Certificate and Secret Management Service (CSM)

Project Name:

- Proposed name for the project: `Certificate and Secret Management Service`
- `Proposed name for the repository : csm`

Meetings

- [csm] Team ONAP7, Thr. UTC 04:00 / Thr. China 12:00 / Wed Eastern 23:00 / Wed Pacific 20:00
- Slides presented at Beijing Developer Forum, Santa Clara.

Project description:

This project proposal address two areas in the ONAP deployment structure from a security perspective.

1. Secure Communication between microservices.
    - Current state and need
        - ONAP consists of multiple micro services which talk to each other.
          There are two types of communication.

            a. REST API based communication.
            b. DMAAP publish/subscriber based communication.

          Since the communication is mostly over HTTP, there is a need to protect services from:

            - Bad actors stealing the data on the wire.
            - Receiving messages from bad actors
    - Requirement:
        - Enable TLS1.2+ for securing communication among the services. Java and Python libraries do support this functionality, but easy certificate provisioning is required for Mutual TLS. This project aims to simplify PKI - certificate provisioning via a simple and secure CA service that stores private keys (CA private key at CA and user certificate private keys) securely using hardware security.

2. Storage of sensitive information such as passwords.
    - Current state and gaps
        - Many services in ONAP use password based authentication. Eg: Database servers, publish/subscribe brokers etc.
        - Passwords are stored in plain text files in many services.
        - With multiple instances of these services, the attach surface area becomes very big.
        - Hence there is a need to ensure that attack surface related to password exposure is reduced.
    - Requirement:
        - Need for secure secret management. Services are expected to get the secret only on needed basis using secret reference and remove the secrets once they are used up.

This project aims to provide solutions to the above needs by:

1. Provide Certificate Management Service (CA Service) to provision signed certificates required for Mutual TLS.
2. Provide Certificate Request Agent SDK
3. Provide hardware security plugin for storing private keys and for performing crypto operations that require private keys.
4. GUI/CLI for Certificate Management Service
5. Provide Secret Management Service for adding/deleting/updating/reading secrets.
6. Provide Secret Client Agent SDK
7. GUI/CLI for Secret Management Service.

Scope:

Certificate Management Service

The proposed project will provide a Certificate Management Service which will be used for certificate enrollment by micro services. The ultimate goal is to make sure that all micro services communicate securely between each other using the CA for enrollment and then use TLS to establish secure communication channels between each other.

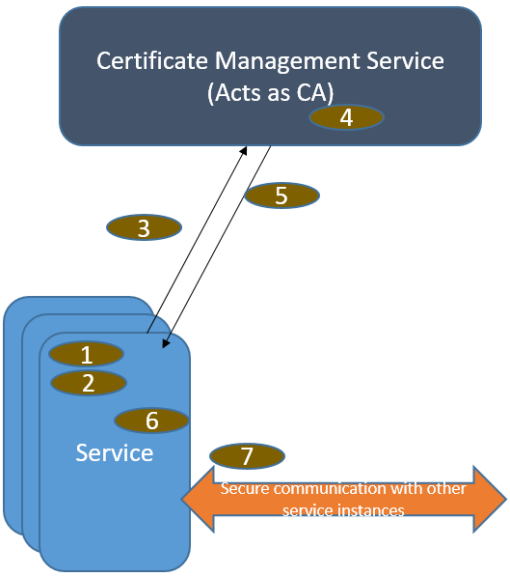
The Certificate Management Service will support the following:

- RESTful API support for Certificate Request Operations by micro services
    - Generate Certificate
    - Revocation of Certificate
    - Usage report updates
    - Token Authentication
- An Admin interface
    - That will generate a self signed CA
    - Upload any admin generated CA Cert + Private Key pair
    - Usage usage reports on each key
    - Revoke certificates
    - Get CA Certificate in PEM/DER format
    - Token service to provide temporary tokens

There will also be a Client that will be part of the project written in either Python or Java that will be used to communicate with the CA Broker Service to enroll certificates.
It will have the following roles/abilities:

- Generate RSA/ECDSA key pair using PKCS11
- Securely store the private key.
    - Store the private key using TPM if it is available
- PKCS10 CSR generation
- Communicates with the previously described Certificate Management Service over REST API
- Periodically generates a usage report
- Certificate Renewal
- Discovery of Certificate Management Service

The below diagram illustrates Best Practices of Certificate Enrollment that is end-point initiated.

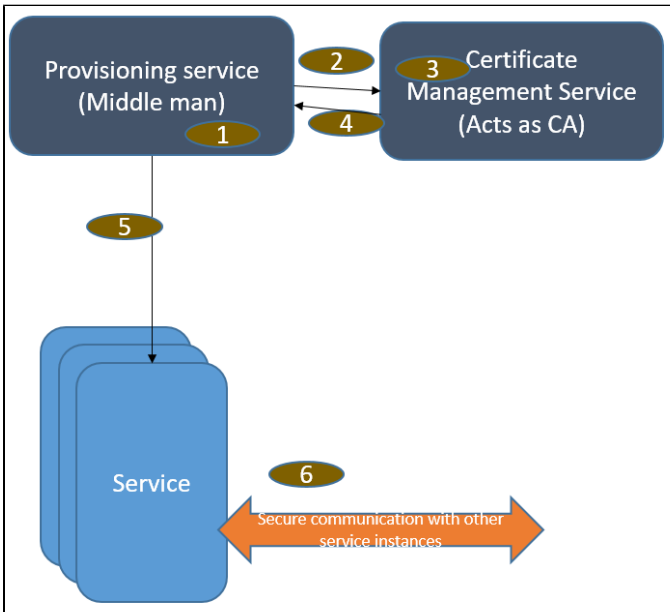


In Micro Services, before it communicates with other micro services, it needs to get certificate enrolled by CA.

