From Adoption to Innovation: LinkedIn's SPIRE Journey

Junyuan Zeng
Staff SWE
LinkedIn

Wei Zhang
Senior Staff SWE
Linkedin

OPEN SOURCE SecurityCon

Agenda

- Background and Motivations
- Adoption at LinkedIn On-Prem
- Key Challenge
- Design & Innovation
  Building Trust
  Identity Issuance
  An Agentless Architecture
- Lessons Learned
- Q&A

Limitations of Our Legacy In-House CA

Before SPIRE, LinkedIn's internal PKI framework relied on an in-house CA to provide certificates as identities for various parties.

High maintenance effort	Hand-written HTTP server and client, hard to maintain and scale
Limited scalability & availability	Server directly handled all certificate requests, limiting throughput and causing high latency for users fetching certificates; Easily overwhelmed by traffic spikes
No standard identity	Custom SAN – not interoperable with standard PKI implementations; e.g., caused "invalid name syntax" (LibreSSL)
Lack of certificate lifecycle management	Lacked automated certificate rotation and tracking; Cannot issue short-lived certificates due to legacy design
Limited security & extensibility	lacks hardware root of trust; attestation limited to DNS/IP

What is SPIFFE & SPIRE

SPIFFE

- Set of open-source standards for identity
- SPIFFE Verifiable Identity Document (SVID)
  X.509 Certificate, JWT Token Format
- **SPIFFE URI**: spiffe://<trustdomain>/<identity path>

SPIRE

- Open-source SPIFFE implementation
- **Server:** Central authority managing and signing identities
- **Agent:** Node-level attestation and certificate management
- API-centric, with extensible plugin frameworks

Why We Chose It

1. **Foundational**
   Can run on bare-metal with minimal dependencies

2. **Scalable & Highly Available**
   Reliably scales and maintains high availability through caching and resilient design

3. **Extensible**
   Easily extended for security hardening and on-prem integrations

4. **Strong community support**
   Continuously maintained and improved by an active community

Adoption at LinkedIn's On-Prem

Trust Model

- Each deployment environment has its own trust domain (e.g., prod.li, stg.li).
- Each trust domain has a corresponding signing CA chain with a name constraint.
- Existing infrastructure distributes CA bundles to all hosts, including those beyond SPIRE.

SPIFFE Identity

- Establishes an internal standard for SPIFFE identities across all systems.
- URI Example: spiffe://prod.li/application/..., spiffe://prod.li/user/...
- X.509: Additional metadata, such as region, is encoded in certificate extensions.

AuthN / AuthZ Integration

- Update in-house existing authentication and authorization libraries to integrate new SPIFFE based identities.

Adoption at LinkedIn's On-Prem - Overall diagram

SPIRE Server

- Deployed in regional clusters
- Use HSM-backed Root CA
- Standard & custom plugins, plus additional controllers

