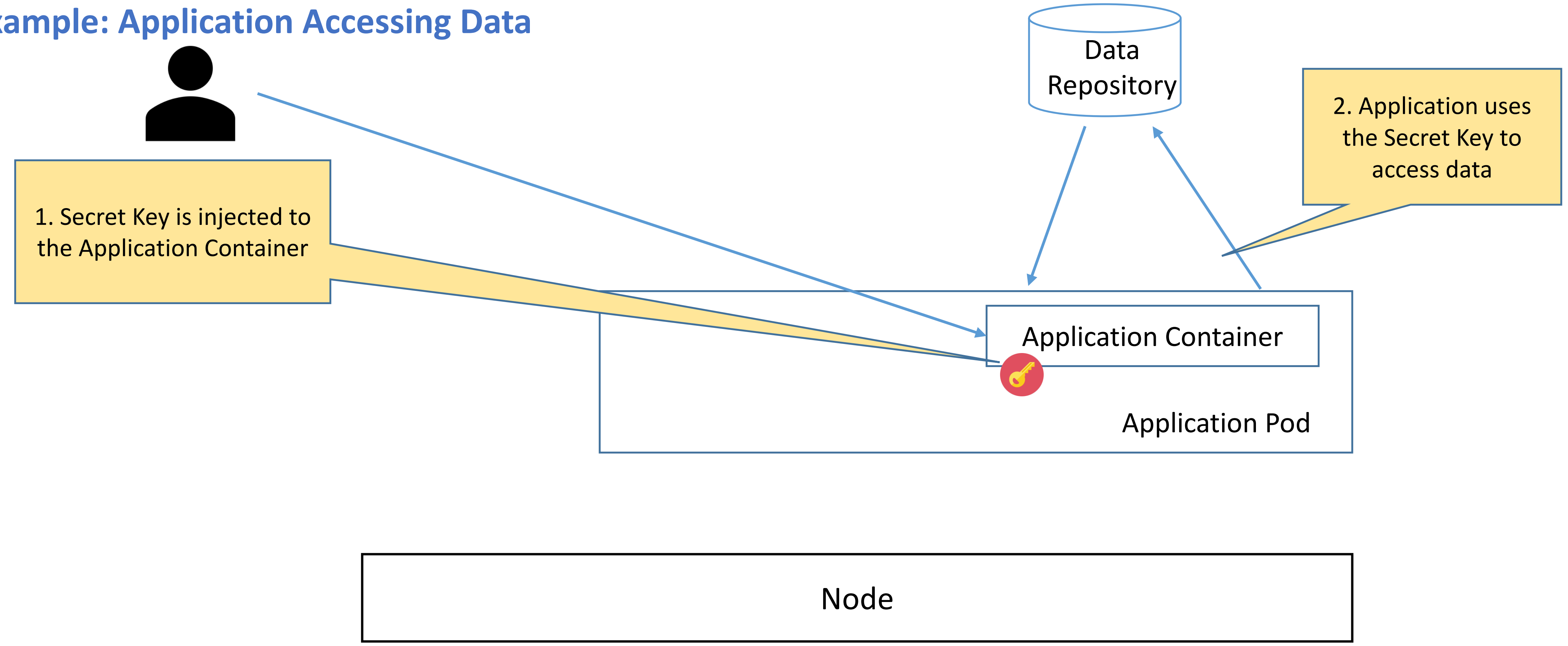
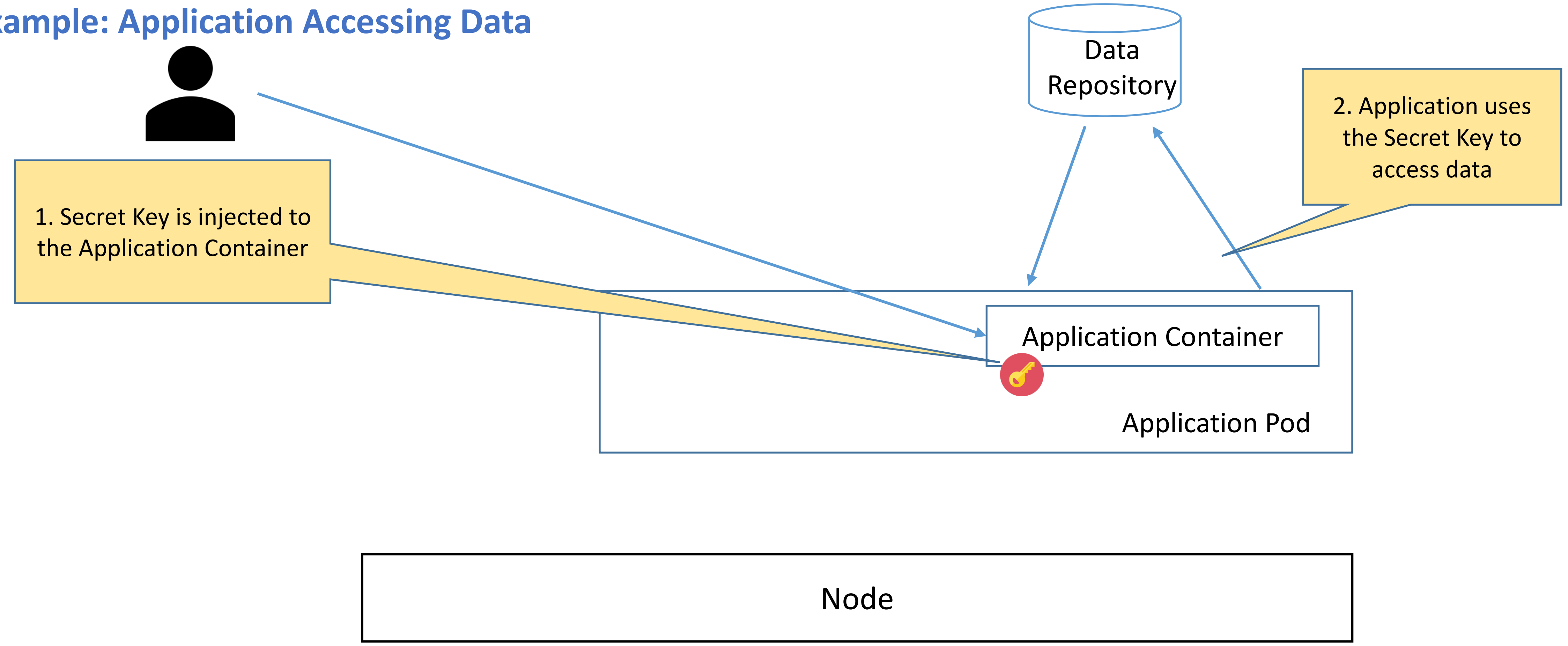
Attestation with SPIRE

Keylime and Confidential Containers

Example: Application Accessing Data


Data Repository
2. Application uses the Secret Key to access data
1. Secret Key is injected to the Application Container
Application Container
Application Pod
Node

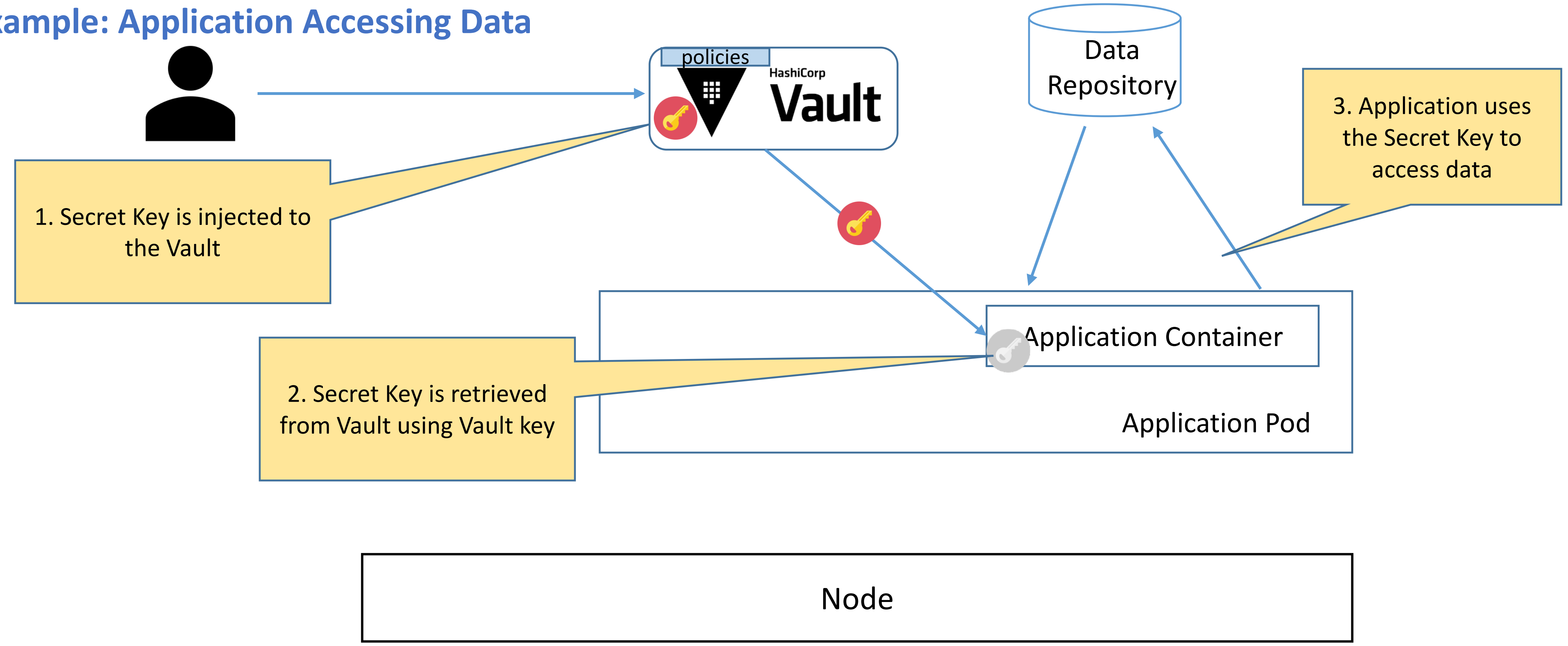
Example: Application Accessing Data


Data Repository
2. Application uses the Secret Key to access data
1. Secret Key is injected to the Application Container
Application Container
Application Pod
Node

