Draft NISTIR 8320C

Hardware Enabled Security:

Machine Identity Management and Protection

Michael Bartock
Murugiah Souppaya
Mourad Cherfaoui
Jing Xie
Paul Cleary

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

National Institute of
Standards and Technology
U.S. Department of Commerce

Draft NISTIR 8320C

Hardware Enabled Security:

Machine Identity Management and Protection

Michael Bartock
Murugiah Souppaya
Computer Security Division
Information Technology Laboratory

Mourad Cherfaoui
Intel Corporation
Santa Clara, California

Jing Xie
Paul Cleary
Venafi
Salt Lake City, Utah

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

April 2022

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

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

National Institute of Standards and Technology Interagency or Internal Report 8320C
35 pages (April 2022)

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

Public comment period: April 20, 2022 - June 6, 2022

Submit comments on this publication to: 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).

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.

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. 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).

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 to improve machine identity management and protection.

Acknowledgments

The authors thank everyone who contributed their time and expertise to the development of this report.

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 Appendices

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

The purpose of this publication is to describe an effective approach for managing machine identities so that they are protected from malware and other security-related vulnerabilities. This publication 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. The publication provides sufficient details about the prototype implementation so that organizations can reproduce it if desired. The publication 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 publication 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 publication.

This publication 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*.[1] 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."[2]
- **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:

[1] 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

[2] Polydys ML, Wisseman S (2009) Software Assurance in Acquisition: Mitigating Risks to the Enterprise. https://apps.dtic.mil/dtic/tr/fulltext/u2/a495389.pdf

    - 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 Venafi 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 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.

Machine identities come in many shapes and forms. Sometimes they are an implementation of X.509 certificates consisting of a public and private pair, where the private part, in the form of a secret key, should be always kept confidential. In other cases, the whole machine identity is represented by a secret key (e.g., application programming interface [API] key) that should be kept confidential through its entire lifecycle. If a secret key is compromised, that machine identity can be misused by malicious actors, bringing financial and reputational damage to an organization. Secret keys have long been a highly prized target in cyberattacks.

Secret keys can be compromised because they are vulnerable in three situations: at rest, in transit, and in use. A highly discouraged yet popular practice is to store secret keys on the file system and rely on operating system (OS) provided controls such as file permissions for protection. A large number of OS-level vulnerabilities have been discovered, disclosed, and exploited to get secrets from restricted places in the file system. Secret keys can be protected with encryption, but this usually raises the problem of managing and protecting the encryption key. Essentially, this moves the problem from protecting one key to protecting another key. An effective way to protect secret keys at rest is to store them in an attached or network-based hardware security module (HSM). Protection of secret keys and sensitive data in general while in transit is a problem that the industry has successfully addressed with technologies like Transport Layer Security (TLS) and Internet Protocol Security (IPsec), which provide network channels that are protected against disclosure, malicious impersonation, and data tampering.

One concern that has yet to be addressed is the lack of protection for sensitive data in use in memory, which is often at risk for memory targeting attacks such as memory scraping. The problem becomes worse as organizations move workloads to dynamic, multi-tenant, third-party cloud infrastructures that rely on large numbers of humans to operate and manage the hardware and software. Organizations that are especially active in the cloud will want to take extra care to safeguard their machine identities being used in memory. A machine identity management solution addresses the problems of lifecycle management for machine identities, such as TLS certificates and key pairs. But there is still some work to be done when it comes to protecting secret keys that are constantly used in memory to prove ownership of machine identities.

The ultimate goal is to be able to use “trust” as a boundary for 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 Section 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 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 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 intended to be 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.

3.1 Solution Overview

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.

All machine identity management strategies must start with protecting the machine identities used today. These include:

- TLS certificates: from load balancers to application servers and next-generation firewalls
- Cloud-native environments: from Kubernetes to Istio, including new and emerging workloads
- Internet of Things (IoT) and mobile: from enterprise mobility to cloud-based IoT platforms
- Secure Shell (SSH) keys: from enterprise Linux infrastructure to cloud IaaS
- Code signing certificates: from desktop applications to container registries

Across all these types of machine identities, management should be powered by technology that consistently provides:

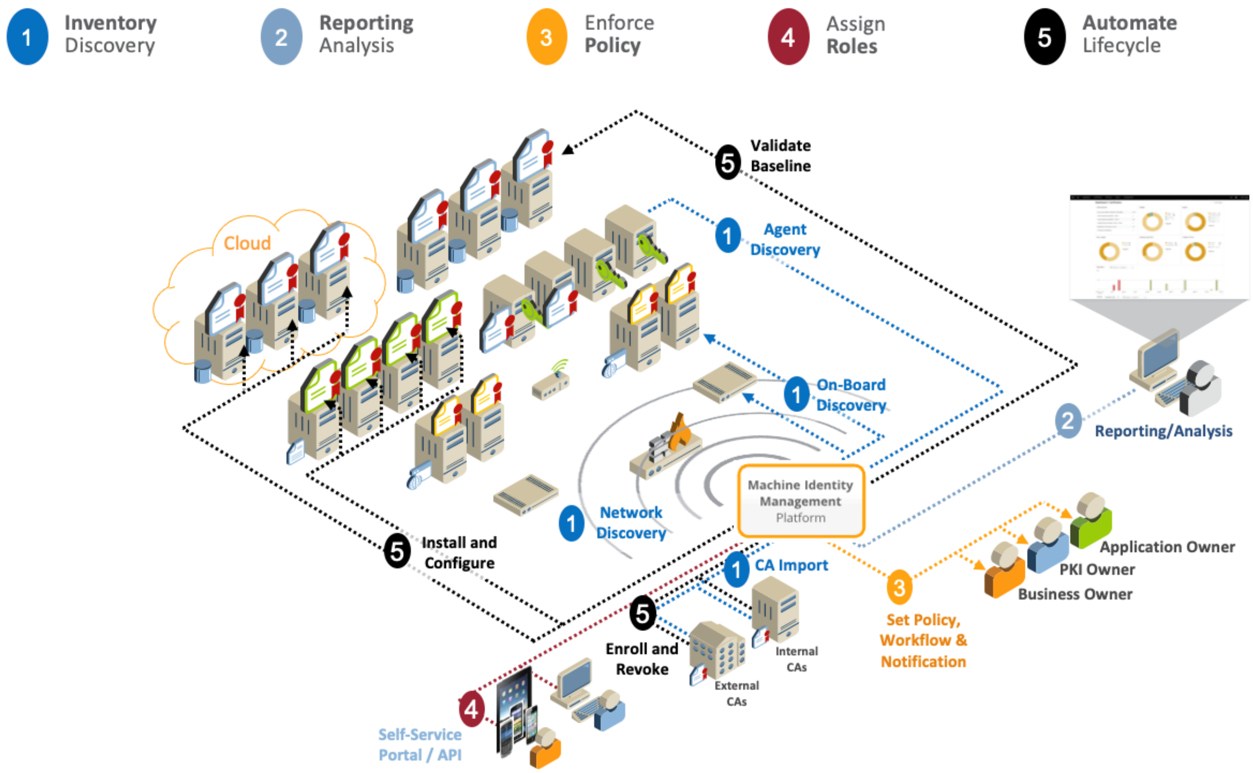
- **Visibility** of every machine identity in use throughout the organization is paramount. Something cannot be secured if it is not known or is undiscoverable. To ensure a complete and accurate inventory, discovery of unknown machine identities must be enterprise-wide and include on-premises, virtual, cloud environments, and even IoT. Visibility should also provide continuous awareness of where, how, and why machine identities are being used across cloud and enterprise networks.
- **Intelligence** should provide comprehensive and continually updated information about all machine identities. This level of intelligence is necessary to understand, communicate, and reduce levels of risk while increasing the speed of accessibility and deployment for a variety of audiences, including developers, security teams, and executives.
- **Automation** should be built around accurate, automated processes that operate at machine speed and are tightly integrated. The goal of automation is to eliminate outages, errors, and vulnerabilities—especially within the wide range of APIs, open-source projects, and cloud-native environments being used today.