SPIRE Agent

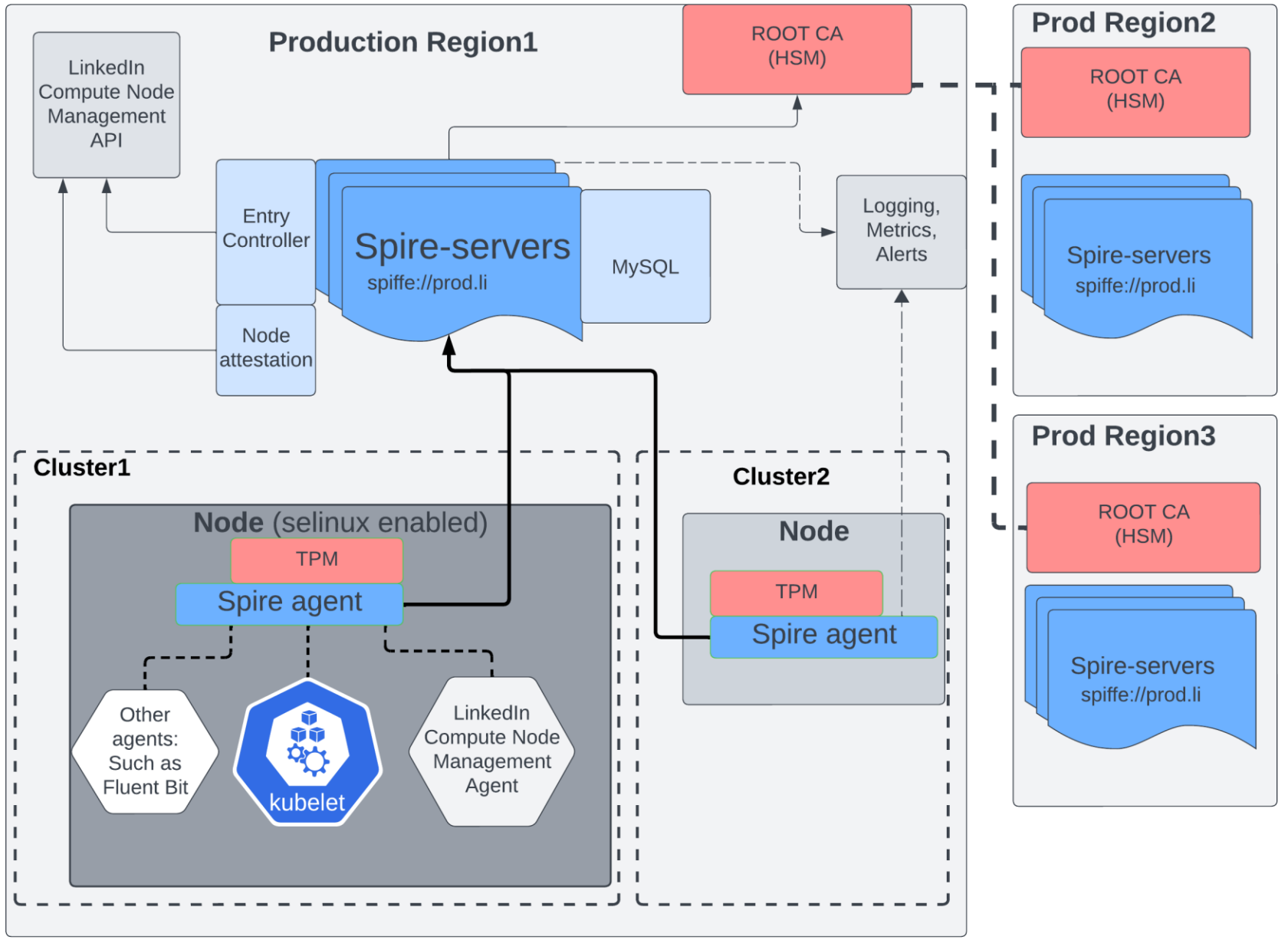
- Run as Systemd Service on SELinux-enabled host
- Leverage TPM when available

High Availability & Scale

- 5 server instances per cluster with client-side load balancing
- Supports up to 50,000 nodes per cluster

Observability

- Integrated with LinkedIn logging and metrics



Key Challenges

I. **Building Trust**
   How to build a strong chain of trust to securely run SPIRE Servers and agents?

II. **Issuing Identities**
   How to dynamically register entries?

III. **What if Running SPIRE Agent Isn't Always Applicable?**

Building Trust of SPIRE Server - CA Bootstrap

Deployment

- SPIRE Server runs on security-hardened host (e.g., full machine isolation etc.).

Signing CA and Leaf Certificate Generation

- Authenticate to HSM via TPM-sealed PIN.
- The HSM issues a signing CA, which is used to issue leaf certificates.

Signing CA Storage: Security vs. Performance Trade-offs

- HSM and TPM signing adds additional overhead.
- Chose SPIRE in-memory CA signing to meet performance requirements.

HSM Root CA → **HSM prod.li Int. CA** → **SIRE In-memory CA** → Leaf Certificate

Building Trust of Agent Node: TPM based Node Attestation

OPEN SOURCE SecurityCon

What is TPM

- TPM is a locked box proving a node is trusted (Authenticity and Integrity).
- Trust is rooted in a unique factory-embedded Endorsement Key (EK).
- EK consists of a public & private key pair.

What is TPM-based Node Attestation

- Use TPM to verify nodes and grant access only to trusted ones.
- Trust Chain of Authenticity: Vendor Certificate → EK → TPM → Node Attestation.

What is Needed Before TPM-based Node Attestation

- Verifies TPM authenticity, not org-managed TPM or node.
- An TPM managed by an attacker can pass the trust chain verification.
- Identify managed nodes and register their EKs to establish the root of the trust chain.