Problem

- The Secret Key must be securely delivered to the Application Container, but nowhere else
- The Secret Key cannot be embedded with CI/CD pipeline; therefore, it must be handled by trusted human administrator, preventing automation
- The Secret Key can be read by anyone with *an exec* access to the Signing Container
- The Secret Key remains static and cannot be easily changed without human interaction

Example: Application Accessing Data


policies
HashiCorp Vault
Data Repository
3. Application uses the Secret Key to access data
1. Secret Key is injected to the Vault
Application Container
2. Secret Key is retrieved from Vault using Vault key
Application Pod
Node

Problem

- The Secret Key must be securely delivered to the Application Container, but nowhere else
- The Secret Key cannot be embedded with CI/CD pipeline; therefore, it must be handled by trusted human administrator, preventing automation
- The Secret Key can be read by anyone with *an exec* access to the Signing Container
- ~~The Secret Key remains static and cannot be easily changed without human interaction~~

Identity Based Secret Distribution with SPIRE

Problem

- The Secret Key must be securely delivered to the Application Container, but nowhere else
- The Secret Key cannot be embedded with CI/CD pipeline; therefore, it must be handled by trusted human administrator, preventing automation
- The Secret Key can be read by anyone with *an exec* access to the Signing Container
- The Secret Key remains static and cannot be easily changed without human interaction

Solution w/ SPIRE

- The key (or set of keys) remain stored securely in Vault
- Different keys are used for different pipelines/applications reducing the blast radius
- The keys are periodically rotated
- The Vault policies enforce what pipelines/application can get access to what keys based on the workload identity managed by SPIRE
- The Sidecar Container, assisting the Application Container, manages the Identity Token and requests the Secret Key from the Vault on the behalf of the Application Pod
- The SPIRE Agent is attested by the Kubernetes platform

Zero Trust Workload Identity Framework

CLOUD NATIVE COMPUTING FOUNDATION

spiffe

- Identity Specification
- Defines identity format
- Specifies how workloads securely obtain identity

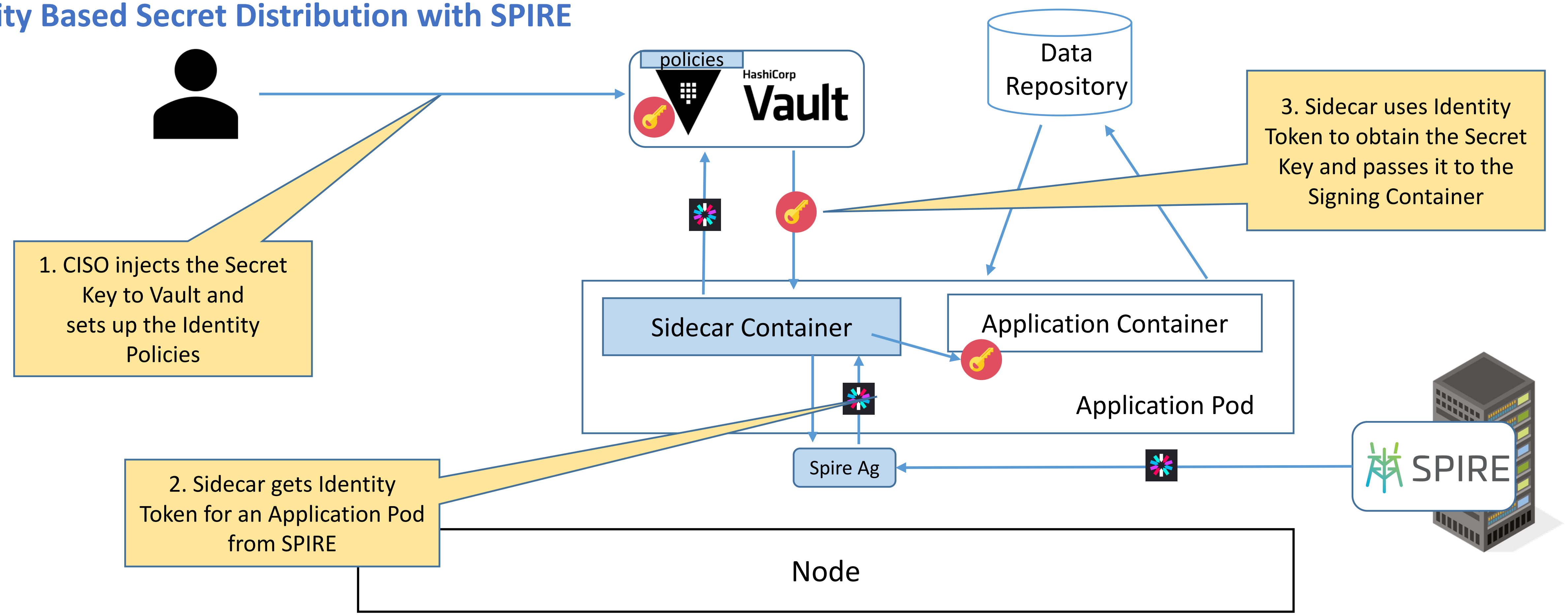
SPIRE

- Implementation of SPIFFE (runtime)
- Issuance/rotation of x509/JWT
- Provide zero trust attestation of workload & infrastructure
- Provide single point of federation with OIDC discovery

tornjak

- Control plane/UI for SPIRE
- Provides visibility/management for workload identities
- Together with k8s registrar, provides universal workload identity

Identity Based Secret Distribution with SPIRE



policies
HashiCorp Vault
Data Repository
3. Sidecar uses Identity Token to obtain the Secret Key and passes it to the Signing Container
1. CISO injects the Secret Key to Vault and sets up the Identity Policies
Sidecar Container
Application Container
Application Pod
SPIRE
Spire Ag
2. Sidecar gets Identity Token for an Application Pod from SPIRE
Node

Example of Universal Workload identity

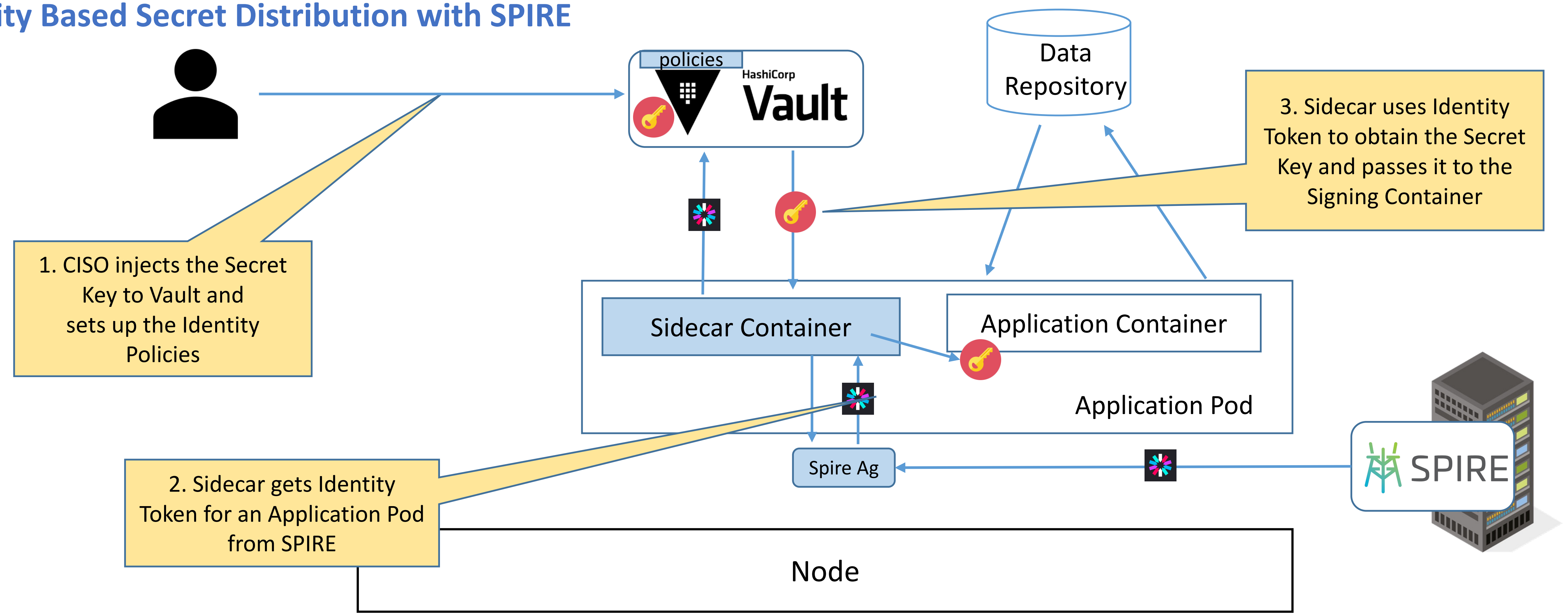
```
spiffe://<TrustDomain>/region/<Region>/cluster_name/<ClusterName>/ns/<Namespace>/sa/<ServiceAccount>/pod_name/<PodName>
```

