SecurityAndNetworking

The Identity Provider is a web application that runs behind a web server. It isn't a terribly sophisticated application by modern standards, but in a few respects it's very unusual and these are the aspects many newcomers find confusing, particularly if Java is a foreign platform to them.

The IdP also makes use of cryptography in ways that you'll need to understand in at least a rudimentary way because the security of everything you do with the IdP depends upon it.

Java Container Considerations

Java servlet applications run in so-called Java containers, such as Jetty, Tomcat, JBoss, or Glassfish. Most servlet containers are themselves web servers, and have sufficient features, flexibility, and performance to support most deployments, at scale, without the use of a separate web server like Apache. The IdP also occasionally (less often these days) requires certain web server features that are easier to support when Jetty or Tomcat are used alone than other containers or non-Java web servers, which will be discussed further below.

In cases where use of a native web server is required, this is typically done by "connecting" the native web server to the container using either the AJP protocol (recommended where possible) or by HTTP reverse-proxying. In this case, the container generally must be configured to receive connections **only** from the native web server and failure to limit this will lead to significant security risk.

Ports and Connectors

As a standard web application, the IdP primarily operates on a "client-facing" port, generally 443 (the standard HTTP over TLS port). Typically it isn't necessary to worry about allowing access to port 80 to redirect traffic back to 443 and doing so will generally not work with most SAML profiles anyway. You should essentially never operate the IdP without TLS.

The key and certificate you use on port 443 should not be related to any of the keys or certificates you use in configuring the IdP software, and should be managed as you would for any secure web server. Use of an Extended Validation (EV) certificate may be beneficial because of the role the IdP plays.

HTTP Connectors in Java Containers

By default, most Java containers are provided with some kind of HTTP connector/listener on ports 8080 and 8443. This is because non-Windows operating systems restrict processes from binding to ports under 1024 unless they run as root. Some newer systems have more advanced policy control over this.

> **Privileged Ports**
>
> If you choose to run a Java-based web server by itself (without Apache or the like), the first thing you'll need to do is to prepare an environment for it to run under a non-root account but allow use of port 443 and possibly 80 (optional) using a technique such as a `setuid` mechanism or port forwarding. **Don't** skip this step and don't worry about the IdP until this is done. Your overall experience will be much more confusing and painful if you ignore this advice.

You'll typically see some kind of property on the port 8080 HTTP connector that references port 8443, usually called "redirectPort" or something similar. Make sure you set this to 443; the purpose of that setting is to auto-redirect non-secured traffic when certain Java application settings are used.

Don't confuse the use of port 8443 in these default configurations with the (historical, but less common recently) IdP use of port 8443 that is described in the next section. That's an entirely different matter that you need to think about separately, and it's easiest to do that if you eliminate the references to this port that exist by default first.

Back-Channel Support

Historically, Shibboleth IdP deployments have operated with a second HTTP connector for SOAP requests from Service Providers. This connector by convention usually listens on port 8443. The separate port is used primarily because the connector has generally relied on client certificate authentication at the TLS layer to authenticate requests from SPs. It's possible to host these SOAP services on the same port as the rest of the IdP's services, but this used to require changes to SP configuration, so is beyond the scope of this introductory topic.

Back-Channel Use Cases

Most newer deployments do **not** need to support the back channel. While there are a range of features that rely on the back channel, and probably more in the future, basic use of the IdP for SAML-based SSO does **not** require it. That said, the following features require a back channel, along with a brief explanation of why you might need them.

Attribute Query

SAML queries for attribute data are historically common in the Shibboleth world because the earliest versions based on SAML 1.1 relied on an attribute query to request attributes after processing a SAML 1.1 SSO response delivered by the browser. This is to avoid including attributes up front because SAML 1.1 does not allow for data to be encrypted, allowing it to be observed passing through the client.

Newer deployments may not require support for SAML 1.1 at all, since most SPs today have support for SAML 2.0. Even if you do need to support SAML 1.1, you may not consider the exposure of data in the browser to be a major consideration, depending on the sensitivity of the attributes. You could choose to enable so-called "attribute push" by including the attributes with SSO by enabling the `includeAttributeStatement` option.