1. Typically at startup, generates RSA/ECDSA public/private key pair.
2. Generates PKCS10 CSR (Certificate request) – Which involves signing the message with private key.
3. Request Certificate by sending PKCS10 request to CA.
4. CA verifies that genuine service is requesting for certificate, verifies PKCS10 request, generates X.509v3 certificate, signs it using CA certificate-private key.
5. Sends signed X.509v3 certificate and CA certificate.
6. Service stores the information.
7. It uses this information during TLS handshake to establish secure communication channels.

- Good for Micro Services that are long lived.
- Possibility of keeping the private key secure within container service.
- Good for any workloads (Local/remote)
- Slow startup – Need for certificate auto provisioning.

The below diagram illustrates Certificate Enrollment that is Middle Man initiated



As part of service instantiation, provisioning service provisions certificate and private key (either via environment variables, volume backend, cloud-init)

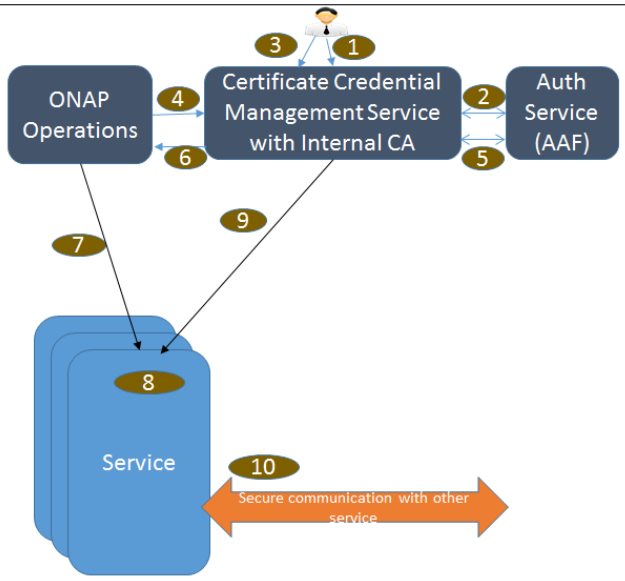
Provisioning Service is deployed with token/authentication credentials with permissions that allow CA to issue certificates.

For Every Service instance

1. Generates RSA/ECDSA key pair, generates subject name, PKCS10 CSR
2. Request signed certificate by sending CSR to CMS
3. CMS verifies the token/auth-credentials, generates x.509v3 certificate and signs it using CA private key
4. CMS sends the x.509v3 certificate to Provisioning service
5. Provisioning service sends certificate + private key to service.
6. Service uses these credentials to create Mutual TLS session with others

- Good for Micro services that are short lived and where startup performance is very important.
- Disadvantage : Keys are clear

This diagram shows mapping of Certificate Provisioing in ONAP context.



It is expected that Certificate management and Auth services are brought up first.

Creating CA Instance (Typically once per deployment)

1. Admin user creates authentication session by passing username/password OR grant token given by Auth Service.
2. CMS validates the Admin user credentials using Auth Service
3. Admin user instructs CMS to create self signed CA.

ONAP (Rest of) Service instance bring up (Getting the CMS-token)

4. Operations Admin provides username/password or Auth grant token to CMS to get the CMS-token.
5. CMS validates the credentials with Auth Service.
6. CMS returns the CMS-token.
7. CMS-token is passed to Service as part of its bring up

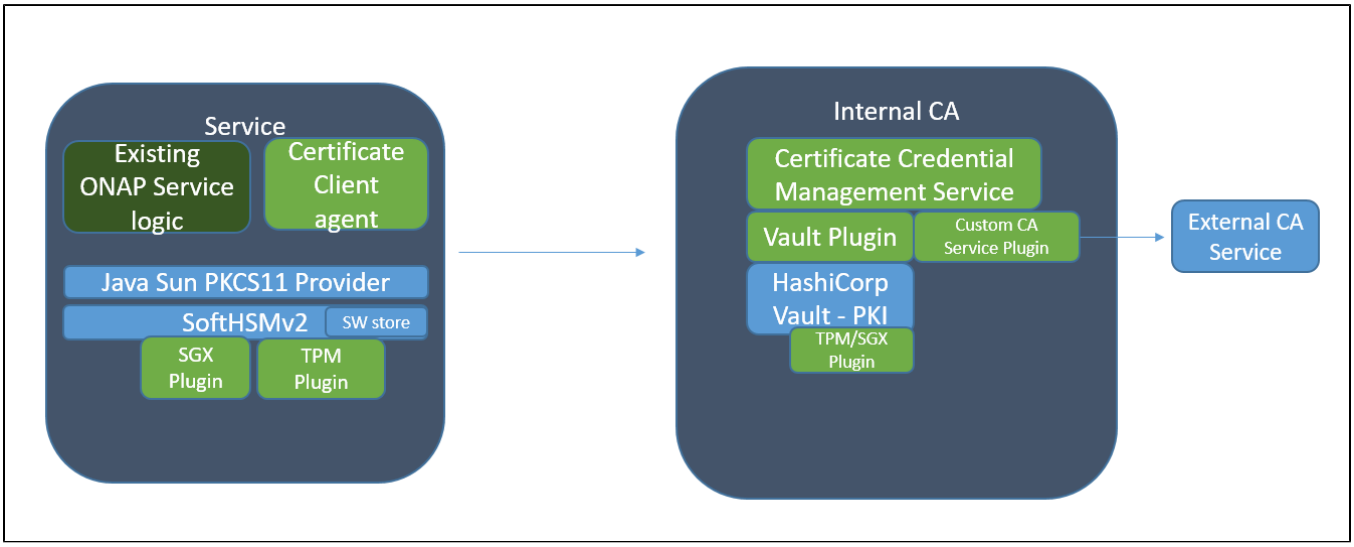
ONAP Service Certificate provisioning (Service startup time)

8. As part of bring up, generates RSA/ECDSA key pair and PKCS10 (CSR) request.
9. Sends PKCS10 CSR with CMS-token to CMS. CMS validates the token. If valid generates certificate and signs it with its CA. Returns certificate and CA certificate.