Access Policy for Vault secrets :

- Region must be in the US "us-*"
- Any cluster name
- Namespace is "space"
- ServiceAccount is "nasa"
- PodName starts with "mars-mission"

```
"spiffe://openshift.space-x.com/region/us-ykt/cluster_name/css/
  ns/space/sa/nasa/pod_name/mars-mission-7874fd667c-72mtp"
```

Identity Based Secret Distribution with SPIRE

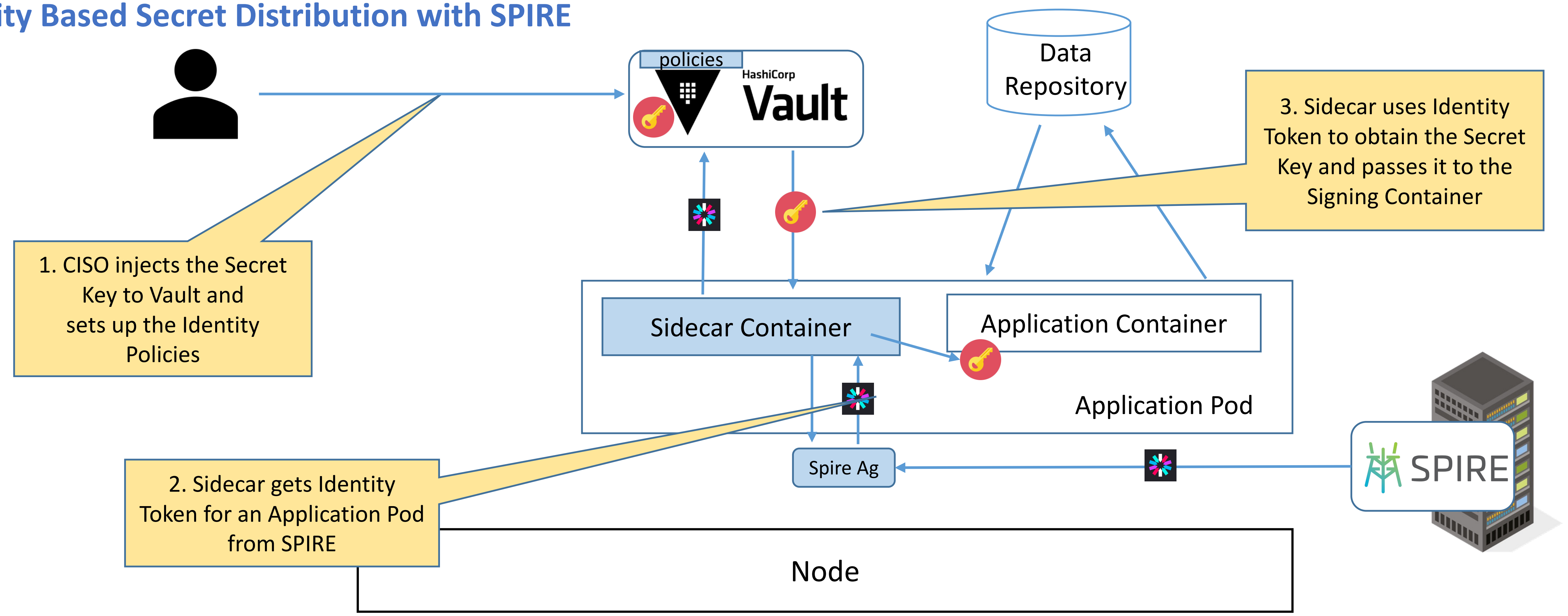


1. CISO injects the Secret Key to Vault and sets up the Identity Policies

2. Sidecar gets Identity Token for an Application Pod from SPIRE

3. Sidecar uses Identity Token to obtain the Secret Key and passes it to the Signing Container

Identity Based Secret Distribution with SPIRE


policies
HashiCorp Vault
Data Repository
3. Sidecar uses Identity Token to obtain the Secret Key and passes it to the Signing Container
1. CISO injects the Secret Key to Vault and sets up the Identity Policies
Sidecar Container
Application Container
Application Pod
Spire Ag
SPIRE
2. Sidecar gets Identity Token for an Application Pod from SPIRE
Node

New Problem

- Even though the Workload Identity is tied to Kubernetes, it is not tied to the hardware, infrastructure or a Cloud Provider
- Can we really trust the host? Do we know what is running there?
- Underlying software stack might have been tampered with

Identity Based Secret Distribution with SPIRE

New Problem

- Even though the Workload Identity is tied to Kubernetes, it is not tied to the hardware, infrastructure or a Cloud Provider
- Can we really trust the host? Do we know what is running there?
- Underlying software stack might have been tampered with

Solution w/ SPIRE + Keylime + TPM/vTPM

Node is attested by Keylime (and TPM). This ties the Workload Identity with Hardware Root of Trust:

- It guarantees ***the identity of the node beyond any doubt***
- It **attests the software stack**, from booting to the kernal. Enforcement of the software bill of materials (SBOM)
  We know the firmware, packages, libraries
- It measures and enforces the ***integrity of files*** (IMA)

Comments:

- *Keylime *servers* are trusted. Nodes on which Keylime agent runs are *not* trusted.*
- *The node stored private key must be well protected after it has been injected by Keylime, so no other entity can use it to sign other attestation challenges.*

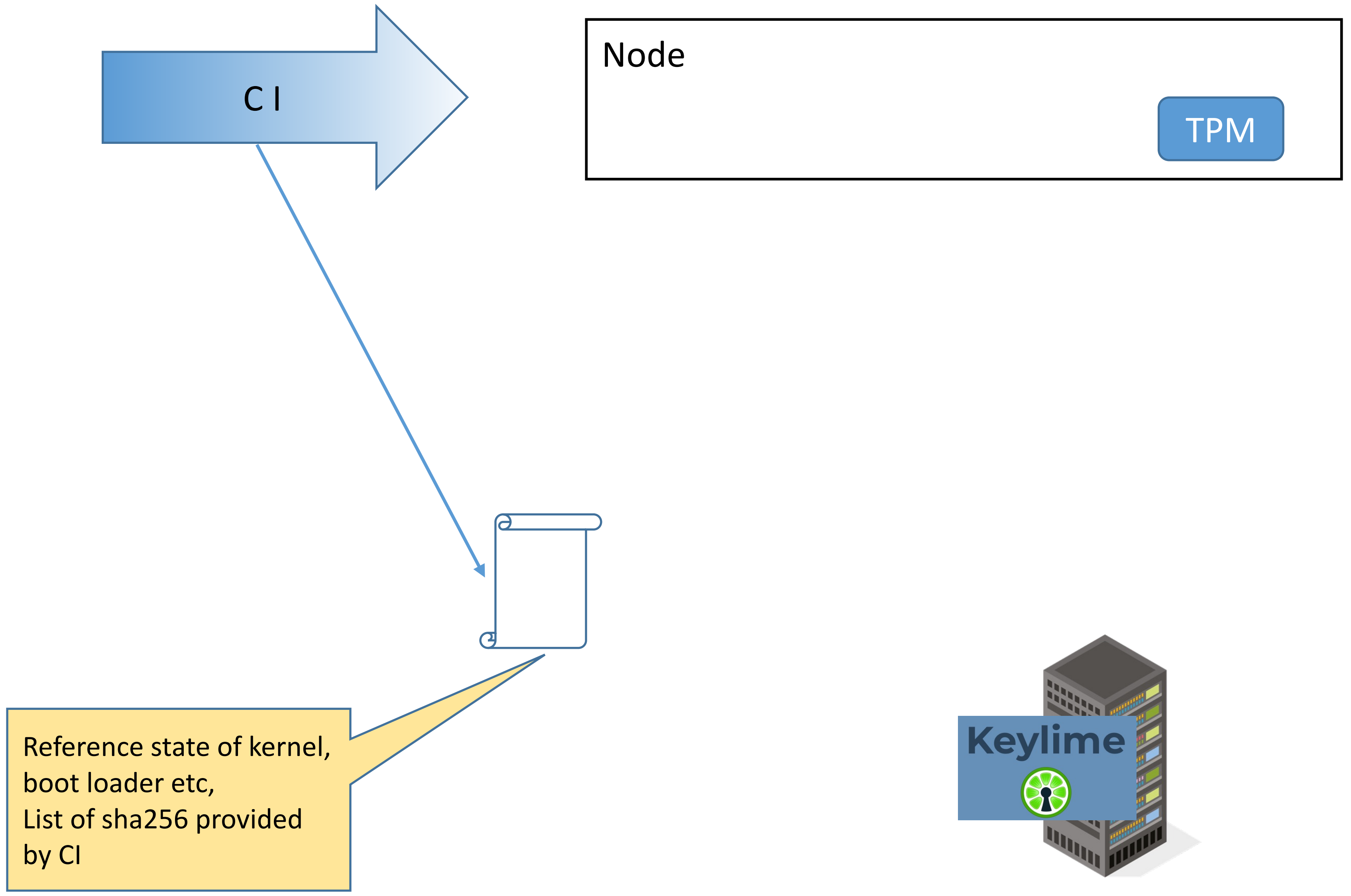
"Keylime guarantees the identity of a node beyond doubt"

