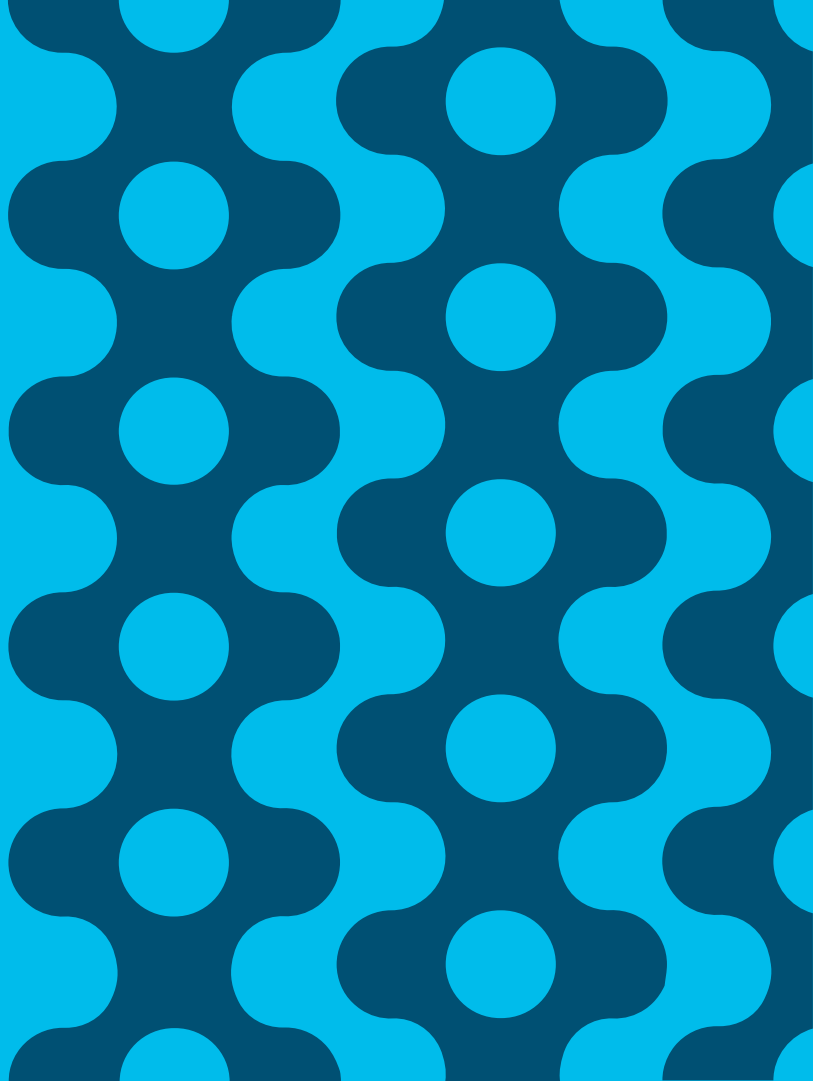
Security in the World of Service Meshes

John A. Joyce

Principal Engineer – Cisco Systems

November 2020



Agenda

- Purpose
- Mesh Security 101 Definitions & Concepts
  - Identity
  - Authn
  - Transport Auth vs. Origin Auth
  - TLS (Client side vs. Server side) vs. Mutual TLS
  - Authz
- System components and options
- System architectural patterns
- Demo

Purpose

Abstract/Purpose

This talk will provide a comprehensive picture of security within the world of service meshes. We will start with an introduction of the key security concepts and describe the key system components that implement those concepts. This will include references to Open Source offerings which can provide these components. The talk will progress to architectural patterns which will showcase the security capabilities available for production use. Finally, the talk includes a demo combining several CNCF projects for a scenario which can be replicated by attendees. We will touch on the following CNCF projects in this talk: Envoy, Linkerd, SPIFFE, SPIRE, and Network Service Mesh.

Mesh Security 101
Definitions & Concepts

General Definitions

- **Identity** – A characteristic or a set of data to uniquely identify a user, process or device. Sometime expressed with an Identity document (e.g. SVID)
- **Authn** – Authentication – The verification of the identity of a user, process, or device.
  - Transport or Peer Authn – Authentication with the immediate peer of the connection even when acting as an intermediary of a flow. Supported by most service meshes.
  - Origin or Request Authn - Authn based on the source and destination of the flow (ie. request) regardless of intermediaries. Not widely supported by service meshes.
- **Authz** – Authorization – The determination of the access levels or privileges that should be granted to an identity.
- **Certificate –** Contains a public key and an identity. Typically X.509 in this space.
- **SVID** – SPIFFE Verifiable Identity Document – An identity document defined in the SPIFFE specification.
- **Transport Layer Security (TLS) –** A cryptographic protocol designed to provide communications security over a computer network.
- **Zero Trust –** Security concepts that assume no peers can be trusted regardless of whether they appear to be within the same network or security domain.

Definitions – TLS flavors

- **Mutual TLS (mTLS) –** A form of TLS where both sides authenticate the identity of the peer. The “gold standard” for service mesh implementations of peer Authn. Not yet supported by all service meshes.
- **Server (side) TLS –** A form of TLS where the server presents a certificate to the client which is used to authenticate the server’s identity. Commonly used in “web surfing” scenarios as the clients are unknown or outside of server’s security domain.
- **Client (side) TLS –** A form of TLS where the client presents a certificate to the upstream server which is used to authenticate the client’s identity. Least commonly used variant.

References

- **Network Service Mesh –** https://networkservicemesh.io/
- **Linkerd –** https://linkerd.io/
- **Istio –** https://istio.io/
- **SPIFFE –** https://spiffe.io/
- **SPIRE** - https://github.com/spiffe/spire
- **X.509 -** https://www.itu.int/rec/T-REC-X.509
- **Transport Layer Security (TLS) –** https://tools.ietf.org/html/rfc8446

Note: Much of the content in this talk was directly or indirectly derived from the above sources.

Establishing Identity

Service Identities – The starting point

- In a service mesh world, establishing the identity of the workload providing a service is critical. Examples:
  - **Kubernetes**: Kubernetes service account
  - **GKE/GCE**: may use GCP service account
  - **GCP**: GCP service account
  - **AWS**: AWS IAM user/role account
  - **On-premises (non-Kubernetes)**: user account, custom service account, service name, Istio service account, or GCP service account. The custom service account refers to the existing service account just like the identities that the customer's Identity Directory manages.

https://istio.io/docs/concepts/security/#istio-identity

The new kid on the block for establishing identity is SPIFFE (specification) and SPIRE (implementation) – more on that below

Conversion of that identity into a certificate

- A private key within the workload pod is generated and made available to the proxy.
- A certificate signing request is sent to the control plane.
- The control plane signs either with a self-signed root certificate or an external source (e.g. vault).
- The control plane provides the proxy a certificate scoped to the identity of the POD (e.g. K8s service-account).
- Control plane will manage rotation.
- Some meshes may use certificates mounted to the proxies via other means.