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. A proper, enterprise-wide machine identity management strategy enables security teams to keep up with the rapid growth of TLS machine identities, while at the same time allowing the organization to scale securely by using automation to minimize risks introduced by humans. 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.

3.2 Solution Architecture

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 Figure 1:

1. **Inventory/Discovery**: The first step to enterprise-wide machine identity management is the ability to discover all machine identities in use throughout the infrastructure, regardless of the platform or product where the machine identity is being used.
2. **Reporting/Analysis**: Once a complete and accurate inventory is in place, analysis of the results gives the organization a clear picture of where vulnerabilities in machine identities may be present (small key sizes, weak algorithms, self-signed certificates, etc.).
3. **Enforce Policy**: The next step is to create and enforce machine identity policies so that all newly issued and renewed machine identities are free of the vulnerabilities discovered in step 2.
4. **Assign Roles**: The ability to enforce policy provides the core capabilities that security or public key infrastructure (PKI) teams need to support self-service for other teams in the organization, eliminating unnecessary bottlenecks caused by lengthy approval workflows.
5. **Automate Lifecycle**: The final and arguably most important component is complete automation of the machine identity lifecycle. You should validate your automation to ensure it is configured correctly so that human intervention is not necessary at any point of the lifecycle—from initial creation to renewal, and finally provisioning to the end consumers of machine identities.

352 

353 **Figure 1 - Stage 0 Implementation: Typical Enterprise-Grade Machine Identity Management**

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.

4.1 Solution Overview

Attached and network-based HSMs and key file encryption protect secret keys at-rest and aim to secure transport channels such as TLS protect secret keys in-transit. However, secret keys are usually not protected when they are loaded into RAM for cryptographic processing. This is a gap in end-to-end secret key protection. Secret keys are targets for malicious agents when they are in RAM. Malware on the host can exploit vulnerabilities in the host management software (OS, hypervisor, firmware) to evade software protections. A malicious administrator can get hold of a secret key by taking a memory snapshot. Poor operational practices can also disclose the secret key without malicious intent: if a virtual machine (VM) snapshot contains a secret key that is not properly protected, malicious agents who get access to the VM snapshot can extract the key.

Mechanisms to protect secret keys in-use exist. An attached or network-based HSM performs cryptographic processing inside the HSM[3] 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 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.

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."[4] The evidence can contain the public key part of an ephemeral public/private key pair generated inside the TEE.[5] The *relying*

---

[3] See Section 7.5, "Protecting Keys and Secrets" in NIST IR 8320.

[4] 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

[5] 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.

party can wrap secrets with the TEE public key[6] before sharing them with the TEE. Considerations such as the freshness of the evidence and protection against replay attacks are TEE technology-dependent.

Another problem that the solution addresses is provisioning the private key into the TEE. The provisioning cannot be done by the workload because the key would be exposed in RAM and possibly on disk as part of the configuration of the workload. This is solved by provisioning the workload not with the key itself but rather with a key identifier (ID). The workload passes the key ID to the adapter. The adapter could request the key transfer from a network HSM using the key ID. However, this would expose the key in cleartext in RAM. This is where TEE attestation comes in. The adapter requests TEE evidence from the TEE. The evidence contains the public key part of the public/private key pair generated inside the TEE. The adapter can now request the transfer of the private key from the network HSM using the TEE evidence to prove that the private key will be kept confidential in the TEE. Only the TEE can unwrap the workload private key since it has the private key corresponding to the public wrapping key. Now that the TEE has the private key, it can perform cryptographic operations requested by the workload, such as the operations involved in the TLS handshake.

For more detailed information on confidential computing and the use of TEE, see Section 5.2, Application Isolation, of NIST IR 8320.

4.2 Solution Architecture

Figure 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. These are the components shown in Figure 2:

- **Client**: Typically, a client needs to perform a TLS handshake with the workload. It can also be a client that requests a cryptographic operation from the workload involving the private key, such as the generation of a digital signature of a payload.
- **Workload**: The workload performs cryptographic operations using the private key. The workload in this figure uses the Public Key Cryptography Standards 11 (PKCS 11) API and is configured with the key ID (like a PKCS#11 Uniform Resource Identifier [URI]). Other cryptographic APIs could be used.
- **TEE Adapter**: This is a thin layer library that receives cryptographic API calls from the workload and relays them to the TEE over the communication protocol supported by the specific TEE technology. The TEE adapter hides the TEE communication details from the workload so that code changes are not necessary. When the workload wants to get a key handle to the private key using its key ID, the TEE adapter retrieves the private key from the network HSM after performing a TEE attestation and loads it into the TEE, which then returns a key handle. To perform the TEE attestation, the TEE adapter requests the TEE to generate TEE evidence. The TEE evidence contains the TEE measurement and the public key part of a TEE public/private key pair. The private key

[6] This can be done in two steps. First, a Software 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.

returned by the network HSM is wrapped with the TEE public key. The TEE adapter cannot see the private key since it is wrapped.

