

Getting mutually familiar with mTLS

Alan Scherger @flyinprogrammer

Agenda

- mTLS
 - How did we get here?
 - What could go wrong?
- Make and Investigate Certificates
- mTLS with Zookeeper
- Certificate Cross-Signing/Rotation for Great Good!

Tools we'll use:

- [FiloSottile/mkcert](#)
- [Nginx](#)
- [OpenSSL](#)/[LibreSSL](#) - spoiler
- [Zookeeper](#)

Things you will learn:

- TLS basics - the why and how.
- Lessons learned from others using TLS.
- Local development with certificates is delightful.
- Investigating certificates is easy.
- Writing code that uses certificates takes effort.
- Cross-signing and rotation of certificates requires thoughtful automation.

SSL Certificate

TLS Certificate

HTTPS Certificate

X.509 v3 Certificate

What could go wrong?

TL;DR - A 🙌 Lot 🙌.

Datadog	May, 30 2020
CloudFoundry CredHub	Feb 12, 2020
Microsoft Teams	Feb 3, 2020
HashiCorp Nomad	Jan 28, 2020
Etc	Jan 2, 2019
LinkedIn	May 2019, Nov 2017
Ericsson	Dec 6, 2018
Docker	Sept 10, 2018
Microsoft Azure Storage	Feb 22, 2013

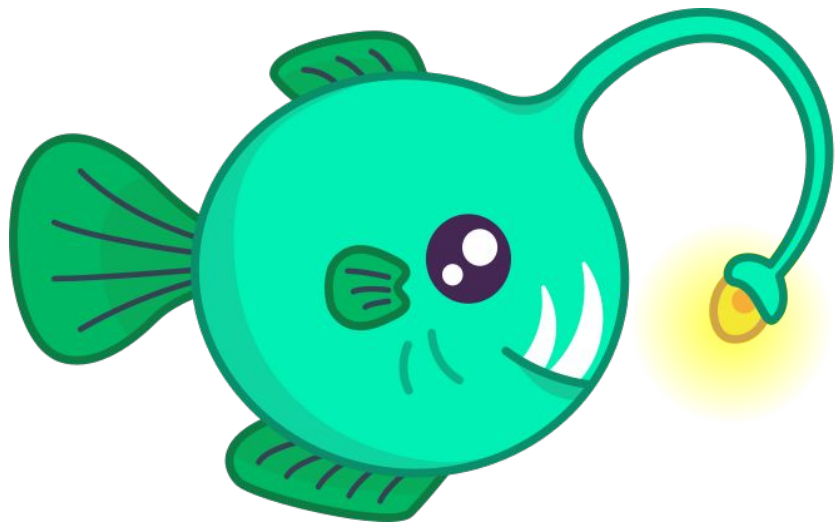
TL;DR - A 🙌 Lot 🙌.

Datadog	May, 30 2020
CloudFoundry CredHub	Feb 12, 2020
Microsoft Teams	Feb 3, 2020
HashiCorp Nomad	Jan 28, 2020
Etc	Jan 2, 2019
LinkedIn	May 2019, Nov 2017
Ericsson	Dec 6, 2018
Docker	Sept 10, 2018
Microsoft Azure Storage	Feb 22, 2013

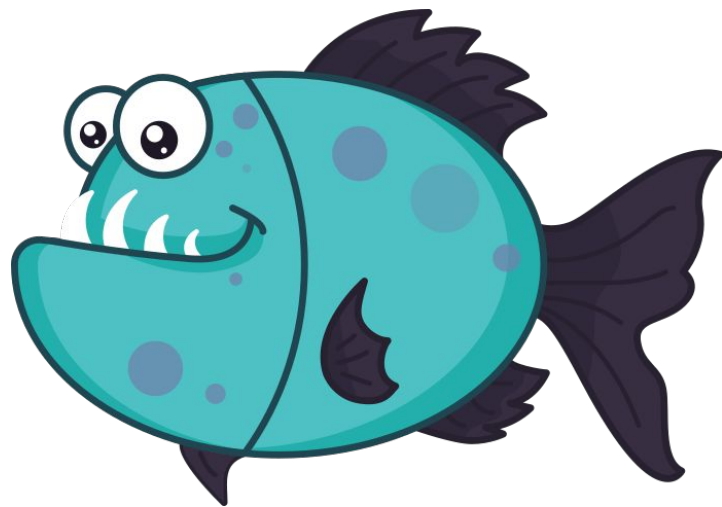
How did we get here?

Alice and Bob want to communicate...

Alice

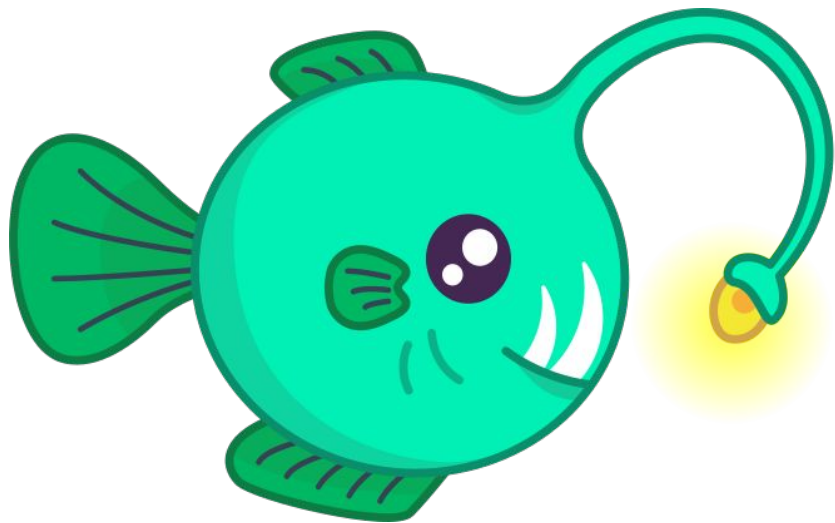


Bob

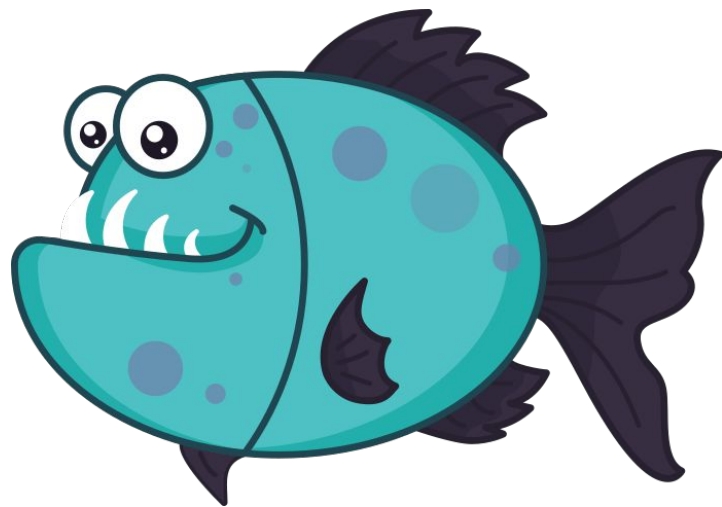


They simply do.

Alice



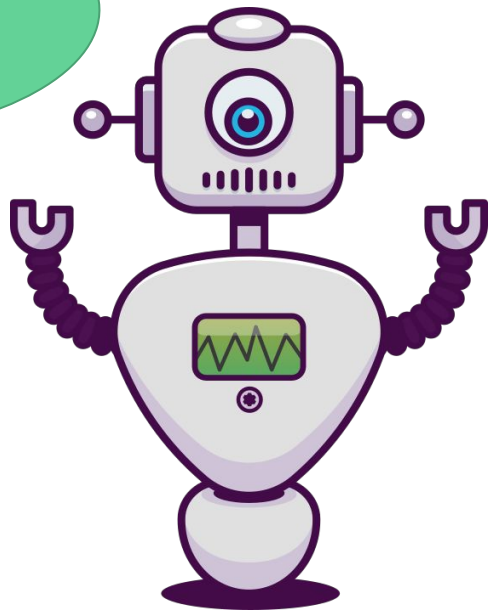
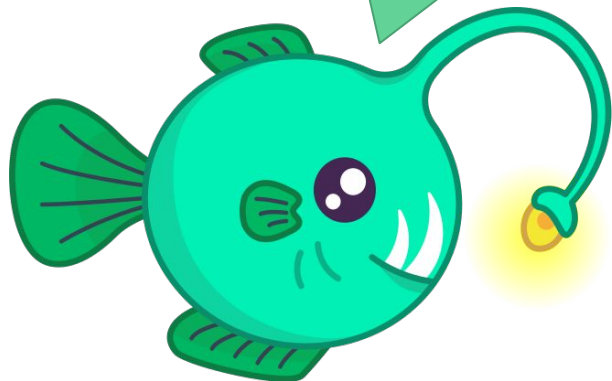
Bob



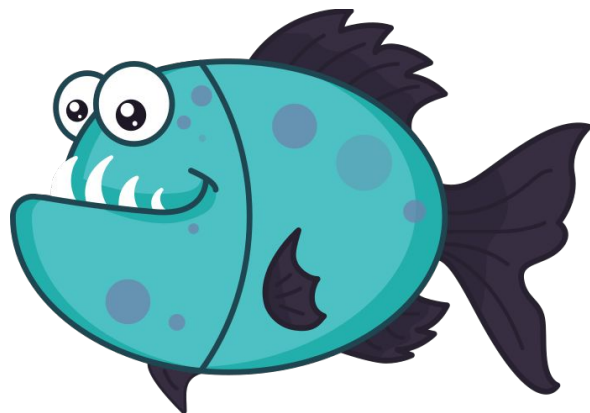
However, in the real world...
there are 1000s of curious robots between them.

Alice

How do we keep the
robot from knowing
what we're talking
about?



Bob



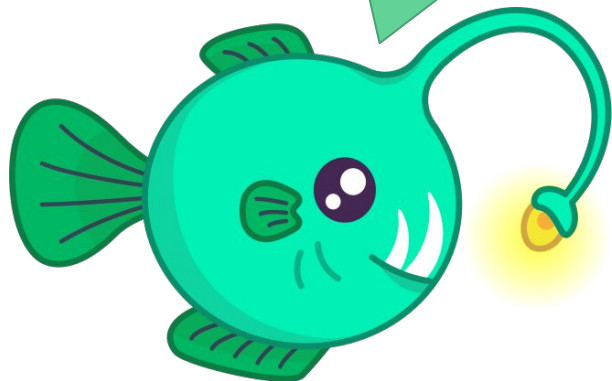
How do we ~~encrypt~~
jumble our text?

Encryption

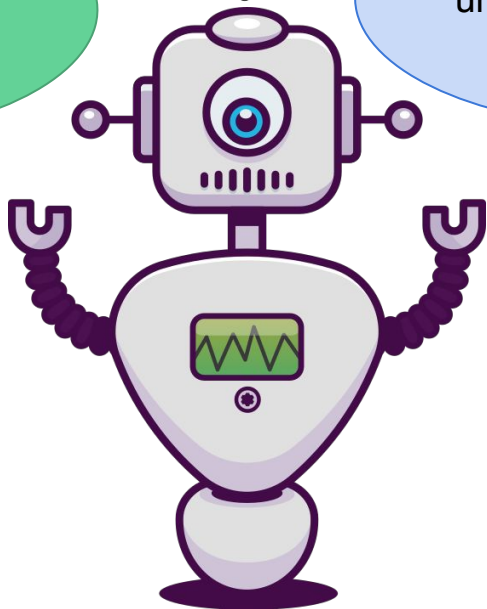
Use mathematics to figure out how to shift our bits via RSA & ECDSA

Alice

uryyb oyhr svfu

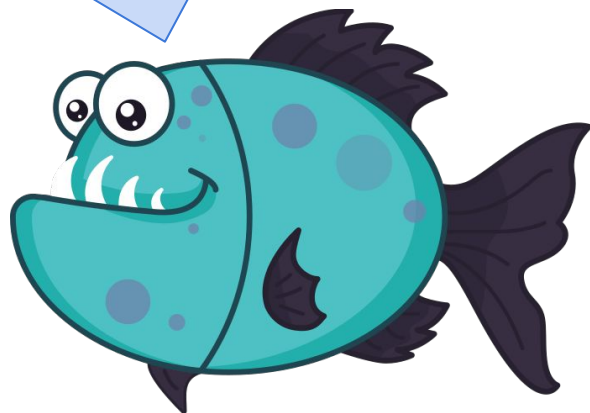


What are
they
saying?



Bob

uryyb terra svfu



What prevents Robot
from acting like Bob?

What prevents Robot from being Bob?

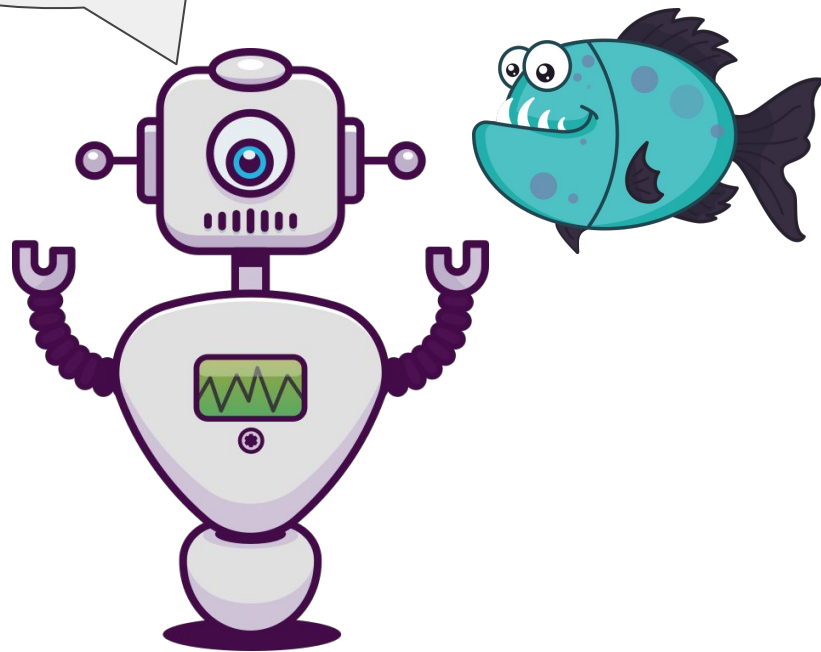
Alice



uryyb oyhr svfu

uryyb terra svfu

Bob

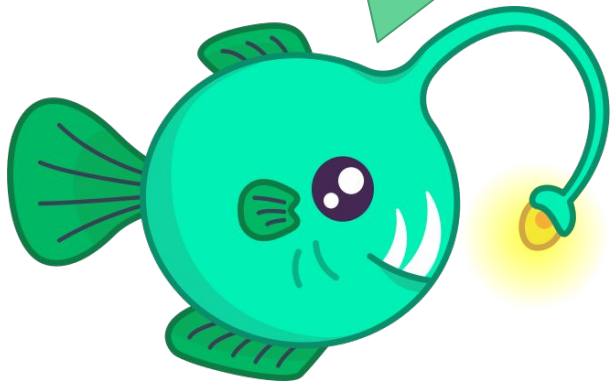


What prevents Robot
from changing Bob's
message?

What prevents Robot from changing Bob's message?

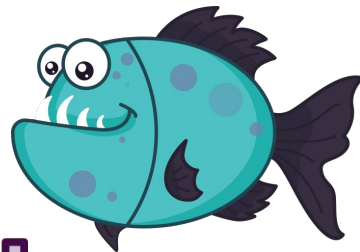
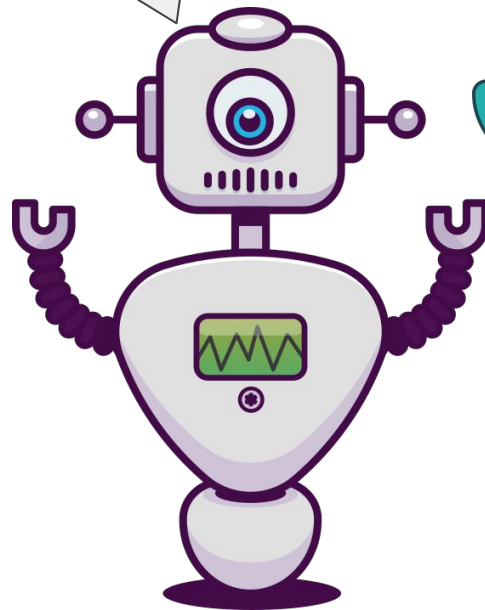
Alice

urzyb oyhr svfu



v nz ebobg

Bob



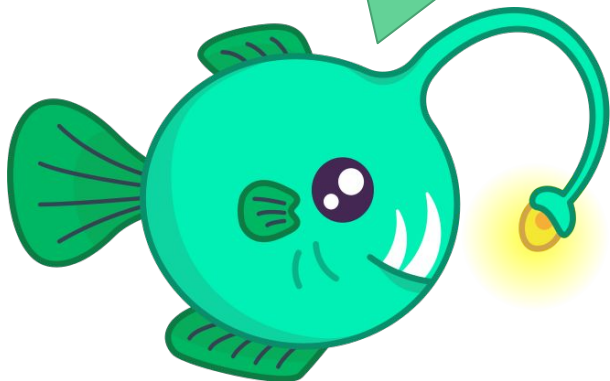
How do we ensure
integrity of our data?

Hashing

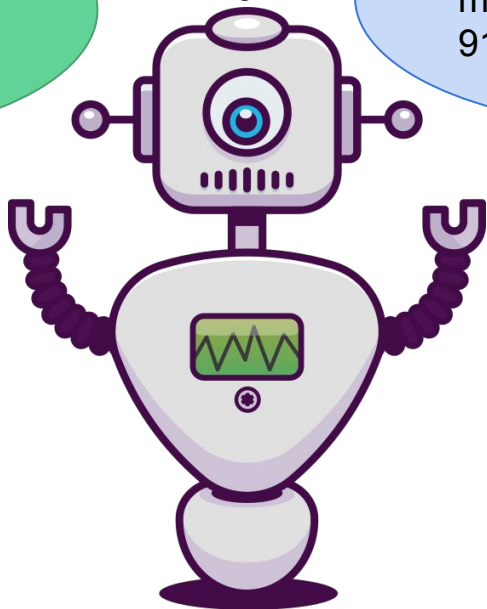
Use mathematics to calculate a 1-way value that representing our message via SHA-2 and SHA-3

Alice

uryyb oyhr svfu
md5:fb217fead3259a
daa2c80f9cfc9ab7e4

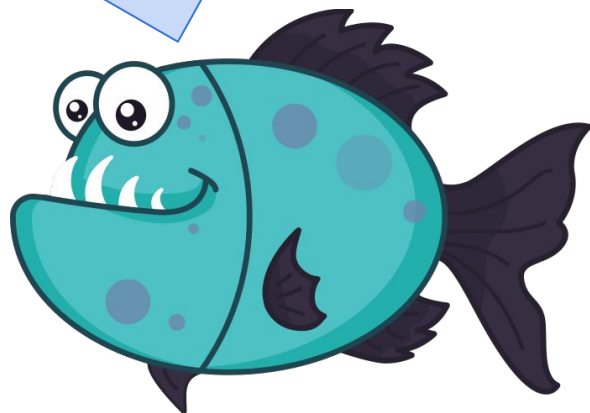


This is
harder to
hack.



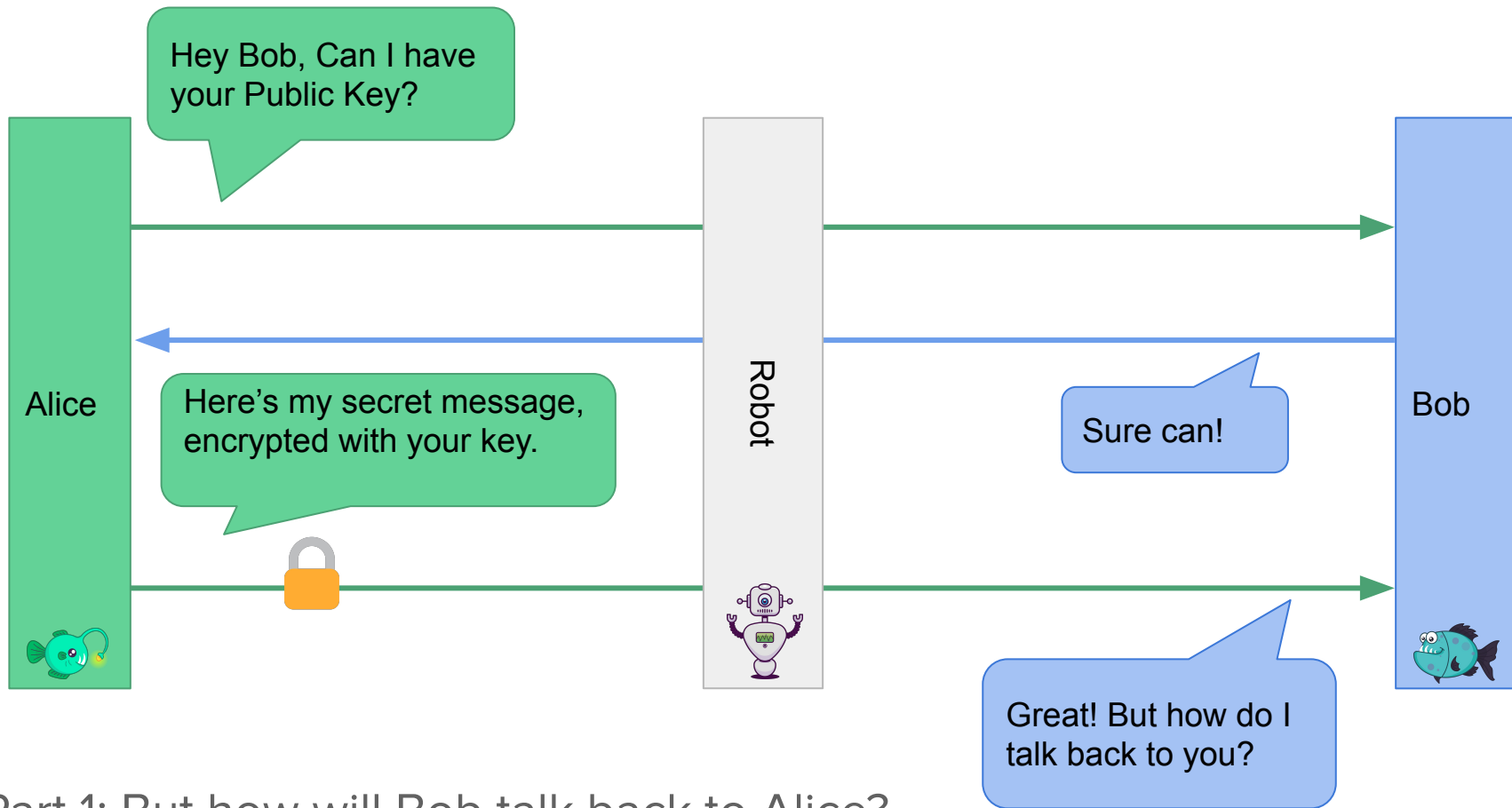
Bob

uryyb terra svfu
md5:539114041f9fa30
91b405125f3d2d6b1

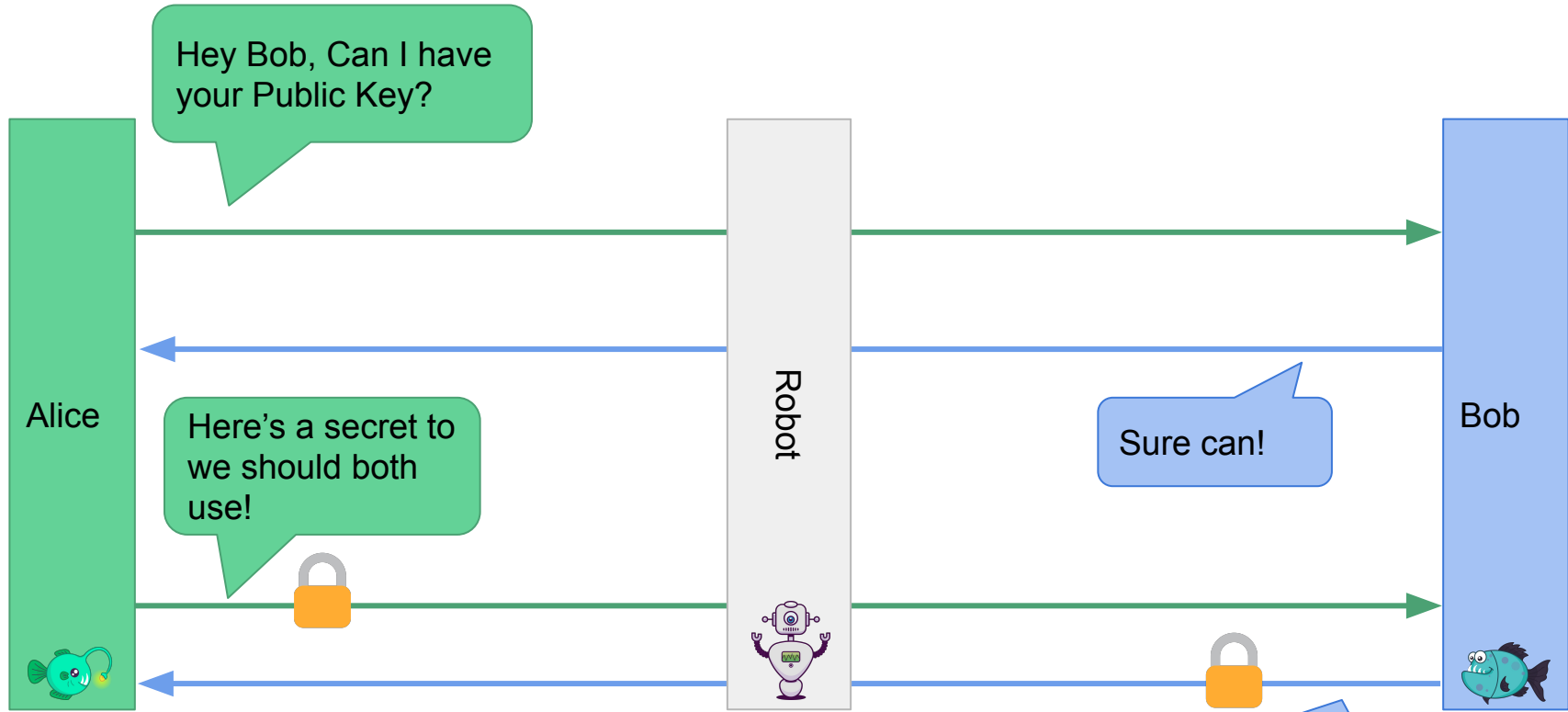


How do we formalize this?

