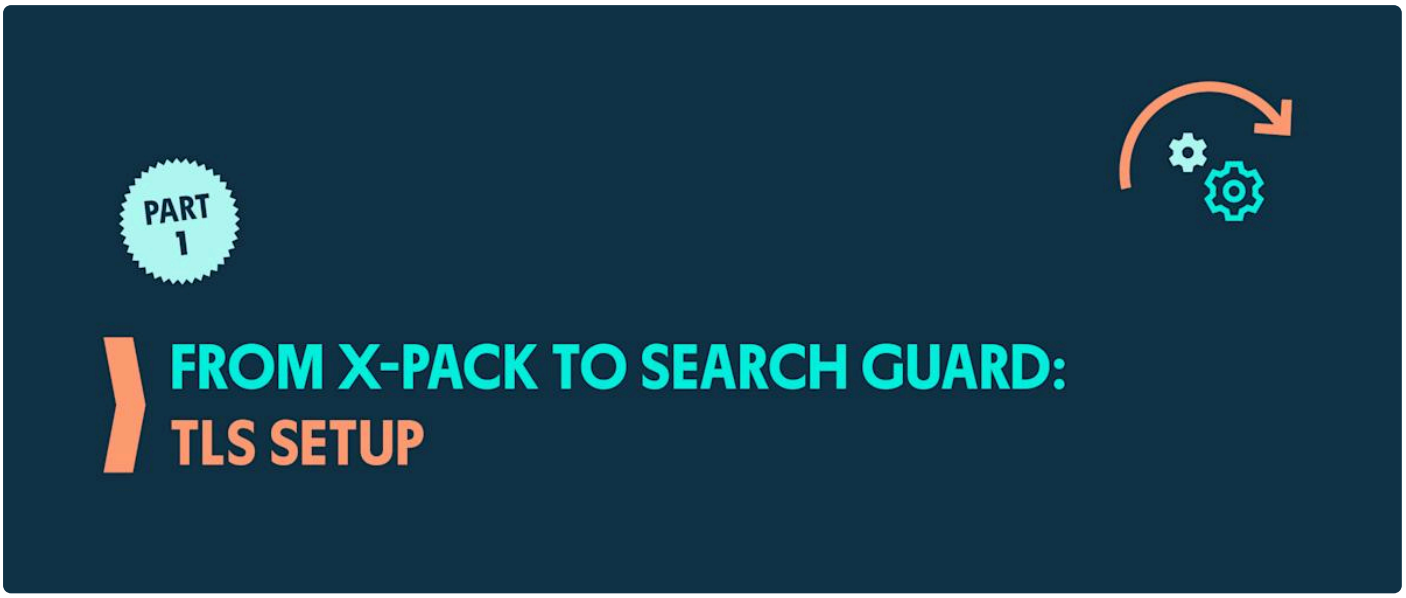
From X-Pack to Search Guard: Part 1 - TLS Setup

Jochen Kressin · 2025-06-12



Migrating your Elasticsearch cluster from X-Pack Security to Search Guard can be straightforward, especially for TLS encryption. This guide focuses on transitioning the TLS configuration (for both inter-node communication and the REST API) while **reusing your existing Certificate Authority (CA)**. By using the same internal CA that X-Pack was using, you ensure that all clients (like Kibana and applications) which already trust this CA continue to work seamlessly.

We will walk through identifying/exporting the X-Pack CA, using the Search Guard TLS Tool to generate new certificates signed by this CA, and updating the Elasticsearch and Kibana configurations step-by-step. This guide assumes you have Search Guard installed for Elasticsearch 8.x and are familiar with basic TLS concepts.

Differences in TLS Between X-Pack and Search Guard

Before diving into the migration steps, it's important to understand how TLS usage differs between X-Pack (Elasticsearch's built-in security) and Search Guard:

Node Certificate Validation:

Search Guard requires you to explicitly list the distinguished names (DNs) of the node certificates that are allowed to form the cluster. In Search Guard, you configure `searchguard.nodes_dn` in `elasticsearch.yml` to enumerate which cert DNs are trusted for inter-node communication. This prevents unknown certificates (even if signed by the same CA) from joining the cluster without permission. Wildcards are also supported to make the configuration easier.

