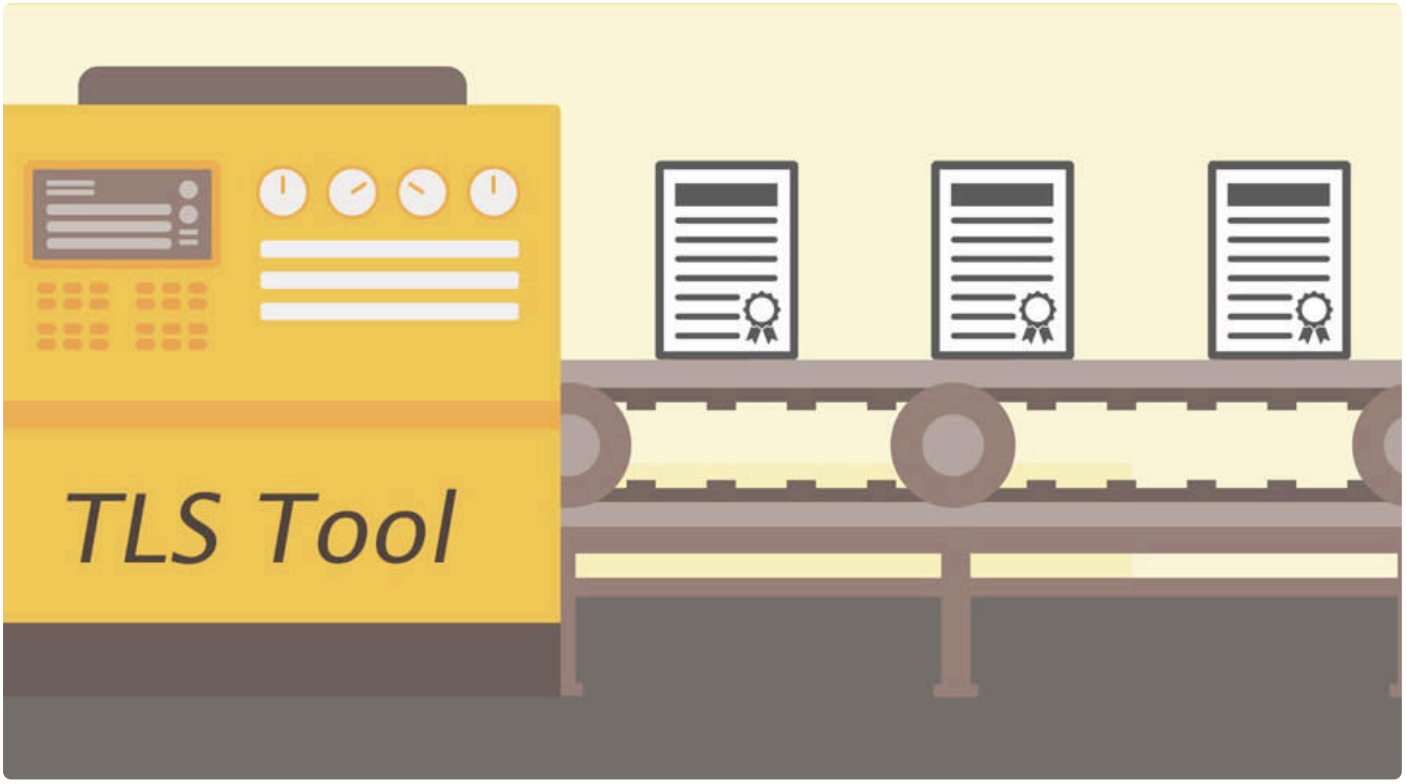
Generating production-ready certificates with the TLS tool

Cliff Staley · 2018-02-25



Generating TLS certificates can be a pain sometimes. If you do not have a PKI infrastructure already in place, you need to create a root CA, generate certificate signing requests for all your nodes, sign them with the root CA, and configure them correctly in *elasticsearch.yml*. Our new TLS tool comes to the rescue: Specify what certificates you need, execute the tool and you're ready to go. Since you have full control over the CA and the certificates, they can be safely used also for production clusters. And if you just need to create CSRs or validate existing certificates quickly: We have that too!

