Enhance Cloud Security Posture Management with Confidential VM

A solution guide using Confidential VM, Vault & Intel Security Libraries

Authors

Yiwen Li
Intel Corporation

Haidong Xia
Intel Corporation

David Lu
Intel Corporation

Heqing Zhu
Intel Corporation

Michihiro Koyama
Intel Corporation

Timothy Knoll
Intel Corporation

Sudeendra Raj
Intel Corporation

1. Introduction

The public cloud attracts enterprise customer adoption with its convenient deployment of services and abundant choices of configuration. However, while public cloud service providers offer extensive security for their hosting services, some still debate its trustworthiness and consider them an "untrusted boundary." Sensitive data is processed by the Cloud Service Provider (CSP), typically in a multi-tenant environment, posing significant risks if a security breach were to occur. Therefore, enterprise may choose to store sensitive data and secrets in the private cloud, which is considered within the "trusted boundary". Often these secrets are stored in Hardware Security Module (HSM) or software Key Management System (KMS) such as HashiCorp Vault.

One critical problem faced by customers is that there exists a "trust gap" between the public cloud and the private cloud. Although data stored in the private cloud can be considered secure, in most cases, customer applications are deployed and run in the public cloud and require secrets such as private keys to perform functionality. Is it possible to ensure that sensitive data can be securely communicated between the "trusted boundary" (private cloud) and the "untrusted boundary" (public cloud)?

Recently, Confidential Virtual Machine (VM) technology has been introduced by leading CSPs which can significantly improve cloud security posture. A Confidential VM is a VM created on a platform that supports confidential computing using trusted computing technologies, such as Intel® Software Guard Extensions (Intel® SGX). As enterprises move more services into the public cloud, more secrets will be exposed to these application workloads. Confidential VM provides an extra security mechanism that is necessary to protect this sensitive data in the public cloud.

How can a public cloud instance attest that it is a Confidential VM? Is it possible to prevent secrets from being sent to a non-confidential, traditional VM? This requires a new attestation solution which needs to work with secrets managers, HSMs, and key management services across public and private clouds. This paper presents a new security mechanism to connect the Confidential VM in the public cloud with the trusted private cloud. A validated solution to solve the "trust gap" problem by leveraging Intel Confidential Computing technology (Intel SGX empowered Confidential VM), Intel® Security Libraries (Attestation Service and Key Broker Service), and the HashiCorp Vault Key Management System is proposed.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	07/28/2022	Initial draft (for internal review)
002	08/22/2022	Revision to address internal review feedback

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
VM	Virtual Machine
Intel® SecL	Intel® Security Libraries
HSM	Hardware Security Module
KMS	Key Management System
CSP	Cloud Service Provider
Confidential VM	Confidential Virtual Machine
CSPM	Cloud Security Posture Management
TEE	Trusted Execution Environment
SGX	Software Guard Extensions
FIPS	Federal Information Processing Standard
PKCS	Public Key Cryptography Standards
TDX	Trust Domain Extensions

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
HashiCorp Vault	https://www.vaultproject.io/
AWS S3 Buckets Exposed Millions of Facebook Records	https://www.securityweek.com/aws-s3-buckets-exposed-millions-facebook-records https://www.cnbc.com/2019/04/03/facebook-dips-on-report-that-user-records-were-exposed-on-aws-servers.html
Introduction to Intel SGX	https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html
Intel SGX Product Brief	https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html
Vault HSM Integration – Seal Wrap	https://learn.hashicorp.com/tutorials/vault/seal-wrap
Vault Enterprise HSM Security Details	https://www.vaultproject.io/docs/enterprise/hsm/security
G-Core Labs: New high-frequency machines and virtual machines with Intel SGX	https://gcorelabs.com/blog/new-high-frequency-machines-and-virtual-machines-with-intel-sgx/
Why did we integrate Intel SGX into G-Core Labs Cloud?	https://gcorelabs.com/blog/why-did-we-integrate-intel-sgx-into/

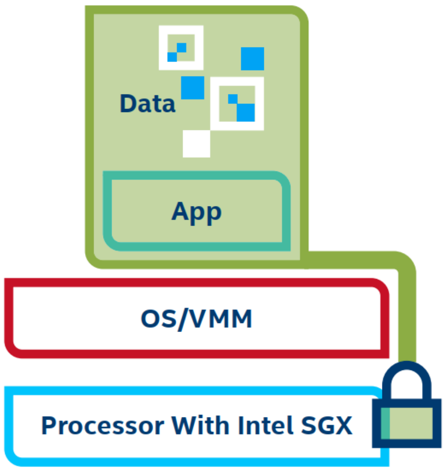
2. Cloud Security Posture Management

Why cloud security posture management is important

Cloud Security Posture Management (CSPM) is a continuous process of cloud security improvement and adaptation to reduce the likelihood of a successful attack. It is especially needed for the public cloud environment. Enterprise use of the public cloud can contain tens of thousands of different regions, accounts, and resources. This size and complexity could easily cause permission misconfigurations that lead to a data breach. For example, in 2019, 540 million member records of a leading social media company were exposed by an unsecure AWS S3 bucket. Besides cloud misconfigurations, the fact that all the data is in the hands of the CSP keeps users from fully controlling the privacy of their data.

