
KubeCon
CloudNativeCon
North America 2019

TEE-based KMS Plugin for encryption of Kubernetes Secrets

Raghu Yeluri & Haidong Xia

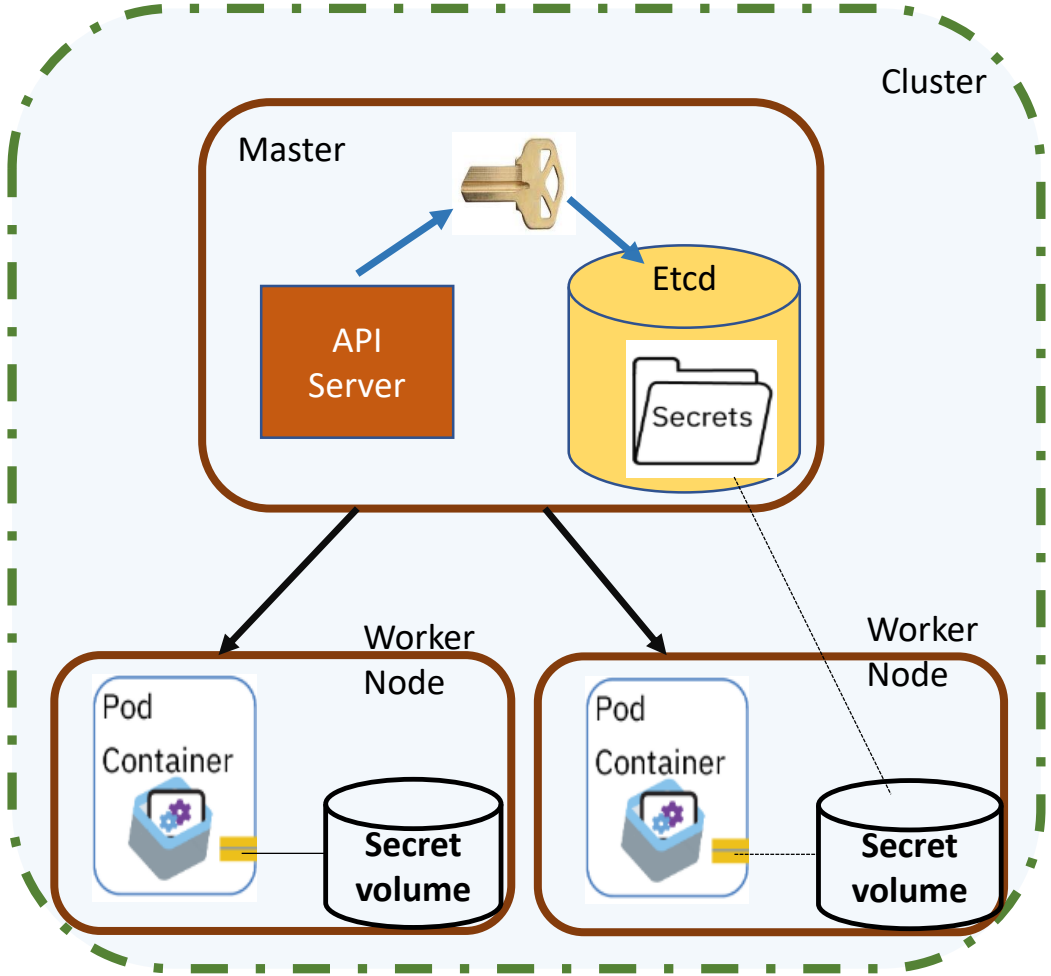
Intel Corporation

Agenda

- ✓ K8s Secrets encryption - Overview
- ✓ TEE-based KMS plugin - our proposal
- ✓ Demo
- ✓ Summary & Next Steps

