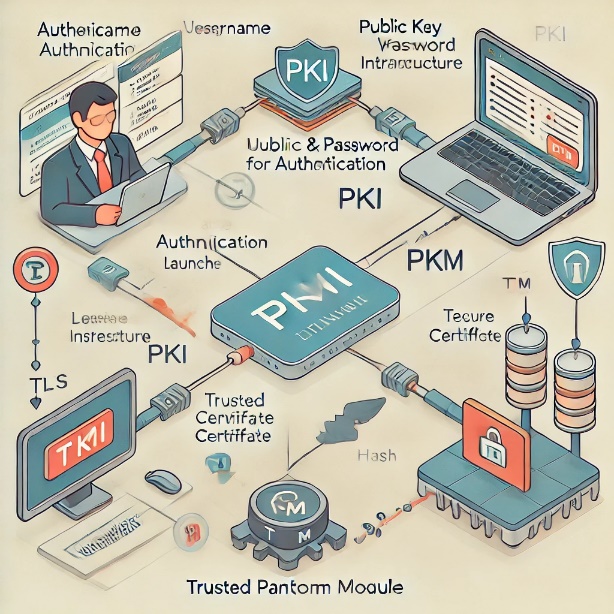
**Part 1**

An overview of the proposed architecture and steps to design and implement it

**Architecture Overview**

* **Licensing Server**: Manages user authentication, validates credentials, and launches enclaves on demand. It uses a PKI-based system to verify the integrity and authenticity of users and enclaves.
* **Public Key Infrastructure (PKI)**: Provides the cryptographic mechanisms to authenticate users and verify enclaves via certificates and asymmetric encryption.
* **Enclaves**: Secure computational environments where users can run sensitive operations. Each enclave has a unique TLS certificate.
* **Kubernetes Cluster**: A container orchestration platform for deploying and managing the licensing server, enclaves, and related services.

**Authentication Workflow**



**User Authentication with Credentials (Username/Password)**:

*Users authenticate using their credentials (username and password) to the licensing server. This can be handled via an HTTPS request, secured using TLS.*

*Credentials are verified against an internal/external identity management system (such as Active Directory or OAuth).*

*Upon successful verification, the server initiates the next step.*

**PKI Authentication**:

*Once the user is authenticated via username and password, the licensing server generates a user-specific session key, signs it, and returns a client certificate.*

*This certificate is used by the client to establish a secure PKI-based connection to the enclave.*

**Enclave Launch**:

*The licensing server launches a secure enclave instance on request. This can be automated via Kubernetes, where the server deploys the enclave as a containerized instance.*

*Each enclave instance has its own unique TLS certificate for secure communication. This certificate will be created as part of the enclave deployment.*

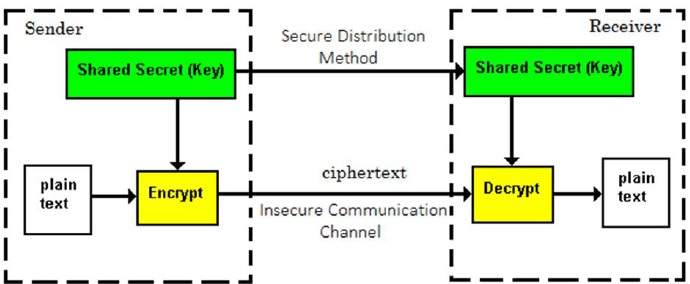
**TPM Quote Verification**:

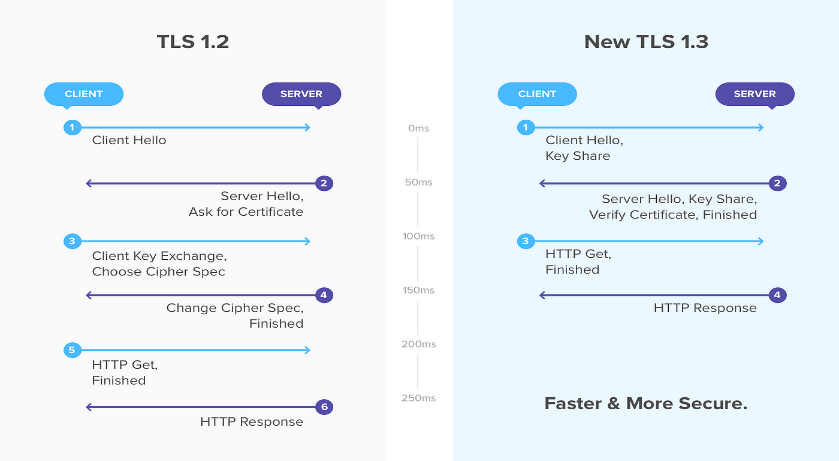
*When launching the enclave, the licensing server will retrieve the enclave’s TLS certificate and include its hash in a TPM quote.*

*The server returns the TPM quote and its public attestation key to the client.*

*The client verifies the quote and compares the enclave’s TLS certificate hash with the hash in the quote to ensure the enclave is genuine and not tampered with.*