Use Confidential VM to improve the security mechanism

Confidential Computing technology uses hardware to isolate data. Data being processed in the memory is protected by the Trusted Execution Environment (TEE). This secure and isolated environment prevents unauthorized access or tampering with applications and data while they are in use. Therefore, Confidential Computing can increase the security level of organizations that manage sensitive data.



Intel® Software Guard Extensions (Intel® SGX) offers hardware-based memory encryption that isolates specific application code and data in memory. Intel SGX allows user-level code to allocate private regions of memory, called enclaves, which are designed to be protected from processes running at higher privilege levels. Intel SGX enables Confidential Computing solutions that allow users to

- **Enhance Confidentiality and Integrity:** protects sensitive data even in the presence of privileged malware at the OS, BIOS, VMM, or SMM layers.
- **Remotely Attest and Provision:** a dependent part can verify an application enclave's identity and enhance security of provisioning keys, credentials, and other sensitive data in the enclave.
- **Reduce Attack Size:** Bypassing the OS and VM, applications can communicate directly with the CPU.

Figure 1. Data security using hardware-based technology Intel SGX

A Confidential VM is a Virtual Machine that leverages Confidential Computing technology to provide a secure execution environment. As security becomes an increasingly important issue for customers, especially for public cloud deployment, major CSPs have rolled out their Confidential VM options for customers. Table 3 below summarizes the currently available Confidential VM options based on Intel SGX technology.

Table 3. Confidential VM options offered by CSPs Supporting Intel SGX

	CSP	Instance Type
Cloud Confidential VM	Azure	DCsv2-series and DCsv3/DCdsv3-series
	IBM Cloud	Bare Metal instances based on z15 series
	Alibaba Cloud	ECS Bare Metal instances
	G-Core Labs	SGX instances

How to keep secrets secure

Secret Management Software

Secrets, or digital authentication credentials, such as passwords, keys, or tokens are used extensively in applications and services deployed in the cloud environment. It is crucial to manage secrets securely to prevent security breaches. Usually, secrets are stored in the private cloud for safety reasons and secret management software is used to properly manage the secrets.

Vault is a tool for securely managing secrets. The key features of Vault include:

- **Secure Secret Storage**: Arbitrary key/value secrets can be stored in Vault. Vault encrypts these secrets prior to writing them to persistent storage
- **Dynamic Secrets:** Vault can generate secrets on-demand for some systems, such as AWS or SQL databases. For example, when an application needs to access an S3 bucket, it asks Vault for credentials, and Vault will generate an AWS keypair with valid permissions on demand. After creating these dynamic secrets, Vault will also automatically revoke them after the lease is up.
- **Data Encryption:** Vault can encrypt and decrypt data without storing it. This allows security teams to define encryption parameters and developers to store encrypted data in a location such as a SQL database without having to design their own encryption methods.
- **Leasing and Renewal:** All secrets in Vault have a lease associated with it. At the end of the lease, Vault will automatically revoke that secret. Clients are able to renew leases via built-in renewal APIs.
- **Revocation:** Vault has built-in support for secret revocation. Vault can revoke not only single secrets, but a tree of secrets, for example all secrets read by a specific user, or all secrets of a particular type. Revocation assists in key rolling as well as locking down systems in the case of an intrusion.

HSM for High Security

A **Hardware Security Module (HSM)** is a specialized, highly trusted physical device that performs all major cryptographic operations, such as encryption, decryption, authentication, key management, etc. HSMs have a robust OS and restricted network access protected via a firewall. HSMs are tamper-resistant and tamper evident devices. These features make HSMs the ideal tool to help store and protect secrets in the private cloud environment.

Vault integration with HSM for Secure Secrets Management

HashiCorp Vault provides a software platform that securely manages secrets and protects sensitive data. In combination with HSMs, Vault can enhance the security of secrets management.

Vault HSM integration provides the following three key functionalities:

- Root Key Wrapping: Vault protects its root key (Master key) by transiting it through the HSM for encryption rather than splitting into key shares
- Automatic Unsealing: Vault stores its encrypted root key in storage, allowing for automatic unsealing
- Seal Wrapping: Provides Federal Information Processing Standard (FIPS) key storage-conforming functionality for critical security parameters