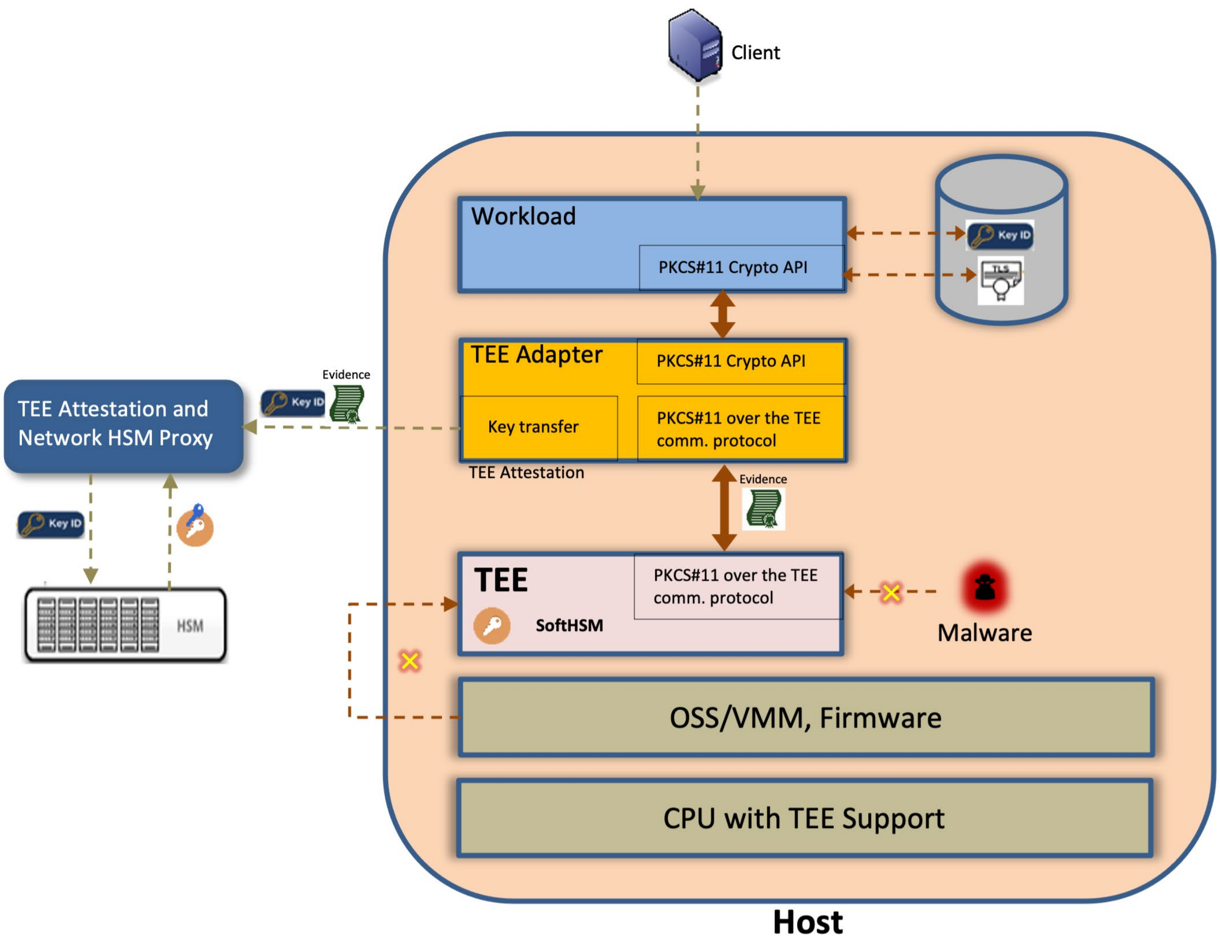


Figure 2 - Private Key Protection Flows

- **TEE**: This is where the private key is stored at runtime and where cryptographic processing is done. Figure 2 shows SoftHSM as the cryptographic software running in the TEE. SoftHSM implements PKCS#11 in software.[7] A solution that uses a cryptographic API other than PKCS#11 would need to implement this API in the TEE.
- **TEE Attestation and Network HSM Proxy**: The private key should be released to the TEE only upon a successful TEE attestation. However, network HSMs are typically not TEE-aware. The TEE attestation and network HSM proxy acts as a proxy for the network HSM. It performs the TEE attestation verification, and the key retrieval request is passed on to the network HSM only if the verification is successful. Depending on the TEE technology, the TEE evidence verification could involve an interaction with online services managed by the TEE technology manufacturer. The private key in the network HSM must be wrapped with the TEE public key before it gets returned to the TEE. The private key wrapping by the TEE public key could be done in the TEE attestation and

[7] SoftHSM uses Openssl or Botan to perform the actual cryptographic operations.

network HSM proxy. If the private key wrapping by the TEE public key is done in the backend network HSM, the private key is only visible in the network HSM and the TEE. No other software, including privileged software, can see the private key.

The private key transfer from a network HSM flow is triggered when a client interaction with the workload involves the private key. The workload instructs its cryptographic library (such as PKCS#11) to return a key handle passing the private key ID that it is configured with as input. The request from the workload is first received by the TEE adapter, which needs to transfer the private key from a network HSM. The TEE attestation and network HSM proxy ("proxy") receives the request from the TEE adapter and challenges it to prove that the key will be protected in a TEE. The TEE adapter requests the generation of TEE evidence and sends it to the proxy. Upon successful verification of the TEE evidence, the proxy registers the TEE public key in the network HSM. The network HSM wraps the private key with the public key of the TEE and returns it to the proxy. The private key wrapping can also be done in the proxy, but this exposes the private key outside of the network HSM. The wrapped private key is returned to the TEE adapter, which loads it into the TEE and gets a key handle back. The key handle is then returned to the workload. From then on, all the cryptographic requests by the workload are just channeled to the TEE by the TEE adapter using the TEE communication protocol without any other processing.

The following are notes on stage 1 for implementers:

- The prototype description does not detail how components authenticate to other components and how authorization verification is done. For example, the proxy might want to authenticate and authorize the TEE adapter in addition to verifying the TEE evidence before giving access to the private key. Conversely, the TEE adapter must ensure that it is talking to an authenticated proxy. Authentication and authorization are implementation-dependent and can use any existing authentication and authorization mechanism such as certificates, API keys, or JavaScript Object Notation (JSON) Web Tokens (JWT).
- Access control to the TEE is TEE technology-dependent and is not covered in this document.
- The TEE in this solution might theoretically run on a different host. However, this introduces network latency that might not be acceptable in some deployments.

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.

5.1 Solution Overview

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.

5.2 Solution Architecture

Figure 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.

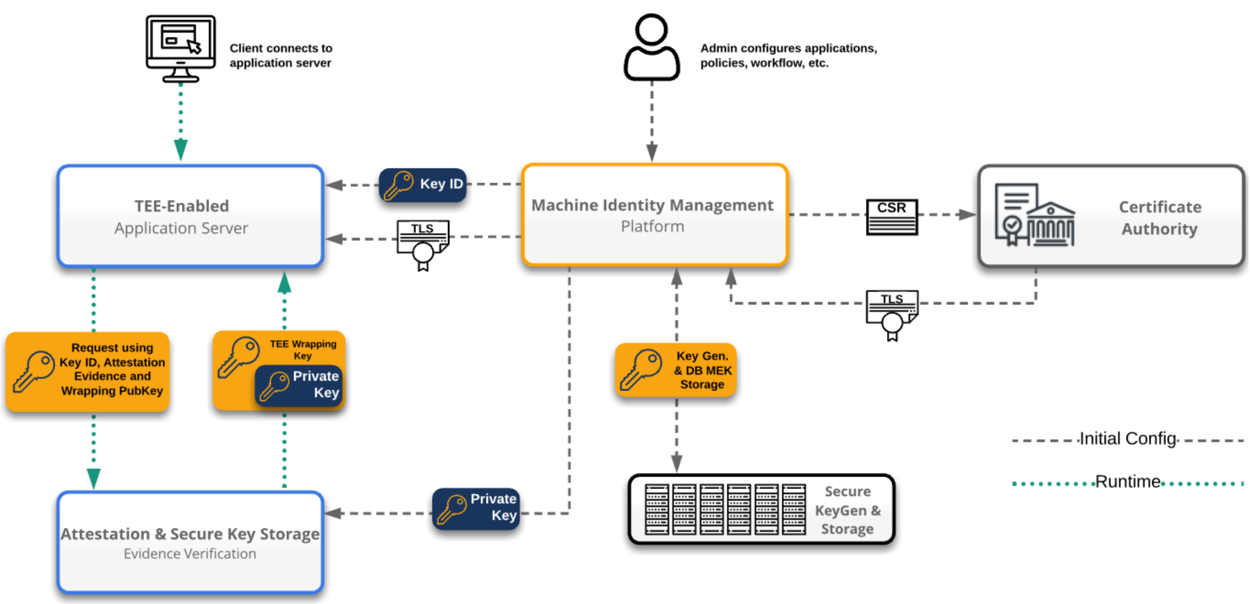


Figure 3 - High-Level Prototype Architecture

The following steps detail the configuration and provisioning flows:

1. An administrator who is responsible for a machine identity connects to the machine identity management platform and provisions the TEE-enabled application server in the machine identity management internal database. The administrator enters all the TEE-enabled application server configuration information such as the access credentials and policies. Note: An example of a policy is the time interval after which the TEE-enabled

application server public/private key pair and the corresponding X.509 certificate must be regenerated.