Watch the video in the online version of this article.

The Search Guard TLS tool can be used to

- Generate Root and Intermediate CAs
- Generate node-, client- and admin certificates
- Generate the Search Guard TLS configuration
- Generate CSRs which you can submit to your PKI
- Validate existing certificates and TLS settings

You can find the TLS tool on our Maven server:

Search Guard TLS tool

Download the *zip* or *tar.gz* archive, and unpack it in any directory. The tool is platform independent, so it runs on Linux, Mac, and Windows. After unpacking, you will see three directories:

- deps: The tool itself with all dependencies
- config: Sample configuration files
- tools: tlstool and tlsdiag executable scripts, for Linux, Mac and Windows

Creating the root CA and intermediate CA

First, we need to create a root CA. In this example, we will also create an intermediate CA (sometimes also called signing CA) which we will use to sign all other certificates. While an intermediate CA is not strictly required, most real-world PKIs use one or more intermediate certificates. The reason is quite simple: You can revoke intermediate certificates individually without having to change the root CA: In case of a data breach, you revoke the intermediate CA that was used to sign your node- and client certificates, thus rendering them invalid as well.

The TLS tool uses a YAML configuration file to generate the certificates. You can place the configuration file anywhere on your system, and then tell the tool where to find it via the *-c/–config* command line argument. If you do not want to start from scratch, you can find an example configuration and a template in the config folder.

Let's define our root and intermediate CA in a file called *config.yml* first:

```
ca:
   root:
      # The distinguished name of this CA. You must specify a distinguished name.
      dn: CN=root.ca.example.com,OU=CA,O=Example Com\, Inc.,DC=example,DC=com
      pkPassword: auto
      file: root-ca.pem
   intermediate:
      dn: CN=signing.ca.example.com,OU=CA,O=Example Com\, Inc.,DC=example,DC=com
      pkPassword: auto
      file: intermediate-ca.pem
```

In this example, we only specified the minimum configuration settings needed to create the CAs. You can add more settings, like the validity period of the certificates or the length of the private key. Have a look at our official documentation for a complete reference. The *dn* key defines the Distinguished Name to use for the certificate and is mandatory.

The *pkPassword* key specifies the password for the private keys. In our case, we let the TLS tool auto-generate a key for us, but we could have used our own password here as well. Unencrypted keys are also supported, but not recommended for security reasons. Finally we define the filenames for the generated artefacts.

First run

Let's run the TLS tool and see what artefacts are generated. We specify the location of our configuration file with the *-c* switch, and tell the tool to create the root and intermediate CA by adding the *-ca/_–*

create-ca *switch. By default, the files will be placed in a directory named* out*, relative to the execution directory. If you want to put them somewhere else, add the target location with the* -t/–target_ switch.

```
./sgtlstool.sh -c ./config.yml -ca -t /tmp/certificates
```

The tool has created the following files:

- root-ca.pem: The root certificate
- root-ca.key: The private key for the root certificate
- intermediate-ca.pem: The intermediate certificate
- intermediate-ca.key: The private key for the intermediate certificate
- root-ca.readme: The auto-generated passwords for the private keys

Generating node- and admin certificates

Now we're ready to generate our node- and admin certificates. Since we have chosen to use auto-generated passwords, we first need to add them to the *config.yml*. You can omit this step if you use your own passwords or unencrypted keys.