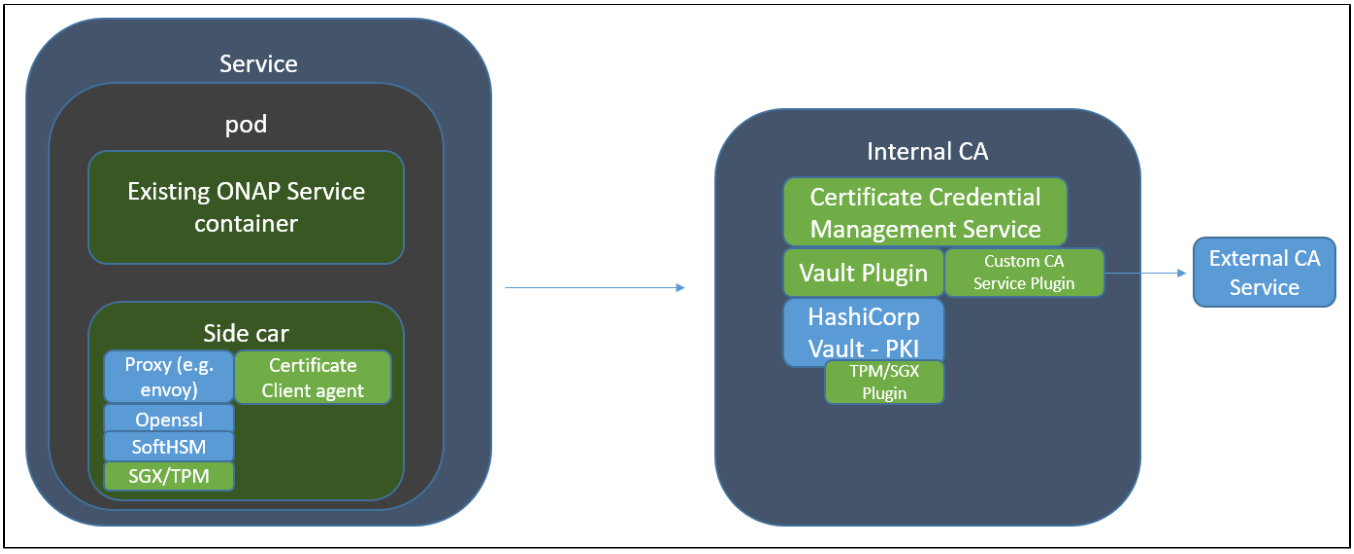
ONAP TLS Communication

10. ONAP service uses certificate and private key during TLS handshake

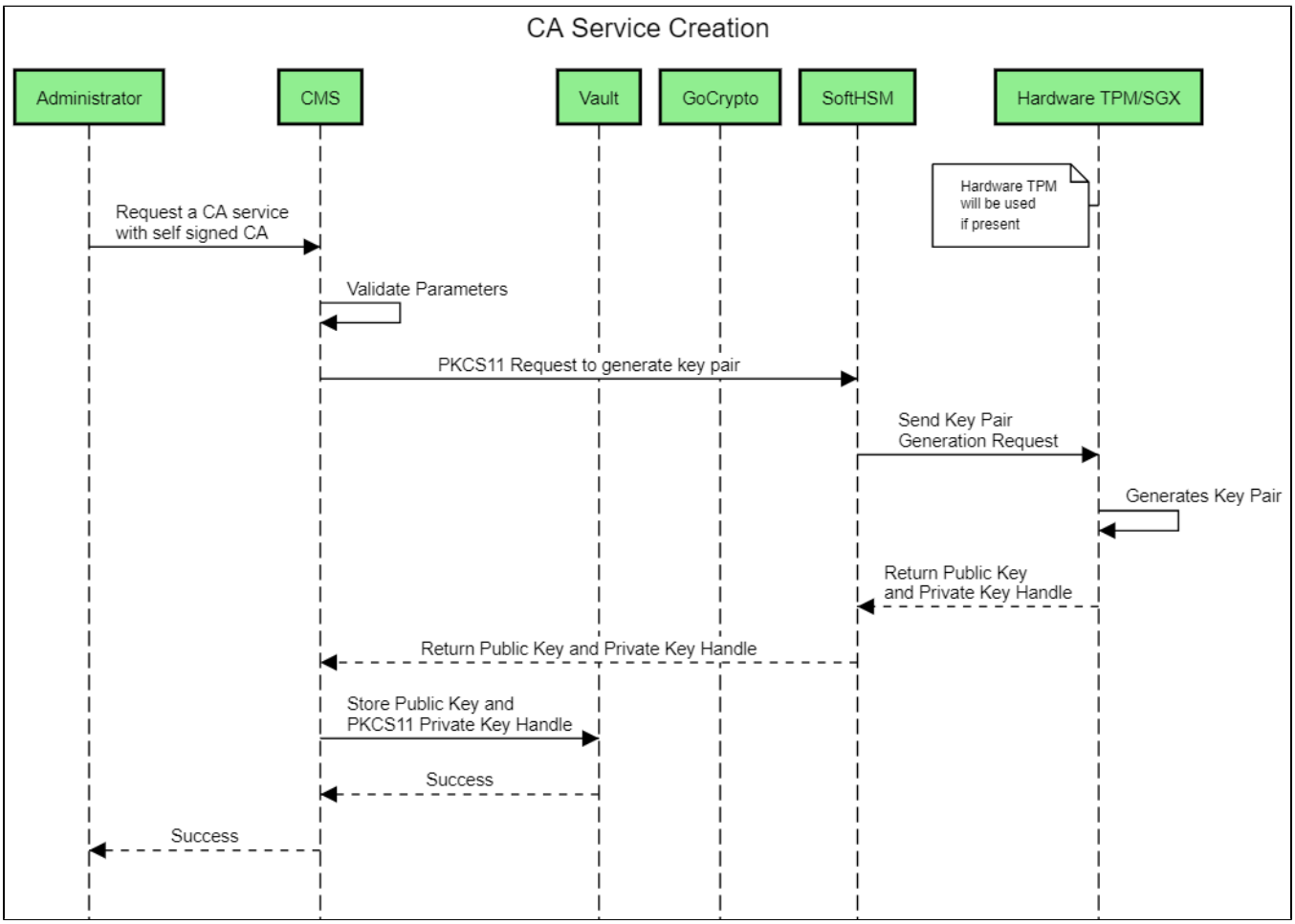
The below diagram details the architechture blocks used previously in detail:

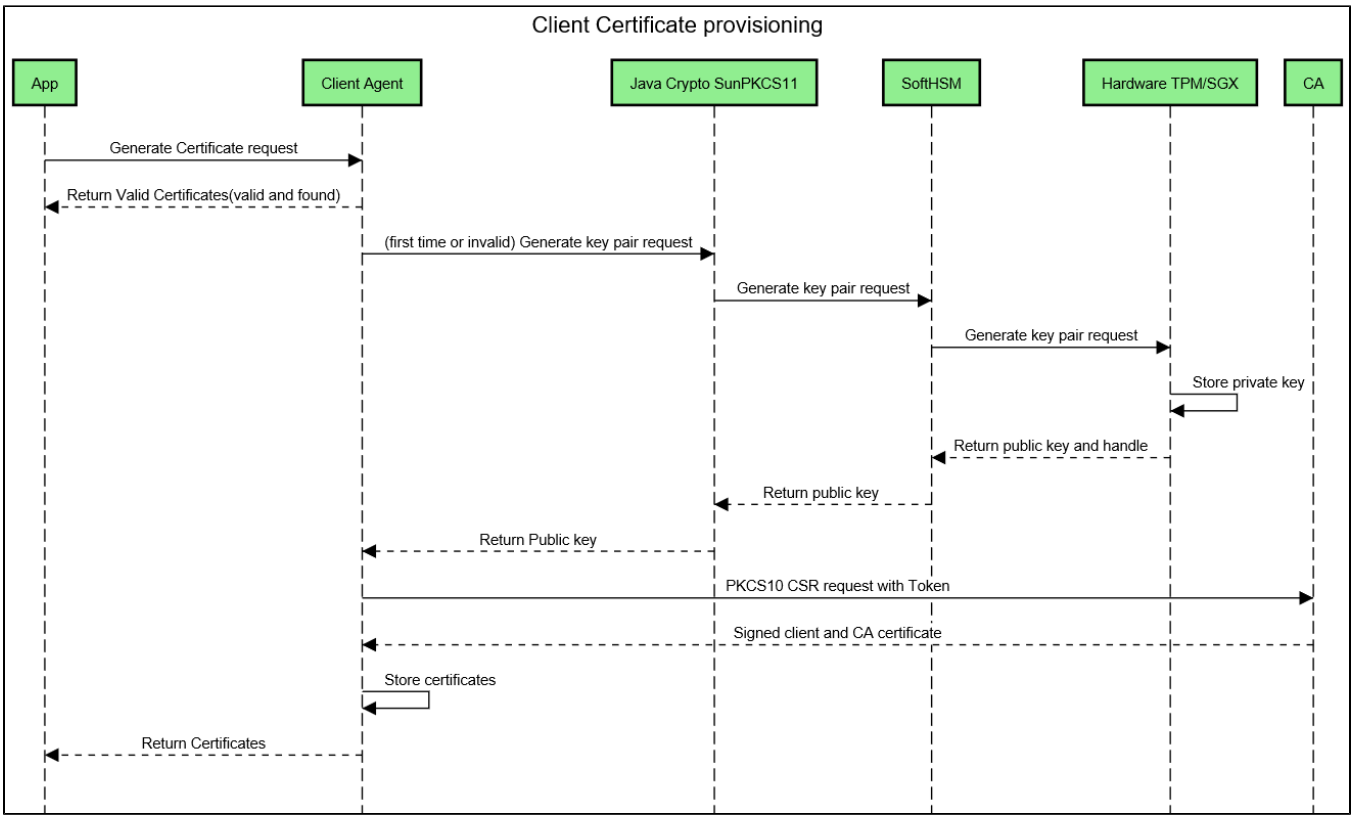


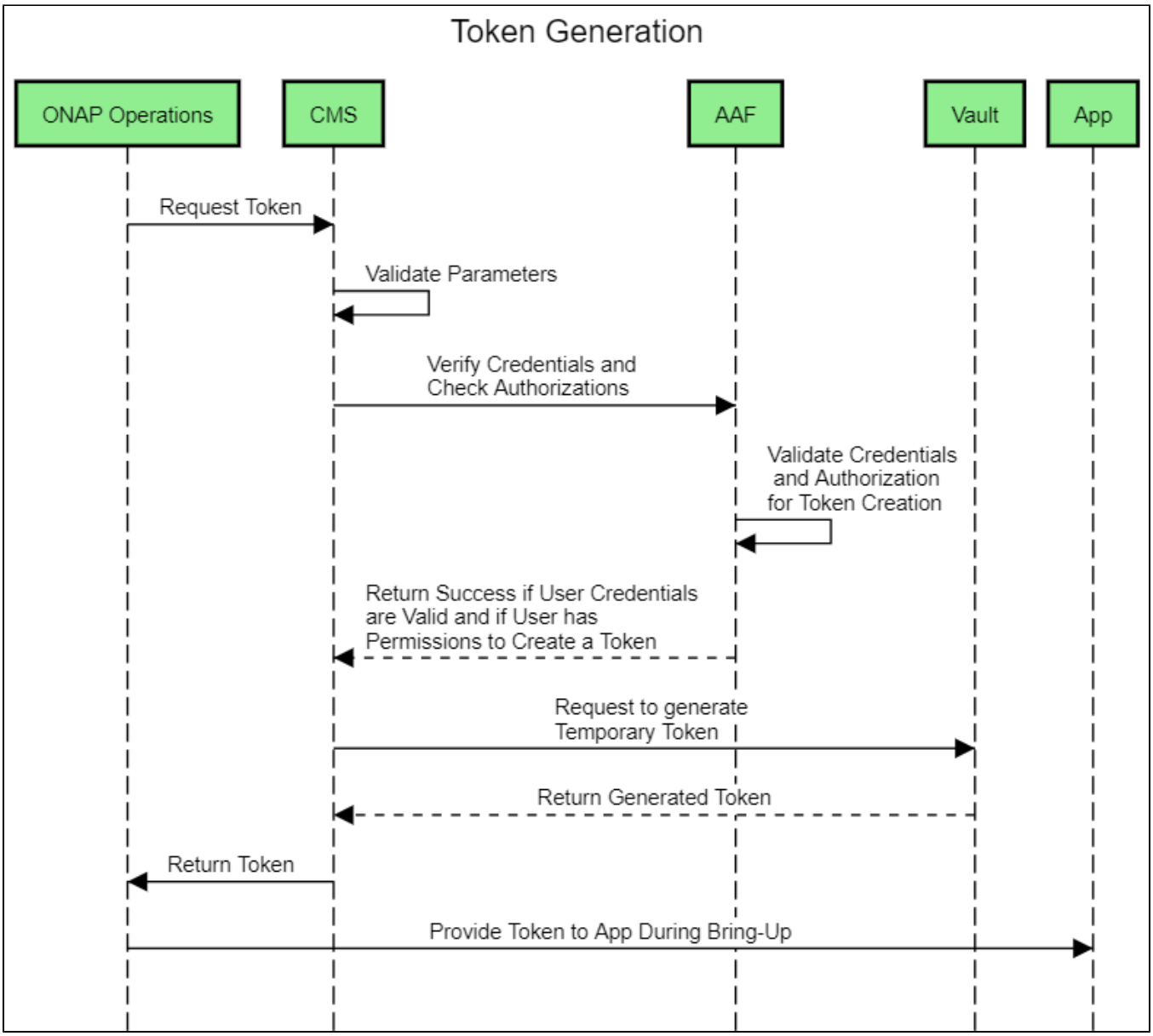
The below diagram the same architecture blocks as above with a Sidecar service:

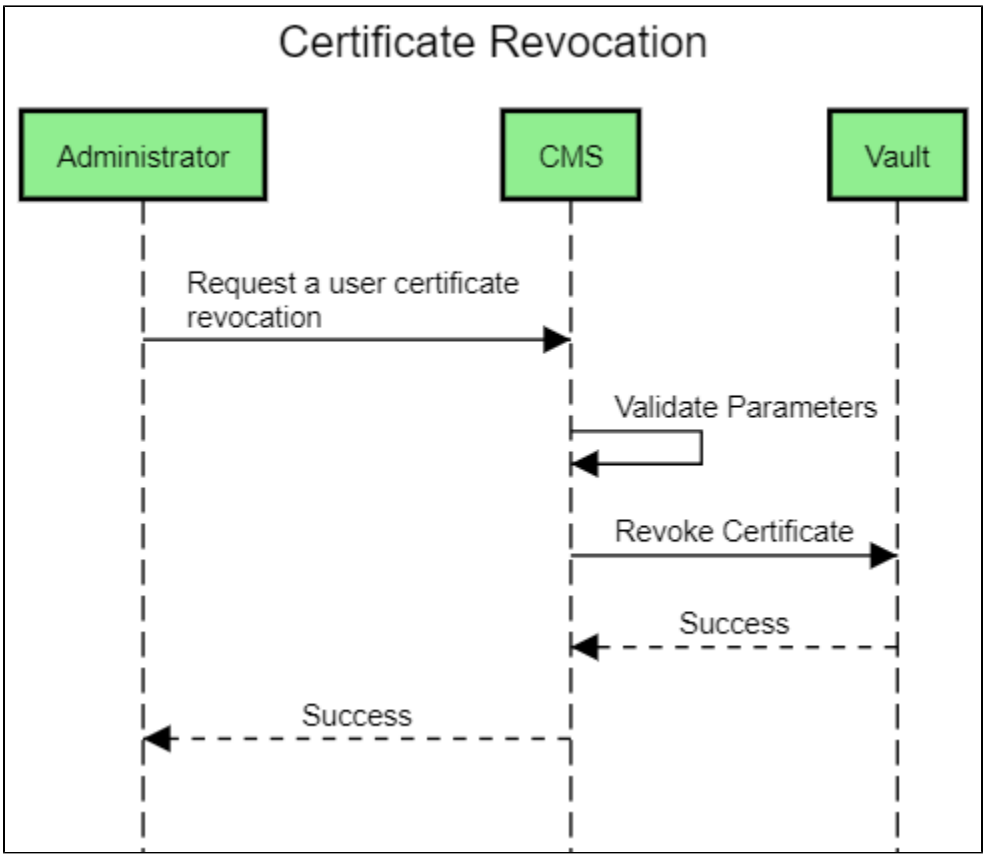


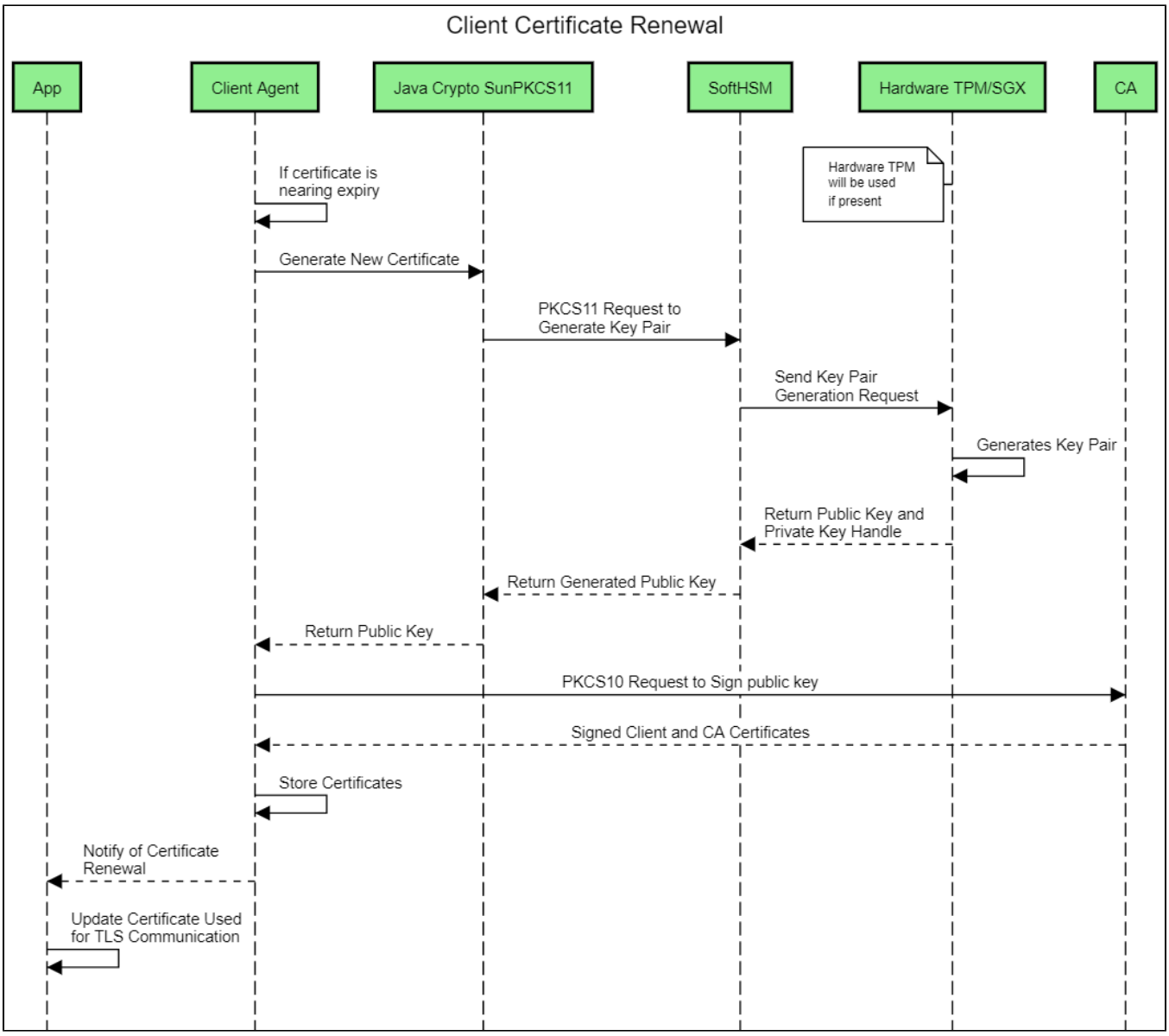
Use Case Sequence Diagrams




Client Certificate provisioning
App
Client Agent
Java Crypto SunPKCS11
SoftHSM
Hardware TPM/SGX
CA
Generate Certificate request
Return Valid Certificates(valid and found)
(first time or invalid) Generate key pair request
Generate key pair request
Generate key pair request
Store private key
Return public key and handle
Return public key
Return Public key
PKCS10 CSR request with Token
Signed client and CA certificate
Store certificates
Return Certificates


Token Generation
ONAP Operations
CMS
AAF
Vault
App
Request Token
Validate Parameters
Verify Credentials and
Check Authorizations
Validate Credentials
and Authorization
for Token Creation
Return Success if User Credentials
are Valid and if User has
Permissions to Create a Token
Request to generate
Temporary Token
Return Generated Token
Return Token
