Collibra

IstioSummit 2021

Know Your Peers

Knok, knok, who's there?

Alex Van Boxel

Slide disclaimer: They contain a lot of text. Keywords are highlighted.

This is against all presentation rules… but hey, it makes the slides useful afterwards.

What's the goal of this talk?

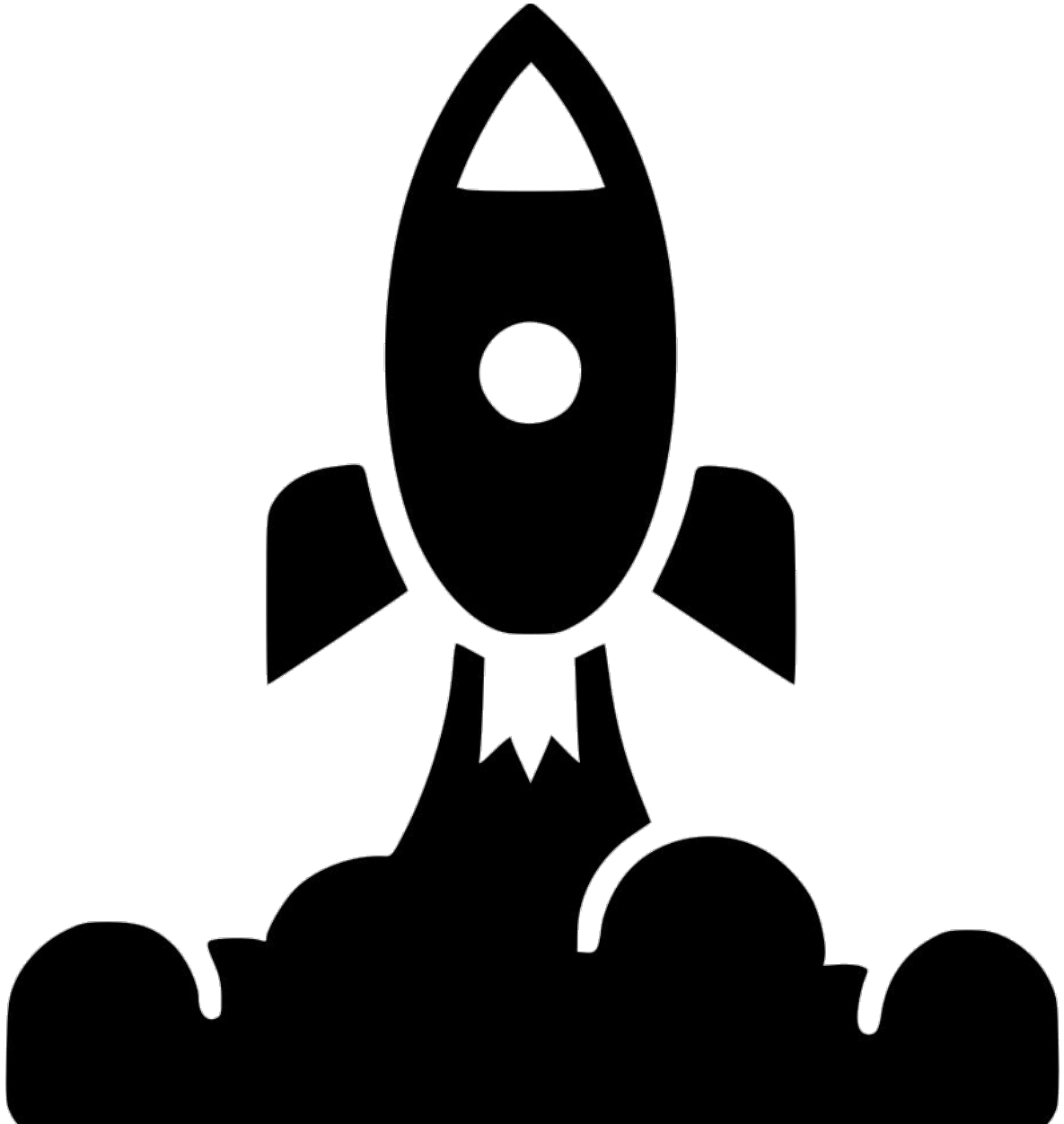
Gain a deeper understanding on how **identity** works within a service mesh. How to leverage this knowledge in your application **architecture**.

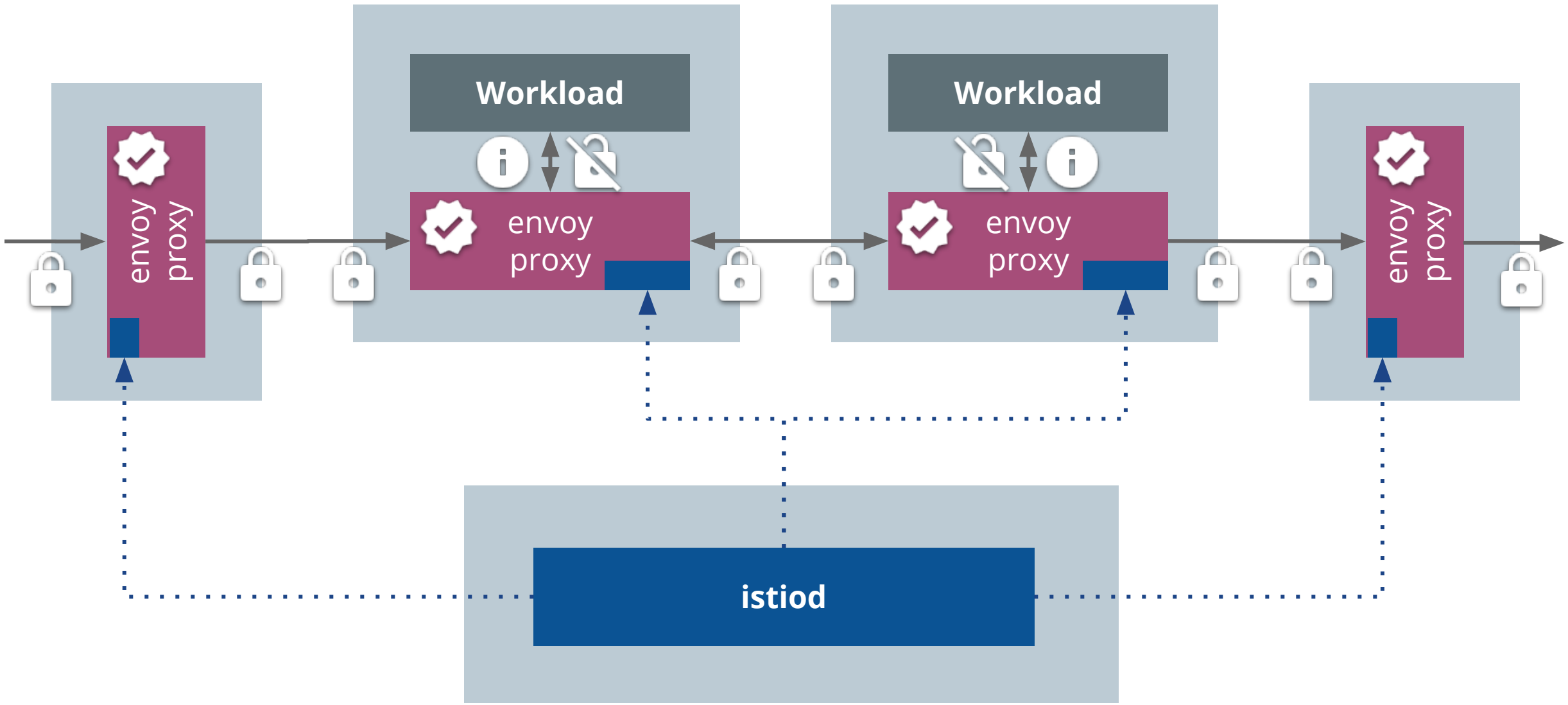
What will you not learn in this talk?

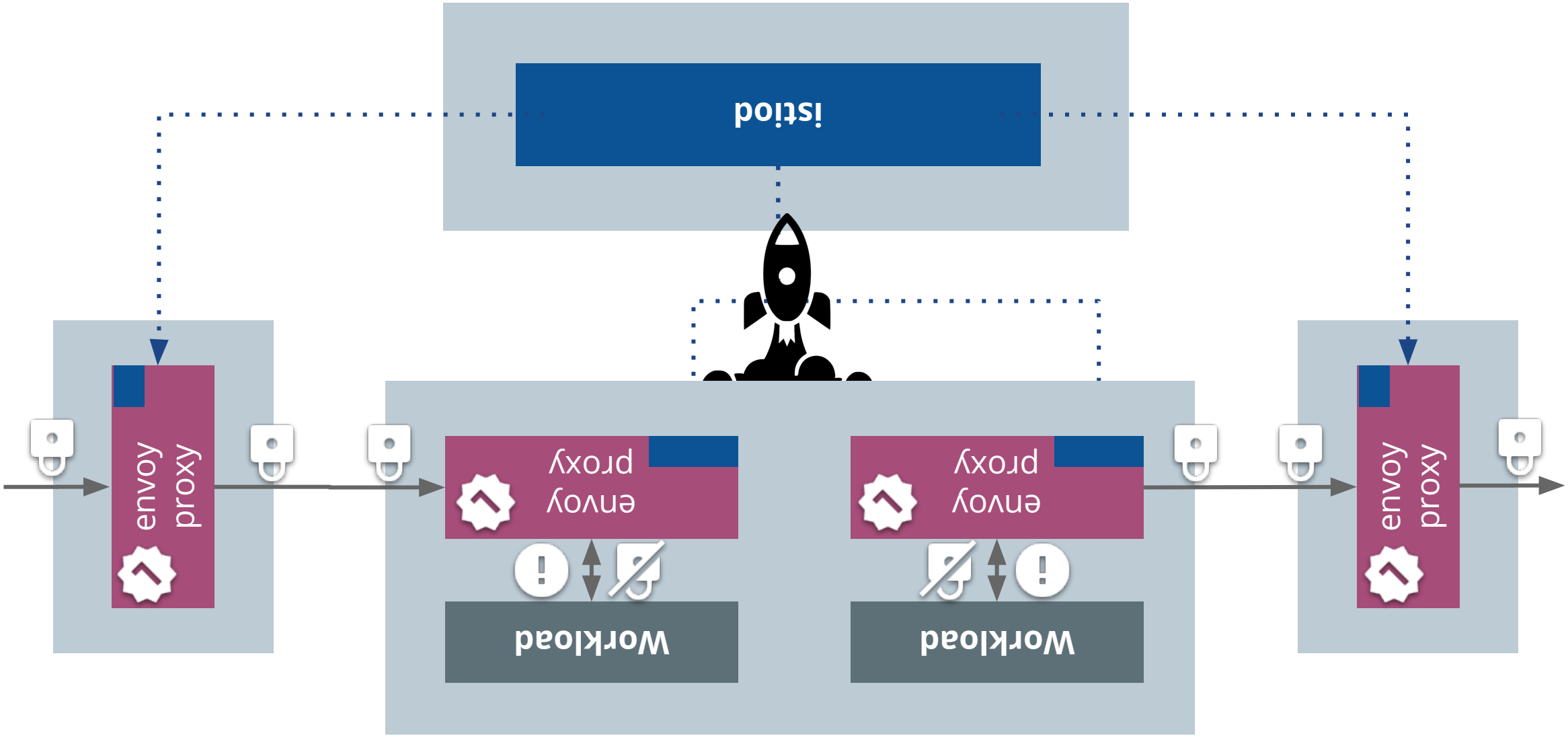
You will not learn the **hottest** new istio features. Just a bit of behind the curtain plumbing.

Fear comes from not knowing what to expect...

— Chris Hadfield, An Astronaut's Guide to Life on Earth




Workload
Workload
envoy proxy
envoy proxy
envoy proxy
envoy proxy
istiod


istiod
envoy proxy
envoy proxy
envoy proxy
Workload
Workload

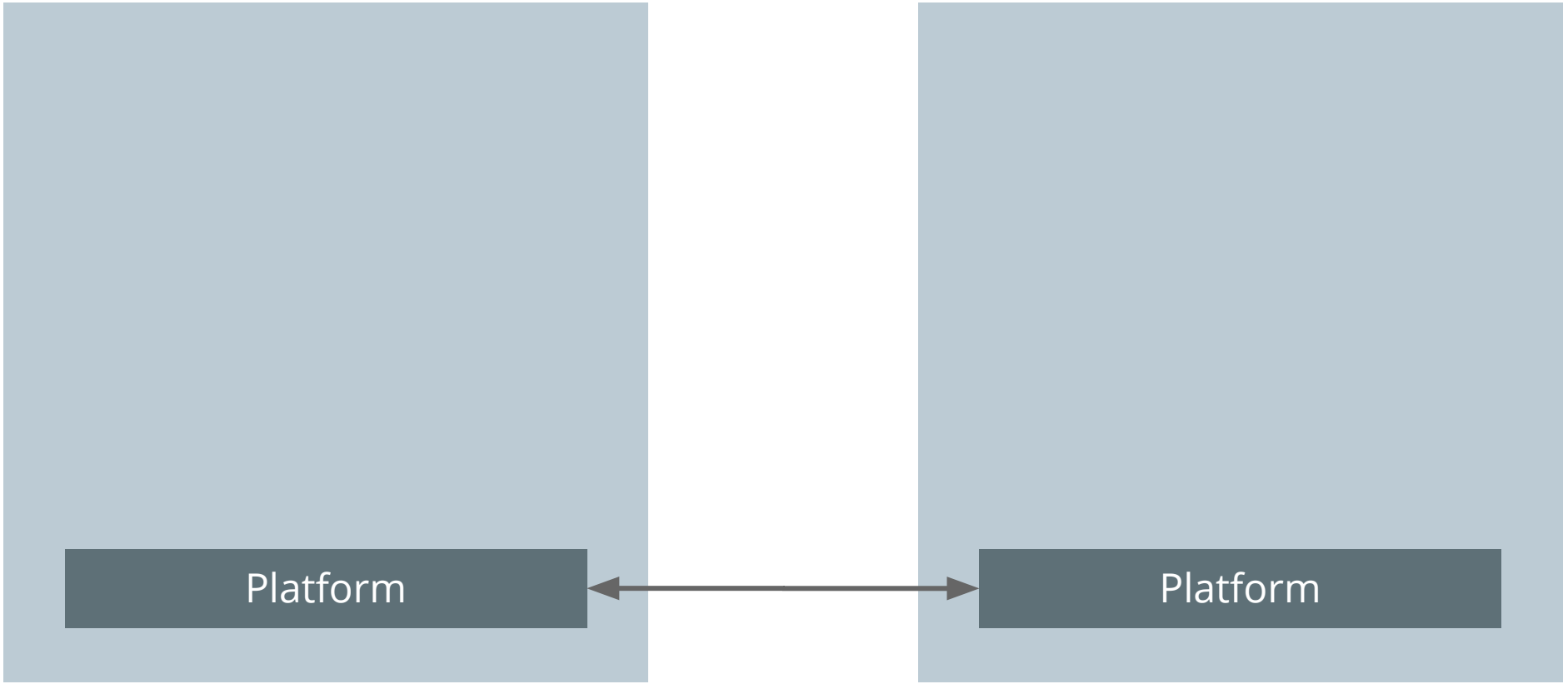
SPIFFE

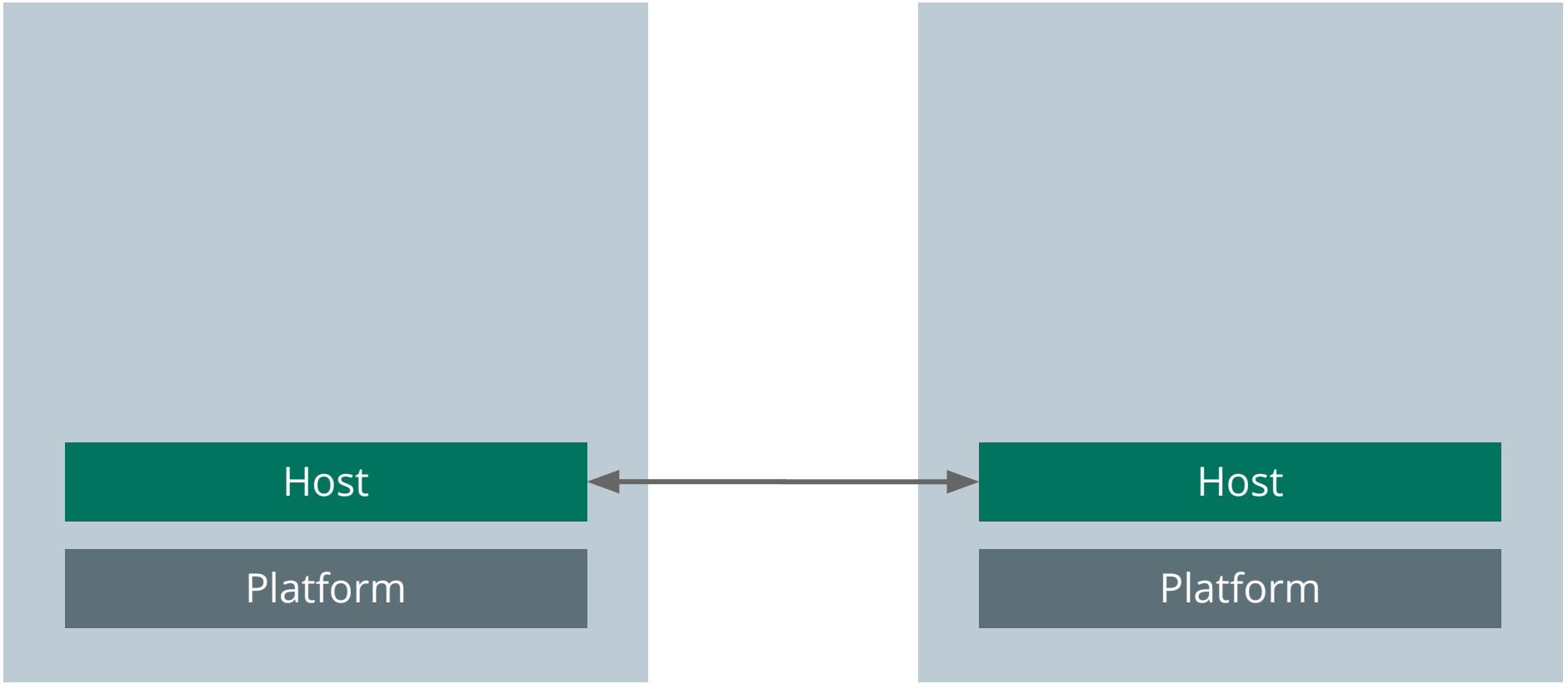
The Cloud Native Identity

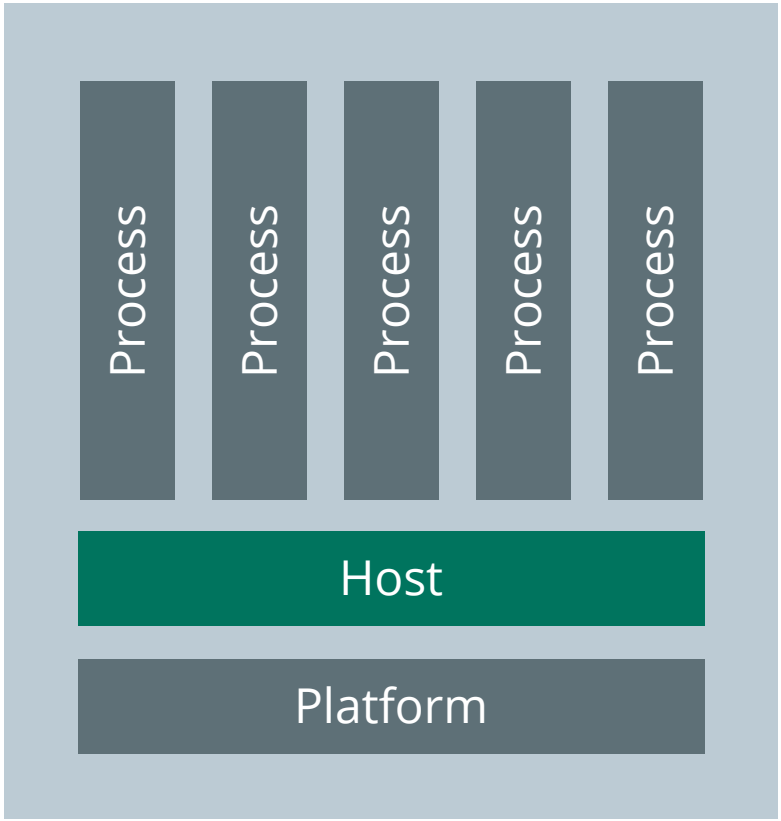
spiffe

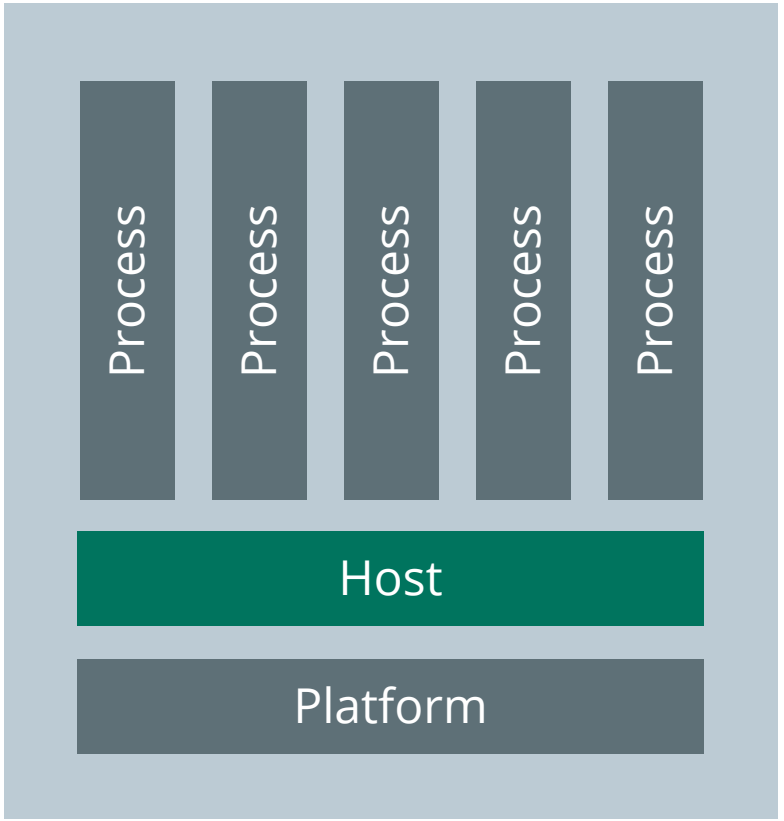
Secure Production Identity Framework for Everyone