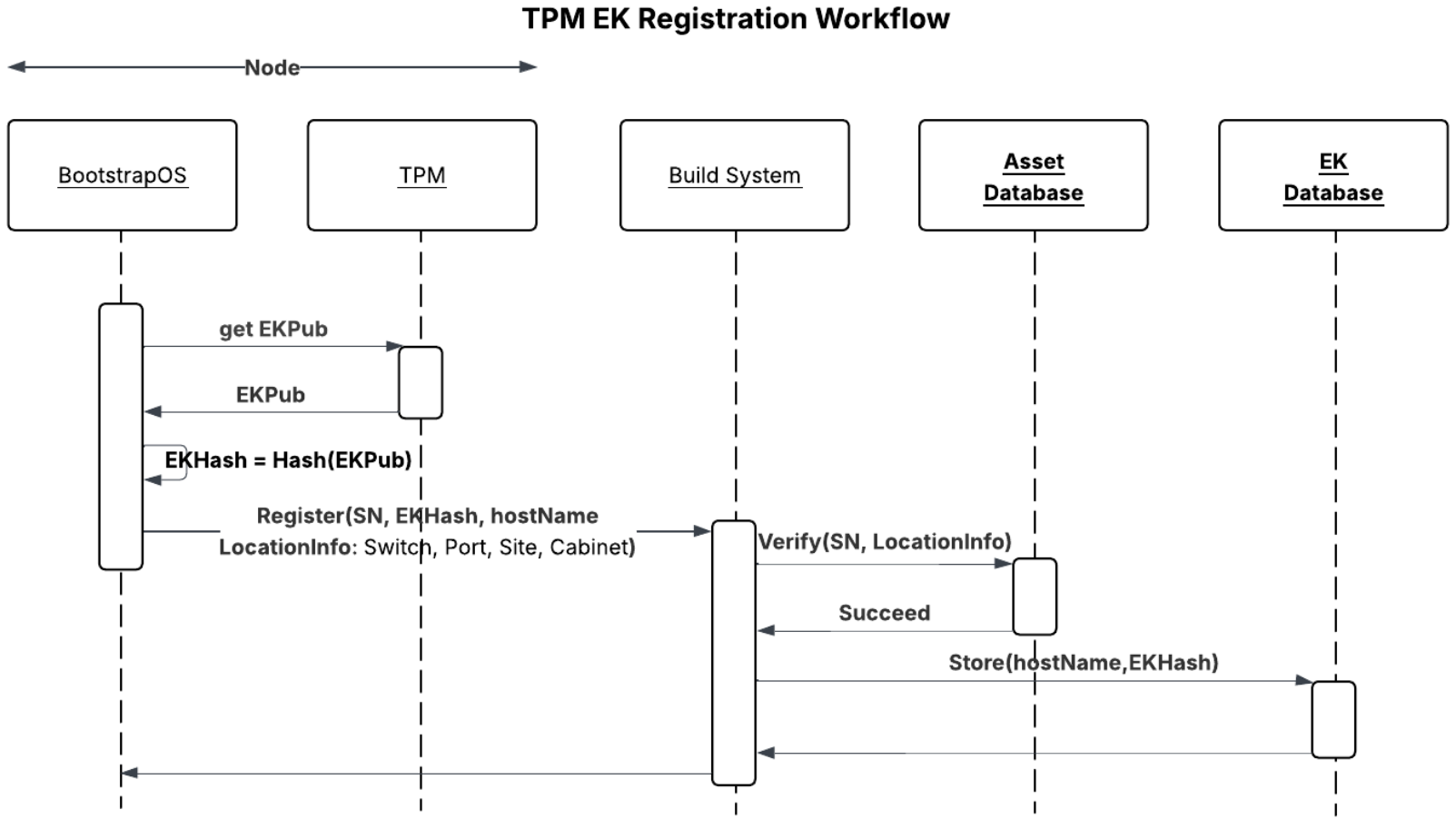
Building Trust of Agent Node: Prior to TPM-based Node Attestation

Data Center Provisioning

- **Record Device Physical Location**
  Store switch/port/site/cabinet in **Asset DB**
  Complete before first power-on
  Tampering requires physical access

TPM EK Registration

- **Identify Managed Node**
  According to recorded locations
- **Register TPM EK pub key**
  Only for managed nodes in **EK DB**