Provide Token to App During Bring-Up


Certificate Revocation
Administrator
CMS
Vault
Request a user certificate revocation
Validate Parameters
Revoke Certificate
Success
Success

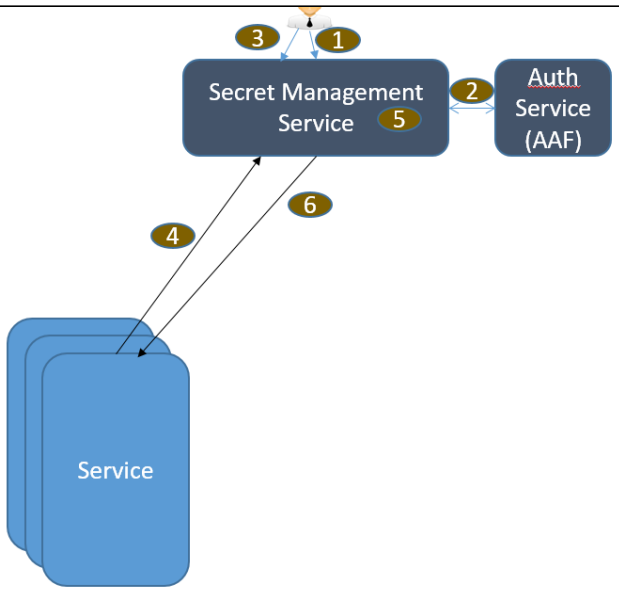


Secret Management Service

The project will also provide a Secret Management Service with the following features and capabilities:

- Support multiple Secret domains
  - Each domain can be used to multiple secrets
  - Each domain is associated with various policies
- Each secret can have multiple key value pairs
- Certificate based authentication
- Authenticate users with AAF
- Token based authentication
- Securely store secrets using AES encryption
  - Use TPM/SGX for key storage if available
- RESTful API support for ADD, UPDATE, DELETE of secrets

The below diagram illustraces the Secret Service High Level Flow in an ONAP Context



It is expected that Certificate management and Auth services are brought up first.