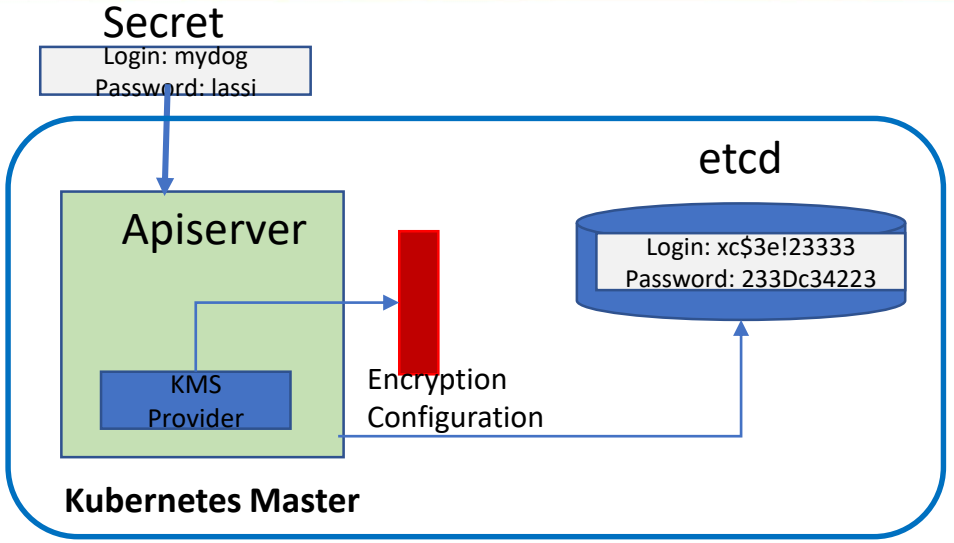
K8s Secrets - basics

- ✓ K8s Secrets: credentials, configuration, API key, keys, etc.
  - used by the System/Containers at build time or runtime
- ✓ Secrets stored in etcd
  - etcd = distributed Key-Value data store
- ✓ Default K8s setup: etcd contents not encrypted.
  - Secrets are stored in plaintext (base64 encoded)
- ✓ K8s 1.7+ introduced at-rest encryption for etcd.
  - API Server supports multiple Encryption Providers. (local and remote)



K8s Secret Encryption: Local Encryption Provider

- ✓ Encryption Keys stored on API Server
  - AESCBC/AESGCM providers. Key(s) in YAML EncryptionConfig file on API Server.
- ✓ Secrets encrypted prior to storage in etcd. Decrypted in API Server prior to use.
- ✓ Threat Model:
  - Mitigates : Attacker accessing etcd database (etcd compromise).
  - Doesn't mitigate: Adversary accessing the API Server (host compromise)



EncryptionConfiguration.yaml

Kind:EncryptionConfiguration

apiversion:apiserver.config.K8s.io/v1

resources:

-secrets

providers:

-aescbc:

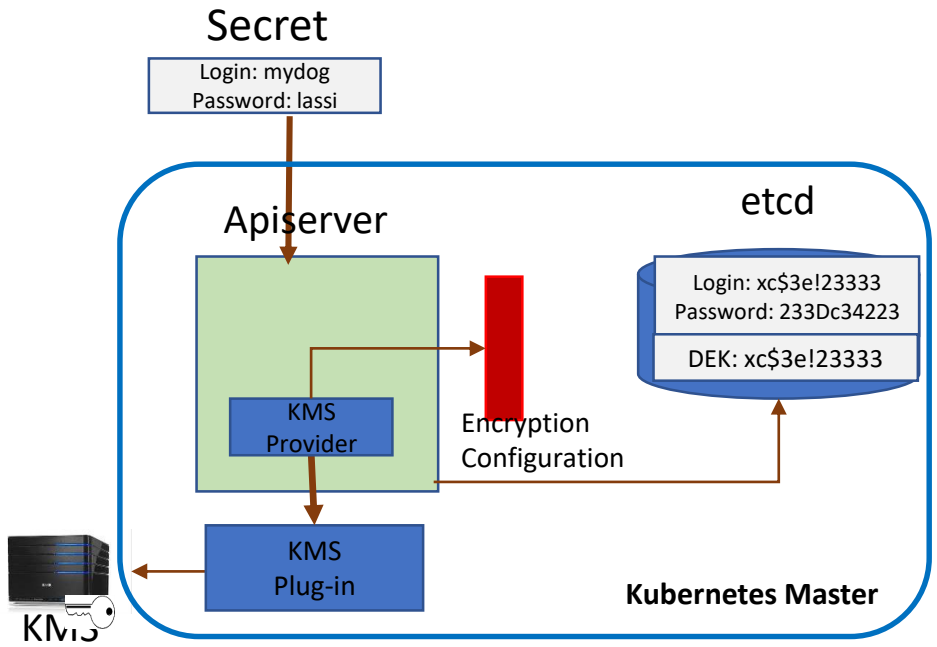
keys:

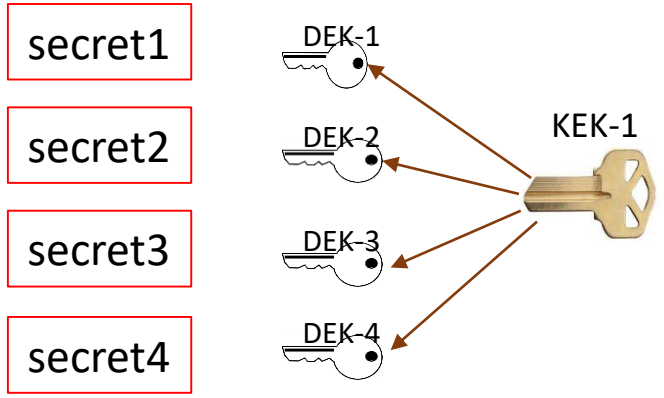
-name: key1

key : 9rihlvmie6+Ixv0cjcuak==

K8s Secrets Encryption: KMS Encryption Provider

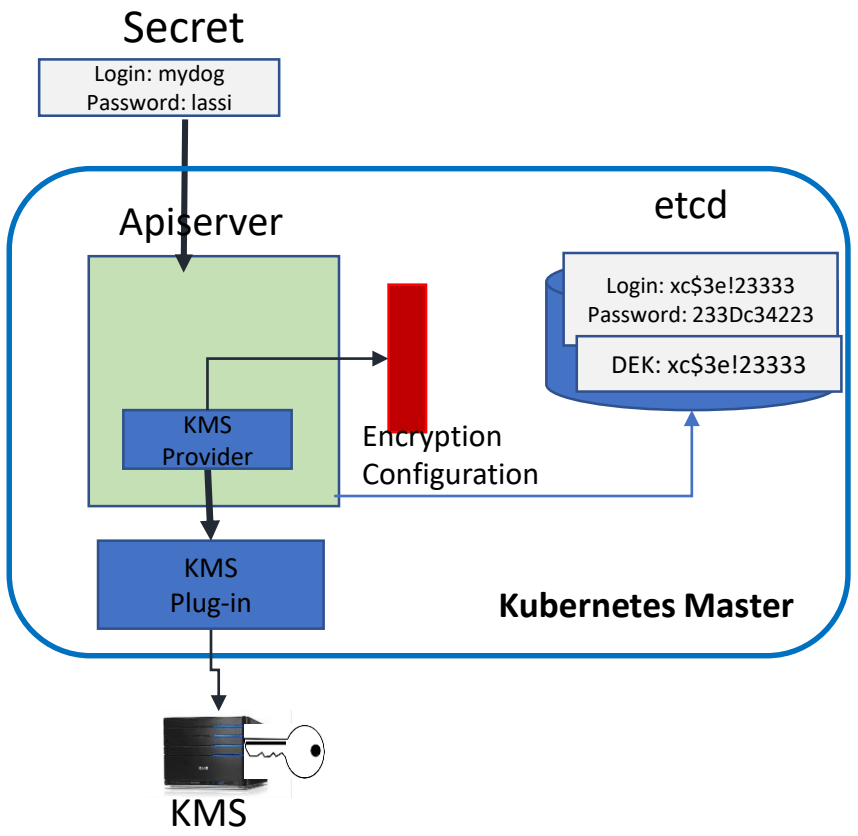
- ✓ Encryption keys not stored on the API Server – key (s) stored in a remote KMS.
- ✓ Uses envelope encryption scheme
  - Data/secret encrypted with a data encryption key (DEK)
  - New DEK is generated for each encryption
  - DEKs are wrapped with key encryption key (KEK)
  - Encrypted secrets and encrypted DEKs stored in etcd.
  - KEKs stored and managed in remote KMS
- ✓ Mitigates :
  - Attacker accessing etcd database (etcd compromise).
  - Access to API Server doesn't provide access to KEK. So, can't access DEKs and hence can't access secrets.