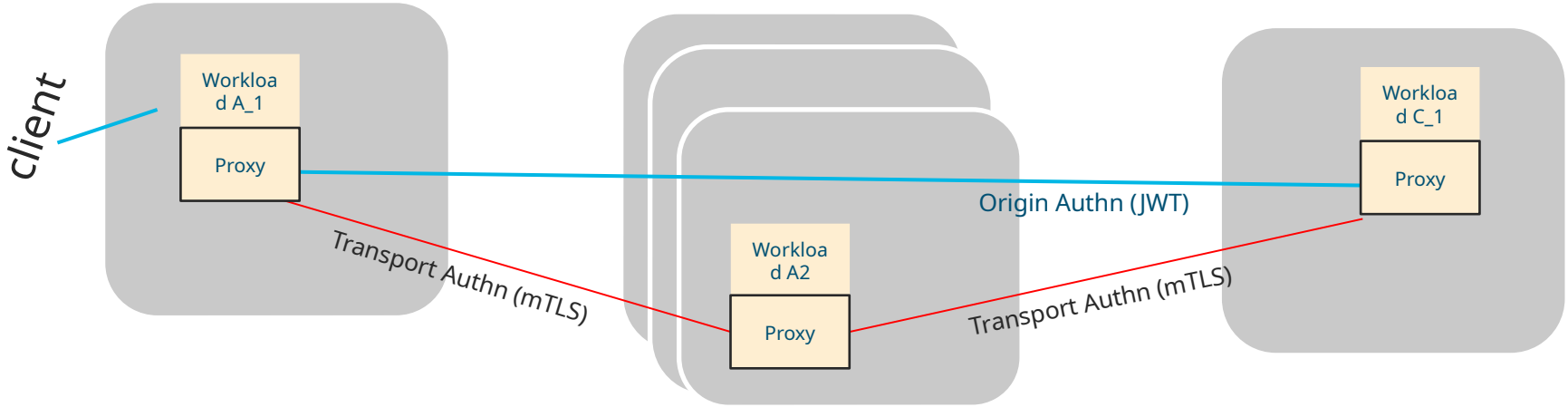
Authn - Authentication

Authentication Layers

Peer or Transport Authn—tied explicitly to the network layer parameters/proto

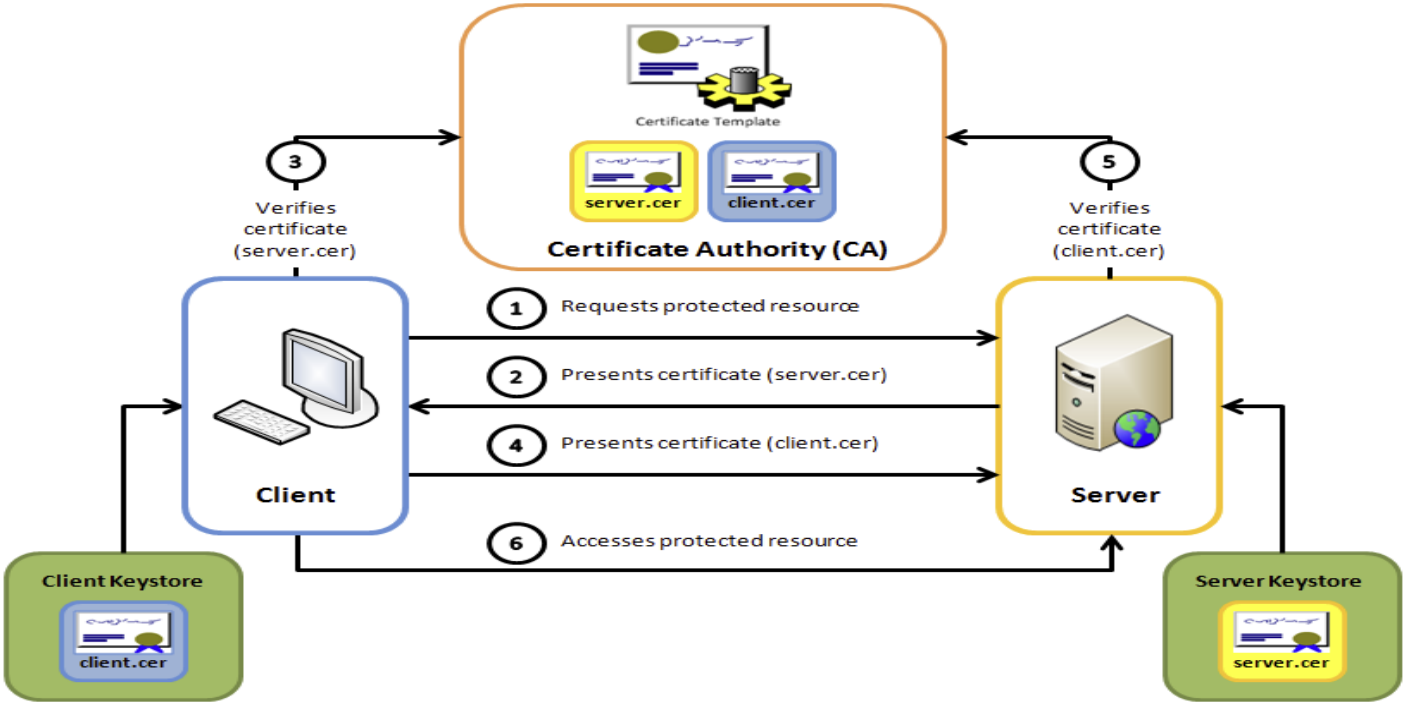
e.g. mTLS handshake certificate content

Request or Origin Authn—ability to correlate the origin of the flow (ie. request) regardless of intermediaries



All meshes offer some form of Peer/Transport Authentication. Some meshes additionally support Request/Origin Auth in the proxy. Request/Origin auth can still be done by the workload without participation by the mesh or proxy.

Authn sequence for mTLS



Mutual SSL authentication / Certificate based mutual authentication

https://www.codeproject.com/Articles/326574/An-Introduction-to-Mutual-SSL-Authentication

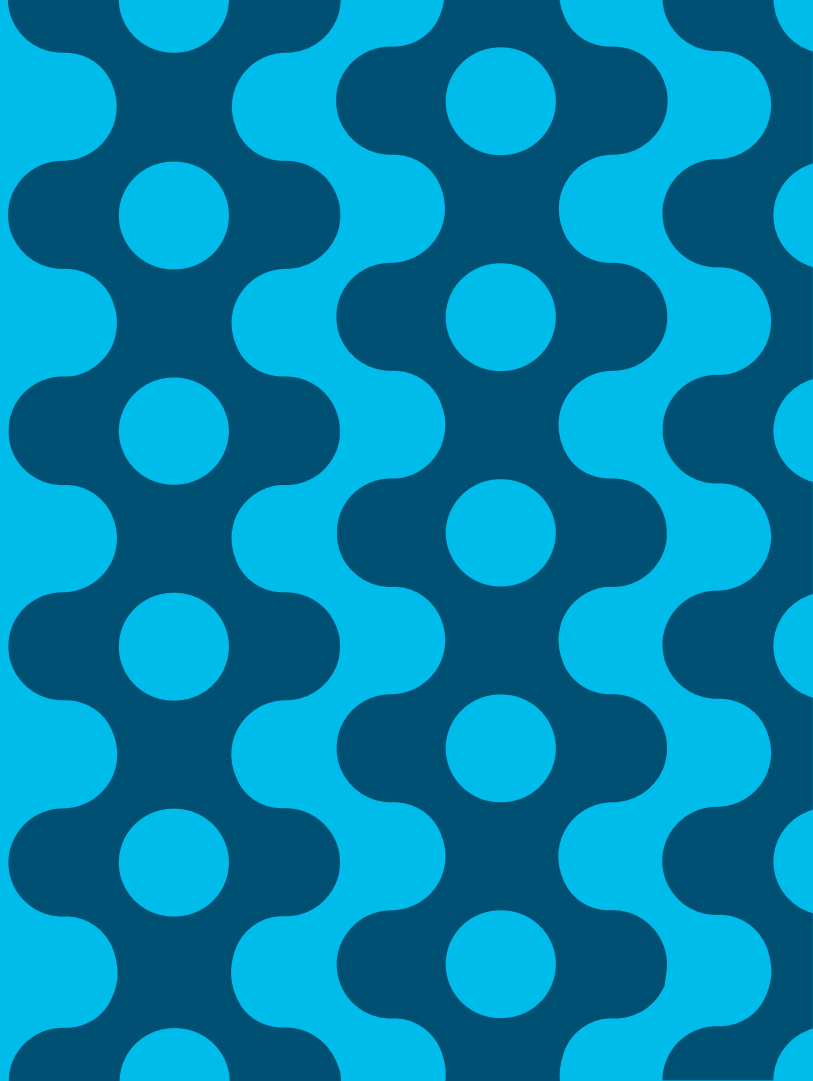
Single-sided TLS vs. mTLS

- Although mutual TLS is the gold standard other options must be supported for several reasons:
  - It is necessary to accept traffic into the mesh from clients that can’t be authenticated (e.g. clients not within the same administrative or security domain).
  - Similarly, it is necessary to send to clients or servers that can’t be authenticated.
  - Multiple meshes without a common identity framework may need to share traffic.
- For these cases one sided TLS authentication is useful.
  - Server side is available as an option in most service meshes. The server side presents its identity via a certificate that is tracible back to a well know issuer.
  - Client side is also possible but not widely supported. The client side presents its identity, and the server is configured with a means to authenticate the client. Often this requires some amount of direct certificate population.
- Clear text is a final fall back – but diminishes one of the key benefits of using a service mesh in the first place.

