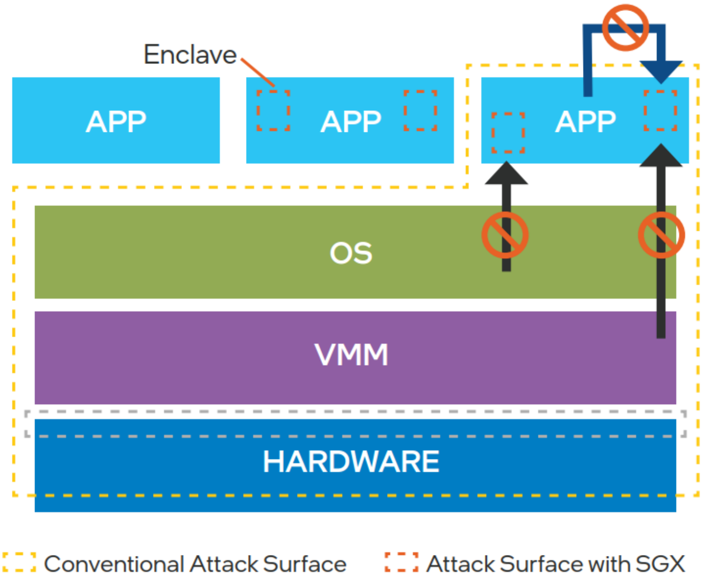


Fig. 6. Intel SGX Enclave

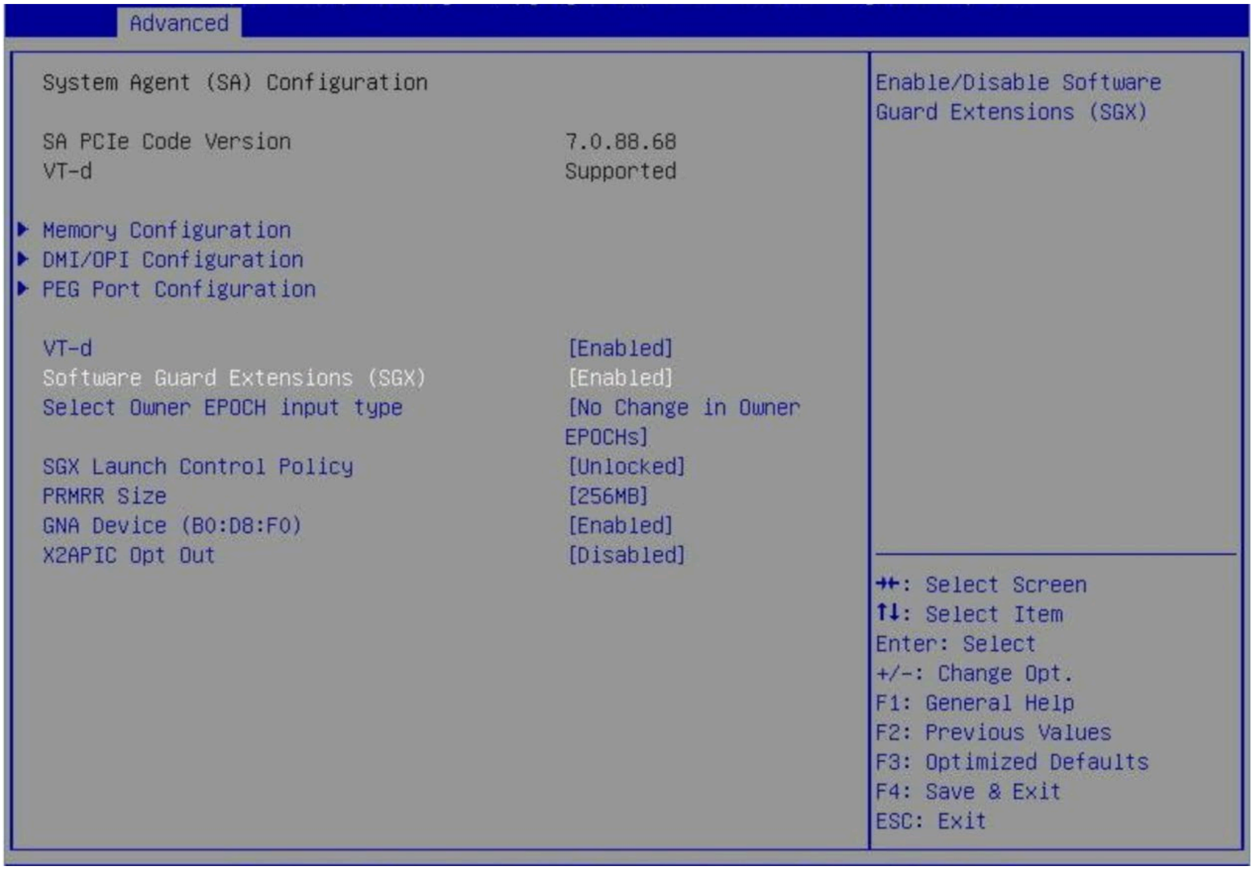
Intel SGX attestation allows a remote relying party to verify that an SGX enclave is genuine. This is achieved by generating enclave attributes using the Intel SGX software development kit (SDK) during the enclave build time. Intel SGX attributes include the enclave signer (MRSigner), the measurement (MREnclave, a fingerprint of the enclave code and initial data), and the ID. At runtime, a remote-relying party can request the generation of evidence (called a *quote* in Intel SGX) containing these same attributes and compare them against those generated by the SDK. An Intel SGX quote also contains the patch levels of the firmware and the Intel SGX supporting software, which the relying party can use to determine if the Intel SGX enclave can be trusted. An Intel SGX quote also contains any data that the enclave wants to share with the relying party. Intel SGX quotes are signed by a verifiable Intel key, so the relying party has the assurance that the attributes' values are authentic.