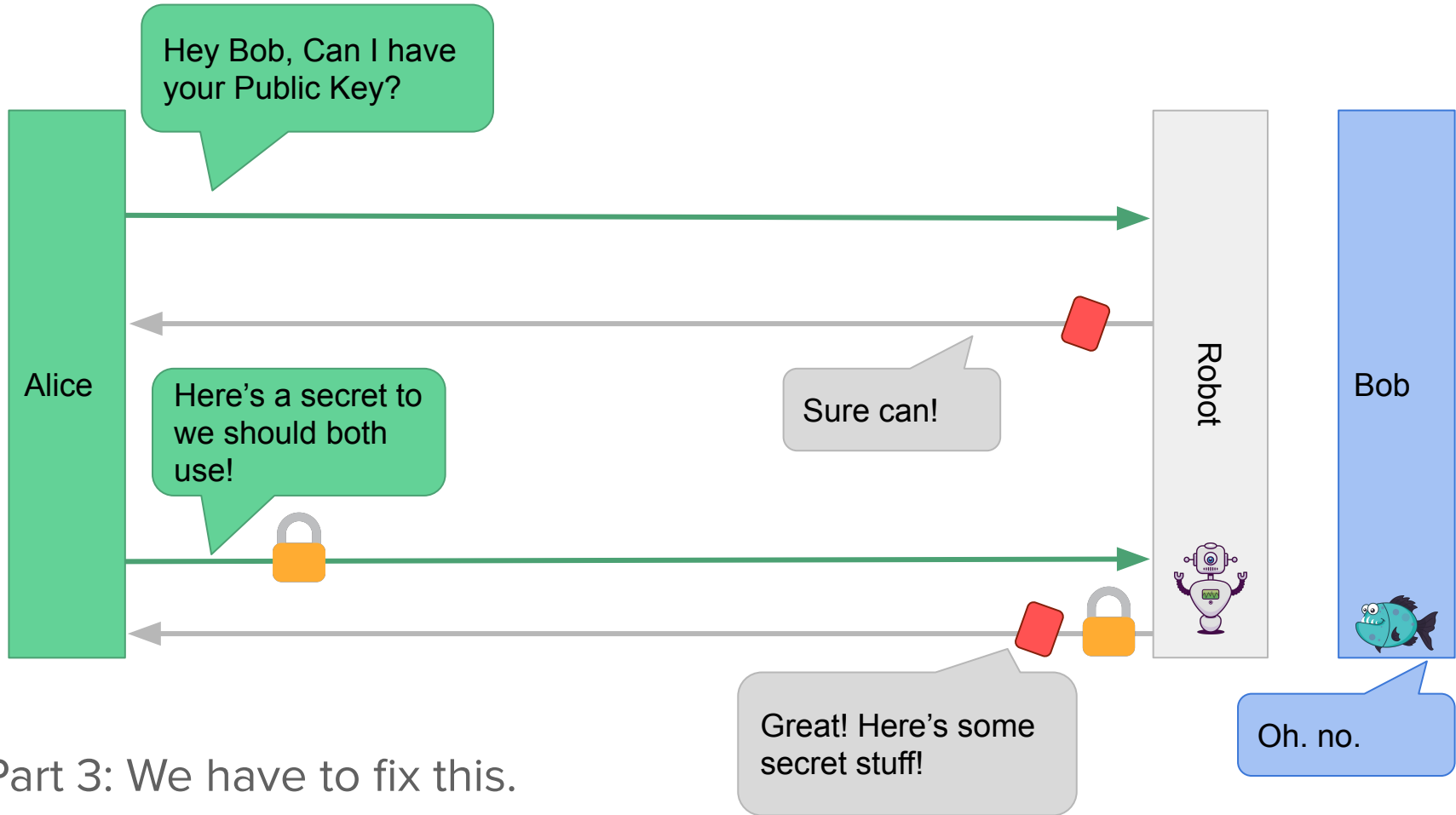
Public Key Cryptography



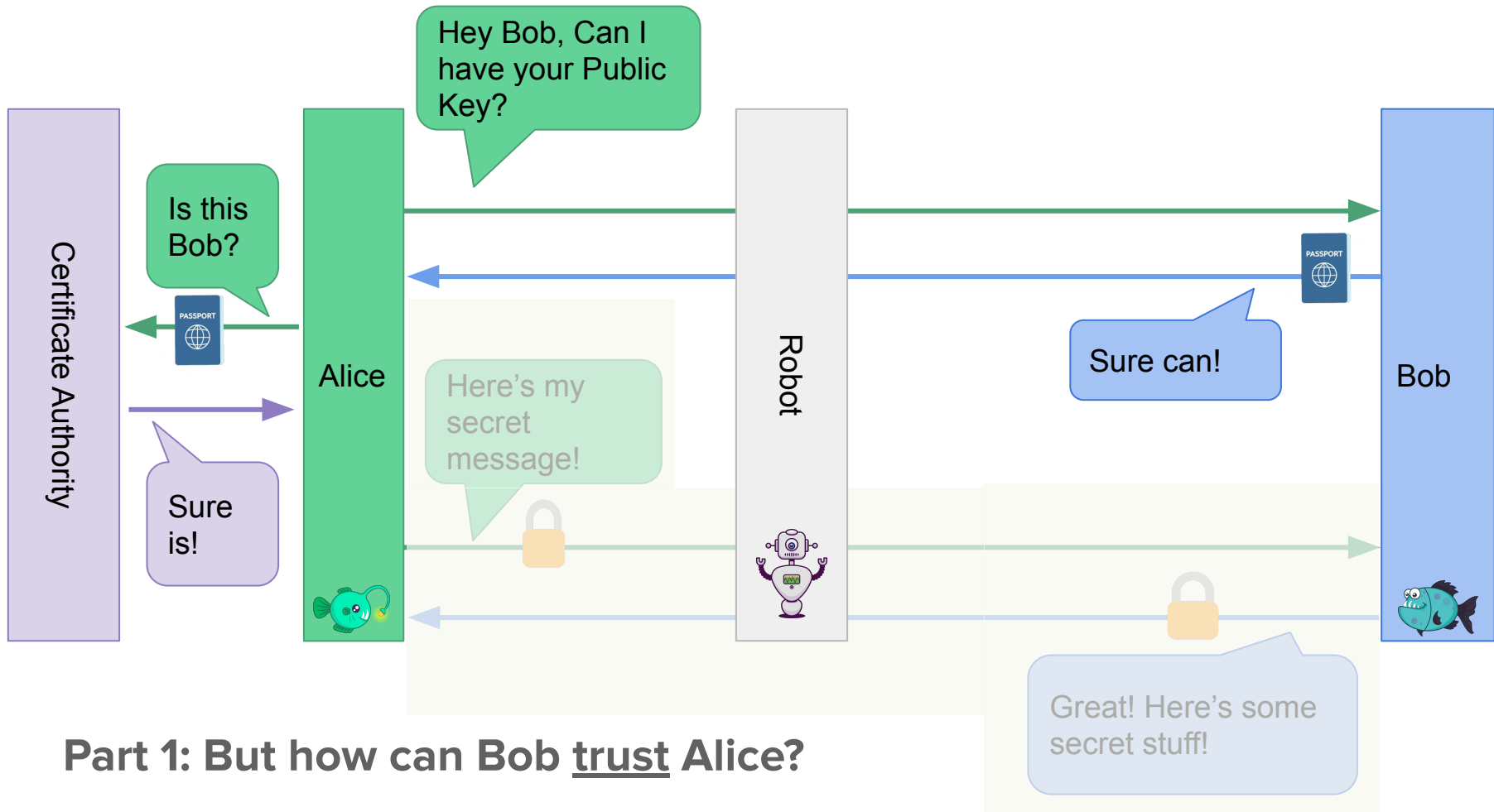
Part 1: But how will Bob talk back to Alice?

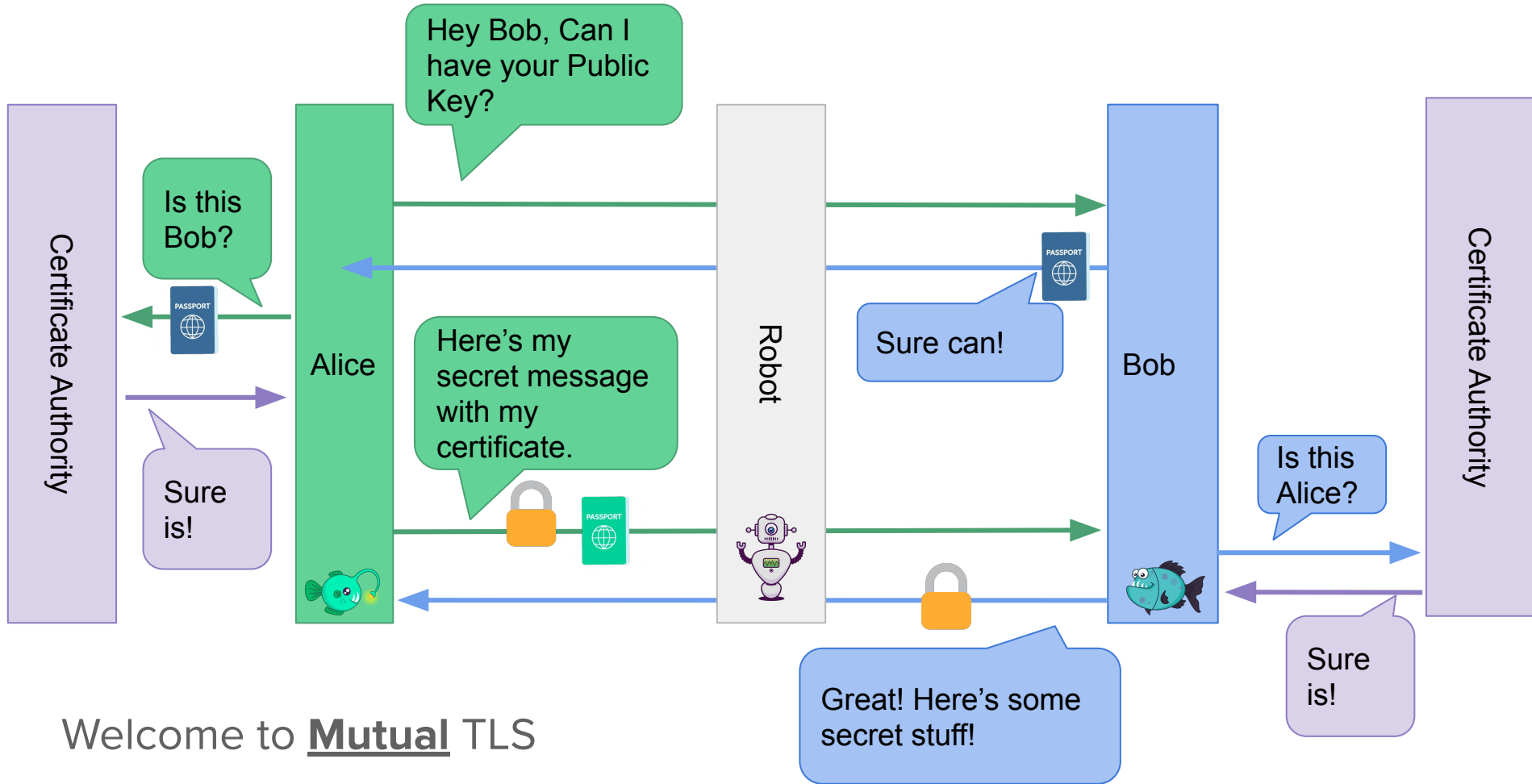


Part 2: But how do we know we can trust Bob is who he claims to be?



Certificate Authorities





The Few Parts to mTLS

X509 v3 Certificate
<ul style="list-style-type: none">● Encryption Algorithm<ul style="list-style-type: none">○ RSA○ ECDSA● Hashing Algorithms<ul style="list-style-type: none">○ SHA-2○ SHA-3

Cryptographic Protocols
SSL v2/v3 & **TLS 1.3/2/1**



X509 v3 Certificate
<ul style="list-style-type: none">● Encryption Algorithm<ul style="list-style-type: none">○ RSA○ ECDSA● Hashing Algorithms<ul style="list-style-type: none">○ SHA-2○ SHA-3

What could go wrong?

Datadog

Root Certificate

Expired

May 30, 2020

SOLUTIONS



ABOUT BLOG DOCS LOGIN

CERTIFICATE_VERIFY_FAILED error

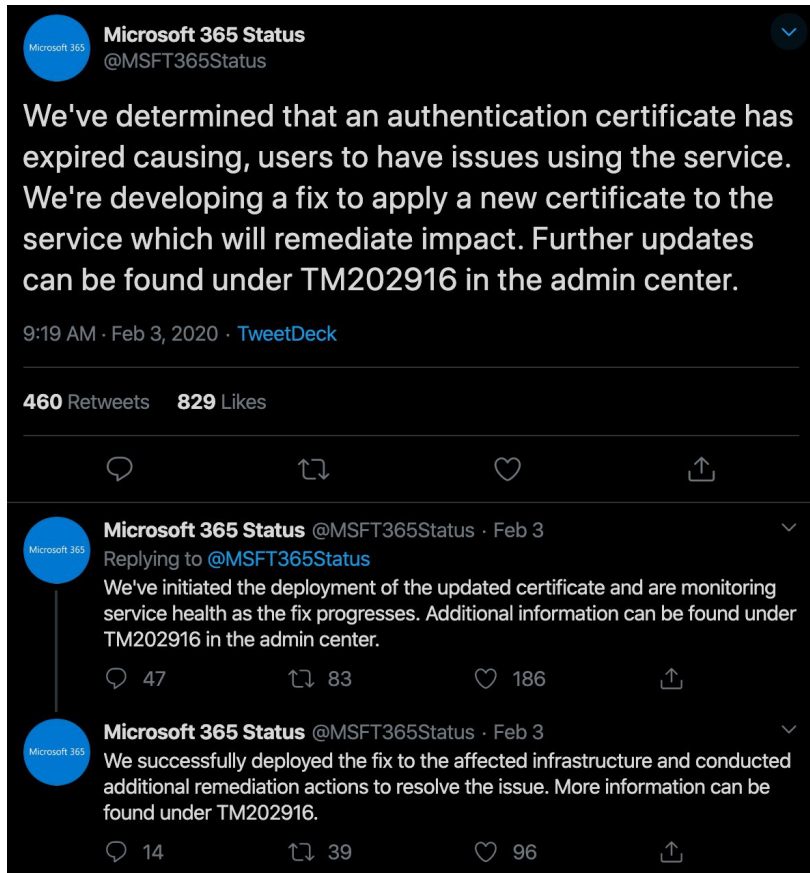
What happened?

On Saturday May 30th, 2020, at 10:48 UTC, an SSL root certificate used to cross-sign some of the Datadog certificates expired, and caused some of your Agents to lose connectivity with Datadog endpoints. Because this root certificate is embedded in certain Agent versions, you will need to take action to restore connectivity.

What versions of the Agent are affected?

Agent versions spanning 3.6.x to 5.32.6 embed the expired certificate and are affected.

Agent versions 6.x and 7.x are fine and don't need to be updated.



Microsoft Teams

3 Hour Outage

Feb 3, 2020

Ericsson

Millions of UK phones go offline

Dec 6, 2018

Update on software issue impacting certain customers

PRESS RELEASE | DEC 06, 2018 15:50 (GMT +00:00)

#Ericsson

Following network disturbances in a number of Ericsson's (NASDAQ:ERIC) customer networks, Ericsson has taken immediate action to minimize impact and support the restoration of services.

During December 6, 2018, Ericsson has identified an issue in certain nodes in the core network resulting in network disturbances for a limited number of customers in multiple countries using two specific software versions of the SGSN-MME (Serving GPRS Support Node – Mobility Management Entity).

Börje Ekholm, President and CEO, Ericsson, says: "The faulty software that has caused these issues is being decommissioned and we apologize not only to our customers but also to their customers. We work hard to ensure that our customers can limit the impact and restore their services as soon as possible."

An initial root cause analysis indicates that the main issue was an expired certificate in the software versions installed with these customers. A complete and comprehensive root cause analysis is still in progress. Our focus is now on solving the immediate issues.

During the course of December 6, most of the affected customers' network services have been successfully restored. We are working closely with the remaining customers that are still experiencing issues.

NOTES TO EDITORS

For media kits, backgrounders and high-resolution photos, please visit www.ericsson.com/press

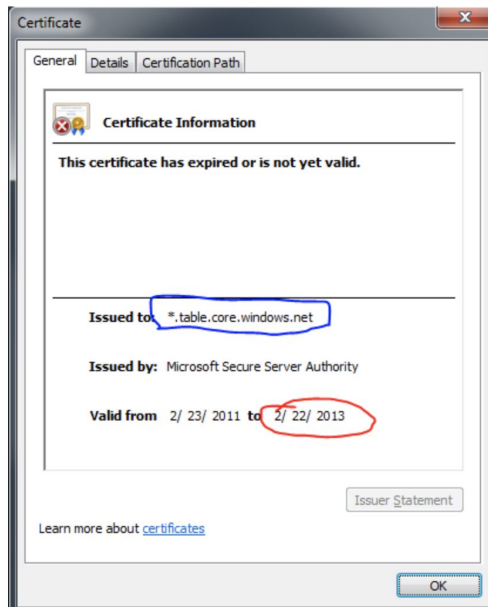
FOLLOWING



13

Sign
in to
vote

So is it just me, or did the HTTPS certificate for Azure Storage just expire?



Might want to fix that, ASAP. It also wouldn't hurt to put a sticky note on someone's monitor so they remember to update that before it expires next time.

UPDATE:

After reading through most of the replies here, the best solution seems to be changing your configuration through the management portal. Change all of your storage connection strings to use HTTP instead of HTTPS – EXCEPT for the Diagnostics connection string. That one must remain as HTTP (thanks to @Matt in Cambridge for that detail)

Microsoft Azure Storage Global Outage

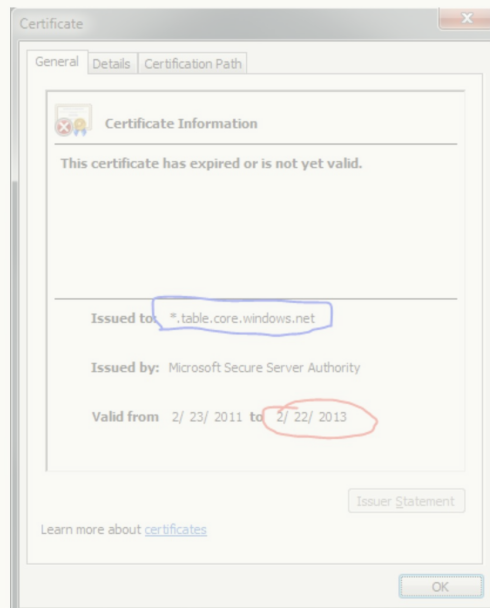
Rant mode. *Activate.*



13

Sign
in to
vote

So is it just me, or did the HTTPS certificate for Azure Storage just expire?



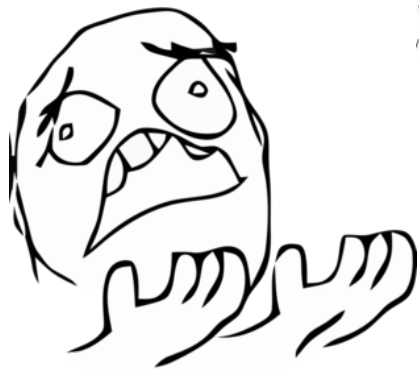
Might want to fix that, ASAP. It also wouldn't hurt to put a sticky note on someone's monitor so they remember to update that before it expires next time.

UPDATE:

After reading through most of the replies here, the best solution seems to be changing your configuration through the management portal. Change all of your storage connection strings to use HTTP instead of HTTPS – EXCEPT for the Diagnostics connection string. That one must remain as HTTPS (thanks to @Matt in Cambridge for that detail)

Microsoft Azure Storage Global Outage

Feb 22, 2013



1161

votes

A: How can I make git accept a self signed certificate?

/SSL verification for a single git command try passing -c to git with the proper config variable, or use Flow's answer: `git -c http.sslVerify=false clone https://example.com/path/to/git` To **disable** ... To permanently accept a specific certificate Try `http.sslCAPath` or `http.sslCAInfo`. Adam Spiers's answer gives some great examples. This is the most secure solution to the question. To **disable** TLS ...

answered Jul 23 '12 by [Christopher](#)

575

votes

A: Warning about SSL connection when connecting to MySQL database

Your connection URL should look like the below, `jdbc:mysql://localhost:3306/Peoples?autoReconnect=true&useSSL=false` This will **disable** SSL and also suppress the SSL errors. ...

answered Dec 24 '15 by [Priyank Gosalia](#)

531

votes

A: How do I set GIT_SSL_NO_VERIFY for specific repos only?

You can do git config `http.sslVerify false` in your specific repo to **disable** SSL certificate checking for that repo only. ...

answered Jan 25 '12 by [Joachim Isaksson](#)

312

votes

Q: Warning about SSL connection when connecting to MySQL database

applications not using SSL the `verifyServerCertificate` property is set to 'false'. You need either to explicitly **disable** SSL by setting `useSSL=false`, or set `useSSL=true` and provide truststore for server ... With the two classes below, I've tried connect to a MySQL database. However, I always get this error: Wed Dec 09 22:46:52 CET 2015 WARN: Establishing SSL connection without server's identity ...

[java](#) [ssl](#) [database-connection](#) [mysql-error-1064](#)asked Dec 9 '15 by [Milos86](#)

2

votes

Q: Disable SSL cert verification

I am working on an app and need to **disable** SSL verification to test locally. In this environment the SSL certificate doesn't validate so I need to **disable** the verification. How does one do that on Windows Phone 7.1? ...

[windows-phone](#) [windows-phone-7.1](#)asked Jun 17 '13 by [Steven](#)

0

votes

Q: Disable SSL in XMPPpy?

I need to **Disable** SSL Connection while connecting to gmail, I had try this code : `import xmpp username = 'username' passwd = 'password' to='friend@gmail.com' msg='hello :)' client = xmpp.Client ... ('gmail.com') client.connect(('talk.google.com',5223),None,'None',None) client.auth(username, passwd, 'omar') client.sendInitPresence() message = xmpp.Message(to, msg) message.setAttr('type', 'chat') client.send(message) "None" must be ssl=None without (") but it's not working , Any Help ? ...`

1

answer

1161

votes

[A: How can I make git accept a self signed certificate?](#)

/SSL verification for a single git command try passing -c to git with the proper config variable, or use Flow's answer: `git -c http.sslVerify=false clone https://example.com/path/to/git` To **disable** ... To permanently accept a specific certificate Try `http.sslCAPath` or `http.sslCAInfo`. Adam Spiers's answer gives some great examples. This is the most secure solution to the question. To **disable** TLS ...

answered Jul 23 '12 by Christopher

575

votes

[A: Warning about SSL connection when connecting to MySQL](#)

Your connection URL should look like the below, `jdbc:mysql://localhost:3306/dbname?autoReconnect=true&useSSL=false` This will **disable** SSL

What's this?