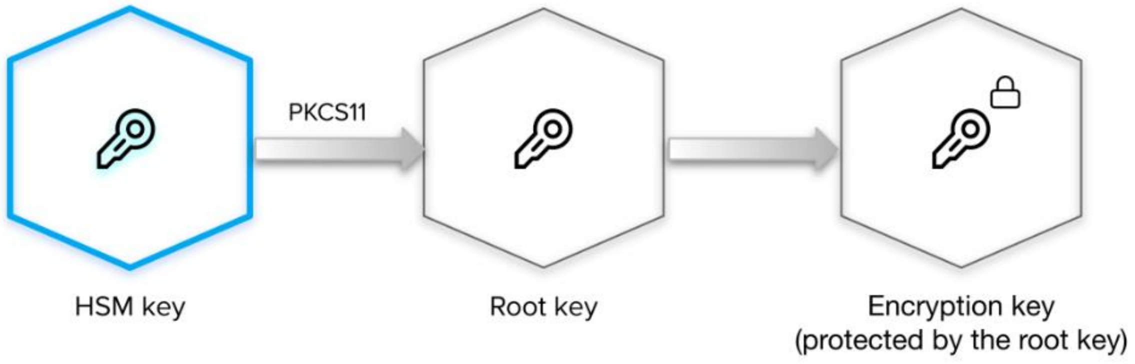


Figure 2. Vault Enterprise HSM integration – Seal Wrap

Vault pulls its encrypted root key from storage and transits it through the HSM for decryption via **PKCS #11 API**. Once the root key is decrypted, Vault uses the root key to decrypt the encryption key to resume Vault operations (Shown in Figure 2.)

Trust boundaries and secure communication between Public Cloud and Private Cloud

Trust boundaries define areas in a deployment environment with different security assumptions. A “trusted boundary” refers to the deployment environment that is considered secure because of reinforced Software and Hardware protection, and restricted access control. An “untrusted boundary” is the environment with the assumption of less security guarantees, and often poses potential security risks. In a modern cloud environment, the public cloud, although having many security measures in place, in some cases is considered by customers as an “untrusted boundary”, mainly because the underlying infrastructure is owed and managed by the CSP. While the private cloud or on-prem data center is considered the “trusted boundary”. Although it is safe to manage sensitive data, such as secrets, in the “trusted boundary”. It is not realistic to deploy everything only in the “trusted boundary”. In fact, enterprise customers run the majority of their software services and platforms in the public cloud. These software services, running within the "untrusted boundary", require secrets from the "trusted boundary". Therefore, it becomes critical to come up with a solution for secure communication across the trust boundaries. In this paper, we present such a solution with Intel Confidential Computing technology, Intel Attestation Service, and Key Broker Service.

3. Incremental Security Mechanisms to Enhance Cloud Security Posture

To effectively enhance cloud security posture to better protect enterprise customer's sensitive data, we propose three "**good – better – best**" incremental security mechanisms for multi-cloud deployment.

- **Good**: store sensitive data in the private cloud, using Vault (secrets management software) + HSM + public cloud service
  This mechanism stores and manages sensitive user data in the private cloud. Thus can effectively prevent security breaches in the public cloud, and therefore is a "good" strategy. In this situation, the customer has service deployed in the public cloud, and may still need to use secrets for the service. Therefore, this security model can be improved to protect computing in the public cloud.
- **Better**: store sensitive data in the private cloud, and keep secrets in Confidential VM in the public cloud
  In addition to storing sensitive data in the trusted private cloud, this mechanism adds another security layer by leveraging the Confidential VM technology to protect sensitive data when it has to be used in the untrusted public cloud boundary.
- **Best**: store sensitive data in the private cloud, keep secrets in Confidential VM in the public cloud, and use remote attestation to verify that the environment is secure before transferring secrets
  Only allow secrets to be transferred from the private cloud to the public cloud when it is attested and verified that the public cloud is secured with Confidential VM technology. This method is strict in the application of its attestation requirement and thus provides the best security.

Our "**good-better-best**" incremental security mechanism recommendation is summarized in the following diagram:

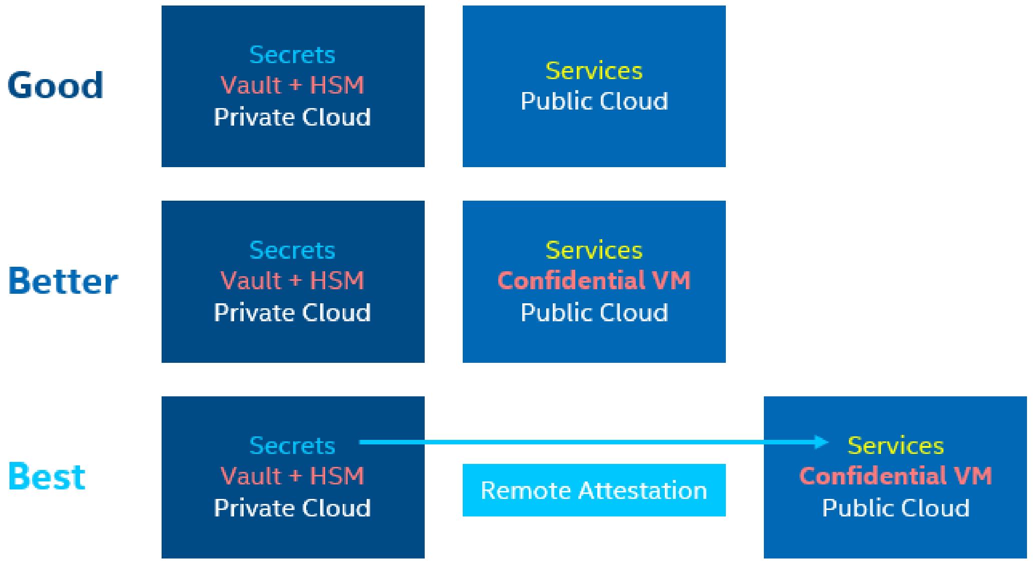


Figure 3. "Good-Better-Best" security mechanism recommendation

4. Security Services Provided by Intel® Security Libraries

Intel Security Libraries ("ISecL" or "Intel Security Libs") is an open-source remote attestation implementation comprising of a set of building blocks that utilize Intel security features to discover, attest, and enable critical foundation security and confidential computing use-cases. It applies the remote attestation fundamentals and standard specifications to maintain a platform data collection service and an efficient verification engine to perform comprehensive trust evaluations. ISecL-DC middleware provides building blocks (Libraries and components) that discover, attest, and utilize Intel security features to enable critical cloud security & confidential computing use-cases. It supports attestation of different TEEs (TPM and Intel SGX), and different use cases for Application Data Protection & Key Management. The components that are relevant to the solution in this paper are illustrated in Figure 4.

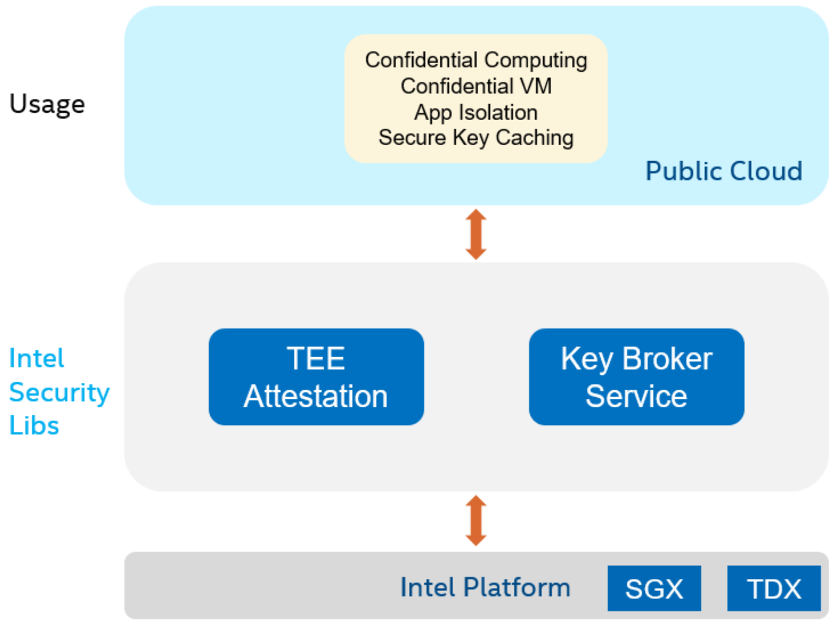


Figure 4. Intel® Security Libraries Key Components

The Intel Security Libraries have the following important security services that are key components in our solution:

- **Intel SGX Attestation Service**
- **Key Broker Service (KBS)**

4.1 Attestation Service

An attestation service verifies the trustworthiness of a workload or computing asset and is the foundation for confidential computing. The ISecl attestation service consists of three constituent microservices: the TEE Caching Service (TCS), the Quote Verification Service (QVS), and the Appraisal Service (AS). TCS caches the Intel SGX collaterals (TCBInfo, CRL, etc) used by QVS to verify if the evidence provided by a workload or compute asset is legitimate and update to date.AS is an additional service to help create policies to verify the workload itself, for example, its measurement hash, signer, etc.

The generic attestation service (including TCS, QVS, and AS) architecture is illustrated in the following diagram, with interaction with a relying party (can be KBS, described below), and a relying party client.