To enable the remote attestation of Intel SGX enclaves, the host must register to Intel online services and get provisioned with an Intel SGX signing certificate called a *provisioning* *certification key (PCK) certificate*. This must be completed before Intel SGX enclaves are loaded on the host.

Intel Secure Key Caching (SKC) is an implementation of the private key protection in-use using Intel SGX. SKC is a library that wraps an implementation of the PKCS#11 (Public Key Cryptography Standards) interface in an Intel SGX enclave. When a workload requests a key via its PKCS#11 URI, SKC retrieves the key from a remote key management system (KMS) after attestation. Intel SKC is open source: https://github.com/intel-secl/docs/blob/master/README.md#secure-key-caching.

The prototype has been implemented using an Intel Mehlow (E3) Server procured from Supermicro, which is Intel SGX-enabled. The following steps illustrate how to enable SGX on the Supermicro Mehlow server in the BIOS:

1. From the first screen in the BIOS, choose **Enter Setup**.
2. Under the **Advanced** tab, select **Chipset Configuration**.
3. Next, select **System Agent (SA) Configuration**.
4. Finally, enable Intel SGX as shown in Fig. 7.



```
System Agent (SA) Configuration

SA PCIe Code Version                      7.0.88.68
VT-d                                      Supported

▶ Memory Configuration
▶ DMI/OPI Configuration
▶ PEG Port Configuration

VT-d                                      [Enabled]
Software Guard Extensions (SGX)           [Enabled]
Select Owner EPOCH input type             [No Change in Owner
                                          EPOCHs]

SGX Launch Control Policy                 [Unlocked]
PRMRR Size                                [256MB]
GNA Device (B0:D8:F0)                     [Enabled]
X2APIC Opt Out                            [Disabled]
```

Enable/Disable Software Guard Extensions (SGX)

Fig. 7. Enable SGX in BIOS

Refer to the vendor specifications and Intel SGX configuration steps if the server is procured from another vendor.

The prototype can also work on Intel Xeon Scalable Processor (SP) based platforms. Intel SGX configuration for these platforms is detailed in https://cdrdv2.intel.com/v1/dl/getContent/632236 (Intel Developer Zone [IDZ] account required).

