Step v0.8.6: Bring development closer to production with valid HTTPS certificates

smallstep.com/blog/step-v0-8-6-valid-https-certificates-for-dev-pre-prod

25 February 2019

Smallstep Certificate Manager | Your Hosted Private CA

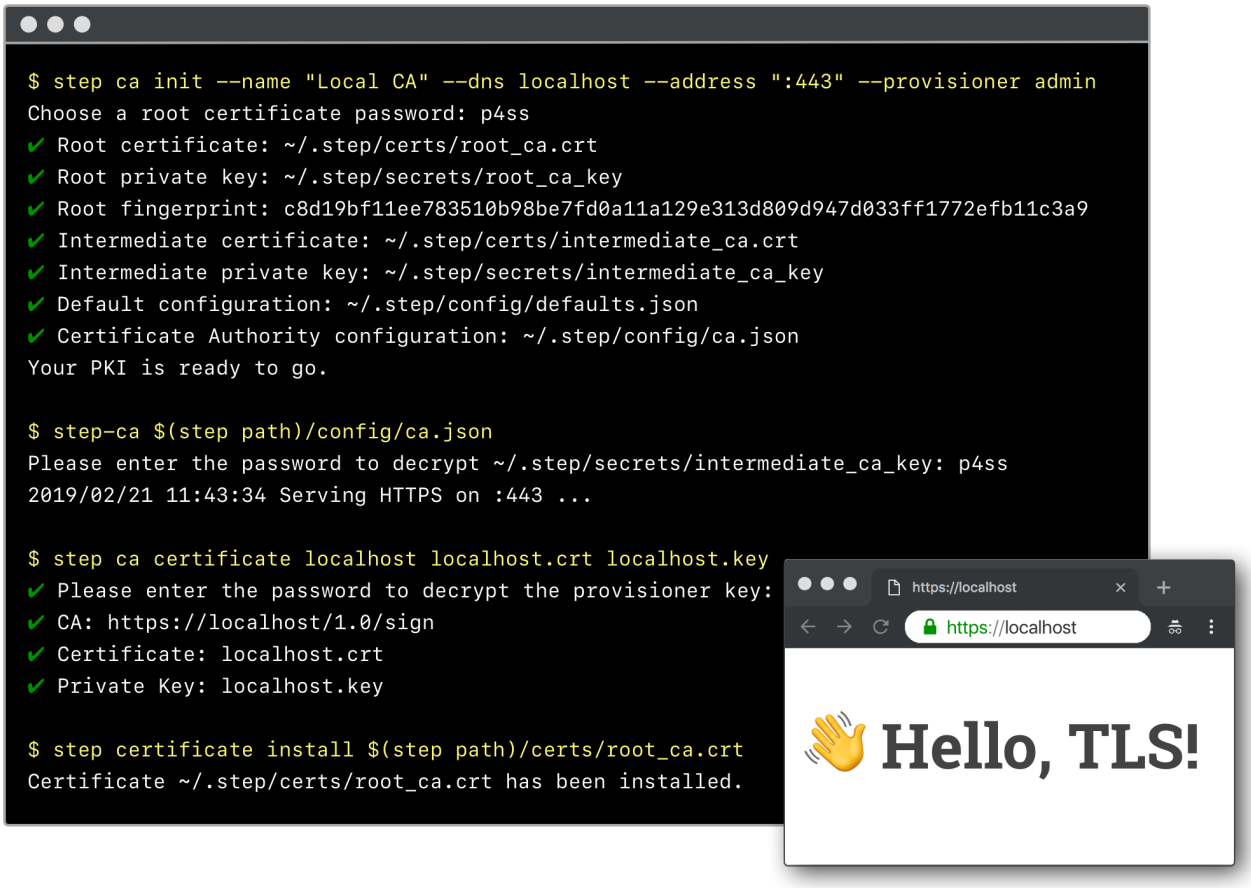
Learn More >

Sebastian Tiedtke

2019-02-25

follow smallstep on Twitter

Almost 80% of web page loads now use TLS. But almost no one uses TLS in development and pre-production. Why? Because it's hard. That sucks. When dev and staging don't match prod, bad things happen. Today's `step` release, version `0.8.6`, makes using TLS in dev & pre-prod environments a whole lot easier.