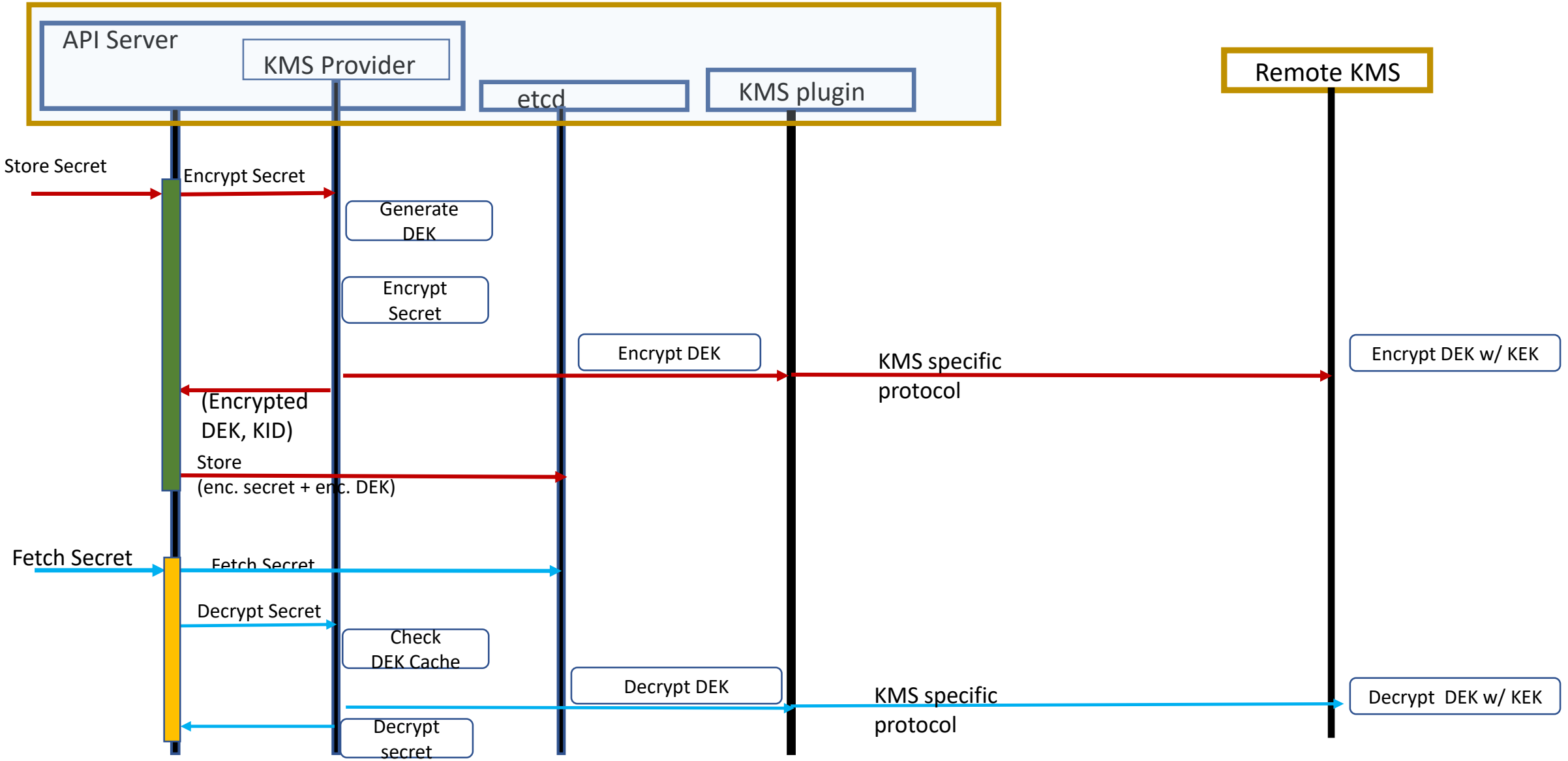


KMS Encryption Provider – how it works

- ✓ KMS Provider uses KMS plugin to interface with remote KMS.
- ✓ KMS plugin: gRPC Server running on the Master node.
- ✓ To save a Secret in etcd:
  - KMS Provider generates unique DEK using AESCBC.
  - KMS Provider encrypt secrets with the DEK locally.
  - KMS Plug-in sends DEK to remote KMS. DEK is wrapped with KEK at the remote KMS.
  - Wrapped DEKs and encrypted secrets stored in etcd database. Plaintext DEKs are not saved to disk or etcd.
- ✓ Process happens in reverse for reading Secrets.



KMS Encryption Provider – the flow.



KMS Provider - Key Observations

- ✓ API Server has to go to remote KMS for:
  - encryption of DEKs, prior to writing the secrets to etcd.
  - decryption of DEKs, while reading the secrets from etcd

  Performance and latency concern.

- ✓ KMS Provider supports caching of DEKs (configurable)… but..
  - with a cache: DEKs are in the clear in the API Server memory.

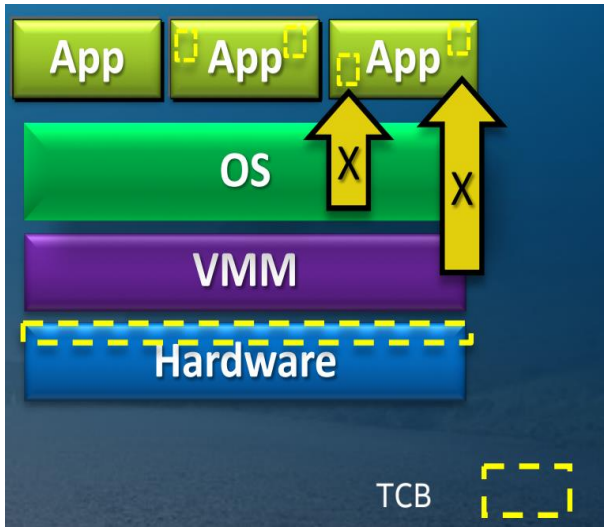
- ✓ DEKs are in the clear in API Server memory
  - Compromised API server/host, can compromise access to DEKs -> access to secrets in etcd (offline)

Our Solution Proposal: TEE-based KMS Plugin

- **Two objectives:**
  - Address Performance/Latency concerns – reduce/minimize remote KMS interactions with out compromising security.
  - Address the following threats:
    - etcd compromise
    - Attacker accessing DEKs in memory of API Server (Host compromise)