SGX Integration Requirements

SGX integration requires registering a token in the Intel Platform Security Services portal. To get the token value for **INTEL_PROVISIONING_SERVER_API_KEY_SANDBOX**, follow these steps:

1. Visit https://api.portal.trustedservices.intel.com/products and click "create a new IDZ account."
2. After account creation, return to the link in the previous step and sign in with your new account.
3. Visit the Intel SGX provisioning certification service.
4. Click **Subscribe**, then **Add subscription**.
5. Collect the primary key by clicking **Show**.

SGX integration also requires BIOS settings such as the following to be updated. Note that these are sample BIOS settings; settings may be different from different vendors.

- **Socket Configuration** > **Processor Configuration** > **Total Memory Encryption** > **Enable**
- **Socket Configuration** > **Common RefCode Configuration** > **UMA-Based Clustering** > **Disable**
- **Socket Configuration** > **Processor Configuration** > **SW Guard Extensions (SGX)** > **Factory Reset**
- **Socket Configuration** > **Processor Configuration** > **SW Guard Extensions (SGX)** > **Enable**
- **Socket Configuration** > **Processor Configuration** > **SGX Packet Info In-band** > **Enable**
- **Socket Configuration** > **Processor Configuration** > **Processor DFx Configuration** > **SGX Registration Server** > **Auto**

Appendix D. Machine Identity Runtime Protection and Confidential Computing Integration

This appendix contains supplementary information explaining how the components are integrated with each other to provide runtime protection of machine identities.

D.1. Solution Overview

Machine identity runtime protection leverages the Intel SGX Attestation Infrastructure to support the SKC use case. SKC provides key protection at rest and in-use using Intel SGX. Intel SGX implements the TEE paradigm.

Using the SKC Client – a set of libraries – applications can retrieve keys from the Intel Security Libraries for Datacenter (SecL-DC) Key Broker Service (KBS) and load them to an Intel SGX-protected memory (called an *Intel SGX enclave*) in the application memory space. KBS performs the Intel SGX enclave attestation to ensure that the application will store the keys in a genuine Intel SGX enclave. The attestation involves KBS verification of a signed Intel SGX quote generated by the SKC Client. The Intel SGX quote contains the hash of the public key of an enclave-generated RSA key pair.

Application keys are wrapped with a Symmetric Wrapping Key (SWK) by KBS prior to transferring to the Intel SGX enclave. The SWK is generated by KBS and wrapped with the enclave RSA public key, which ensures that the SWK is only known to KBS and the enclave. Consequently, application keys are protected from infrastructure administrators, malicious applications, and compromised hardware/BIOS/OS/VMM. SKC does not require refactoring the application because it supports a standard PKCS#11 interface.

D.2. Solution Architecture

Fig. 8 shows how the components of the solution interact with each other in the step-by-step process to launch NGINX workloads utilizing Intel SKC to protect its key.