Admin Certificate vs. Internal User:

In X-Pack, administrative changes (like security setting updates) are typically done by an internal superuser (e.g., the built-in `elastic` user) using username/password. In Search Guard, **admin tasks are performed using a TLS client certificate** with elevated privileges. Search Guard requires an "admin"

certificate (a special client certificate) to run its administrative tools or use the REST management API. This means part of the TLS setup includes generating an admin certificate and listing its DN in `searchguard.authcz.admin_dn` (instead of relying on the Elastic superuser).

Configuration Format:

The TLS settings in `elasticsearch.yml` use different keys. X-Pack uses settings under `xpack.security.*` (for example, `xpack.security.transport.ssl.enabled`, `xpack.security.transport.ssl.keystore.path`, etc.), whereas Search Guard uses `searchguard.ssl.*` settings for transport and HTTP (REST) TLS. We will see examples of these in the configuration step. Search Guard supports similar certificate formats (PEM, JKS, PKCS#12) as X-Pack, but it often prefers PEM files for configuration. If your X-Pack certificates are in a Java Keystore or PKCS#12, you can still use them with Search Guard by configuring keystore/truststore paths, but here we will use PEM for simplicity.

TLS on Transport vs. REST:

Both X-Pack and Search Guard require TLS on the **transport (inter-node)** layer for security, while TLS on the **REST API (HTTP)** layer is optional but recommended. In Search Guard, you enable HTTPS on the REST layer with `searchguard.ssl.http.enabled: true`, similar to X-Pack's optional HTTP TLS. Search Guard allows using the same certificate for both layers or separate certificates for transport and HTTP per node. We'll generate certificates for both layers for completeness.

With these differences in mind, let's proceed with the migration steps.

Step 1: Identify and Export the Existing X-Pack CA

First, determine what Certificate Authority was used in your X-Pack setup. In many cases, you might have created a CA when enabling X-Pack security (for example, using the `elasticsearch-certutil` tool). This CA is what signed your Elasticsearch node certificates under X-Pack, and we want to reuse it for Search Guard.

Locate the CA:

If you followed Elastic's documentation, you might have an **Elastic Stack CA** stored in a file. Common scenarios include:

- A PKCS#12 keystore file (often named `elastic-stack-ca.p12` or similar) containing the CA. By default, running `elasticsearch-certutil` the first time generates a CA and stores it in a P12 file.
- A PEM file for the CA certificate (e.g., `ca.crt`) and possibly a PEM for the CA key (`ca.key`) if you used the `--pem` option with `certutil` or otherwise extracted them. In X-Pack, you might not have needed the CA private key on disk if you only provided signed certs to Elasticsearch, but for Search Guard's certificate generation, you'll need the CA's private key available.

Export the CA from a keystore (if applicable):

If your CA is in a P12 or JKS keystore, you'll need to export the CA certificate and private key to PEM format. You can use OpenSSL for PKCS#12, or Keytool for JKS. For example, assuming `elastic-stack-ca.p12` is your CA keystore and you know its password, you can run:

- Export CA certificate to PEM

```
openssl pkcs12 -in elastic-stack-ca.p12 -cacerts -nokeys -out xpack-ca.pem
```

- Export CA private key to unencrypted PEM (PKCS#8 format)

```
openssl pkcs12 -in elastic-stack-ca.p12 -nocerts -nodes -out xpack-ca.key
```

The first command extracts the CA certificate. The second extracts the CA's private key (the `-nodes` flag writes it without encryption for now; you can omit `-nodes` to keep it encrypted, but then you'll need to supply a password in the next steps). The output files are `xpack-ca.pem` (CA cert) and `xpack-ca.key` (CA key).

Note: If the private key is not already in PKCS#8 format (Search Guard requires PKCS#8), you may need to convert it. For example:

```
openssl pkcs8 -in xpack-ca.key -topk8 -nocrypt -out xpack-ca-pkcs8.key
```

This command takes an RSA key and outputs a PKCS#8 PEM key without encryption. You could add `-passout` and omit `-nocrypt` to set a password. If you do set a password, remember it for the TLS tool config.