Creating Secret Domain

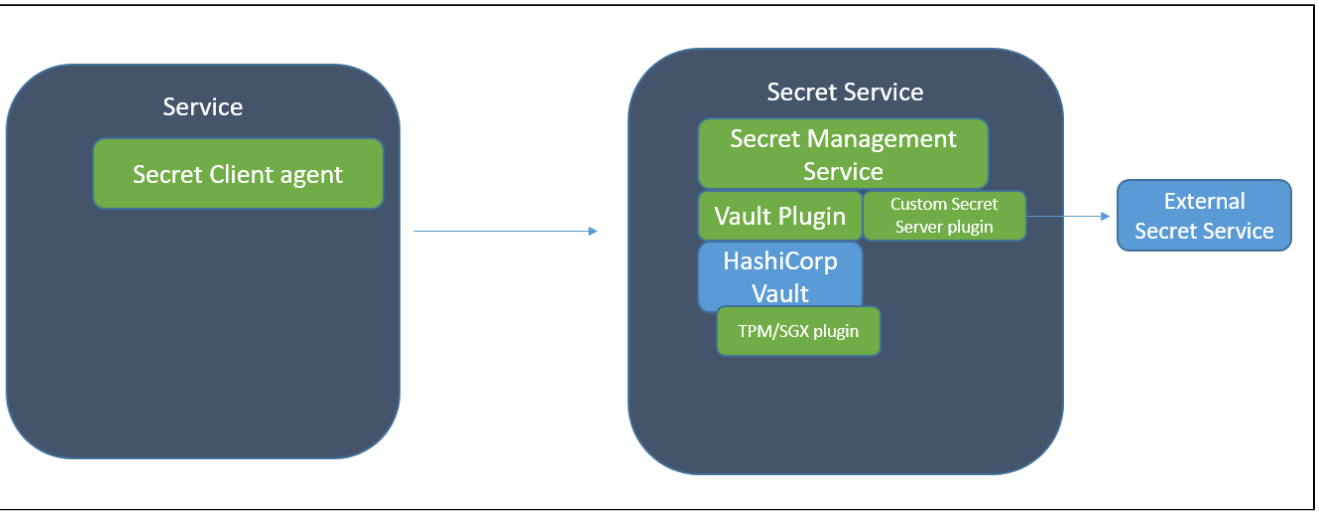
1. Admin user creates authentication session by passing username/password OR grant token given by Auth Service.
2. SMS validates the Admin user credentials using Auth Service
3. Admin user instructs SMS to create secret domain (Policies, CA-Cert, Set of Subject name prefix to allow vs permissions)

Service instance bring up (Assumes that service already got the certificate provisioned)

4. Service makes Secret service request via TLS. (Create Secret/Read Secret)
5. SMS validates the certificate credentials, if valid and if policy allows it, perform the operation. Also, create and return SMS-token for future operations.
6. SMS returns the results

Service uses secrets for service specific processing

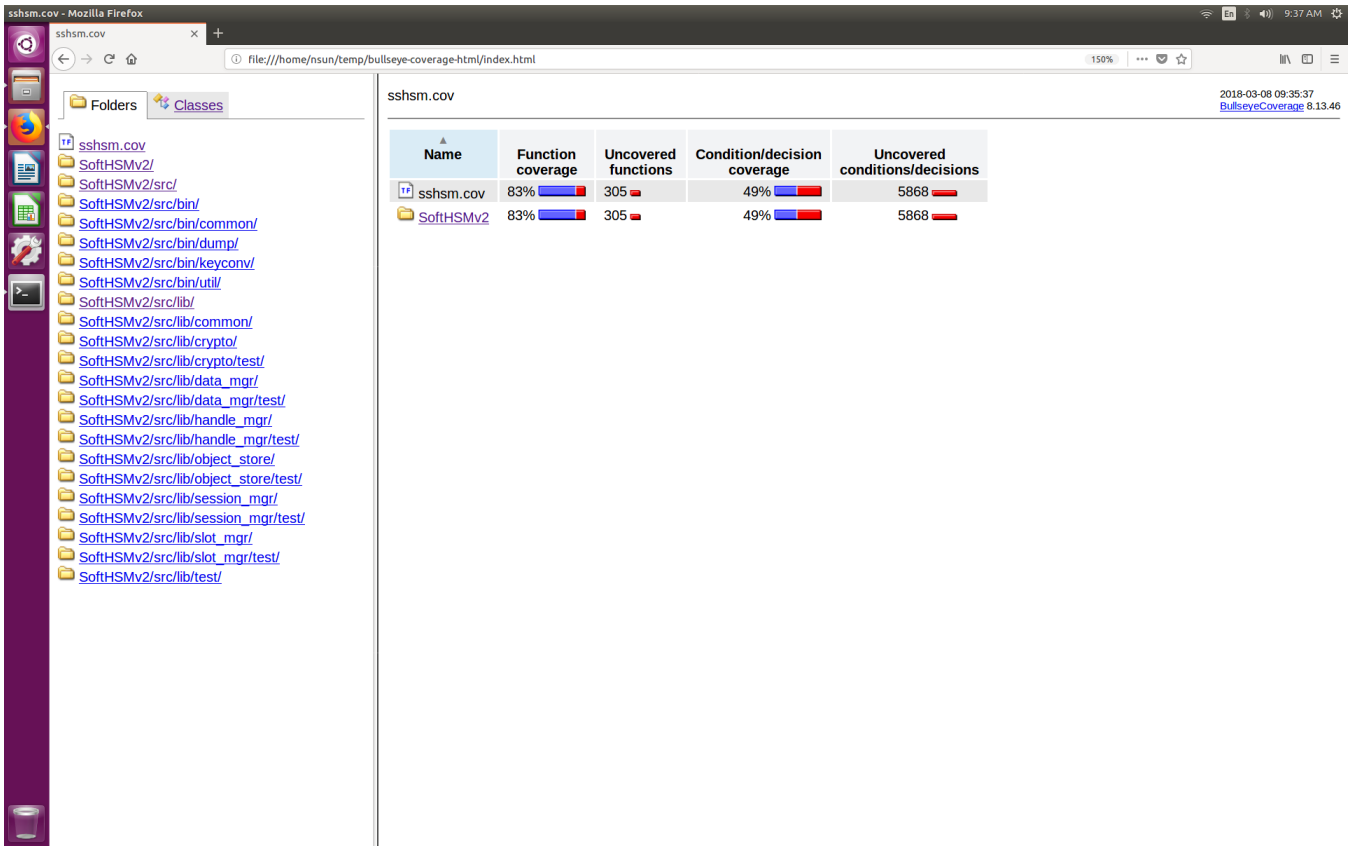
The below diagram illustrates how a micro service will use the Secret Client Agent to talk to the Secret Service to store or retrieve passwords.