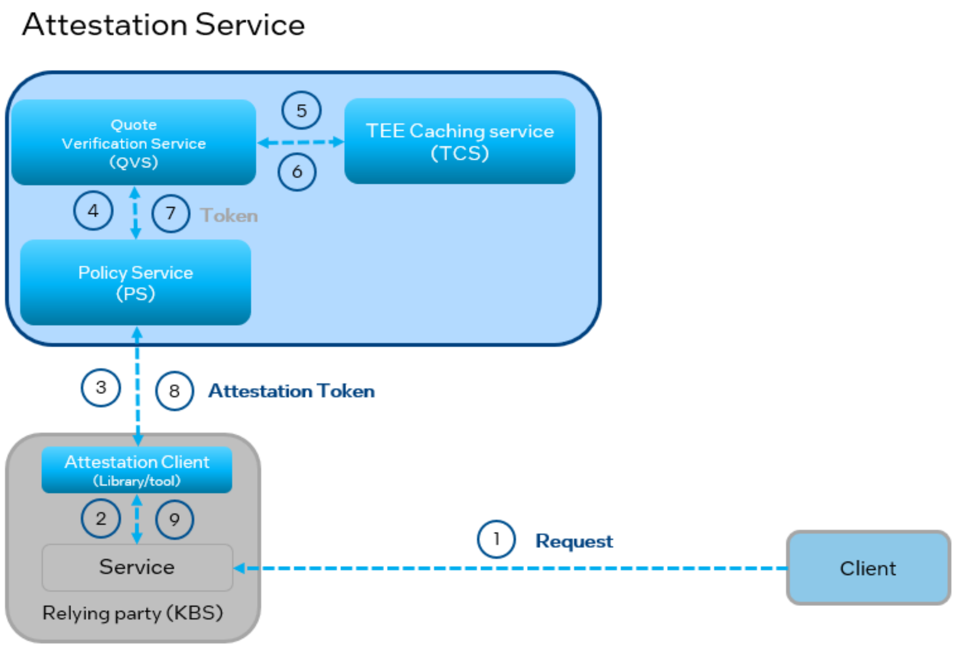


Figure 5. Attestation Service Architecture

For the solution proposed in this paper, the client will be confidential VM. The relying party is the Key Broker Service.

4.2 Key Broker Service (KBS)

Key Broker Service (KBS) manages and releases keys based on key policies which traditional key management service (KMS) platforms do not provide. KBS uses KMS as a backend for key management and storage. KBS acts as a broker in front of KMS for additional key policy verification.

KBS architecture is illustrated in the following diagram:

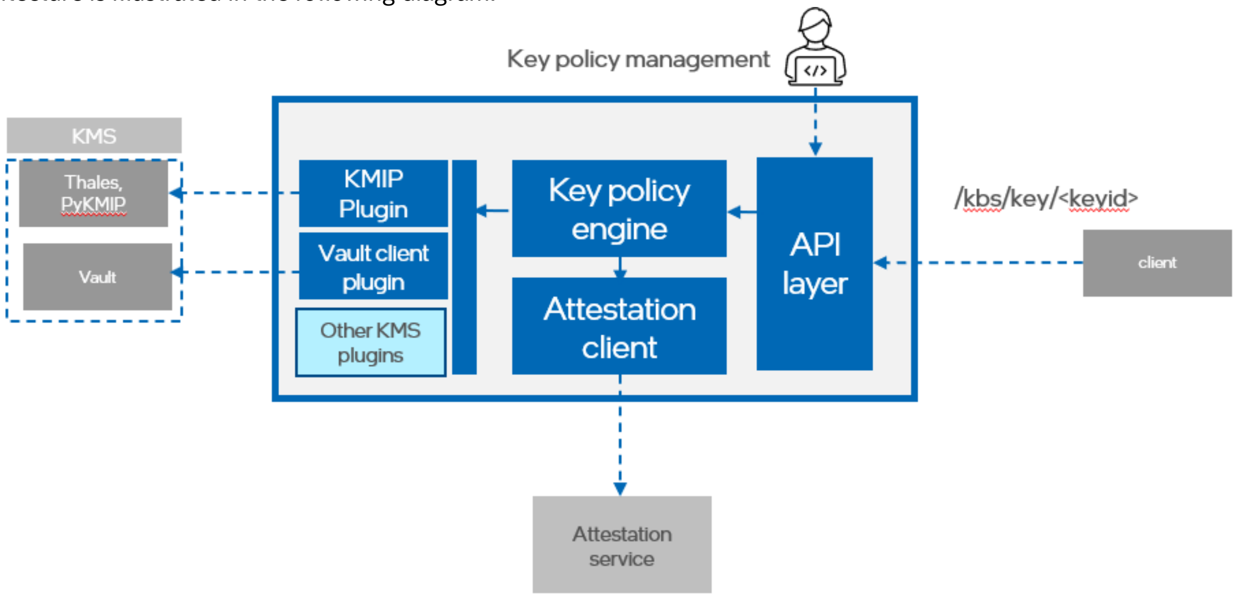


Figure 6. Key Broker Service Architecture

KBS includes four major components:

- API layer – a client interface and a policy management interface for KBS administrators.
- Key policy engine – manages policies associated with each key
- Attestation client – a client module to interact with attestation service to verify the evidence from KBS client
- KMS client module – a plug in model to communicate with backend KMS. Different plugins can be added for different types of KMS, such as KMIP server, HSM, Vault, etc.

KBS administrators are responsible for the creation of KMS policy. This policy defines the conditions necessary for key distribution (e.g., only after successful Intel SGX attestation).

When a client requests a key from KBS, it will be challenged based on the policy associated with the key. For example, the policy may define that the client needs to run in a trusted execution environment, such as Intel SGX. In this case, the client will be challenged with an Intel SGX attestation request and must provide an Intel SGX quote as evidence to KBS. KBS then calls an external attestation service to verify the evidence. Based on the attestation result, KBS makes the decision to release the key or reject the request.

4.3 KBS with Vault

KBS with Vault is a tool for securely accessing *secrets* from the ISecL-DC Key Broker Service (KBS) and loading them to an SGX-protected memory (called Intel SGX enclave) in the application memory space.

A *secret* is anything that you want to tightly control access to, such as API keys, passwords, or certificates. KBS Vault provides a unified interface to any secret, while providing tight access control and recording a detailed audit log.

KBS acts as an access broker, providing a policy enforcement layer between a relying party and secrets the relying party wants to access. In the case of SGX, KBS applies a policy requiring a valid SGX attestation to release keys. Vault acts as a specific KMIP backends.