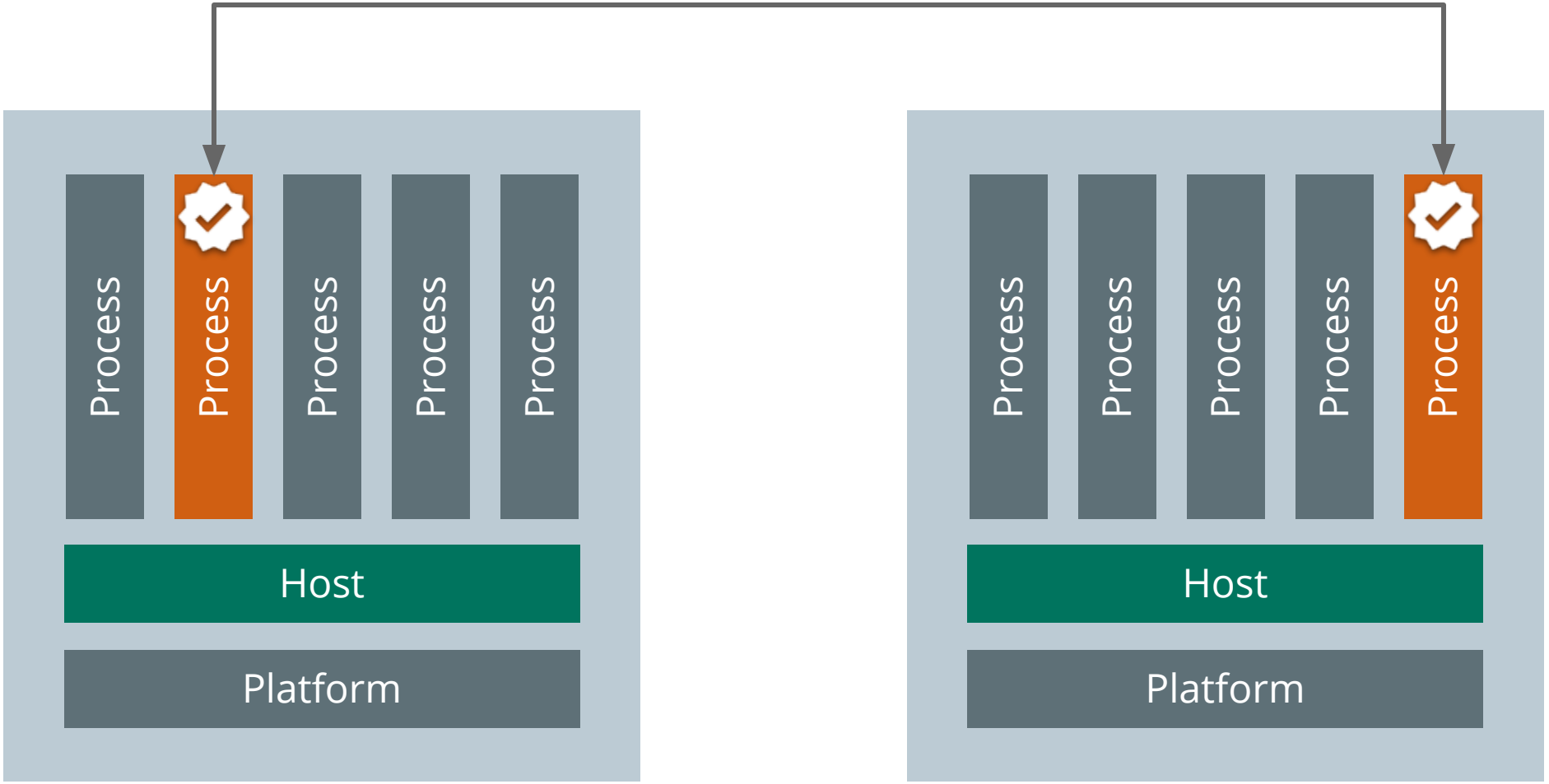
I need to Google it every time. nobody really remembers this really


Platform
Platform


Host
Platform
Host
Platform


Process
Process
Process
Process
Process
Host
Platform


Process
Process
Process
Process
Process
Host
Platform


Process
Process
Process
Process
Process
Host
Platform
Process
Process
Process
Process
Process
Host
Platform

SPIFFE ID

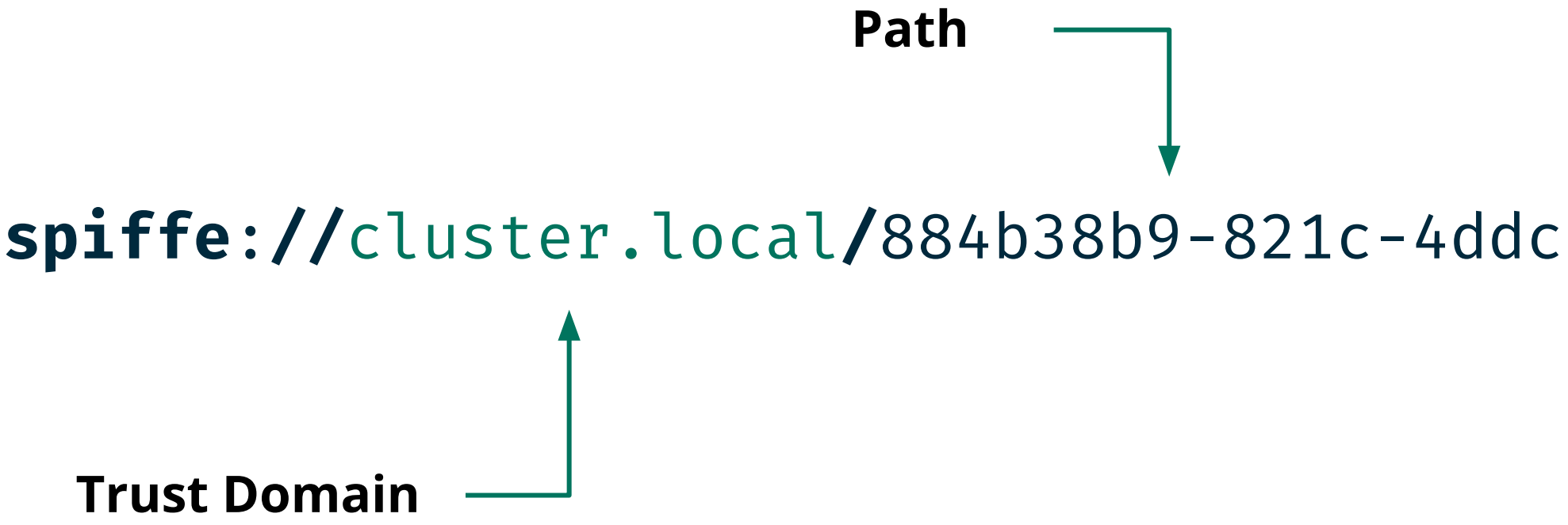
The core of the spec: How to name your workloads! This is called a SPIFFE identifier (or SPIFFE-ID). It’s always a Uniform Resource Identifier (**URI**).

SPIFFE ID is an URI

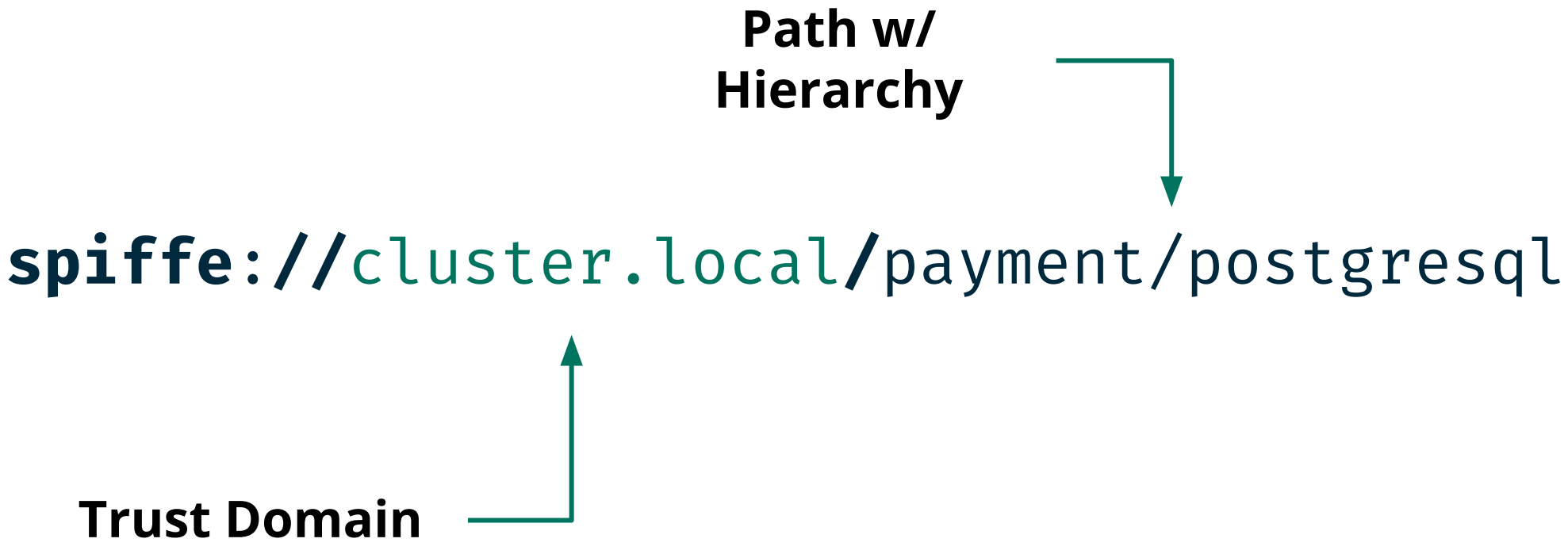
`spiffe://cluster.local`

Trust Domain

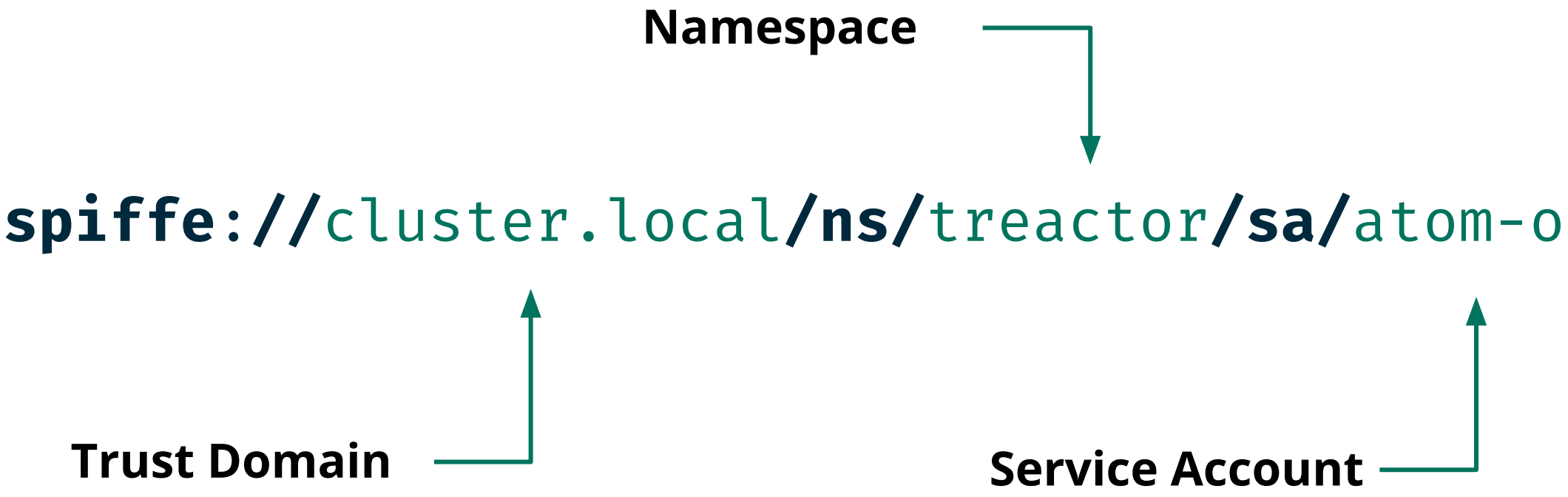
SPIFFE ID must have a path component for workloads



SPIFFE ID can have hierarchy



SPIFFE ID in istio



Trust Domain, a perimeter from workload identities

An important part of the SPIFFE-ID is the trust domain.

By default it’s cluster.local in Istio, but when having multiple cluster or domains it well worth thinking about it. Trust can be established between different trust domains.

Nice to have a naming convention, but how do I trust that identity?

It's not enough to have a consistent naming convention

SPIFFE Verifiable Identity Document aka SVID

An SVID (SPIFFE Verifiable Identity Document) is a SPIFFE-ID signed by an authority. There are two existing implementations right now.

JWT SPIFFE Verifiable Identity Document

A JWT tokens can be an SVID if it complies to certain rules, but let’s ignore this type as we want to use the JWT token for the application layer.

X.509 SPIFFE Verifiable Identity Document

Istio uses X.509 SVID for identity over mTLS.

SPIFFE Specification, what does it contain?

The SPIFFE Identity and Verifiable Identity Document aka **SPIFFE-ID**

SPIFFE Verifiable Identity Document aka **SVID**

The SPIFFE Workload API

Be sure to check out SPIRE for your non-mesh workloads

SPIRE is hosted at the same site where the SPIFFE specification is hosted

X.509

Certificates in the context of SPIFFE

X.509 specification was first published at November 25, 1988

It’s not because it’s old that it’s broken!

X.509 Example

```
-----BEGIN CERTIFICATE-----
MAbCCzCCAbCgAwIBAgICEAxdCgYIKoZI030EAwIwXALXMAkGA1UEBhMCQkUxETAP
BgNVBAoMCENvbGxpYnJhMRswGQYAVBQLDBJDb2xsaWJyYSBBdXRob3JpdHkxHjAc
BgNVBAMMFUNatIstioSummitF1Ghvcl0eSBYMDAeFw0yMDExMjMxMTU2MzZaFw0y
MTEyMDMxMTU2MzZaMEMxCzAJBgNVBAYTAkJFMREwDwYDVISTIOhDb2xsaWJyYTES
MBAGA1UECwwJVGVsZW1ldHJ5MQ0wCwYDVQQDDARzaW5rMFkwEwYHKoZIzj0CAQYI
KoZIzj0DAQcDQgAEr3qNb45rpiY5xJ09R8C1A+SzN/ge39CcBJ8EaMuY8Yp9tMXF
4BTxqU/XDDZm7NepPTMNvVbO48Fii7fXg7qM86N6MHgwdgYSPIFFEQH/BGwwaoYp
c3BpZmZlOi8vdGVsZW1ldHJ5LmNvbGxpYnJhY2xvdWQuY29tL3NpbmuCIHNpbmsu
dGVsZW1ldHJ5LmNvbGxpYnJhY2xvdWQuY29thwSC0x/shwQjux92gglsb2NhbGhv
cALEXvanBOXELgYIKoZIzj0EAwIDSQAwRgIhAIaeYILh5WRg81Eysyqv/C8uF2Ja
tDm9ku689mjRJUFAAiEAkU9JvHVcP2Vv5ZF1jHZSua8zV0u8dAGplqEurugOOGI=
-----END CERTIFICATE-----
```

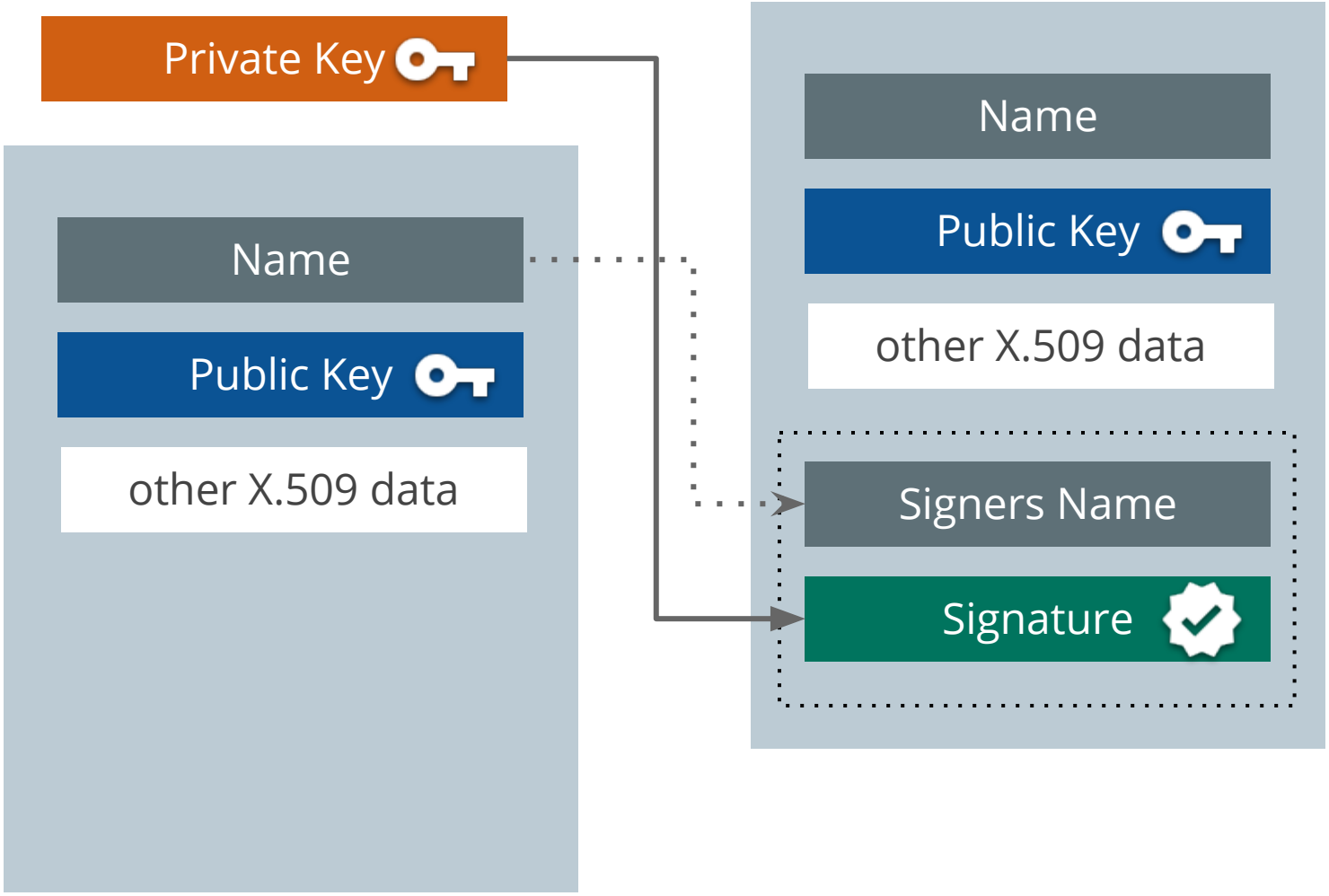
A common way you will see X.509 certificates being moved around: **PEM encoded**

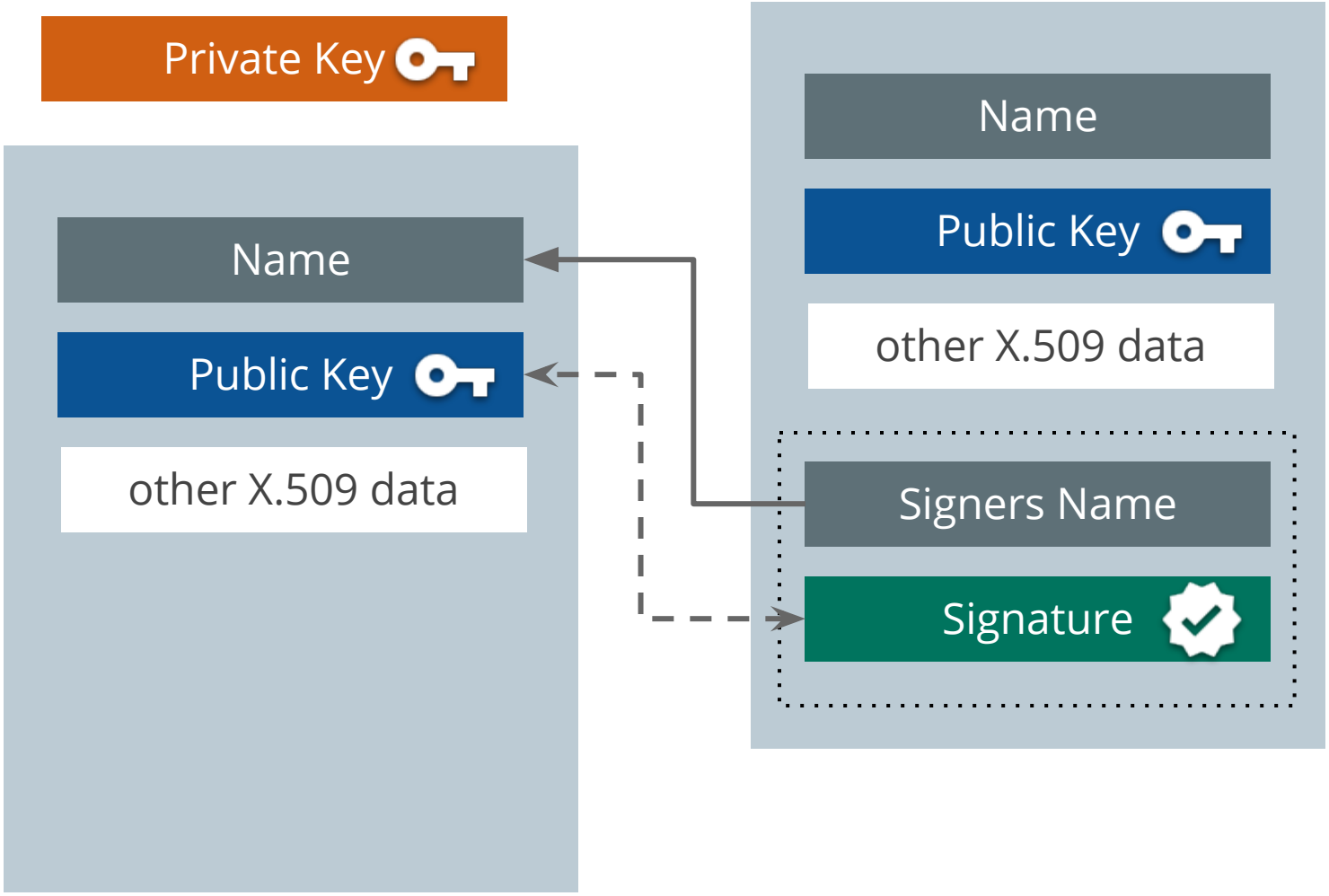
X.509 Example

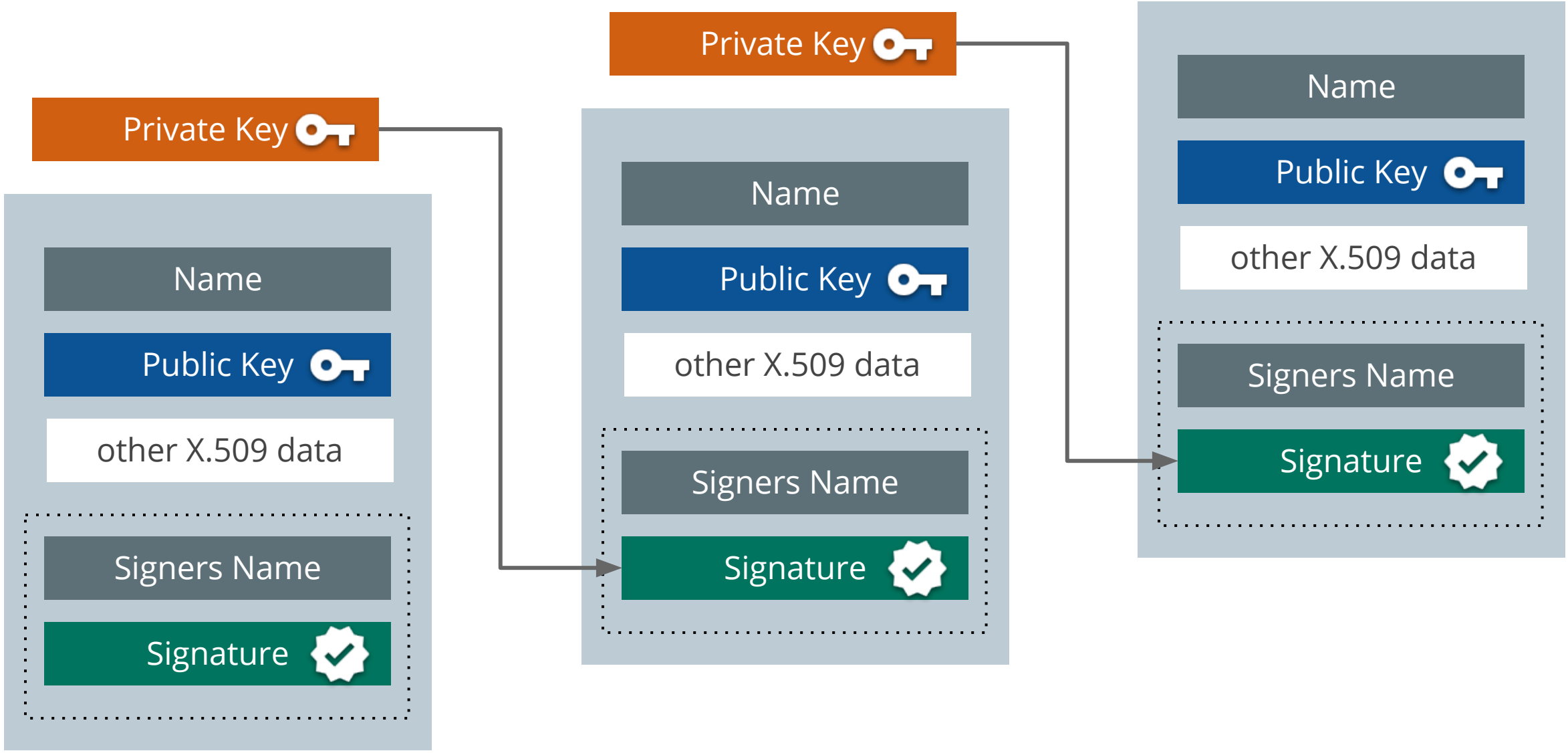
```
        Version: 3 (0x2)
        Serial Number: 4103 (0x1007)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = BE, O = Example, OU = Authority, CN = Authority X0
        Validity
            Not Before: Nov 23 11:56:36 2020 GMT
            Not After : Dec  3 11:56:36 2021 GMT
        Subject: C = BE, O = Example, OU = Unit, CN = Something
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
        X509v3 extensions:
            X509v3 Subject Alternative Name: critical
                URI:spiffe://trust.domain.link/x, DNS:localhost, IP Address:127.0.0.1
    Signature Algorithm: ecdsa-with-SHA256
         30:46:02:21:00:86:9e:60:82:e1:e5:64:60:f3:51:32:b3:2a:
-----BEGIN CERTIFICATE-----
```