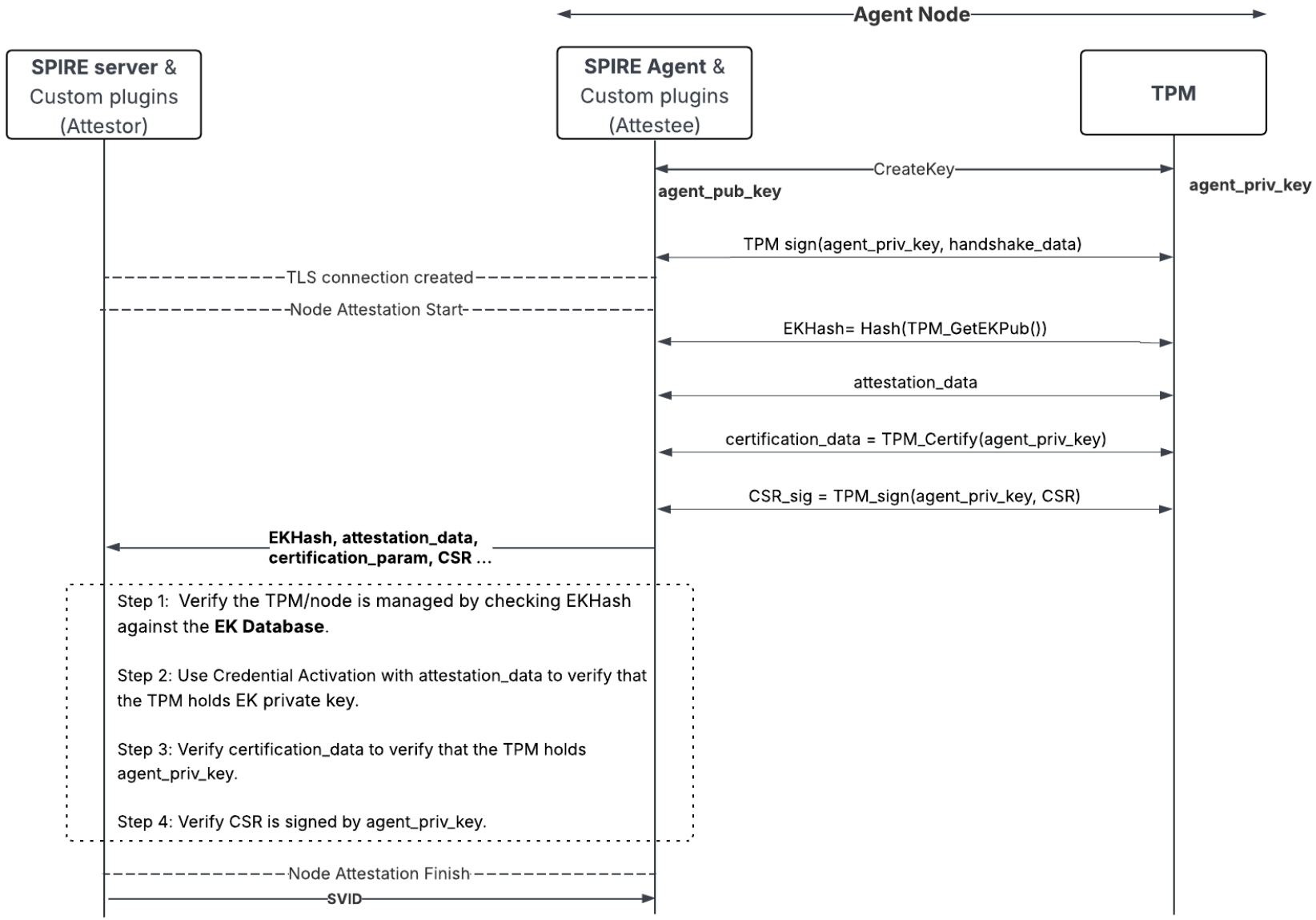


Building Trust of Agent Node: TPM-based Node Attestation

1. EK DB → Managed TPM/Node
2. EK DB → EK private key in TPM (Authenticity)
3. EK private key in TPM → SPIRE Agent private key in TPM
4. CSR_sig → Certificate bound to SPIRE Agent's private key

Future Work

- Authenticity + Ownership → Integrity
- Integrate Measured Boot for Node Bootstrap Integrity



Identity Issuance: Workload Registration and Attestation

Workload Registration

- Define which workloads can receive specific identities in registration entries.
- Our controllers monitor API events and updates entries.