/ Trial

531

votes

[A: How do I ...](#)

You

with the two
Dec 09 22:46:52

asked Dec 9 '15 by Milos86

to **disable** SSL verification to test locally. In this environment the **SSL**
so I need to **disable** the verification. How does one do that on Windows Phone 7.1? ...

windows-phone-7.1

asked Jun 17 '13 by Steven

0

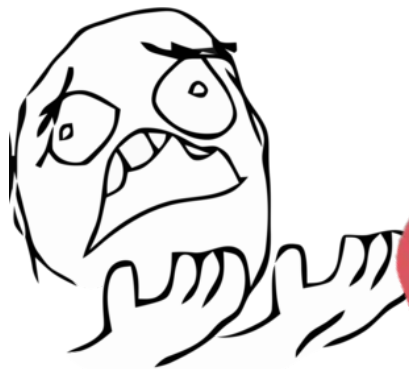
votes

[Q: Disable SSL in XMPPpy?](#)

1

answer

I need to **Disable** SSL Connection while connecting to gmail, I had try this code : `import xmpp username = 'username' passwd = 'password' to='friend@gmail.com' msg='hello :)'` client = xmpp.Client ... ('gmail.com')
client.connect(("talk.google.com",5223),None,"None",None) client.auth(username, passwd, 'omar')
client.sendInitPresence() message = xmpp.Message(to, msg) message.setAttr('type', 'chat')
client.send(message) "None" must be **ssl=None** without (") but it's not working , Any Help ? ...





```
// Create all-trusting host name verifier
HostnameVerifier validHosts = new HostnameVerifier() {
    @Override
    public boolean verify(String arg0, SSLSession arg1) {
        return true;
    }
}
```

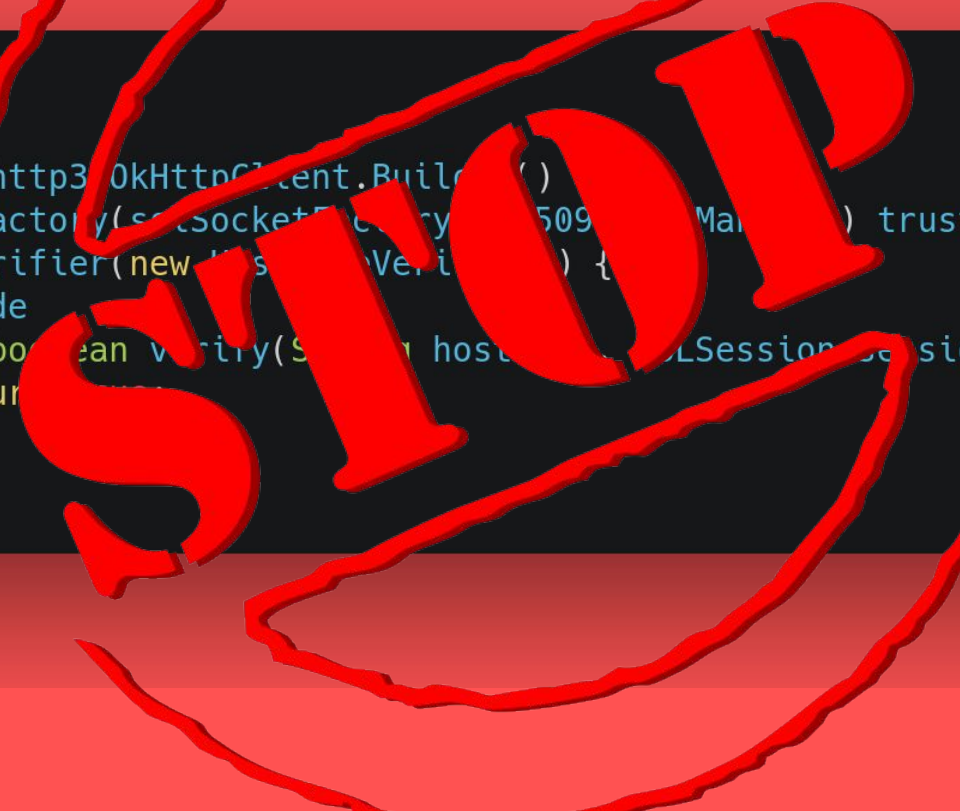


```
// Create all-trusting hostname verifier  
HostnameVerifier myHostnameVerifier = new HostnameVerifier() {  
    @Override  
    public boolean verify(String arg0, SSLSession arg1) {  
        return true;  
    }  
}
```

STOP



```
return new okhttp3.OkHttpClient.Builder()  
    .sslSocketFactory(sslSocketFactory, (X509TrustManager) trustAllCerts[0])  
    .hostnameVerifier(new HostnameVerifier() {  
        @Override  
        public boolean verify(String hostname, SSLSession session) {  
            return true;  
        }  
    }).build();
```

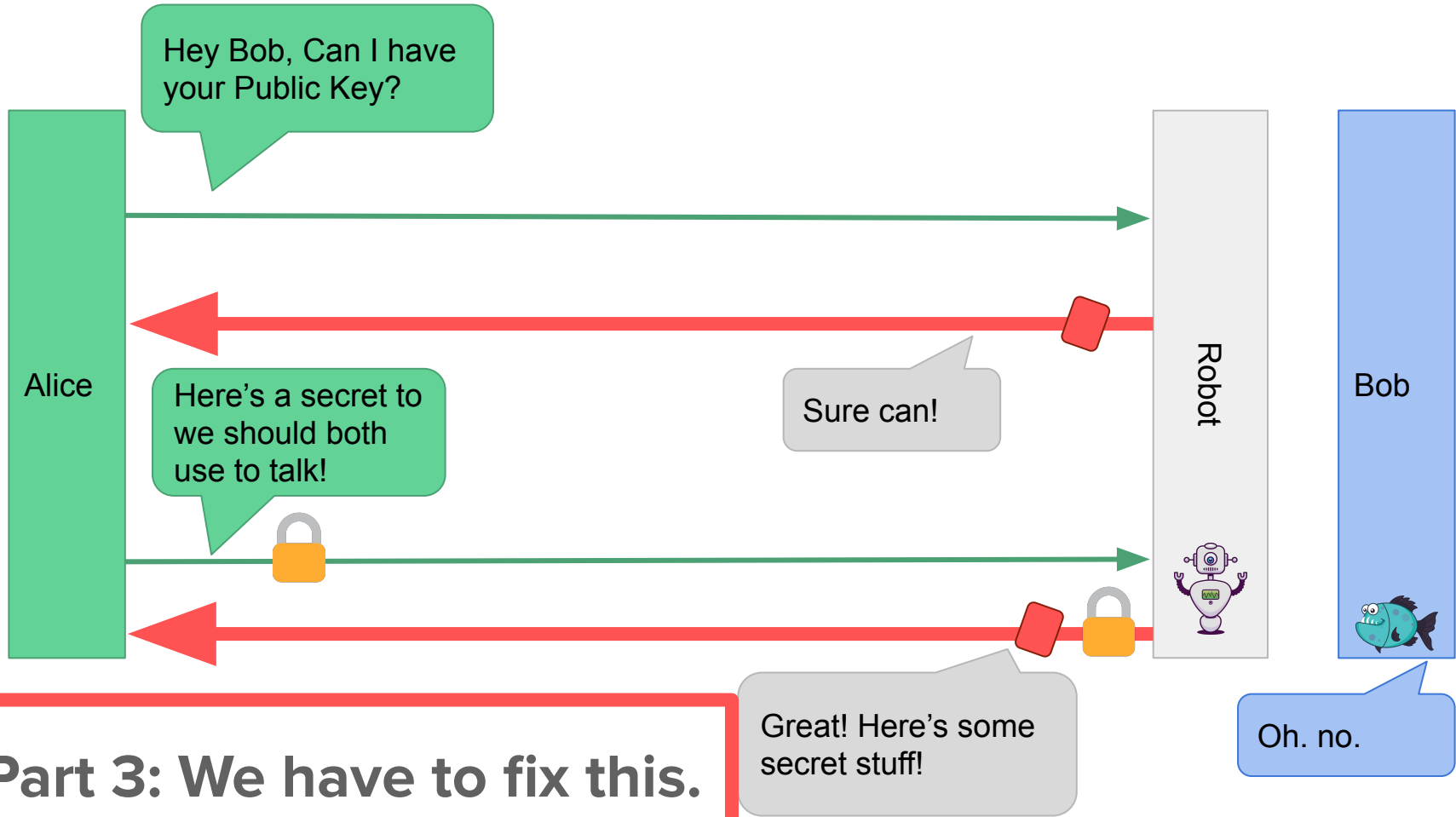
```
return new OkHttpClient.Builder()
    .sslSocketFactory(sslSocketFactory, 509)
    .hostnameVerifier(new HostnameVerifier() {
        @Override
        public boolean verify(String hostname, SSLSession session) {
            return true;
        }
    }).build();
```


Is this what you
want?

A gate with no fence.



What good is this?



WE  ~~CAN MUST~~
WILL  FIX  THIS.

Rant mode.

Deactivate.

Easily preventable errors.

Why do they keep
reoccurring?

Automation is hard.



⚠ Not Secure | expired.badssl.com

Guest



Your connection is not private

Attackers might be trying to steal your information from **expired.badssl.com** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_DATE_INVALID

Errors messages are
specific to clients,
and oftentimes
meaningless.

¬_(ツ)_/

#GarbagersGarbage



badssl.com

Memorable site for testing clients against bad SSL configs.

What ***else*** could go
wrong?

Severity

High

Vendor

Cloud Foundry Foundation

Description

Cloud Foundry CredHub, versions prior to 2.5.10, connects to a MySQL database without TLS even when configured to use TLS. A malicious user with access to the network between CredHub and its MySQL database may eavesdrop on database connections and thereby gain unauthorized access to CredHub and other components.

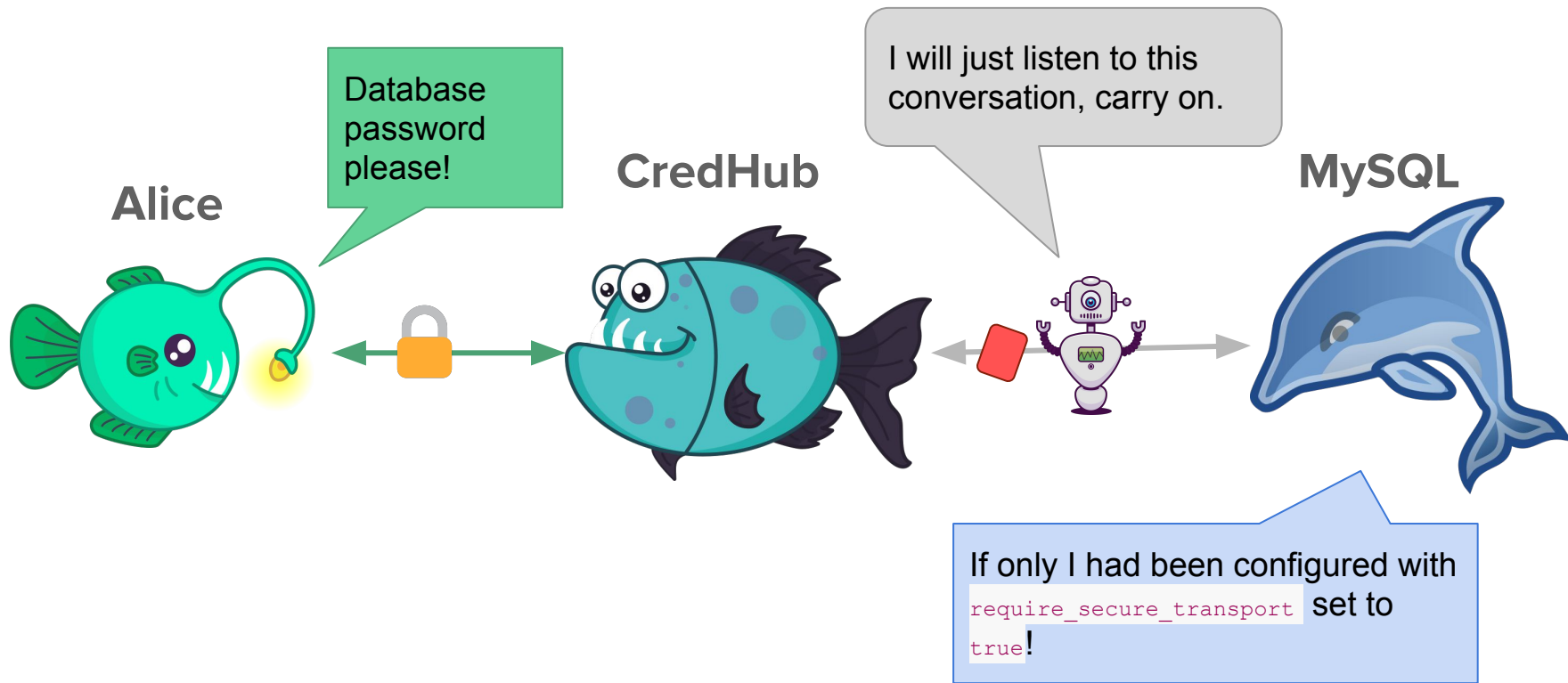
Affected Cloud Foundry Products and Versions

Severity is high unless otherwise noted.

- CF Deployment
 - All versions prior to v12.29.0
- CredHub
 - All versions prior to 2.5.10

Cloud Foundry
CredHub doesn't use
TLS despite it being
configured.

CVE-2020-5399




Unencrypted Database access despite configuration.

HashiCorp Nomad

CVE-2020-7956

Jan 28, 2020


 hashicorp / nomad

[Watch](#) 387 [Star](#) 5.8k [Fork](#) 1k

[Code](#) [Issues 562](#) [Pull requests 55](#) [Actions](#) [Projects 0](#) [Security](#) [Insights](#)

CVE-2020-7956: Privilege escalation due to incorrect certificate validation for role/region #7003 [New issue](#)

[Closed](#) schmichael opened this issue 16 days ago · 0 comments · Fixed by [#7023](#)

 **schmichael** commented 16 days ago · edited by preetapan Member ...


Vulnerability ID: CVE-2020-7956
Versions: Previous versions of Nomad and Nomad Enterprise; fixed in 0.10.3.

Nomad 0.10.3 includes a fix for a privilege escalation vulnerability in validating TLS certificates for RPC with mTLS. Nomad RPC endpoints validated that TLS client certificates had not expired and were signed by the same CA as the Nomad node, but did not correctly check the certificate's name for the role and region as described in the Securing Nomad with TLS guide. This allows trusted operators with a client certificate signed by the CA to send RPC calls as a Nomad client or server node, bypassing access control and accessing any secrets available to a client.

Nomad clusters configured for mTLS following the Securing Nomad with TLS guide or the Vault PKI Secrets Engine Integration guide should already have certificates that will pass validation. Before upgrading to Nomad 0.10.3, operators using mTLS with `verify_server_hostname = true` should confirm that the common name or SAN of all Nomad client node certs is `client..nomad`, and that the common name or SAN of all Nomad server node certs is `server..nomad`.

[Start timer](#)

Assignees

 schmichael

Labels

None yet

Projects

None yet

Milestone

No milestone

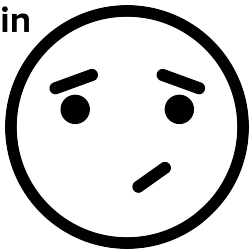
Linked pull requests

Nomad Particulars

Nomad's use of mTLS provides the following properties:

- Prevent unauthorized Nomad access
- Prevent observing or tampering with Nomad communication
- **Prevent client/server role or region misconfigurations**
- Prevent other services from masquerading as Nomad agents

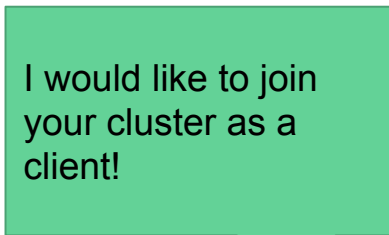
SysAdmin



verify_server_hostname = true

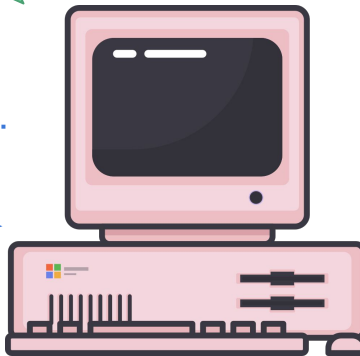


Nomad Leaders



I would like to join
your cluster as a
client!

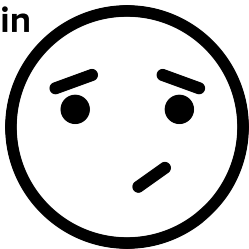
cn=client.region.nomad



Nomad Client

Welcome! Do some work.

SysAdmin



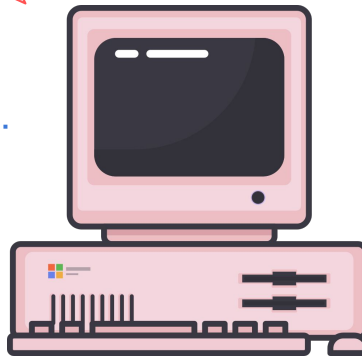
verify_server_hostname = true



Nomad Leaders

I would like to join
your cluster as a
server!

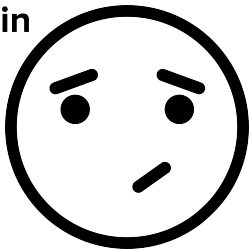
cn=**client**.region.nomad



Nomad Client

You are not a server - goodbye!

SysAdmin



verify_server_hostname = true

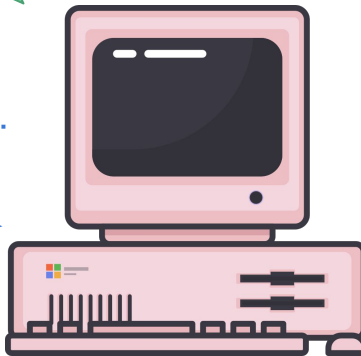


Nomad Leaders

cn=client.**regionA**.nomad



cn=server.**regionA**.nomad



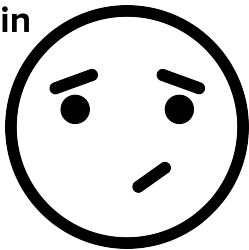
Nomad Client

I would like to join
your cluster as a
client!

Welcome! Do some work
in Region A.

I will!

SysAdmin



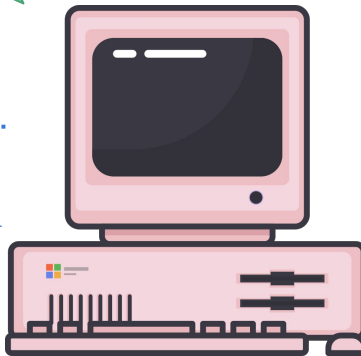
verify_server_hostname = true



Nomad Leaders

I would like to join
your cluster as a
client in regionA!

cn=client.**regionA**.nomad



Nomad Client

cn=server.regionB.nomad

Hello!

I only work for
Region A. Goodbye.

SadAdmin



verify_server_hostname = true



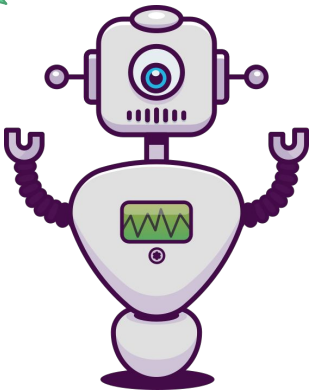
Nomad Leaders

I am a server!

cn=real.bad.guy



I found this
certificate signed
by your CA.
Let's see what I
can use it for!



Rogue Nomad Client

Welcome! You can take control!

etcd-io / etcd

Watch 1.3k

Stars

Code Issues 200 Pull requests 77 Actions Security 0 Insights

*: disable CommonName auth for gRPC-gateway #10366

Merged hexfusion merged 4 commits into etcd-io:master from hexfusion:fx_cn on Jan 8, 2019

Conversation 25 Commits 4 Checks 1 Files changed 6



hexfusion commented on Jan 2, 2019 • edited

Member

When client-cert-auth is enabled gRPC-gateway proxy request to etcd server will use the CN in client server cert if populated. We should not use this for authentication. In general gRPC-gateway does not support CN authentication and we also should document as such.

The check provided in this PR is based on headers send from gRPC-Gateway

<https://github.com/grpc-ecosystem/grpc-gateway/blob/8976e602c518589cb6d40aca52a7dd8aef83706a/runtime/context.go#L183#>

/cc @gyuho @mitake

Review

m



91

Assign

No on

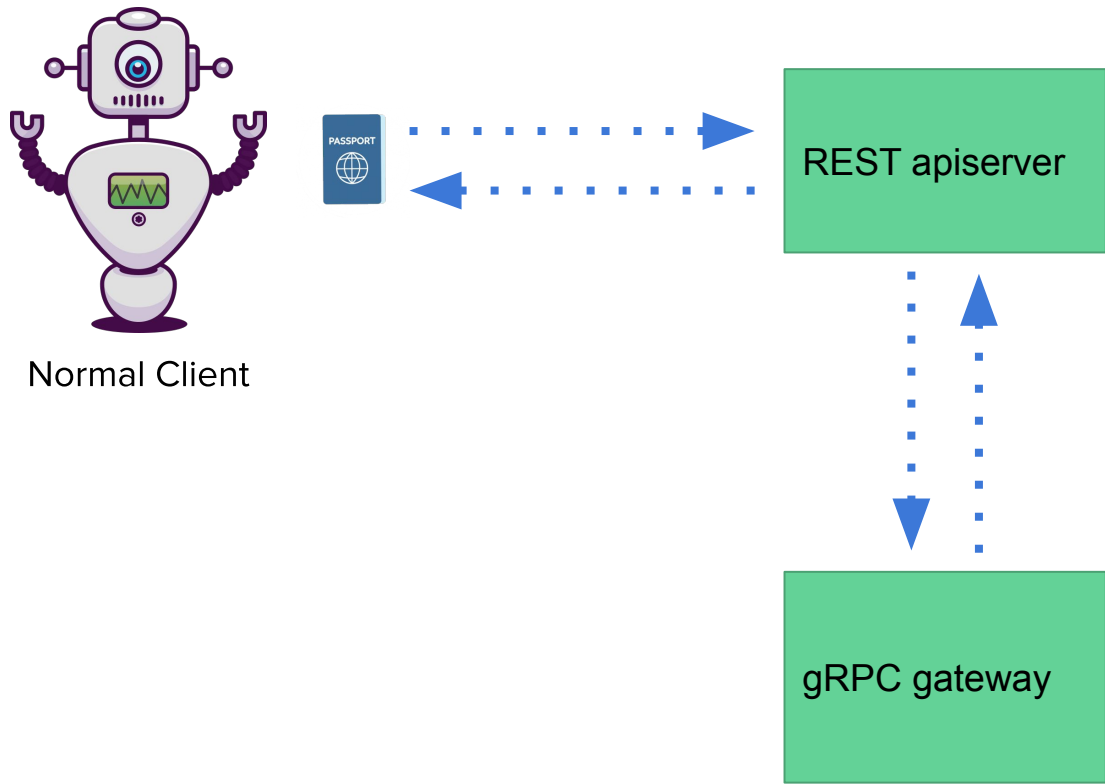
Label:

Release

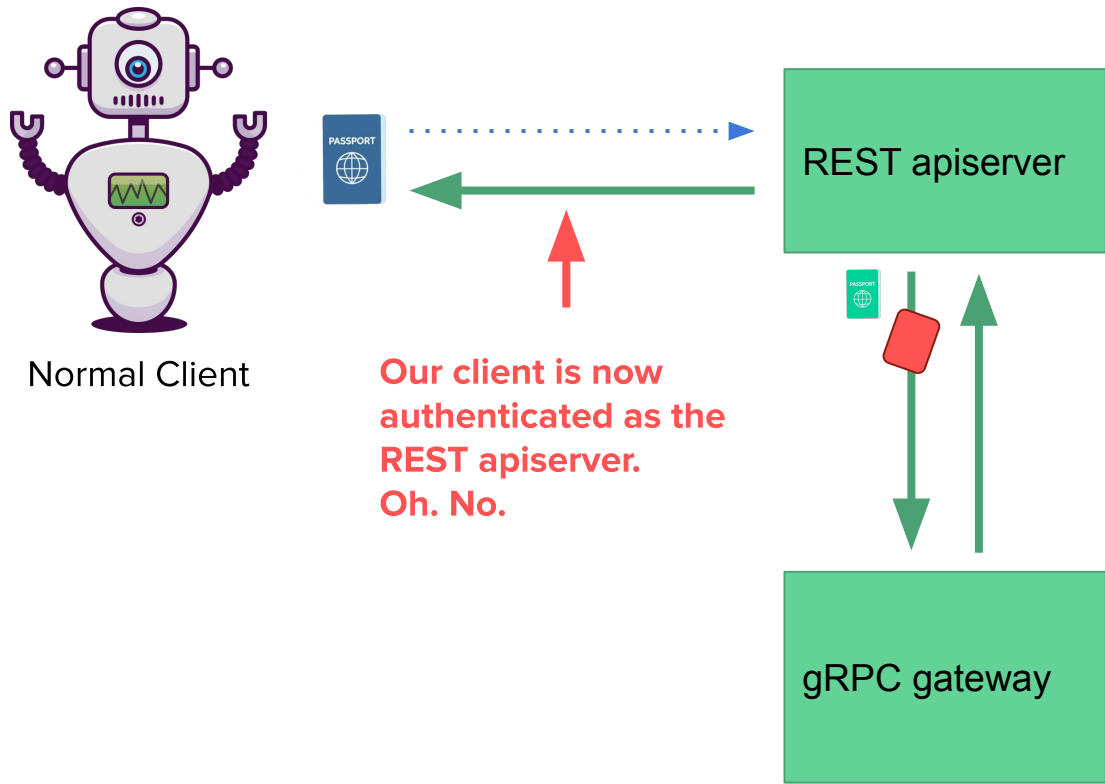
If an etcd client server's TLS certificate contains a Common Name (CN) which matches a valid RBAC username, a remote attacker may authenticate as that user with any valid (trusted) client certificate in a REST API request to the gRPC-gateway.