With Vault, KBS performs the Intel SGX enclave attestation to ensure that the application will store the keys in a genuine Intel SGX enclave. Application keys are wrapped with an enclave public key by KBS prior to transferring to the application enclave. Consequently, application keys are protected from infrastructure admins, malicious applications and compromised HW/BIOS/OS/VMM.

1. The KBS can accept a pre-generated key to store in Vault (include the “key_string” and do not include any key IDs, only the policy ID)
2. The KBS can accept a request to generate a new key itself and store it in Vault (include the algorithm, key length, and transfer policy ID, but do not include the key string or any key IDs)
3. The KBS can accept a key ID handle for a key that already exists in Vault, associating that key with an existing policy (Include the “kmip_key_id” and do not include the key string)

These all use the same API, POST /kbs/v1/keys:

```
{
   "key_information": {
    "algorithm": "string",
    "curve_type": "string",
    "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "key_length": 0,
    "key_string": "string",
    "kmip_key_id": "string"
  },
   "label": "string",
  "transfer_policy_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "usage": "string"
}
```

5. Deployment Model

This section describes the best deployment model described in Section 3, which leverages the two services (attestation service and key broker service) provided by Intel SecL and its secure key caching solution.

The deployment model is illustrated in Figure 7. below and describe as following:

- **Workloads in the public cloud**
  Customer workloads are deployed to a CSP, such as Azure, that provides confidential VMs. Confidential VMs enables an entire workload or part of a workload (trusted part) to be protected outside of the trust boundary, in the public cloud using confidential computing technology such as Intel SGX.
- **Key Management in the private cloud**
  Sensitive data such as secrets and keys are kept in the private cloud or enterprise data center. These sensitive data should be protected in transit and should only be released to workloads running within a TEE enclave for protection while in use. In Figure below, the secrets are protected either with Vault, a software HSM, or HSM.
- **Key policy management with ISecL Key Broker Service**
  Key policy defines when, where, and how the keys are released to the workload running public cloud. ISecL Key broker service provides the API for administrator to manage the key policy. It enforces the attestation of the workload, and identifies the identity of the workload so the keys will not be released to malicious workload or attackers.
- **Attestation Service**
  Attestation service verifies the evidence provided by the workload running in the public cloud. It checks that the workload is running in a TEE enclave, a trusted region outside of the trust boundary to which keys may be released.