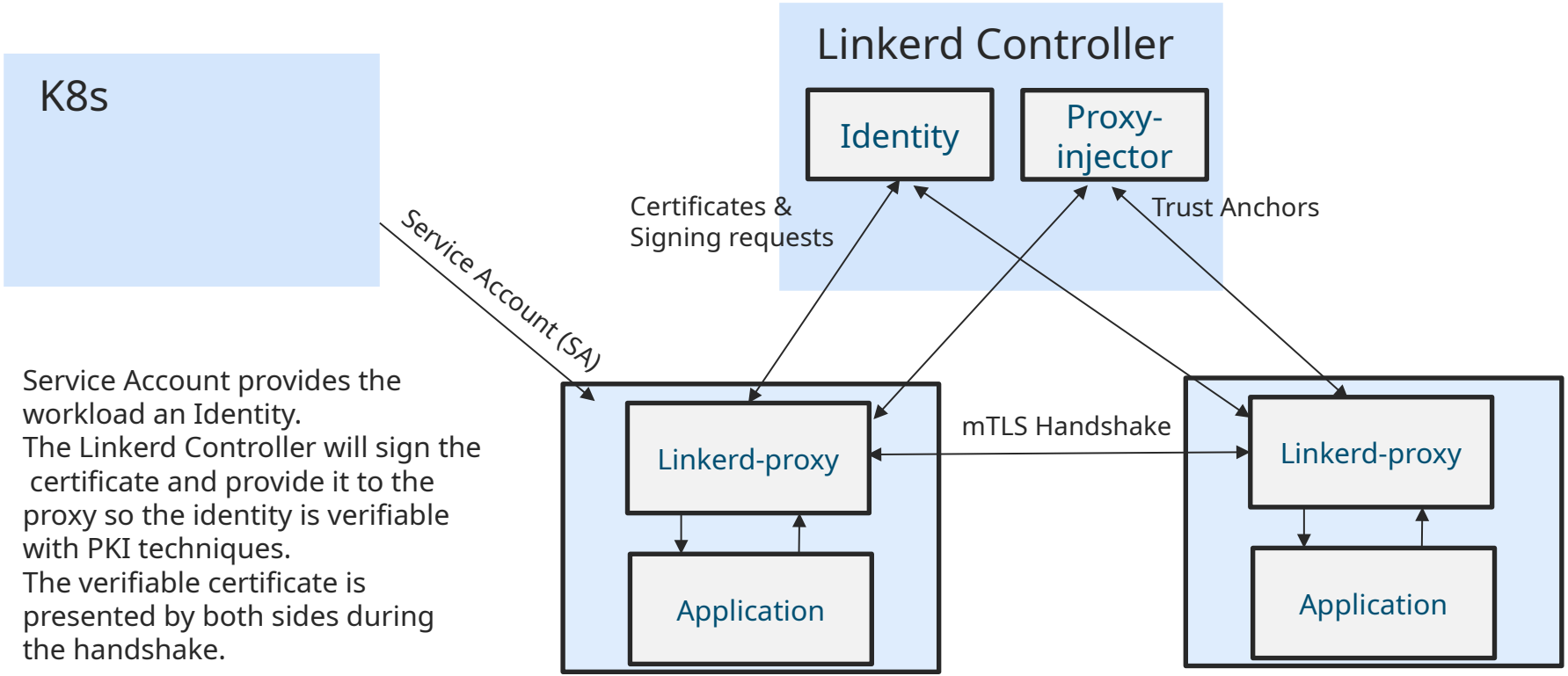
Authorization

- Once a peer has been authenticated, the next step is to decide if it is authorized.
- Authorization determines whether a sender and receiver have the necessary permissions or privileges to talk to one another.
- The authorization is often expressed in a policy and sometimes referred to as access control.
- To be effective there must be some means of coupling the authorization with the identities determined with Authn.
- Although common in service meshes, it is not universally supported in all service meshes.
- The authorization may be implemented in either the client-side proxy or server-side proxy or both.
- NOTE – Authorization here only covers Service Mesh authorization. The application may have its own authorization functionality independent of the service mesh.

Components

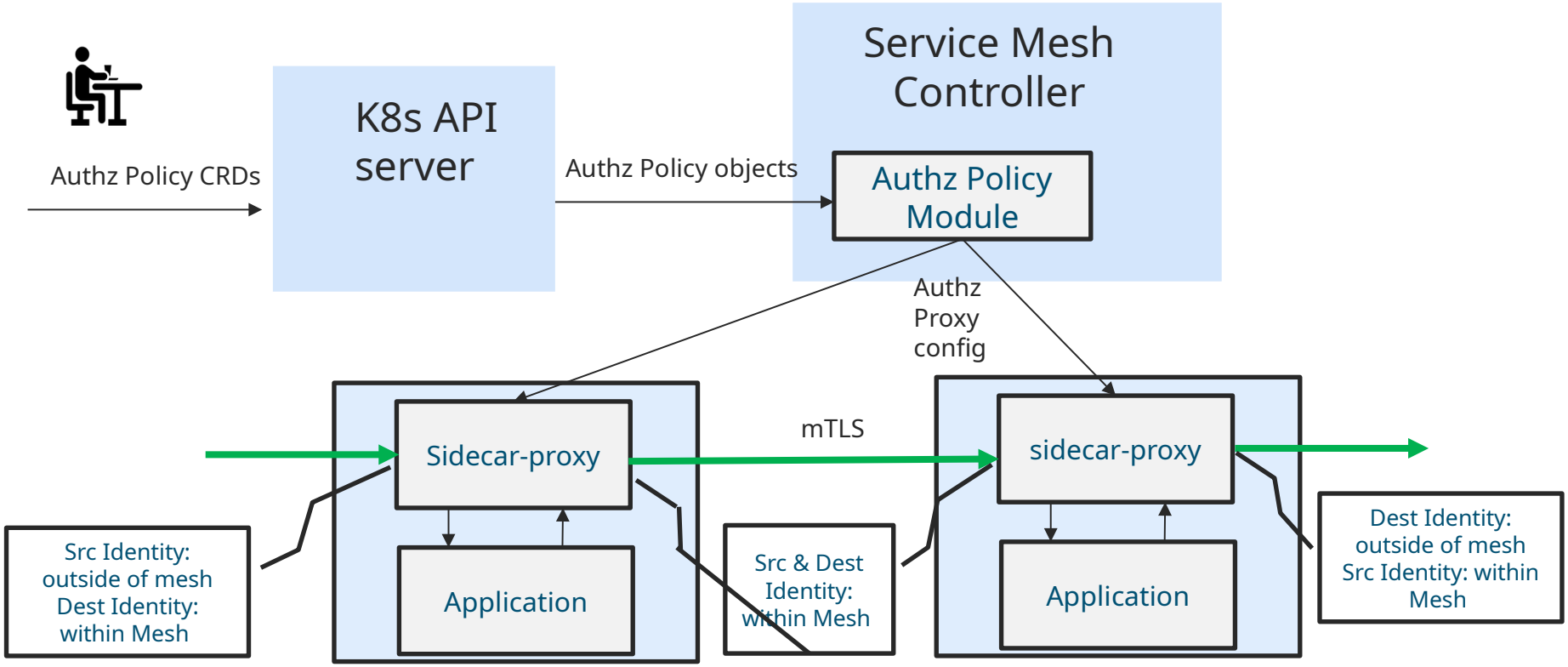


Authn Architecture (with Linkerd)



- Service Account provides the workload an Identity.
- The Linkerd Controller will sign the certificate and provide it to the proxy so the identity is verifiable with PKI techniques.
- The verifiable certificate is presented by both sides during the handshake.