Workload Attestation

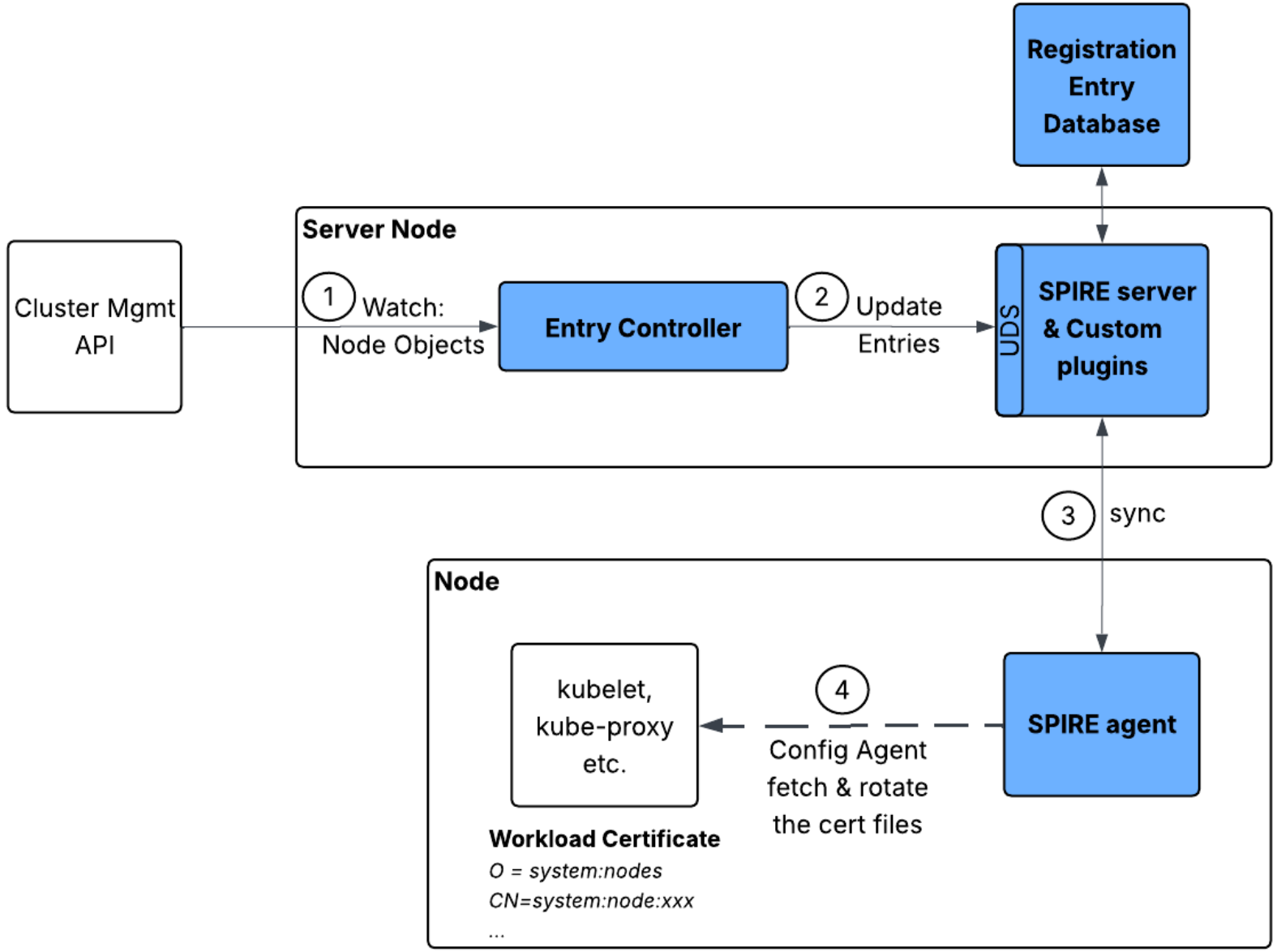
- Collect workload properties and match them against registration entries.
- Uses OSS SPIRE plugins (*systemd*, *unix* etc.).
- Develop call stack attestation for other workloads.

Examples of Workloads

- Kubelet, Kube-proxy
- Fluent-bit
- Node management agent
- Host health monitoring agent etc.