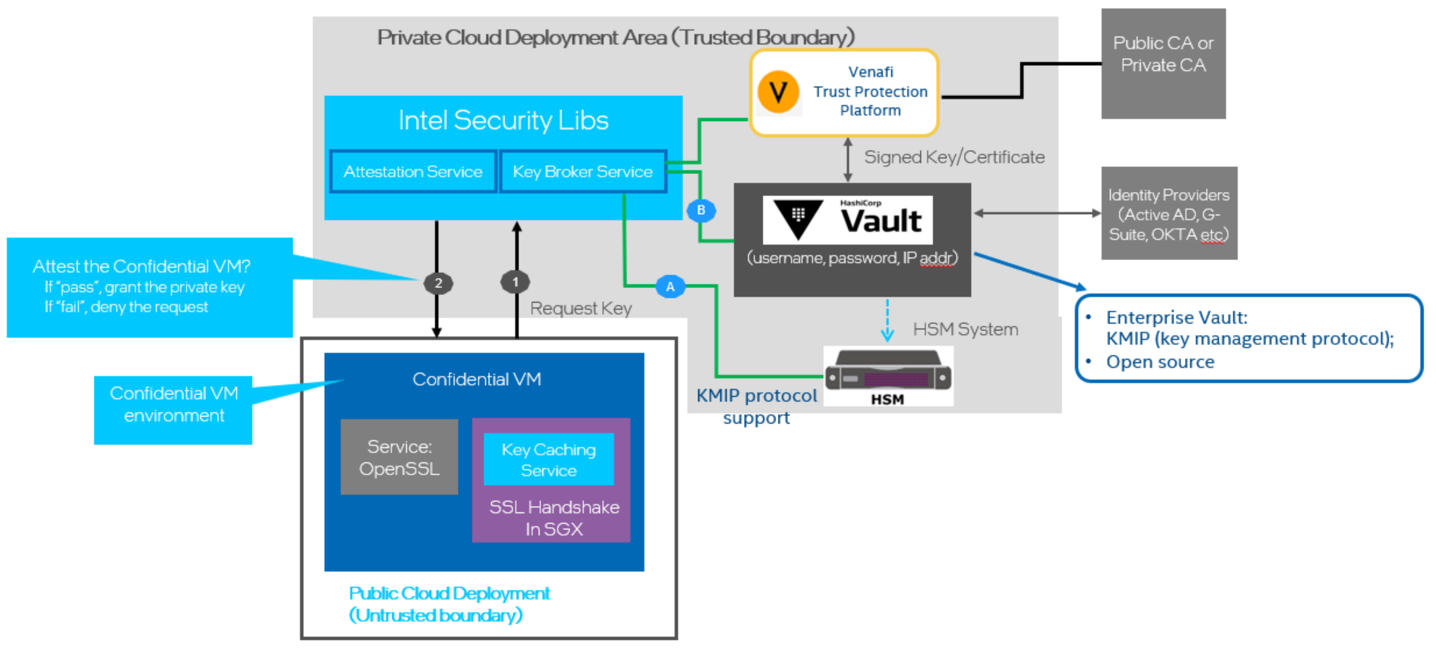

Private Cloud Deployment Area (Trusted Boundary)
Public CA or Private CA
Venafi Trust Protection Platform
Intel Security Libs
Attestation Service
Key Broker Service
Signed Key/Certificate
HashiCorp Vault
(username, password, IP addr)
Identity Providers (Active AD, G-Suite, OKTA etc)
Attest the Confidential VM? If "pass", grant the private key If "fail", deny the request
Request Key
HSM System
Enterprise Vault: KMIP (key management protocol); Open source
Confidential VM environment
Confidential VM
Service: OpenSSL
Key Caching Service
SSL Handshake In SGX
KMIP protocol support
HSM
Public Cloud Deployment (Untrusted boundary)
1
2
A
B

Figure 7. Deployment Model Overview

6. Proof of Concept Setup and Configuration

This section provides a proof-of-concept setup for the "best" solution described above. It includes the setup and configuration of Vault, KBS, and customer workload (using Nginx application) as an example. The details of the Attestation Service are described in Section 4.1 above.

6.1 Vault setup

- Install Hashicorp Vault
- After installation, create a "vault.conf" configuration file:

```
storage "raft" {
  path = "./vault/data"
}

listener "tcp" {
  address = "0.0.0.0:8200"
  tls_disable = "true"
}

api_addr = "http://0.0.0.0:8200"
cluster_addr = "https://127.0.0.1:8201"
ui = true
```

- Create the Vault data directory. Note that this must match the relative path in the `vault.conf` configuration file specified for the raft storage path (in the example, `./vault/data`)
- Start the Vault instance using the configuration file:

```
vault server -config=vault.conf
```

- Initialize Vault. Note that the `VAULT_ADDR` variable must be set for many Vault interactions.

```
export VAULT_ADDR='http://127.0.0.1:8200'
vault operator init
```

Included in the output will be a set of unseal keys, along with a root access token. Keep these for later use. Vault is sealed by default and requires a quorum of at least three separate unseal keys to be unsealed.

Sample output of vault operator init:

```
Unseal Key 1: we...vZoFr
Unseal Key 2: O3M...cjj0ke
Unseal Key 3: Il...js4EWSoYo
Unseal Key 4: DwX...avm
Unseal Key 5: qrx...9Ywcy
Initial Root Token: s.H9...Ek
```

- Unseal Vault. This will require the same "`vault operator unseal`" command executed three times, providing a different unseal key each time:

```
export VAULT_TOKEN=<root access token from the "vault operator init" output>
```

Sample output of Vault unsealing:
 # vault operator unseal

```
Key (will be hidden):
Sealed: true
Key Shares: 1
Key Threshold: 3
Unseal Progress: 1
```

When all three unseal key shares have been provided:

```
# vault operator unseal

Key (will be hidden):
Sealed: false
Key Shares: 3
Key Threshold: 3
Unseal Progress: 3
```

- enable a key-value secrets engine for the Key Broker:

```
vault secrets enable -path=keybroker kv
```

6.2 KBS setup

- Configure the `kbs.env` installation answer file including Vault as the backend key management service.

```
SERVER_PORT=<KBS port number, 9443 by default>
SERVER_IP=<KBS IP address or hostname>
ENDPOINT_URL=https://<KBS IP or hostname>:<KBS port>/kbs/v1
CMS_BASE_URL=https://<CMS IP or hostname>:<CMS port>/cms/v1/
AAS_API_URL=https://<AAS IP or hostname>:<AAS port>/aas/v1
KBS_SERVICE_USERNAME=<username for KBS service account>
KBS_SERVICE_PASSWORD=<Password for KBS service account>
TLS_COMMON_NAME="KBS TLS Certificate"
SKC_CHALLENGE_TYPE="SGX"
CMS_TLS_CERT_SHA384=<SHA384 hash of the CMS TLS Certificate>
TLS_SAN_LIST=<Comma-separated list of KBS IP address(es) and hostname>
BEARER_TOKEN=<Installation access token from AAS or populate-users.sh script>
SESSION_EXPIRY_TIME=60
KEY_MANAGER=VAULT
CLIENT_TOKEN=<Vault access token, from the "vault operator init" step>
KMIP_SERVER_IP=<Vault IP address>
KMIP_SERVER_PORT=<Vault port>
```

- Install KBS v4.1 with Vault plugin extension on Ubuntu 20.04

After the KBS is installed and started, the KBS log at `/var/log/kbs/kbs.log` should show output indicating the correct URL for the Vault backend and a successful Vault client initialization:

```
INFO...: Vault Address: http://127.0.0.1:8200; name=default
INFO...: vaultclient/vaultclient:InitializeClient() Vault client initialized; name=default
```

- Create an RSA key by calling KBS management API and the key is stored in vault
- ALTERNATE RSA key step: The KBS build collaterals include sample scripts (found in `binaries/kbs_scripts/`). The "run.sh" script will create an RSA keypair and corresponding certificate for use with Nginx.