Verify the CA files:

Open the `xpack-ca.pem` in a text editor or use OpenSSL to ensure it's the correct CA (look at the subject/issuer). Also verify `xpack-ca.key` is a valid private key (for example, `openssl rsa -in xpack-ca.key -check` if it's RSA, or `openssl pkey` command). **Keep these files safe**, as the CA key is sensitive. We will use them with the Search Guard TLS tool next.

Step 2: Prepare the Search Guard TLS Tool Configuration

Search Guard provides an **Offline TLS Tool** that streamlines certificate generation. We will use this tool to create new nodes and admin certificates signed by your existing CA, instead of generating a new CA.

- **Download the TLS Tool:** If you haven't already, download the Search Guard TLS Tool (for the version compatible with your Search Guard). It comes as a zip or tar.gz – unpack it on a workstation where you can run it (it's a Java application, but packaged with a script). For example:

```
wget https://maven.search-guard.com/search-guard-tlstool/com/floragunn/search-guard-
tlstool/3.0.2/search-guard-tlstool-3.0.2.zip
unzip search-guard-tlstool-3.0.2.zip
cd search-guard-tlstool
```

The tool is invoked via `tools/sgtlstool.sh` (or `.bat` on Windows).

- **Create a TLS config file:** Next, create a YAML configuration file (e.g., `tlsconfig.yml`) that describes your CA and the certificates you need. We will include sections for the root CA, node certificates, and an admin certificate. Below is a template to illustrate the necessary sections:

```
ca:
  root:
    dn: CN=Your X-Pack CA, O=YourOrg, L=YourCity, C=YourCountry
    file: xpack-ca.pem
    pkPassword: none
defaults:
  validityDays: 730
  pkPassword: auto
  httpsEnabled: true
  reuseTransportCertificatesForHttp: false
nodes:
  - name: node1
    dn: CN=node1.example.com,OU=IT,O=YourOrg,L=YourCity,C=YourCountry
    dns:
      - node1.example.com
      - node1
    ip:
      - 10.0.0.11
  - name: node2
    dn: CN=node2.example.com,OU=IT,O=YourOrg,L=YourCity,C=YourCountry
    dns:
      - node2.example.com
      - node2
    ip:
      - 10.0.0.12

clients:
  - name: admin
    dn: CN=admin,OU=IT,O=YourOrg,L=YourCity,C=YourCountry
    admin: true
```

Let's break down the important parts of this config:

CA Section:

We define the root CA under `ca.root`. The `dn` should match the Subject DN of your existing X-Pack CA (this is mainly for documentation; the tool will not regenerate it since we won't use the `-ca` flag). We specify `file: xpack-ca.pem` so the tool knows the filename of the CA certificate. **Important:** Before running the tool, place your `xpack-ca.pem` and `xpack-ca.key` files into the tool's output directory (by default, a folder named `out/` in the tool directory). The tool will look for the CA cert and key when we run it in certificate generation mode. Also, set `pkPassword` to "none" if your CA key is unencrypted, or provide the password if it is encrypted (this tells the tool how to read the CA key).

Defaults:

We set some defaults for the certificates:

- `validityDays` defines how long the new certificates should be valid (here 730 days = 2 years, adjust as needed).
- `pkPassword` is set to `auto` which means each generated certificate's private key will be encrypted with a random password (the tool will output these passwords in a `*.readme` file). If you prefer to have no password on the new keys (for simpler config, at the cost of storing unencrypted keys), use `none`

instead. Alternatively, you can specify a fixed password string if you want all keys to share a specific password.

- `httpsEnabled: true` ensures the tool will generate certificates for the HTTP/REST layer (HTTPS).
- `reuseTransportCertificatesForHttp: false` means the tool will create separate certificates for the transport layer and the HTTP layer for each node. This is the default behavior. With separate certificates, you could, for example, use a different CN or SAN for the public REST API endpoint if needed. If you prefer to use one certificate per node for both transport and REST (as X-Pack often did), set this to `true` to reuse the transport cert for HTTP.