Most deployments do not need support for SAML 2.0 attribute queries, though they are supported. In most cases it's advisable **not** to advertise support for that feature in your metadata. Including it when unneeded can cause some wasted traffic from Shibboleth SPs if you happen to issue a SAML 2.0 response to one that includes no attributes. In some cases, a redundant SAML 2.0 attribute query can cause a spurious error at the SP.

Lastly, you should be aware that use of this feature is essentially incompatible with attribute release consent. Consent should **not** enabled for any relying party that has the ability to issue a query.

SAML Artifacts

The Artifact feature in SAML is a pass-by-reference approach to the exchange of a SAML SSO assertion that relies on a redirect to the SP with a small artifact string that is "resolved" by sending a SOAP request to the IdP. Ordinarily, the default SAML flow used in Shibboleth is a POST to the SP in which the assertion is pushed in the body of a form. The primary advantage of the artifact approach is that it handles cases where an unprotected page includes embedded references to images or other content that are protected. It also has security properties that are considered to be an advantage by some people.

Artifact usage is not common today, and it introduces complications when clustering an IdP and when testing upgrades or configuration changes. If not supported, it's very important for correct operation of SPs that your metadata **not** include the endpoints that indicate support for this feature.

Back-Channel Logout

SAML logout can be initiated either with the browser or as a SOAP notification from an SP to the IdP. The latter is by far the most reliable way to process a logout, but is also very limited in effectiveness because many SAML implementations and applications require access to previously-set cookies to perform a logout.

Support for SOAP logout does require that the session store be on the server side, which is not the V3 default, and greatly complicates clustering. As a result, this isn't a feature likely to matter very much in most cases.

Back-Channel Support Implications

If you do decide you need to support the back channel, then the complication is to be able to accept **any** certificate presented by an SP. Most implementations of TLS certificate authentication in web servers are primitive at best and broken at worst, and have to be "convinced" to allow this.

Java Containers

When Java containers are used to handle this traffic, it is typical to require some kind of Java plugin to override the certificate handling. For containers that are formally supported, the installation pages for each container will include a reference to a plugin component and information on how to configure it. For other containers, you will be on your own and may need to forgo supporting the back channel.

Apache

Using Apache as the web server is possible, though discouraged, with two conditions:

- You typically want to be using an AJP-based proxy connector to your Java container, because only AJP allows the client certificate to be passed into the Java server so the IdP can evaluate it. Failing this, you will have to ensure that any requests are made with signed messages, and that isn't a default of Shibboleth SPs at the moment.
- You must set the option **SSLVerifyClient optional_no_ca** in your back-channel virtual host configuration.

The latter works, but is subject to certain limitations that will cause problems for some SPs because even when this option is used, the Apache module still applies some restrictions on the client certificate that are not strictly required by the IdP itself. So this will limit the interoperability of your deployment.

Server Certificates

While the back-channel connector is superficially "just" another TLS-enabled server port, you **SHOULD NOT** use a short-lived, commercially-obtained server certificate on this virtual host, as discussed further below.

Keys and Certificates

There are a number of different keys and certificates used by the IdP, each for a specific purpose. Most are holdovers from earlier versions (which is noted where appropriate) but a couple of the use cases are new and therefore introduce some different scenarios to be aware of.

Read the general topic on SAML Keys and Certificates before reading this section.

Your IdP Keys and Certificates

The following summarizes all of the keys and certificates that are part of the operation of an IdP. *It is the responsibility of the IdP operator to securely manage all private keys*.

Browser-facing TLS Server Key and Certificate

As always, you have to install a private key and certificate on the server to enable TLS to encrypt the traffic from clients (and nominally, but in reality just a fiction, to authenticate the server to the browser client). This is generally going to require a commercially-issued certificate rooted in a CA all browsers will trust. Make sure that you can change this key and certificate any time you care to or need to.

Use of browser-facing TLS key and certificate

This key and certificate is not used by Shibboleth directly, and you **SHOULD NOT** use this key (or certificate) in any of the other capacities described below. Doing so will unnecessarily complicate the management of your system and lead to frequent key rollover events, which are among the most expensive operations you will undertake in a federated deployment.

This is not a change from V2.

Lifecycle

The browser-facing key typically changes annually or semi-annually, and increasingly tends to be managed as part of a load balancer environment rather than on the IdP servers.