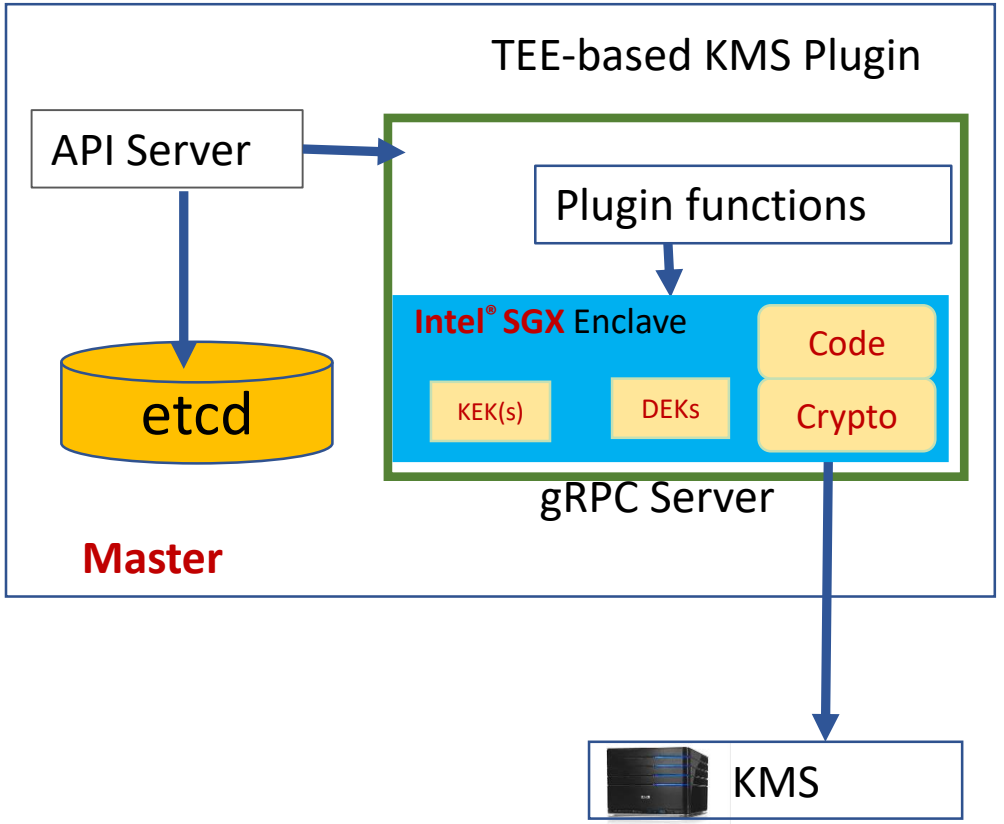
What is a TEE?

- ✓ A **Trusted Execution Environment** (**TEE**) is a secure area protected by the processor. (aka. Enclave)
- ✓ Provides hardware-enforcement so that:
  - Code loaded inside TEE is operator-authorized code.
  - Data inside TEE cannot be read/modified from outside the TEE.
- ✓ Guarantees code and data confidentiality and integrity.
- ✓ Threats protected:
  - Malicious/compromised admin
  - Malicious/compromised tenant of a hypervisor
  - Malicious/compromised network
  - Compromised operating system/BIOS
- One Example of TEE: Intel® SGX.