2. When it is time to regenerate the key pair and X.509 certificate, the machine identity management platform generates a new key pair, with the private key encrypted by a database master encryption key (DB MEK) in an attached or network-based HSM. The private key is then pushed to the attestation and secure key storage component over a secure TLS channel where it gets protected in a number of ways.
    a. By default, the private key is stored on the file system of the Attestation and Key Storage Service (not recommended).
    b. The private key is stored in a backend KMS via the KMIP integration of the Attestation and Key Storage Service (recommended).
3. The machine identity management platform generates a certificate signing request (CSR) and requests an X.509 certificate from a certificate authority (CA). A TLS X.509 certificate is returned.
4. The machine identity management platform provisions the TLS X.509 certificate and the private key ID (<u>not the private key</u>) on the TEE-enabled application server.

The following steps detail the runtime flows:

1. At runtime, when the TEE-enabled application server needs to prove its identity to a client, it transfers the secret key into the TEE. This triggers the generation of the TEE evidence.
2. The TEE evidence contains a public wrapping key that is generated inside the TEE. The TEE-enabled application server sends the evidence along with the private key ID to the attestation and secure key storage.
3. The attestation and secure key storage verifies the evidence, possibly using an online verification service managed by the TEE technology manufacturer.
4. If the verification is successful, the secret key is wrapped with the TEE wrapping public key and returned to the TEE-enabled application server.
5. Now the secret key is provisioned and unwrapped in the TEE. The TEE-enabled application server can perform cryptographic operations such as a TLS handshake with the secret key stored in the TEE.

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 four servers that reside in geographically separate locations. Three 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 VMs in the NIST lab are:

1. Windows Server 2019 with Active Directory, Domain Name System (DNS), and CA roles installed
2. Windows Server 2019 with Venafi Trust Protection Platform solution and Intel® Software Guard Extensions (SGX) plugin installed
3. Red Hat Enterprise Linux (RHEL) 8 server with Intel® Security Libraries (ISecL) installed

The fourth server is in the Intel lab. It is running RHEL and it has an Intel SGX-enabled chipset to protect key material.

The prototype implementation network is a flat management network for the Venafi components and Intel compute server. Figure 4 shows the high-level architecture of how the four servers in the prototype are connected.

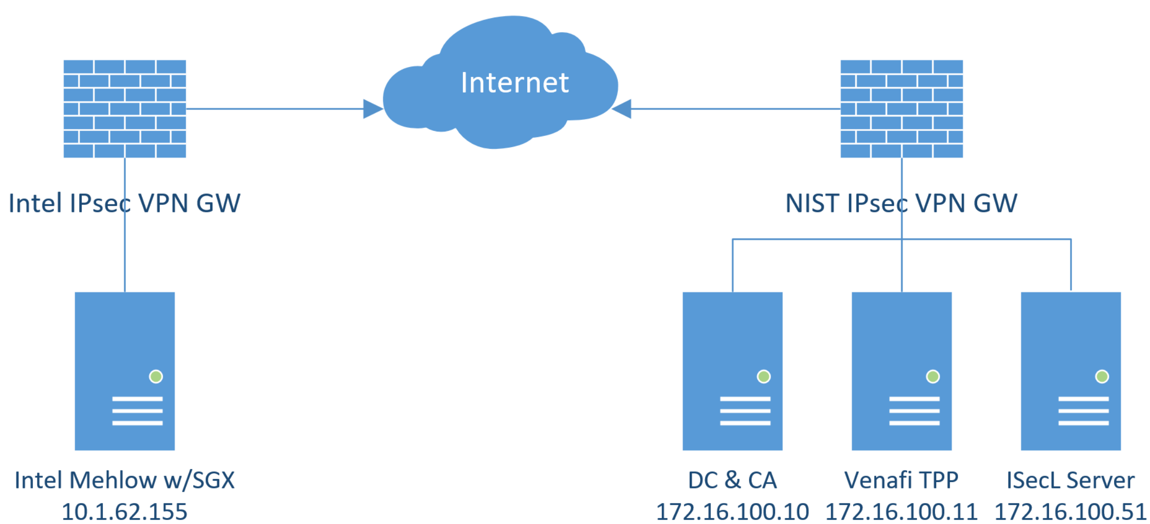


Figure 4 - Prototype Architecture

Appendix B provides additional details for installing and configuring the Venafi Trust Protection Platform 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—Venafi Machine Identity Management Implementation

This appendix contains supplementary information describing the components and the steps needed to set up the prototype implementation for Venafi Trust Protection Platform.

Table 1 lists the virtual hardware requirements for Venafi Trust Protection Platform. For more detailed information, use Venafi Customer Support credentials to log in and see the Venafi Installation and Upgrade guide at https://docs.venafi.com/Docs/current/TopNav/Content/Install/r-install-SysReq-ALLVenProducts.php?tocpath=Get%20Started%7CInstallation%20and%20Upgrade%20Guide%7C_____1.

Table 1 - Trust Protection Platform VM Requirements

Feature	Requirement
Processor	4 processing cores
Memory	16 GB RAM
Disk space for the Trust Protection Platform application	5 GB (NOTE: The Trust Protection Platform application can be installed on a secondary partition.)
Disk space for SQL database	50 GB (Microsoft SQL Server versions 2019, 2017, and 2016 SP2 are supported)
OS	Microsoft Windows Server 2016 and 2019 are supported. Trust Protection Platform only supports English Language Installation Media from Microsoft. While it does support region setting configurations to ensure that date and times appear correctly, the Windows servers on which you install Trust Protection Platform must be derived from Windows English installation media.

The Venafi Configuration Console is built upon the Microsoft Management Console (MMC) Framework. Some of the nodes, such as the Venafi Event Viewer and Venafi Code Signing, are snap-ins that can be installed on other Windows servers and workstations, even if they are not set up to be Venafi servers. If you plan to leverage this functionality, it can only be installed on Windows systems that meet the following requirements:

- .NET 4.7.2 or greater
- Windows 8.1 or later
- Windows MS SQL 2016 SP2 or later

In order to download the latest version of Venafi Trust Protection Platform, perform the following steps:

1. Navigate to https://download.venafi.com and log in with your Venafi Customer Support credentials. If your account does not have access to the downloads site, and you think it should, please contact Venafi Customer Support. If you don't have an account, you can register at https://success.venafi.com/signin/register.
2. Expand the **Trust Protection Platform** group, then expand **Current**.

3. In each folder inside the **Current** folder, download the zip file.

   Note: The “dot zero” version is the full installation and includes all the files necessary to run the system. Any higher “dot” versions are patches and only contain updated files, not the entire package. Therefore, you need to install the full version before you can install any patches. All patches are cumulative, so an x.1.2 patch includes the files from the x.1.1 patch.