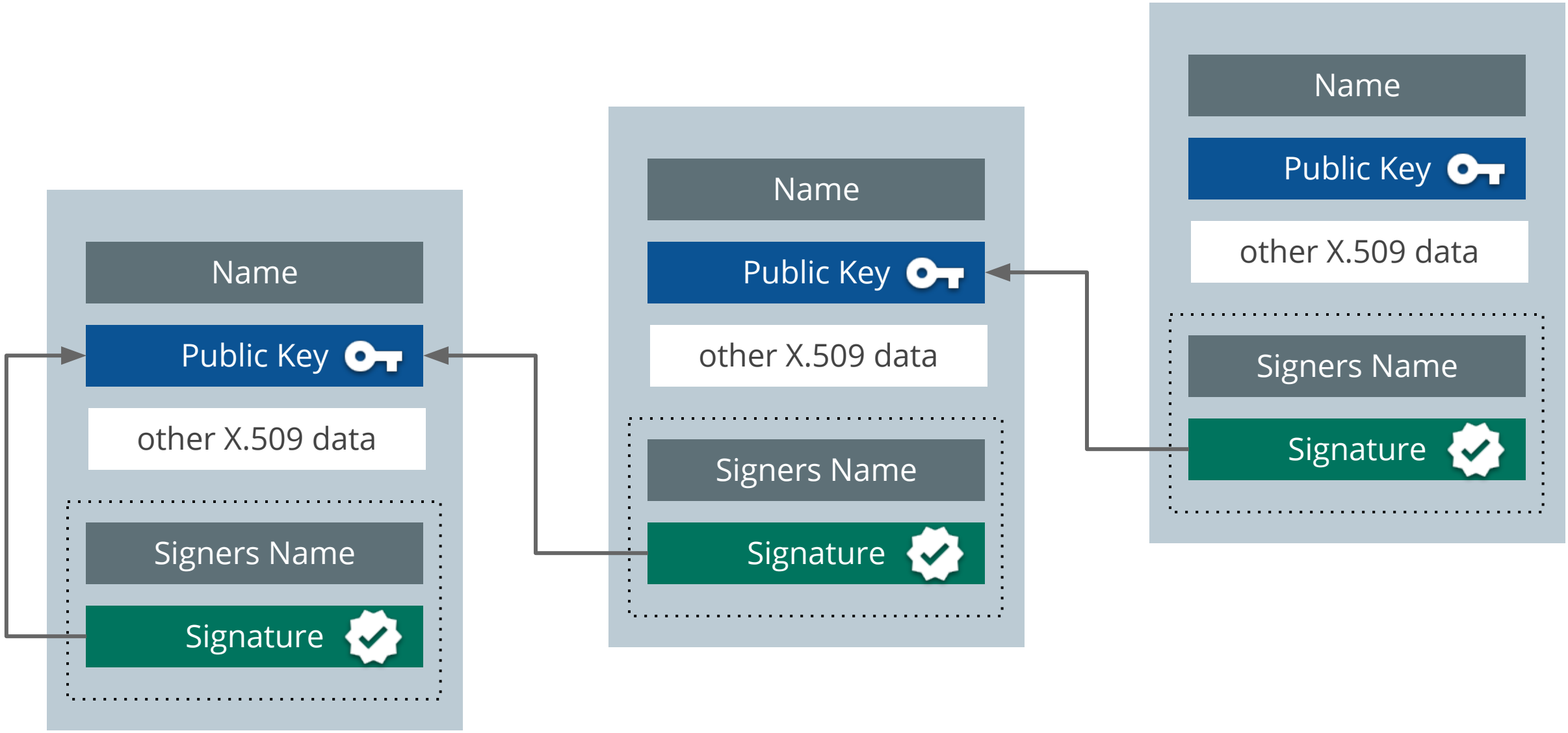
My favorite OpenSSL command

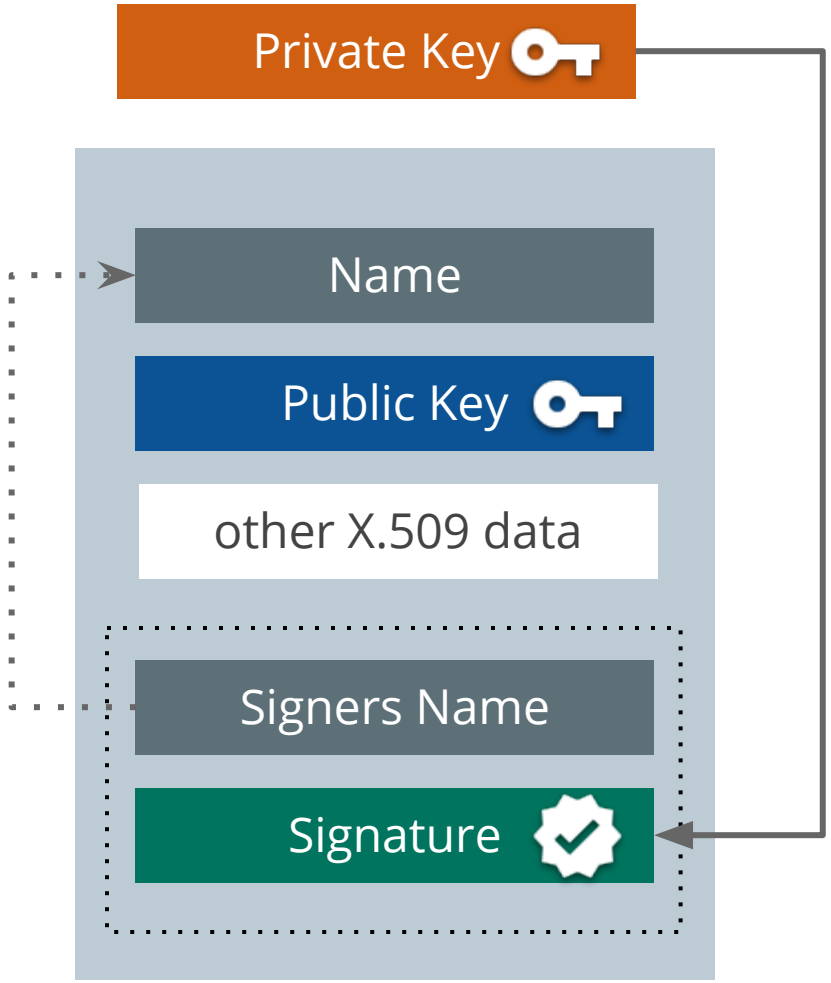
```
> openssl x509 -text -in any-certificate.crt
```

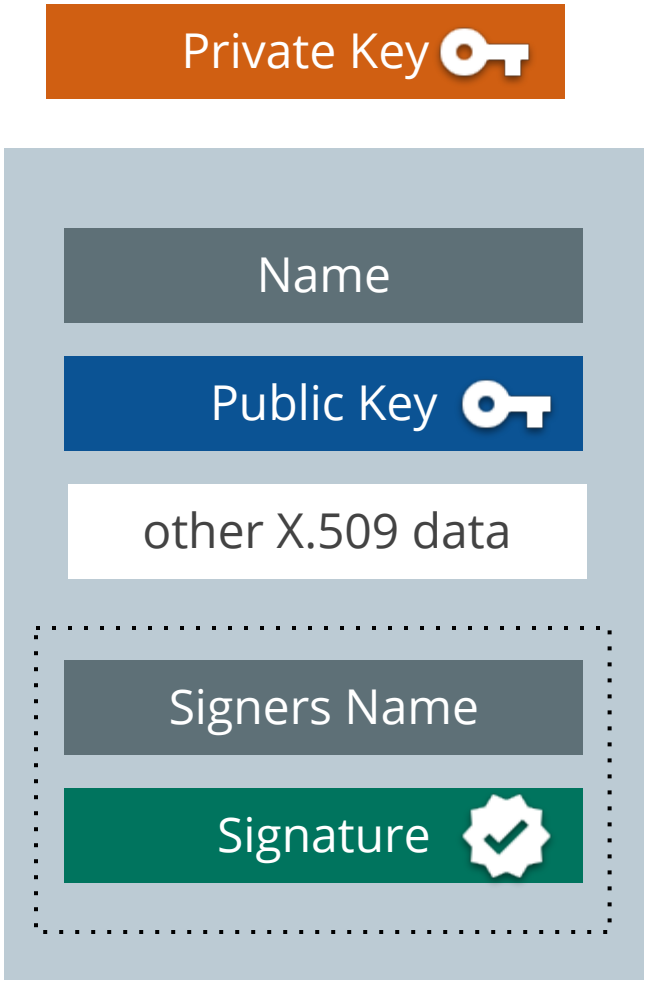

Private Key
Name
Public Key
other X.509 data
Name
Public Key
other X.509 data
Signers Name
Signature


Private Key
Name
Public Key
other X.509 data
Name
Public Key
other X.509 data
Signers Name
Signature


Private Key
Name
Public Key
other X.509 data
Signers Name
Signature
Private Key
Name
Public Key
other X.509 data
Signers Name
Signature
Name
Public Key
other X.509 data
Signers Name
Signature


Name
Public Key
other X.509 data
Signers Name
Signature
Name
Public Key
other X.509 data
Signers Name
Signature
Name
Public Key
other X.509 data
Signers Name
Signature


Private Key
Name
Public Key
other X.509 data
Signers Name
Signature


Private Key
Name
Public Key
other X.509 data
Signers Name
Signature

SPIFFE ID in Subject Alternative Name extension

SPIFFE ID is set as a **URI** type in Subject Alternative Name (SAN), only one URI SAN is allowed, but other information can be encoded in, like IP and DNS names.

```
...
        X509v3 extensions:
            X509v3 Subject Alternative Name: critical
                URI:spiffe://trust.domain/x, DNS:localhost, IP Address:127.0.0.1
    Signature Algorithm: ecdsa-with-SHA256
         30:46:02:21:00:86:9e:60:82:e1:e5:64:60:f3:51:32:b3:2a:
-----BEGIN CERTIFICATE-----
```

Signing Certificates

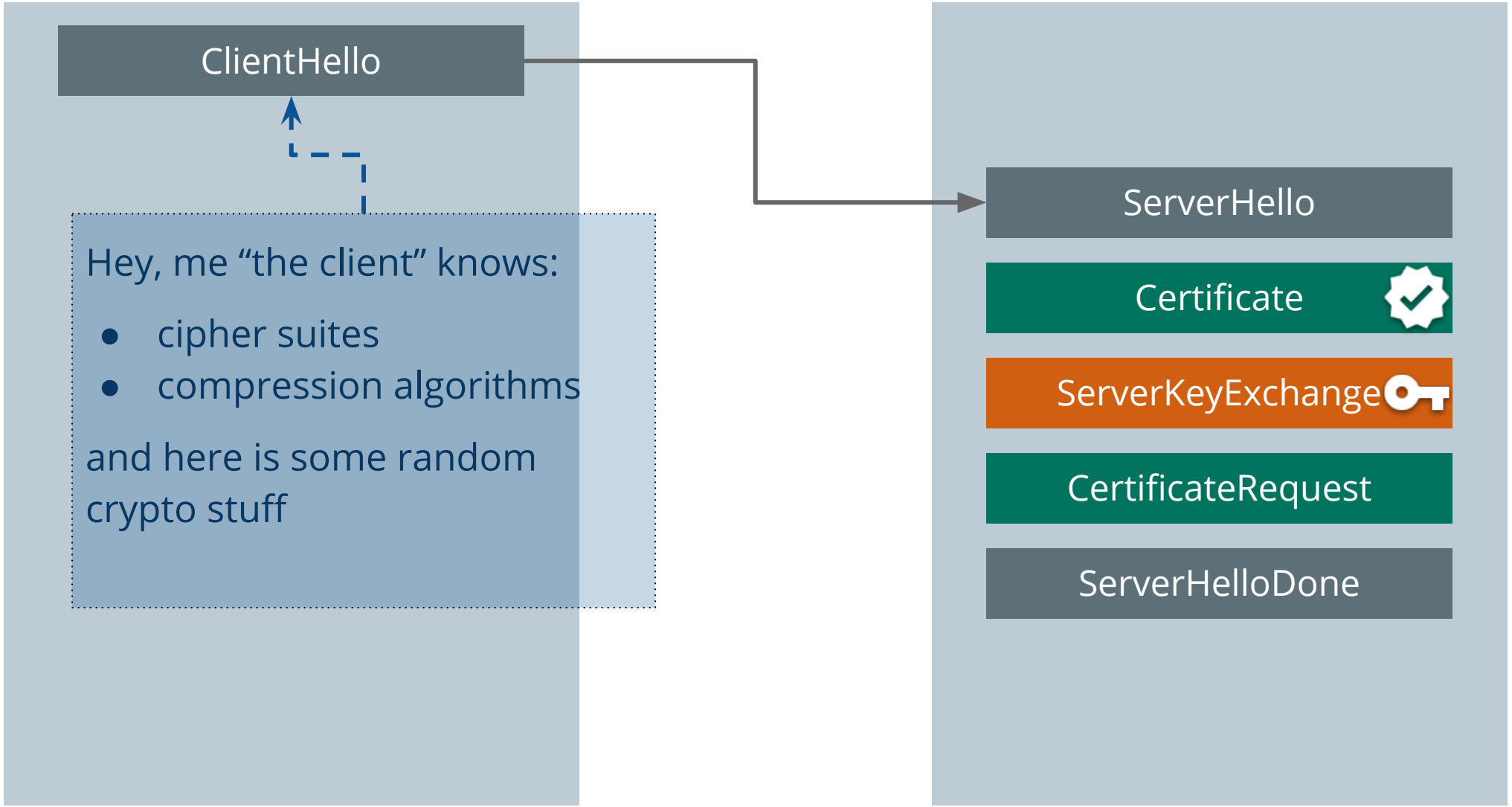
While an SVID is nothing more than an X.509, some restrictions apply. A signing certificate needs to have CA set to true and the keyCertSign in the key usage extension set. Signing Certificates should never be used to identify workloads.

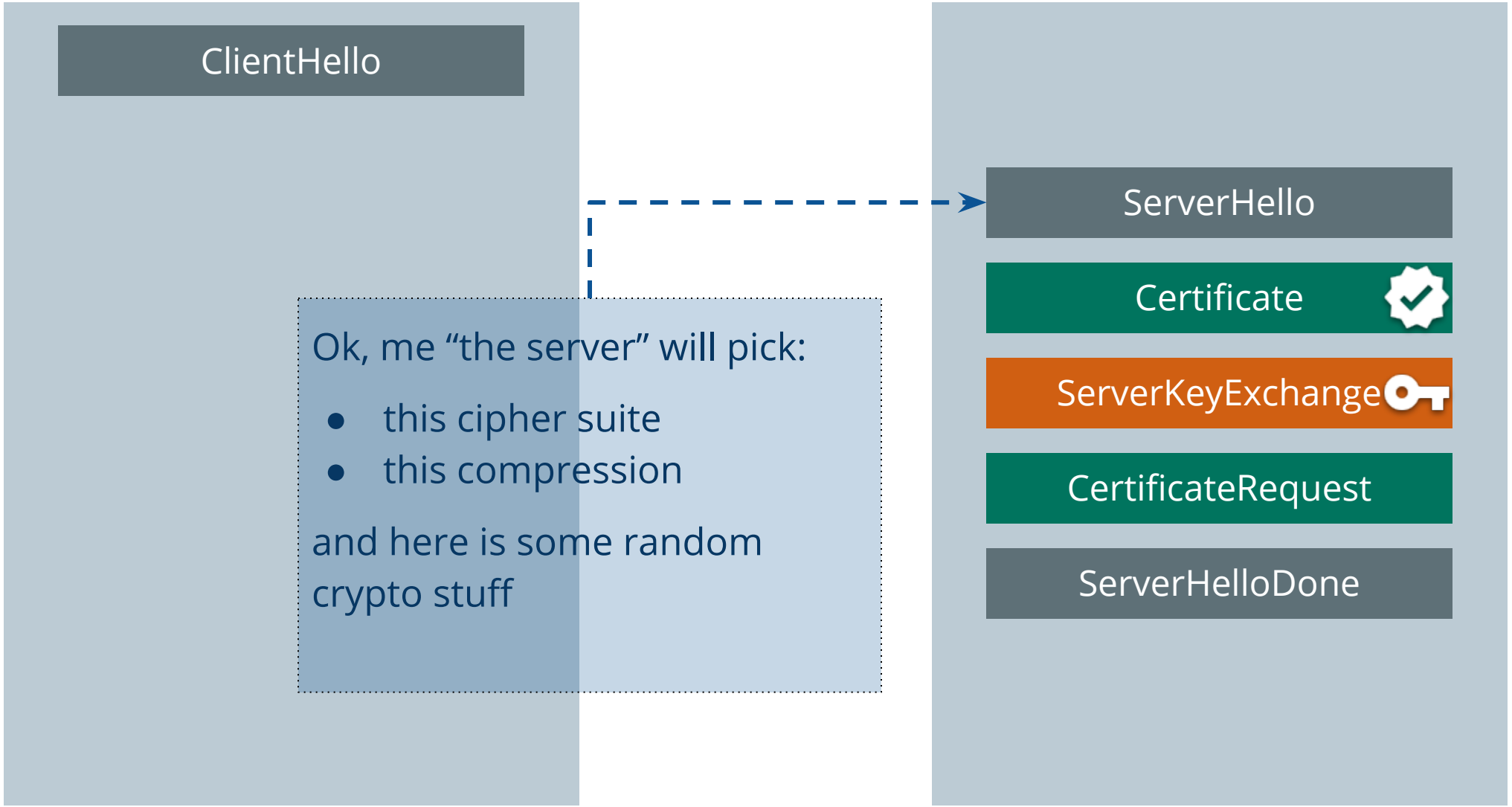
Leaf Certificates

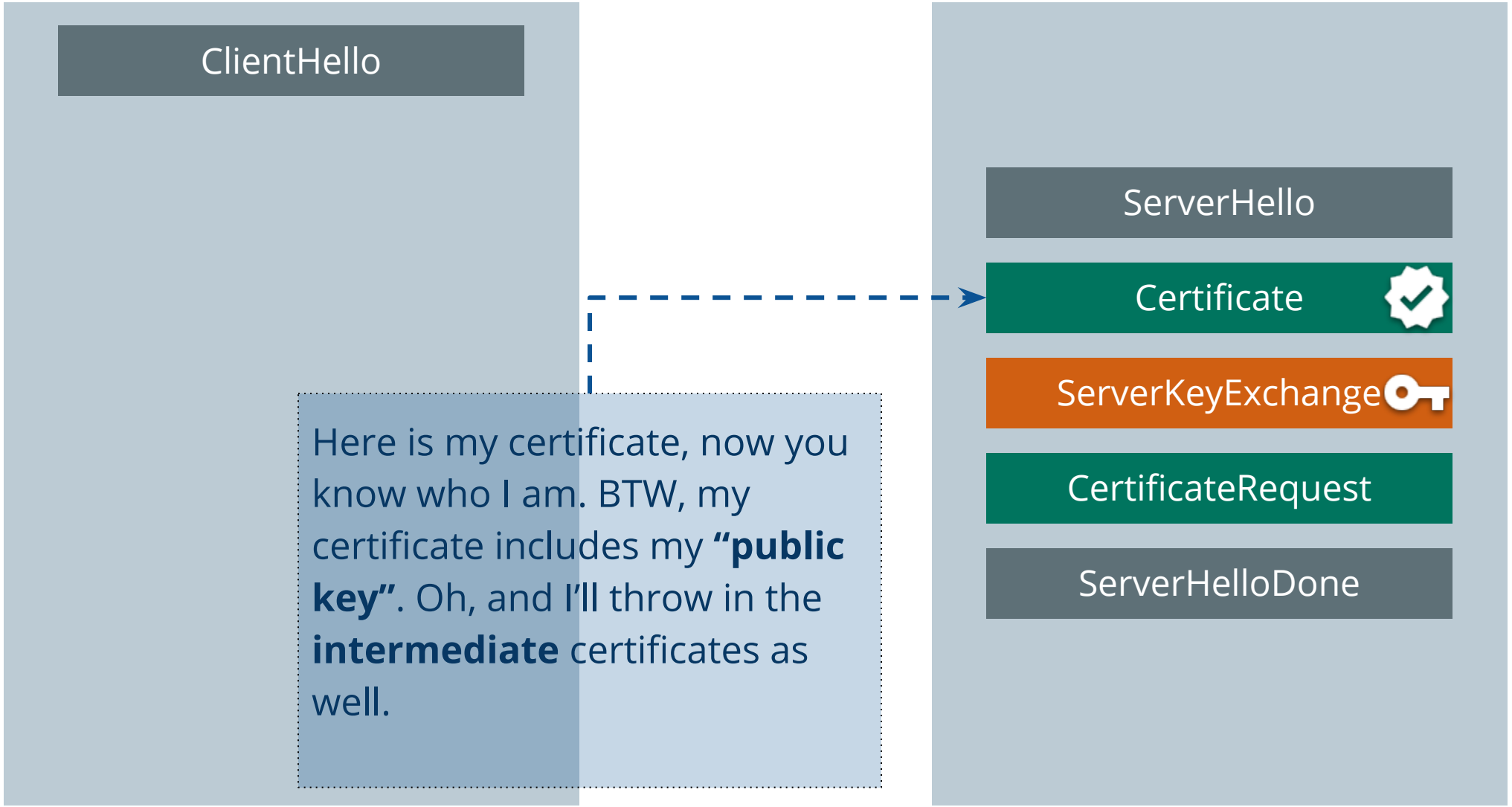
A leaf certificate is used to identify a resource or caller, it’s used in authentication.
The SPIFFE IDs MUST have a non-root path component.