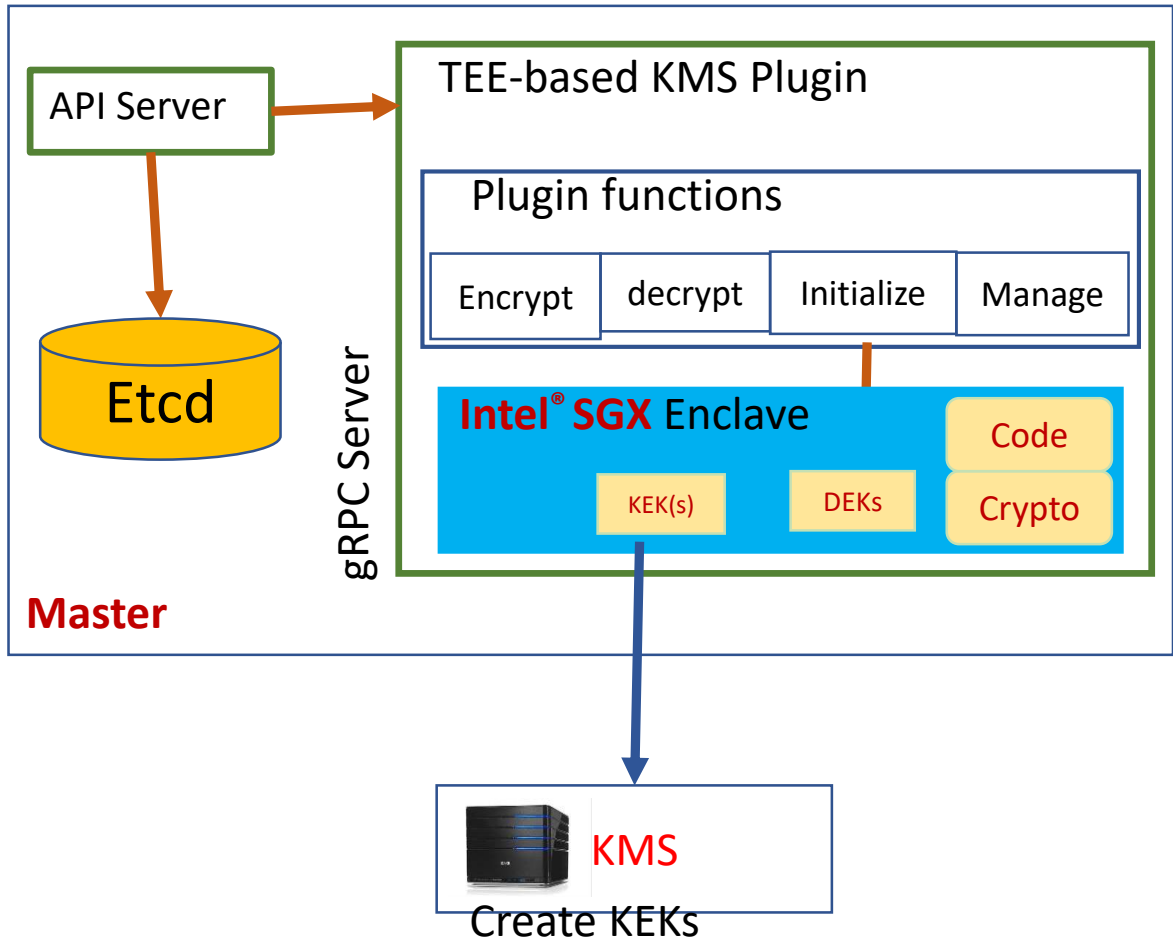
TEE-based KMS Plugin - Concept

- ✓ gRPC Server with an embedded **Intel® SGX** Enclave.
- ✓ With Enclave Attestation, cache the KEK(s) from KMS into the Enclave.
- ✓ Encryption/decryption Secrets in the Enclave
  - Create new DEKs in Enclave.
  - Cache DEKs in Enclave. So, never in the clear.
- ✓ Encrypt/decrypt DEKs in Enclave
  - Minimize going to remote KMS.
- ✓ Encrypted secrets & encrypted DEKs written to etcd by API Server.
- ✓ Decrypted Secrets volume mounted (tmpfs) or environment variables for Pods.



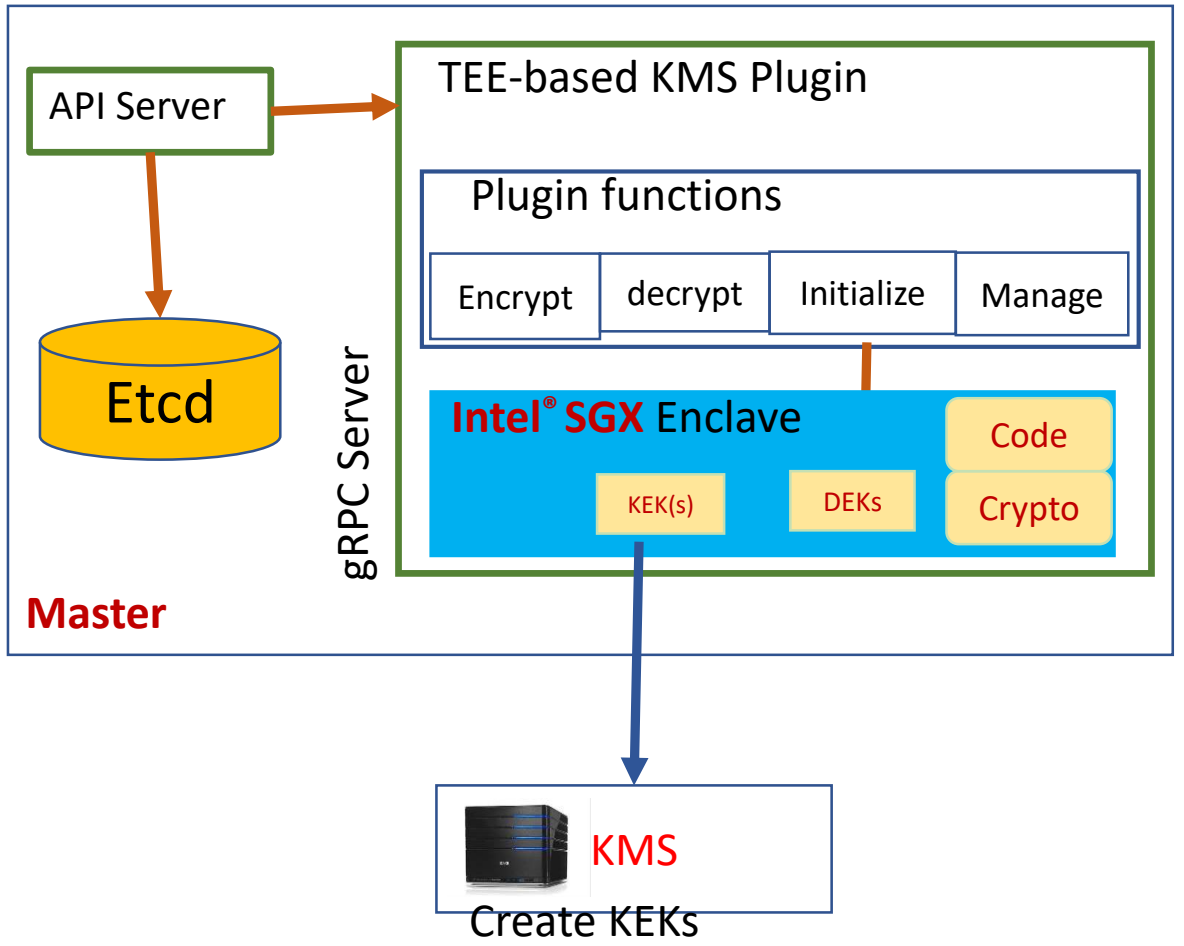
TEE-based KMS Plugin – Details (1)