Authorization Architecture



Authorization Example (From SMI)

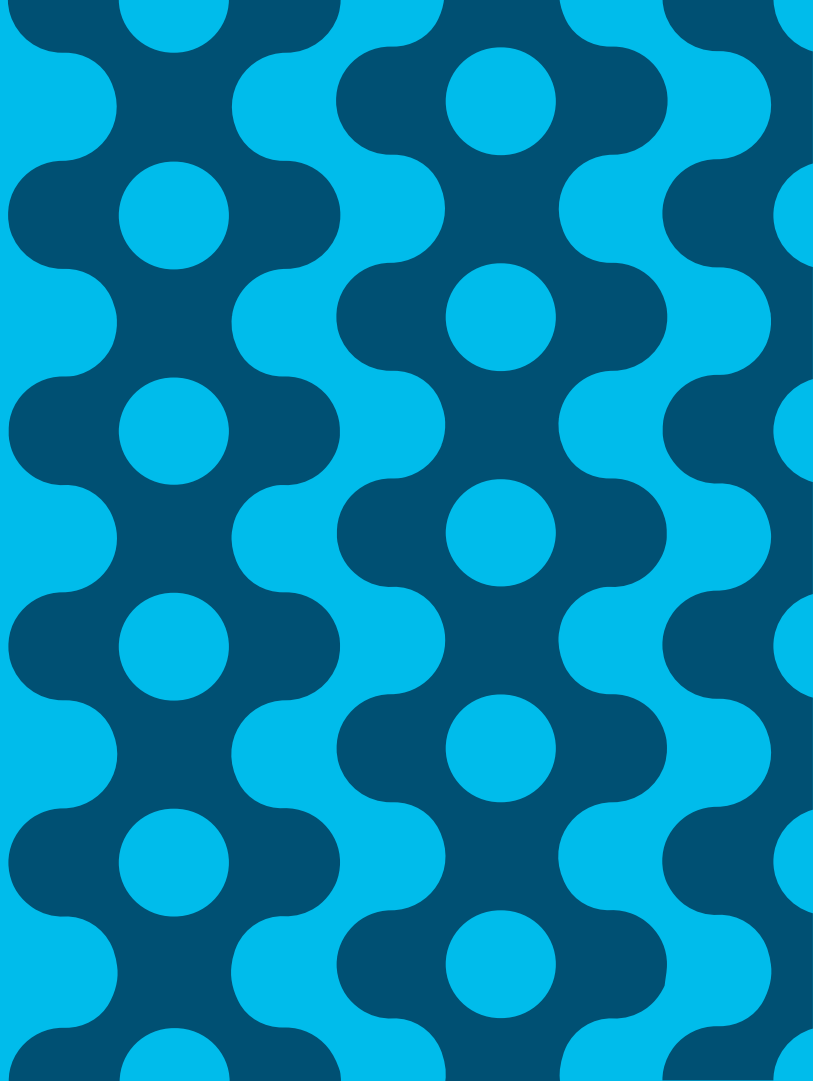
```
kind: TrafficTarget
metadata:
  name: path-specific
  namespace: default
spec:
  destination:
    kind: ServiceAccount
    name: service-a
    namespace: default
    port: 8080
  rules:
  - kind: HTTPRouteGroup
    name: the-routes
    matches:
    - metrics
  sources:
  - kind: ServiceAccount
    name: prometheus
    namespace: default
```

Destination identity specified with service account

Match criteria

Source identity specified with service account

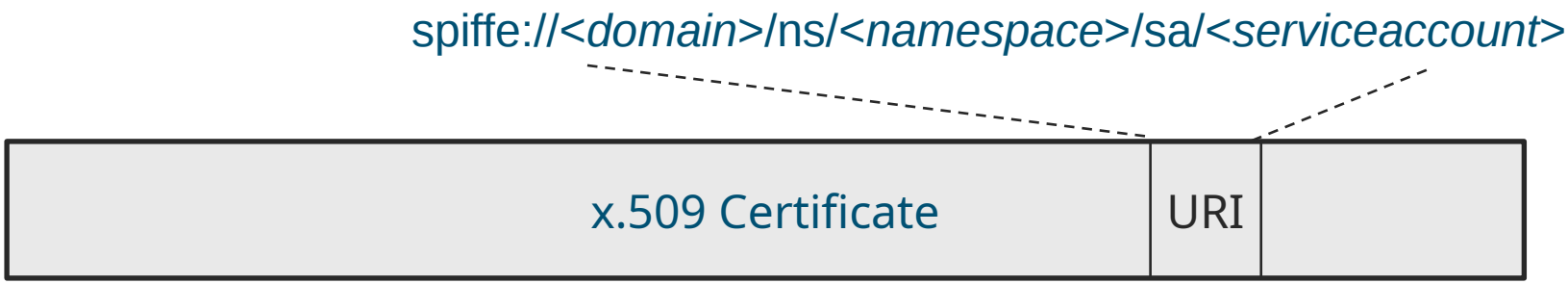
SPIFFE/SPIRE:
A new way to establish Identity