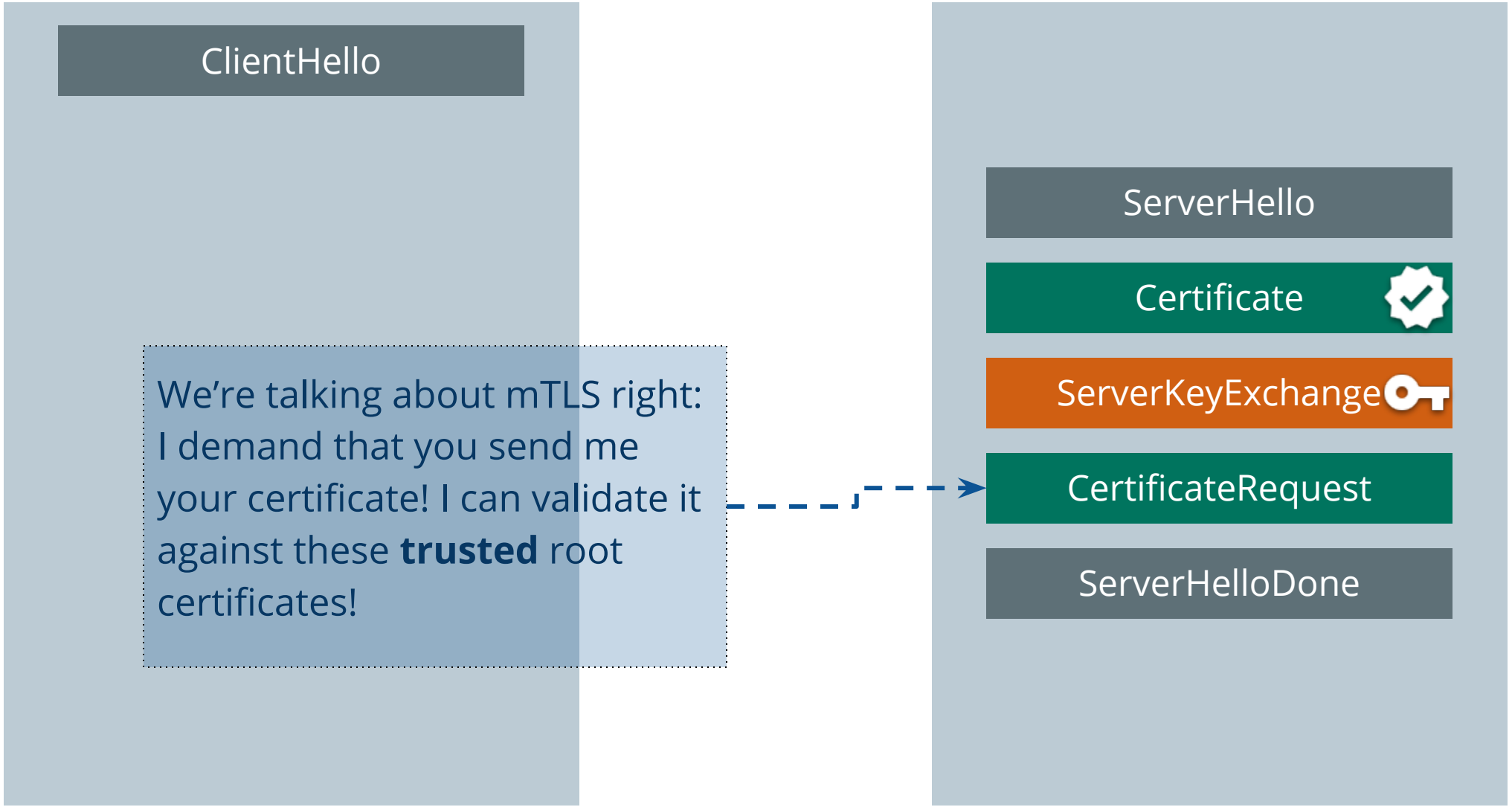
TLS Handshake

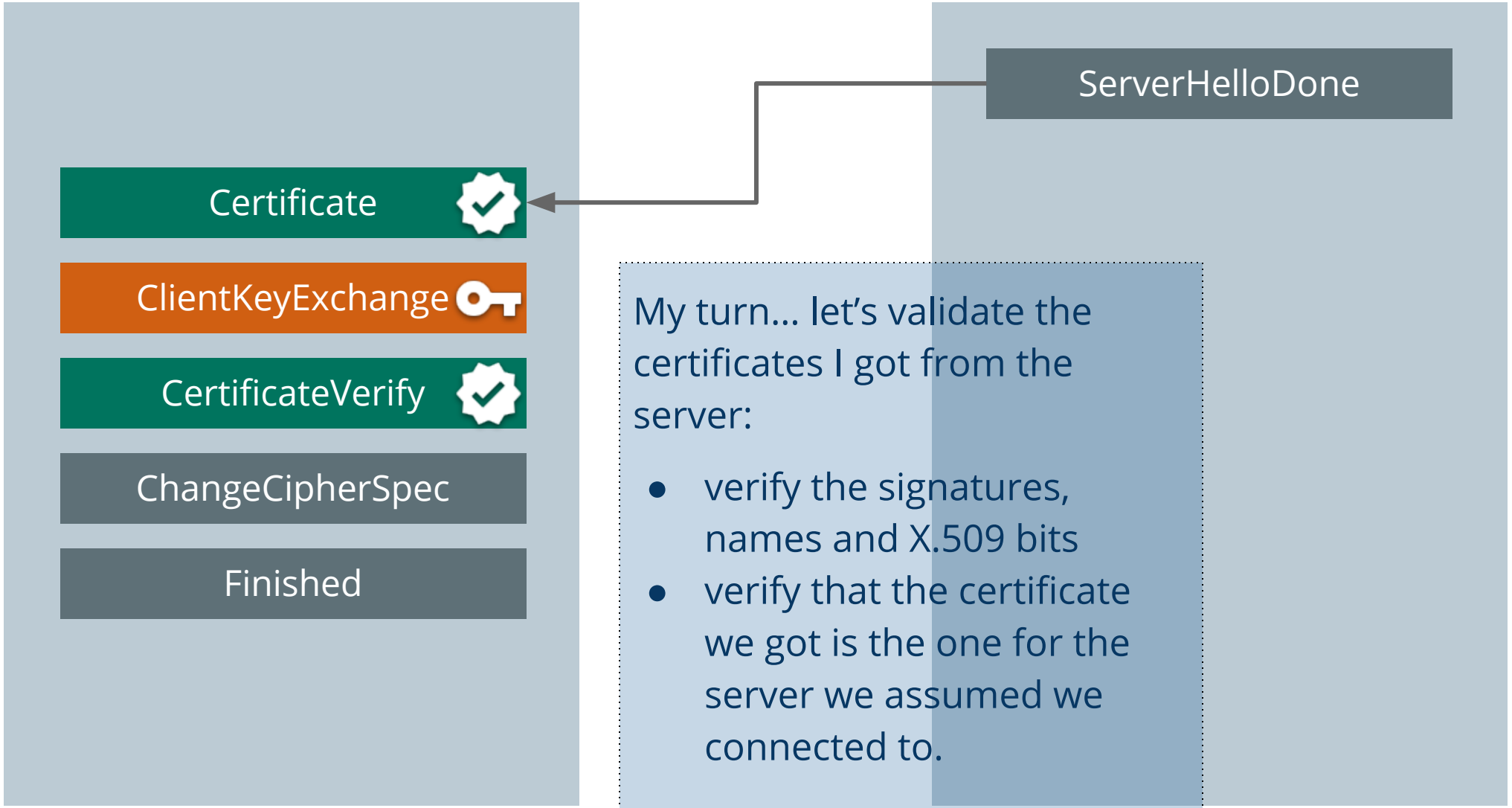
Server and Client throwing X.509 stuff at each other

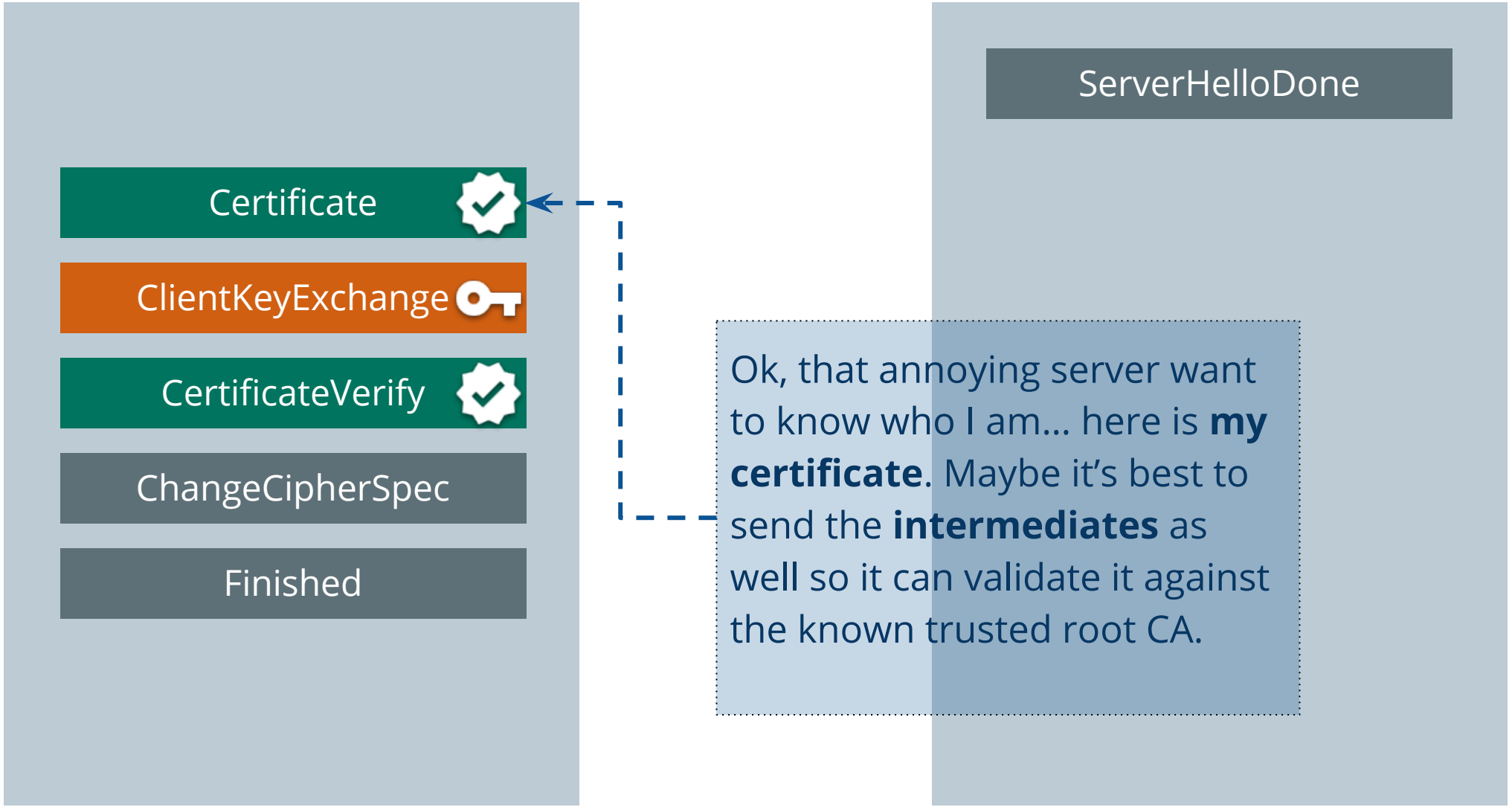

ClientHello
Hey, me “the client” knows:
cipher suites
compression algorithms
and here is some random crypto stuff
ServerHello
Certificate
ServerKeyExchange
CertificateRequest
ServerHelloDone

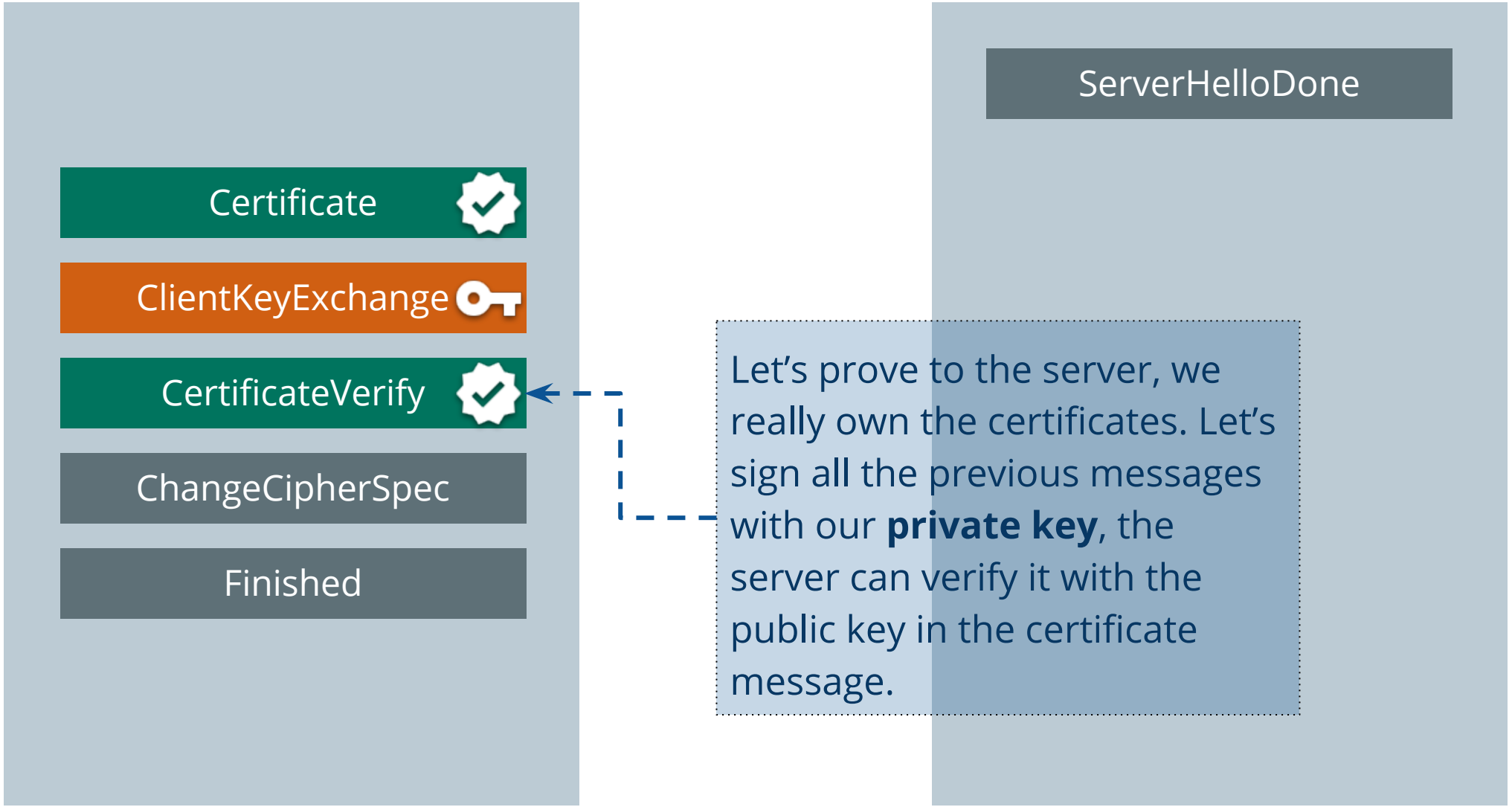

ClientHello
Ok, me “the server” will pick:
this cipher suite
this compression
and here is some random crypto stuff
ServerHello
Certificate
ServerKeyExchange
CertificateRequest
ServerHelloDone

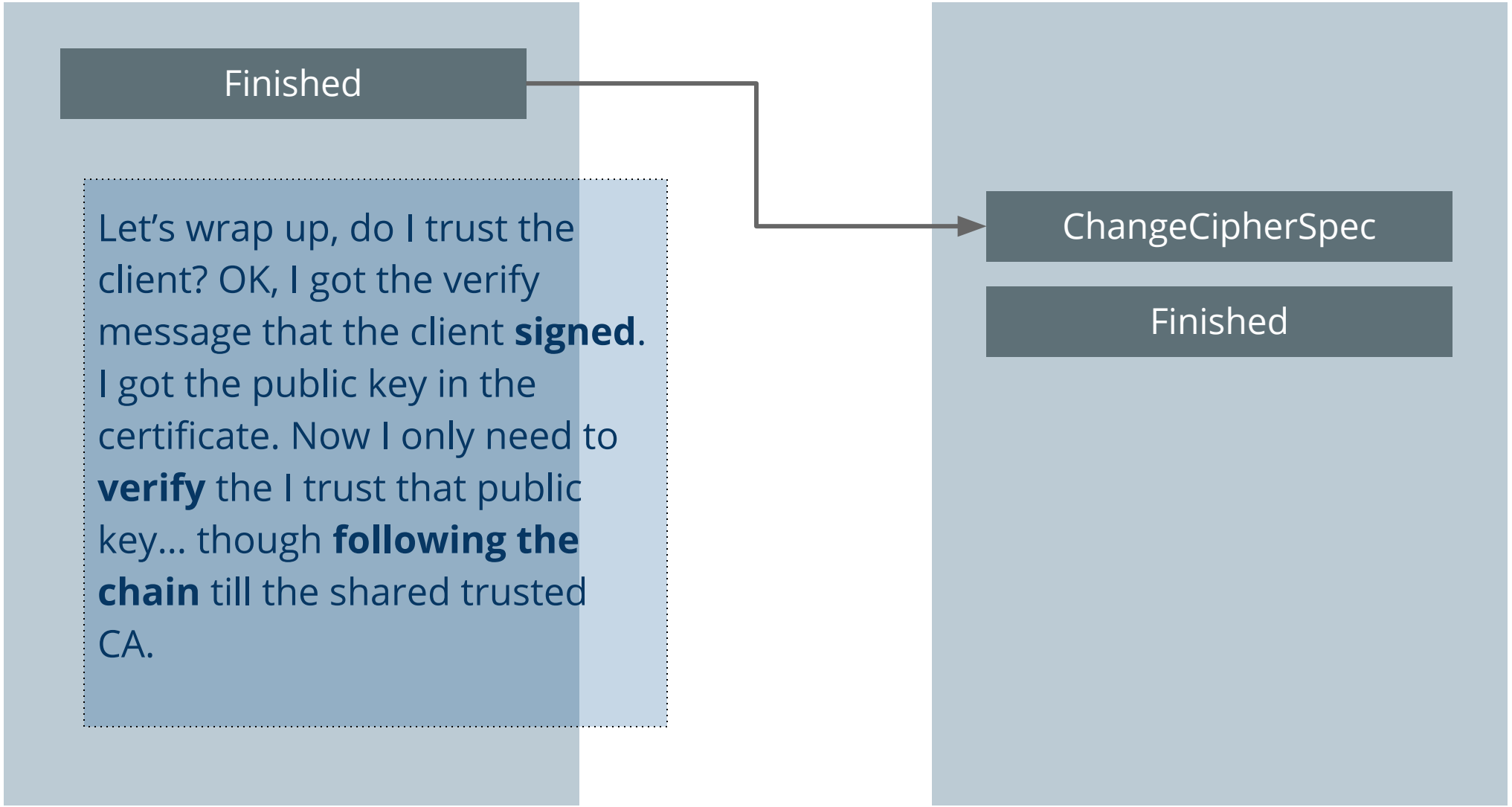

ClientHello
ServerHello
Certificate
ServerKeyExchange
CertificateRequest
ServerHelloDone
Here is my certificate, now you know who I am. BTW, my certificate includes my **“public key”**. Oh, and I’ll throw in the **intermediate** certificates as well.

