TLS Server Certificates

> ⚠ This document contains **DRAFT** material intended for discussion and comment by the InCommon participant community. Comments and questions should be sent to the InCommon participants mailing list.

TLS Server Certificates

This topic describes InCommon policy, requirements, and recommended practices regarding the security of browser-facing endpoints in metadata. The focus is on the keys and certificates used for browser-facing TLS. See the Key Usage topic regarding keys used for XML Signature and XML Encryption.

Server Endpoint Security

A necessary condition for a protocol endpoint to be secure is that the endpoint be protected by TLS.

> ⚠ **Policy regarding endpoints in metadata**
>
> All endpoints in IdP metadata MUST be protected with TLS. All endpoints in SP metadata SHOULD be protected with TLS.

TLS is not sufficient for endpoint security, however. Among other things, the TLS server certificate must be trusted. By way of definition, a TLS server certificate is *trusted* if all of the following conditions are true:

1. The certificate is signed by a trusted CA, or a CA that chains to a trusted CA.
2. The certificate has not expired.
3. The certificate has not been revoked.

Revocation is determined by checking an appropriate OCSP responder or CRL file.

An endpoint in metadata is *insecure* if it is not protected by TLS or the TLS server certificate is not trusted. An endpoint in metadata may also be insecure if a vulnerable version of TLS is used or weak ciphers are allowed.

> ⚠ **Policy regarding insecure endpoints in metadata**
>
> An entity descriptor in metadata that contains an insecure endpoint SHALL NOT be exported for the purposes of interfederation.

The Certificate Signing Request (CSR) used to obtain a trusted TLS server certificate SHOULD be generated directly on the IdP (or SP). Since the private key is usually created in conjunction with the CSR, this implies that the private key SHOULD be generated directly on the IdP (or SP). This is most secure and a highly RECOMMENDED practice.

If the CSR is generated on any other system, the private key MUST be encrypted at the time it is generated. While transmitting the private key to the IdP (or SP), a secure channel (such as ssh) SHOULD be used. Once the private key is securely stored on the IdP (or SP), it MAY be decrypted.

Handling the Private Key

A server's private TLS key is used to create a secure channel for transporting messages and content via HTTP. If the private key is lost or stolen, the holder has the power to intercept HTTP messages and content intended for the original server.

> ⛔ **Protect your private keys!**
>
> Positive control of your private keys must be maintained at all times. This includes the private keys used for browser-facing TLS, XML Signature, and XML Encryption.

A private TLS key is necessarily an online key, that is, it must be available to the HTTP server software at runtime. An online key may be encrypted, but the password or passphrase used to decrypt the key generally has to be available in an unencrypted file so that the HTTP server can be restarted in unattended fashion. Therefore an online key is considerably more vulnerable than an offline key, and must be protected accordingly.

If the private TLS key is stored in the file system as an ordinary file, it should have strict permissions to prevent unauthorized copying. For stronger protection, the private key may be stored in a hardware security module (HSM) that prevents export of the private key.

Key Generation

The security and privacy capabilities of your deployment depend on the security of the private key used in conjunction with browser-facing TLS. Develop a strategy for securing your private key **before** you generate it. For instance, the following strategy is highly recommended:

1. Start with a secure system for your IdP or SP…and keep it that way!
2. Generate the Certificate Signing Request (and hence the private key) directly on the secure system (IdP or SP)
3. Prevent the private TLS key from ever leaving the secure system
4. Ensure ongoing access to the private TLS key is strictly controlled

If you generate the Certificate Signing Request (and hence the private key) on any other system, then *that* system must also be secure. Indeed, every system the private key comes in contact with must be secure—at least as secure as the target system—or the private key must be encrypted at rest. Moreover, the private key must be encrypted while in transit to the secure system. All in all, that is *much* more work (and error-prone), so the best advice is don't do it. Generate your private keys on the target IdP (or SP) in the first place.

Here's another way to state these **basic security requirements**:

- *Until the private TLS key is securely stored on the target system (IdP or SP), it needs to be encrypted, both at rest and in transit*.
- *Under no circumstances should an unencrypted private TLS key come to rest on an insecure system or transit the network over an unprotected channel*.

It is easy to generate a private key and a corresponding Certificate Signing Request (CSR) with OpenSSL. On a linux system, type:

```
$ openssl req -new -newkey rsa:2048 -keyout server-key.pem -out server-cert.csr
```

The above command will store the private key in file server-key.pem and the corresponding CSR in file server-cert.csr. The latter is transmitted to a Certification Authority (CA) for signing. The resulting certificate (and the corresponding private key) are installed on the web server, which is configured for browser-facing TLS.

Test your OpenSSL software installation

OpenSSL is a subtly complicated tool having many versions with various capabilities (and bugs). It recommended that *all OpenSSL commands be tested in advance* to ensure that the tool is functioning as expected.

When you issue the above command, you will be prompted to enter a pass phrase for the purpose of decrypting an encrypted private key. If you're generating the private key directly on the IdP (or SP), it is not necessary to encrypt the private key (as discussed earlier). Simply press return when prompted to enter a pass phrase or use the `-nodes` option in the command above to issue an unencrypted private key straightaway.

If, however, you're generating the private key on any other host, you must encrypt the private key as stipulated earlier. Once the private key has been secured on the target IdP (or SP), it may be decrypted *in situ* with the following OpenSSL command:

```
$ openssl rsa -in key.pem -out key.pem
```

Simply press return when prompted to enter a new pass phrase.