4. Store the zip file(s) in your software repository, if applicable.
5. Copy the installer (and any patch) to each of the Venafi servers.
6. Run the Windows installer to have Trust Protection Platform installed on the system.

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 the 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. Figure 5 shows the isolation of Intel SGX enclaves in a host.

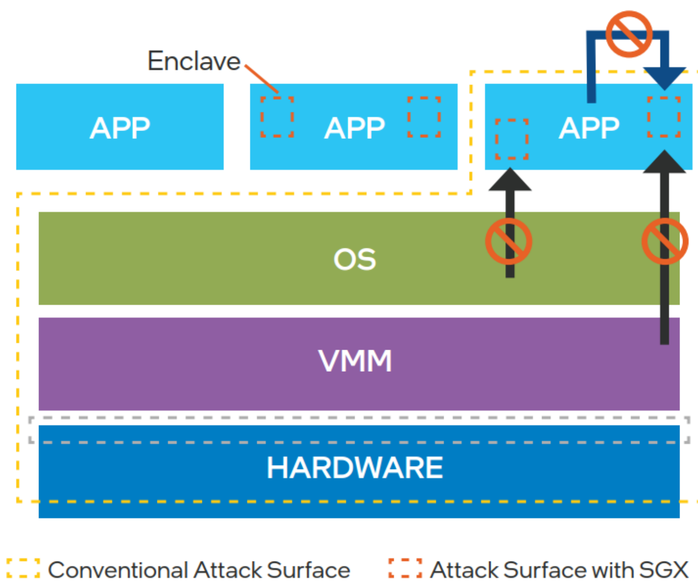


Figure 5 - Intel SGX Enclave

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 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 Basic Input/Output System (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 Figure 6.

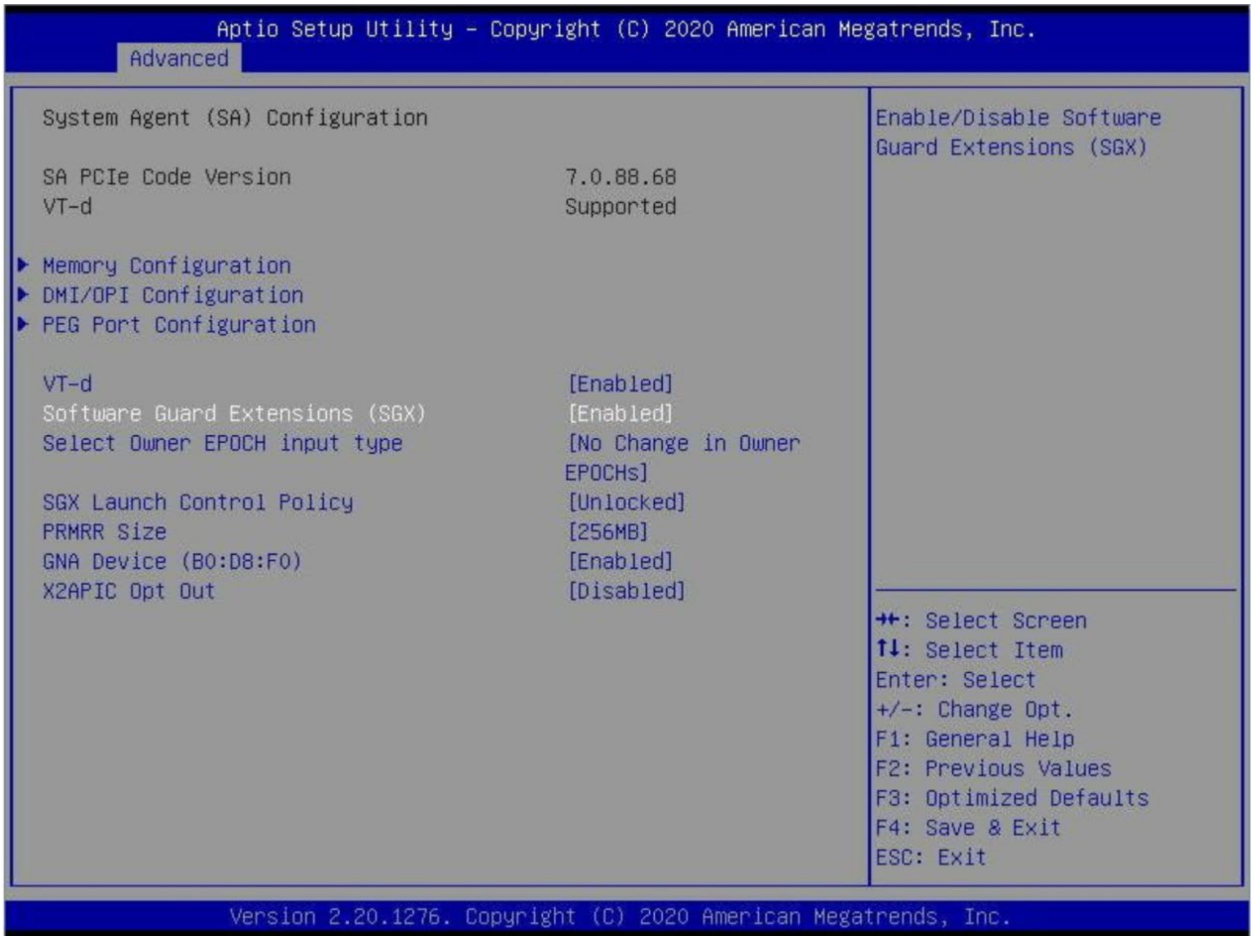


Figure 6 - BIOS Enable SGX

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

The prototype can also work on Intel Xeon SP-based platforms. Intel SGX configuration for these platforms is detailed in https://cdrdv2.intel.com/v1/dl/getContent/632236.

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

Integrating the Venafi Trust Protection Platform with Intel SGX provides an alternative for managing the last mile of machine identity lifecycles that helps elevate end-to-end protection with ease and sustainability. The integrated solution helps avoid abuse and compromise of machine identities not only at rest or in transit, but also when they are inevitably in use in RAM, especially in untrusted environments such as public cloud.

Figure 7 shows the major difference between this solution (right) and the status quo (left). With app servers running on Intel SGX-enabled machines, the design strives to deprive the attacker of the ability to capture the secret key—even when it is used at runtime in memory.