Node Certificates:

Under `nodes:`, list each node in your cluster. For each node:

- `name` is an identifier used for the output filenames. The tool will produce files like `<name>.pem`, `<name>.key` (for transport) and `<name>_http.pem`, `<name>_http.key` (for HTTP, if separate).
- `dn` is the Distinguished Name for the certificate's subject. Typically you'd use the node's fully qualified domain name (FQDN) as the CN, plus organization info. Make sure the CN or one of the SAN entries matches exactly what the node's actual hostname is (especially if you enable hostname verification, discussed later).
- `dns` is a list of DNS names to include in the Subject Alternative Name (SAN) extension of the cert. Include any hostnames that clients or nodes will use to connect. At minimum this usually includes the node's primary hostname. You can include multiple entries (e.g., an alias or short name).
- `ip` is a list of IP addresses to include in SAN if your nodes communicate via IP or clients might connect via IP address.

Ensure that for each Elasticsearch node you have a corresponding entry with the correct identifiers. This will let Search Guard verify the node's identity and also allow clients (like Kibana or curl) to verify the server certificate.

Admin Certificate:

Under `clients:`, we listed one entry for `admin`. This will produce a client certificate used for administrative tasks. We marked it with `admin: true`, which signals the TLS tool to indicate in the output config snippet that this certificate's DN should be added to `searchguard.authcz.admin_dn`. We gave it CN=admin (you can choose another name/OU, but avoid using wildcards here – Search Guard will only accept an exact match for admin DNs). The tool will output `admin.pem` and `admin.key`.

Feel free to add additional client certificates under `clients` if you need, for example, separate certificates for Kibana or other applications. (Those would typically not have `admin: true` since they're end-user clients, not administrators.)

Step 3: Generate the Certificates with the Search Guard TLS Tool

With `tlsconfig.yml` configured and your CA files in place, we can run the TLS tool to generate the new certificates. Since we are reusing an existing CA, we will **not** create a new CA; instead we'll use the CA files provided. Ensure that the `out/` directory (or the directory you plan to use as output) contains `xpack-`

`ca.pem` and `xpack-ca.key` (named exactly as in the config file). Then run the tool in **certificate generation mode**:

```
./tools/sgtlstool.sh -c tlsconfig.yml -crt
```

Here, `-c tlsconfig.yml` points the tool to your configuration file. Using the `-crt` flag without `-ca` tells the tool to create certificates using an **existing** local CA (it expects the CA cert/key to be present). If everything is set up correctly, the tool will read your `xpack-ca.pem` and `xpack-ca.key`, then generate the specified certificates.

Tool Output: After running, check the `out/` directory (or the specified `-t` target directory if you used one). You should see files for each node and client:

For each node (e.g., node1):

- `node1.pem` – the certificate (with the node's full certificate chain, including the CA if configured to include it).
- `node1.key` – the node's private key (encrypted if you used a password like `auto` or a custom password in config; unencrypted if `pkPassword: none`).
- `node1_http.pem` and `node1_http.key` – the REST layer certificate and key (since we set `reuseTransportCertificatesForHttp: false`). These will have a similar subject but possibly different SAN usage (the tool often just duplicates the entries unless you specify differently).
- `node1_elasticsearch_config_snippet.yml` – a handy snippet of configuration that Search Guard generates for convenience. This YAML file contains the recommended `elasticsearch.yml` settings for this node's certificates.

For admin client:

- `admin.pem` – the admin certificate (X.509 cert, likely with the CA chain).
- `admin.key` – the admin private key.

You will also see `root-ca.pem` (which should be essentially the same as `xpack-ca.pem` you provided, possibly copied) and `root-ca.key` (same as your CA key) in the output if they were not already present. The tool might copy/rename them based on `file:` name. For example, if you set `file: xpack-ca.pem`, it might output `xpack-ca.pem` and `xpack-ca.key` in the directory (or simply use those as-is). Check that these exist and match your originals.

Additionally, there may be a `root-ca.readme` (containing any passwords generated for the CA if `auto` was used) and a `client-certificates.readme` containing the auto-generated passwords for each certificate's key (if you used `auto` for `pkPassword`). Open this file to retrieve the passwords for `node1.key`, `node2.key`, etc., and `admin.key` if any were set. Keep these passwords secure.