- That's a bold claim. Concepts and technologies that make this possible:
  - Endorsement Key aka EK – private/public keypair unique to a TPM device. Private key is locked forever inside the device. The identity of the TPM device is provable modulo RSA or ECC collisions (we are not there yet).
  - EK certificate – proves the TPM genuine by (a) signing the EK (b) signed by TPM manufacturer root CA
  - Measured Boot Attestation aka MBA – a technique to identify ("measure") and validate ("attest") every component of the software stack as it is being built up during boot.
  - Integrity Measurement Attestation aka IMA – a technique to measure each executable in a running operating system and attest that it genuine.
  - Root of trust: one of several hardware techniques to ensure TPM device is initialized before any other SW runs on machine, and therefore TPM initialization is not "contaminated"
  - Chain of trust: i.e. "measure before you run" – each component of the software stack measures the next piece before it runs. Measurement digests are recorded in the TPM device.
  - Remote attestation -- perform TPM identity test, MBA, IMA and possibly other checks on a host *remotely*.
    - The untrusted node records MBA and IMA measurements, also recording digests in the TPM device
    - The [verifiably] genuine TPM provides an unfalsifiable record of the digests above
    - The MBA and IMA measurements, and a TPM "quote" are downloaded to a remote (trusted) keylime server
    - The server authenticates the measurements using the TPM quote
    - The server checks measurements against the SBOM and declares attestation success or failure.
  - Keylime (https://github.com/keylime/keylime) is an open-source implementation of remote attestation.

SPIRE Nodes Attestation with Keylime

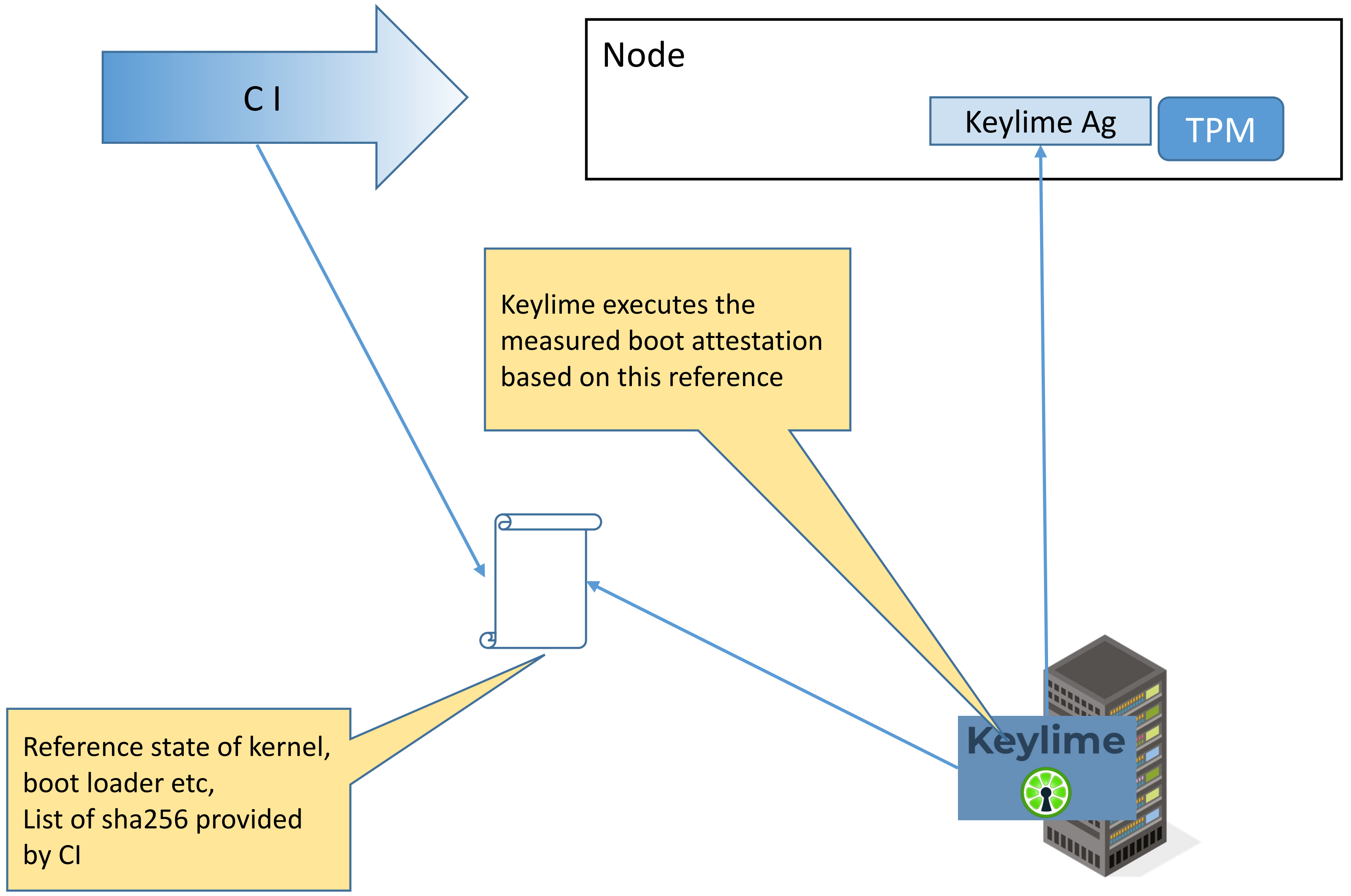
- *Root CA* is stored securely on the SPIRE Server
- *Root CA* signs *an intermediate CA* for distributions
- Keylime executes a suite of attestations and checks, before the node is marked as *verified*
- Once the node is verified, Keylime delivers there *an intermediate CA* (encrypted; cannot be read on a different node)
- Verified node creates ***node x509*** signed with *intermediate CA*
- SPIRE agent uses ***node x509*** to attest with SPIRE Server
- When SPIRE Agent is successfully attested, SPIRE Server assigns identity for it, and for all the workloads hosted by this node
- ***node x509*** certs are short-lived and they are continuously updated as a result of successful Keylime attestation
- Keylime node attestation failure (e.g., measured boot) causes **ban** of the SPIRE agent and stops renewing **node x509** (node is untrusted)
- No identities can be issued for the workloads hosted on this node