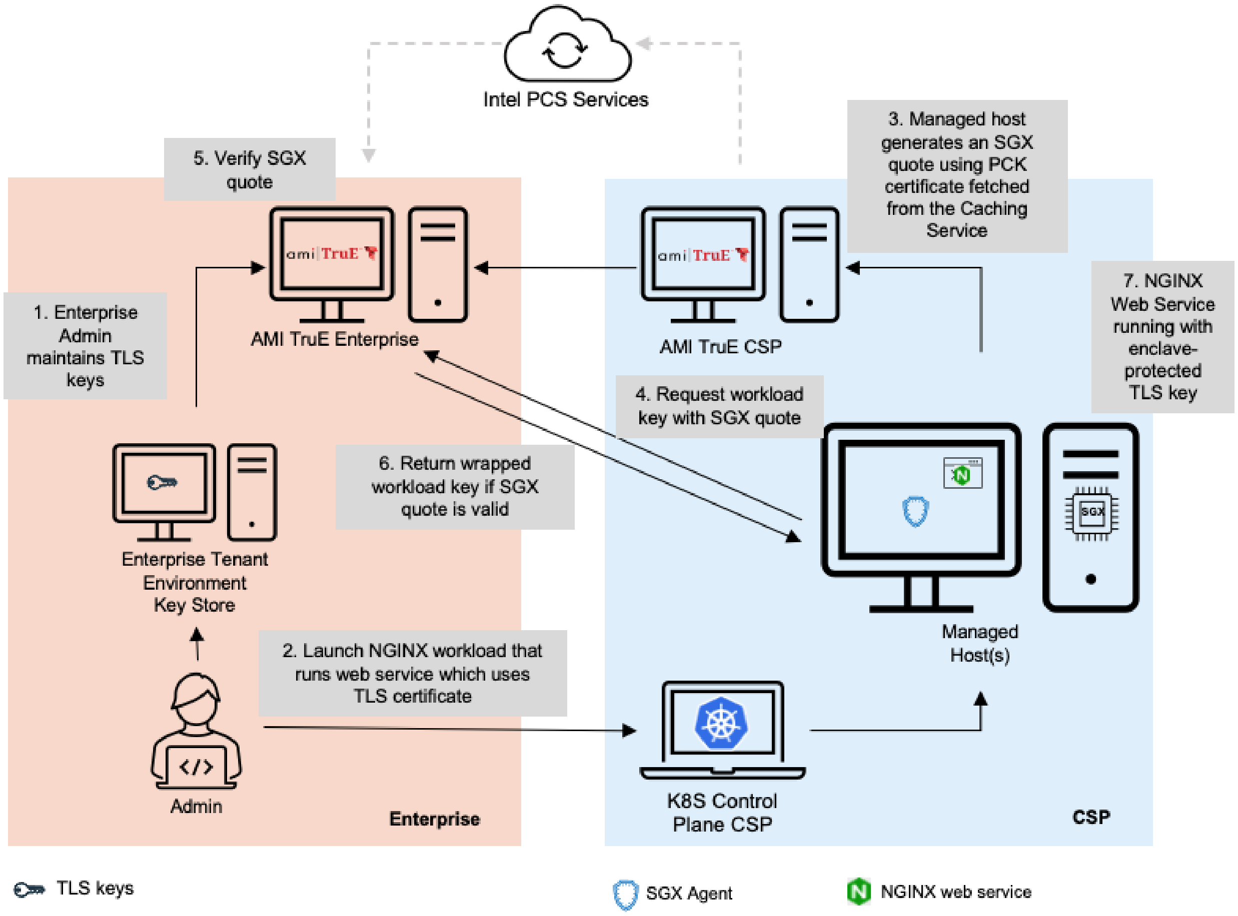


Fig. 8. NGINX Key Transfer Workflow with Secure Key Caching

Some workloads deployed by tenants in datacenters that are under the control of a third party (CSPs, edge provider, and enterprise private cloud) use sensitive cryptographic keys. These keys must be adequately protected by tenants. Keys can also be disclosed because of the vulnerabilities in the third-party infrastructure.

Key protection can be achieved using an HSM, but this requires ad hoc cloud or edge environment that allows physical access to servers. With SKC, tenants can continue to use standard cloud and edge environments without compromising the confidentiality of their sensitive keys and without additional tools.

Fig. 9 details the call flows between the individual components of the solution with the specific information that is transmitted for each interaction in the process of launching NGINX workloads utilizing Intel SKC to protect its key.