CVE-2018-16886

Jan 2, 2019



How certificate authentication should work.



How certificate authentication actually worked.

Docker trusts Client
certificates signed by
provided CA root
certificate and any
system root certificate.

CVE-2018-12608

Use exclusive root pools if a CA cert file is specified in the daemon #33182

 **Merged** mlaventure merged 1 commit into moby:master from cyli:exclusive-root-pools-in-daemon on May 13, 2017

 Conversation 5  Commits 1  Checks 0  Files changed 2



cyli commented on May 12, 2017

Contributor  ...

Fixes #33173.

#31705 added ExclusiveRootPools: true when setting up the docker client configuration, but this should also be applied to the daemon.

If a file containing CAs for validating clients is provided, only the certs used in that file should be used to validate client connections, and not both the certs in that file *and* the system root certs.

If the union of the system certs and the provided CA certs is desired, the additional CA certs should be added to the system pool, or the system certs added to the provided CA file.

cc @dmcgowan @thaJeztah

Also cc @diogomonica for visibility

☐ cute

 GordonTheTurtle added the status/0-triage label on May 12, 2017

 Start timer

Reviewers

 cpuguy83

Assignees

No one assigned

Labels

priority/P0

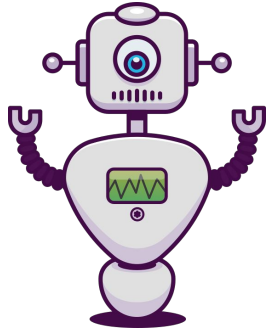
status/0-triage

status/2-code

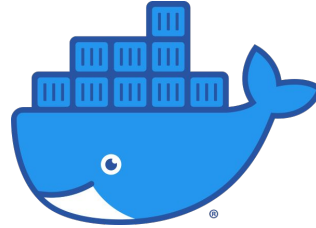
Projects

None yet

Let me in, I have a
certificate from your
Certificate Authority.

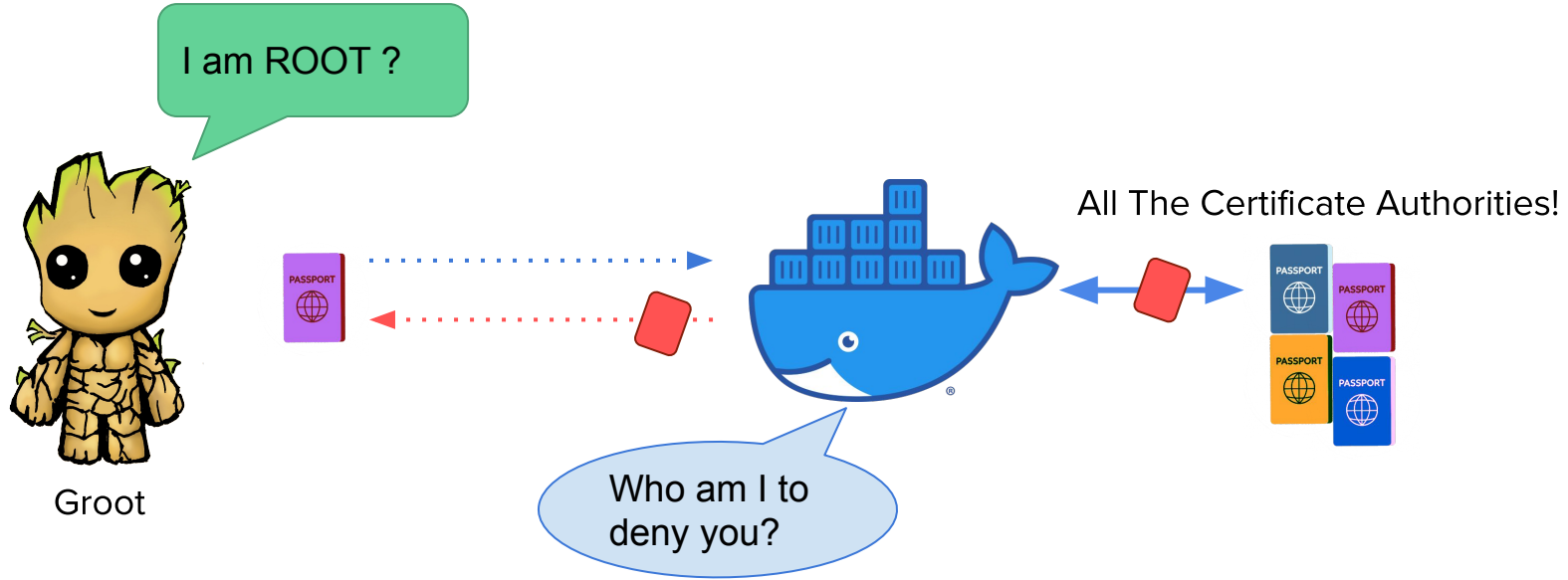


Normal Client



Single Certificate Authority





So how do we start
learning?



bit.ly/learn-mtls

A workshop that will be much more in-depth than these slides.

SSL Certificate

TLS Certificate

HTTPS Certificate

X.509 v3 Certificate

~~SSL Certificate~~

~~TLS Certificate~~

~~HTTPS Certificate~~

X.509 v3 Certificate

236 Pages

[Blue Book](#)

[Volume VIII - Fascicle VIII.8](#)

Data Communication Networks Directory
Recommendations X.500-X.521



Let's make certificates
with mkcert

macOS

```
brew install mkcert  
brew install nss # if you use Firefox
```

```
sudo port selfupdate  
sudo port install mkcert  
sudo port install nss # if you use Firefox
```

Windows

```
choco install mkcert  
-or-  
scoop bucket add extras  
scoop install mkcert
```

Linux

```
sudo apt install libnss3-tools  
-or-  
sudo yum install nss-tools  
-or-  
sudo pacman -S nss  
-or-  
sudo zypper install mozilla-nss-tools
```

Latest release

v1.4.1
90341b0

Compare

The little things

FiloSottile released this on Nov 9, 2019

Note: packagers building from source now need to set the version like `-ldflags "-X main.Version=$VERSION"`

- Use `sudo` when necessary to install in system-wide NSS stores (#192)
- Add a `-version` flag (#191)
- Speed up macOS execution by 4x for most users (#135)
- Minor usability improvements (#182, #178, #188)

Assets 6

mkcert-v1.4.1-darwin-amd64	5.29 MB
mkcert-v1.4.1-linux-amd64	4.7 MB
mkcert-v1.4.1-linux-arm	4.21 MB
mkcert-v1.4.1-windows-amd64.exe	4.8 MB
Source code (zip)	
Source code (tar.gz)	

Install **mkcert** on your computer.

Linux

```
mkcert -install  
# Using the local CA at "/home/ascherger/.local/share/mkcert" ✨  
# The local CA is already installed in the system trust store! 👍  
# The local CA is now installed in the Firefox and/or Chrome/Chromium trust store  
# (requires browser restart)! 🦊
```

Run `mkcert -install`

Linux

```
mkcert -install
# Using the local CA at "/home/ascherger/.local/share/mkcert" ✨
# The local CA is already installed in the system trust store! 👍
# The local CA is now installed in the Firefox and/or Chrome/Chromium trust store
# (requires browser restart)! 🦊
```

Run `mkcert -install`

Linux

```
mkcert -install
# Using the local CA at "/home/ascherger/.local/share/mkcert" ✨
# The local CA is already installed in the system trust store! 👍
# The local CA is now installed in the Firefox and/or Chrome/Chromium trust store
# (requires browser restart)! 🦊
```

Only works if you have NSS installed.

Run `mkcert -install`

```
ls -al /home/ascherger/.local/share/mkcert
# total 16
# drwxr-xr-x  2 ascherger ascherger 4096 Feb 18 16:34 .
# drwx----- 38 ascherger ascherger 4096 Jun  5 08:00 ..
# -r-----  1 ascherger ascherger 2484 Feb 18 16:34 rootCA-key.pem
# -rw-r--r--  1 ascherger ascherger 1740 Feb 18 16:34 rootCA.pem
```

rootCA-key.pem is the private key.
rootCA.pem is the public certificate.

Let's see what certificates were created.

macOS

```
openssl version  
# LibreSSL 2.3.8
```

Linux

```
openssl version  
# OpenSSL 1.1.1d 10 Sep 2019
```

Sometimes OpenSSL is LibreSSL

```
> man openssl
```

```
OPENSSL(1SSL)
```

```
OPENSSL(1SSL)
```

```
OpenSSL
```

NAME

```
openssl - OpenSSL command line tool
```

SYNOPSIS

```
openssl command [ command_opts ] [ command_args ]
```

```
    openssl list [ standard-commands | digest-commands | cipher-commands | cipher-algorithms |  
digest-algorithms | public-key-algorithms ]
```

```
    openssl no-XXX [ arbitrary options ]
```

DESCRIPTION

OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them.

The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for

- o Creation and management of private keys, public keys and parameters
- o Public key cryptographic operations
- o Creation of X.509 certificates, CSRs and CRLs
- o Calculation of Message Digests
- o Encryption and Decryption with Ciphers
- o SSL/TLS Client and Server Tests
- o Handling of S/MIME signed or encrypted mail
- o Time Stamp requests, generation and verification

OpenSSL [man page](#) is your friend!

```
openssl rsa -help
Usage: rsa [options]
Valid options are:
  -help           Display this summary
  -inform format  Input format, one of DER PEM
  -outform format Output format, one of DER PEM PVK
  -in val         Input file
  -out outfile    Output file
  -pubin         Expect a public key in input file
  -pubout        Output a public key
  -passout val    Output file pass phrase source
  -passin val     Input file pass phrase source
  -RSAPublicKey_in Input is an RSAPublicKey
  -RSAPublicKey_out Output is an RSAPublicKey
  -noout         Don't print key out
  -text          Print the key in text
  -modulus       Print the RSA key modulus
  -check         Verify key consistency
  -*            Any supported cipher
  -pvk-strong    Enable 'Strong' PVK encoding level (default)
  -pvk-weak     Enable 'Weak' PVK encoding level
  -pvk-none      Don't enforce PVK encoding
  -engine val    Use engine, possibly a hardware device
```

OpenSSL RSA [man page](#)



```
openssl rsa -help
Usage: rsa [options]
```

Valid options are:



```
-help          Display this summary
-inform format  Input format, one of DER PEM
-outform format Output format, one of DER PEM PVK
-in val        Input file
-out outfile    Output file
```



```
-pubin         Expect a public key in input file
-pubout        Output a public key
-passout val    Output file pass phrase source
-passin val     Input file pass phrase source
-RSAPublicKey_in  Input is an RSAPublicKey
-RSAPublicKey_out Output is an RSAPublicKey
-noout         Don't print key out
-text          Print the key in text
-modulus       Print the RSA key modulus
-check         Verify key consistency
-*             Any supported cipher
-pvk-strong     Enable 'Strong' PVK encoding level (default)
-pvk-weak      Enable 'Weak' PVK encoding level
-pvk-none      Don't enforce PVK encoding
-engine val     Use engine, possibly a hardware device
```

OpenSSL RSA [man page](#)

```
openssl rsa -in rootCA-key.pem -text -noout
RSA Private-Key: (3072 bit, 2 primes)
modulus:
 00:d1:...
publicExponent: 65537 (0x10001)
privateExponent:
 56:5e:...
prime1:
 00:ef:...
prime2:
 00:df:...
exponent1:
 00:cb:...
exponent2:
 00:df:...
coefficient:
 11:69:...
```

Let's examine at our rootCA-key.pem

RFC 3447 Section 3.2

Discussion

Just large numbers

```
openssl rsa -in rootCA-key.pem -text -noout
RSA Private-Key: (3072 bit, 2 primes)
modulus:
 00:d1:...
publicExponent: 65537 (0x10001)
privateExponent:
 56:5e:...
prime1:
 00:ef:...
prime2:
 00:df:...
exponent1:
 00:cb:...
exponent2:
 00:df:...
coefficient:
 11:69:...
```

Let's examine at our rootCA-key.pem

```
openssl x509 -help
Usage: x509 [options]
Valid options are:
  -help          Display this summary
  -inform format  Input format - default PEM (one of DER or PEM)
  -in infile      Input file - default stdin
  -outform format Output format - default PEM (one of DER or PEM)
  -out outfile    Output file - default stdout
  -keyform PEM|DER Private key format - default PEM
  -passin val     Private key password/pass-phrase source
  -serial         Print serial number value
  -subject_hash   Print subject hash value
  -issuer_hash    Print issuer hash value
  -hash           Synonym for -subject_hash
  ...
```

OpenSSL x509 help

```
openssl x509 -in rootCA.pem -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      ba:26:56:af:26:bd:3c:1a:e5:05:9d:fa:0b:83:40:26
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: O = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
    Validity
      Not Before: Feb 18 22:34:27 2020 GMT
      Not After : Feb 18 22:34:27 2030 GMT
    Subject: O = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (3072 bit)
      Modulus:
        00:d1:...
      Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Key Usage: critical
        Certificate Sign
      X509v3 Basic Constraints: critical
        CA:TRUE, pathlen:0
      X509v3 Subject Key Identifier:
        B9:D5:B3:06:55:B4:E6:CE:CB:CB:56:B3:4A:35:96:A3:AA:5F:2D:C4
    Signature Algorithm: sha256WithRSAEncryption
      c1:c2:...
```

Let's examine at our rootCA.pem public certificate.


```
openssl x509 -in rootCA.pem -noout -serial  
serial=BA2656AF26BD3C1AE5059DFA0B834026
```

Serial Number - Unique per Certificate Authority

```
openssl x509 -in rootCA.pem -noout -dates  
notBefore=Feb 18 22:34:27 2020 GMT  
notAfter=Feb 18 22:34:27 2030 GMT
```

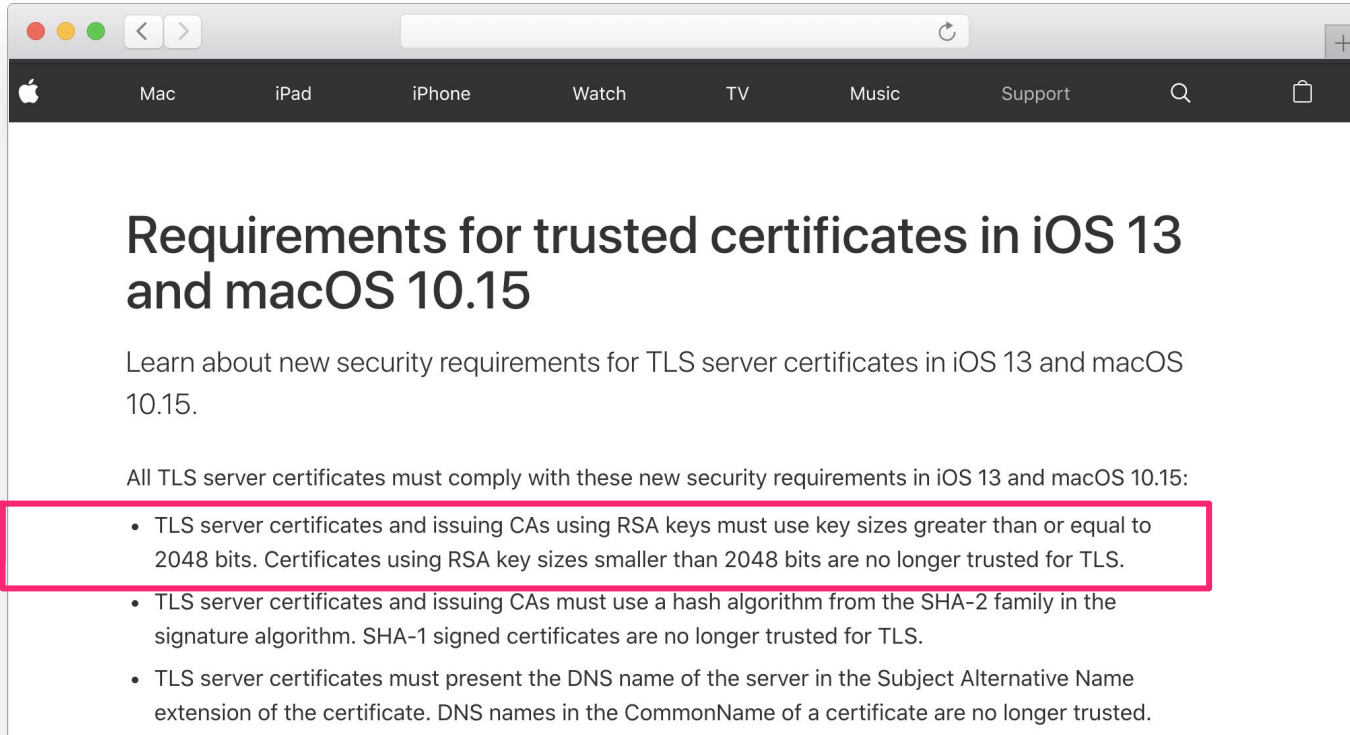
Validity Dates - “Expired Certificates”

Quick Q/A

Tangent Time

iOS 13 and macOS 10.15 - Gotchya

<https://support.apple.com/en-us/HT210176>

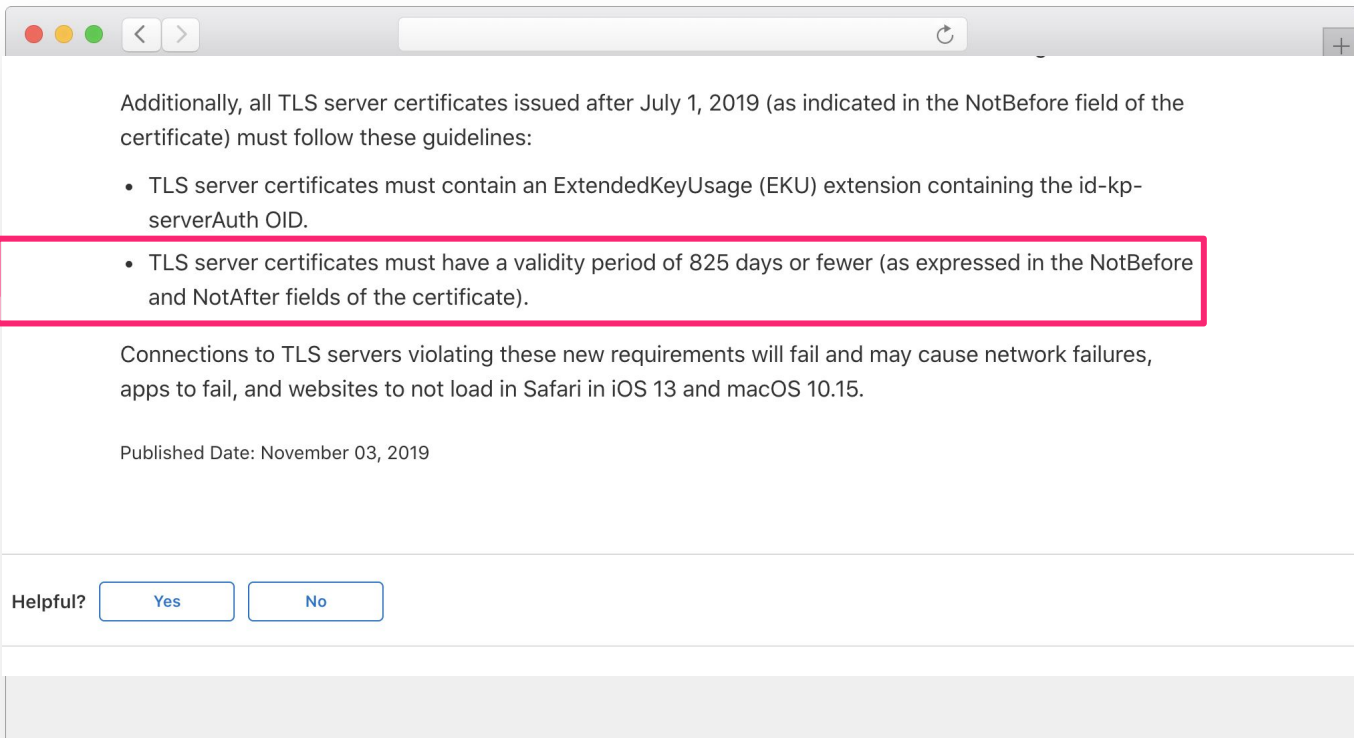


iOS 13 and macOS 10.15 - Gotchya

<https://support.apple.com/en-us/HT210176>



Why must we play these games?



Shorter Certification Expiration

Pros

- Limit damage from:
 - Key compromise
 - Mis-issuance
- Encourage Automation

What about?