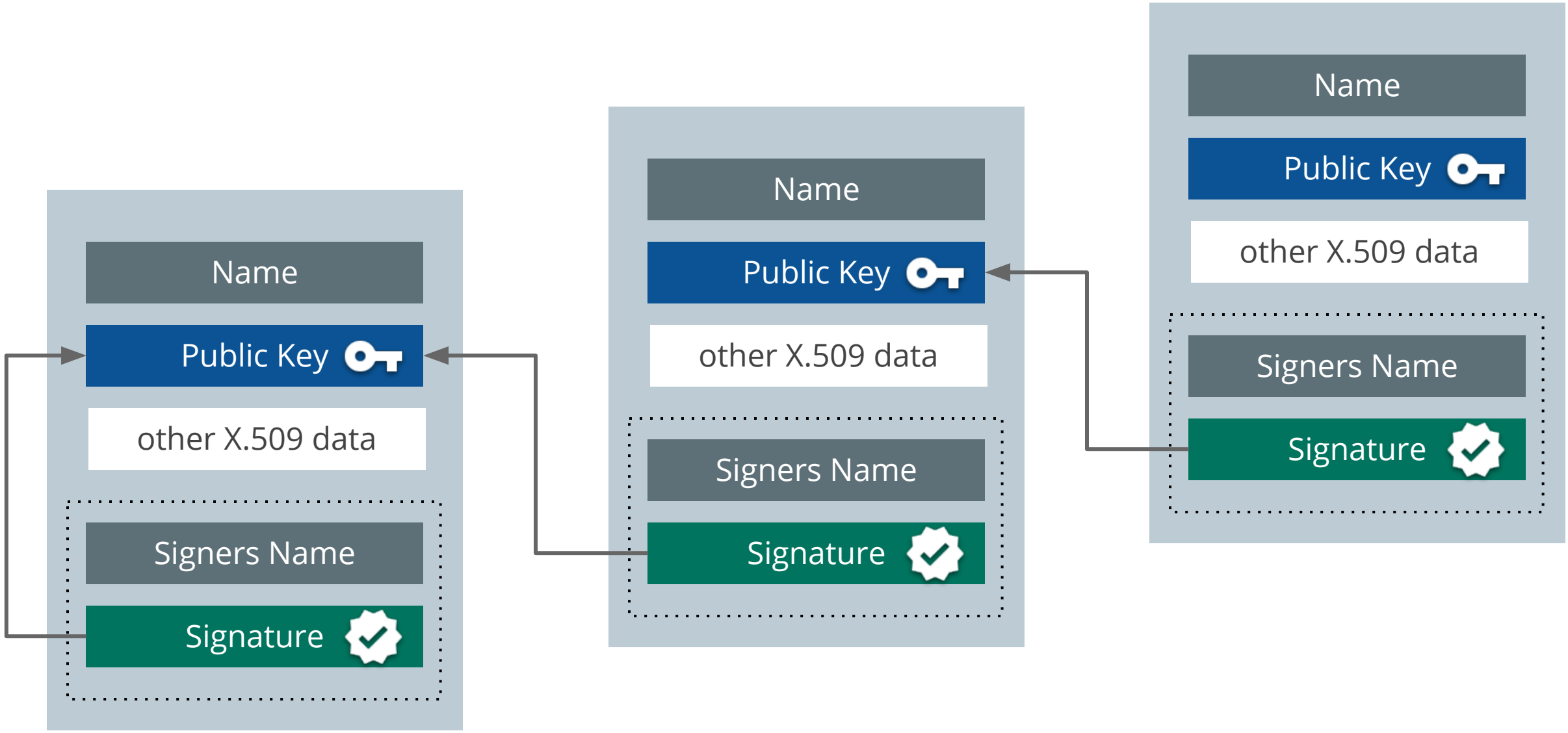

ClientHello
We’re talking about mTLS right: I demand that you send me your certificate! I can validate it against these **trusted** root certificates!
ServerHello
Certificate
ServerKeyExchange
CertificateRequest
ServerHelloDone

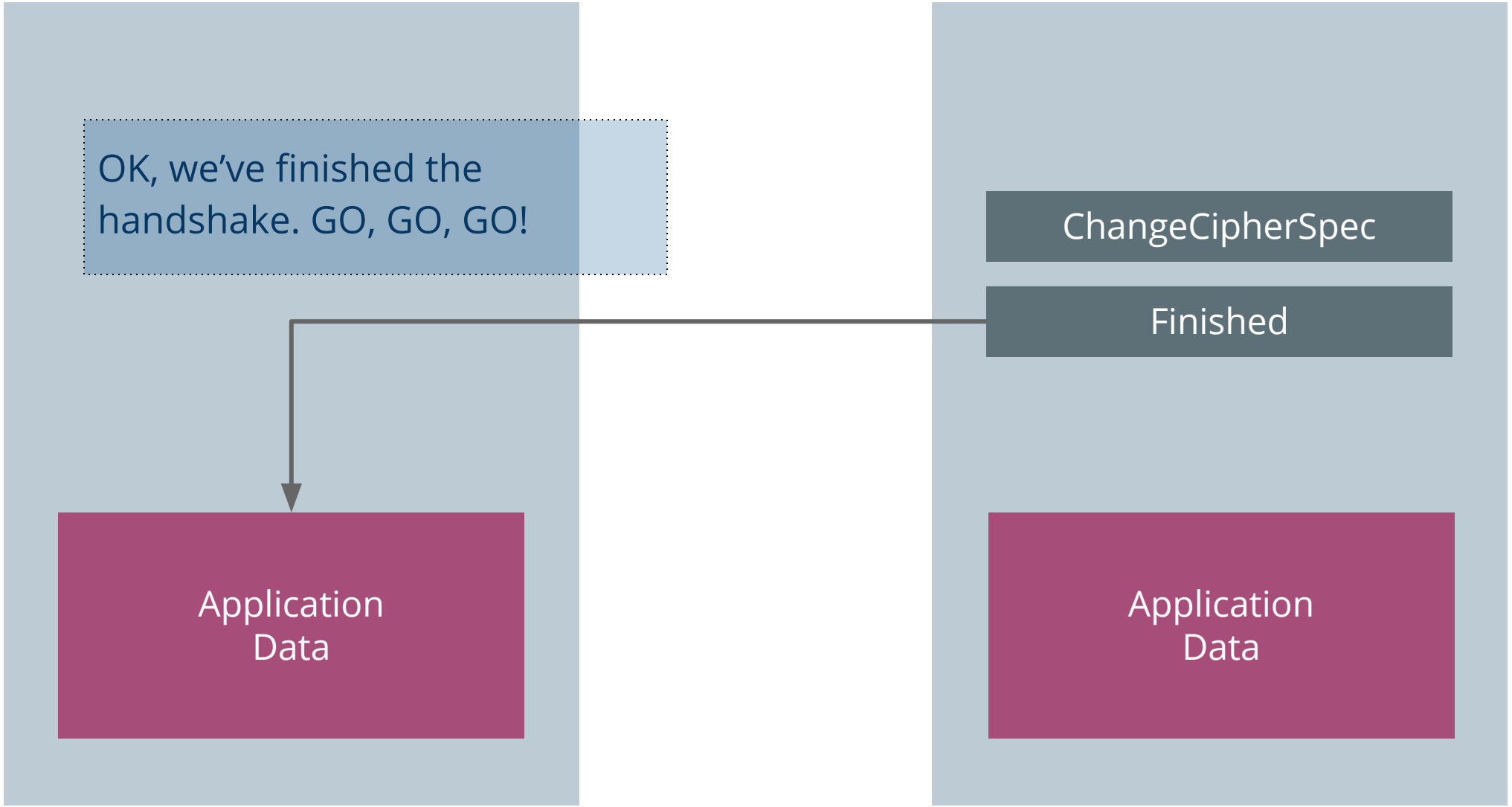

ServerHelloDone
Certificate
ClientKeyExchange
CertificateVerify
ChangeCipherSpec
Finished
My turn… let's validate the certificates I got from the server:
verify the signatures, names and X.509 bits
verify that the certificate we got is the one for the server we assumed we connected to.

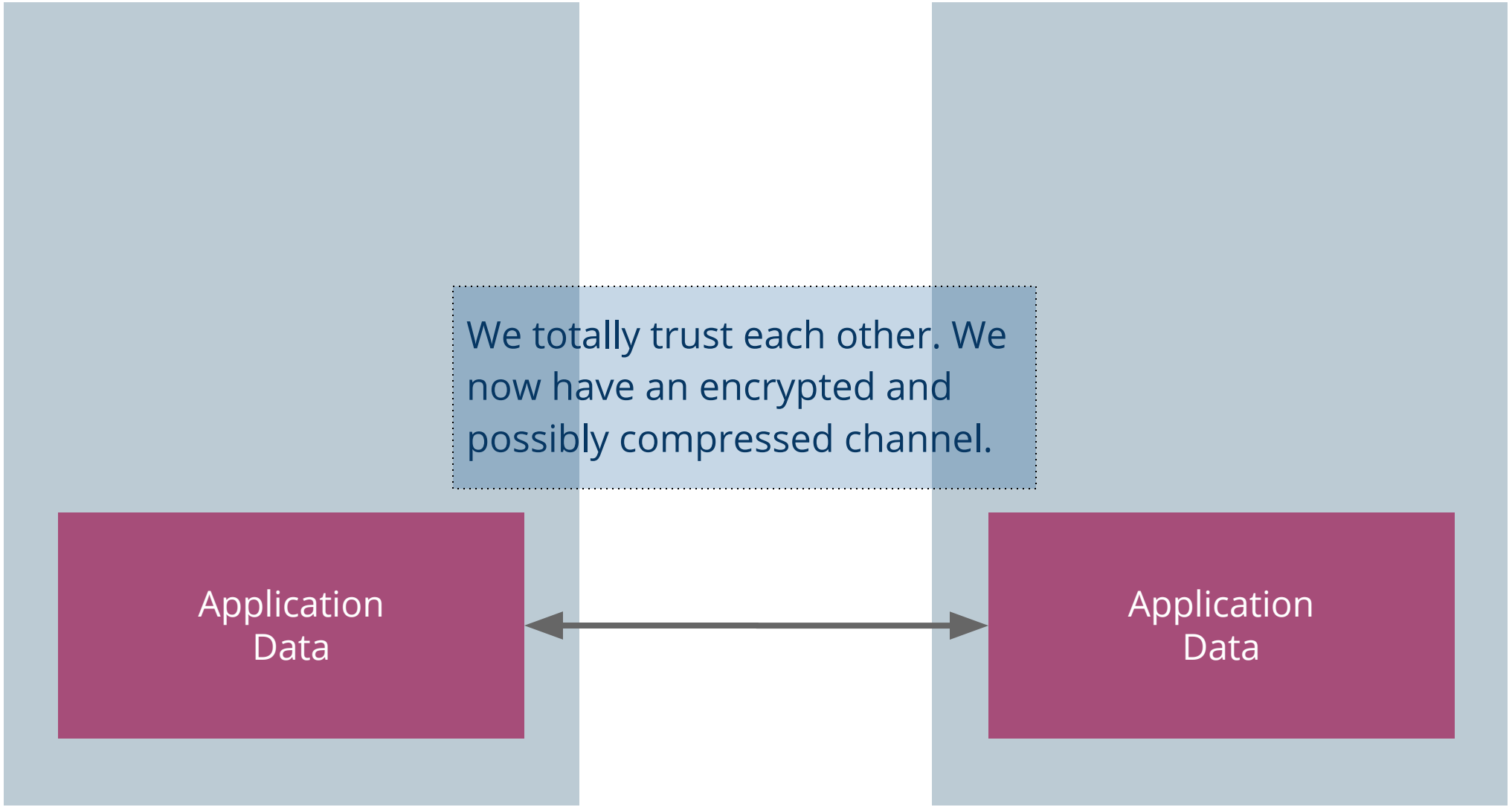

ServerHelloDone
Certificate
ClientKeyExchange
CertificateVerify
ChangeCipherSpec
Finished
Ok, that annoying server want to know who I am... here is **my certificate**. Maybe it's best to send the **intermediates** as well so it can validate it against the known trusted root CA.


ServerHelloDone
Certificate
ClientKeyExchange
CertificateVerify
ChangeCipherSpec
Finished
Let’s prove to the server, we really own the certificates. Let’s sign all the previous messages with our **private key**, the server can verify it with the public key in the certificate message.


Finished
Let’s wrap up, do I trust the client? OK, I got the verify message that the client **signed**. I got the public key in the certificate. Now I only need to **verify** the I trust that public key… though **following the chain** till the shared trusted CA.
ChangeCipherSpec
Finished


Name
Public Key
other X.509 data
Signers Name
Signature
Name
Public Key
other X.509 data
Signers Name
Signature
Name
Public Key
other X.509 data
Signers Name
Signature


OK, we've finished the handshake. GO, GO, GO!
ChangeCipherSpec
Finished
Application Data
Application Data


We totally trust each other. We now have an encrypted and possibly compressed channel.
Application Data
Application Data

The TLS handshake wrap-up.

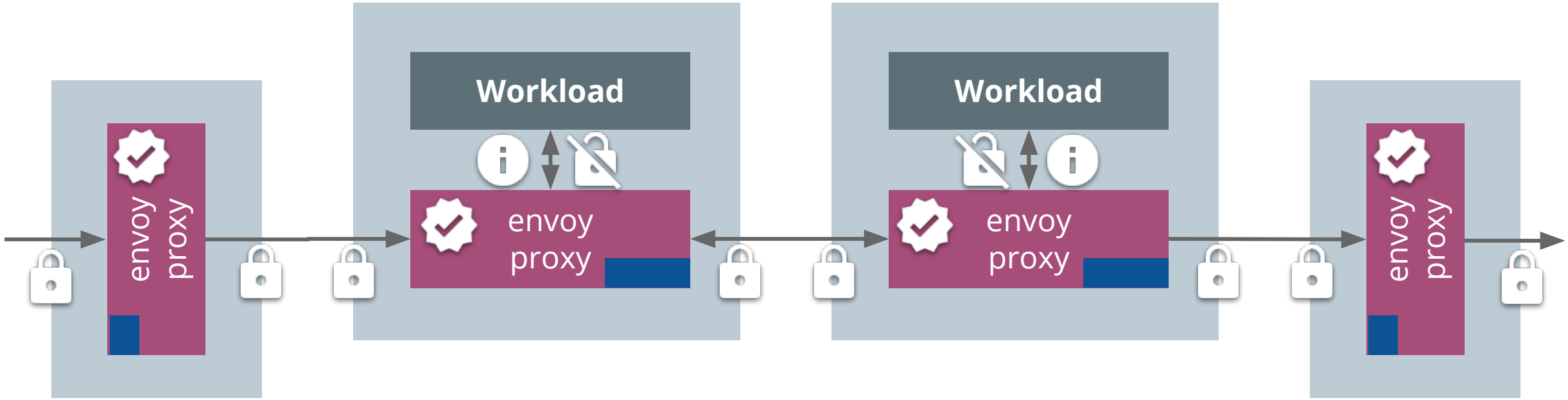
After the handshake, istio or better the envoy proxy (at both sides of the channel) knows who the **other** party is. With that information **decisions** can be made.

Envoy

Steps for introduction this to the platform

In the Istio ant colony, envoy is the worker ant

It sits between each all your network traffic, twice!


Workload
envoy
proxy
Workload
envoy
proxy
envoy
proxy
envoy
proxy

x-forward-client-cert proxy header indicating certificate information

because application want to know

x-forward-client-cert supported the following keys:

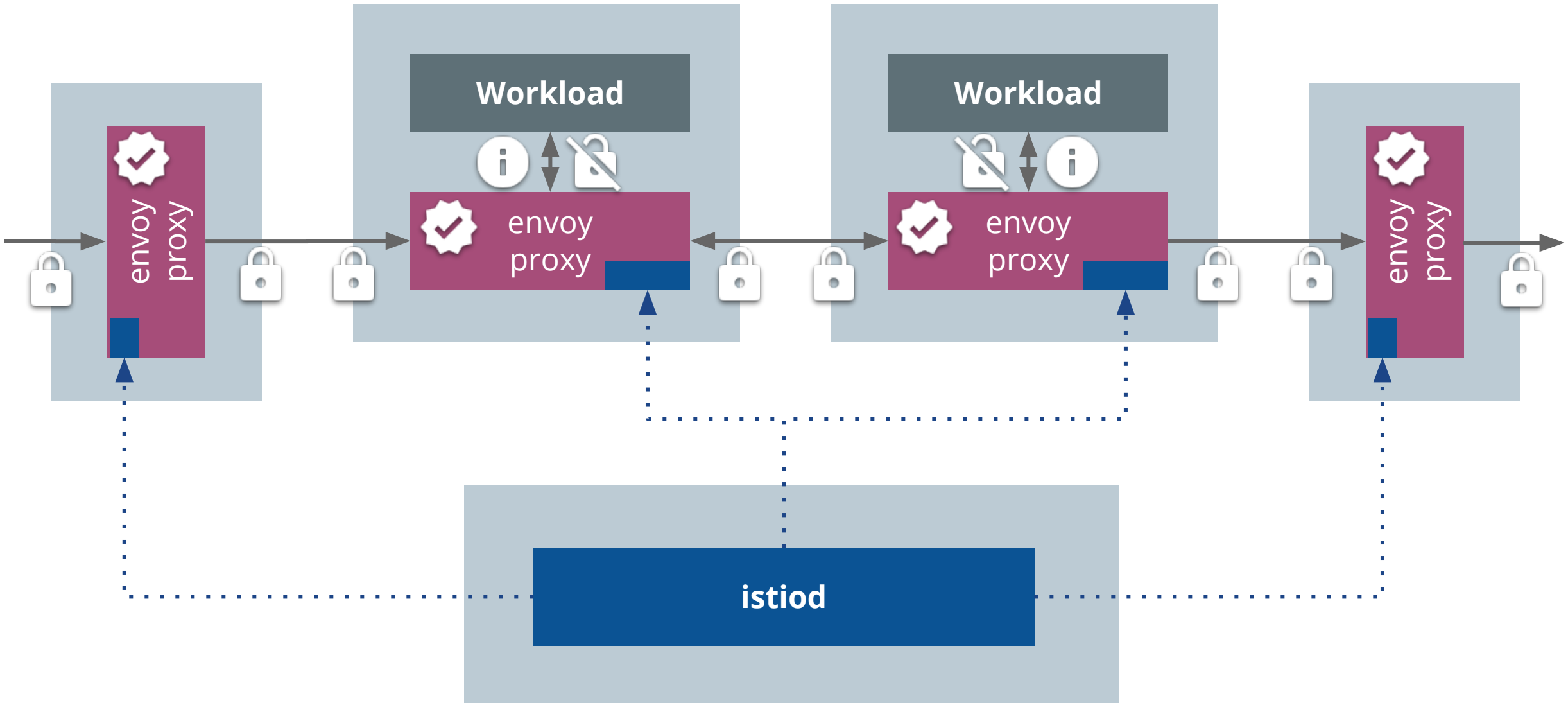
- **By:** our Subject Alternative Name (URI type)
- **Hash**: The SHA 256 digest of the current client certificate.
- **Cert**: The entire client certificate in URL encoded PEM format.
- **Chain**: The entire client certificate chain (including the leaf certificate)
- **Subject**: The Subject field of the current **client** certificate.
- **URI**: The URI type SAN field of the current **client** certificate.
- **DNS**: The DNS type SAN field of the current **client** certificate.

x-forward-client-cert example:

```
By=spiffe://cluster.local/ns/my-namespace/sa/my-service [*1]
;
Hash=cbb4cb46004bdbce15856...78e823fcedfad364 [*2]
;
Subject=\"\" [*3]
;
URI=spiffe://cluster.local/ns/client-namespace/sa/client-service [*4]
```

Istio

Steps for introduction this to the platform

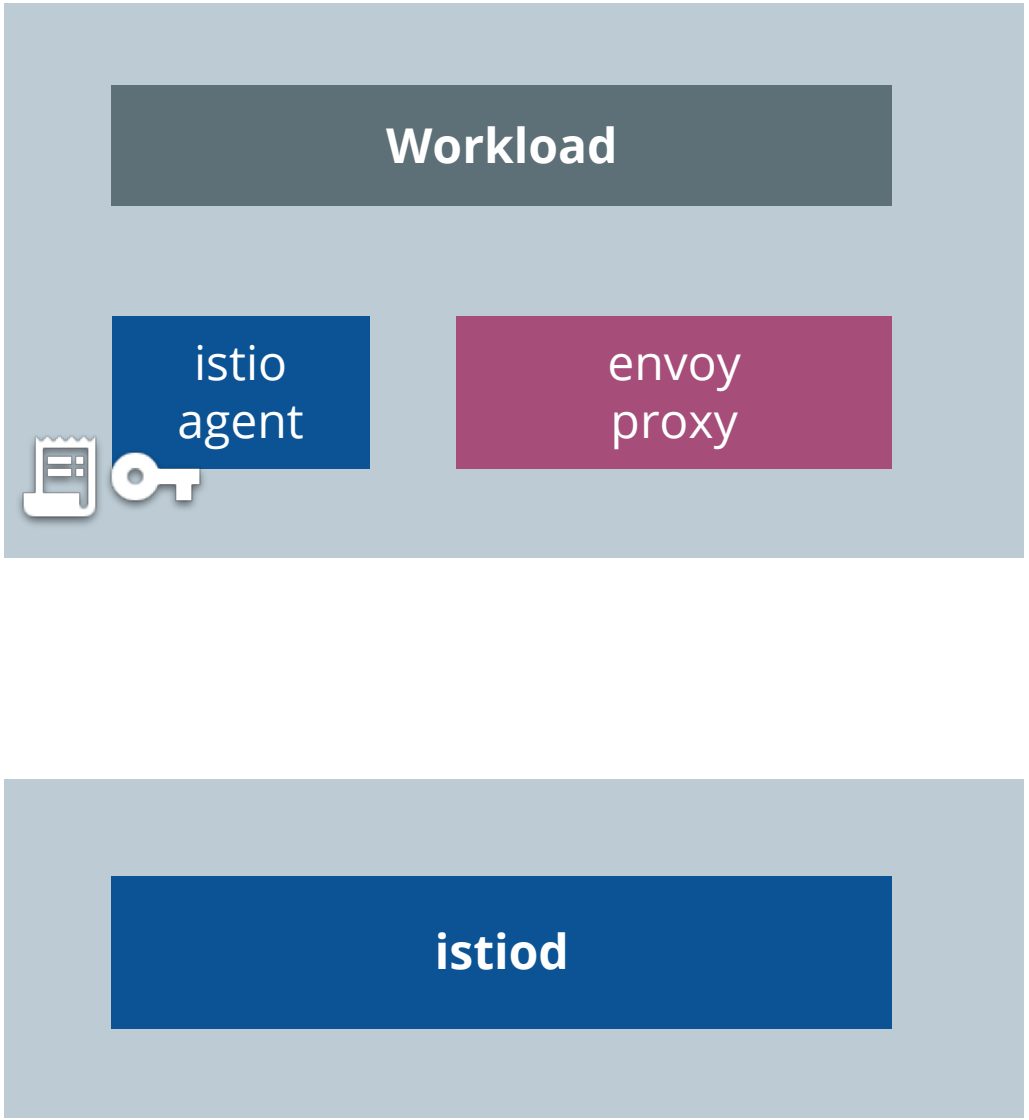

Workload
Workload
envoy proxy
envoy proxy
envoy proxy
envoy proxy
istiod

Bringing it all together, how?