`step` is a zero trust command-line swiss army knife. Today's release focuses on improvements to `step certificates`, a sub-project that adds certificate management tools to `step`.

step cli

step certificates

> **Zero trust** is the idea that perimeter-based security using VPNs & VPCs is broken and should be replaced with identity-based security using cryptography (e.g., TLS). This is unequivocally a good idea. Partly because it's "better security" but mostly because it *works everywhere*: you can connect on-prem to cloud, AWS to GCP, serverless to containers, IoT to edge… *x* to *y* for any *x* and *y*. It's universal: based on math, not trust in some vendor's platform-specific solution.

Installing `step`

To follow along at home (which I highly encourage) you'll need to install `step`. We're happy to announce that `step` is now in homebrew core. To install on macOS simply run:

```
brew install step
```

Linux users can install `step 0.8.6` using `dpkg` (or build from source).

```
curl -Os https://github.com/smallstep/cli/releases/download/v0.8.6/step-cli_0.8.6_amd64.deb && sudo dpkg -i step-cli_0.8.6_amd64.deb
```

If you use docker you can launch our base image locally:

```
docker run -it --rm smallstep/step-cli:0.8.6
```

Or in a kubernetes cluster:

```
kubectl run step -it --rm --image smallstep/step-cli:0.8.6 --restart Never
```

Valid certificates for HTTPS everywhere

Historically, obtaining TLS (HTTPS) certificates has been annoying and expensive. Let's Encrypt solved that problem by allowing anyone to obtain a free certificate automatically using the ACME protocol. It's a fantastic service. You should be using it (or something similar) for your public websites and APIs.

But certificates from Let's Encrypt and other *"Web PKI"* certificate authorities (CAs) are no good for service-to-service communication or anything else you want to keep private and internal. They're rate limited, can't issue certificates for internal names (e.g., `localhost` or `foo.default.svc.cluster.local`), and the ecosystem suffers from some serious trust issues.

We built `step certificates` to make running your own CA and managing certificates for internal services as easy as Let's Encrypt.

Simply run the `init` workflow:

```
step ca init --name "Local CA" --provisioner admin \
             --dns localhost --address ":443"
```

You'll be prompted for a password to encrypt key material.

Next, start your CA using the generated configuration (you'll be prompted for your sudo password so we can listen on port 443, then your `step` admin password):

```
sudo step-ca $(step path)/config/ca.json
```

You can get a certificate from your CA for any name, including `localhost`. In another terminal run (you'll be prompted for your `step` admin password):

```
step ca certificate localhost localhost.crt localhost.key
```

> If you'd rather not run an online CA try the lower level`step certificate`command group.

Internal services can be easily configured to trust your CA. That means they can securely communicate across the public internet using mutual TLS without a VPN. Huzzah!

We built `step certificates` to manage certificates for production workloads. But we've received several inquiries about a slightly different use case: simulating *Web PKI* so TLS works as expected in development and pre-production environments. In other words, people want dev & pre-prod certificates that work by default in browsers just like their production certificates from Let's Encrypt.

Using TLS in development & pre-prod simplifies code by eliminating non-TLS code paths and helps catch bugs like mixed-content warnings sooner. More generally, it brings pre-prod environments closer to production and gives engineers the opportunity to gain operational experience with TLS outside of emergency situations.

That all seemed pretty reasonable to us, and we'd already built most of what was required. There were just a few small gaps to fill to make it easy. Today's release adds a powerful new command to fill one of them:

```
$ step certificate install
```

Inspired by Filippo Valsorda's mkcert utility, `step certificate install` makes it easy to trust a new CA locally. After installation, certificates issued by that CA will work in browsers and elsewhere, just like a certificate from Let's Encrypt.