If the tool printed any errors, read them carefully. Common issues might be:

- Misnamed files or paths (the tool couldn't find `xpack-ca.key` or the password was wrong).
- The CA key not being in the expected format (ensure it's PEM PKCS#8 as noted).
- Typos in the YAML (the tool will usually point out YAML syntax issues or missing fields).

Assuming generation succeeded, you now have all the necessary certificate files to configure Search Guard TLS.

Step 4: Deploy the Certificates and Update Elasticsearch Configuration

Now that you have the certificates, the next step is to put them in place and adjust your `elasticsearch.yml` on each node to use Search Guard's TLS settings.

Distribute the certificates:

- Copy each node's certificate and key (`nodeN.pem` , `nodeN.key` and `nodeN_http.pem` , `nodeN_http.key` if separate) to the respective Elasticsearch node's configuration directory (usually the `config/` folder). Protect the private keys with appropriate filesystem permissions (e.g., readable only by the Elasticsearch process owner).
- Also copy the CA certificate (`root-ca.pem` or your `xpack-ca.pem`) to each node's config directory (you might put it as `config/certs/ca.pem` for consistency). This will be used as the trust anchor for TLS.

You do not need to copy the CA **private key** to the nodes – only the certificate (public key) is needed for trust. The CA private key should be kept offline after you finish generating certs.

Elasticsearch `elasticsearch.yml` changes: For each node, open its `elasticsearch.yml` and replace the X-Pack TLS settings with the Search Guard settings. Here's an example of what to add/update:

```
# Enable TLS for transport and HTTP:
searchguard.ssl.transport.enabled: true
searchguard.ssl.http.enabled: true

# Transport layer TLS settings:
searchguard.ssl.transport.pemcert_filepath: node1.pem
searchguard.ssl.transport.pemkey_filepath: node1.key
searchguard.ssl.transport.pemkey_password: "<password>"   # if the node1.key is encrypted; omit if
none
searchguard.ssl.transport.pemtrustedcas_filepath: ca.pem

# HTTP (REST) layer TLS settings:
searchguard.ssl.http.pemcert_filepath: node1_http.pem
searchguard.ssl.http.pemkey_filepath: node1_http.key
searchguard.ssl.http.pemkey_password: "<password>"        # password for HTTP key if encrypted
searchguard.ssl.http.pemtrustedcas_filepath: certs/ca.pem

# Specify allowed node certificate DNs (from your generated certs):
searchguard.nodes_dn:
  - CN=node1.example.com,OU=IT,O=YourOrg,L=YourCity,C=YourCountry
  - CN=node2.example.com,OU=IT,O=YourOrg,L=YourCity,C=YourCountry

# Alternatice: Use wildcards in the DNs:
searchguard.nodes_dn:
  - CN=*.example.com,OU=IT,O=YourOrg,L=YourCity,C=YourCountry

# Specify admin certificate DN(s) for administrative access:
searchguard.authcz.admin_dn:
  - CN=admin,OU=IT,O=YourOrg,L=YourCity,C=YourCountry
```

Let's explain some of these settings:

We explicitly **enable** TLS on both layers with `searchguard.ssl.transport.enabled: true` and `searchguard.ssl.http.enabled: true`.

For each layer, we point Search Guard to the PEM files:

- `pemcert_filepath` is the path (relative to the `config/` directory) to the certificate file for this node.
- `pemkey_filepath` is the path to the private key file for this node.
- `pemkey_password` is needed if the key file is encrypted with a password. Use the password from the TLS tool (for example, from the readme or whatever you set in the config). If you left the keys unencrypted (`pkPassword: none`), you can omit these password settings.
- `pemtrustedcas_filepath` is the path to the CA certificate file that will be used to verify peers. Here we use the same CA for both transport and HTTP. If you had an intermediate CA in your chain, you could either include the whole chain in one PEM file or specify the appropriate trust anchors.

The `searchguard.nodes_dn` section is crucial in Search Guard. It lists the DNs of the certificates that are allowed as node certificates in the cluster. We list the subject DNs of **all** node certificates from our generated set. This ensures that each node will trust only those certificates (signed by the CA) for inter-node communication. Wildcards or regex can be used here for convenience (e.g., `CN=*.example.com,...` to match all nodes in a domain).