- Wildcard usage
 - *.example.com
- Host security
- Issuer security
- Transparency in the industry

#GarbagelsGarbage



```
$ openssl x509 -in /Users/ascherger/Library/Application\ Support/mkcert/rootCA.pem -noout -startdate  
notBefore=Apr 16 03:15:22 2020 GMT
```



```
$ openssl x509 -in /Users/ascherger/Library/Application\ Support/mkcert/rootCA.pem -noout -enddate  
notAfter=Apr 16 03:15:22 2030 GMT
```

```
$ openssl x509 -in localhost+1.pem -noout -startdate  
notBefore=Jun 1 00:00:00 2019 GMT
```



```
$ openssl x509 -in localhost+1.pem -noout -enddate  
notAfter=Jun 5 14:13:28 2030 GMT
```

FiloSottile/mkcert [GH-174](#)

All better.

```
openssl x509 -in rootCA.pem -noout -issuer  
issuer=0 = mkcert development CA,  
        OU = ascherger@incontrol (Alan Scherger),  
        CN = mkcert ascherger@incontrol (Alan Scherger)
```

Issuer - who created the cert

```
openssl x509 -in rootCA.pem -noout -subject  
subject=0 = mkcert development CA,  
          OU = ascherger@incontrol (Alan Scherger),  
          CN = mkcert ascherger@incontrol (Alan Scherger)
```

Subject - who is the cert

```
openssl x509 -in rootCA.pem -noout -issuer  
issuer=0 = mkcert development CA,  
OU = ascherger@incontrol (Alan Scherger),  
CN = mkcert ascherger@incontrol (Alan Scherger)
```

```
openssl x509 -in rootCA.pem -noout -subject  
subject=0 = mkcert development CA,  
OU = ascherger@incontrol (Alan Scherger),  
CN = mkcert ascherger@incontrol (Alan Scherger)
```

For Root Certificates, they should always be the same.

```
openssl x509 -in rootCA.pem -noout -ext keyUsage  
X509v3 Key Usage: critical  
Certificate Sign
```

Key Usage

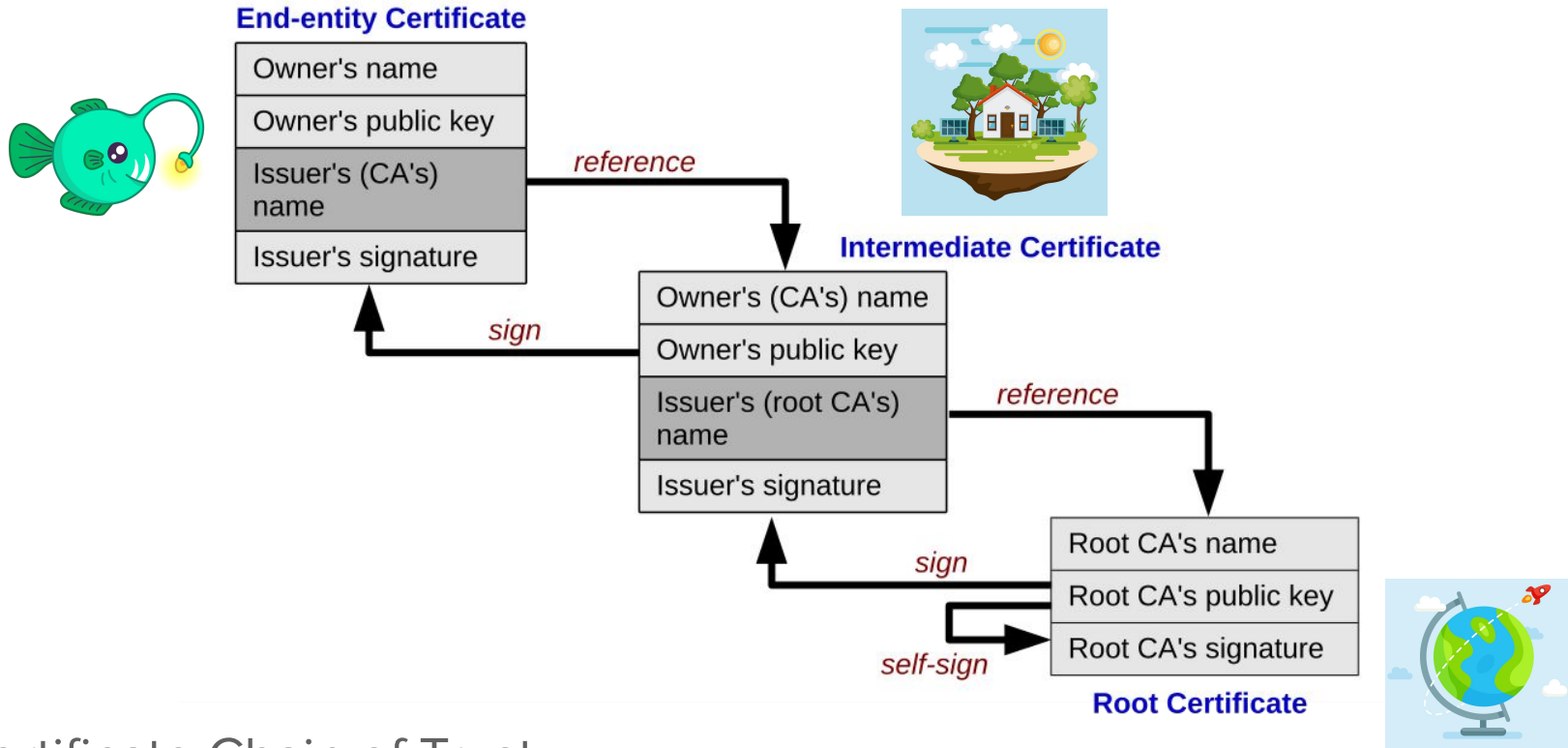
```
openssl x509 -in rootCA.pem -noout -ext subjectKeyIdentifier  
X509v3 Subject Key Identifier:
```



B9:D5:B3 06:55:B4:E6:CE:CB:CB:56:B3:4A:35:96:A3:AA:5F:2D:C4

Subject Key Identifier

Offroad Time



Certificate Chain of Trust

And we're back.

```
mkcert localhost 127.0.0.1
```

Using the local CA at `"/home/ascherger/.local/share/mkcert"` ✨

Created a new certificate valid for the following names 📄

- "localhost"
- "127.0.0.1"

The certificate is at `"./localhost+1.pem"` and the key at `"./localhost+1-key.pem"` ✓

Create End-Entity/Leaf Certificate

```
openssl x509 -in localhost+1.pem -noout -dates  
notBefore=Jun  1 00:00:00 2019 GMT  
notAfter=Jun  8 22:47:51 2030 GMT
```

Dates

```
openssl x509 -in localhost+1.pem -noout -issuer  
issuer=0 = mkcert development CA,  
        OU = ascherger@incontrol (Alan Scherger),  
        CN = mkcert ascherger@incontrol (Alan Scherger)
```

```
openssl x509 -in localhost+1.pem -noout -subject  
subject=0 = mkcert development certificate,  
        OU = ascherger@incontrol (Alan Scherger)
```

Issuer (who **created** the cert) & Subject (who **is** the cert)

Root Cert:

```
openssl x509 -in rootCA.pem -noout -ext subjectKeyIdentifier  
X509v3 Subject Key Identifier:
```



```
B9:D5:B3:06:55:B4:E6:CE:CB:CB:56:B3:4A:35:96:A3:AA:5F:2D:C4
```

Localhost Cert:

```
openssl x509 -in localhost+1.pem -noout -ext authorityKeyIdentifier  
X509v3 Authority Key Identifier:
```




```
keyid:B9:D5:B3:06:55:B4:E6:CE:CB:CB:56:B3:4A:35:96:A3:AA:5F:2D:C4
```


Authority Key Identifier matches CA Subject Key Identifier

```
openssl x509 -in localhost+1.pem -noout -ext basicConstraints  
X509v3 Basic Constraints: critical  
CA:FALSE
```

Basic Constraints - this is not a CA.



```
openssl x509 -in localhost+1.pem -noout -ext keyUsage
X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
```



```
openssl x509 -in localhost+1.pem -noout -ext extendedKeyUsage
X509v3 Extended Key Usage:
                TLS Web Server Authentication
```

Key Usage and Extended Key Usage



```
mkcert -client localhost 127.0.0.1
```

Using the local CA at `"/home/ascherger/.local/share/mkcert"` ✨

Created a new certificate valid for the following names 📄

- "localhost"
- "127.0.0.1"

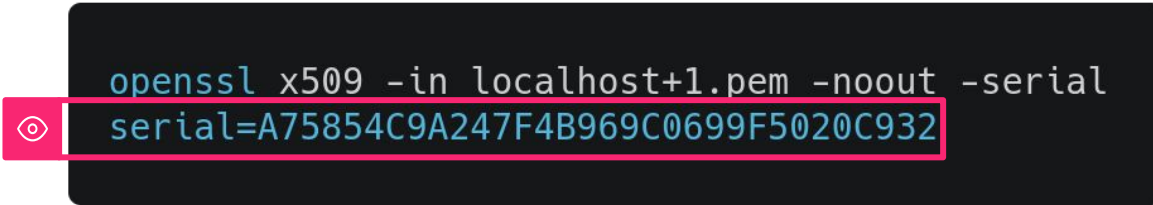
The certificate is at `"/.localhost+1-client.pem"` and the key at `"/.localhost+1-client-key.pem"` ✓



```
openssl x509 -in localhost+1-client.pem -noout -ext extendedKeyUsage  
X509v3 Extended Key Usage:
```

```
TLS Web Client Authentication, TLS Web Server Authentication
```

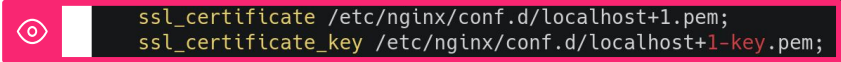
Let's make a Client cert, and see the difference.



```
openssl x509 -in localhost+1.pem -noout -serial  
serial=A75854C9A247F4B969C0699F5020C932
```

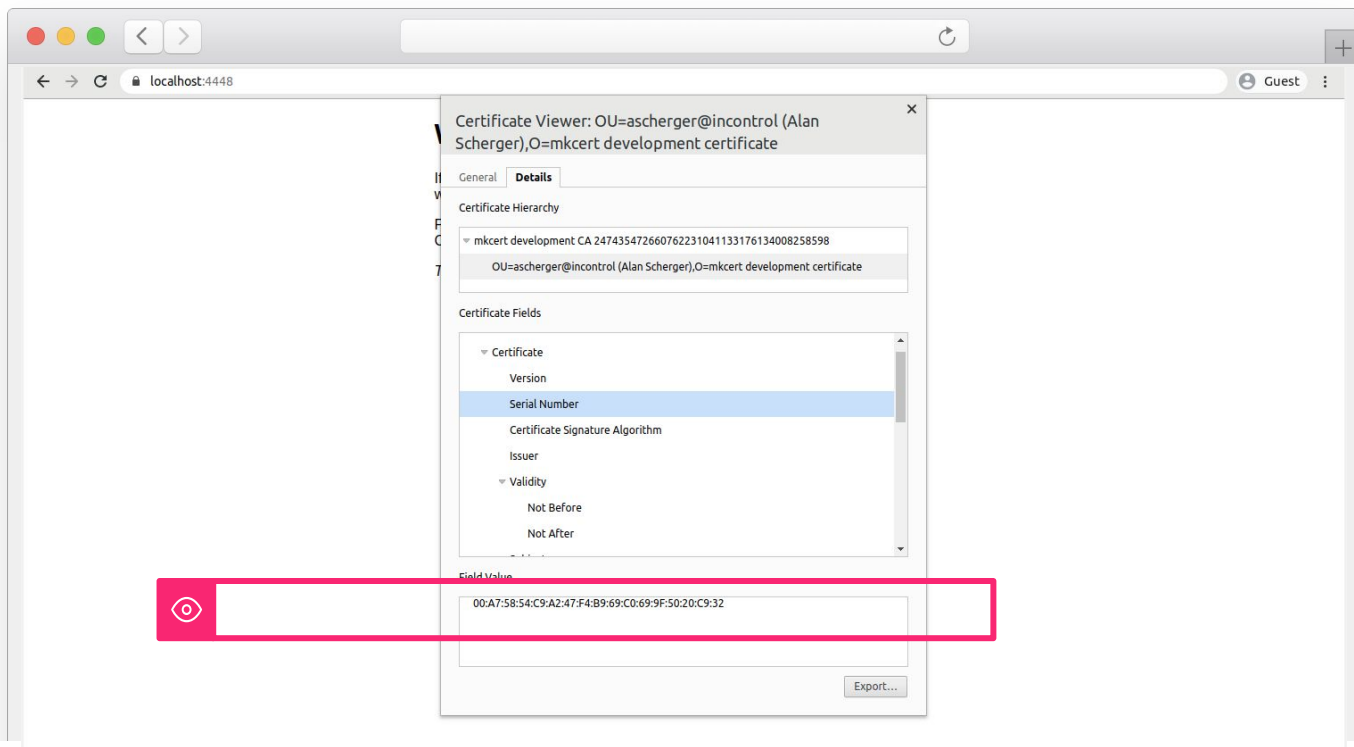
Remember the Web Server Serial Number.

```
docker run -p 4448:443 -v "`pwd`::/etc/nginx/conf.d" nginx
```

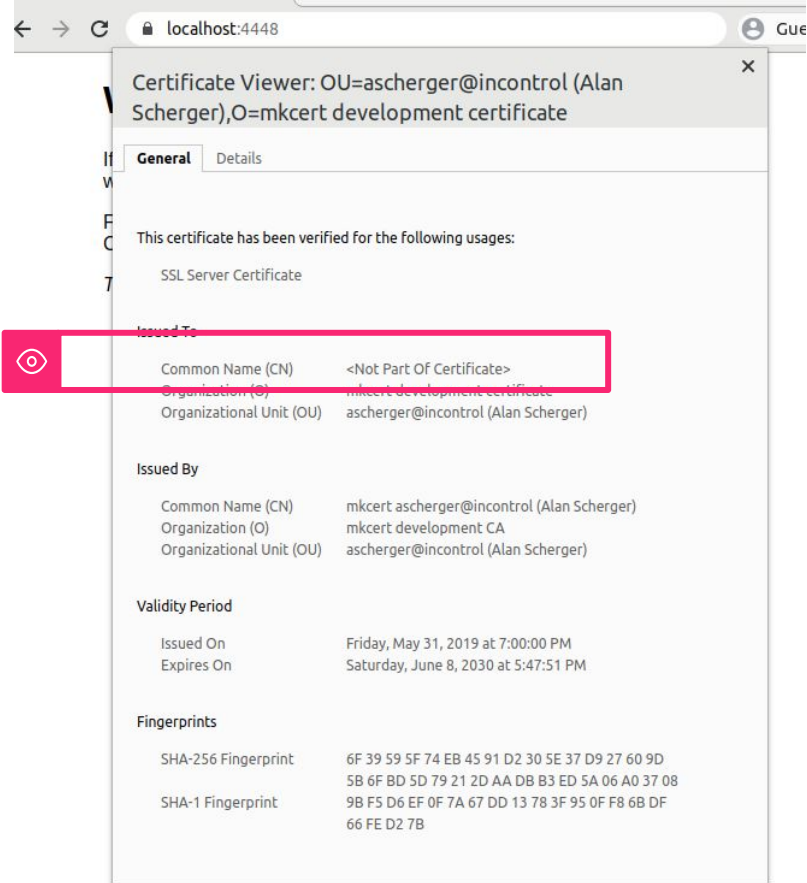


```
server {  
    listen [::]:443 ssl http2 ipv6only=on;  
    listen 443 ssl http2;  
    server_name localhost;  
    ssl_certificate /etc/nginx/conf.d/localhost+1.pem;  
    ssl_certificate_key /etc/nginx/conf.d/localhost+1-key.pem;  
    location / {  
        root /usr/share/nginx/html;  
        index index.html index.htm;  
    }  
    error_page 500 502 503 504 /50x.html;  
    location = /50x.html {  
        root /usr/share/nginx/html;  
    }  
}
```

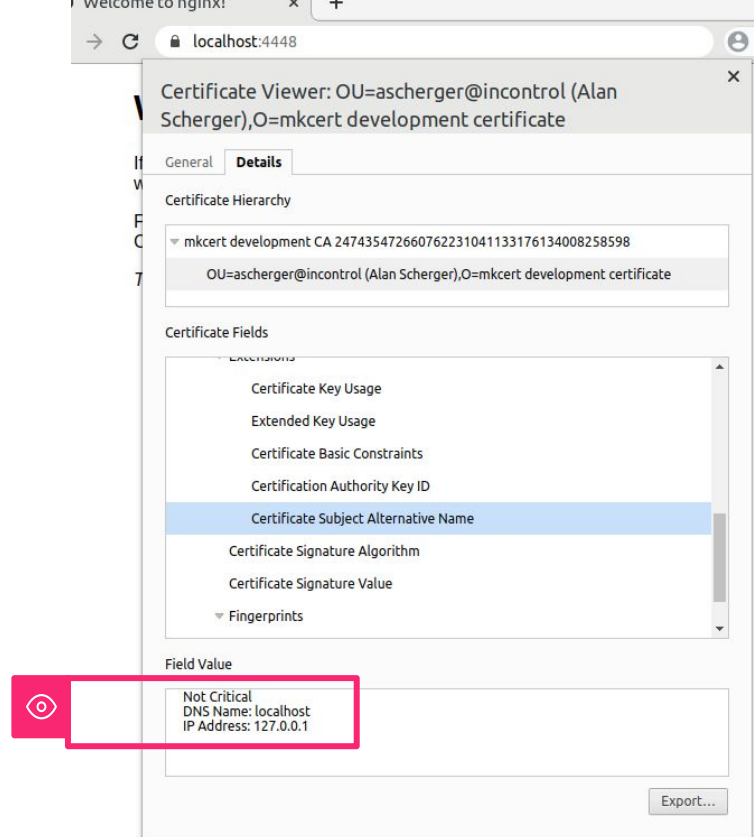
Spin up a web server!



We see the serial number matches.



No Common Name




Localhost is listed in Subject Alternative Name

```
mkcert 192.168.1.100
Using the local CA at "/home/ascherger/.local/share/mkcert" ✨

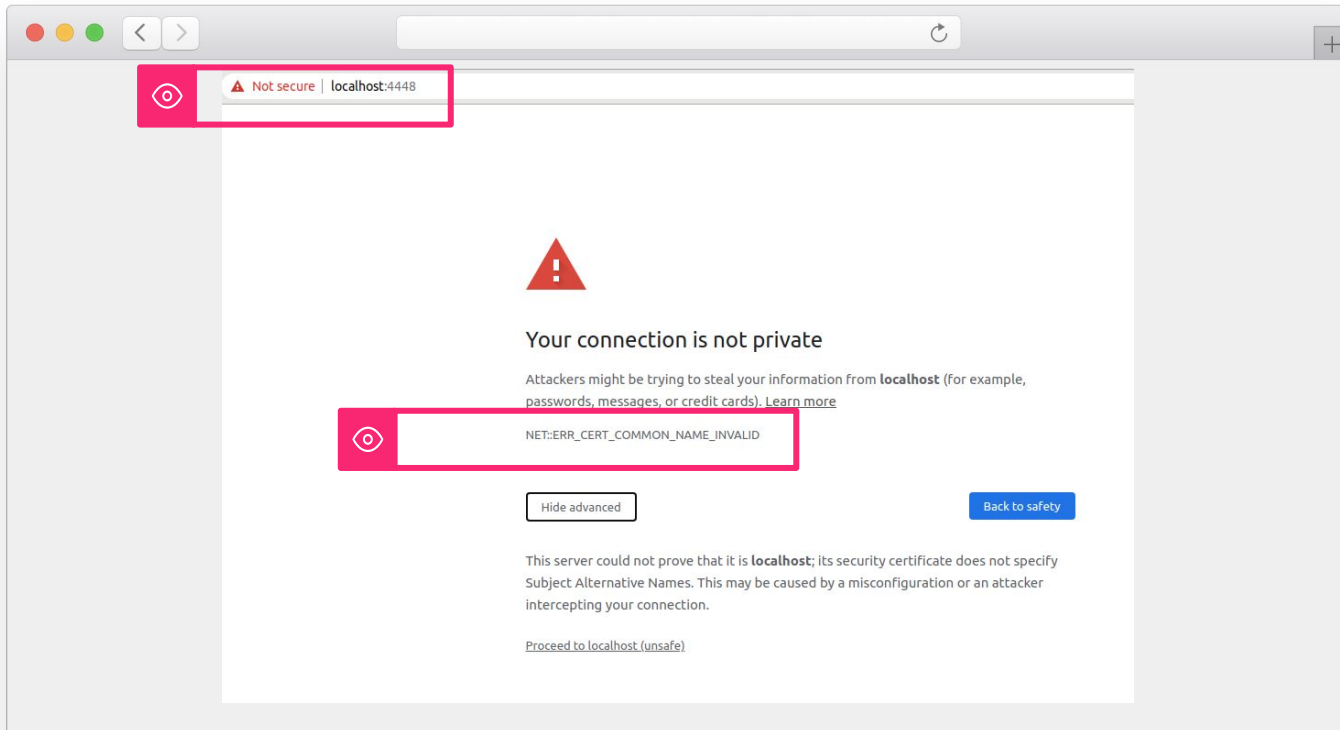
Created a new certificate valid for the following names 📄
- "192.168.1.100"

The certificate is at "./192.168.1.100.pem" and the key at "./192.168.1.100-key.pem" ✓
```

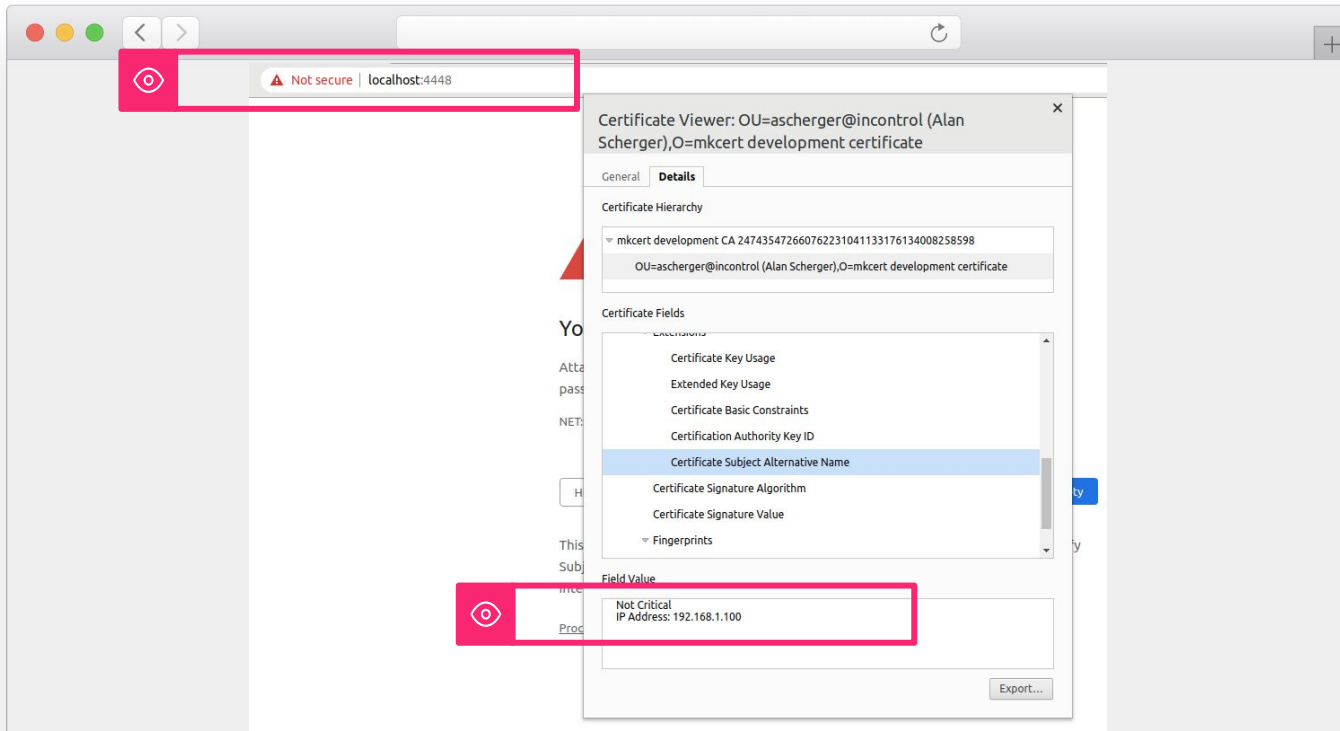


```
server {
    listen [::]:443 ssl http2 ipv6only=on;
    listen 443 ssl http2;
    server_name localhost;
    ssl_certificate /etc/nginx/conf.d/192.168.1.100.pem;
    ssl_certificate_key /etc/nginx/conf.d/192.168.1.100-key.pem;
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

Make a bogus cert.



Common Name in invalid. **#GarbagersGarbage**



Address not found in Subject Alternative Name either.

But wait...

This is not our final form.

```
$ openssl s_client -connect google.com:443
CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, CN = GlobalSign
verify return:1
depth=1 C = US, O = Google Trust Services, CN = GTS CA 101
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google LLC, CN = *.google.com
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Mountain View, O = Google LLC, CN = *.google.com
  i:C = US, O = Google Trust Services, CN = GTS CA 101
 1 s:C = US, O = Google Trust Services, CN = GTS CA 101
  i:OU = GlobalSign Root CA - R2, O = GlobalSign, CN = GlobalSign
---
Server certificate
-----BEGIN CERTIFICATE-----
<certificate content>
-----END CERTIFICATE-----
subject=C = US, ST = California, L = Mountain View, O = Google LLC, CN = *.google.com

issuer=C = US, O = Google Trust Services, CN = GTS CA 101

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits

SSL handshake has read 3798 bytes and written 392 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
```

openssl s_client -connect google.com:443

```
openssl s_client -connect localhost:4448
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)
verify error:num=21:unable to verify the first certificate
verify return:1
---
Certificate chain
 0 s:0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)
  i:0 = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
---
Server certificate
<cert>
subject=0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)

issuer=0 = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1757 bytes and written 386 bytes
Verification error: unable to verify the first certificate
---
```

Certificate cannot be verified - because the CA was not installed.



```
openssl version -d
```

```
OPENSSLDIR: "/home/ascherger/miniconda3/ssl"
```

```
$ ls -al /home/ascherger/miniconda3/ssl/
total 264
drwxrwxr-x  3 ascherger ascherger  4096 Mar 26 18:27 .
drwxrwxr-x 15 ascherger ascherger  4096 Mar 26 18:27 ..
-rw-rw-r--  6 ascherger ascherger 223687 Jan 30 09:16 cacert.pem
lrwxrwxrwx  1 ascherger ascherger   10 Mar 26 18:27 cert.pem -> cacert.pem
-rw-rw-r--  2 ascherger ascherger  412 Feb 10 11:04 ct_log_list.cnf
-rw-rw-r--  2 ascherger ascherger  412 Feb 10 11:04 ct_log_list.cnf.dist
drwxrwxr-x  2 ascherger ascherger  4096 Mar 26 18:27 misc
-rw-rw-r--  2 ascherger ascherger 10909 Feb 10 11:04 openssl.cnf
-rw-rw-r--  2 ascherger ascherger 10909 Feb 10 11:04 openssl.cnf.dist
```

Newer versions of OpenSSL can show you its current dir.