- Initialize:
  - Create TEE (Intel SGX), Cache KEK (s) in the TEE, when KMS plugin starts (or, on demand).
    - Cache KEK after Attestation of TEE by remote KMS.
- Encrypt Secret:
    - Secret sent to TEE.
    - Generate new key (DEK) and encrypt Secret.
    - Encrypt DEK with the KEK.
    - Return encrypted secret and wrapped DEK to API Server.
    - API Server stores encrypted secret+enc DEK in etcd.



TEE-based KMS Plugin – Details (2)

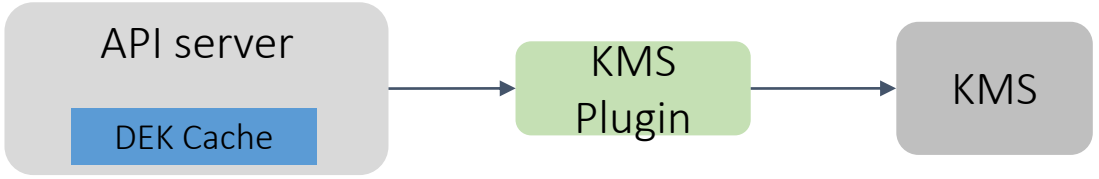
- Decrypt Secret:
  - Decrypt secret request sent from API Server to Plugin functions.
  - Plugin function separates Cipher Secret and Cipher DEK
  - Decrypt DEK if not in Enclave Cache (with KEK or go to remote KMS).
  - Decrypt secret in Enclave.
  - Plugin returns secret to API Server.



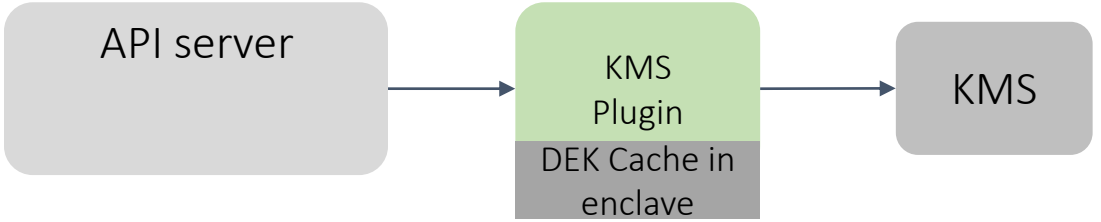

DEMO

Demo Scenarios

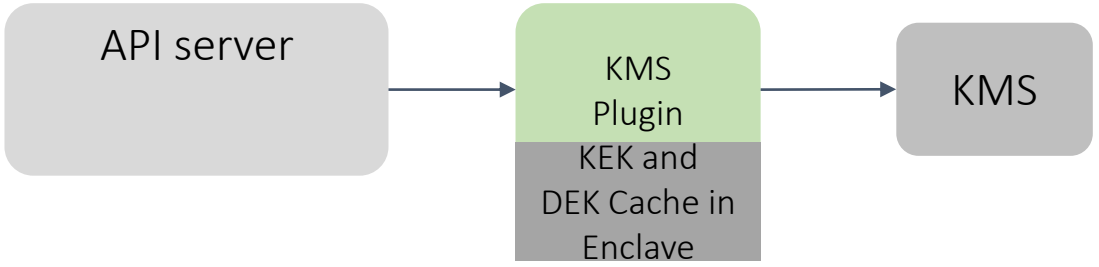
Default Implementation:



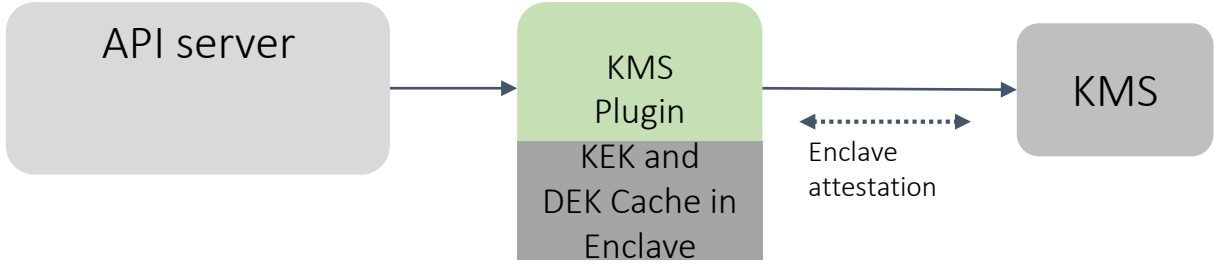
Demo 1: DEK cached in SGX Enclave of the KMS plugin