Node Attestation and SPIRE Agent Revocation with Keylime

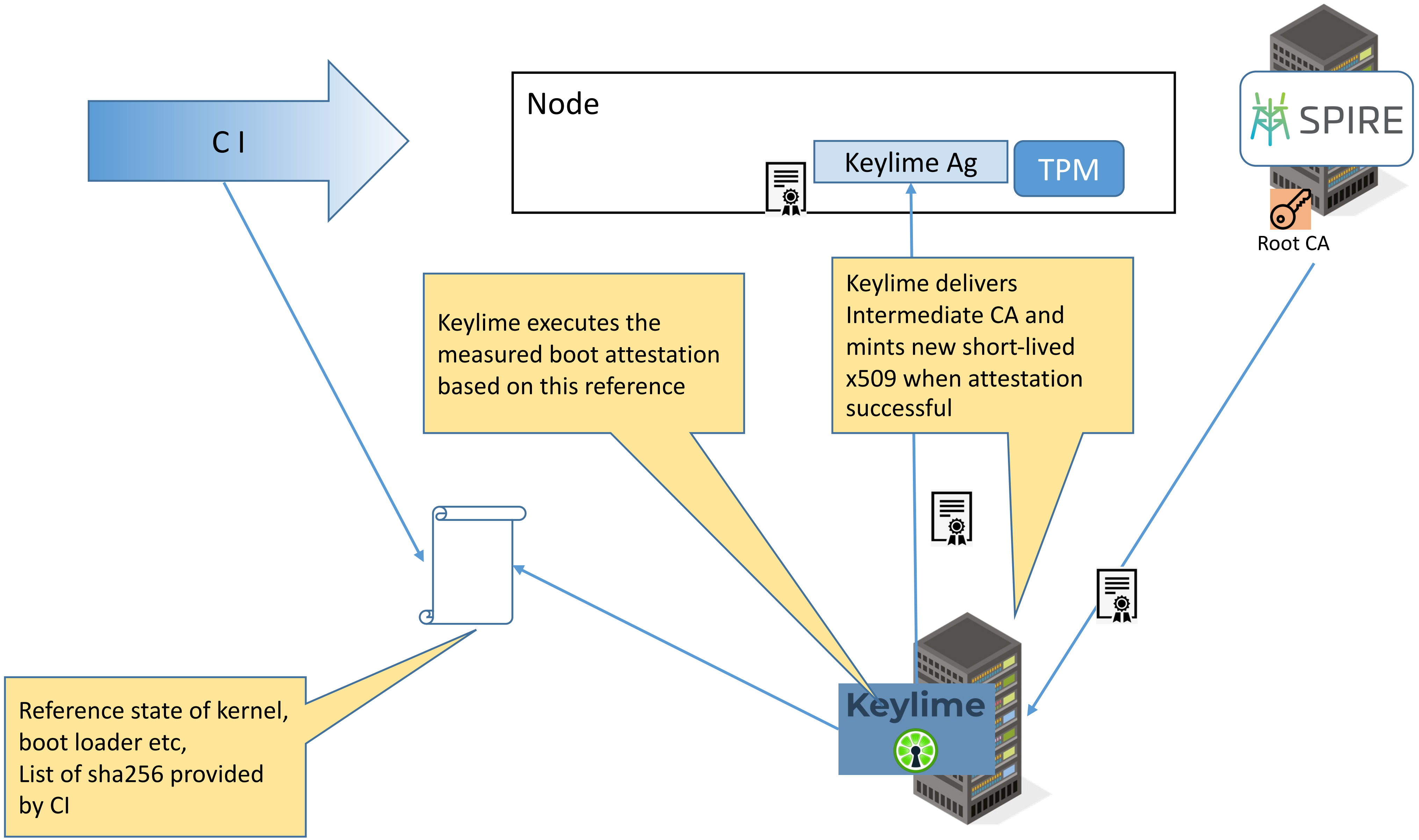


Reference state of kernel, boot loader etc, List of sha256 provided by CI

Node Attestation
and SPIRE Agent Revocation with Keylime


C I
Node
Keylime Ag
TPM
Keylime executes the measured boot attestation based on this reference
Reference state of kernel, boot loader etc, List of sha256 provided by CI
Keylime

Node Attestation and SPIRE Agent Revocation with Keylime

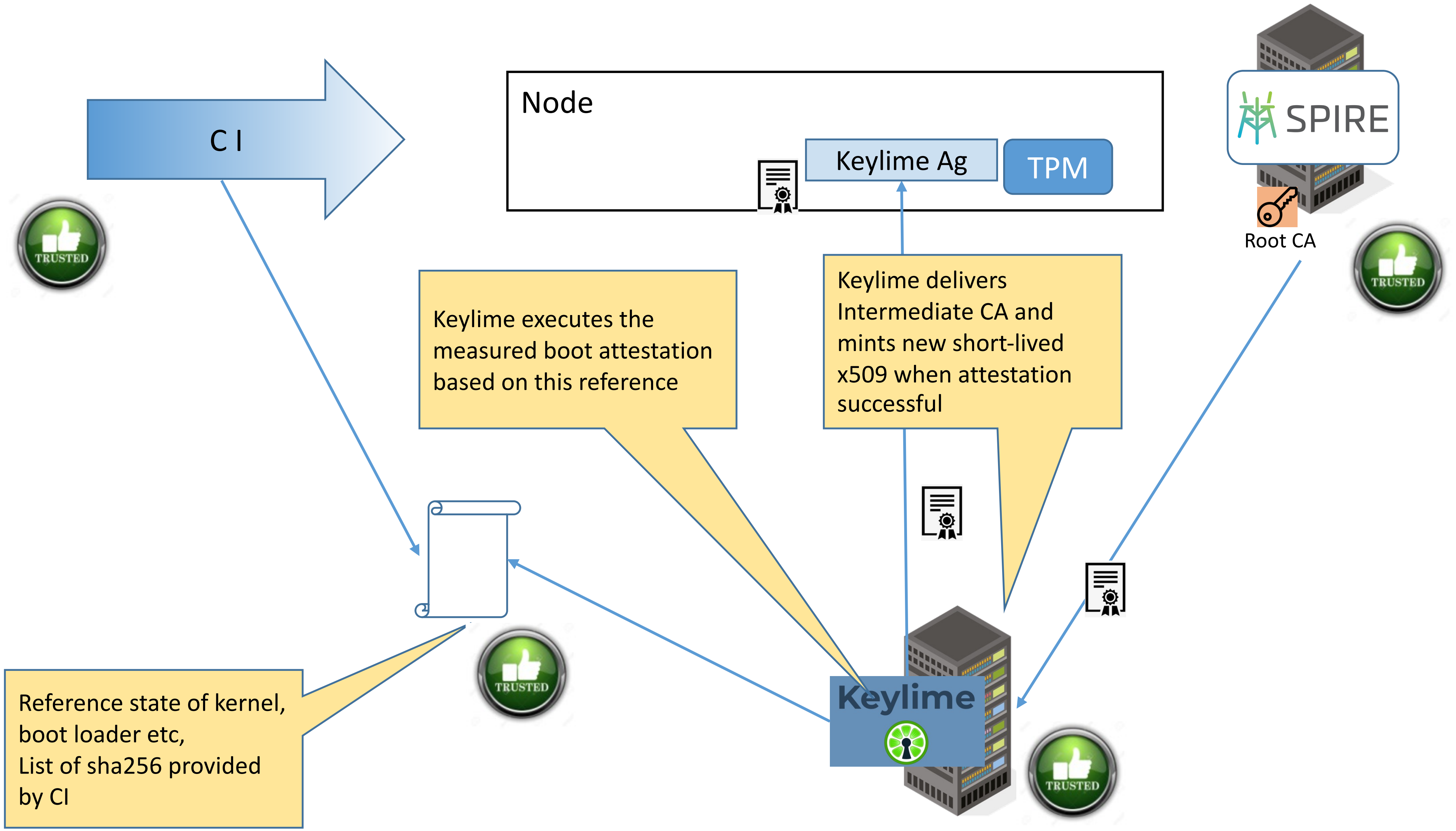


Keylime executes the measured boot attestation based on this reference

Keylime delivers Intermediate CA and mints new short-lived x509 when attestation successful