Identity Issuance: Kubelet TLS Bootstrapping Use Case

1. Watch API object events
2. Update entries via server APIs over UDS
3. Periodically sync entries to local cache
4. Attest workloads based on UID and perform delegation



Why SPIRE Agent Isn’t Always a Fit

Ephemeral Workloads with Short-lived Certs

- Overhead of dynamic workload registrations
- No need for security benefits like identity rotation

Constrained Environments

- Cloud-managed env (e.g., Azure) — no agent-based node attestation needed
- Limited or no access to the infrastructure — hard agent management

Platform and Operational Overhead

- Non-Linux workloads (e.g., macOS)
- Agent deployment adds complexity across these platforms

An Agentless Architecture

Simplified Architecture

- No SPIRE agent
- No entry registration workflow

Add Server-Side gRPC Service

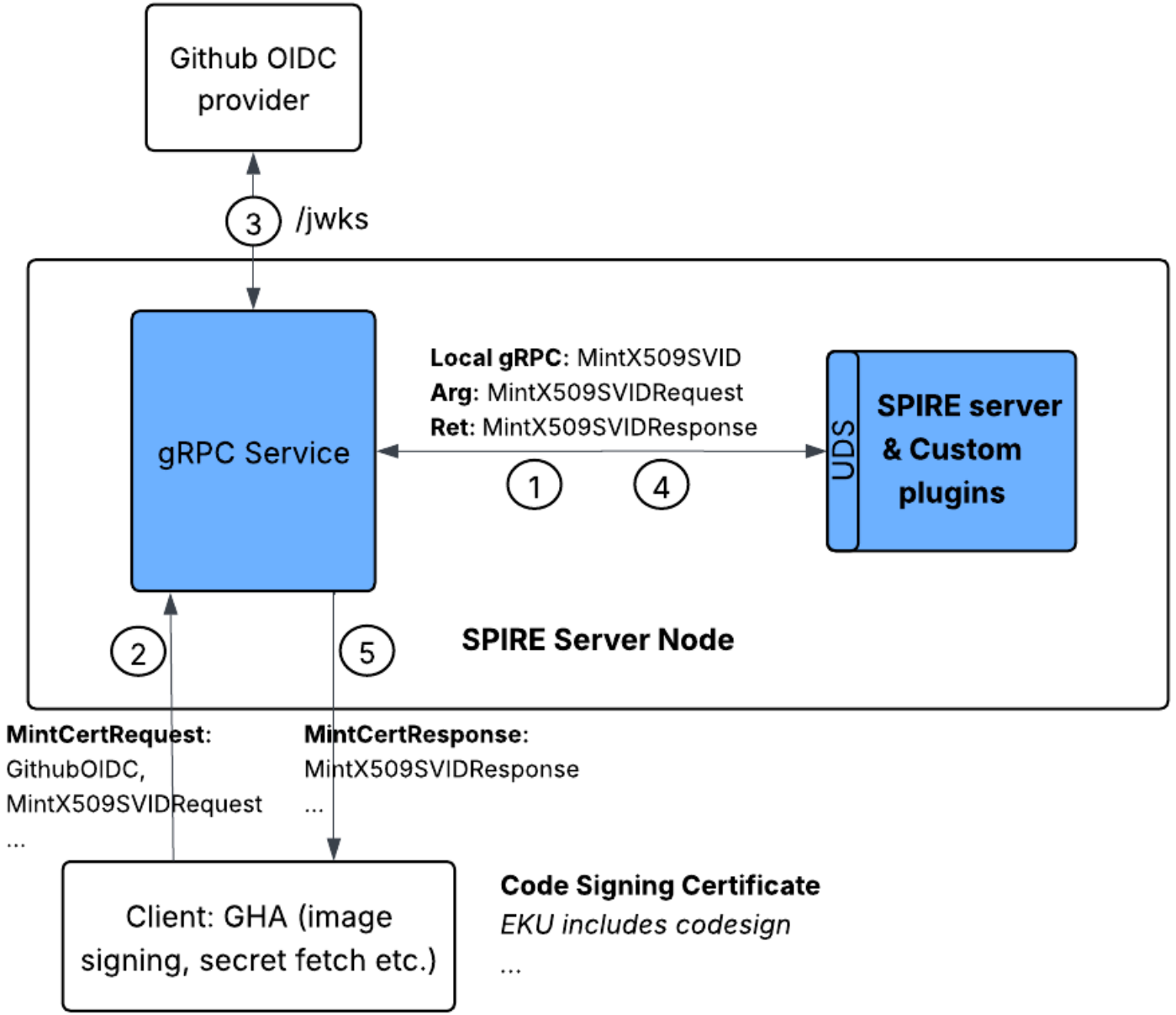
- Handles request authentication
- Interact with SPIRE server APIs via Unix Domain Socket (UDS)

Workloads → gRPC Service Directly

- One-way TLS connection

An Agentless Architecture: GitHub Action Use Case

1. gRPC service obtains its own identity.
2. GHA client retrieves certs for specific uses.
3. OIDC token is verified.
4. GHA client identity is retrieved.
5. Identity is forwarded to the client.