Signing Key and Certificate

The main key underlying most IdPs is the digital signing key. This is a private key used to sign SAML messages.

Protect your private signing key!

Make no mistake, a compromised signing key allows anybody with the key to impersonate your IdP and by extension all of its users.

The V3 installer (when not upgrading an older installation) will generate a private signing key and a long-lived, self-signed certificate containing the public key. The certificate is just a convenient container for the public key. In Shibboleth, or any compliant SAML system, the content of the certificate other than the key is totally ignored.

The certificate is communicated via a `<md:KeyDescriptor use="signing">` element in SAML metadata. The `use` XML attribute is important, as will be made clear below.

In a non-updated install of V3, the generated signing key pair will be prefixed **idp-signing** (the names can be overridden if desired via the *idp.properties* file).

In V2, the generated signing key pair was named **idp.key** and **idp.crt** (the names typically configured via *relying-party.xml).* An upgraded install of V3 will maintain the V2 approach and files by default.

Lifecycle

This keypair ideally never changes outside of a catastrophic event. If your policies require you to change it, you should budget a large amount of time from the IdP operations team and all of your affected customers to coordinate the change, or you will need to accept a significant amount of service disruption. Some disruption is also unavoidable in most deployments even if you do everything correctly.

Make sure you don't back up this key as part of general server backup. If you do, you will essentially give all of your backup operations staff the ability to compromise your organization through impersonation attacks with the key and that won't sit well with your auditors. If you want to avoid frequent demands to change the key, don't back it up "normally". Use a dedicated strategy to archive the key safely and limit access to a small number of authorized staff.

Back-Channel TLS Key and Certificate

This section only applies should you choose to operate the IdP over the second back-channel port, but can be ignored otherwise.

A second key pair essential to the security of the IdP is the private key and certificate used to enable TLS on the back channel, discussed extensively above. Depending on the SAML options used, compromise of this key could also lead to compromise of the IdP.

The deployment of this key and certificate involves your web server, rather than the IdP application, so actual configuration of this key is outside the scope of this documentation. And of course if you don't need to support the back channel, then you can ignore this key pair.

The V3 installer (when not upgrading an older installation) will generate a key pair to use for your back-channel TLS configuration. The certificate is communicated via a `<md:KeyDescriptor use="signing">` element in SAML metadata. (The `use="signing"` XML attribute refers to authentication generally, which includes back-channel TLS.) While most of the content of this certificate is ignored by Shibboleth SPs, the name of the server hosting the TLS endpoint does have to match the certificate's CommonName or a subjectAltName extension, following usual TLS practice. The installer does this for you automatically as best it can.

The older practice of using the same key for both digital signing and back-channel TLS is still permitted but is discouraged to limit key usage to specific purposes, which is a more secure practice. In particular, there are many more known attacks in recent years that have compromised TLS server keys than on keys used "behind the scenes".

In a non-updated install of V3, the generated back-channel key pair will be prefixed **idp-backchannel** but the files are not explicitly referenced by the IdP configuration because they are meant for use by the container to configure its back-channel TLS connector/listener.

In V2, the back-channel key pair was also not part of the configuration but conventionally was the same as the signing key pair. An upgraded install of V3 will not generate a new key pair and is generally assumed to be reusing the existing back-channel configuration choices.

Lifecycle

This keypair ideally never changes outside of a catastrophic event. The advice above for the signing key applies.

Encryption Key and Certificate

A third key pair, new to this version of the IdP, supports inbound XML Encryption. The private key is used to decrypt data encrypted under the public key. The latter is bound to a certificate that is communicated via a `<md:KeyDescriptor use="encryption">` element in SAML metadata.

There are only a few scenarios in which SPs are likely to need to encrypt data transmitted to the IdP and none of them are common. The most likely use case would be to encrypt the subject of a SAML logout request but Shibboleth SPs (and most other implementations) don't do this by default.

While it's been common practice with the Shibboleth SP to generate a single key pair for both signing and encryption, the IdP V3 installer generates a separate key pair for each, again for "best practice" reasons. It is particularly advisable to separate out the encryption key pair at the IdP because it has very different security properties. Compromising a decryption key allows data to be exposed in relatively rare cases while compromising the other keys is fatal to the security of the entire system.