You'll get that elusive lock icon with your own locally issued certs:

```
$ step certificate install $(step path)/certs/root_ca.crt
$ python <<EOF
import BaseHTTPServer, ssl
class H(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self): self.send_response(200); self.end_headers();
self.wfile.write(b'<h1>&#128075; Hello, TLS!</h1>')
httpd = BaseHTTPServer.HTTPServer(('', 8443), H)
httpd.socket = ssl.wrap_socket (httpd.socket, server_side=True,
keyfile="localhost.key", certfile="localhost.crt")
httpd.serve_forever()
EOF
```

We're very grateful for Filippo's awesome work with `mkcert`. His code for working with root trust stores on various platforms made this feature much easier to build. We factored that code out into a separate golang library to make it *even easier* for others.

Like `mkcert`, `step` strives for simplicity and does the right thing by default. And, like `mkcert`, we're working on adding ACME support to our online CA to make local development using TLS a simple matter of switching your ACME endpoint URL.

Unlike `mkcert`, `step` aims to be comprehensive and useful from development through to production. If you just want locally issued certificates for development, you should use `mkcert`. If you also want CA infrastructure for your microservices in pre-production and production, you want more control over the keys and certificates being generated, or you want an easier way to install your existing enterprise certificates on endpoints, you should check out `step certificates`.

And if `step` doesn't do something you want it to do, please let us know!

What else is new?

Support for multiple SANs

Web servers often respond to multiple names (e.g., `amazon.com` and `www.amazon.com`). To support this you can bind multiple subject alternative names (SANs) in a certificate. The new `--san` flag lets you do so using `step`:

```
step ca certificate example.com example.crt example.key \
  --san www.example.com \
  --san "*.admin.example.com"
```

Extract public keys from certificates

The public key bound in a certificate can be used to verify a signature or encrypt data for the certificate's owner. To do so the key must be extracted, which can be a bit tricky. So we added the `step certificate key` command:

> ⚠️ Remember to always verify certificates before trusting them (e.g., using `step certificate verify`) and respect key use restrictions!

```
$ export CIPHERTEXT=$(echo "hello, localhost" | step crypto jwe encrypt --key
<(step certificate key localhost.crt))
$ echo $CIPHERTEXT
{"protected":"eyJhbGciOiJFQ0RILUVTIiwiZW5jIjoiQTI1NkdDTSIsImVwayI6eyJrdHkiOiJFQyIsI
d","ciphertext":"5C2IHKd2MF_cK9p0hvUyFAM","tag":"ZnVzNzuBzL3pnUE9hCWOkw"}
$ echo $CIPHERTEXT | step crypto jwe decrypt --key localhost.key
hello, localhost
```

The `step certificate key` command also works with certificate signing requests (CSRs). As an added convenience we've also updated the `step crypto jwe encrypt` and `step crypto jwt verify` commands to accept X.509 certificates as arguments to the `--key` flag.

Inspect DER certificates

Certificates can be represented using different formats. By default `step` uses PEM. This release adds support for inspecting DER-encoded certificates:

```
$ step certificate format localhost.crt --out localhost.der
Your certificate has been saved in localhost.der.
$ step certificate inspect --format json localhost.der | jq -r
.subject.common_name[0]
localhost
```

Fingerprint remote certificates

Certificate fingerprints are really useful for debugging. This release adds the ability to fingerprint a remote server's certificate to verify that the correct certificate is in use:

```
$ step certificate fingerprint https://localhost:8443
e3e95bf6047496086417ce72b486b9d5b0a144bc065da7e932fc6b41187b7b50
$ step certificate fingerprint localhost.der
e3e95bf6047496086417ce72b486b9d5b0a144bc065da7e932fc6b41187b7b50
```

Certificate uninstall

If you've been following along with the examples here you might want to remove the certificate you installed earlier. There's a command for that, too:

```
step certificate uninstall $(step path)/certs/root_ca.crt
```

> When you run `step ca init` it generates artifacts that are stored in `~/.step`. To remove `step` from your system simply remove that directory. To use a different directory set the `STEPPATH` environment variable.

More coming soon

That's all for now, but there's lots more coming soon! We're working on tighter integrations with common environments like AWS, GCP, and kubernetes. We're also working on better visibility into certificate operations using certificate transparency.

step cli

step certificates

Stay tuned by following us on twitter and GitHub!

Subscribe to updates

Unsubscribe anytime, see Privacy Policy

Smallstep Certificate Manager | Your Free Hosted Private CA

Learn More >
Technical step Certificates