```
cat ~/.local/share/mkcert/rootCA.pem >> ~/.miniconda3/ssl/cacert.pem
```

```
openssl s_client -connect localhost:4448
CONNECTED(00000003)
Can't use SSL_get_servername
depth=1 0 = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
verify return:1
depth=0 0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)
verify return:1
---
Certificate chain
 0 s:0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)
  i:0 = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
---
Server certificate
<certificate>
subject=0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)

issuer=0 = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1757 bytes and written 386 bytes
Verification: OK
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session: <session data>
---
```

Adding the certificate, gets us a valid connection.

But wait...

We don't even need NGINX.

```
openssl s_server \  
-key localhost+1-key.pem \  
-cert localhost+1.pem \  
-CAfile ~/Library/Application\ Support/mkcert/rootCA.pem \  
-www -accept 4448 -Verify 1
```

```
openssl s_client \  
-CAfile ~/Library/Application\ Support/mkcert/rootCA.pem \  
-connect localhost:4448
```

OpenSSL has an **s_server**

```
ACCEPT
```

```
depth=1 0 = mkcert development CA, OU = ascherger@MacBook-Pro.localdomain (Alan Scherger), CN = mkcert  
ascherger@MacBook-Pro.localdomain (Alan Scherger)
```

```
verify return:1
```

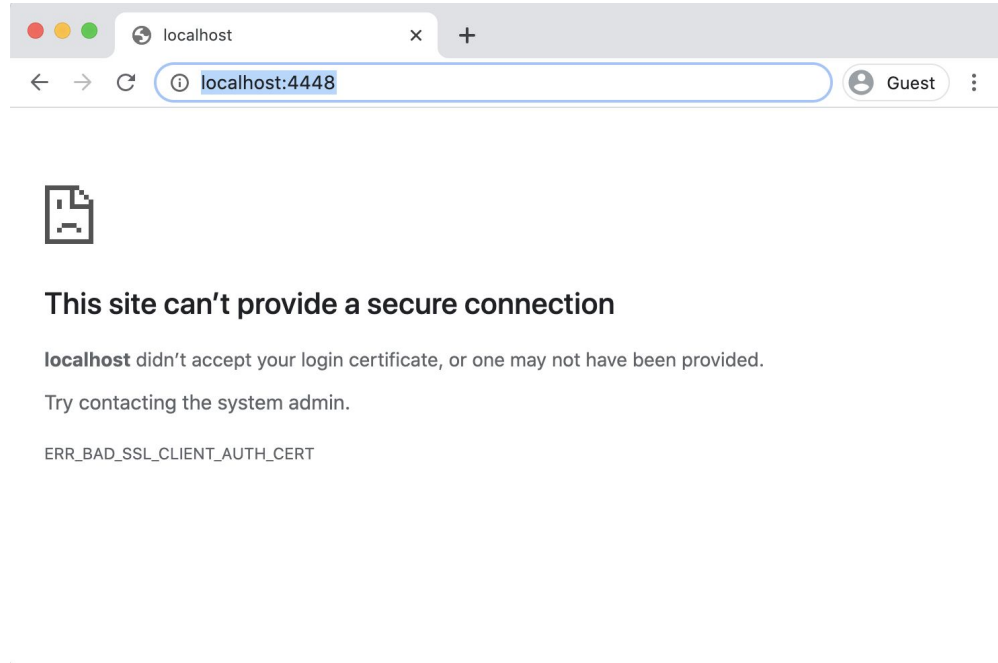
```
depth=0 0 = mkcert development certificate, OU = ascherger@MacBook-Pro.localdomain (Alan Scherger)
```

```
verify return:1
```

OpenSSL can validate Client certificates.


```
openssl pkcs12 -export \  
-inkey ./localhost+1-client-key.pem \  
-in ./localhost+1-client.pem \  
-certfile ~/Library/Application\ Support/mkcert/rootCA.pem \  
-out local.pfx
```

```
Keychain >  
login > My Certificates >  
File > Import Items
```



OpenSSL can validate Client certificates.



ascherger@MacBook-Pro.localdomain (Alan Scherger)

Issued by: mkcert ascherger@MacBook-Pro.localdomain (Alan Scherger)

Expires: Saturday, June 8, 2030 at 9:17:43 PM Central Daylight Time

✓ This certificate is valid

Name

▶ ascherger@MacBook-Pro.localdomain (Alan Scherger)



ascherger@MacBook-Pro.localdomain (Alan Scherger)

Issued by: mkcert ascherger@MacBook-Pro.localdomain (Alan Scherger)

Expires: Saturday, June 8, 2030 at 9:17:43 PM Central Daylight Time

✗ This certificate was signed by an untrusted issuer

▼ **Trust**

When using this certificate: Use Custom Settings ▾ ?

Secure Sockets Layer (SSL) Always Trust ▾

Secure Mail (S/MIME) no value specified ▾

Extensible Authentication (EAP) no value specified ▾

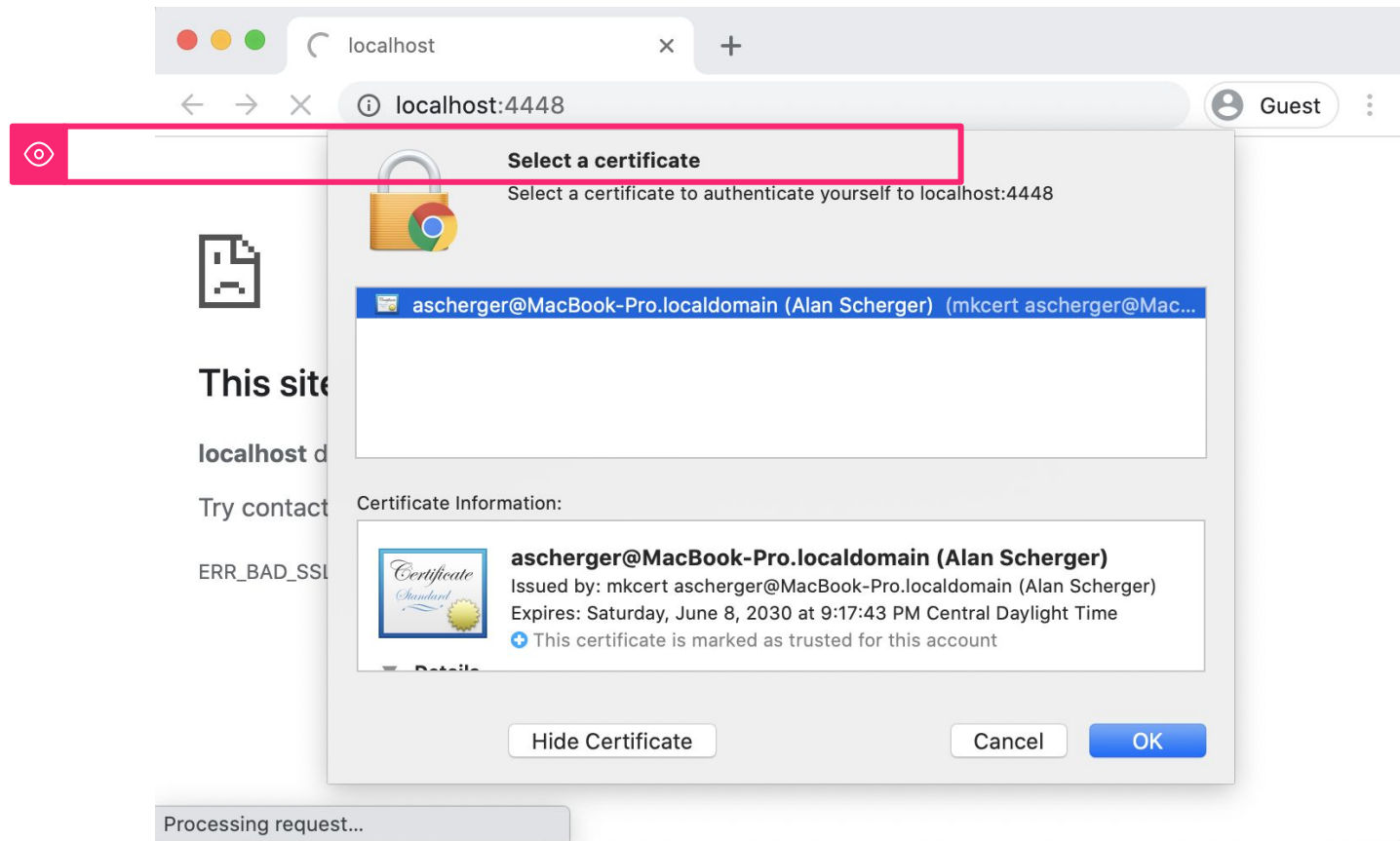
IP Security (IPsec) no value specified ▾

Code Signing no value specified ▾

Time Stamping no value specified ▾

X.509 Basic Policy no value specified ▾

Always Trust SSL



Select our Client Certificate

To Recap

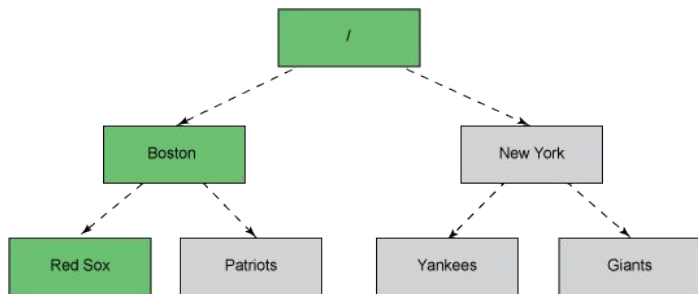
- mkcert to create local certs
 - OpenSSL to Debug Certs
 - Nginx to play with certs
 - OpenSSL CLI can be a server and a client
 - Error messages:
 - #GarbagelsGarbage
 - badssl.com
-

Quick Q/A

Zookeeper gets
mTLS support!

Architecture Overview

Hierarchical Key-Value Store



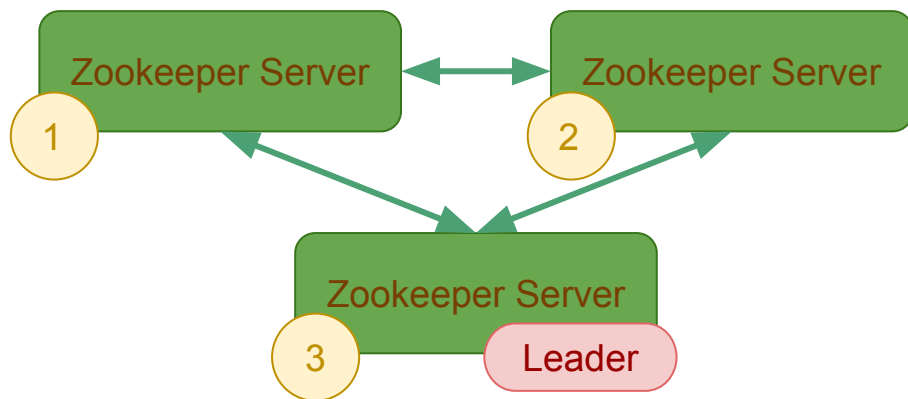
Minimum 3 node **Ensemble**

Zookeeper Server

Zookeeper Server

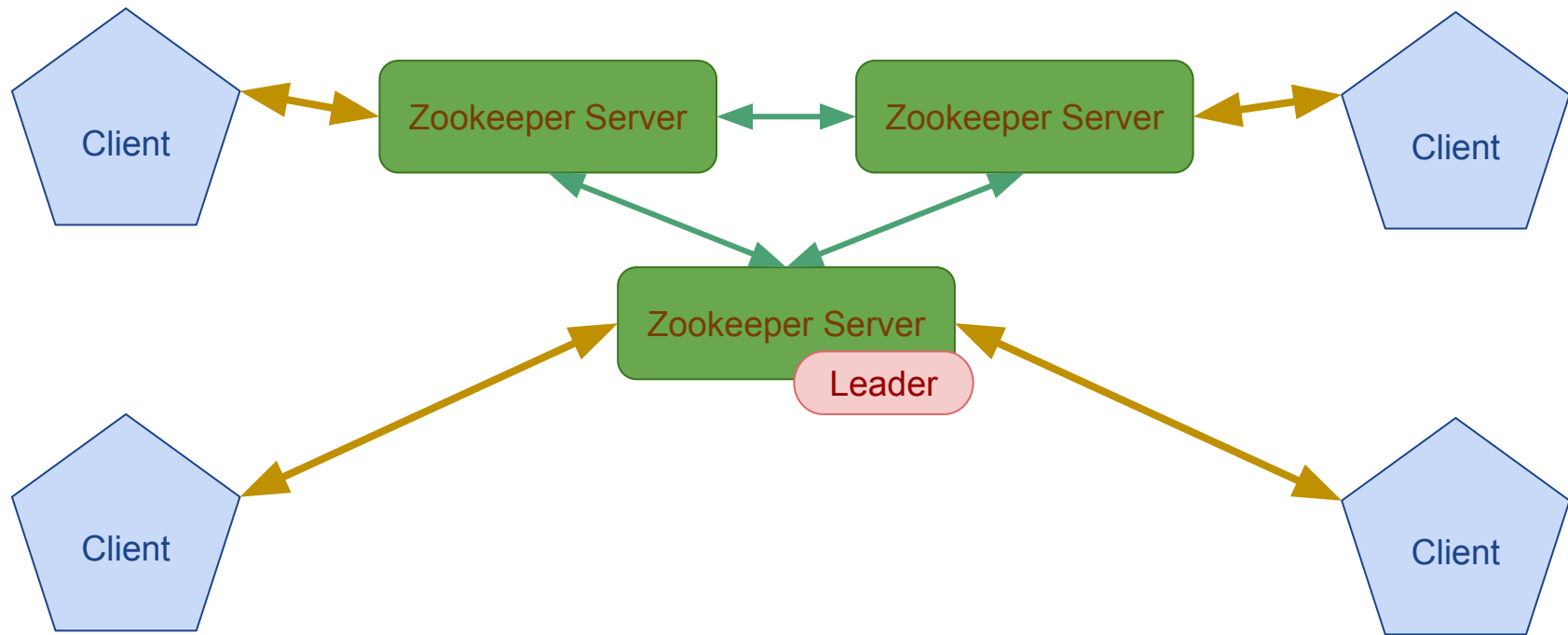
Zookeeper Server

The Ensemble forms a **Quorum**, and elects a **Leader**.

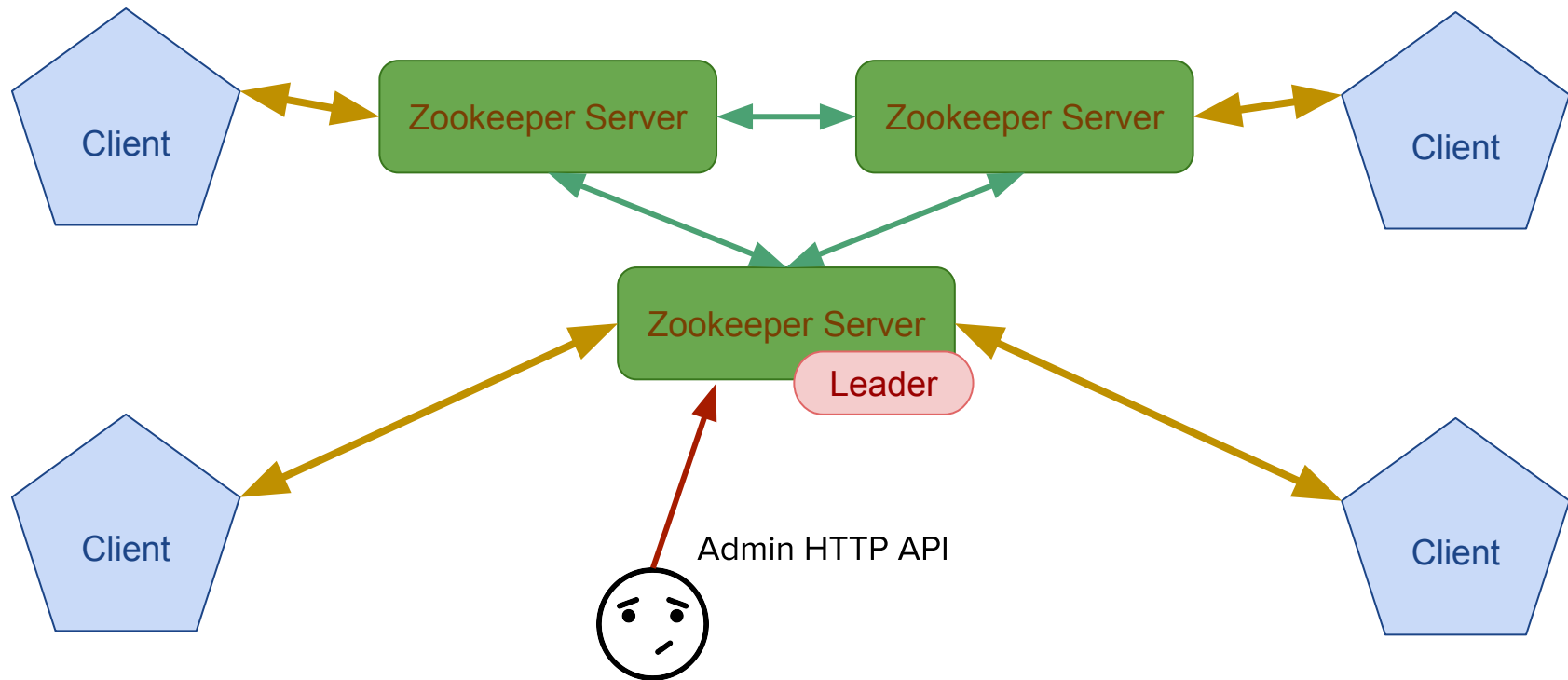