Reference state of kernel, boot loader etc,
List of sha256 provided by CI

Node Attestation and SPIRE Agent Revocation with Keylime

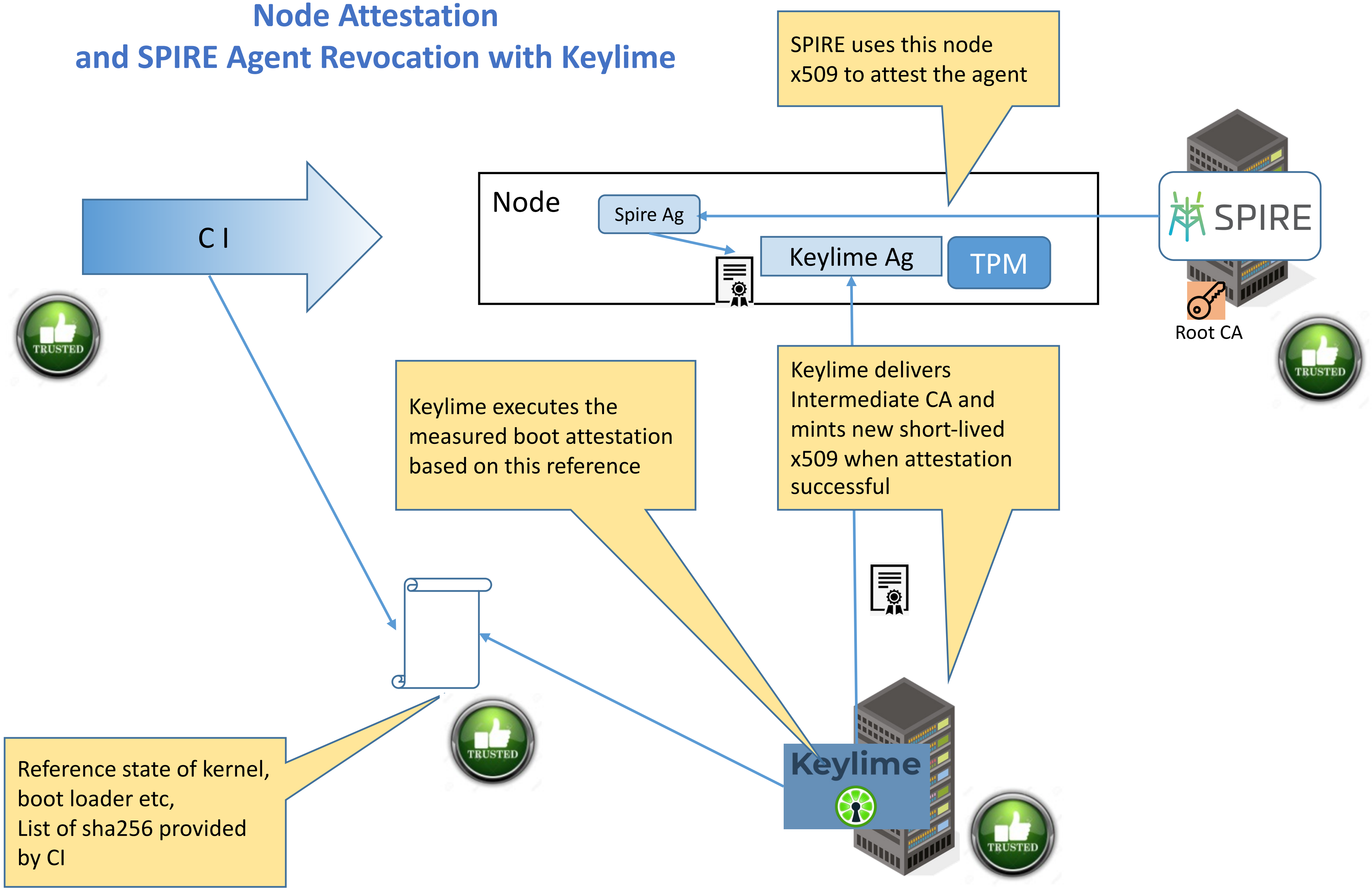


Keylime executes the measured boot attestation based on this reference

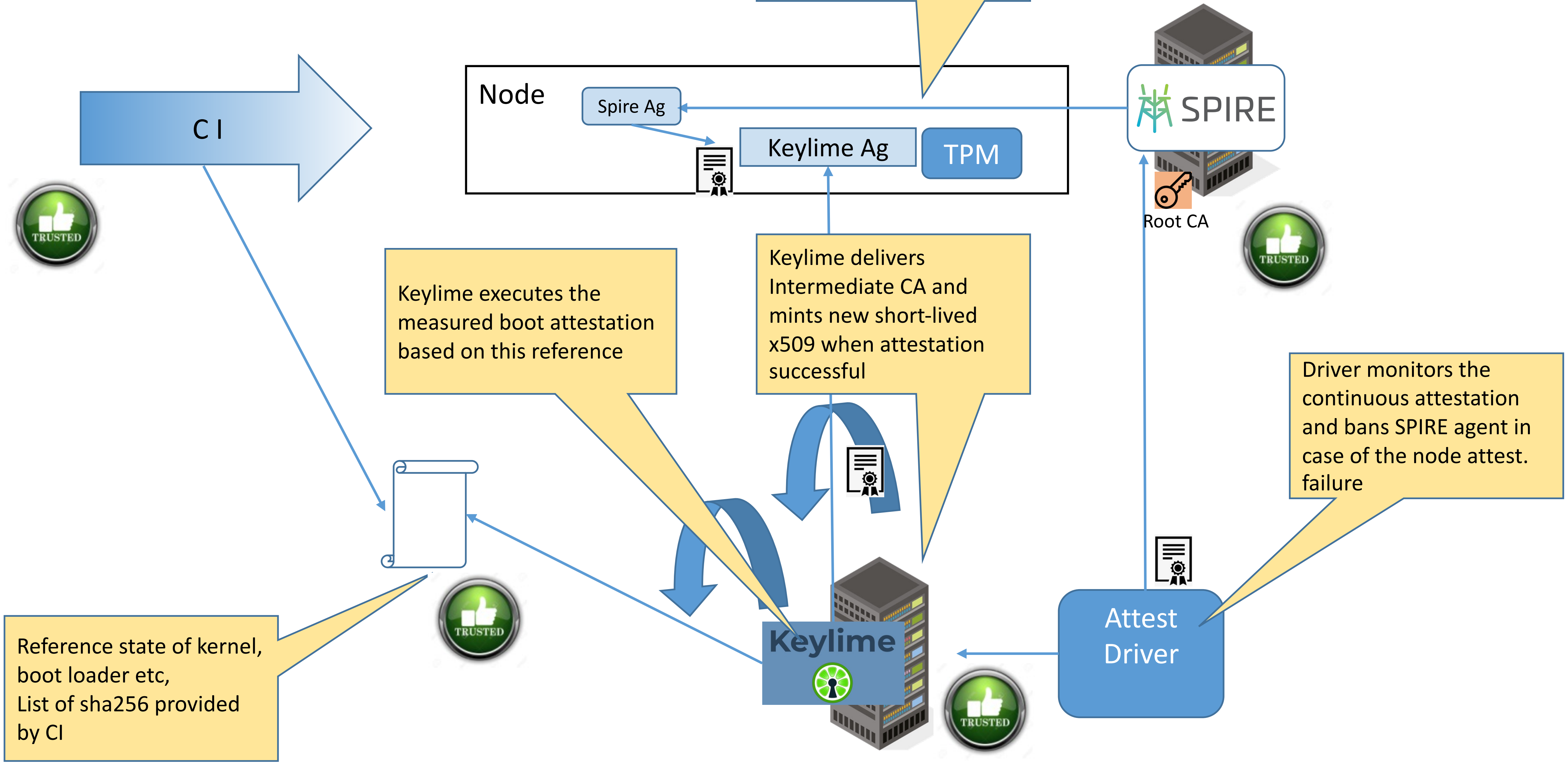
Keylime delivers Intermediate CA and mints new short-lived x509 when attestation successful

Reference state of kernel, boot loader etc, List of sha256 provided by CI

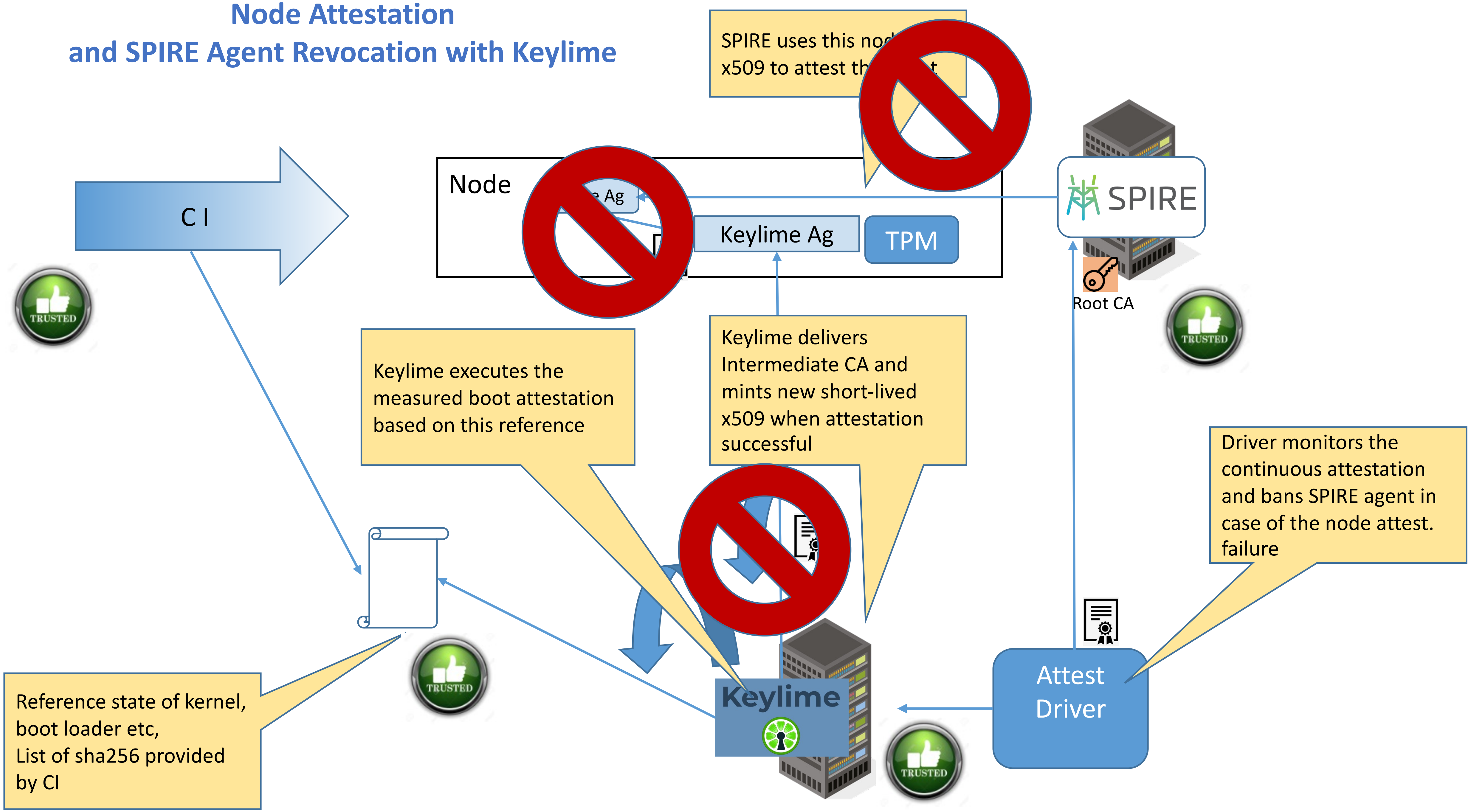
Node Attestation and SPIRE Agent Revocation with Keylime


SPIRE uses this node x509 to attest the agent
C I
Node
Spire Ag
Keylime Ag
TPM
SPIRE
Root CA
TRUSTED
Keylime executes the measured boot attestation based on this reference
Keylime delivers Intermediate CA and mints new short-lived x509 when attestation successful
TRUSTED
Reference state of kernel, boot loader etc, List of sha256 provided by CI
TRUSTED
Keylime
TRUSTED

Node Attestation and SPIRE Agent Revocation with Keylime


SPIRE uses this node x509 to attest the agent
C I
Node
Spire Ag
Keylime Ag
TPM
SPIRE
Root CA
TRUSTED
Keylime executes the measured boot attestation based on this reference
Keylime delivers Intermediate CA and mints new short-lived x509 when attestation successful
Driver monitors the continuous attestation and bans SPIRE agent in case of the node attest. failure
Reference state of kernel, boot loader etc, List of sha256 provided by CI
Keylime
Attest Driver