We first open the *root-ca.readme* file and copy the generated keys. The *config.yml* now looks like:

```
ca:
   root:
      # The distinguished name of this CA. You must specify a distinguished name.
      dn: CN=root.ca.example.com,OU=CA,O=Example Com\, Inc.,DC=example,DC=com
      pkPassword: wkLkF2zXLAWv
      file: root-ca.pem
   intermediate:
      dn: CN=signing.ca.example.com,OU=CA,O=Example Com\, Inc.,DC=example,DC=com
      pkPassword: hH2a1aY0Fy94
      file: intermediate-ca.pem
```

Setting the defaults

While you can customize each certificate individually, you most likely want to set some defaults that are applied to all certificates globally. These can be set in the *defaults* section of our *config.yml*. Again, please refer to official documentation for a complete reference. We use the following settings:

```
defaults:
      validityDays: 3650
      pkPassword: auto
      httpsEnabled: true
      verifyHostnames: true
      resolveHostnames: true
```

This will generate certificates that are valid for 3650 days with auto-generated passwords. Since we are targeting a production environment, we enable HTTPS, hostname verification and hostname lookups. This means Search Guard will compare the hostname in the certificate with the hostname of your

Elasticsearch node, and also performs a DNS lookup for the hostname. This provides an extra layer of security but requires of course that the hostnames match and can be resolved via your DNS.

Configuring node- and admin certificates

Now we can add all our Elasticsearch nodes to the configuration and specify the admin certificate(s) which can be used together with sgadmin:

```
nodes:
  - name: node1
    dn: CN=node1.example.com,OU=Ops,O=Example Com\, Inc.,DC=example,DC=com
    dns: node1.example.com
  - name: node2
    dn: CN=node2.example.com,OU=Ops,O=Example Com\, Inc.,DC=example,DC=com
    dns:
      - node2.example.com
      - es2.example.com
  - name: node3
    dn: CN=node3.example.com,OU=Ops,O=Example Com\, Inc.,DC=example,DC=com
    dns: node3.example.com
```

This configuration defines three Elasticsearch nodes with their respective hostnames. The hostnames will be part of the Subject Alternative Name (SAN) field. If you plan to use hostname verification, this is a mandatory field. Next, we add the client- and admin certificates:

```
clients:
  - name: spock
    dn: CN=spock.example.com,OU=Ops,O=Example Com\, Inc.,DC=example,DC=com
  - name: kirk
    dn: CN=kirk.example.com,OU=Ops,O=Example Com\, Inc.,DC=example,DC=com
    admin: true
```

We have defined two client certificates, and promote the “kirk” certificate to an admin certificate by adding *admin: true*. You need to configure at least one admin certificate, otherwise you will not be able to use *sgadmin* to administer Search Guard. Now we’re ready to generate the certificates:

```
./sgtlstool.sh -c ./config.yml -crt -t /tmp/certificates
```

For each configured node our tool will output a certificate and private key for the Transport- and REST-layer, and an *elasticsearch.yml* configuration snippet. For each node:

- Copy *root-ca.pem* to the Elasticsearch config directory
- Copy *.pem, .key, http.pem and* _http.key_ to the config directory
- Add the contents of *elasticsearch*config_snippet.yml to elasticsearch.yml

In our example, the configuration snippet looks like:

```
searchguard.ssl.transport.pemcert_filepath: node1.pem
searchguard.ssl.transport.pemkey_filepath: node1.key
searchguard.ssl.transport.pemkey_password: iIFKZZ5C9wqF
searchguard.ssl.transport.pemtrustedcas_filepath: root-ca.pem
searchguard.ssl.transport.enforce_hostname_verification: true
searchguard.ssl.transport.resolve_hostname: true
searchguard.ssl.http.enabled: true
searchguard.ssl.http.pemcert_filepath: node1_http.pem
searchguard.ssl.http.pemkey_filepath: node1_http.key
searchguard.ssl.http.pemkey_password: II4Cqno3pt0l
searchguard.ssl.http.pemtrustedcas_filepath: root-ca.pem
searchguard.nodes_dn:
- CN=node1.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
- CN=node2.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
- CN=node3.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
searchguard.authcz.admin_dn:
- CN=kirk.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
```