```
server.1=zk-1.private.${domain}:2888:3888  
server.2=zk-2.private.${domain}:2888:3888  
server.3=zk-3.private.${domain}:2888:3888
```

And **Clients** connect to the servers.



Admins connect to them.



Why do we care about mTLS?

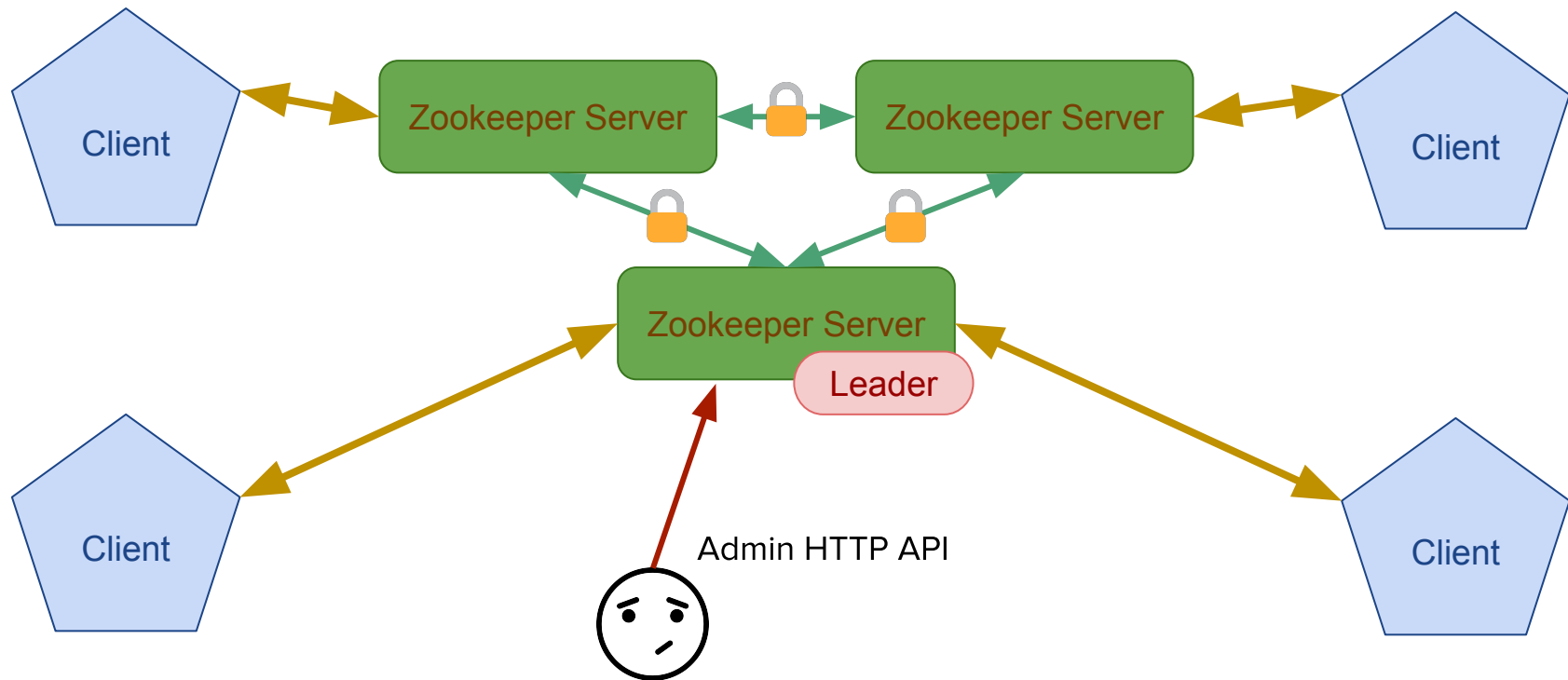
- Can you trust your hypervisor?
 - XEN CVE Count: 296
- Can you trust all the hardware in your Datacenter?
 - Intel Management Engine
 - AWS Nitro
 - Google Titan
- Can you trust the other VMs running next to you in hardware?
 - Meltdown and Spectre
- How do you know you aren't currently being owned by a Zero-day?
 - New threats that your monitoring cannot detect
- How do you know the coffee pot on a misconfigured VLAN isn't stealing your data?

How's your
insider threat
training?

Build a chain of trust.

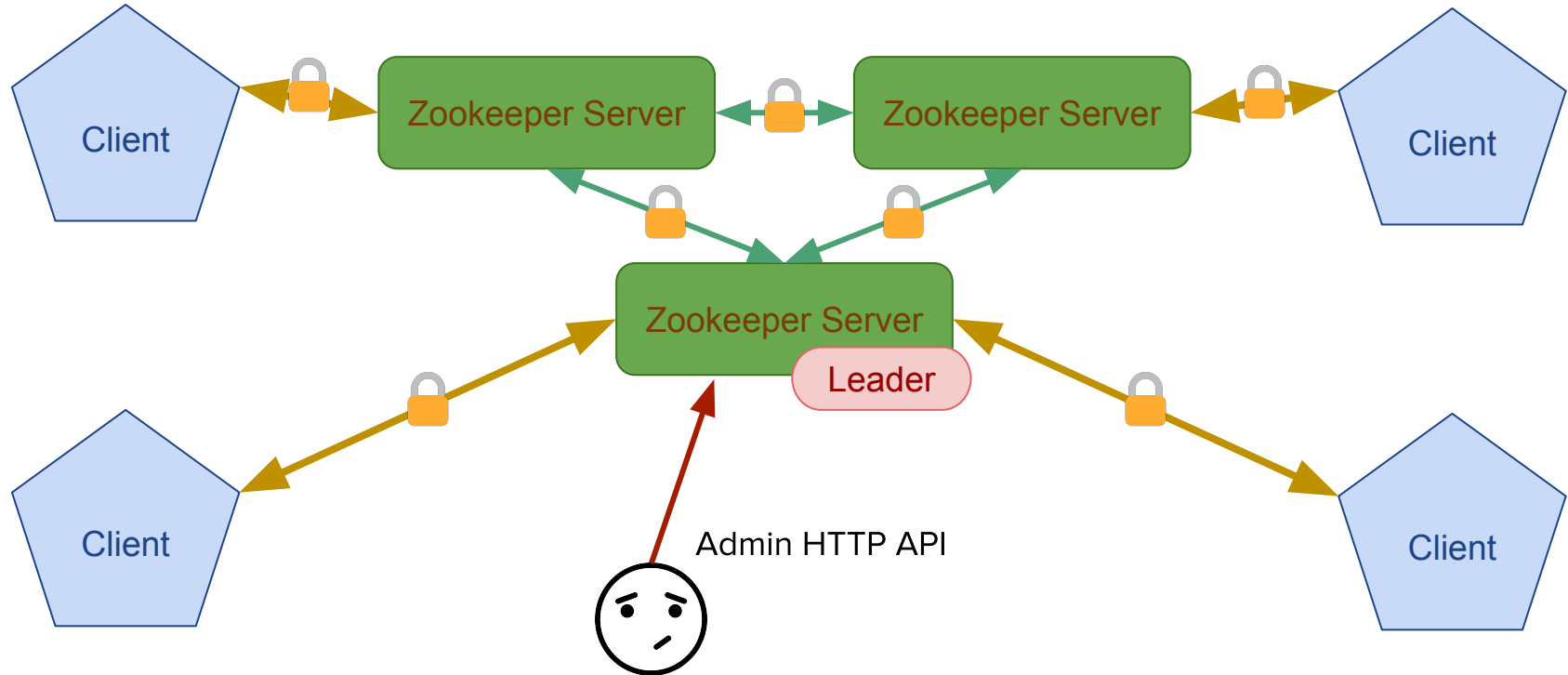
Protect the server
communication.

We can protect **Quorum** traffic.



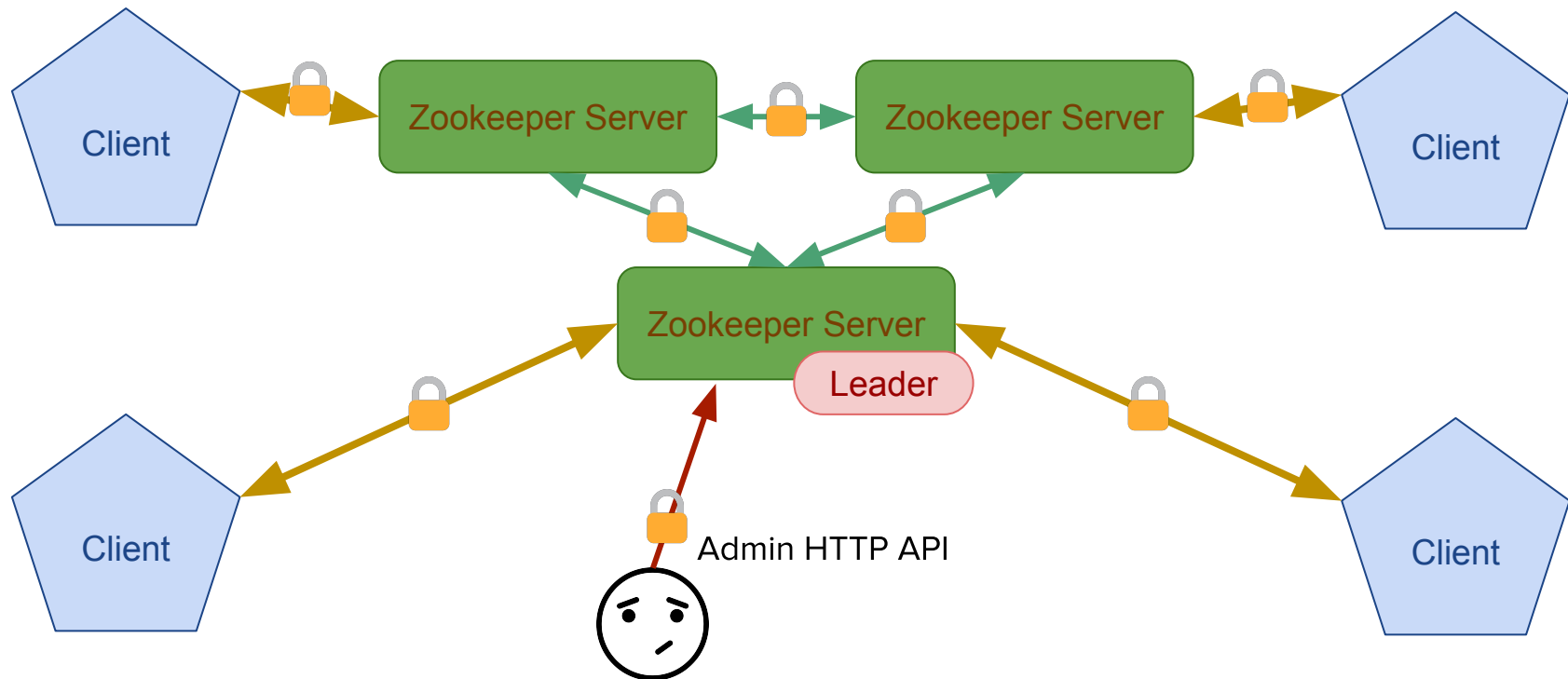
Protect the client
communication.

We can protect **Client** traffic.

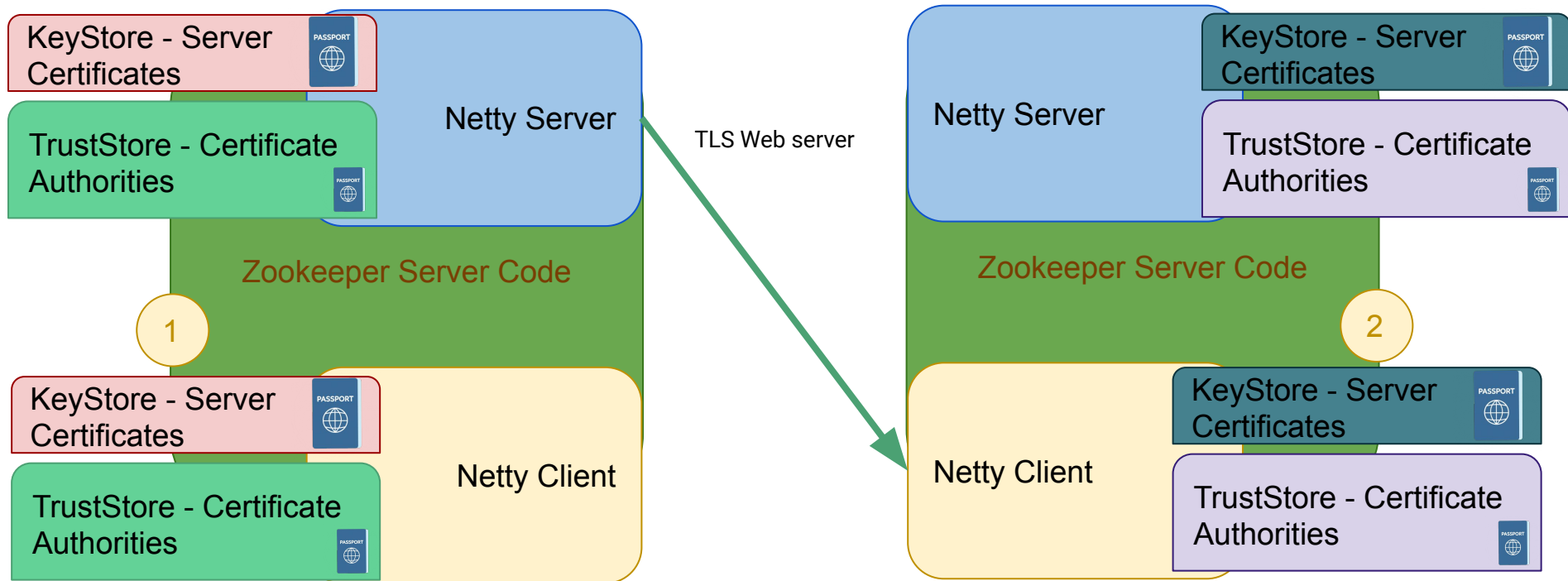


Protect how the
Admins administrate
the cluster.

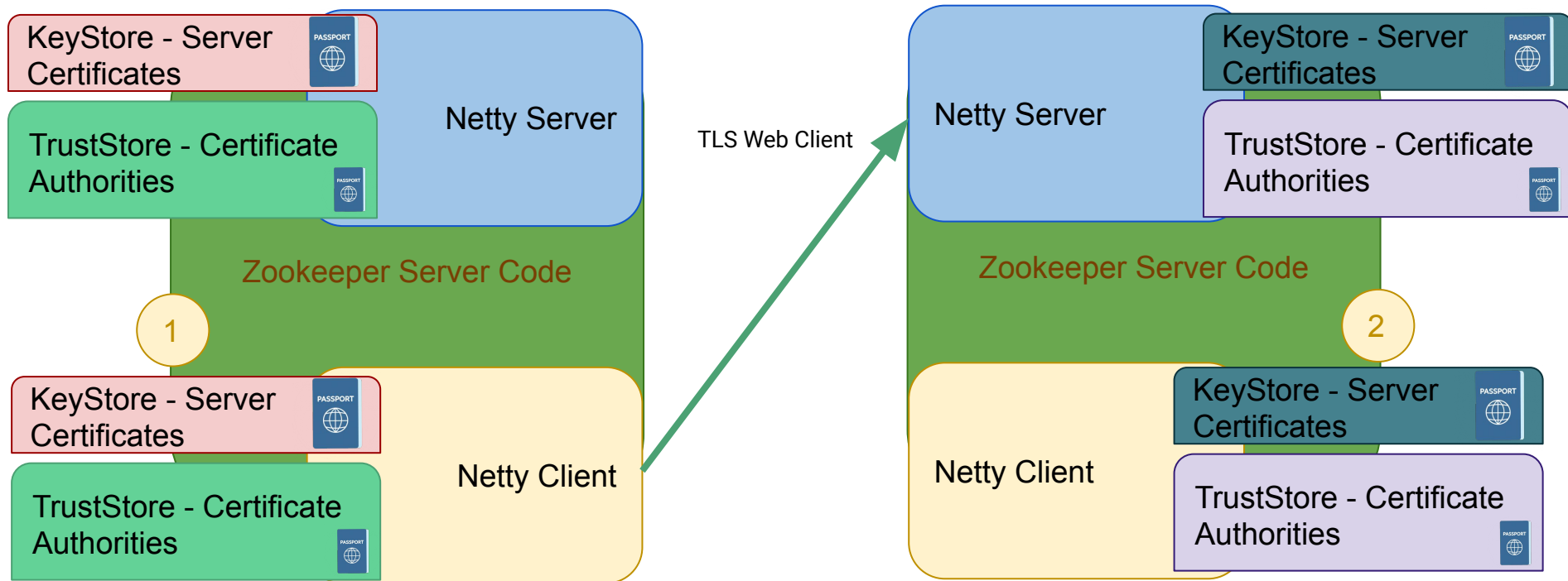
We can protect **Admin** access.



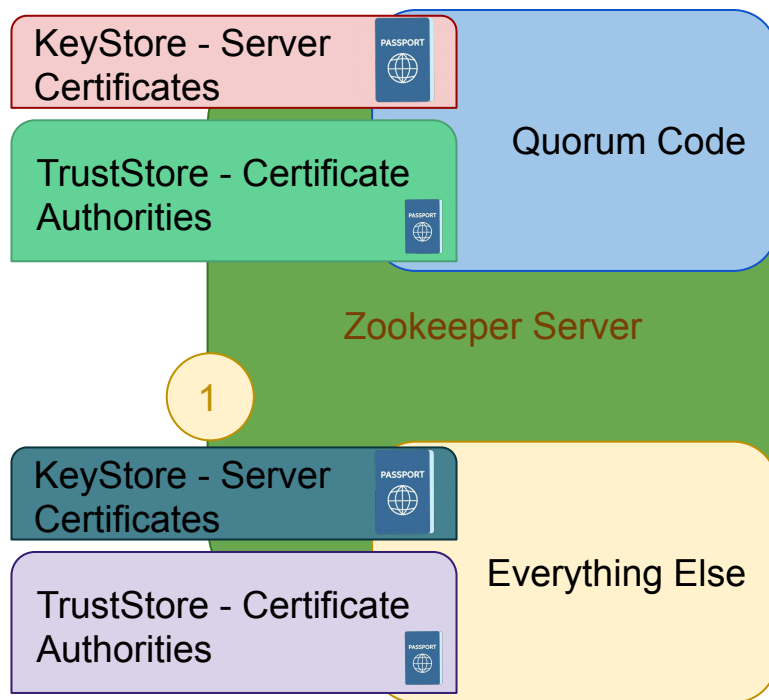
Zookeeper Quorum Code act as Servers & Clients



Zookeeper Quorum Code act as Servers & Clients



Zookeeper Configuration



Quorum (Server <-> Server) for Netty Server & Client

```
ssl.quorum.keyStore.location=/opt/zookeeper/conf/keystore.jks  
ssl.quorum.keyStore.password=password  
ssl.quorum.trustStore.location=/opt/zookeeper/conf/truststore.jks  
ssl.quorum.trustStore.password=password
```

Cluster <-> Client for Netty Server & Client

```
ssl.keyStore.location=/opt/zookeeper/conf/keystore.jks  
ssl.keyStore.password=password  
ssl.trustStore.location=/opt/zookeeper/conf/truststore.jks  
ssl.trustStore.password=password
```

```
keytool -genkeypair -alias zk-1.private.zkocean.hpy.dev \  
-keyalg RSA -keysize 2048 -dname "cn=zk-1.private.zkocean.hpy.dev" \  
-keypass password \  
-keystore /opt/zookeeper/conf/keystore.jks \  
-storepass password \  
-storetype pkcs12
```

zk-1.private.zkocean.hpy.dev	10.10.20.4
zk-1.zkocean.hpy.dev	64.227.26.221

Make some servers, setup some DNS - use private IPs and addresses for hostnames.

2

```
Caused by: java.security.cert.CertificateException: Failed to verify both host address and host name
    at org.apache.zookeeper.common.ZKTrustManager.performHostVerification(ZKTrustManager.java:163)
    at org.apache.zookeeper.common.ZKTrustManager.checkClientTrusted(ZKTrustManager.java:79)
    at sun.security.ssl.ServerHandshaker.clientCertificate(ServerHandshaker.java:2037)
    ... 26 more
```

1

```
Caused by: javax.net.ssl.SSLPeerUnverifiedException: Certificate for <10.10.20.2> doesn't match common
name of the certificate subject: zk-2.private.zkocean.hpy.dev
    at org.apache.zookeeper.common.ZKHostnameVerifier.matchCN(ZKHostnameVerifier.java:238)
    at org.apache.zookeeper.common.ZKHostnameVerifier.verify(ZKHostnameVerifier.java:184)
```

I agree - 10.10.20.2 != zk-2.private.zkocean.hpy.dev
But why do we think it ever would?

matchCN does throw this error, now let's go UP the stack.

```
private static void matchCN(final String host, final String cn) throws SSLException {
    final String normalizedHost = host.toLowerCase(Locale.ROOT);
    final String normalizedCn = cn.toLowerCase(Locale.ROOT);
    if (!matchIdentityStrict(normalizedHost, normalizedCn)) {
        throw new SSLPeerUnverifiedException("Certificate for <" + host + "> doesn't match "
            + "common name of the certificate subject: " + cn);
    }
}
```

<https://github.com/apache/zookeeper/blob/79a99ac97cea942967a2a08e7873a0cea9a2e046/zookeeper-server/src/main/java/org/apache/zookeeper/common/ZKHostnameVerifier.java#L234-L241>

Turns out CN matching is a thing of the past.

```
// CN matching has been deprecated by rfc2818 and can be used
// as fallback only when no subjectAlts are available
final X500Principal subjectPrincipal = cert.getSubjectX500Principal();
final String cn = extractCN(subjectPrincipal.getName(X500Principal.RFC2253));
if (cn == null) {
    throw new SSLException("Certificate subject for <"
        + host
        + "> doesn't contain "
        + "a common name and does not have alternative names");
}
matchCN(host, cn);
```

<https://github.com/apache/zookeeper/blob/79a99ac97cea942967a2a08e7873a0cea9a2e046/zookeeper-server/src/main/java/org/apache/zookeeper/common/ZKHostnameVerifier.java#L173-L186>

Wait wut... RFC 2818 - May 2000

3.1 Server Identity

If a subjectAltName extension of type dNSName is present, that MUST be used as the identity. Otherwise, the (most specific) Common Name field in the Subject field of the certificate MUST be used. **Although the use of the Common Name is existing practice, it is deprecated and Certification Authorities are encouraged to use the dNSName instead.**



```
Caused by: java.security.cert.CertificateException: Failed to verify both host address and host name
    at org.apache.zookeeper.common.ZKTrustManager.performHostVerification(ZKTrustManager.java:163)
    at org.apache.zookeeper.common.ZKTrustManager.checkClientTrusted(ZKTrustManager.java:79)
    at sun.security.ssl.ServerHandshaker.clientCertificate(ServerHandshaker.java:2037)
    ... 26 more
Caused by: javax.net.ssl.SSLPeerUnverifiedException: Certificate for <10.10.20.2> doesn't match common
name of the certificate subject: zk-2.private.zkocean.hpy.dev
    at org.apache.zookeeper.common.ZKHostnameVerifier.matchCN(ZKHostnameVerifier.java:238)
    at org.apache.zookeeper.common.ZKHostnameVerifier.verify(ZKHostnameVerifier.java:184)
```

Turns out the fact that it doesn't match Host Address or Host Name is actually the bigger problem.



```
@Override
public void checkClientTrusted(
    X509Certificate[] chain,
    String authType,
    Socket socket) throws CertificateException {
    x509ExtendedTrustManager.checkClientTrusted(chain, authType, socket);
    if (clientHostnameVerificationEnabled) {
        performHostVerification(socket.getInetAddress(), chain[0]);
    }
}
```

We will always use IP addresses to validate Client certificates.
Because the socket only ever has the address.

```
public class QuorumX509Util extends X509Util {  
  
    @Override  
    protected String getConfigPrefix() {  
        return "zookeeper.ssl.quorum.";  
    }  
  
    @Override  
    protected boolean shouldVerifyClientHostname() {  
        return true;  
    }  
  
}
```

Turns out we also **FORCE** verifying Client HostName - however the method name is ***misleading*** because the socket only ever has an **Address**.

Client Hostname verification is based on `sslServerHostnameVerificationEnabled` && effectively a hardcoded true.



```
boolean sslServerHostnameVerificationEnabled =  
config.getBoolean(this.getSslHostnameVerificationEnabledProperty(), true);  
boolean sslClientHostnameVerificationEnabled = sslServerHostnameVerificationEnabled &&  
shouldVerifyClientHostname();
```

<https://github.com/apache/zookeeper/blob/79a99ac97cea942967a2a08e7873a0cea9a2e046/zookeeper-server/src/main/java/org/apache/zookeeper/common/X509Util.java#L362-L363>

But wait...

The docs reference this
`ssl.quorum.clientAuth`
thing.

There's also `ssl.quorum.clientAuth=none` which controls netty's configuration.

```
if (!isClientSocket) {
    switch (clientAuth) {
        case NEED:
            sslParameters.setNeedClientAuth(true);
            break;
        case WANT:
            sslParameters.setWantClientAuth(true);
            break;
        default:
            sslParameters.setNeedClientAuth(false); // also clears the wantClientAuth flag according to docs
            break;
    }
}
```

<https://github.com/apache/zookeeper/blob/11c07921c15e2fb7692375327b53f26a583b77ca/zookeeper-server/src/main/java/org/apache/zookeeper/common/SSLContextAndOptions.java#L155-L167>

```

r.sh[2622]: 2020-06-09 03:41:49,502 [myid:1] - WARN [ListenerHandler-ZK-1-private.zklocal.npy.dev/10.10.20.4:3000:QuorumCnxManager@629] - Exception readin
r.sh[2622]: javax.net.ssl.SSLException: Connection has been shutdown: javax.net.ssl.SSLHandshakeException: Received fatal alert: certificate_unknown
r.sh[2622]:     at sun.security.ssl.SSLSocketImpl.checkEOF(SSLSocketImpl.java:1554)
r.sh[2622]:     at sun.security.ssl.AppInputStream.read(AppInputStream.java:95)
r.sh[2622]:     at org.apache.zookeeper.server.quorum.UnifiedServerSocket$UnifiedInputStream.read(UnifiedServerSocket.java:693)
r.sh[2622]:     at java.io.BufferedInputStream.fill(BufferedInputStream.java:246)
r.sh[2622]:     at java.io.BufferedInputStream.read1(BufferedInputStream.java:286)
r.sh[2622]:     at java.io.BufferedInputStream.read(BufferedInputStream.java:345)
r.sh[2622]:     at java.io.DataInputStream.readFully(DataInputStream.java:195)
r.sh[2622]:     at java.io.DataInputStream.readLong(DataInputStream.java:416)
r.sh[2622]:     at org.apache.zookeeper.server.quorum.QuorumCnxManager.handleConnection(QuorumCnxManager.java:601)
r.sh[2622]:     at org.apache.zookeeper.server.quorum.QuorumCnxManager.receiveConnection(QuorumCnxManager.java:554)
r.sh[2622]:     at org.apache.zookeeper.server.quorum.QuorumCnxManager$Listeners$ListenerHandler.acceptConnections(QuorumCnxManager.java:1079)
r.sh[2622]:     at org.apache.zookeeper.server.quorum.QuorumCnxManager$Listeners$ListenerHandler.run(QuorumCnxManager.java:1033)
r.sh[2622]:     at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
r.sh[2622]:     at java.util.concurrent.FutureTask.run(FutureTask.java:266)
r.sh[2622]:     at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
r.sh[2622]:     at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
r.sh[2622]:     at java.lang.Thread.run(Thread.java:748)
r.sh[2622]: Caused by: javax.net.ssl.SSLHandshakeException: Received fatal alert: certificate_unknown
r.sh[2622]:     at sun.security.ssl.Alerts.getSSLException(Alerts.java:198)
r.sh[2622]:     at sun.security.ssl.Alerts.getSSLException(Alerts.java:159)
r.sh[2622]:     at sun.security.ssl.SSLSocketImpl.recvAlert(SSLSocketImpl.java:2041)
r.sh[2622]:     at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:1145)
r.sh[2622]:     at sun.security.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1388)
r.sh[2622]:     at sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1416)

```

This configuration does work, as a client cert is never returned, however our verification logic (hard coded True) still explodes because there's now no certificate to check.

Quorum Certificates
therefore must have
IPs in them.

```
keytool -genkeypair -alias zk-1.private.zkocean.hpy.dev \  
-keyalg RSA -keysize 2048 -dname "cn=zk-1.private.zkocean.hpy.dev" \  
-keypass password \  
-keystore /opt/zookeeper/conf/keystore.jks \  
-storepass password \  
-ext san=ip:10.10.20.4 \  
-storetype pkcs12
```

Add IP addresses to certs.

```
root@zk-1:~# curl -s localhost:8080/commands/leader | jq .
```

```
{  
  "is_leader": false,  
  "leader_id": 3,  
  "leader_ip": "zk-3.private.zkocean.hpy.dev",  
  "command": "leader",  
  "error": null  
}
```

```
root@zk-2:~# curl -s localhost:8080/commands/leader | jq .
```

```
{  
  "is_leader": false,  
  "leader_id": 3,  
  "leader_ip": "zk-3.private.zkocean.hpy.dev",  
  "command": "leader",  
  "error": null  
}
```

```
root@zk-3:~# curl -s localhost:8080/commands/leader | jq .
```

```
{  
  "is_leader": true,  
  "leader_id": 3,  
  "leader_ip": "zk-3.private.zkocean.hpy.dev",  
  "command": "leader",  
  "error": null  
}
```

Success... well almost... Admin is over HTTP...




```
curl -v https://10.10.20.4:8080/commands/leader
* Trying 10.10.20.4:8080...
* TCP_NODELAY set
* Connected to 10.10.20.4 (10.10.20.4) port 8080 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
*   Capath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self signed certificate
* Closing connection 0
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html
```

Use HTTPS to connect to the API via IP Address.


```
curl -v https://10.10.20.4:8080/commands/leader
* Trying 10.10.20.4:8080...
* TCP_NODELAY set
* Connected to 10.10.20.4 (10.10.20.4) port 8080 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
*   Capath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self signed certificate
* Closing connection 0
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html
```

curl explodes because we never added the certs to our CApath or CAfile.



```
curl -v https://10.10.20.4:8080/commands/leader
* Trying 10.10.20.4:8080...
* TCP_NODELAY set
* Connected to 10.10.20.4 (10.10.20.4) port 8080 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
*   Capath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self signed certificate
* Closing connection 0
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html
```

curl politely tells us where those are.

```

curl --cacert /usr/local/share/ca-certificates/zk-all.crt -v https://10.10.20.4:8080/commands/leader
* Trying 10.10.20.4:8080...
* TCP_NODELAY set
* Connected to 10.10.20.4 (10.10.20.4) port 8080 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*  CAfile: /usr/local/share/ca-certificates/zk-all.crt
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: CN=zk-1.private.zkocean.hpy.dev
*  start date: Jun  9 06:08:08 2020 GMT
*  expire date: Sep  7 06:08:08 2020 GMT
*  subjectAltName: host "10.10.20.4" matched cert's IP address!
*  issuer: CN=zk-1.private.zkocean.hpy.dev
*  SSL certificate verify ok.
> GET /commands/leader HTTP/1.1
> Host: 10.10.20.4:8080
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Tue, 09 Jun 2020 06:23:27 GMT
< Strict-Transport-Security: max-age=86400; includeSubDomains
< Content-Type: application/json
< Content-Length: 134
< Server: Jetty(9.4.24.v20191120)
<
{
  "is_leader" : false,
  "leader_id" : 3,
  "leader_ip" : "zk-3.private.zkocean.hpy.dev",
  "command" : "leader",
  "error" : null
}
* Connection #0 to host 10.10.20.4 left intact

```

Adding --cacert flag gets our cert to work.