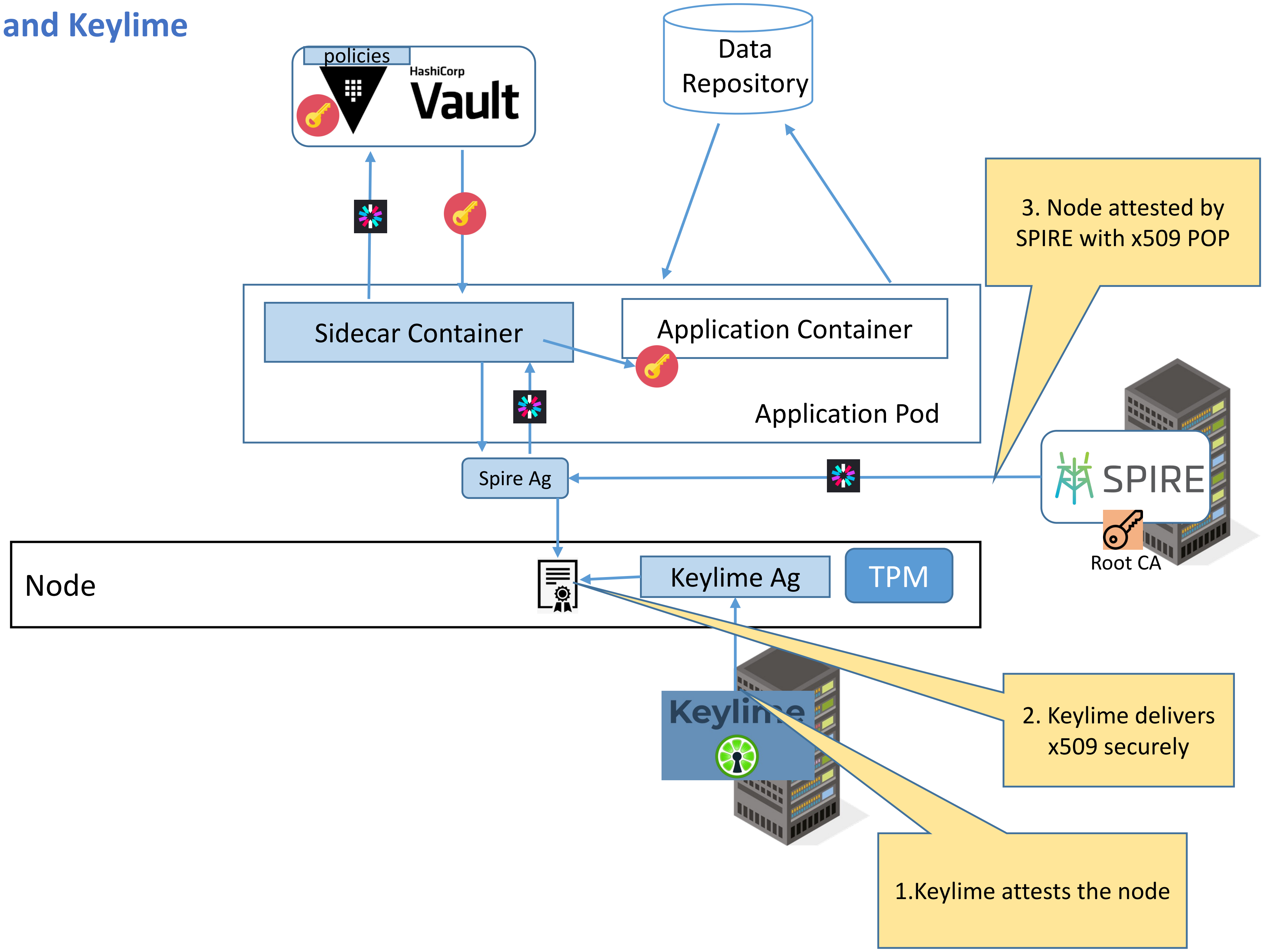
Node Attestation and SPIRE Agent Revocation with Keylime


SPIRE uses this nod x509 to attest th t
Node
e Ag
Keylime Ag
TPM
SPIRE
Root CA
C I
TRUSTED
Keylime executes the measured boot attestation based on this reference
Keylime delivers Intermediate CA and mints new short-lived x509 when attestation successful
Driver monitors the continuous attestation and bans SPIRE agent in case of the node attest. failure
Keylime
Attest Driver
Reference state of kernel, boot loader etc, List of sha256 provided by CI

Node Attestation and SPIRE Agent Revocation with Keylime

- Tooling for accessing SPIRE Server remotely and securely
- SPIRE Agent returns cached identities when it is banned: https://github.com/spiffe/spire/issues/2700
- Re-attest existing Agent after Agent NodeAttestor configuration changes: https://github.com/spiffe/spire/issues/3133
- Re-attestation of agents: https://github.com/spiffe/spire/pull/3031

Node Attestation with TPM and Keylime



Node Attestation - SPIRE Server

- keep **Root CA private key** secure and **Certificate** for validation
- create **Intermediate CA key** and sign **Intermediate Certificate** to be distributed to worker nodes

Steps:
1. Generate rootCA key --> rootCA.key
2. Create rootCA cert (rootCA.key) --> rootCA.pem
3. Generate intermediate Key --> interim.key
4. Create intermediate cert (interim.key, rootCA.key, rootCA.pem) --> interim.pem

Server NodeAttestor just needs the rootCA cert for verification (rootCA.pem)

```
NodeAttestor "x509pop" {
 plugin_data {
    ca_bundle_path = "/opt/spire/sample-keys/rootCA.pem"
  }
}
```

Node Attestation - SPIRE Agent

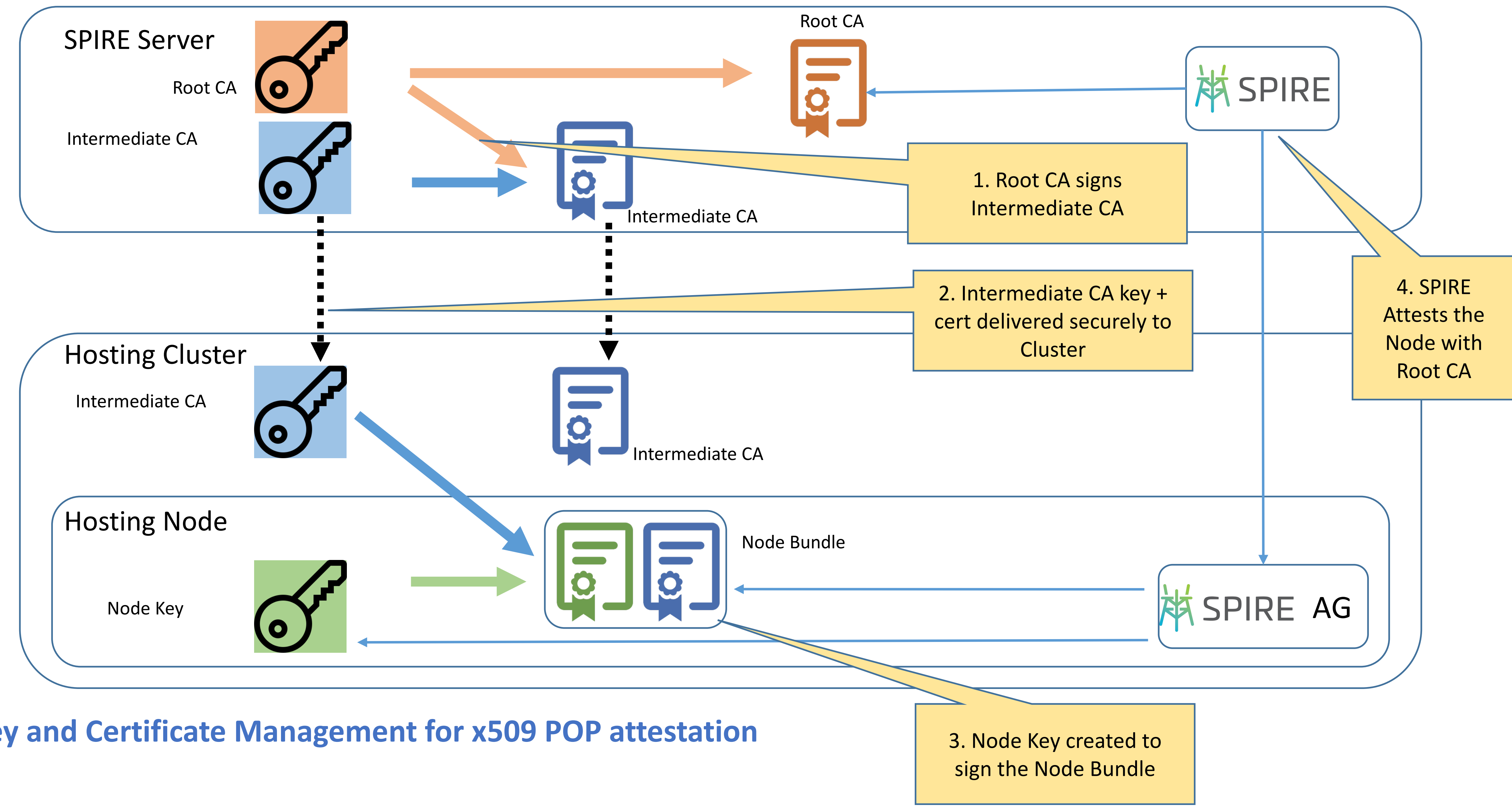
- obtain **Intermediate CA** and **key**
- create **node Certificate** signed with **Intermediate CA** and store on the host
- submit it to SPIRE Server for verification