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

  The output will include a key ID and certificate path:

  ```
  Key Certificate Path: /<path>/binaries/kbs_script/output/<ID string>.crt
  Created Key: <Key ID>
  ```

  Note the certificate path and key ID for later use.

- Create key policy for the RSA key generated above to only release the key after successful Intel SGX attestation
- Create certificate for the RSA key above (used by nginx workload later)

6.3 Deploying the SKC Library

The SKC library is the actual Intel SGX enclave code. This is the component that will actually create an enclave, send an Intel SGX quote to the KBS to request a key, and perform all of the cryptographic functions needed by Nginx using that key in the secure enclave.

- Copy `skc_library.tar`, `skc_library.sha2` and `skclib_untar.sh` from the binaries/ directory to a directory on the SGX node or confidential VM. Use the `skclib_untar.sh` script to untar the required library files.

```
./skclib_untar.sh
```

- Update the create_roles.conf file

```
AAS_PORT=<AAS port number>
AAS_IP=<AAS IP address>
SKC_USER=<Username of the SKC user that will be created in the following step.  "skcuser" for example>
SKC_USER_PASSWORD=<SKC user password>
ADMIN_USERNAME=<Username for an account with Administrator permissions on the AAS>
ADMIN_PASSWORD=<Password for the AAS administrator account>
PERMISSION="nginx,USA"
```

- Execute the `skc_library_create_roles.sh` script to create the needed roles and user on the AAS

```
./skc_library_create_roles.sh
```

The output of this script will include a bearer token, used in the next step.

- Update the `skc_library.conf` file.

Note that the CMS information is provided twice; in some deployments, both the Cloud Service Provider (CSP) and the workload owner (the "enterprise") will have their own CMS services, and this allows both to be specified. In this case it's expected only one CMS will be used, and the same information can be provided to both variables.

```
KBS_HOSTNAME=<hostname of KBS>
KBS_IP=<IP address of KBS>
KBS_PORT=<KBS port number, 6443 by default>
CMS_IP=<CMS IP address>
CMS_PORT=<CMS port, 8445 by default>
CSP_SCS_PORT=<SCS port number, 9000 by default>
```

```
CSP_SCS_IP= <SCS IP address>
CSP_CMS_IP=<CMS IP address>
CSP_CMS_PORT=<CMS port, 8445 by default>
SKC_USER=<username for the SKC user defined in the create_roles.conf file>
SKC_TOKEN=<Bearer token from the skc_library_create_roles.sh script>
```

- Deploy the SKC library

```
./deploy_skc_library.sh
```

6.4 Nginx application installation on Intel SGX node / Confidential VM

- Install nginx
- update openssl configuration /etc/ssl/openssl.cnf to use pkcs11 engine

openssl_conf = openssl_def

[openssl_def]
engines = engine_section
oid_section = new_oids

[engine_section]
pkcs11 = pkcs11_section

[pkcs11_section]
engine_id = pkcs11
dynamic_path =/usr/lib/x86_64-linux-gnu/engines-1.1/pkcs11.so
MODULE_PATH =/opt/skc/lib/libpkcs11-api.so
init = 0

- Update the Nginx configuration in /etc/nginx/nginx.conf to include the path to the SSL certificate (generated in the run.sh step) and set the SSL certificate key to use PKCS11 to retrieve the key from the KBS at server start.

```
server {
        listen       2443 ssl http2 default_server;
        listen       [::]:2443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "<Path to the certificate generated using run.sh>";
        ssl_certificate_key "engine:pkcs11:pkcs11:token=KMS;object=RSAKEY;pin-value=1234";
```

- Create a new file in /root/ named “keys.txt.” This file is referenced in the SKC library and configure it to know which key ID to retrieve from the KBS. This key ID should be the ID output from the run.sh step and must correspond to the certificate created.

```
pkcs11:token=KMS;id=<Key ID>;object=RSAKEY;type=private;pin-value=1234;
```

- run Nginx

Once Nginx starts, it calls into the OpenSSL engine to get its TLS certificate and RSA private key, which calls into the PCKS11 engine provided by SGX client library, which triggers key request to KBS with Intel SGX attestation involved. After successful Intel SGX attestation, the RSA private key is wrapped with session generated in the Intel SGX enclave in transit, and eventually stored in the Intel SGX enclave created for Nginx.

7. Summary

This paper proposes utilizing Intel Confidential Computing technology to deploy customer software services inside a Confidential VM in the public cloud, and leveraging Intel SGX Attestation Service and Key Broker Service to first verify the secure Confidential VM environment, and then safely transfer the secrets from the "trusted" private cloud to the "untrusted" public cloud. This solution closes a gap between the different trust boundaries in the multi-cloud environment, and can significantly enhance Cloud Security Posture Management.

MMYY/DN/WIPRO/PDF 691286-001US