```
curl --cacert /usr/local/share/ca-certificates/zk-all.crt -v https://localhost:8080/commands/leader
* Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*  CAfile: /usr/local/share/ca-certificates/zk-all.crt
*  Capath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: CN=zk-1.private.zkocean.hpy.dev
*  start date: Jun  9 06:08:08 2020 GMT
*  expire date: Sep  7 06:08:08 2020 GMT
*  subjectAltName does not match localhost
* SSL: no alternative certificate subject name matches target host name 'localhost'
* Closing connection 0
* TLSv1.2 (OUT), TLS alert, close notify (256):
curl: (60) SSL: no alternative certificate subject name matches target host name 'localhost'
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

Using `localhost` doesn't work.

```
curl --cacert /usr/local/share/ca-certificates/zk-all.crt -v https://zk-1.private.zkocean.hpy.dev:8080/commands/leader
* Trying 10.10.20.4:8080...
* TCP_NODELAY set
* Connected to zk-1.private.zkocean.hpy.dev (10.10.20.4) port 8080 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
* CAfile: /usr/local/share/ca-certificates/zk-all.crt
  CPath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
* subject: CN=zk-1.private.zkocean.hpy.dev
* start date: Jun  9 06:08:08 2020 GMT
* expire date: Sep  7 06:08:08 2020 GMT
* subjectAltName does not match zk-1.private.zkocean.hpy.dev
* SSL: no alternative certificate subject name matches target host name 'zk-1.private.zkocean.hpy.dev'
* Closing connection 0
* TLSv1.2 (OUT), TLS alert, close notify (256):
curl: (60) SSL: no alternative certificate subject name matches target host name 'zk-1.private.zkocean.hpy.dev'
More details here: https://curl.haxx.se/docs/sslcerts.html
```

Using DNS doesn't work either, despite CN being set.

A terminal window with a dark background. A red rectangular box highlights the line containing the SAN field in the keytool command. To the left of the terminal window, there is a small red square containing a white eye icon, and a white rectangular box.

```
keytool -genkeypair -alias zk-1.private.zkocean.hpy.dev \  
-keyalg RSA -keysize 2048 -dname "cn=zk-1.private.zkocean.hpy.dev" \  
-keypass password \  
-keystore /opt/zookeeper/conf/keystore.jks \  
-ext san=ip:10.10.20.4,dns:zk-1.private.zkocean.hpy.dev,dns:localhost \  
-storepass password
```

Add more addresses to our SAN field.

```
export CLIENT_JVMFLAGS="-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty
-Dzookeeper.client.secure=true
-Dzookeeper.ssl.keyStore.location=/opt/zookeeper/conf/keystore.jks
-Dzookeeper.ssl.keyStore.password=password
-Dzookeeper.ssl.trustStore.location=/opt/zookeeper/conf/truststore.jks
-Dzookeeper.ssl.trustStore.password=password"

/opt/zookeeper/bin/zkCli.sh -server 127.0.0.1:2281

/usr/bin/java
Connecting to 127.0.0.1:2281
<redacted>
2020-06-09 06:45:58,987 [myid:] - INFO [main:X509Util@77] - Setting -D
jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation
2020-06-09 06:45:59,408 [myid:] - INFO [main:ClientCnxnSocket@239] - jute.maxbuffer value is 1048575
Bytes
2020-06-09 06:45:59,418 [myid:] - INFO [main:ClientCnxn@1703] - zookeeper.request.timeout value is 0.
feature enabled=false
Welcome to ZooKeeper!
2020-06-09 06:45:59,440 [myid:127.0.0.1:2281] - INFO [main-
SendThread(127.0.0.1:2281):ClientCnxn$SendThread@1154] - Opening socket connection to server
localhost/127.0.0.1:2281.
2020-06-09 06:45:59,441 [myid:127.0.0.1:2281] - INFO [main-
SendThread(127.0.0.1:2281):ClientCnxn$SendThread@1156] - SASL config status: Will not attempt to
authenticate using SASL (unknown error)
Jline support is enabled
[zk: 127.0.0.1:2281(CONNECTING) 0] 2020-06-09 06:46:00,007 [myid:127.0.0.1:2281] - INFO
[nioEventLoopGroup-2-1:ClientCnxnSocketNetty$ZKClientPipelineFactory@454] - SSL handler added for
channel: [id: 0x4ece59bc]
2020-06-09 06:46:00,036 [myid:127.0.0.1:2281] - INFO [nioEventLoopGroup-2-1:ClientCnxn$SendThread@986]
- Socket connection established, initiating session, client: /127.0.0.1:36126, server:
localhost/127.0.0.1:2281
2020-06-09 06:46:00,042 [myid:127.0.0.1:2281] - INFO [nioEventLoopGroup-2-
1:ClientCnxnSocketNetty$@184] - channel is connected: [id: 0x4ece59bc, L:/127.0.0.1:36126 -
R:localhost/127.0.0.1:2281]
2020-06-09 06:46:00,587 [myid:127.0.0.1:2281] - INFO [nioEventLoopGroup-2-
1:ClientCnxn$SendThread@1420] - Session establishment complete on server localhost/127.0.0.1:2281,
session id = 0x10000a76d880001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

[zk: 127.0.0.1:2281(CONNECTED) 0]
```

Now zkCli.sh even works.

But how do we make
Client certs that
other applications
can use?


```
X509v3 extensions:
```

```
  X509v3 Subject Alternative Name:
```

```
    IP Address:10.10.20.4, DNS:zk-1.private.zkocean.hpy.dev, DNS:localhost
```

```
  X509v3 Subject Key Identifier:
```

```
    53:DF:DD:F6:79:EA:EB:52:68:15:DC:8E:1D:A5:D9:C7:CF:22:F4:B0
```

Well this is awkward, there is no Root Certificate Authority.

```
mkcert -client zk-1.private.zkocean.hpy.dev zk-1.zkocean.hpy.dev localhost 10.10.20.3 127.0.0.1
mkcert -client zk-2.private.zkocean.hpy.dev zk-2.zkocean.hpy.dev localhost 10.10.20.4 127.0.0.1
mkcert -client zk-3.private.zkocean.hpy.dev zk-3.zkocean.hpy.dev localhost 10.10.20.2 127.0.0.1
```

```
openssl pkcs12 -export -in zk-1.private.zkocean.hpy.dev+4-client.pem -inkey zk-1.private.zkocean.hpy.dev+4-client-key.pem -name zk-1.private.zkocean.hpy.dev -out zk-1-keystore.p12
openssl pkcs12 -export -in zk-2.private.zkocean.hpy.dev+4-client.pem -inkey zk-2.private.zkocean.hpy.dev+4-client-key.pem -name zk-2.private.zkocean.hpy.dev -out zk-2-keystore.p12
openssl pkcs12 -export -in zk-3.private.zkocean.hpy.dev+4-client.pem -inkey zk-3.private.zkocean.hpy.dev+4-client-key.pem -name zk-3.private.zkocean.hpy.dev -out zk-3-keystore.p12
```

```
keytool -importkeystore -destkeystore zk-1-keystore.jks -srckeystore zk-1-keystore.p12 -srcstoretype pkcs12 -deststoretype pkcs12 -srcstorepass password -deststorepass password
keytool -importkeystore -destkeystore zk-2-keystore.jks -srckeystore zk-2-keystore.p12 -srcstoretype pkcs12 -deststoretype pkcs12 -srcstorepass password -deststorepass password
keytool -importkeystore -destkeystore zk-3-keystore.jks -srckeystore zk-3-keystore.p12 -srcstoretype pkcs12 -deststoretype pkcs12 -srcstorepass password -deststorepass password
```

```
keytool -keystore truststore.jks -storepass password -trustcacerts -importcert -alias bundle -file ~/.local/share/mkcert/rootCA.pem -noprompt
```

Create keystores with mkcert!

```
openssl s_client -connect zk-1.zkocean.hpy.dev:2281
```

```
ERROR [nioEventLoopGroup-7-1:NettyServerCnxnFactory$CertificateVerifier@434] - Unsuccessful handshake  
with session 0x0  
WARN [nioEventLoopGroup-7-1:NettyServerCnxnFactory$CnxnChannelHandler@273] - Exception caught  
io.netty.handler.codec.DecoderException: javax.net.ssl.SSLHandshakeException: null cert chain
```

ZK requests Client certificates, and we never provide one. Let's make one.

```
mkcert foobar
Using the local CA at "/home/ascherger/.local/share/mkcert" ✨

Created a new certificate valid for the following names 📋
- "foobar"

The certificate is at "./foobar.pem" and the key at "./foobar-key.pem" ✓
```

```
openssl s_client -CAfile ~/.local/share/mkcert/rootCA.pem -cert ./foobar.pem -key ./foobar-key.pem -
connect zk-1.zkocean.hpy.dev:2281

CONNECTED(00000003)
depth=1 0 = mkcert development CA, OU = ascherger@incontrol (Alan Scherger), CN = mkcert
ascherger@incontrol (Alan Scherger)
verify return:1
depth=0 0 = mkcert development certificate, OU = ascherger@incontrol (Alan Scherger)
verify return:1
140497829164032:error:14094416:SSL routines:ssl3_read_bytes:ssl3 alert certificate
unknown:ssl/record/rec_layer_s3.c:1544:SSL alert number 46
---
...
```

```
Caused by: sun.security.validator.ValidatorException: Extended key usage does not permit use for TLS
client authentication
    at sun.security.validator.EndEntityChecker.checkTLSClient(EndEntityChecker.java:245)
    at sun.security.validator.EndEntityChecker.check(EndEntityChecker.java:146)
```

ZK rejects our certificate because it wasn't created with the correct TLS Web Client Key Usage extension.

```
mkcert -client foobar
Using the local CA at "/home/ascherger/.local/share/mkcert" ✨

Created a new certificate valid for the following names 📋
- "foobar"

The certificate is at "./foobar-client.pem" and the key at "./foobar-client-key.pem" ✓
```

```
openssl s_client -CAfile ~/.local/share/mkcert/rootCA.pem -cert ./foobar-client.pem -key ./foobar-client-key.pem -connect zk-1.zkocan.hpy.dev:2281
```

Adding -client creates the proper certificate.

But wait... Why did it
let “foobar” join, it
never verified my
hostname?

Our non-quorum sockets are configured to never verify client Hostname **#GarbagelsGarbage**

```
public class ClientX509Util extends X509Util {

    private final String sslAuthProviderProperty = getConfigPrefix() + "authProvider";

    @Override
    protected String getConfigPrefix() {
        return "zookeeper.ssl.";
    }

    @Override
    protected boolean shouldVerifyClientHostname() {
        return false;
    }

    public String getSslAuthProviderProperty() {
        return sslAuthProviderProperty;
    }

}
```

<https://github.com/apache/zookeeper/blob/79a99ac97cea942967a2a08e7873a0cea9a2e046/zookeeper-server/src/main/java/org/apache/zookeeper/common/ClientX509Util.java#L31-L33>

Certificate Rotation

Expiring Certificate

Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Valid Not After: 01/01/2030



Web Server Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: foo.com
Subject Sig: 00:00:01
Valid Not After: 01/01/2021

Today's Date: 12/01/2021

LPT: Create and deploy a new certificate.

Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Valid Not After: 01/01/2030



Web Server Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: foo.com
Subject Sig: 00:00:02
Valid Not After: 01/01/2022

Today's Date: 12/01/2021

Basic Things to Monitor

- Root Certificate(s) Expiration Date
 - Intermediate Certificate(s) Expiration Date
 - End-entity Certificate(s) Expiration Date
 - Issuer and Subject Identifiers
-

Ut-Oh - now what?

Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Valid Not After: 01/01/2030



Web Server Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: foo.com
Subject Sig: 00:00:02
Valid Not After: 01/01/2022

Today's Date: 12/01/**2029**

1. Create New Root Certificate

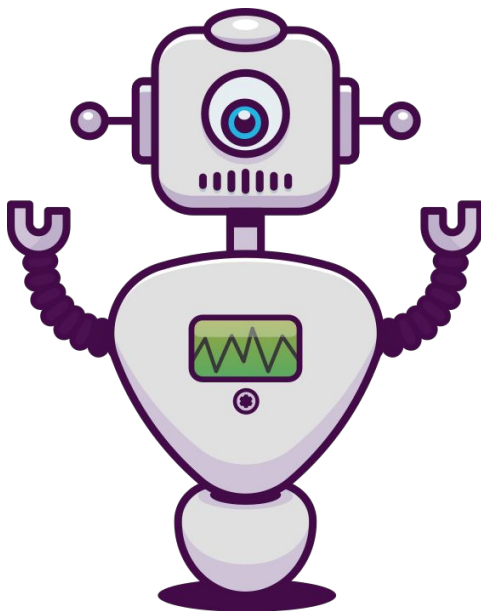
Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Valid Not After: 01/01/2030

Root Certificate 2
Issuer Name: ROOT2
Issuer Sig: 78:90:12
Valid Not After: 01/01/2040

1. Deploy ROOT2 to trust stores - everywhere.

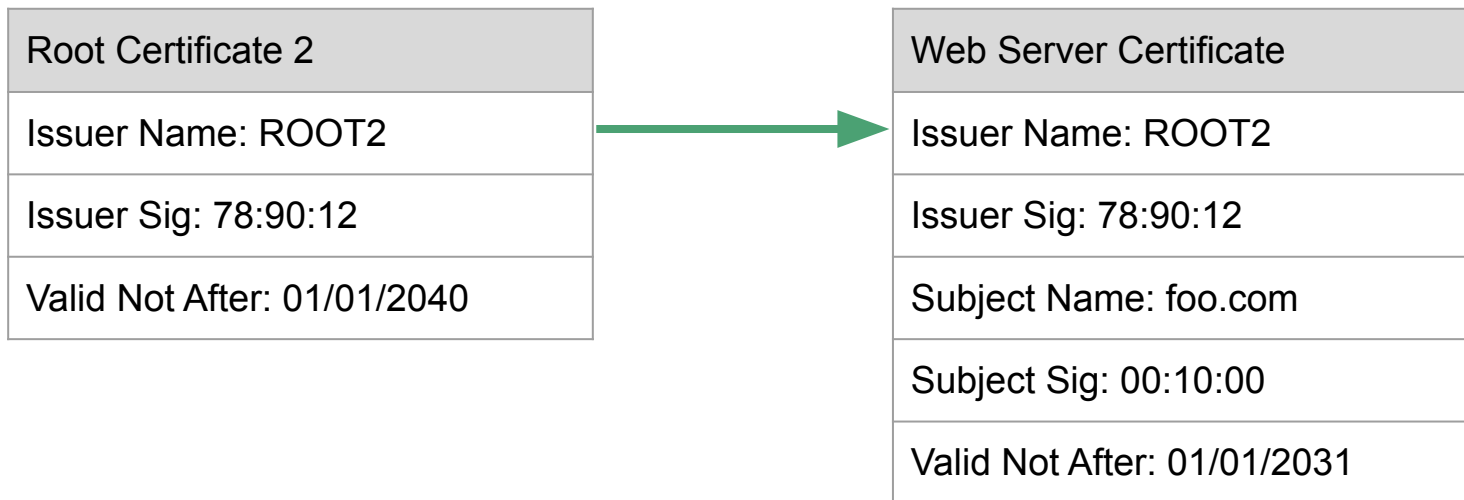
Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Valid Not After: 01/01/2030

Root Certificate 2
Issuer Name: ROOT2
Issuer Sig: 78:90:12
Valid Not After: 01/01/2040



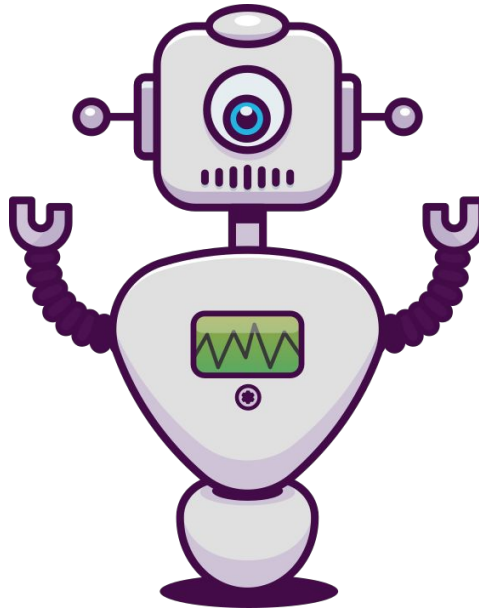
I will make light work of this!

LPT: Create and deploy a new certificate.



1. Delete ROOT from trust stores - everywhere.

Root Certificate 2
Issuer Name: ROOT2
Issuer Sig: 78:90:12
Valid Not After: 01/01/2040




I will make light work
of this!

But what about
Certificate
cross-signing?

TL;DR - Not the droids
you are looking for.

Remember, you can only have **1 Issuer Signature**.

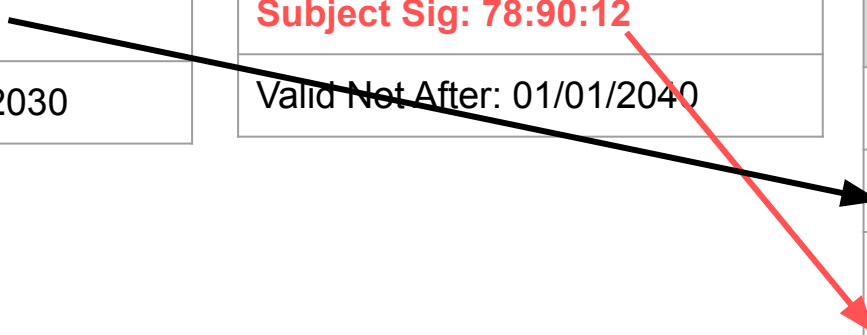
Root Certificate	Root Certificate 2
Issuer Name: ROOT	Issuer Name: ROOT2
 Issuer Sig: 12:34:56	Issuer Sig: 78:90:12
Subject Name: ROOT	Subject Name: ROOT2
Subject Sig: 12:34:56	Subject Sig: 78:90:12
Valid Not After: 01/01/2030	Valid Not After: 01/01/2040

Cross-sign Root 2 by 1:


Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: ROOT
Subject Sig: 12:34:56
Valid Not After: 01/01/2030

Root Certificate 2
Issuer Name: ROOT2
Issuer Sig: 78:90:12
Subject Name: ROOT2
Subject Sig: 78:90:12
Valid Not After: 01/01/2040


Root Certificate 2 - <u>By</u> 1
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: ROOT2
Subject Sig: 78:90:12
Valid Not After: 01/01/2040



Both Certs have **same Subject** Signature/Name

Root Certificate 2	Root Certificate 2 - Signed By 1
Issuer Name: ROOT2	Issuer Name: ROOT
Issuer Sig: 78:90:12	Issuer Sig: 12:34:56
Subject Name: ROOT2	Subject Name: ROOT2
 <u>Subject Sig: 78:90:12</u>	<u>Subject Sig: 78:90:12</u>
Valid Not After: 01/01/2040	Valid Not After: 01/01/2040

Both Certs have **different Issuer** Signature/Name

Root Certificate 2	Root Certificate 2 - Signed By 1
Issuer Name: ROOT2	Issuer Name: ROOT
 <u>Issuer Sig: 78:90:12</u>	<u>Issuer Sig: 12:34:56</u>
Subject Name: ROOT2	Subject Name: ROOT2
Subject Sig: 78:90:12	Subject Sig: 78:90:12
Valid Not After: 01/01/2040	Valid Not After: 01/01/2040

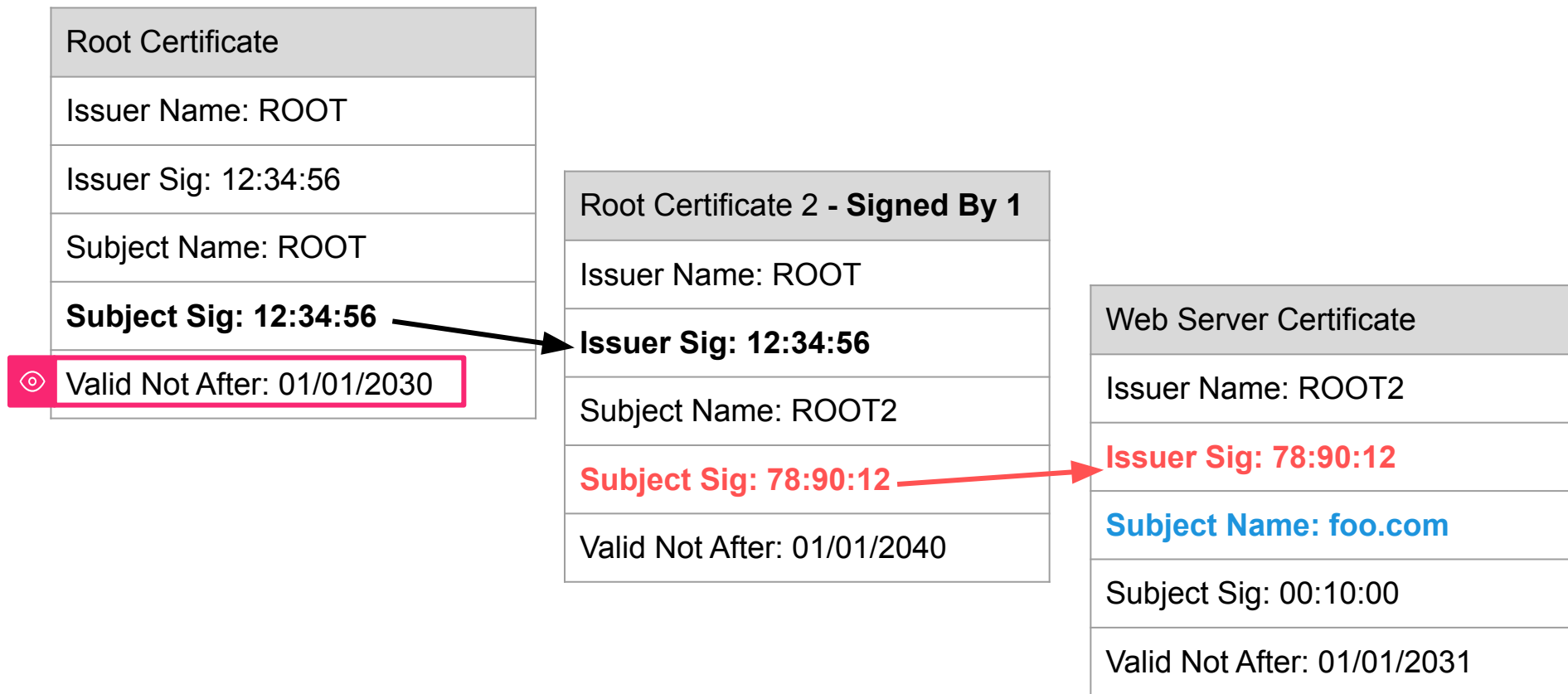
Create a Cert:

Root Certificate
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: ROOT
Subject Sig: 12:34:56
Valid Not After: 01/01/2030

Root Certificate 2 - Signed By 1
Issuer Name: ROOT
Issuer Sig: 12:34:56
Subject Name: ROOT2
Subject Sig: 78:90:12
Valid Not After: 01/01/2040

Web Server Certificate
Issuer Name: ROOT2
Issuer Sig: 78:90:12
Subject Name: foo.com
Subject Sig: 00:10:00
Valid Not After: 01/01/2031

Cross-signing does not fix the expiration of Root.



Certificate Cross-Signing

The myth, the legend.

- Must deploy Root2 somehow:
 - Bundle in new server deployment
 - Deploy to trust stores
- At some point you must rotate off the “by-1” cert
 - Expiration didn’t change
 - More redeploy
- What did any of this buy you?

Things we covered today.

- mTLS combines multiple technologies to enable us to allow create secure connections between our software.
- Certificate expiration dates should be monitored with automation
- Inspecting and validating certificates can all be done with OpenSSL CLI utilities
- Sometimes OpenSSL is LibreSSL
- Shorter Certificate Expiration is important, but also security theater.
- Castle & moat security is dead.
- Writing software to use mTLS takes effort
 - Hard coding configuration is bad
 - Different aspects of your software might need to use different certificates with different configuration.
 - Clients must be able to validate Servers
 - Servers must be able to validate Clients
 - Rich error handling goes a long way towards making things debuggable.
- Certificate rotation requires automation.

Things not covered.

- Revocation Lists
 - Onboarding existing clusters to mTLS
 - Running PKI infrastructure
 - HashiCorp Vault
 - AWS Certificate Manager Private Certificate Authority
-



bit.ly/learn-mtls

A workshop that will be much more in-depth than these slides.

Thanks!

Alan Scherger @flyinprogrammer