Steps (for every Worker Node):
1. Deliver securely *iterim.pem* and *interim.key* to each hosting node
2. Generate node key --> node.key
3. Create node cert (node.key, interim.key, interim.pem) --> node.pem
and store them securely on the node

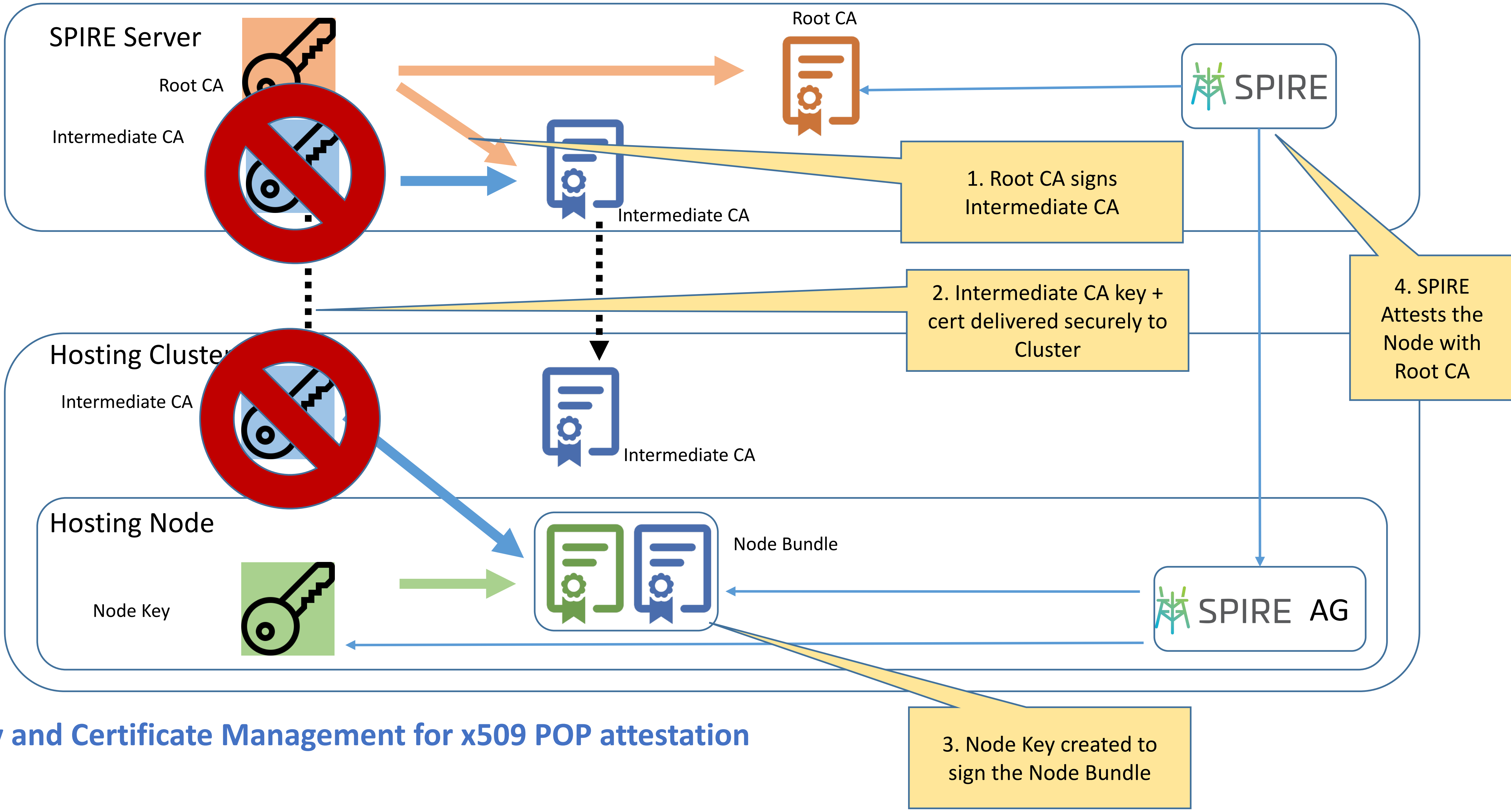
SPIRE Agent uses node.key and a cert bundle (node.pem + interim.pem) --> cert.pem

The files should be stored on individual hosting nodes

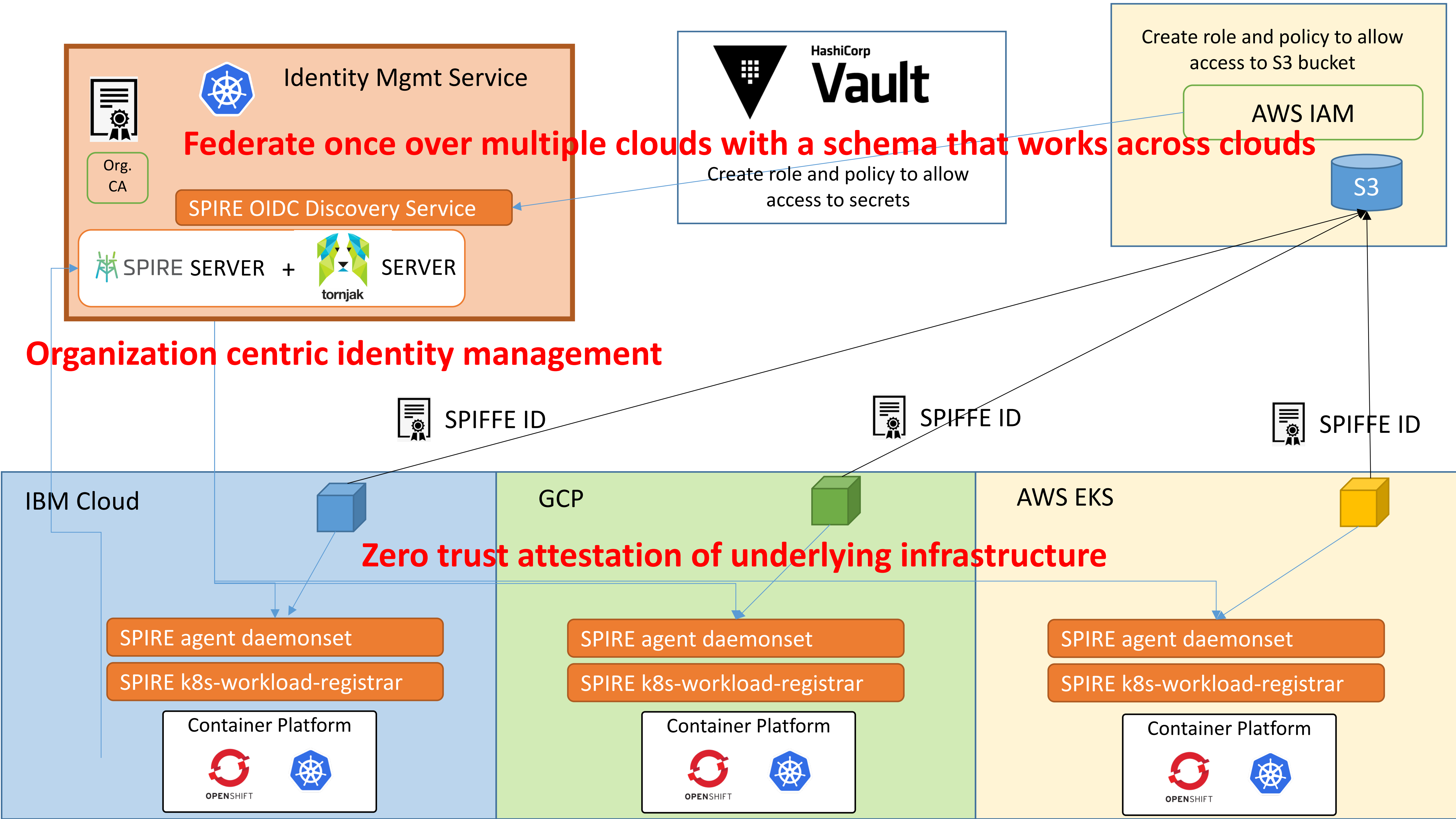
```
plugins {
  NodeAttestor "x509pop" {
    plugin_data {
      private_key_path = "/run/spire/agent/key.pem"
      certificate_path = "/run/spire/agent/cert.pem"
   }
}
```


SPIRE Server
Root CA
Intermediate CA
Root CA
SPIRE
Intermediate CA
1. Root CA signs Intermediate CA
2. Intermediate CA key + cert delivered securely to Cluster
4. SPIRE Attests the Node with Root CA
Hosting Cluster
Intermediate CA
Intermediate CA
Hosting Node
Node Bundle
Node Key
SPIRE AG
3. Node Key created to sign the Node Bundle

Key and Certificate Management for x509 POP attestation


SPIRE Server
Root CA
Intermediate CA
Root CA
SPIRE
Intermediate CA
1. Root CA signs Intermediate CA
2. Intermediate CA key + cert delivered securely to Cluster
4. SPIRE Attests the Node with Root CA
Hosting Cluster
Intermediate CA
Intermediate CA
Hosting Node
Node Bundle
Node Key
SPIRE AG
3. Node Key created to sign the Node Bundle

Key and Certificate Management for x509 POP attestation



Federate once over multiple clouds with a schema that works across clouds

Organization centric identity management

Zero trust attestation of underlying infrastructure

BACKUP