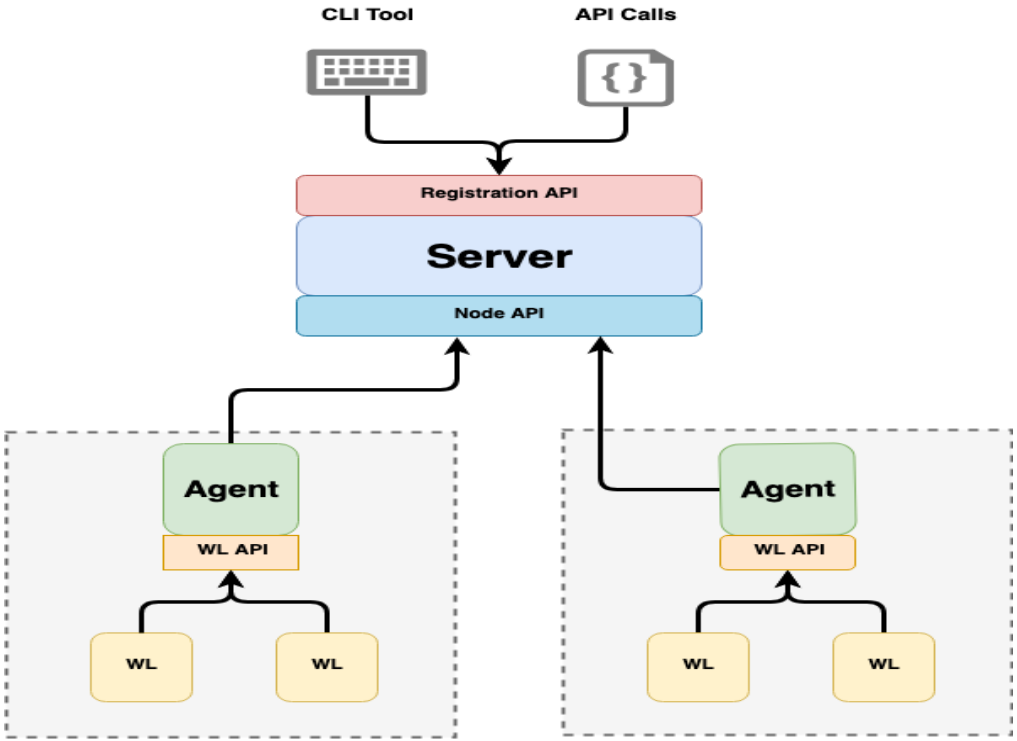
SPIFFE & SPIRE

- SPIFFE is the specification.
- SPIRE is an implementation of the specification.
- SPIFFE is a set of open-source specifications for a framework capable of bootstrapping and issuing identity to services across heterogeneous environments and organizational boundaries. (https://spiffe.io/docs/latest/spiffe/overview/)
- Provides an identity framework suitable for today's cloud native applications.
- Service meshes are moving toward supporting SPIFFE. The extent of support varies based on service mesh.
- As an example, Network Service Mesh has adopted SPIFFE and uses SPIRE to provide the implementation.
- Kuma, Consul Connect and Istio all support parts of the SPIFFE specification.