Bringing istio’s control plain into the mix

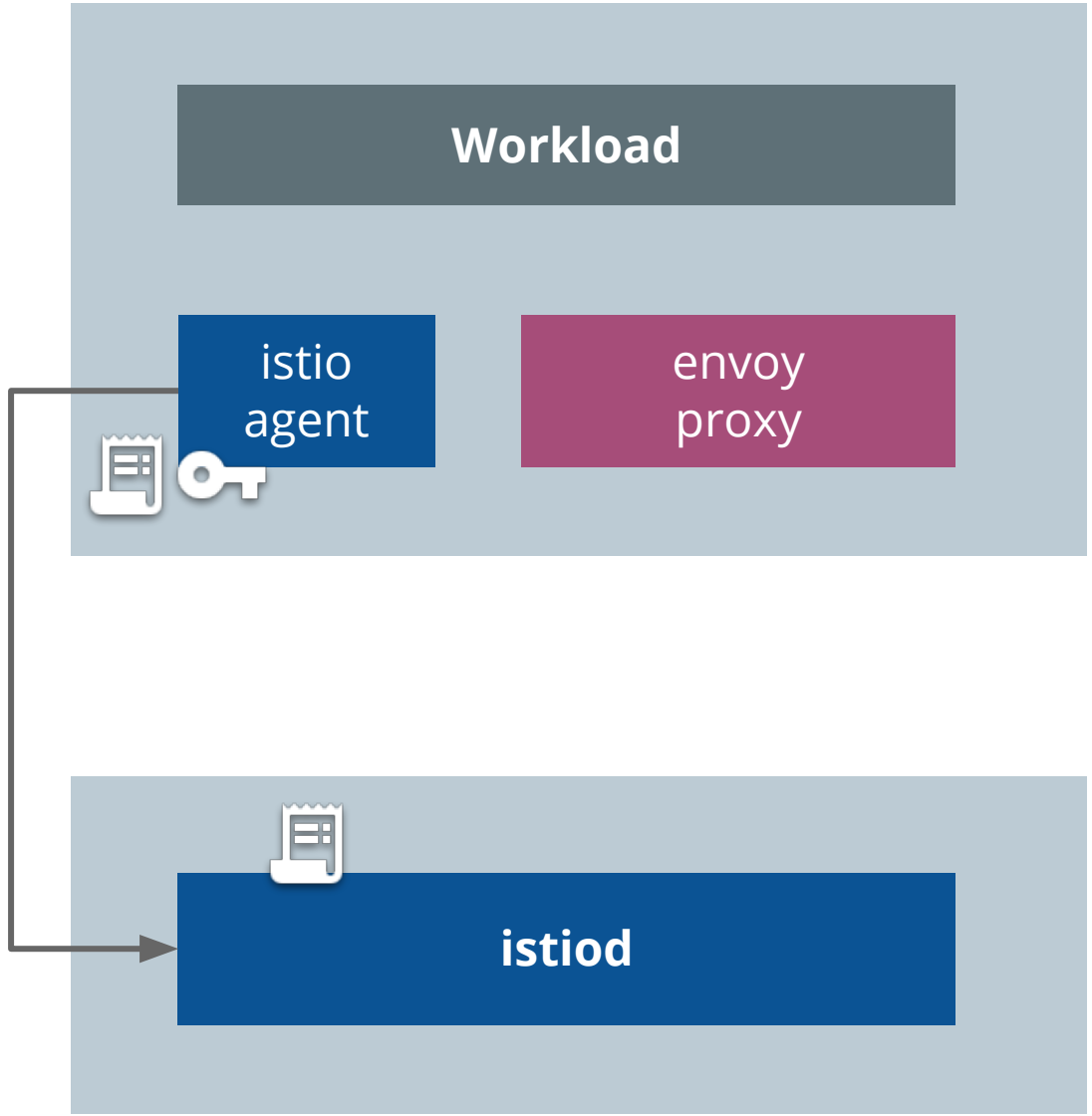
Identity Management

As the **istio-agent** start, along with the pod, a key and CSR (certificate signing request) is generated.



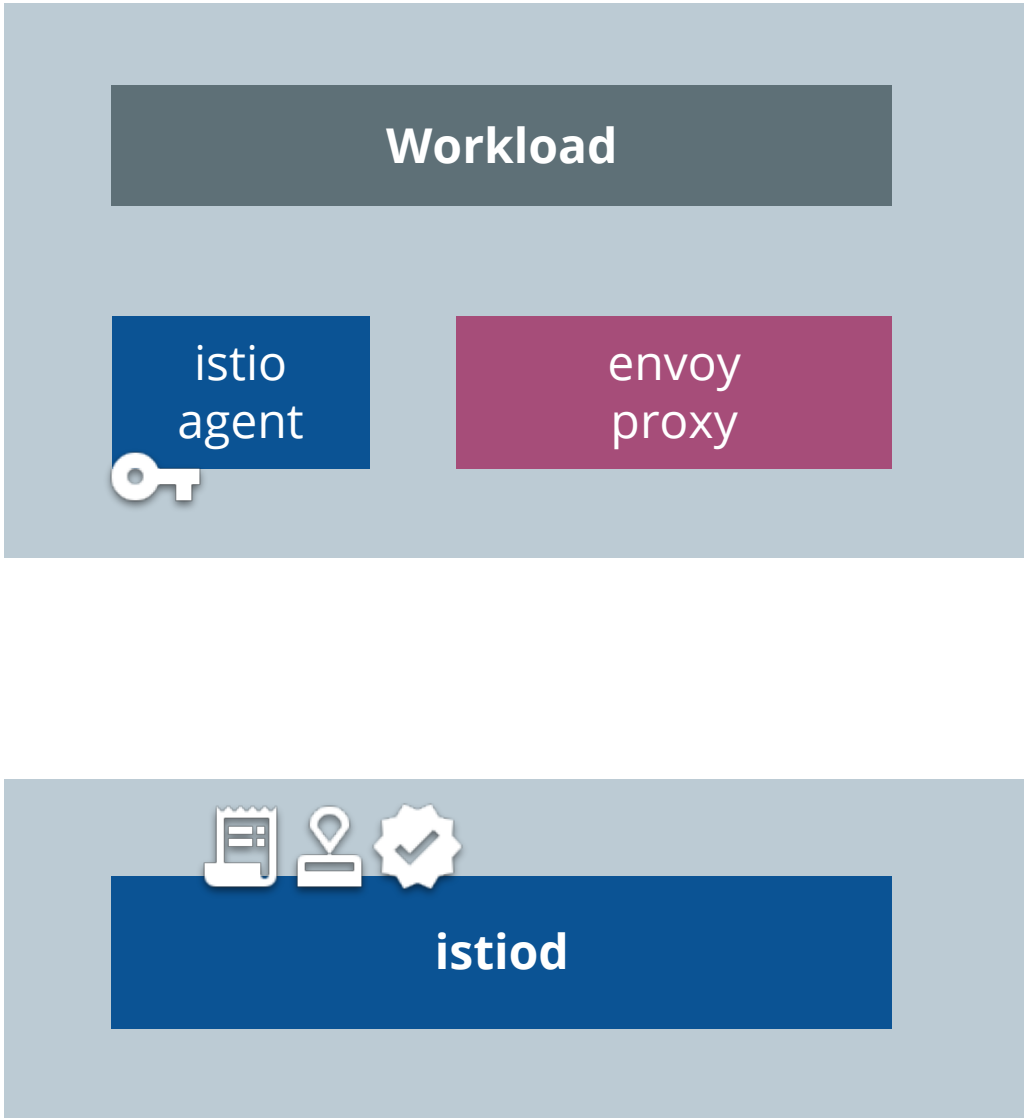
Identity Management

The CSR is send to the **istiod**. A CSR already contains most of the information that should be in certificate. Information like **public key** and **SPIFFE ID** are the most important once.



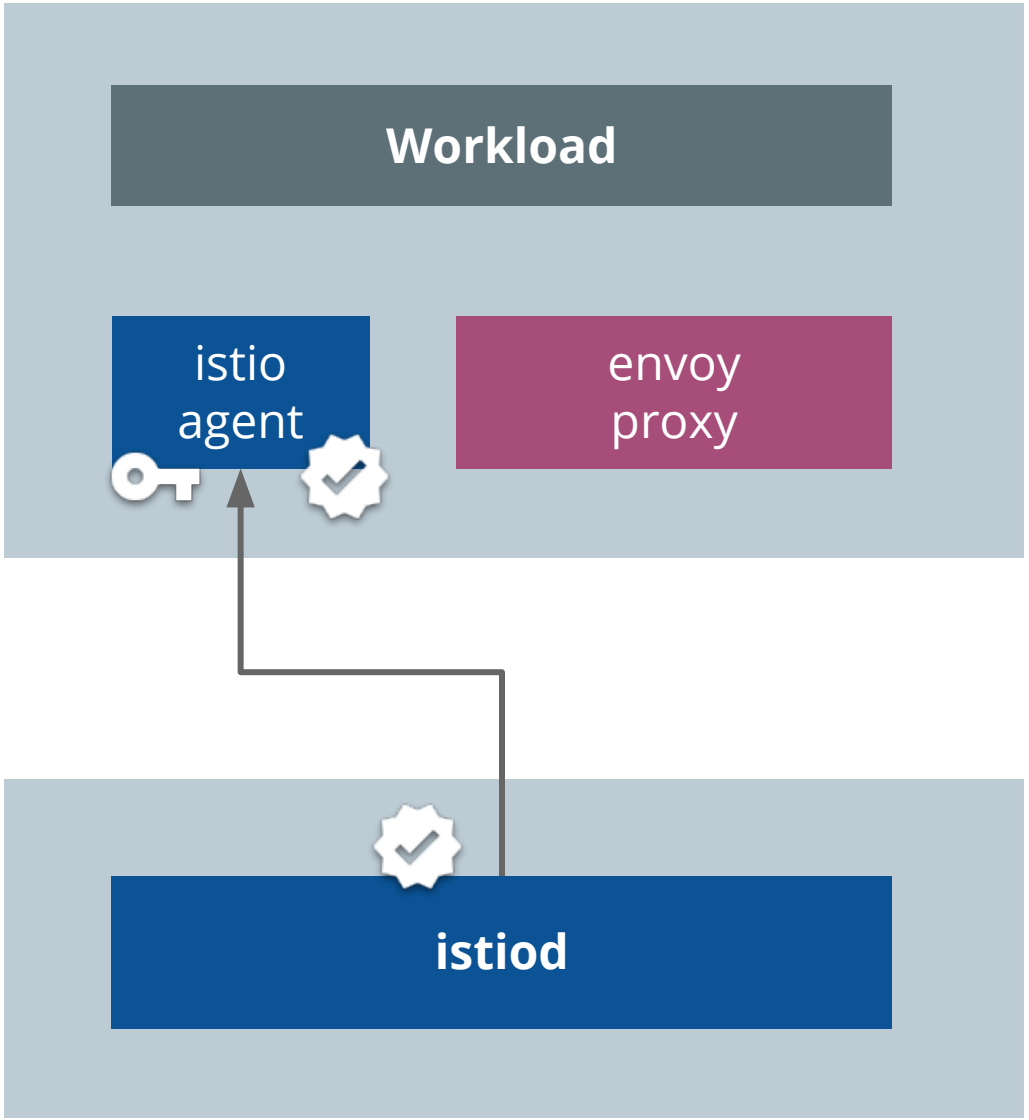
Identity Management

If istiod is certain that the CSR is coming from the correct agent it will **create the certificate** and signs it. Validation of agent is done though use of the kubernetes token passed along the CSR request.



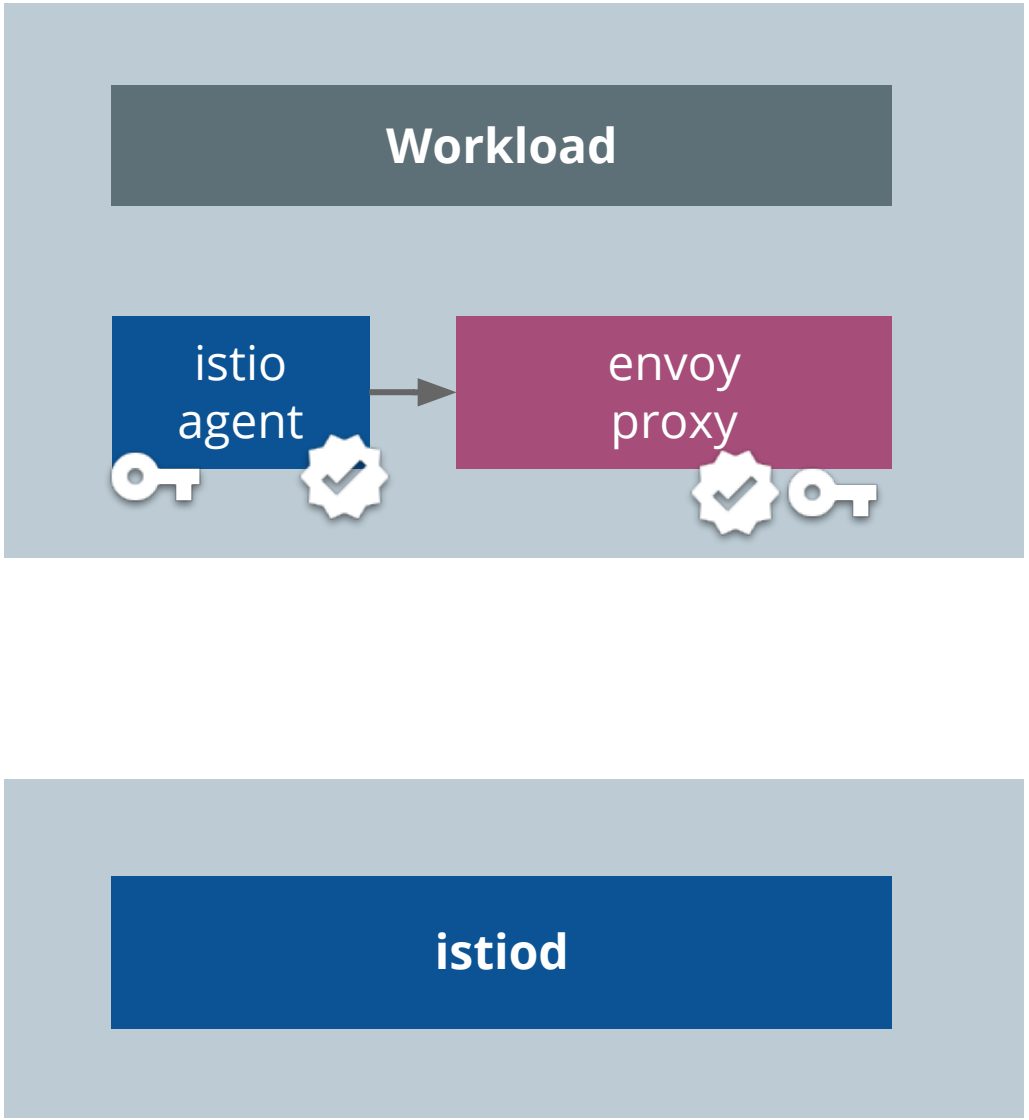
Identity Management

The certificate is send back to **istio-agent**, the CSR is no long needed.



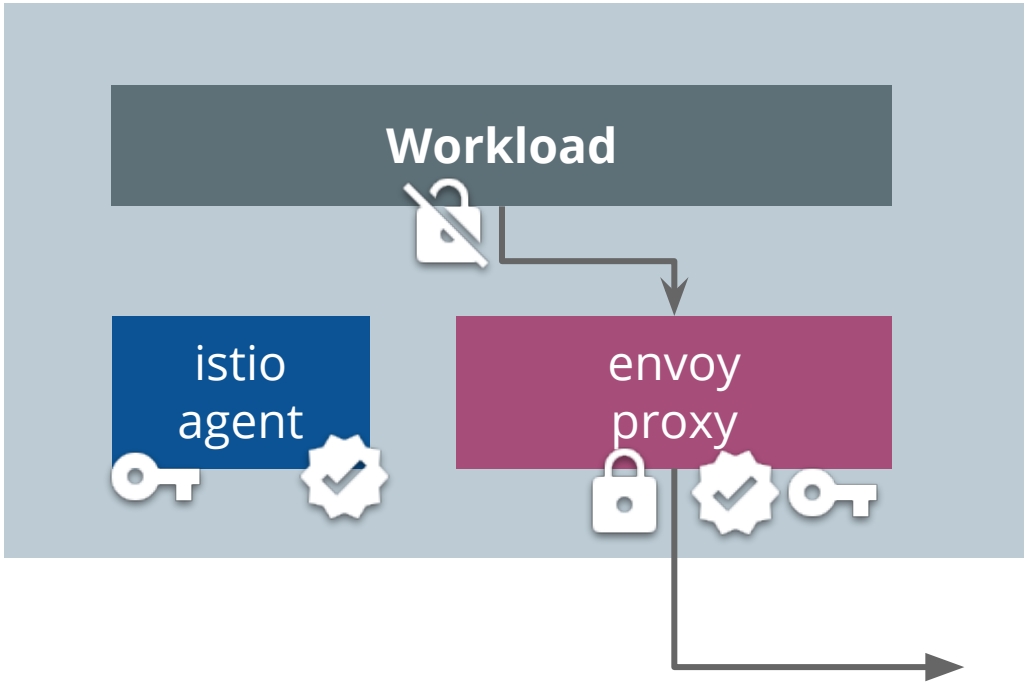
Identity Management

Through **SDS**, an envoy "**Secure Discovery Service**" protocol the private key and certificate is send to the proxy. Now the proxy is ready to prove it's identity.



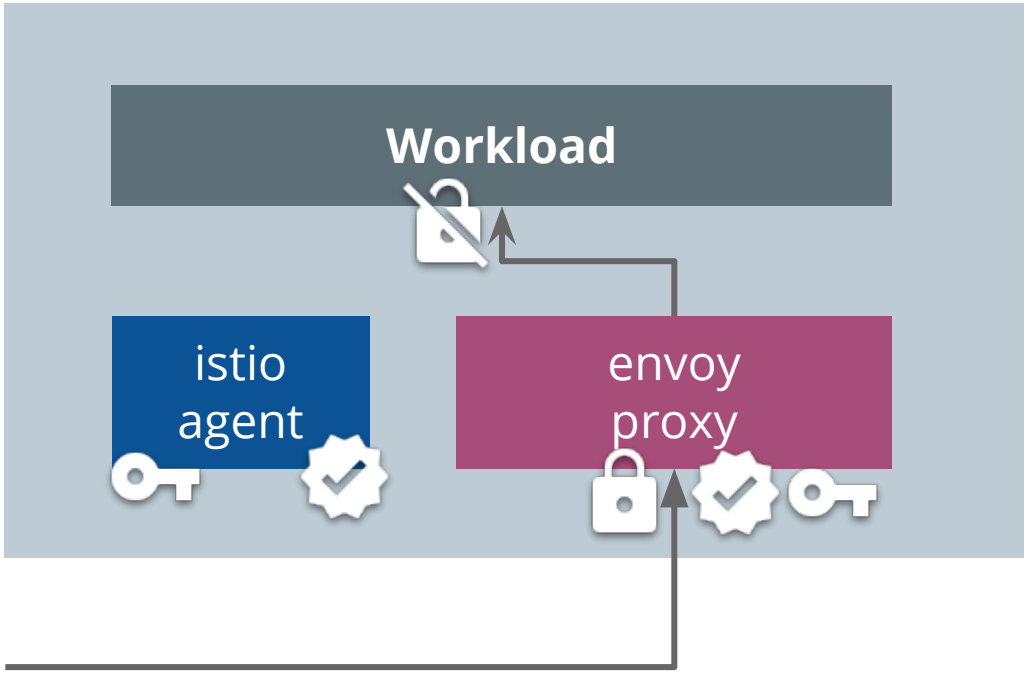
Identity Management

Now the workload can pretend that it lives in an unsecure world and start communicating unencrypted. The proxy will **secure the connection** using the certificate/key pair.



Identity Management

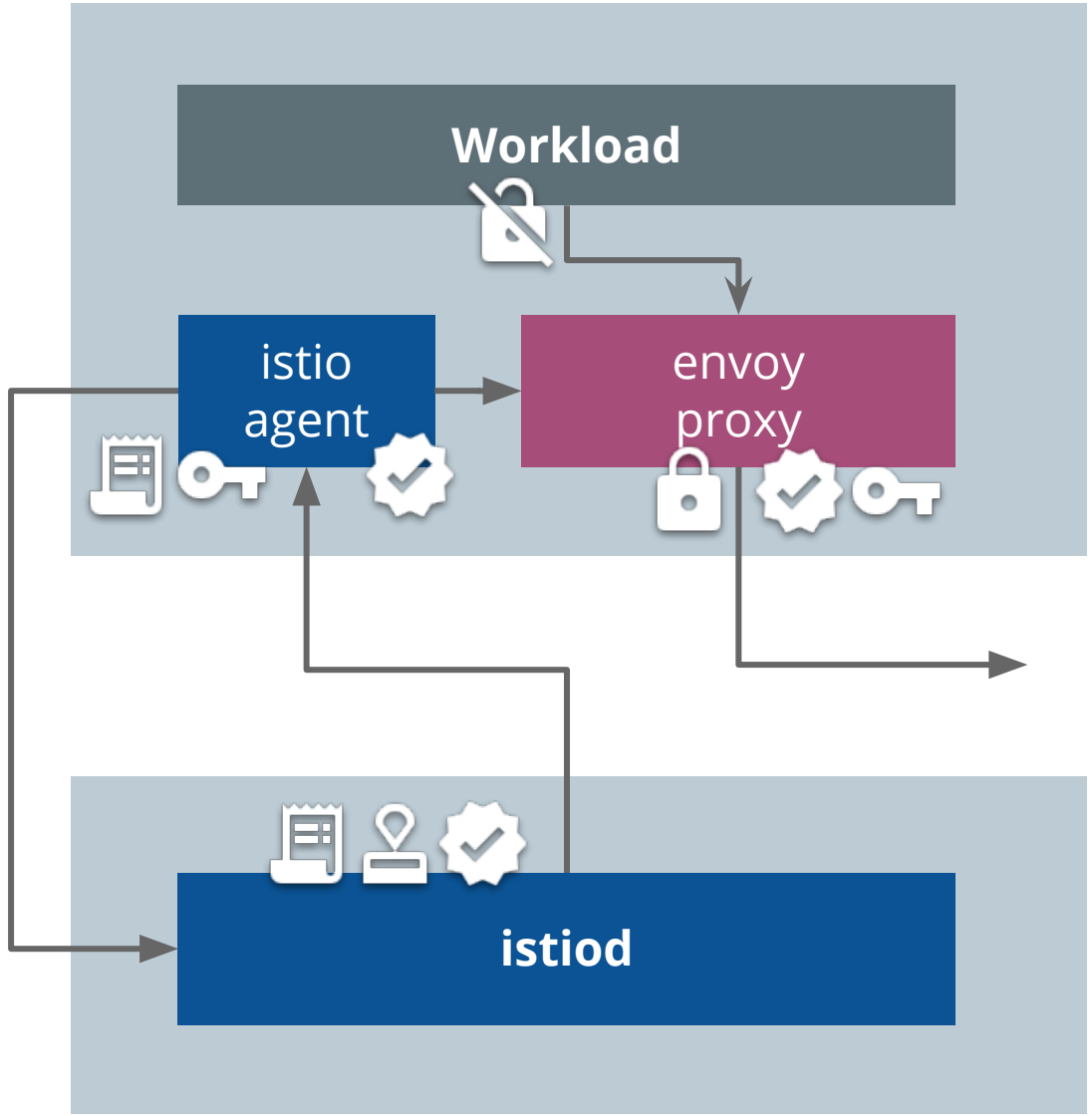
Remember, this is **mutual** TLS (mTLS): The same mechanism is used for clients and servers. The control plan is not required until the key and certificate needs rotation.