SoftHSMv2 +TPM2-Plugin

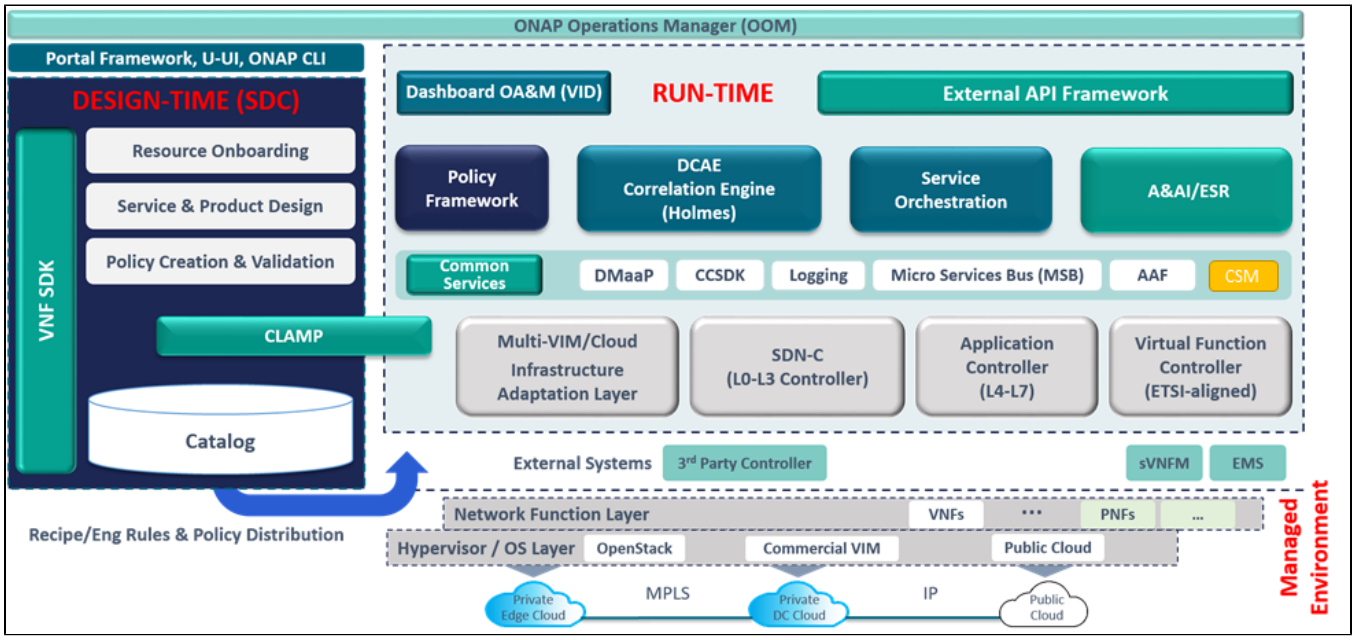
This project provides SoftHSMv2 with an extended capabilities to leverage TPM2.0 hardware capabilities to generate RSA/ECC keypairs and import keys generated outside of TPM2.0 module. This is achieved by modifying SoftHSMv2, adding an adapter layer between SofhHSMv2 and TPM2-Plugin.

Bullseys coverage tool is used to measure the codes coverage:



Architecture Alignment:

CSM is a common service across ONAP components.



Other Information:

- Seed code:
  - Vault project: https://github.com/hashicorp/vault

Key Project Facts:

Primary Contact : Srinivasa Addepalli

Facts	Info
PTL (first and last name)	
Jira Project Name	TBD
Jira Key	TBD
Project ID	TBD
Link to Wiki Space	TBD

Release Components Name:

Note: refer to existing project for details on how to fill out this table

Components Name	Components Repository name	Maven Group ID	Components Description
sms	aaf/sms	org.onap.aaf.sms	Secret Management Service that will contain the webservice as well as client code for managing and accessing secrets.
sshsm	aaf/sshsm	org.onap.aaf.sshsm	A repository for softhsm modifications and hardware security plugin

Resources committed to the Release:

Note 1: No more than 5 committers per project. Balance the committers list and avoid members representing only one company.

Note 2: It is critical to complete all the information requested, that we help to fast forward the onboarding process.

Role	First Name Last Name	Linux Foundation ID	Email Address	Location
PTL	Kiran Kamineni	kirankamineni	kiran.k.kamineni@intel.com	Santa Clara, CA
Committers	Kiran Kamineni	kirankamineni	kiran.k.kamineni@intel.com	Santa Clara, CA
	Girish Havaldar	giri	hg0071052@techmahindra.com	Bangalore, India
Contributors	Vamshi Namilikonda	vamshi.nemalikonda	vn00480215@techmahindra.com	Pune, India
	Manjunath Ranganathaiah	mrangana	manjunath.ranganathaiah@intel.com	Santa Clara, CA, USA
	Ning Sun	ningsun	ning.sun@intel.com	Santa Clara, CA, USA