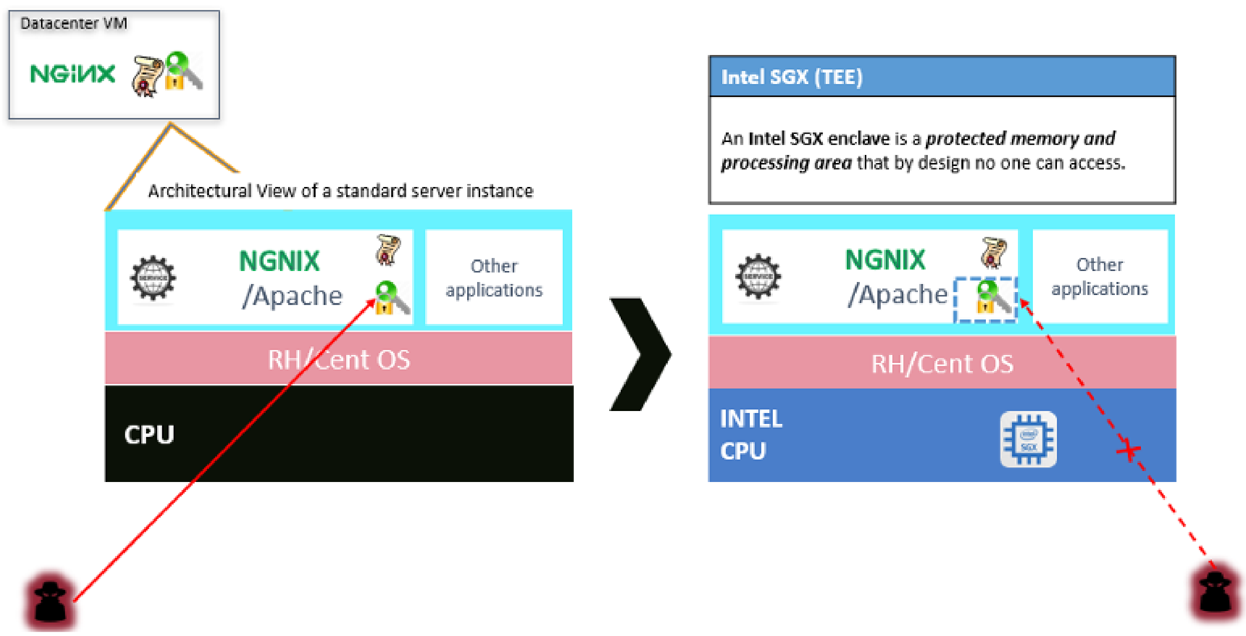


Figure 7 - Machine Identity Secret Key Protection by Intel SGX at Runtime

In terms of hardware, the Intel SGX-enabled server is not all that different from a regular server equipped with Intel CPUs in all other aspects of the computing. These servers are as general-purpose as the computing machines used by enterprise businesses in their data centers. They support the same OSs such as RHEL, CentOS, and other variations of Linux.

While Intel SGX servers are designed for general computing purposes with added security enclave capability, they can also be seamlessly extended to protect machine identities. The Venafi collaboration with Intel SGX helps to eliminate any exposure of the secret key through the full lifecycle of a machine identity with little to no management or operational overhead or cost.

Figure 8 shows a typical deployment of the solution. Venafi Trust Protection Platform deployment and configurations are unchanged. The Intel SKC service can be run on the same server Trust Protection Platform runs on, or it can be run on an independent server. Trust Protection Platform and the SKC service will be connected through standard authentication protocols. From the perspective of the app server, a few lines of change to the app server configuration are the only modifications required.

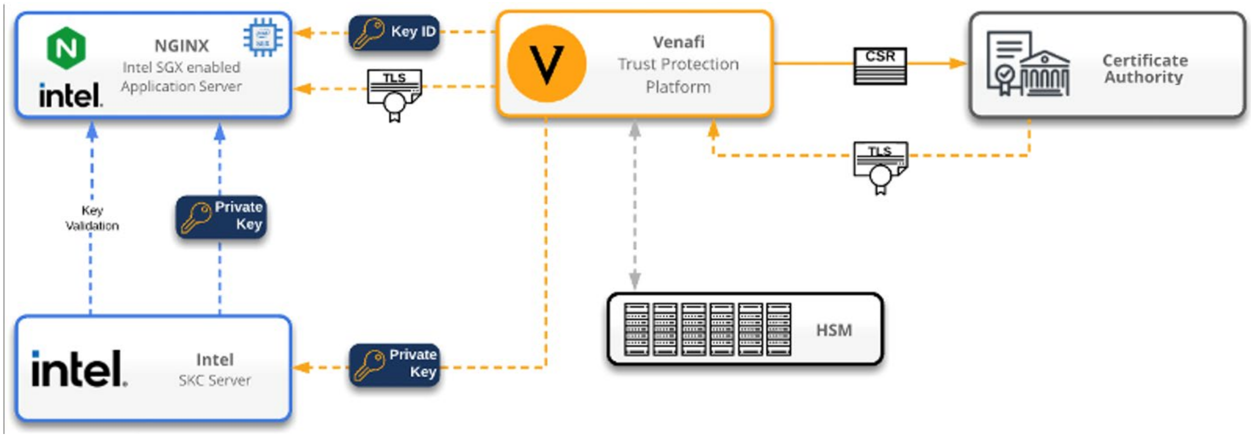


Figure 8 - End-to-End Machine Identity Lifecycle Management Platform

D.2 Solution Architecture

Trust Protection Platform can generate keys directly on the managed server, or it can generate them within the platform and copy them to the managed server. With the SKC integration, another flow is supported. Keys can be generated and stored in the Trust Protection Platform SKC extension, and applications that use these keys can retrieve them at runtime from the Trust Protection Platform SKC extension after proving that the key will be protected in an SGX enclave. The proof is a verifiable SGX quote. The SGX-backed key retrieval from the Trust Protection Platform SKC extension is done by the SKC library that client applications link with. From the client application's perspective, the SKC library is a PKCS#11 module, which is a popular mechanism to protect keys in an HSM. However, unlike HSMs, the SKC solution does not require separate hardware if the application runs on an Intel platform with SGX enabled. This high-level integration is depicted in Figure 9.

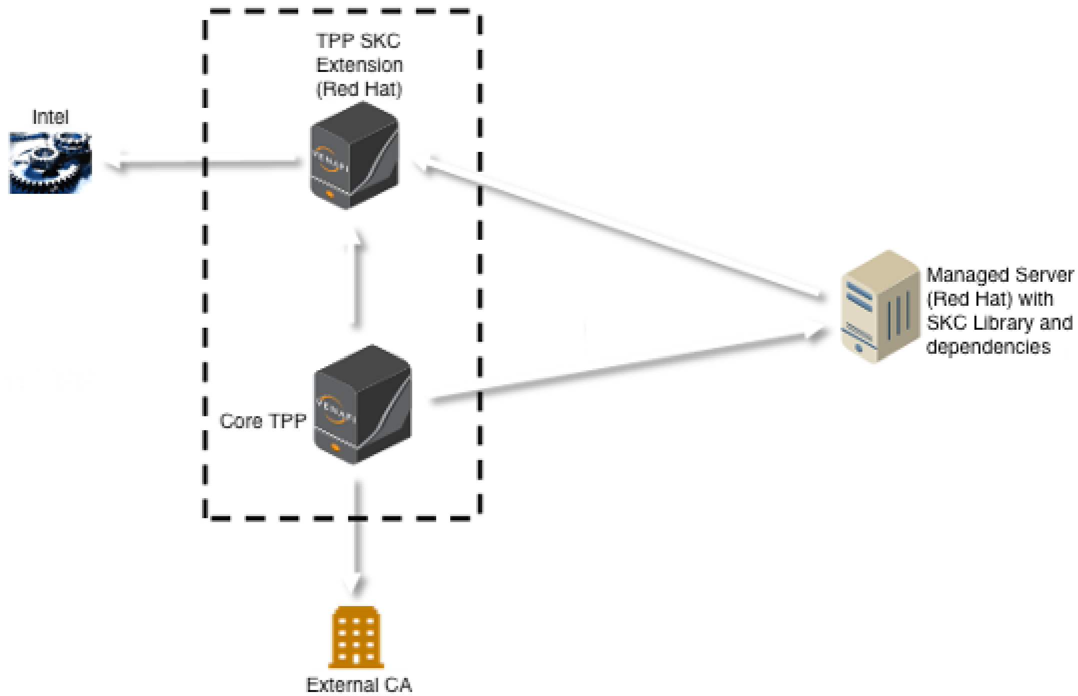


Figure 9 - High-Level View of the Trust Protection Platform and SKC Integration

The Trust Protection Platform SKC extension is a Red Hat server or VM where SKC components will run. Trust Protection Platform communicates with the Trust Protection Platform SKC extension via SSH for installation and configuration and by using Representational State Transfer (REST) API calls at runtime. The connection between the Trust Protection Platform SKC extension and Intel is required to verify the SGX quote that the SKC library presents to prove that the key will be protected in an SGX enclave. Figure 10 shows the communication flows between SKC, Trust Protection Platform, and CA.