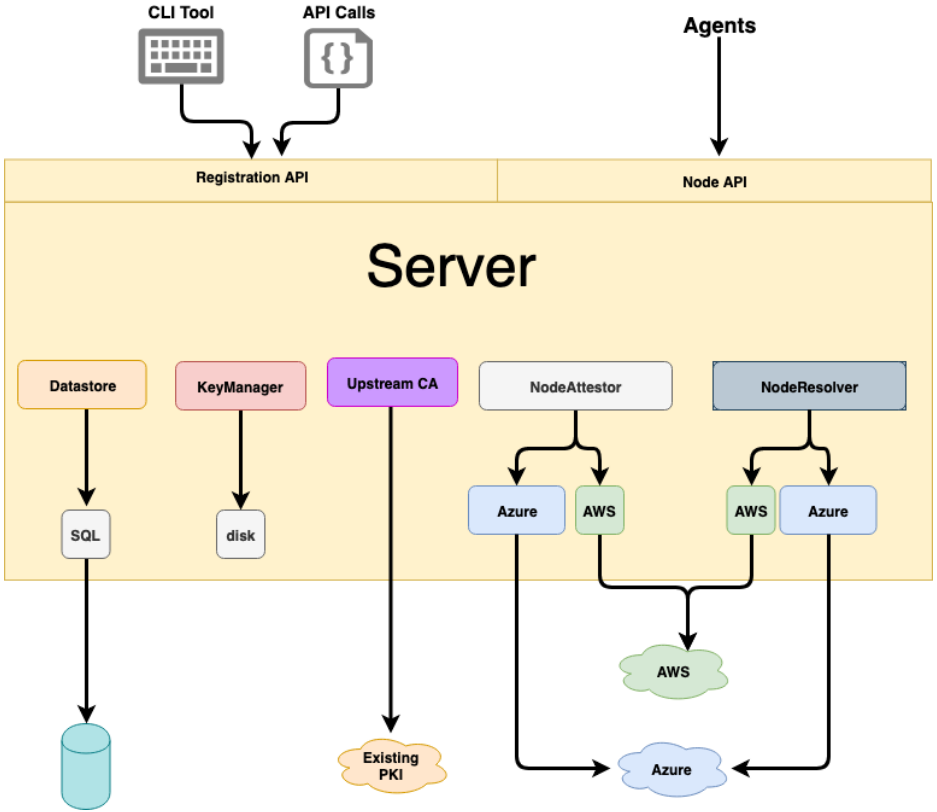
SVID as defined by SPIFFE



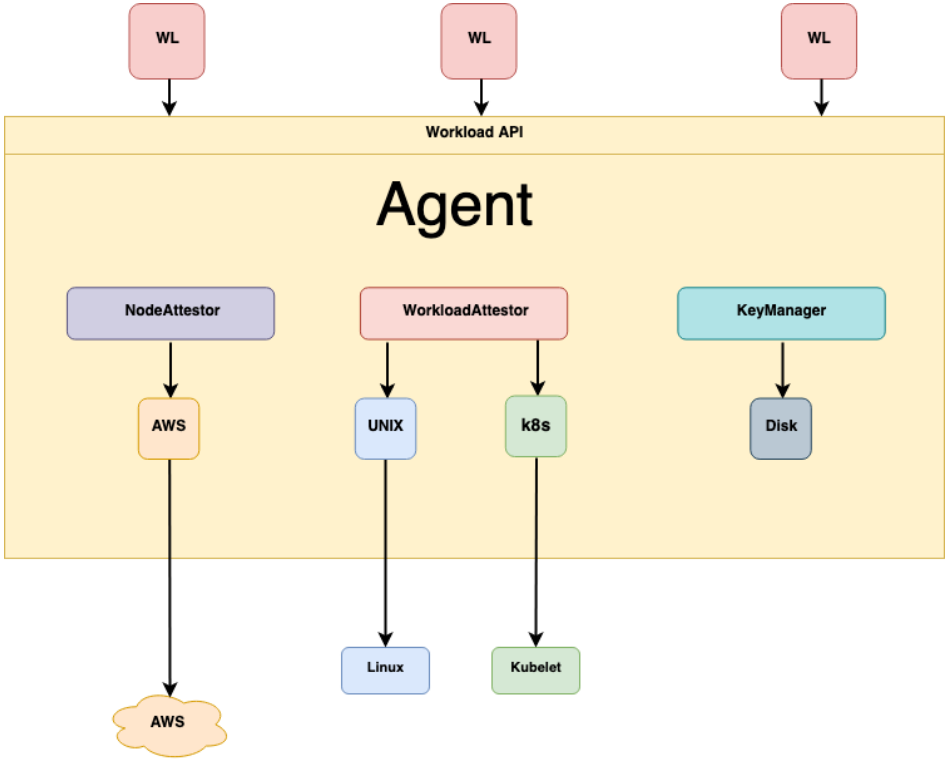
Spire server and Agent



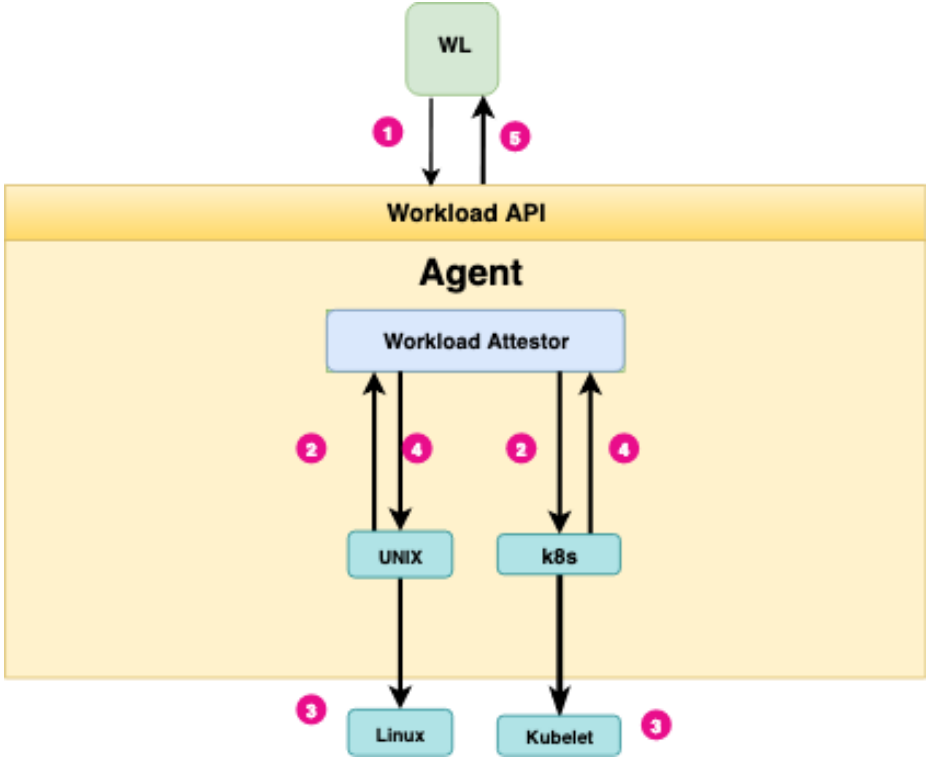
SPIRE Server Design Details



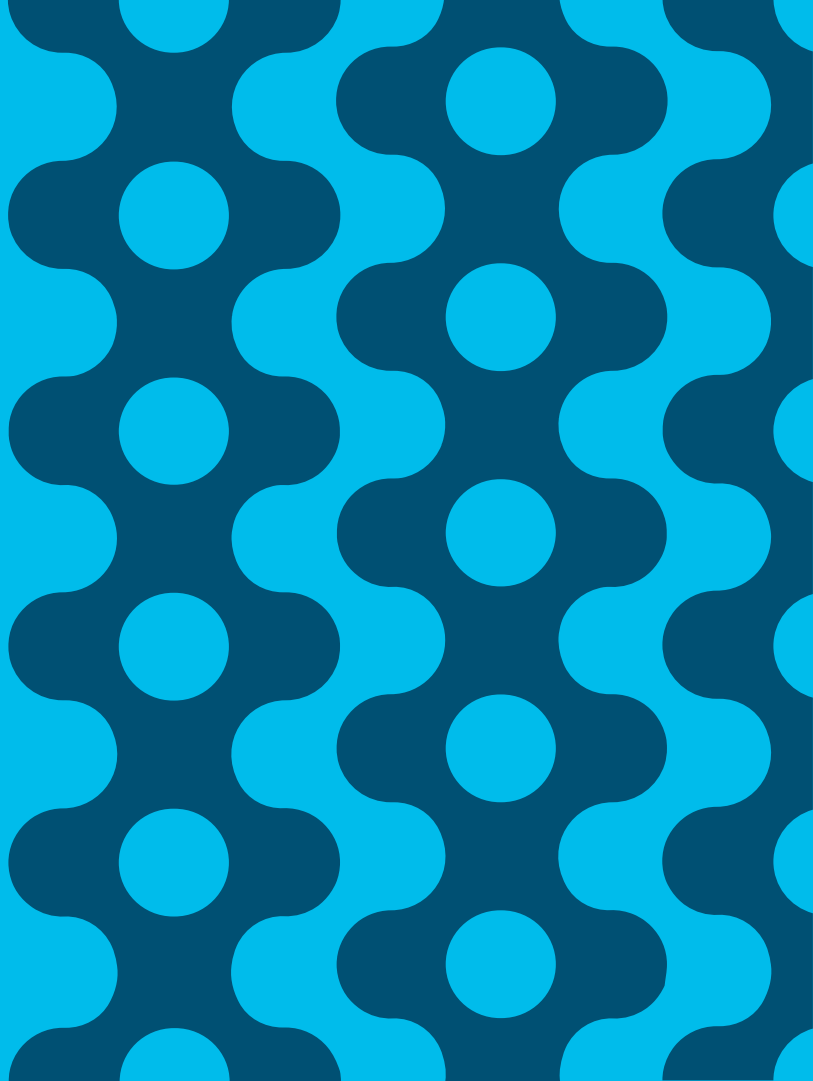
SPIRE Node Agent Design Details



Workload Attestation



Service Mesh Multi-cluster Identity and Trust Models



Shared Root Trust

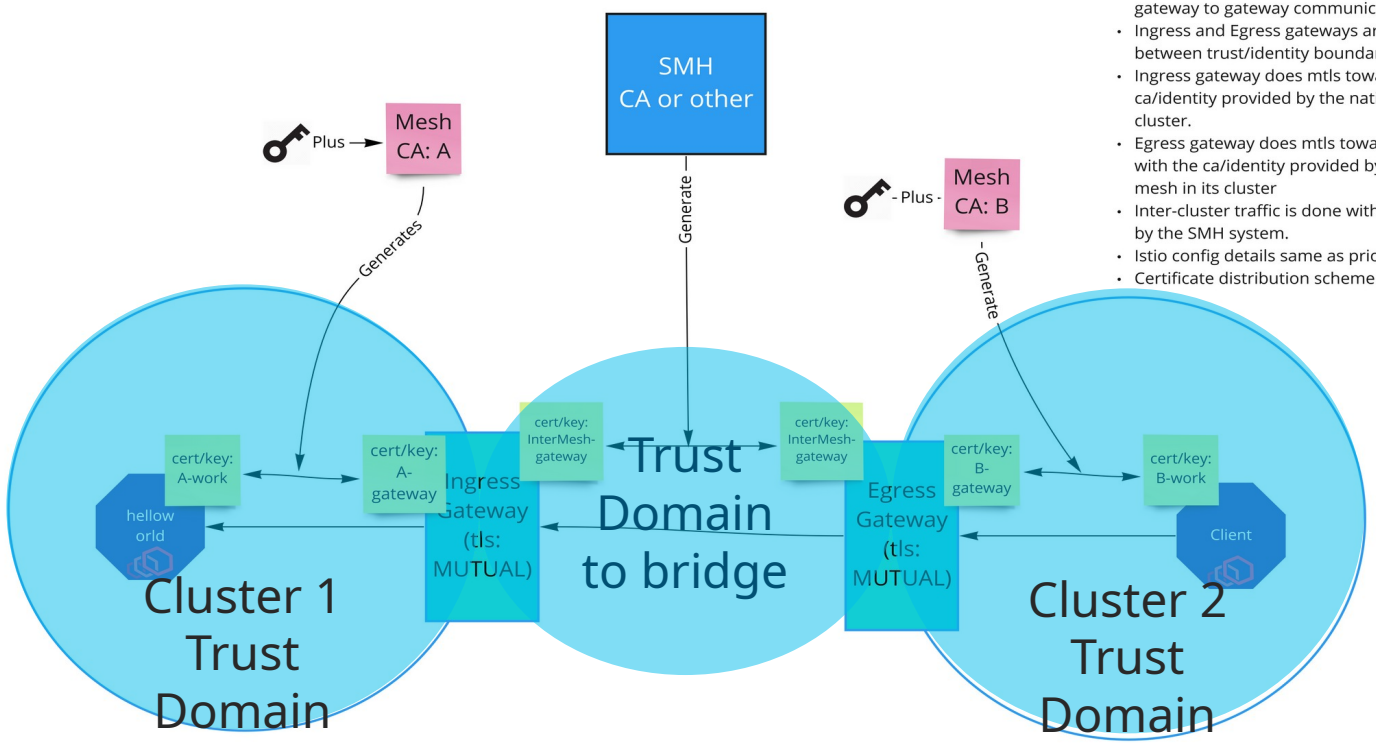
- The most common identity model used for Inter-cluster traffic in service mesh offerings when mTLS is supported between clusters.

- Nearly all service mesh implementations provide examples of how-to config this model.
- Relies on all certificates being traceable back to a common root so all parties can authenticate one another.