The `searchguard.authcz.admin_dn` section lists the DN of the admin certificate. This should match the subject of the `admin.pem` we generated (in our example, "CN=admin,OU=IT,O=YourOrg,..."). This tells Search Guard which client certificate to treat as an "admin" for using sgadmin or the REST API with administrative privileges. Only exact matches are allowed here (no wildcards).

Double-check these settings for each node (each node's config will point to its own certificate files but the same CA file, and the nodes_dn list remains the same on all nodes, containing all node DNs). Remove or comment out any old X-Pack `xpack.security.*.ssl` settings to avoid confusion, and ensure `xpack.security.enabled` is false or X-Pack security is disabled entirely (Search Guard replaces X-Pack security, so they cannot both be active).

Hostname verification: By default, Search Guard enables hostname verification on the transport layer for additional security. This means each node will check that the hostname of the other node matches the certificate's SAN or subject CN. Our certificate config included the actual hostnames, so this should pass as long as each node knows the other's hostname (e.g., via DNS or hosts file). If for some reason your environment doesn't allow proper DNS names (for example, you used IPs that don't match DNS names), you have two options:

Disable hostname verification (and possibly DNS resolution) by adding:

```
searchguard.ssl.transport.enforce_hostname_verification: false
searchguard.ssl.transport.resolve_hostname: false
```

in `elasticsearch.yml`. This will skip these checks.

Or ensure that the `CN` / `dns` in the cert matches what the node's hostname really is and that each node can resolve the other's name. Generally, for production, it's best to keep verification **enabled** and fix any mismatched names, as this adds another layer of protection against man-in-the-middle attacks within the cluster.

After updating the config on all nodes, restart your Elasticsearch cluster (full cluster restart is needed when switching security plugins and TLS settings). All nodes should start up and form a cluster using the Search Guard TLS settings. Verify in logs that each node initializes Search Guard without errors. If a node fails to join, check the logs for TLS handshake issues (common ones include certificate DN not listed in nodes_dn, wrong passwords, file permission issues, or hostname verification failures).

Step 5: Update Kibana and Client Settings to Trust the CA

Since we reused the existing CA, this step should be simple. Any client that was previously trusting the X-Pack certificates (which were signed by this CA) can continue to trust the same CA for Search Guard:

Kibana:
If Kibana was configured to connect to Elasticsearch over HTTPS with X-Pack, you likely had `elasticsearch.ssl.certificateAuthorities: ["path/to/ca.pem"]` configured (or you might have set `elasticsearch.ssl.verificationMode` to "certificate" or "full" along with providing the CA). Continue using the same CA file in Kibana's configuration. Kibana will then trust the new node certificates since they are issued by the same CA. If you didn't have TLS on HTTP before and are enabling it now, you will need to update Kibana's config to use HTTPS and provide the CA so it can validate Elasticsearch's cert.

Other clients/applications:
Similarly, any other internal clients or beats that connect to Elasticsearch should be configured or informed to trust the CA (again, they may already if nothing changed).

At this point, your Elasticsearch cluster should be up and running with Search Guard, using TLS on all channels with your original CA. Kibana should be able to connect via HTTPS, and your users/clients won't notice any certificate trust changes (because the chain of trust is identical to before).

Conclusion

Migrating TLS from X-Pack to Search Guard is largely a matter of reusing your existing trust material and mapping the configuration to Search Guard's requirements. We exported the existing CA from X-Pack (ensuring continuity of trust), then used the Search Guard TLS tool to generate new node certificates and an admin certificate signed by that CA. We updated Elasticsearch's configuration to use these certificates on both the transport and HTTP layers, added the required settings like `nodes_dn` and `admin_dn` for Search Guard, and adjusted Kibana to continue trusting the same CA.

With TLS in place, your Elasticsearch 8.x cluster is now encrypted and using Search Guard. You can proceed to configure Search Guard's other security features as needed, confident that the underlying TLS encryption is solid and based on your established CA. Happy migrating!

More Articles in this Series Coming Soon!

- From X-Pack to Search Guard: Part 2 - Roles and Permissions

- From X-Pack to Search Guard: Part 3 - Users
- From X-Pack to Search Guard: Part 4 - Document- and Field-Level Security

---

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-x-pack-to-search-guard-part-1-tls-setup/