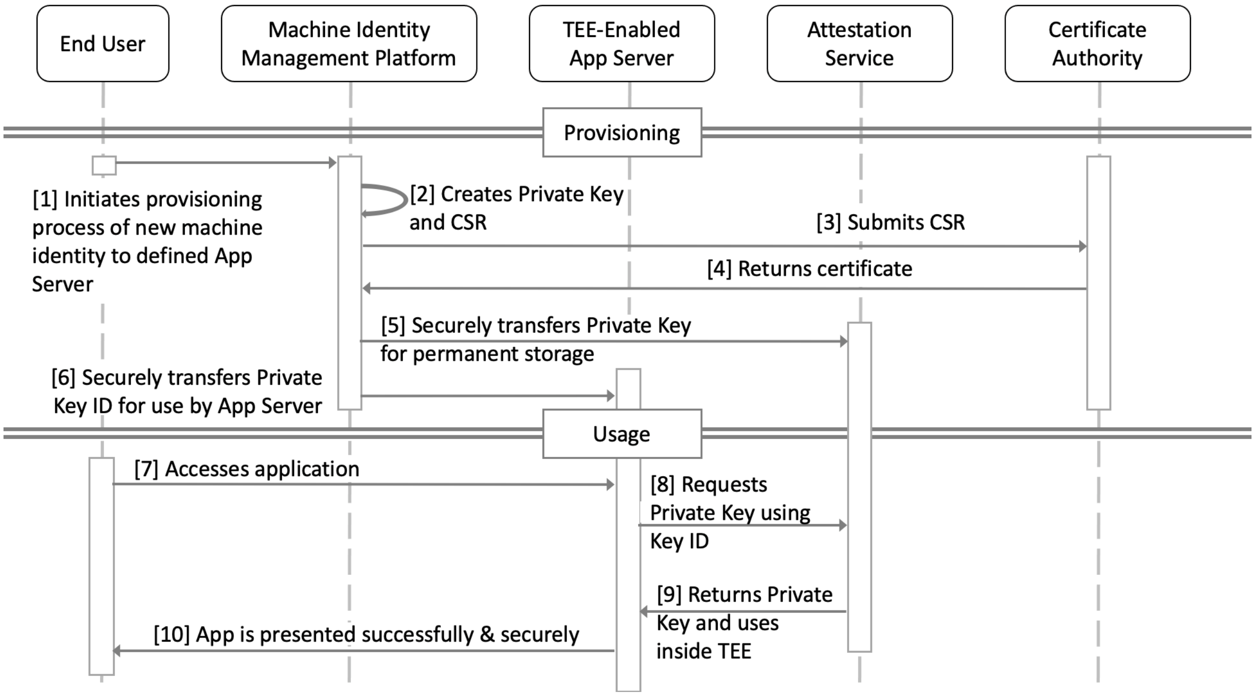


Figure 10 - Communication Flows between SKC, Trust Protection Platform, and CA

D.3 Installation and Configuration

Venafi provides a reference build of Intel's open-source project iSecL (v3.5) as well as a proprietary Venafi implementation of an adaptable driver that facilitates the automated and streamlined installation and configuration of the reference build. This integration is released with the March 24, 2021 version of Intel Security Libraries for Data Center (Intel SecL-DC) iSecL v3.5. For details regarding iSecL, please visit https://github.com/intel-secl/intel-secl. The Venafi Intel SGX Adaptable Driver provides a reference implementation of Intel iSecL v3.5 compatible with Venafi Trust Protection Platform 20.4+ and Intel 3rd Generation Xeon SP servers such as IceLake.

The following prerequisites must be met for a successful deployment.

- **Venafi Trust Protection Platform** - Trust Protection Platform instance runs on a Windows Server. Windows Server 2016 or later is recommended.
- **Intel SGX Server -** An application (e.g., web application) that consumes the machine identity runs on this SGX-enabled IceLake server. RHEL 8.0 or later is recommended.
- **Intel SKC Server -** A set of Intel SKC services (part of the iSecL libraries) runs on a web server. RHEL 8.0 or later is recommended.
- **Network Access -** Venafi Trust Protection Platform needs to have access to the Internet. The Intel SGX and SKC servers need to have both outbound and inbound network access.
- **License -** Servers run RHEL 8.0 or later with a valid activated subscription license.

The following items describe the steps performed on each component to integrate them into a complete solution:

1. **Intel SGX Enabled Application Server Setup**
    a. Enable SGX in BIOS.
    b. Install and configure the web server (e.g., NGINX) and web application or other types of customer applications as you would on a general server machine.
2. **Intel SKC Server Setup**
    a. Ensure a Linux OS is running, preferably RHEL 8.0 or later with the latest updates installed.
3. **Venafi Trust Protection Platform Setup**
    a. This document assumes familiarity with Venafi Trust Protection Platform. The official installation guide is accessible at https://docs.venafi.com/
4. **Adaptable Drive Installation**
    a. Download the Intel SGX driver from the Venafi Marketplace (marketplace.venafi.com). The direct link to the driver is https://marketplace.venafi.com/details/venafi-adaptable-driver-for-intel-sgx/.
    b. There are two PowerShell files: "NGINX Secured by Intel SGX.ps1" and "Deploy Intel SGX Software.ps1".
        i. Copy "Deploy Intel SGX Software.ps1" to the <Venafi Home>\Scripts\AdaptableWorkflow directory on all Trust Protection Platform UI and provisioning servers.
        ii. Copy "NGINX Secured by Intel SGX.ps1" to the <Venafi Home>\Scripts\AdaptableApp directory on all Trust Protection Platform UI and provisioning servers.
    c. Both PowerShell scripts depend on Posh-SSH for SSH client operations. It should be installed automatically when the script is invoked but in the event it is not, you may need to install it manually as described below:
        i. Download zip from https://github.com/darkoperator/Posh-SSH/releases.
        ii. Right-click to ensure Windows isn't blocking the file (General tab).
        iii. Unzip it on your Trust Protection Platform server.
        iv. Move the directory to C:\Windows\system32\WindowsPowerShell\v1.0\Modules.
        v. Rename the directory to "Posh-SSH".
5. **Policy Creation**
    a. Log into the Trust Protection Platform Web Administration ("WebAdmin") console.
    b. Navigate to the **Identity** tree, create a new local user (service account) to be used exclusively by this integration, and grant it Master Admin rights.

c. Navigate to the **Workflow** tree and create a new Reason Code called "START_OF_PROVISIONING".

d. Navigate back to the **Policy** tree and create a policy folder for your SGX-enabled applications called "SGX Apps".