Identity Management

Key takeaways

- Identity is established at pod **startup**
- Uses Kubernetes service account **token**
- No control plane needed after identity has been established



Application Architecture

Steps for introduction this to the platform

Let’s differentiate between Peer- and Request Authentication

Authorization decisions can be taken by combining both

Peer Authentication

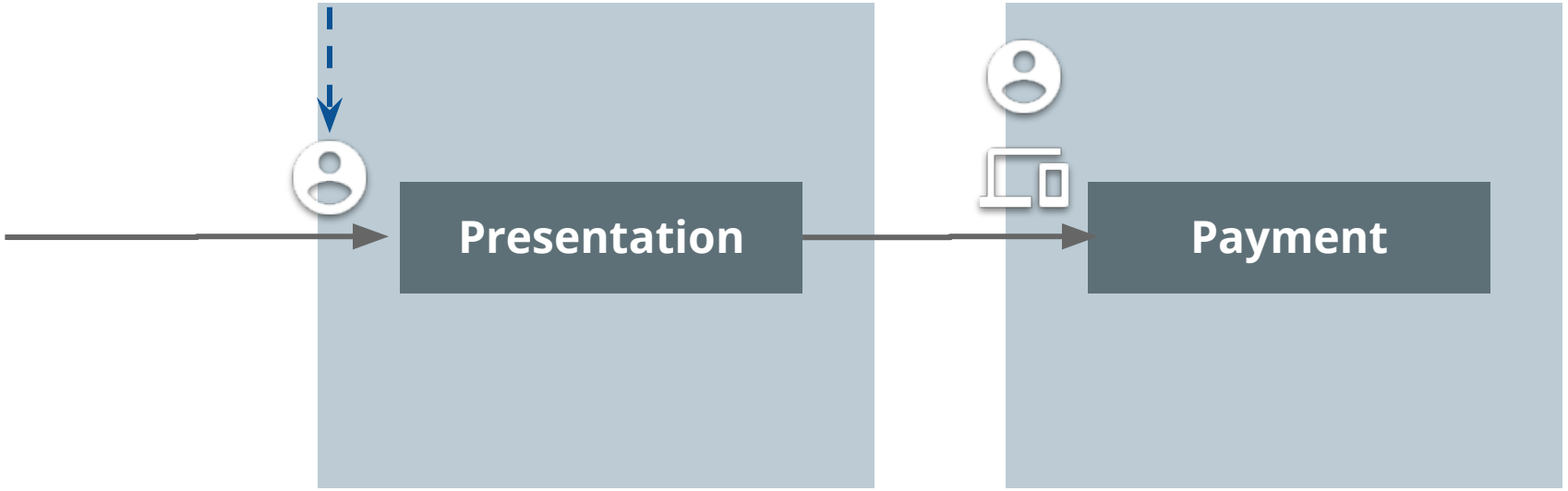
Peer authentication, the mechanism istio manages best. This is what we've been talking about during this talk. Who is the workload that is talking to us? Generally we're mostly talking about **services**, not persons.

Request Authentication

Authentication on each individual request. Requests can be multiplexed over a single pipe, that has been authenticated through peer authn. Best reserved for authenticating **persons**.

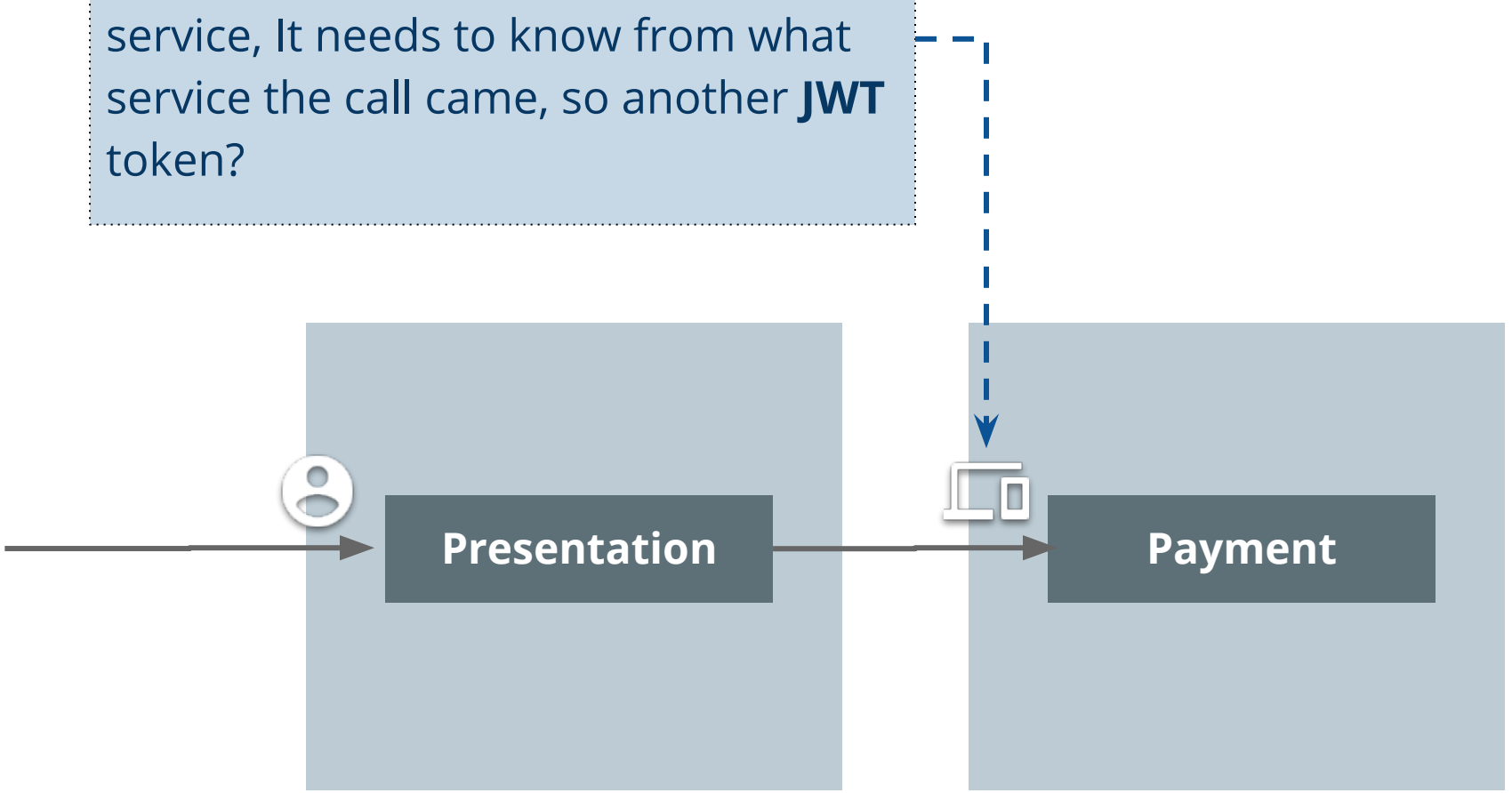
None mesh authn

A user is authenticated via a standard **JWT** token to a presentation layer



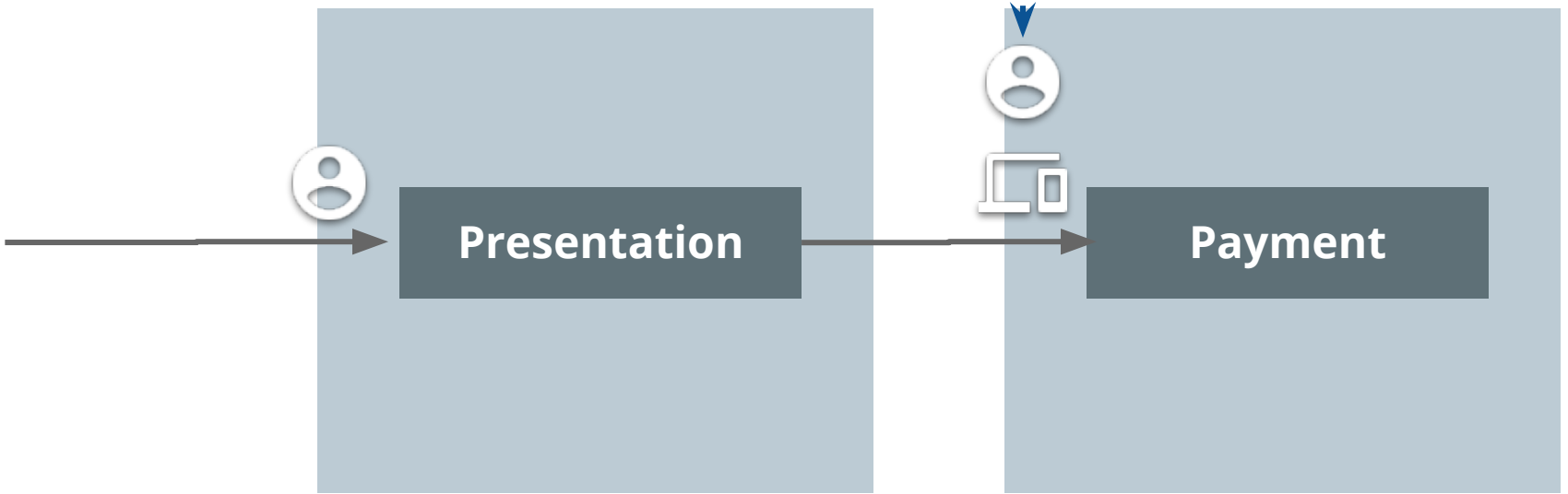
None mesh authn

The presentation layer needs some information from the payment service, It needs to know from what service the call came, so another **JWT** token?

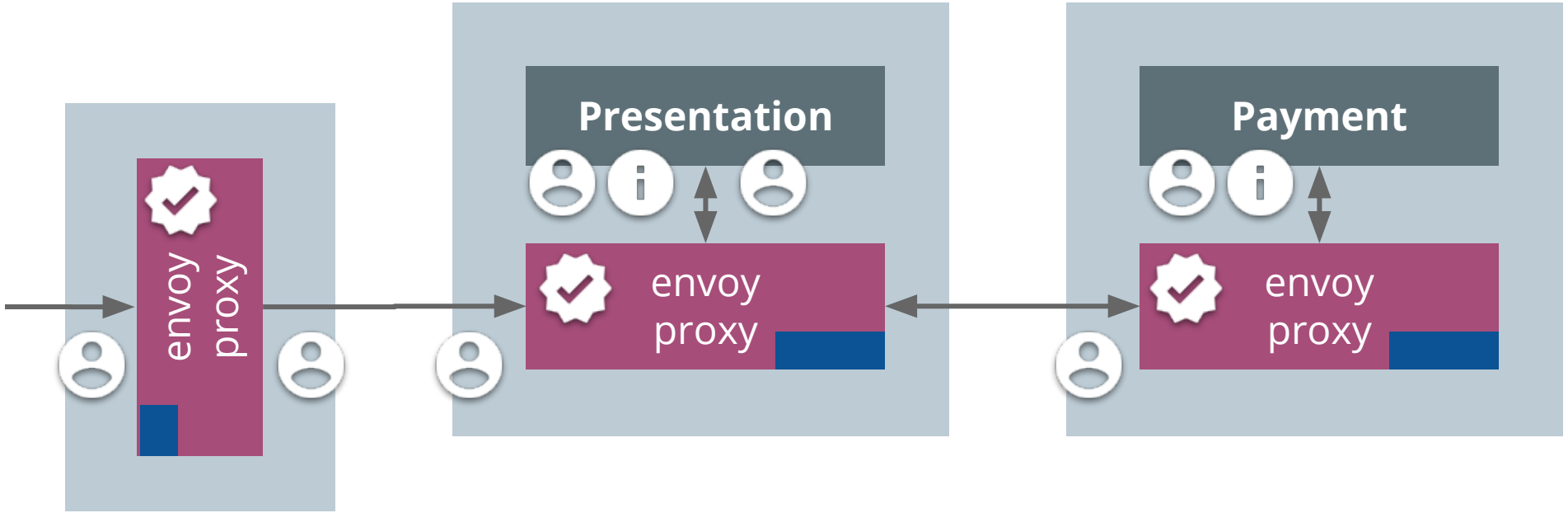


None mesh authn

Looks easy... but **how** do we manage the keys for the JWT tokens? **How** to encode the original caller? Encode it in another header?!

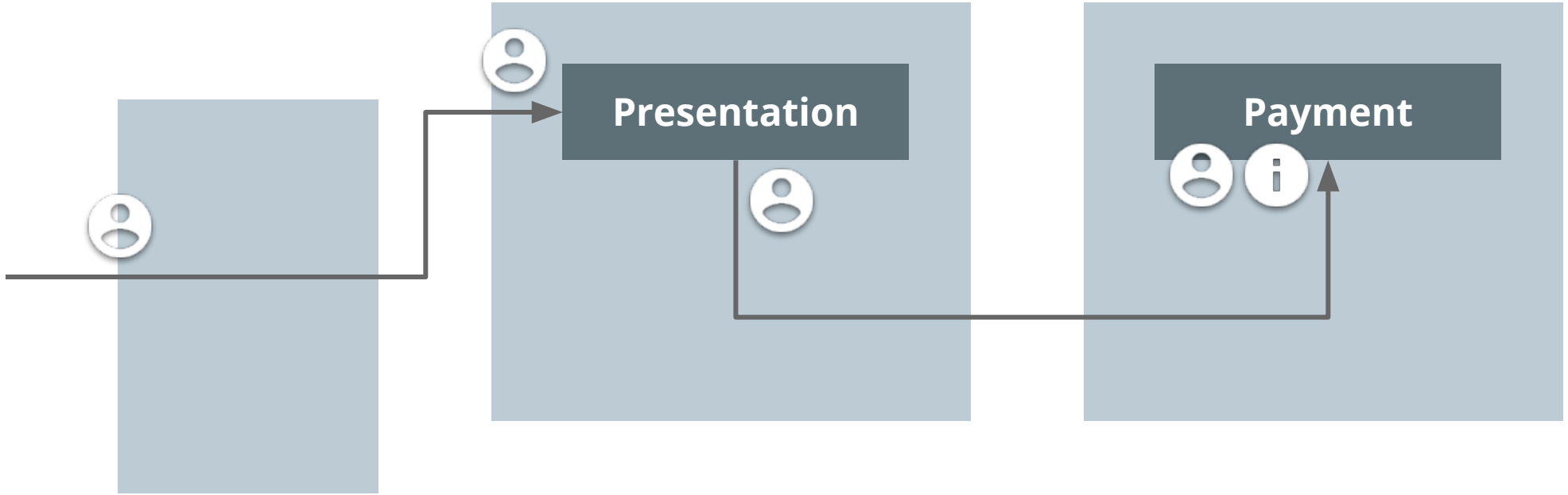


Mesh authn



Granted, the mesh picture looks **scary** in comparison. But...

Engineers view

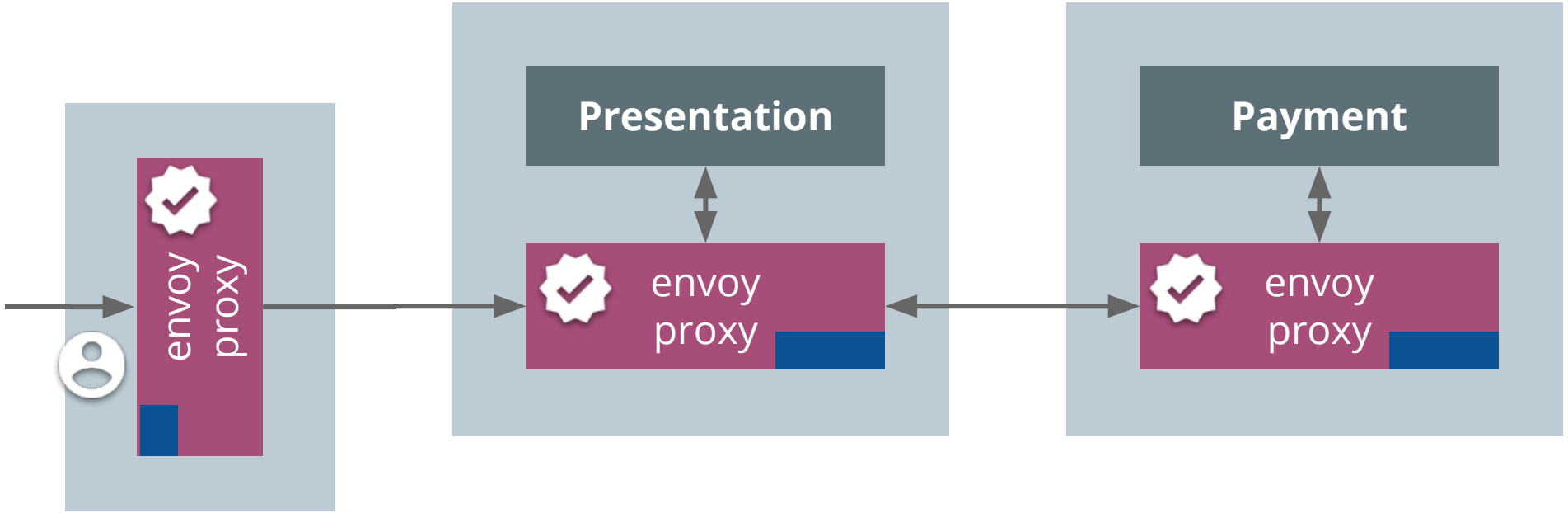


... what a developer needs to worry about is only the services that he writes and “handle” the **JWT**. If the source workload is important, the application can check the **x-forward-client-cert** header.

Full end-to-end example, using Request- and Peer Authn

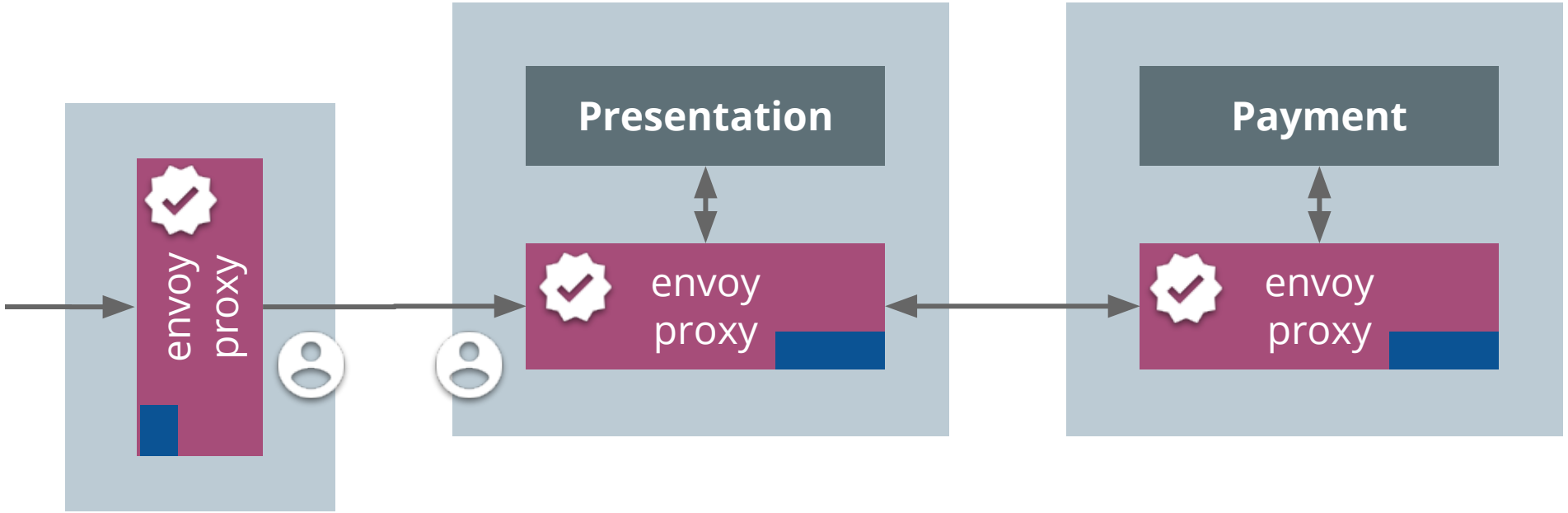
Bring it all together

Mesh Auth



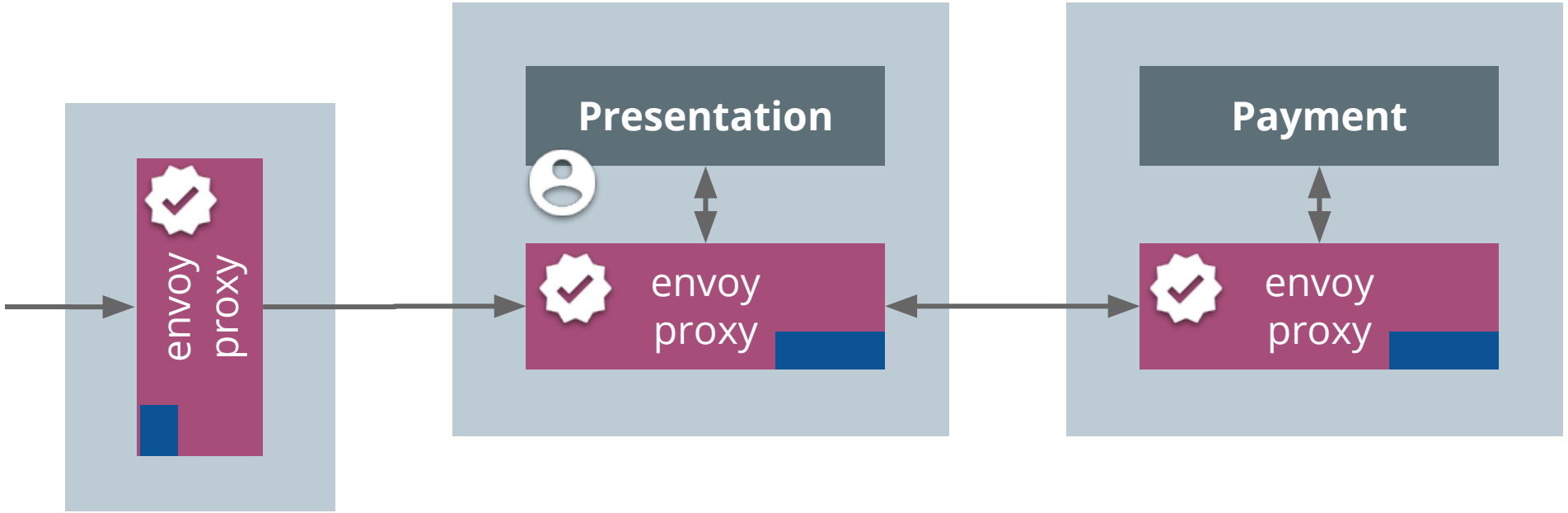
The user crosses the mesh boundary, walks in through the gateway with it's JWT token.