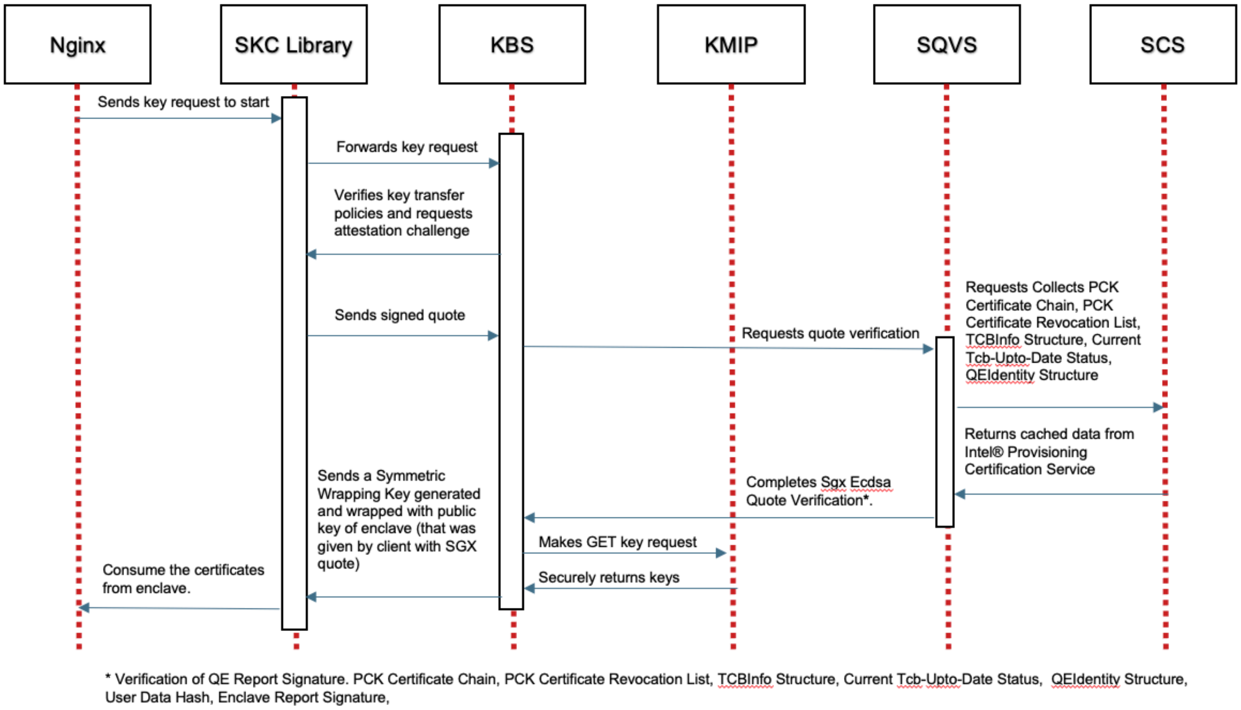


Fig. 9. NGINX Key Transfer Call Flow

D.3. Installation and Configuration

Log in to the Kubernetes control plane node and perform the following steps.

1. Navigate to the ‘kbs’ folder:

```
# cd /root/manifests/kbs
```

2. Open the kbs.conf file and edit it based on the comments inside it.
3. Open the rsa_create.py file in edit mode and update the following value:

```
KMIP_IP = '<K8S-Control-Plane-IP>'
```

Note: Single quotes are mandatory.

Example:

KMIP_IP = '10.0.0.6'

4. Run the following script to generate the KBS public key certificate:

```
# ./run.sh reg
```

5. Record the generated certificate ID for upcoming use.
6. Copy the <kbs_public_cert_ID>.crt file generated in the ‘kbs’ folder to the ‘skc_library/resources’ folder:

```
# cp <kbs_public_cert_ID>.crt ../skc_library/resources
```

7. Edit the SKC Library deployment.yml and service.yml files as described in Table 1.

```
# cd /root/manifests/skc_library
```

Table 1. SKC Library Files to Edit

Filename	Edits

deployment.yml	Update the following sections with the KBS certificate ID:
	`- mountPath: /root/<kbs public certificate>.crt`
	`   name: kbs-cert-secret-volume`
	`   subPath: <kbs public certificate>.crt`
	
	Example:
	- mountPath: /root/de02facf-458f-40a3-b3d8-93f1a26959c9.crt
	name: kbs-cert-secret-volume
	subPath: de02facf-458f-40a3-b3d8-93f1a26959c9.crt

service.yml	Change the https port number to 30463, for example, in case of conflict with 30443 when the Intel SGX Host Verification Service (HVS) is running: Example: ports: - name: https port: 8080 targetPort: 2443 nodePort: 30463 protocol: TCP

8. Edit the SKC Library resource files as described in Table 2.

```
# cd /root/manifests/skc_library/resources
```

Table 2. SKC Library Resource Files to Edit

Filename	Edits
create_roles.conf	Update the variables based on the comments within the file.
<kbs_public_cert>.crt	Ensure that this file is present in the current folder.
hosts	Update the details in placeholders.

keys.txt	Update the KBS public certificate ID in the placeholder for 'id'.
	