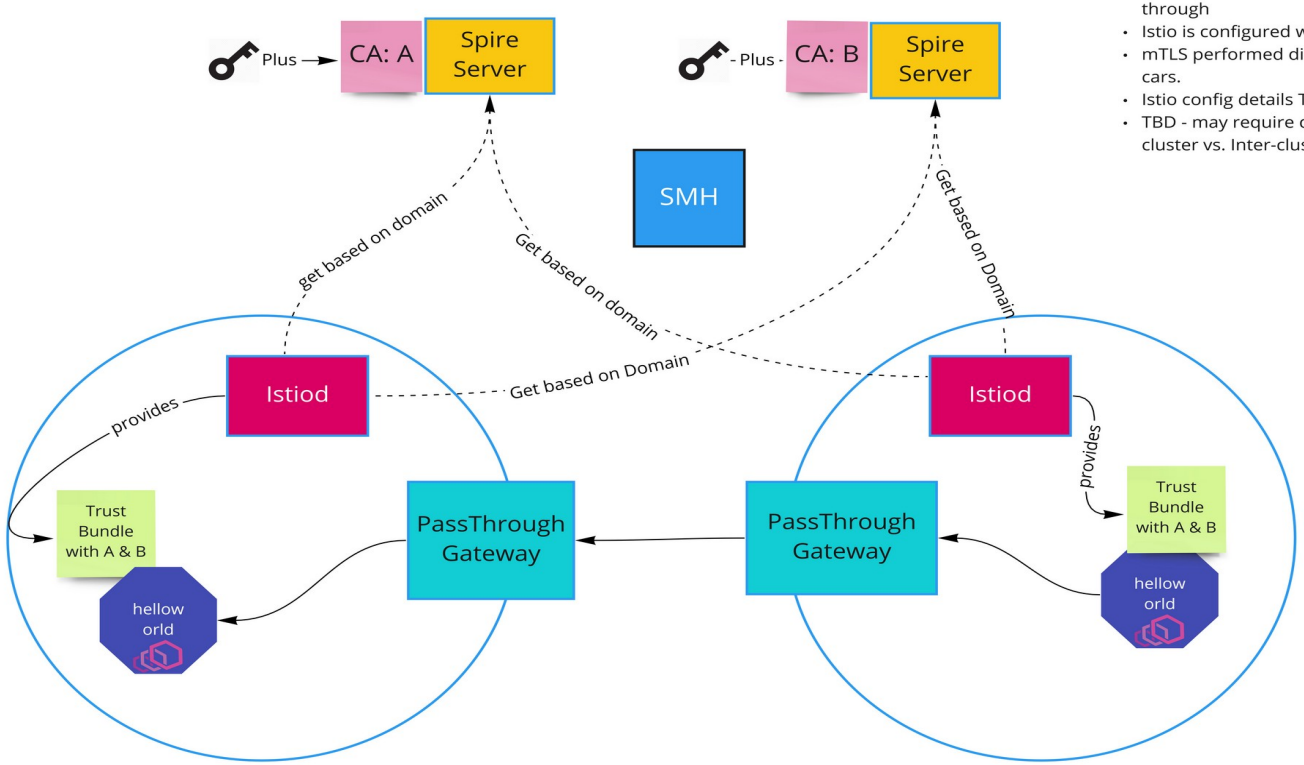
Bridged trust via an intermediary

Via ServiceMeshHub and Istio

- Each cluster has its own CA system
- An additional CA or trust domain is created for gateway to gateway communication.
- Ingress and Egress gateways are the transition point between trust/identity boundaries
- Ingress gateway does mtls toward upstream with the ca/identity provided by the native service mesh in its cluster.
- Egress gateway does mtls toward client/downstream with the ca/identity provided by the natvie service mesh in its cluster
- Inter-cluster traffic is done with CA/identity provided by the SMH system.
- Istio config details same as prior
- Certificate distribution scheme is TBD



Federated Identity – with SPIFFE and SPIRE



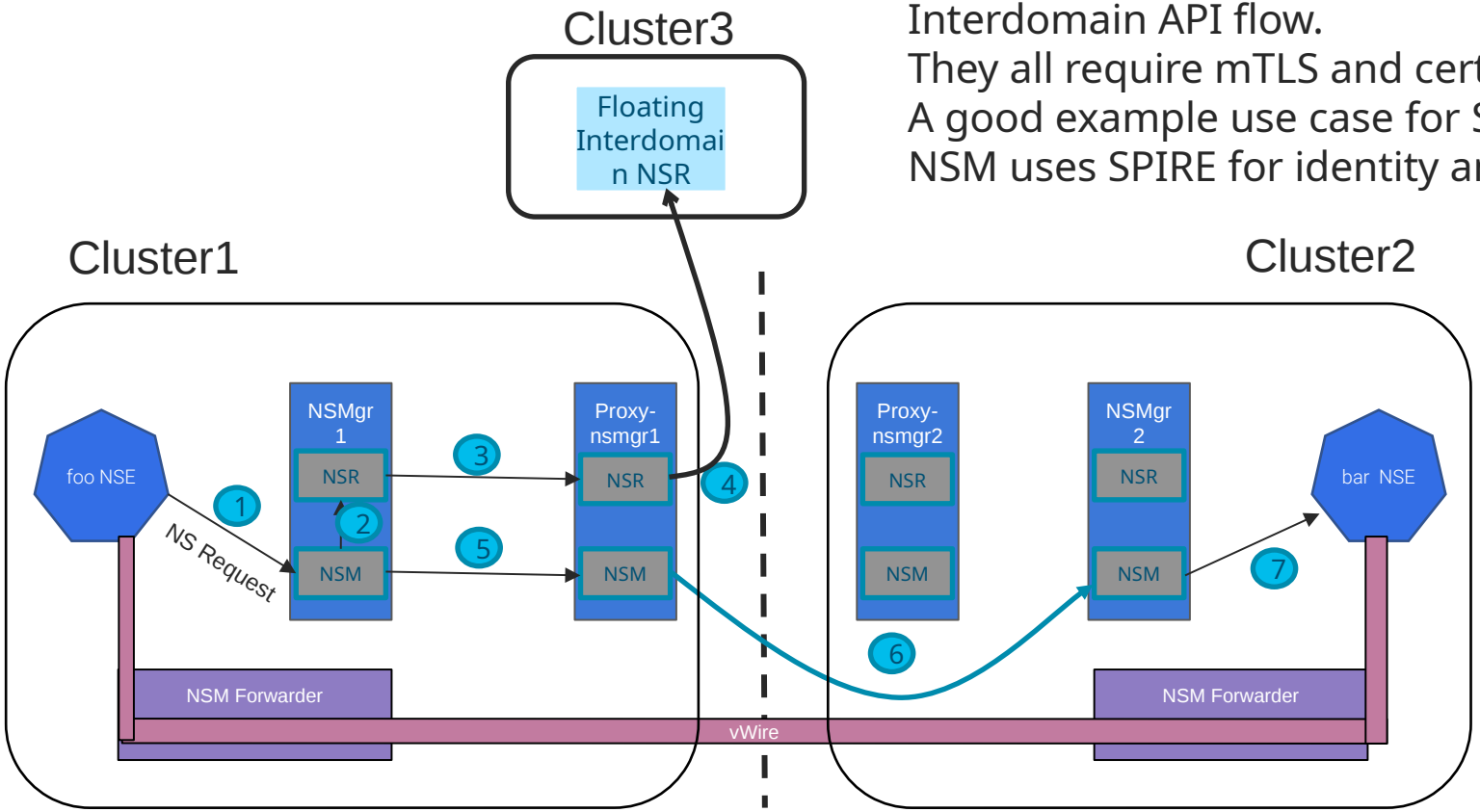
- Each cluster has its own CA system
- Ingress and Egress gateways are set up for pass-through
- Istio is configured with a Spiffe endpoint per domain.
- mTLS performed directly in client and upstream side cars.
- Istio config details TBD
- TBD - may require different request path for in cluster vs. Inter-cluster requests.