**Key Cryptographic Algorithms and Mechanisms**





**Asymmetric Encryption (PKI)**:

*Used for generating key pairs, certificates, and securing communication between clients and the licensing server. RSA (2048-bit or higher) or ECC (e.g., secp256r1 curve) would be preferred for strong encryption.*

*Every user, the licensing server, and each enclave instance will be assigned an X.509 certificate signed by a trusted certificate authority (CA).*

**Hashing**:

***SHA-256/SHA-512****: Used to create the hash of the enclave’s TLS certificate for inclusion in the TPM quote. This ensures that the certificate is verifiable.*

***HMAC****: To verify the integrity of transmitted data (e.g., ensuring the TPM quote is not tampered with).*

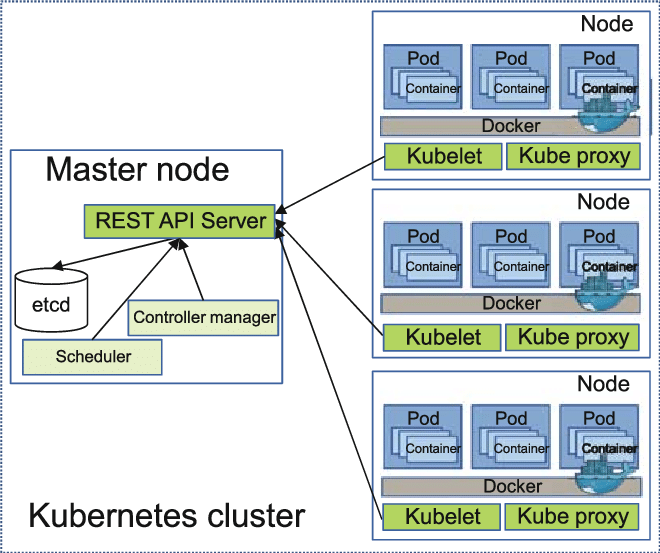
**TLS (Transport Layer Security)**:

***TLS 1.3****: Ensures secure communication between users, the licensing server, and the enclaves. It also ensures mutual authentication using client certificates.*

**TPM (Trusted Platform Module)**:

*Used to generate a quote containing a secure hash of the enclave’s TLS certificate. The TPM on the licensing server securely generates and signs the quote, providing cryptographic proof of enclave integrity.*

**Kubernetes Architecture for Enclave Deployment**



**Kubernetes Deployment**:

The licensing server will be deployed as a **microservice** within a Kubernetes cluster. This service will expose APIs for user authentication and enclave requests.

Each enclave instance is deployed as a **Kubernetes pod**, orchestrated by the licensing server.

**Helm charts** or **Kustomize** can be used to standardize enclave deployments, with each instance pulling its specific TLS certificate during the setup process.

**Secrets Management**: Kubernetes secrets will be used to securely store and manage certificates, private keys, and sensitive information such as user credentials.

**Automatic Scaling**: Kubernetes will allow automatic scaling of enclave instances based on demand. **Horizontal Pod Autoscalers (HPA)** can monitor the number of active users and spin up or down enclave instances accordingly.

**Service Mesh**: A service mesh (e.g., **Istio** or **Linkerd**) can be implemented within Kubernetes to manage secure communication between licensing servers, enclaves, and users.

**Steps to Develop and Deploy the Architecture**

**Design the Licensing Server**; Develop the licensing server to handle user authentication via username/password and PKI.

Implement TPM quote generation and certificate verification logic.

Set up secure API endpoints for enclave requests

**Set Up PKI Infrastructure;** Deploy a Certificate Authority to issue certificates for users, licensing server, and enclaves.

Configure the licensing server to issue client certificates after username/password authentication.

**Configure Kubernetes Cluster**: Deploy the licensing server as a service on Kubernetes.

Set up the Helm/Kustomize configurations to deploy secure enclave pods with unique TLS certificates

**Integrate TPM**: Implement the TPM quote mechanism to hash enclave certificates and securely return the attestation to clients.

**Test the System**: Perform testing to validate the entire workflow, including:

* + - User authentication (username/password, PKI),
    - Enclave launch and certificate verification,
    - TPM quote verification

**Security Considerations**

* **User Authentication**: Use MFA (multi-factor authentication) for added security.
* **TLS Security**: Enforce the use of TLS 1.3 to ensure secure communication.
* **TPM Security**: Ensure TPM chips are properly configured to securely generate and sign quotes.
* **Auditing and Logging**: Implement logging of enclave launches, user authentication events, and TPM quote verifications for auditing purposes

**REFERENCES;**

* + Kohnfelder, L. Z., & Cryptography (1978). Public-key cryptosystems.
  + Further Reading: Stallings, W. (2017). Cryptography and Network Security: Principles and Practice. Pearson.
  + Further Reading: Winters, R. (2005). Trusted Computing for Real-World Security.
  + Canonical. (2020). Kubernetes Security Best Practices.
  + https://itnext.io/confidential-containers-in-kubernetes-simplifying-adoption-through-standardisation-75c28bfce5e9

PART 2.

**Step-by-Step Implementation of the Architecture with gRPC in Docker and Kubernetes**

I'll focus on using **gRPC** for communication between the licensing server and the user, as well as the licensing server and the enclave.