Mesh Auth



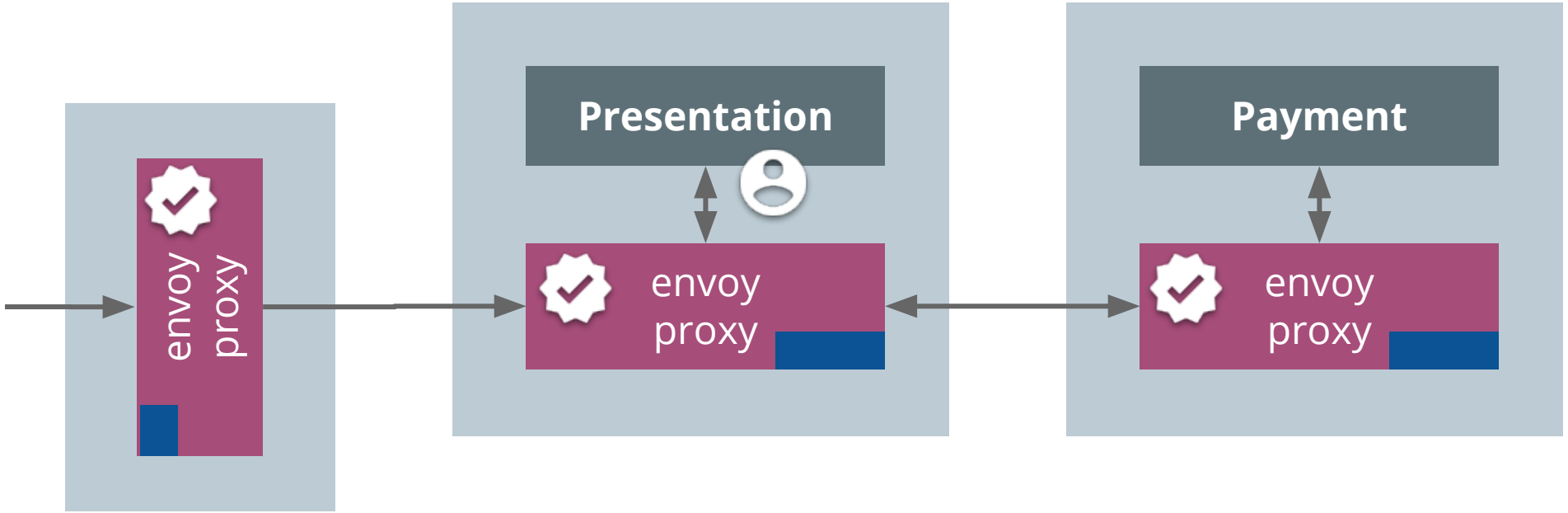
Although you can expose workloads directly you can leverage the gateway to manage public TLS termination, and switch to mesh mTLS. The JWT passes right through.

Mesh Auth



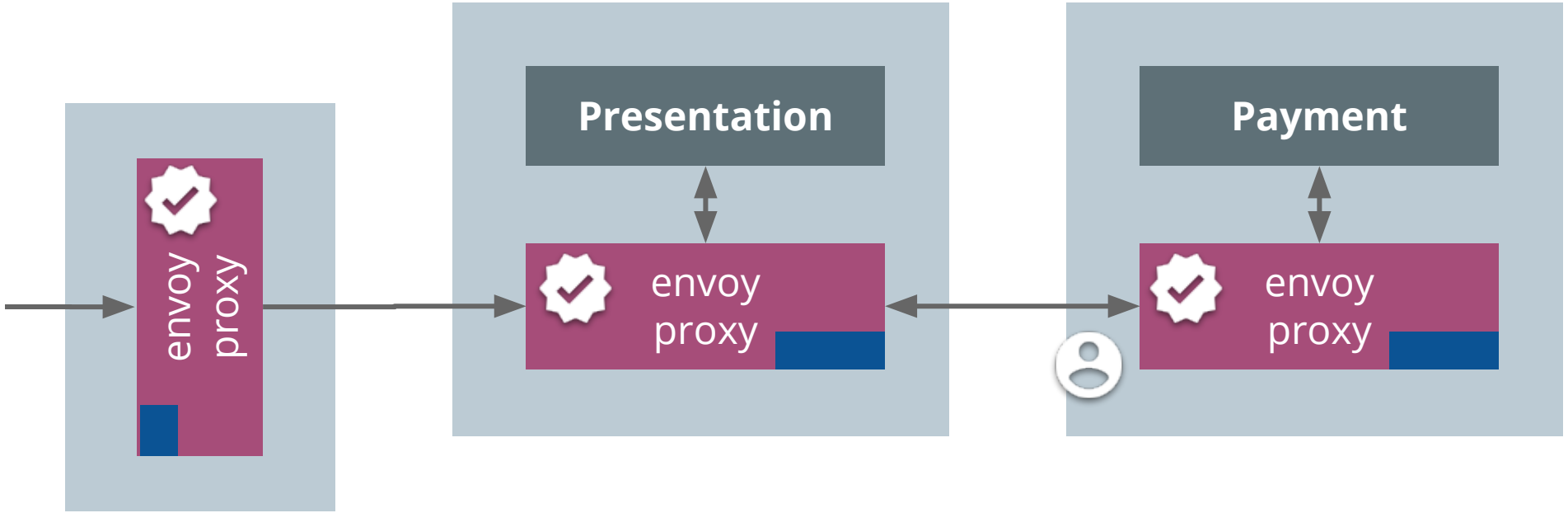
Trust exist though every service in the mesh, including the gateway. Routing to the presentation layer. The presentation layer can use the JWT to validate the user.

Mesh Auth



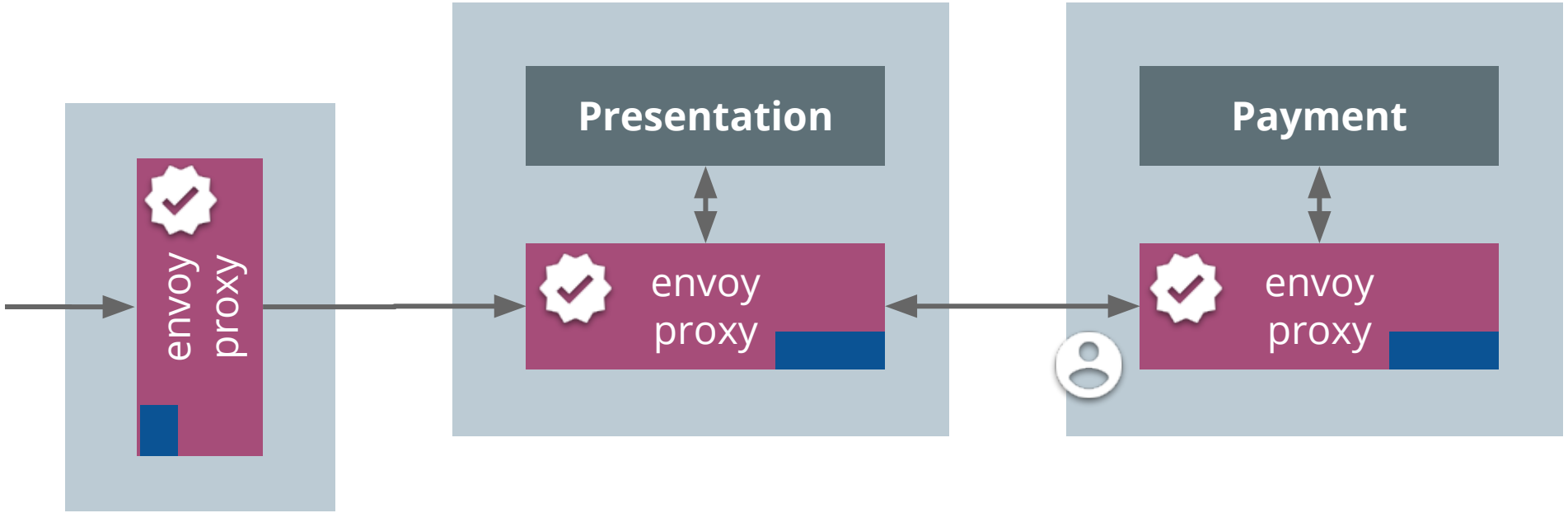
The presentation layer needs some information from the payment service. It does that call to the payment service and passes the original JWT token.

Mesh Auth



Opportunity exist to lock down access only from the presentation layer though **policies (peer)** defined by istio.

Mesh Auth

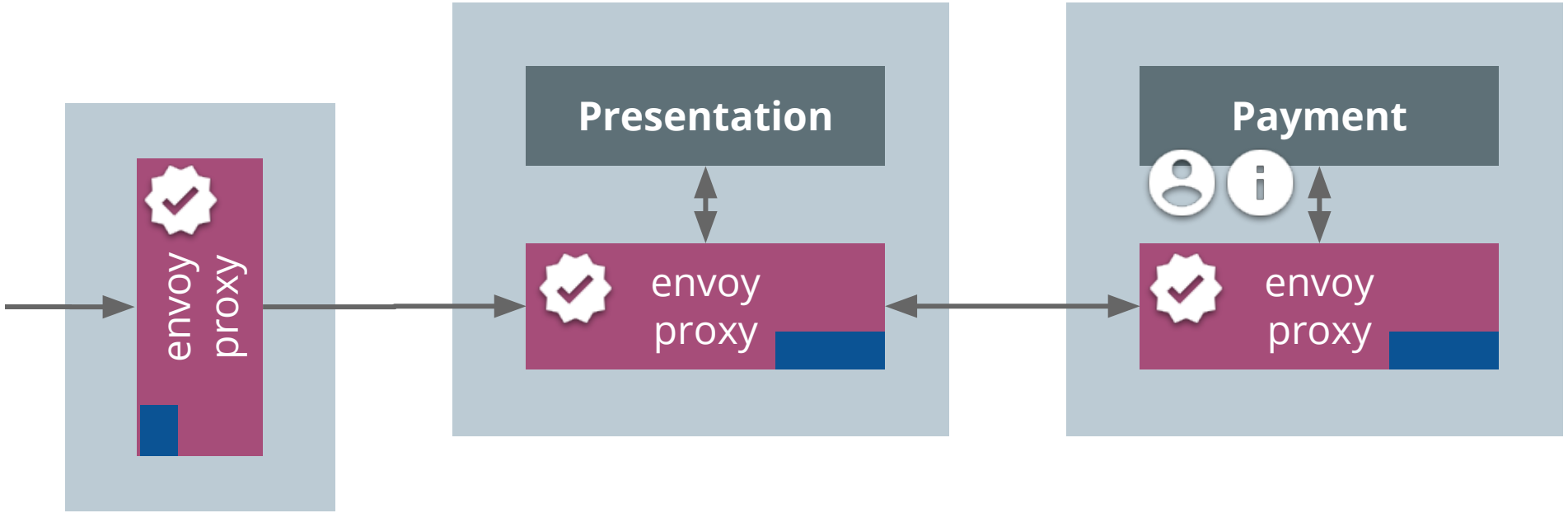


Policies (request) for the JWT tokens can also be pushed to the mesh. All with application independent **audit trail**. Ideal for the polyglot world.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
 action: ALLOW
 rules:
 - from:
   - source:
       principals: ["cluster.local/ns/default/sa/sleep"]
   to:
   - operation:
       methods: ["GET"]
   when:
   - key: request.auth.claims[iss]
     values: ["https://accounts.google.com"]
```

OK, got to show one policy file, too keep it an Istio talk. This is taken right out of the documentation and shows **combining** both Peer- and Request Authentication and taking Authorization decisions.

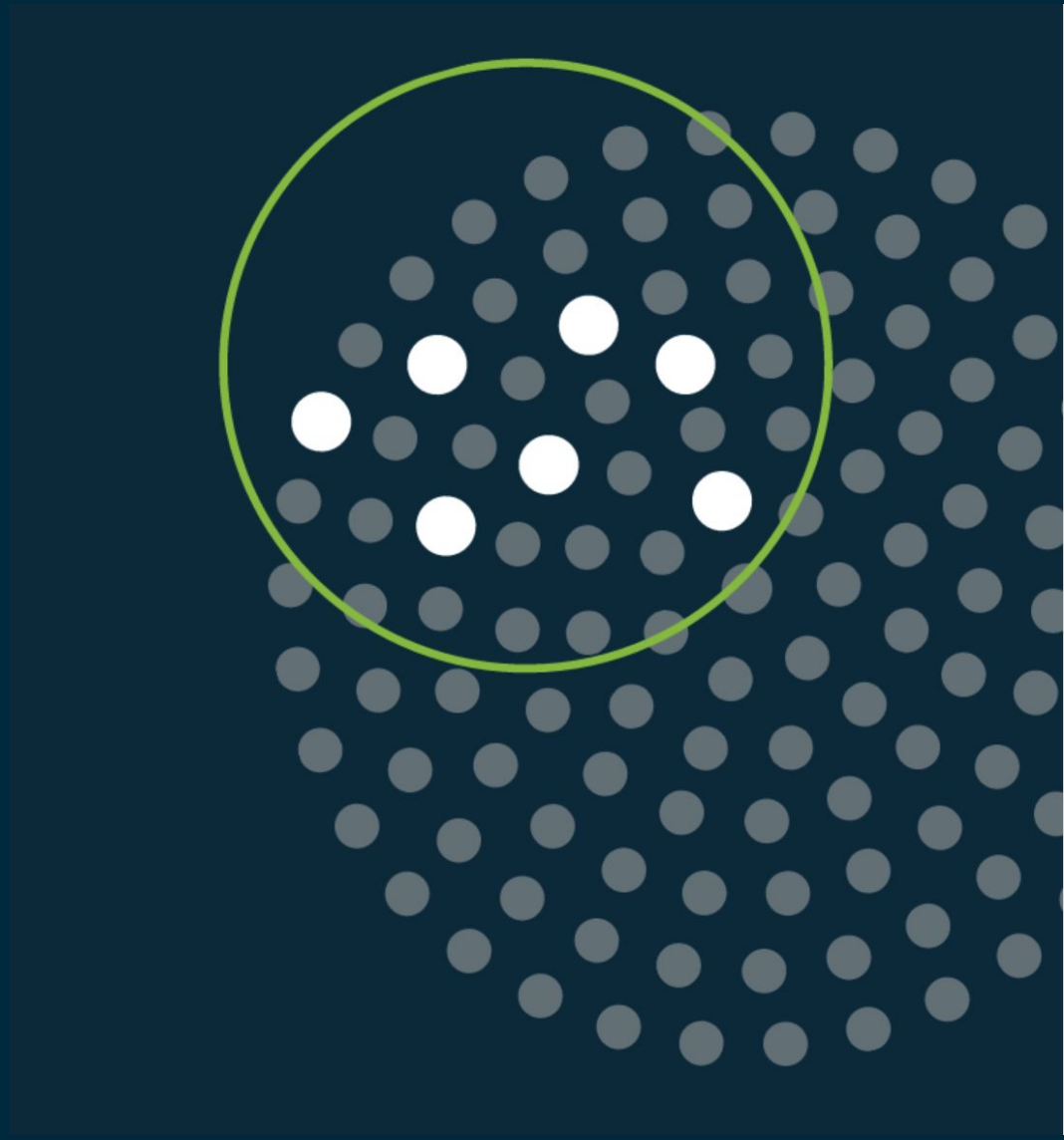
Mesh Auth

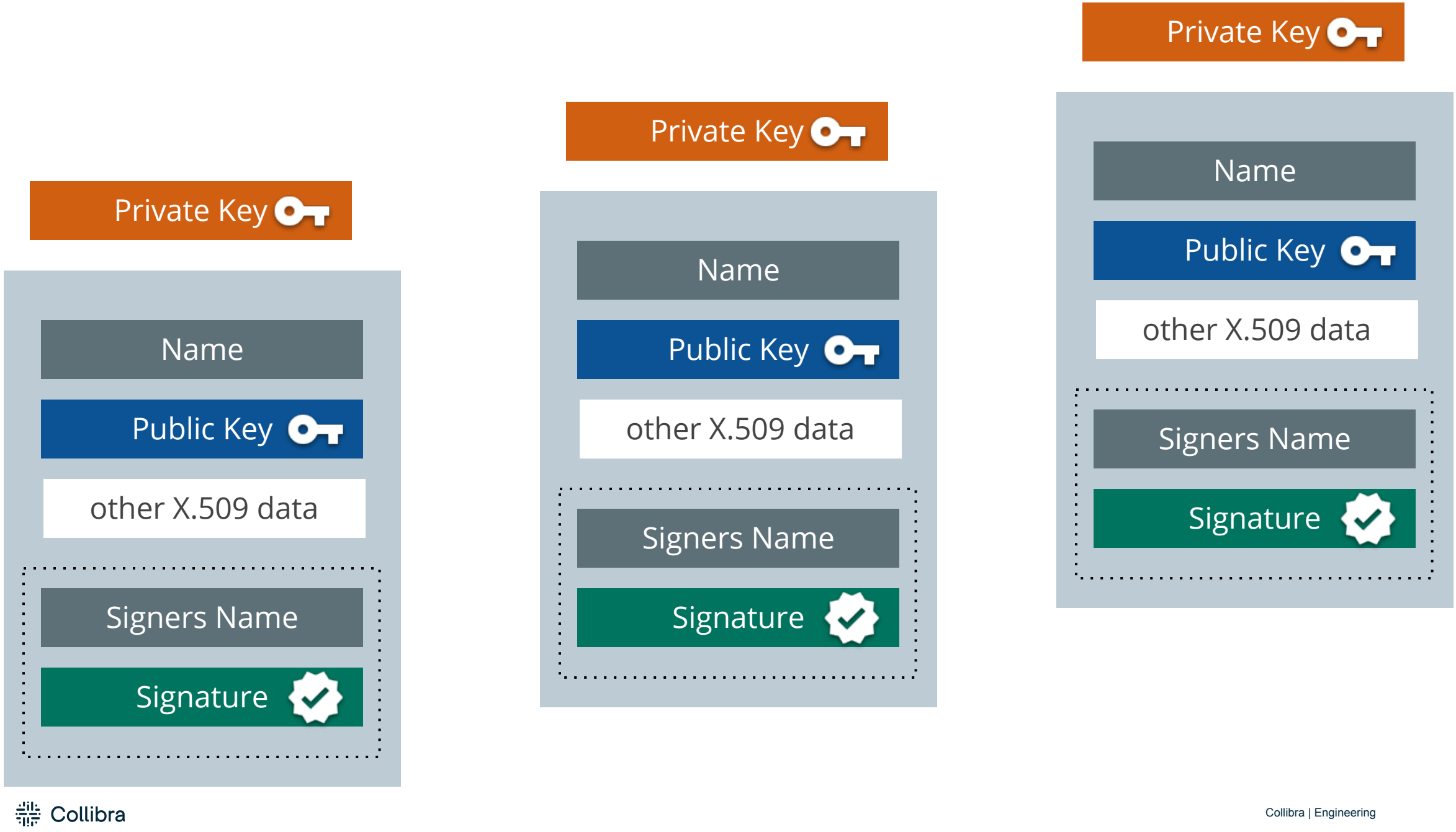


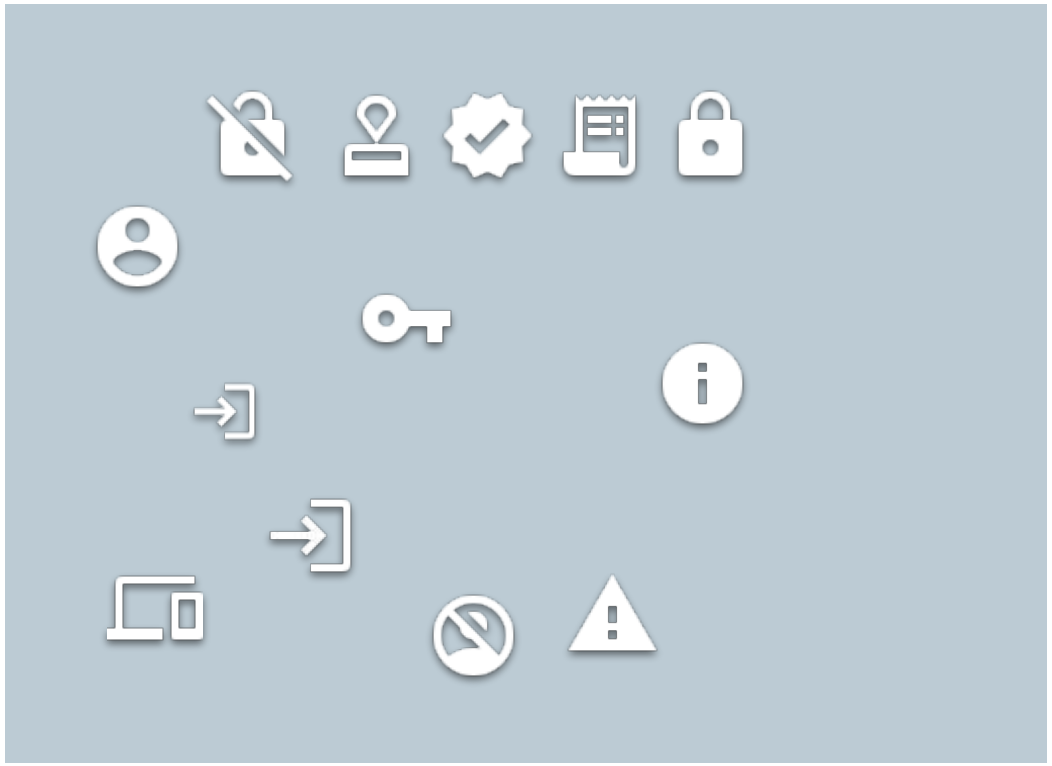
Both the x-forward-client-cert header and the JWT token in the authentication header can be used to make informed decisions in the **application**.

Thank you

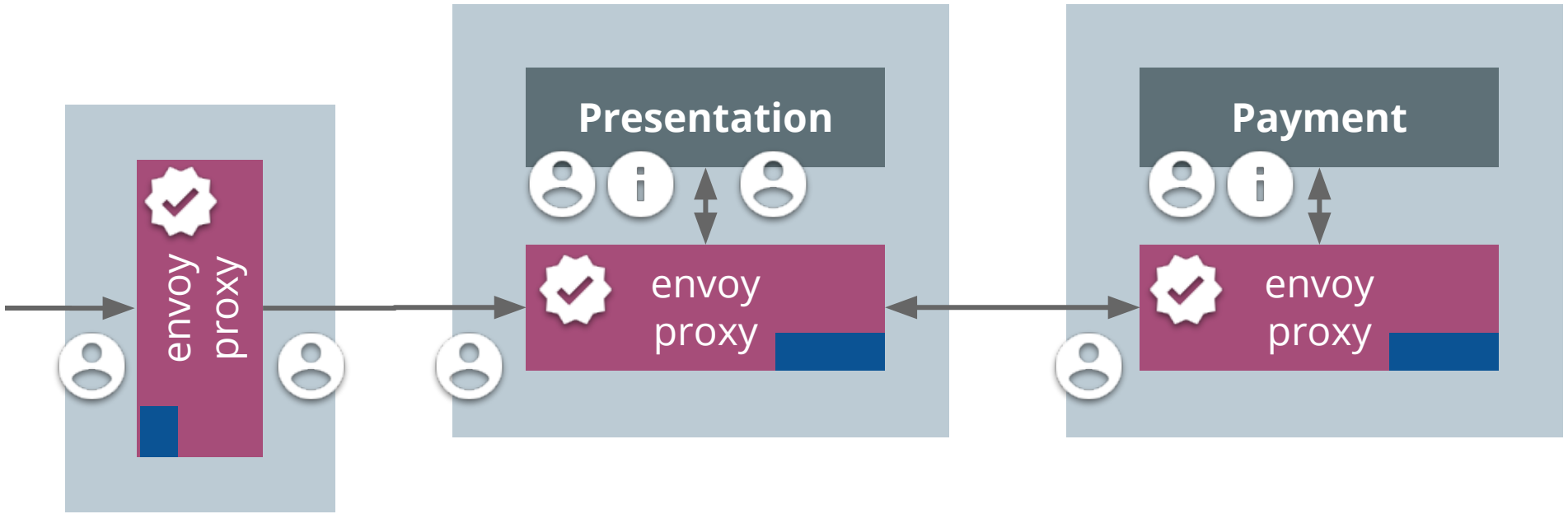
Questions?




Private Key
Name
Public Key
other X.509 data
Signers Name
Signature
Private Key
Name
Public Key
other X.509 data
Signers Name
Signature
Private Key
Name
Public Key
other X.509 data
Signers Name
Signature



Mesh Auth



text