	Example:
	pkcs11:token=KMS;id=de02facf-458f-40a3-b3d8-93f1a26959c9;
	object=RSAKEY;type=private;pin-value=1234;

kms_npm.ini	Update the KBS IP address. Example: server=https://10.0.0.6:30448/kbs

nginx.conf	Update the KBS public certificate ID.
	
	`ssl_certificate "/root/<kb key id>.crt";`
	`ssl_certificate_key "engine:pkcs11:pkcs11:token=KMS;id=<kbs key id>;object=RSAKEY;type=private;pin-value=1234";`
	
	Example:
	ssl_certificate "/root/de02facf-458f-40a3-b3d8-93f1a26959c9.crt";
	ssl_certificate_key "engine:pkcs11:pkcs11:token=KMS;id=de02facf-458f-40a3-b3d8-93f1a26959c9;object=RSAKEY;type=private;pin-value=1234";

sgx_default_qcnl.conf	Update the SGX Caching Service (SCS) IP address.

Filename	Edits
skc_library.conf	Before editing, run the script skc_library_create_roles.sh and get the token. `# ./skc_library_create_roles.sh` Then open the skc_library.conf file, update the token value for SKC_TOKEN, and update other variables based on comments within the file.

9. Update the /root/manifests/isecl-skc-k8s.env file. Uncomment the line below and update the KBS public certificate ID in the placeholder.

```
# KBS_PUBLIC_CERTIFICATE=<key id>.crt
```

10. Launch the skc library deployment:

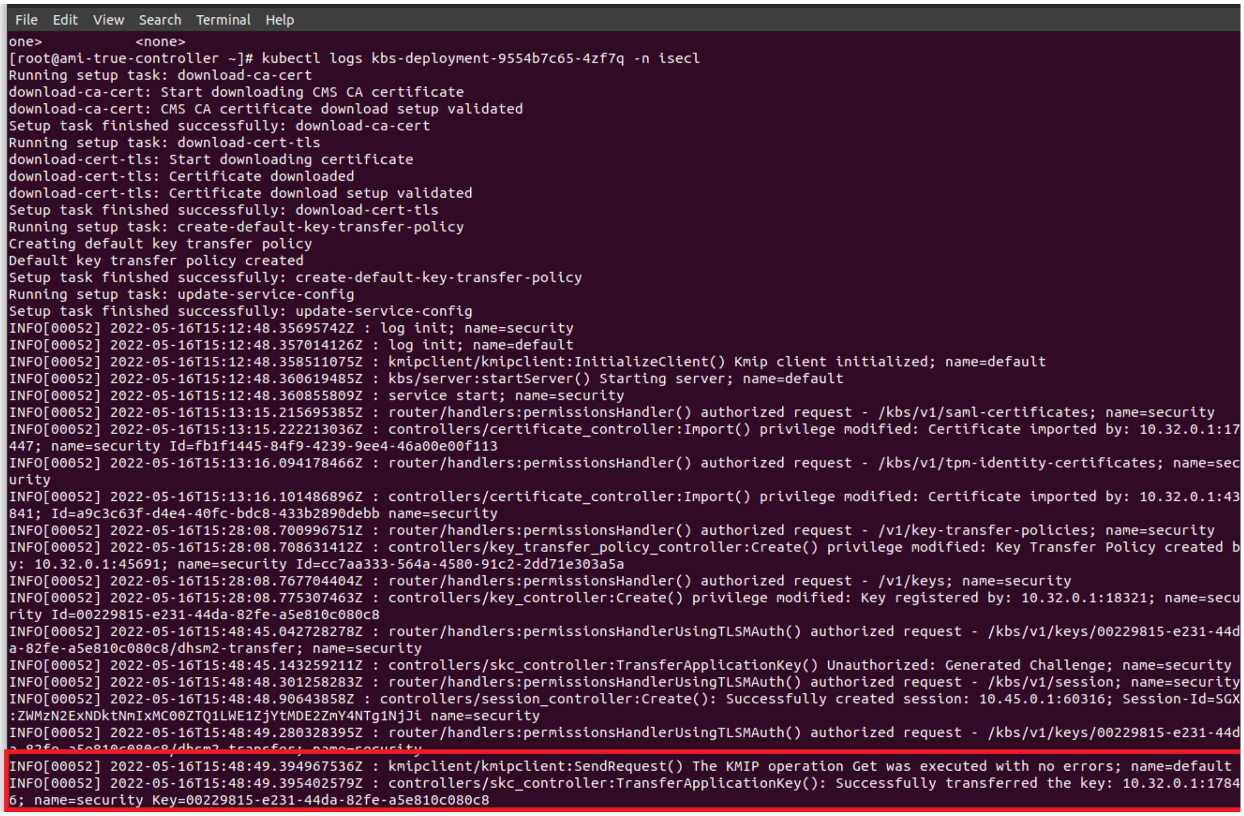
```
# cd /root/manifests
# ./skc-bootstrap.sh up skclib
```