e. Select the "SGX Apps" policy folder and choose **Applications** > **Adaptable**. Assign "NGINX Secured by Intel SGX" for the PowerShell Script, and enter defaults for the Certificate File, NGINX Config File, and Restart NGINX Service. Other mandatory fields will be auto-populated.

f. In the "SGX Apps" policy folder, create three credential objects, one each for:

   i. master admin local user

   ii. SSH credentials for the SKC server

   iii. SSH credentials for the NGINX server

g. Also in the "SGX Apps" policy folder, create an Adaptable Workflow object called "Deploy Intel SGX Software" with the settings as shown in Figure 11:

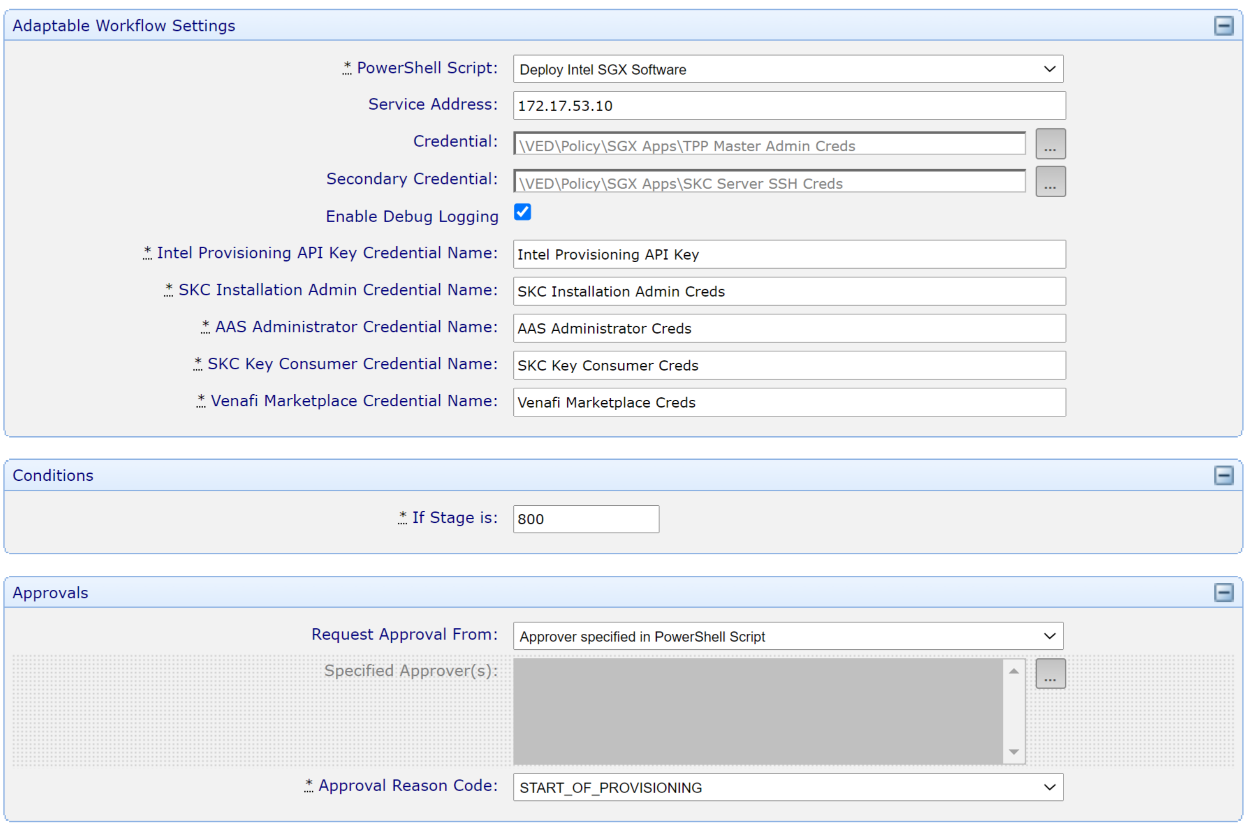


Figure 11 - Create Venafi Adaptable Workflow Object

h. Select the "SGX Apps" policy folder, choose **Settings** > **Workflow,** and then add "Deploy Intel SGX Software" to the Applied Workflows.

6. **New Machine Identity Creation**
    a. Create a device object for the NGINX server, supplying its IP address or DNS-resolvable hostname, and assign the applicable credential object created earlier.
    b. Under the device object, create an Adaptable application object with Enable Debug Logging enabled to help verify everything is OK the first time.
    c. Under the Adaptable application object, create a certificate object, setting the Management Type to Provisioning and assigning a Common Name and Subject Alternative Names. Typically, other certificate and key settings are specified by policy and inherited. This includes the CA. If not assigned by policy, they must be assigned to the certificate object.
    d. Select the certificate object and click the **Renew Now** button. The Trust Protection Platform will generate a new key pair and enroll the certificate using the assigned CA, but provisioning will fail at Stage 800. This is expected and allows the "Deploy Intel SGX Software" script to create other credential objects that are required, including the five specified by the Credential Name fields on the "Deploy Intel SGX Software" workflow object and three that are for internal use only (marked as "DO NOT MODIFY").
    e. Update the following newly created credential objects with appropriate values:
        i. **Intel Provisioning API Key** - Authorizes the SKC server to use SGX
        ii. **SKC Installation Admin Creds** - Master admin for the SKC installation process
        iii. **AAS Administrator Creds** - Administrator credentials for the authentication service
        iv. **SKC Key Consumer Creds** - Used by the library on the managed server
        v. **Venafi Marketplace Creds** - Used to log into the Venafi Marketplace to download dependencies
7. **Machine Identity Provisioning to Target Application**
    a. Return to the certificate object and click the **Retry** button. This time, processing will do the following:
        i. Install the SKC software on the SKC server.
        ii. Install the SKC client library and SGX agent on the NGINX server.
        iii. Provision the private key to the Secure Key Cache.
        iv. Provision the certificate and CA chain to the NGINX server.
        v. Update the NGINX configuration to reference the SKC-based private key.
    b. Restart the NGINX service (if the Adaptable Application is configured to do so).

Appendix E—Acronyms and Other Abbreviations

Selected acronyms and abbreviations used in the report are defined below.

API	Application Programming Interface
BIOS	Basic Input/Output System
CA	Certificate Authority
CPU	Central Processing Unit
CSR	Certificate Signing Request
DB MEK	Database Master Encryption Key
DNS	Domain Name System
FOIA	Freedom of Information Act
GB	Gigabyte
HSM	Hardware Security Module
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
IR	Interagency or Internal Report
ISecL	Intel Security Libraries
IT	Information Technology
ITL	Information Technology Laboratory
JSON	JavaScript Object Notation
JWT	JSON Web Token
KMS	Key Management System
MMC	Microsoft Management Console
NIST	National Institute of Standards and Technology
OS	Operating System
PCK	Provisioning Certification Key
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure

RAM	Random Access Memory
REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
SA	System Agent
SDK	Software Development Kit
SGX	(Intel) Software Guard Extension
SKC	(Intel) Secure Key Caching
SP	Special Publication, Scalable Processor
SP2	Service Pack 2
SSH	Secure Shell
SWK	Software Wrapping Key
TB	Terabyte
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
URI	Uniform Resource Identifier
VM	Virtual Machine
VPN	Virtual Private Network