NSM’s use of SPIFFE/SPIRE

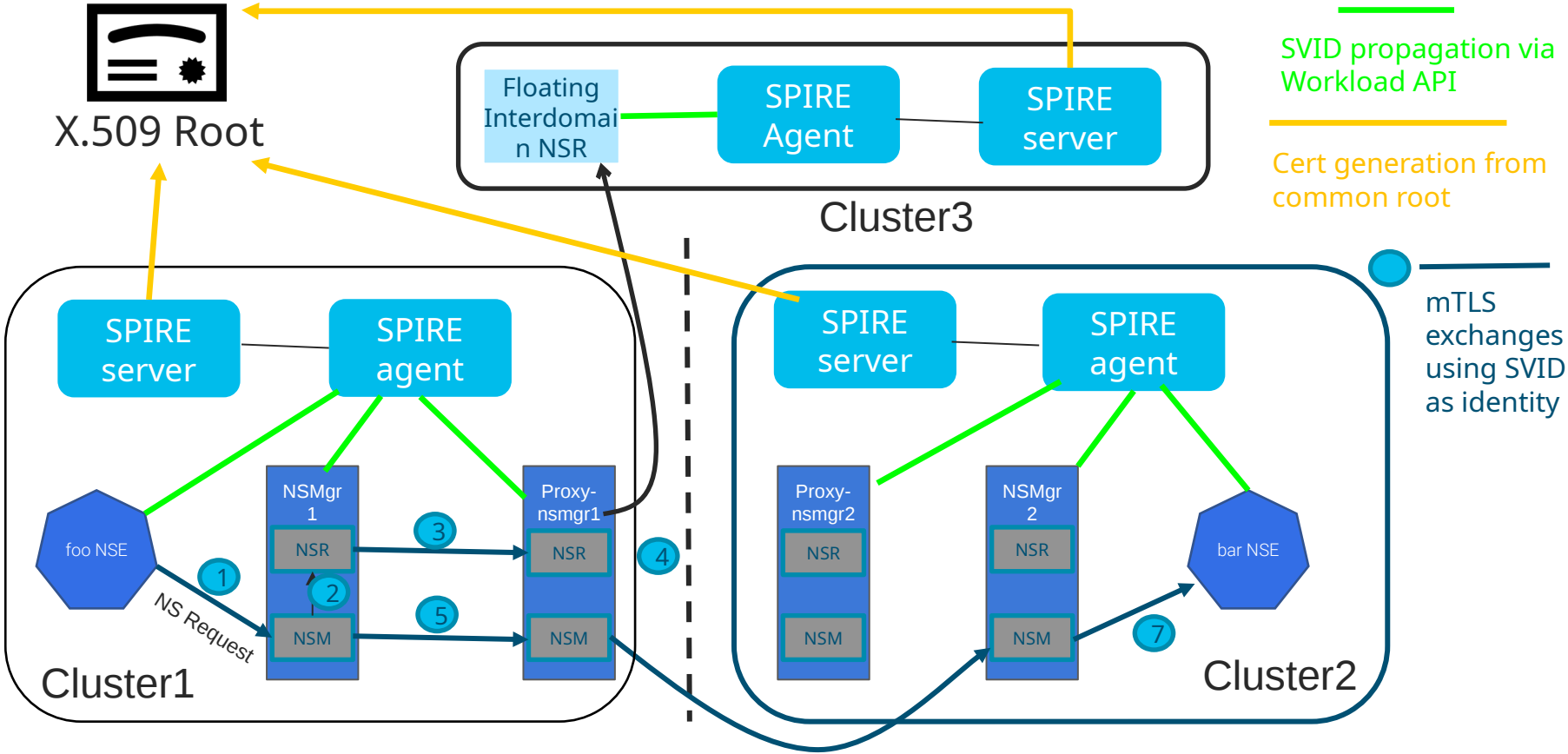
- The Network Service Mesh project use SPIRE servers to provide identity within its implementation.
- It is used to secure API communication between all clients and servers of the API.
- For those familiar with NSM these include NSCs, NSEs, NSRs, NSMs, etc.
- In the floating Interdomain use case the API communication crosses multiple cluster boundaries and potentially multiple security domains.
- As such, it is a great use case to display SPIFFE’s and SPIRE’s capabilities in a multi-cluster environment.

NSM Interdomain Flow

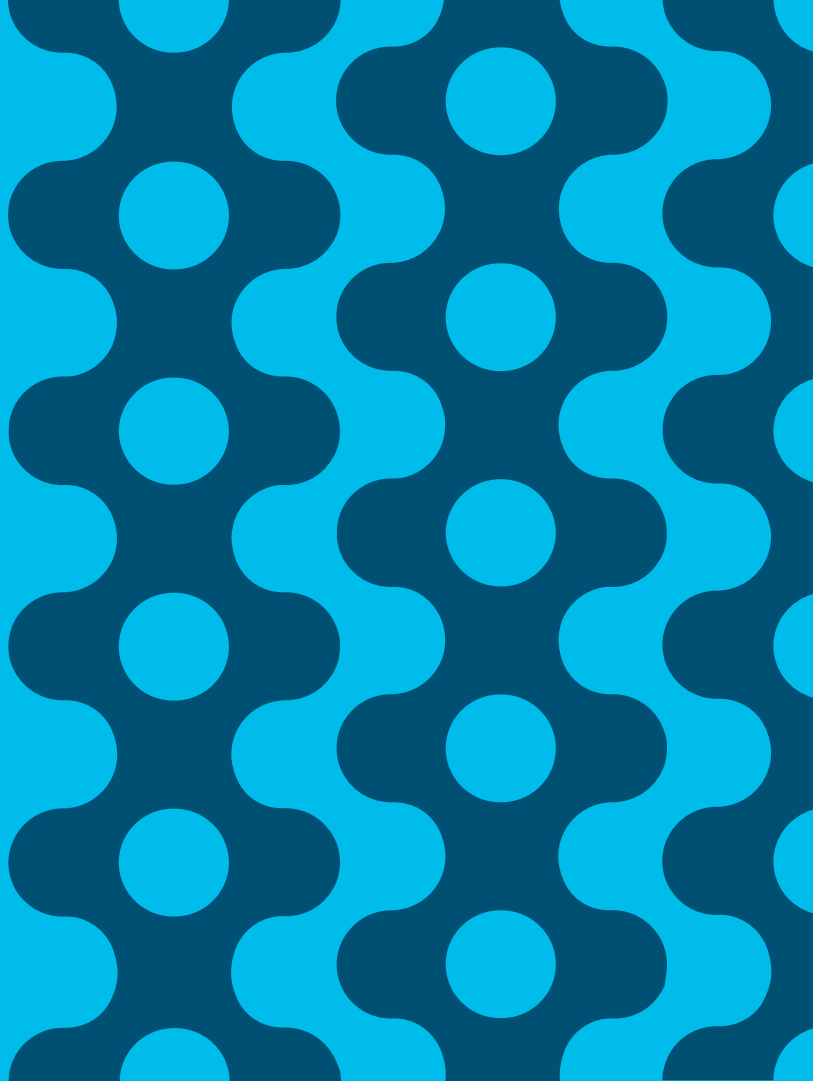
The flow below exemplifies an NSM floating Interdomain API flow.
They all require mTLS and certifiable identity.
A good example use case for SPIFFE/SPIRE.
NSM uses SPIRE for identity and attestation



NSM SPIRE Agent and Server topology



Demo time
NSM with Spiffe/Spire



Acknowledgments

Numerous people helped with this content

- Tim Swanson
- Cosmin-Petru Nechifor
- Mihai Matache
- Pavan Sudheendra
- Shannon McFarland

Q&A