**Licensing Server Implementation**: The licensing server will handle user authentication via gRPC and manage provisioning of enclaves. It will also return TPM quotes and certificates for the enclave.

**gRPC** will be used for secure, efficient communication between the licensing server, user, and enclave.

**Docker Setup**: Each service (licensing server, enclave) will be containerized using Docker.

**Kubernetes Setup**: The services will be orchestrated with Kubernetes, running each container as a pod.

**Service Communication Workflow**

**User to Licensing Server.** User sends their credentials the licensing server via a gRPC call

The server verifies the credentials.

If successful, the server returns a user-specific session certificate.

**Licensing Server to Enclave**:

Once authenticated, the user requests to connect to an enclave.

The licensing server provisions an enclave through Kubernetes, ensuring each enclave has a unique IP address and TLS certificate.

The licensing server also generates a TPM quote containing the enclave’s TLS certificate hash.

**User Verification of the Enclave**:

The licensing server sends the TPM quote to the user.

The user verifies the enclave’s certificate by comparing the hash in the TPM quote with the certificate.

If the match is successful, the user establishes a secure connection with the enclave.

**Security Enhancements and Key Provisioning**

**Key Provisioning**; New TLS certificates are generated per enclave when an enclave is provisioned.

The certificates are securely stored within the licensing server, protected via encryption.

During provisioning, the licensing server includes the enclave’s certificate in the TPM quote.

**Authentication Handling**: The licensing server handles user authentication via username/password, secured with TLS.

After successful authentication, the user is issued a session certificate via gRPC.

**Establishing a Secure Communication Channel**: Once the enclave is provisioned, the licensing server shares the enclave’s IP address or domain and its TLS certificate with the user.

The licensing server provides a TPM quote to the user, which includes the enclave’s certificate hash.

The user verifies the enclave’s certificate against the hash in the quote before connecting.

If verification passes, a secure TLS connection is established using the enclave’s certificate.

**gRPC Code Snippet for Licensing Server**

import grpc

from concurrent import futures

import tpm\_pb2\_grpc # Generated stubs for gRPC

import tpm\_pb2 # Generated message classes

import hashlib

import os

class LicensingServer(tpm\_pb2\_grpc.LicensingServiceServicer):

def AuthenticateUser(self, request, context):

username = request.username

password = request.password

# Perform authentication (e.g., check against a database)

if valid\_credentials(username, password):

# Generate session certificate

session\_cert = generate\_certificate(username)

return tpm\_pb2.AuthResponse(success=True, certificate=session\_cert)

return tpm\_pb2.AuthResponse(success=False, certificate="")

def ProvisionEnclave(self, request, context):

enclave\_cert = generate\_enclave\_certificate()

enclave\_ip = provision\_enclave\_in\_kubernetes()

# Generate TPM quote with enclave cert hash

enclave\_cert\_hash = hashlib.sha256(enclave\_cert.encode()).hexdigest()

tpm\_quote = generate\_tpm\_quote(enclave\_cert\_hash)

return tpm\_pb2.ProvisionResponse(

enclave\_ip=enclave\_ip,

tpm\_quote=tpm\_quote,

enclave\_cert=enclave\_cert

)

def serve():

server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))

tpm\_pb2\_grpc.add\_LicensingServiceServicer\_to\_server(LicensingServer(), server)

server.add\_insecure\_port('[::]:50051')

server.start()

server.wait\_for\_termination()

if \_\_name\_\_ == '\_\_main\_\_':

serve()

**Security Changes and Communication Channel**

**Key Provisioning**:

Each enclave’s TLS certificate is created securely during provisioning. The certificate's hash is included in the TPM quote.

All certificate generation processes happen in a secure environment with minimal exposure, protected using symmetric encryption where necessary.

**User Authentication**:

The user first connects using a secure gRPC channel (protected via TLS).

After authentication, the user receives a signed session certificate, ensuring mutual trust.

**Enclave Connection**:

The enclave’s TLS certificate is verified by the user against the hash in the TPM quote.

If verification passes, the user establishes a secure TLS connection directly with the enclave using the verified certificate.

**Kubernetes Deployment**:

The licensing server and enclave are run as Kubernetes pods. Kubernetes manages scalability, resilience, and fault tolerance.

Enclave instances can be dynamically deployed on request, each with its unique IP or domain and corresponding certificate.

**Description of the above;** The architecture leverages Kubernetes for scalable deployment and incorporates robust security mechanisms for key provisioning, user authentication, and enclave connection. TLS certificates for each enclave are securely generated during provisioning, with the hash of the certificate included in a TPM quote for verification. Users authenticate via a secure gRPC channel protected by TLS, and upon successful authentication, they receive a signed session certificate to establish mutual trust. The user then verifies the enclave’s TLS certificate against the TPM quote before establishing a secure connection to the enclave. This ensures that only trusted, untampered enclaves are accessed. Kubernetes dynamically manages the deployment of licensing servers and enclaves, providing scalability, fault tolerance, and flexibility, with each enclave assigned a unique IP or domain. This architecture ensures a highly secure and efficient deployment of enclaves, protected by PKI and TPM-based verification.