Lessons Learned

Stay Flexible

- Use the plugin system to handle different scenarios
  e.g. TPM & Non-TPM node attestation.
- Leverage the SPIRE Server API to develop custom components.

Issuing Certificates ≠ Done

- Adoption is as challenging as issuing identities.
- Plan library and authorization policy updates early.
- Application adoption is a long-term effort.

Prioritize Observability

- Invest early to support development and adoption.
- Ensure visibility across all plugins and extended components.

Early Alignment

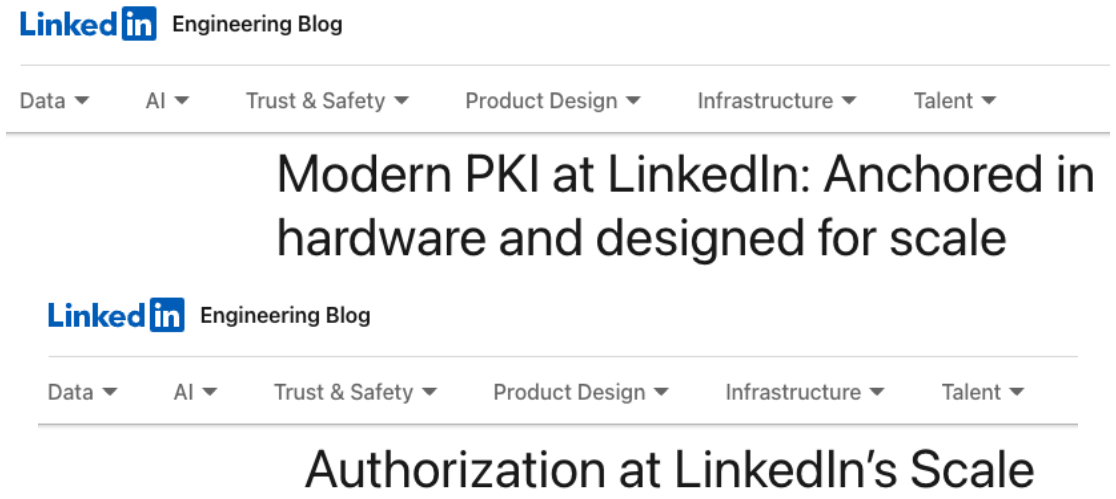
- Integrate with key infrastructure components such as node and cluster bootstrap.
- Success comes from being part of the broader infrastructure.

Thank you!

OPEN SOURCE SecurityCon

We are hiring all levels
in the US
(Bay Area/Seattle)

Feedback



enable kubelet server to dynamically load tls certificate files #124574

Merged k8s-ci-robot merged 3 commits into kubernetes:master from zhangweikop:master on Jun 6, 2024

Change return type of keymanager.base Generator interface to crypto.Signer #5843

Merged amartinezfayo merged 4 commits into spiffe:main from jzeng4:jzeng4/generator on Feb 16

Acknowledgements

- This security infrastructure foundational work would not have been possible without the tireless and dedicated work of our team members: Yogesh Patil, Richard Tosirisuk, Rahul Godha, Di Jin and the entire LinkedIn PKI team.

- Thanks to our Hardware Engineering team, Production Systems Software Engineering team, and Compute and Kubernetes Infra team (Ahmet Alp Balkan, Ronak Nathani) for being able partners in driving this cross organizational initiative.

- We would like to thank the contributions from all the individuals towards this specific initiative including our LinkedIn alumni, Ruihan Guan, Sindhu Ravichandran, Alex Tcherniakovski, Priya Ayyagari, and Sergei Rousakov.

- Special thanks to those who gave their time to review this slides: Amin Gomos and Phincy Pious and Evan Gilman.

- Finally, we would like to thank our engineering leaders: Min-Hank Ho, Omkhar Arasaratnam, Lea Kissner and Sabry Tozin for their strong support throughout this journey.