11. Check whether the skc library pod is running without any restarts/errors:

```
# kubectl get pods -n isecl -o wide
```

12. Access the following URL from your browser. The port number should match the port configured in the service.yml file. An example is *https://10.0.0.133:30463/*

13. Check the key broker service log for the successful key transfer messages. See the screen shot in Fig. 10.



```
one>          <none>
[root@ami-true-controller ~]# kubectl logs kbs-deployment-9554b7c65-4zf7q -n isecl
Running setup task: download-ca-cert
download-ca-cert: Start downloading CMS CA certificate
download-ca-cert: CMS CA certificate download setup validated
Setup task finished successfully: download-ca-cert
Running setup task: download-cert-tls
download-cert-tls: Start downloading certificate
download-cert-tls: Certificate downloaded
download-cert-tls: Certificate download setup validated
Setup task finished successfully: download-cert-tls
Running setup task: create-default-key-transfer-policy
Creating default key transfer policy
Default key transfer policy created
Setup task finished successfully: create-default-key-transfer-policy
Running setup task: update-service-config
Setup task finished successfully: update-service-config
INFO[00052] 2022-05-16T15:12:48.35695742Z : log init; name=security
INFO[00052] 2022-05-16T15:12:48.357014126Z : log init; name=default
INFO[00052] 2022-05-16T15:12:48.358511075Z : kmipclient/kmipclient:InitializeClient() Kmip client initialized; name=default
INFO[00052] 2022-05-16T15:12:48.360619485Z : kbs/server:startServer() Starting server; name=default
INFO[00052] 2022-05-16T15:12:48.360855809Z : service start; name=security
INFO[00052] 2022-05-16T15:13:15.215695385Z : router/handlers:permissionsHandler() authorized request - /kbs/v1/saml-certificates; name=security
INFO[00052] 2022-05-16T15:13:15.222213036Z : controllers/certificate_controller:Import() privilege modified: Certificate imported by: 10.32.0.1:17
447; name=security Id=fb1f1445-84f9-4239-9ee4-46a00e00f113
INFO[00052] 2022-05-16T15:13:16.094178466Z : router/handlers:permissionsHandler() authorized request - /kbs/v1/tpm-identity-certificates; name=sec
urity
INFO[00052] 2022-05-16T15:13:16.101486896Z : controllers/certificate_controller:Import() privilege modified: Certificate imported by: 10.32.0.1:43
841; Id=a9c3c63f-d4e4-40fc-bdc8-433b2890debb name=security
INFO[00052] 2022-05-16T15:28:08.700996751Z : router/handlers:permissionsHandler() authorized request - /v1/key-transfer-policies; name=security
INFO[00052] 2022-05-16T15:28:08.708631412Z : controllers/key_transfer_policy_controller:Create() privilege modified: Key Transfer Policy created b
y: 10.32.0.1:45691; name=security Id=cc7aa333-564a-4580-91c2-2dd71e303a5a
INFO[00052] 2022-05-16T15:28:08.767704404Z : router/handlers:permissionsHandler() authorized request - /v1/keys; name=security
INFO[00052] 2022-05-16T15:28:08.775307463Z : controllers/key_controller:Create() privilege modified: Key registered by: 10.32.0.1:18321; name=secu
rity Id=00229815-e231-44da-82fe-a5e810c080c8
INFO[00052] 2022-05-16T15:48:45.042728278Z : router/handlers:permissionsHandlerUsingTLSMAuth() authorized request - /kbs/v1/keys/00229815-e231-44d
a-82fe-a5e810c080c8/dhsm2-transfer; name=security
INFO[00052] 2022-05-16T15:48:45.143259211Z : controllers/skc_controller:TransferApplicationKey() Unauthorized: Generated Challenge; name=security
INFO[00052] 2022-05-16T15:48:48.301258283Z : router/handlers:permissionsHandlerUsingTLSMAuth() authorized request - /kbs/v1/session; name=security
INFO[00052] 2022-05-16T15:48:48.90643858Z : controllers/session_controller:Create(): Successfully created session: 10.45.0.1:60316; Session-Id=SGX
:ZWMzN2ExNDktNmIxMC00ZTQ1LWE1ZjYtMDE2ZmY4NTg1NjJi name=security
INFO[00052] 2022-05-16T15:48:49.280328395Z : router/handlers:permissionsHandlerUsingTLSMAuth() authorized request - /kbs/v1/keys/00229815-e231-44d
a-82fe-a5e810c080c8/dhsm2-transfer; name=security
INFO[00052] 2022-05-16T15:48:49.394967536Z : kmipclient/kmipclient:SendRequest() The KMIP operation Get was executed with no errors; name=default
INFO[00052] 2022-05-16T15:48:49.395402579Z : controllers/skc_controller:TransferApplicationKey(): Successfully transferred the key: 10.32.0.1:1784
6; name=security Key=00229815-e231-44da-82fe-a5e810c080c8
```