Note that advertising a certificate in metadata with no `use` XML attribute is an indication that the same private key is used for both signing and decryption. After an upgrade, you may have older IdP metadata needing correction that mistakenly indicates this.

In a non-updated install of V3, the generated encryption key pair will be prefixed **idp-encryption** (the names can be overridden if desired via the **idp.properties** file).

In V2, this functionality did not exist and so no allowance for this existed in the configuration. An upgraded install of V3 essentially defers the timing of having such a key pair and some adjustments have to be made at some point later on. If you want to perform an upgrade using a clean install to start with, using the generated key pair is fine as long as you adjust your IdP's metadata shared with **every** SP to reflect the use of a separate key pair for this function, as noted above. Doing this while publishing metadata containing a certificate in metadata with no `use` XML attribute will cause problems.

Lifecycle

This keypair ideally never changes outside of a catastrophic event.

Cookie Encryption Key

A fourth key, also new to this version of the IdP, is an AES secret key used to secure cookies and certain other data by the IdP for its own use. This key is never shared with anybody else and is copied to every server making up a cluster.

An initial key is generated in a special kind of Java keystore file called a "JCEKS" keystore, which stores secret keys instead of private keys and certificates. A parallel file also tracks the key version number. A tool is provided to regularly update this secret key (and increase the version) and thereby continually maintain the secrecy of this key. This should be done at least daily to limit the chance for, and the damage from, exposure.

Lifecycle

This is a secret key that should be regular updated, though this is not absolutely essential. A simple command-line tool is provided to update the keystore and maintain older key versions as required.

Other Keys and Certificates

There are a number of additional certificates in SP metadata that are part of the story when operating an IdP and troubleshooting issues. They are mentioned here primarily to distinguish them from the keys and certificates discussed above.

SP Browser-facing TLS Server Key and Certificate

Like the IdP, most SPs will operate as a secure web application by supporting TLS on port 443 for browser access. This key and certificate has no relevance to the operation of the SP or IdP. The certificate should not be advertised in the SP's metadata unless it's being used for one of the other purposes below (which is generally a bad idea, but not precluded).

SP Signing and Back-Channel TLS Keys and Certificates

Some SPs may have private keys and certificates used for one or more of the following:

- Signing SAML requests (or, in a few cases such as Single Logout, SAML responses)
- Client or server authentication of back-channel SAML requests or responses (SOAP primarily)

Signing-only certificates are communicated via a `<md:KeyDescriptor use="signing">` element in SP metadata. When configured in the usual fashion, the IdP will completely ignore the content of the certificate other than the public key, but in a back-channel scenario the web server may or may not ignore the certificate's content depending on the software and configuration involved (discussed above).

An SP may advertise any number of such keys in its metadata and the IdP will accept any of them. This allows a new key to be added alongside an older key before adding it to the SP configuration, as part of replacing the old key with the new one.

SP Encryption Key and Certificate

Most, though not all, SPs have an encryption key pair. This is the analog of the IdP's encryption key pair above; the public key is used by the IdP to encrypt data while the SP uses its private key to decrypt it.

Encryption-only certificates are communicated via a `<md:KeyDescriptor use="encryption">` element in SP metadata. The IdP, when configured in the usual fashion, will completely ignore the content of the certificate other than the public key.

While an SP may advertise any number of encryption certificates in its metadata, the IdP will pick only one to encrypt with, typically the first certificate it finds that is suitable (almost always just the first one listed, barring some unusual cases). This means an SP cannot put different keys on different servers as part of a cluster of servers (the IdP won't know which one to pick), and it complicates changing the encryption key because the SP must be able to support decrypting data with both the old and new keys at once (Shibboleth can, but not every implementation can).

Metadata Signing Keys and Certificates

A final type of key you may encounter is one used to digitally sign SAML metadata. Shibboleth differs from most SAML implementations in that it favors the use of third-party sources of metadata (informally termed "federations") to vouch for the information supplied, much like a traditional certificate authority signs certificates. The public key certificate used to verify a signed metadata file is essentially a trust anchor, like a CA root certificate.

Most of the time, the public key in a signing certificate is used without regard for the certificate content, but it's also possible to configure the software to evaluate a signing certificate indirectly using traditional certificate path validation approaches as well. Individual metadata sources will usually supply documentation explaining how to verify the metadata they supply.