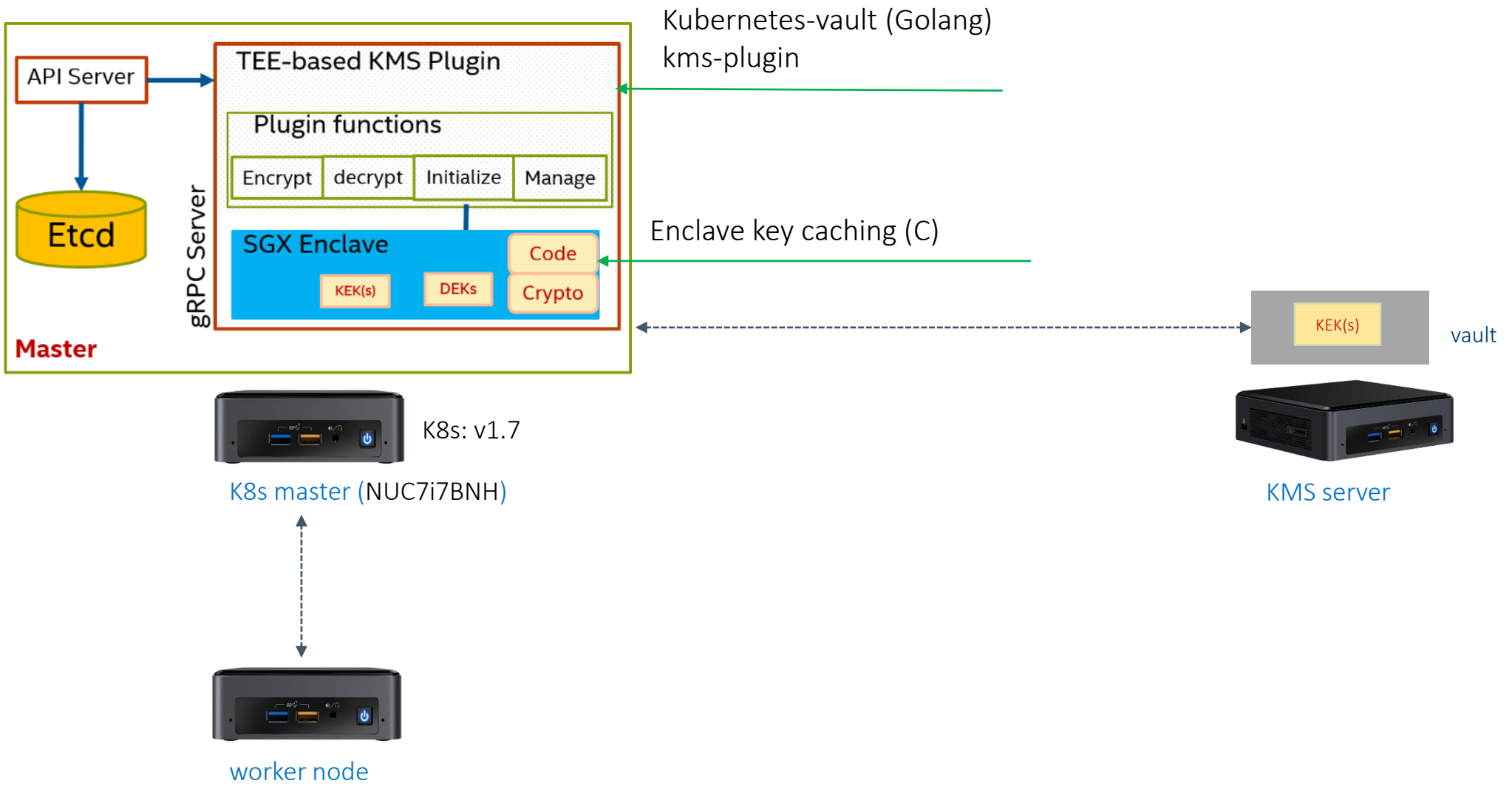
Demo 2: DEK and KEK (manual) cached in Enclave in KMS plugin



Demo 3: Attestation of Enclave to provision KEK from KMS to KMS plugin, Encrypt/Decrypt of secrets in the Enclave



Demo environment



Next Steps

- Complete the demo/POC to show full functionality of the TEE-based KMS plugin.
  - Enclave attestation, caching of KEKs and encrypt/decrypt of secrets in TEE
- KEP for API Server changes for TEE-based KMS Provider/plugin
- SGX-based KMS Plugin reference implementation for the approved KEP

Backup