Fig. 10. Successful Key Transfer Message

Appendix E. Acronyms and Other Abbreviations

API
Application Programming Interface

BIOS
Basic Input/Output System

CA
Certificate Authority

CPU
Central Processing Unit

CSP
Cloud Service Provider

CSR
Certificate Signing Request

DB MEK
Database Master Encryption Key

DCAP
(Intel) Data Center Attestation Primitives

DDR
Double Data Rate

DDR4
Double Data Rate Fourth Generation

DevOps
Development and Operations

DFx
Design for Debug, Test, Manufacturing, and/or Validation

DIMM
Dual In-Line Memory Module

DNS
Domain Name System

GB
Gigabyte

GHz
Gigahertz

GW
Gateway

HSM
Hardware Security Module

HTML
Hypertext Markup Language

HVS
(Intel SGX) Host Verification Service

IDZ
Intel Developer Zone

iMC
Integrated Memory Controller

IP
Internet Protocol

IPsec
Internet Protocol Security

IR
Interagency or Internal Report

K8S
Kubernetes

KBS
(Intel) Key Broker Service

KMIP
Key Management Interoperability Protocol

KMS
Key Management System

NFS
Network File System

OS
Operating System

OSS
Open Source Software

PCK
Provisioning Certification Key

PCS
(Intel) Provisioning Certification Service

PKCS
Public Key Cryptography Standards

PKI
Public Key Infrastructure

QE
Quoting Enclave

RAM
Random Access Memory

RHEL
Red Hat Enterprise Linux

SA
System Agent

SCS
(Intel) SGX Caching Service

SDK
Software Development Kit

SecL-DC
(Intel) Security Libraries for Datacenter

SGX
(Intel) Software Guard Extension

SKC
(Intel) Secure Key Caching

SP
Scalable Processor

SQVS
SGX Quote Verification Service

SWK
Symmetric Wrapping Key

TEE
Trusted Execution Environment

TLS
Transport Layer Security

TruE
(AMI) Trusted Environment

TXT
(Intel) Trusted Execution Technology

UEFI
Unified Extensible Firmware Interface

UI
User Interface

UMA
Uniform Memory Access

URI
Uniform Resource Identifier

VM
Virtual Machine

VMM
Virtual Machine Manager

VPN
Virtual Private Network