Now you can start the node, and use the *kirk* certificate with sgadmin:

```
./sgadmin.sh -key ./kirk-key.pem -keypass 8Faq7qPtP1DH -cert ./kirk.pem -cacert ./root-ca.pem ...
```

The auto-generated keys of the client- and admin certificates are written to *client-certificates.readme* in the same output directory:

Generating everything in one go

In the example above we first generated the root and intermediate CA, and then all other certificates. You can also generate everything in one go by using the -ca and -crt switch at the same time:

```
./sgtlstool.sh -c ./config.yml -crt -ca -t /tmp/certificates
```

Adding certificates

You can add more certificates as needed at any time, given you have the root and intermediate CA files in the target directory. Simply add more entries in the nodes or clients section of your configuration, and execute *sgtlstool* again:

```
./sgtlstool.sh -c ./config.yml -crt -t /tmp/certificates
```

This will not overwrite already existing files but only add new ones. If you want to overwrite existing files, add the *-o/–overwrite*option. Note that if you use auto-generated key passwords, these will change as well.

Making node certificates more flexible

In the example above you may have noticed that all node certificates are listed in *elasticsearch.yml* explicitly:

```
searchguard.nodes_dn:
- CN=node1.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
- CN=node2.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
- CN=node3.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
```

This has a major drawback: Whenever you add a new node to your cluster, you need to change this setting on all other nodes as well. Not something you want to do in production and with large clusters. There are two approaches to address this issue: Wildcards and OID values.

Using wildcards to list the node certificates

The *searchguard.nodes*dn_ configuration also supports wildcards in the DN of the certificate. Instead of listing the certificates separately, you can use:

```
searchguard.nodes_dn:
- CN=*.example.com,OU=Ops,O=Example Com, Inc.,DC=example,DC=com
```

In order to add such an entry in all generated configuration snippets, add the value you want to use in the *defaults* section of the *config.yml*:

```
defaults:
  ...
  nodesDn:
    -"CN=*.example.com,OU=<span class="s1">Ops,O=Example <span class="s1">Com\,
Inc.,DC=example,DC=<span class="s1">com"
  ...
```

Using an OID in the node certificates

The above approach works fine, but there's an even better solution. Certificates can optionally have so-called OIDs in their SAN section. OID stands for object identifier and is an X509 certificate extensions. Search Guard can use OIDs to identify trusted nodes in your cluster. In this case, you do can omit the *searchguard.nodes*dn_ section completely.

In order to add and use OID values in your certificates, set the value you want to use in the defaults section like:

```
defaults:
 ...
 nodeOid: "1.2.3.4.5.5"
 ...
```

You can basically use any value here, but *1.2.3.4.5.5* is the default. If this is set, the OID is added to all generated node certificates automatically. Instead of listing the node certificate DNs, the generated snippet will now contain:

```
searchguard.cert.oid: 1.2.3.4.5.5
```

If a new node wants to join the cluster, its OID value is checked after the certificate has been validated. Only then it is allowed to join. Since all node certificates will have this OID set, there is nothing to change in *elasticsearch.yml* when nodes are added or removed. This provides for the highest flexibility on production clusters.

Summary

You can use the Search Guard TLS tool to generate root CAs, intermediate CAs, node-, client- and admin certificates. Since the CAs are created locally, the generated certificates are safe also for production use. You can add more certificates as needed at any time, and choose which method you like to use for identifying nodes. In addition the tool can also generate CSRs you can submit to your own PKI, and it can validate existing certificates.

Where to go next

- Download the TLS tool from Maven
- Read the official docs for the Search Guard TLS tool
- Learn how to create CSRs with the tool
- Learn how to use the tool to validate existing certificates

Update 27.10.2025: Corrected download link

Image: shutterstock / Visual Generation / T-Kot

This article was published on the Search Guard blog: https://search-guard.com/blog/generating-certificates-tls-tool/