ExplicitKeyTrustEngine

Identified by `type="ExplicitKey"`, this Trust Engine extracts keys and certificates directly from Metadata to evaluate signatures or TLS credentials. It is an enhanced version of the original BasicTrustEngine from older versions of the SP and is a superset, meaning that anything permitted by the old engine is still permitted.

It has the following behavior, implications, and problems.

Attributes

Name	Type	Default	Description
`type`	string	*Required* `ExplicitKey`	Plugin type name.

Child Elements

Name	Cardinality	Description
`<KeyInfoResolver>`	0 or 1	Advanced plugin interface for mapping `<ds:KeyInfo>` elements into keying material. Mostly for future use.

Validating Signatures

Each `<md:KeyDescriptor>` is resolved into a key. If the signature can be verified with one of the keys, then the engine returns success.

The following `<ds:KeyInfo>` children can be resolved into keys without additional plugin support:

- `<ds:KeyValue>/<ds:RSAKeyValue>`
- `<ds:KeyValue>/<ds:DSAKeyValue>`
- `<ds:X509Data>/<ds:X509Certificate>`

Note that under no circumstances is an X.509 certificate evaluated on any level when resolving a key. If it is a correctly encoded certificate, the signed key will be resolved. Valid or expired certificates issued by any signer with any sort of extensions are acceptable.

Revocation

Removing a `<md:KeyDescriptor>` from a metadata role is equivalent to revoking the enclosed key. The frequency of metadata update determines the window of exposure at any given provider, as does the use of the `validUntil` attribute in the metadata.

Key Rollover

Since each key found is evaluated, new keys can be introduced by registering them in metadata, waiting a pre-defined period of time for the change to propagate, and then finally deploying the new signing key.

Validating TLS and X.509 Credentials

Each `<md:KeyDescriptor>` is resolved into a key. If it matches the key inside the client or server TLS certificate presented, then the engine returns success.

The following `<ds:KeyInfo>` children can be resolved into keys without additional plugin support:

- `<ds:KeyValue>/<ds:RSAKeyValue>`
- `<ds:KeyValue>/<ds:DSAKeyValue>`
- `<ds:X509Data>/<ds:X509Certificate>`

Note that under no circumstances is an X.509 certificate evaluated on any level **by Shibboleth** during the operation. Valid or expired certificates issued by any signer with any sort of extensions are acceptable as long as they contain the same key that is presented.

However, see the Known Issues below for caveats caused by other software.

Revocation

Removing a `<md:KeyDescriptor>` from a metadata role is equivalent to revoking the enclosed key. The frequency of metadata update determines the window of exposure at any given provider, as does the use of the `validUntil` attribute in the metadata.

Key Rollover

Since each key found is evaluated, new keys can be introduced by registering them in metadata, waiting a pre-defined period of time for the change to propagate, and then finally deploying the new key.

Note that a change to a certificate will not necessarily require an update to metadata as long as the key hasn't changed.

Known Issues

Since most other SAML implementations are likely to enforce the usual TLS name checking rules when evaluating a TLS server certificate, this checking is not disabled by Shibboleth. As a result, the certificate's CN or subjectAltName(s) are checked against the expected hostname. This does not add any actual security to the system since a non-matching key won't be accepted anyway, but is still a requirement for deployers.

All known versions of Apache with mod_ssl have support for the `SSLVerifyClient optional_no_ca` setting to turn off certificate evaluation. However, even with this option, the depth of the certificate chain presented by the client is still wrongly evaluated in some strange fashion. Therefore, disabling verification must be accompanied by a high setting for `SSLVerifyDepth`.

In addition, newer versions of Apache + mod_ssl, specifically including 2.2.x, will accept expired client certificates from an SP **only** if they are **not** self-signed. Essentially they ignore the intent behind the "no_ca" notion, and construct a path, and if the root of the path is expired, it fails, even though the root is not actually trusted. With a self-signed certificate, the root is the certificate itself, thus an expired certificate fails. The solution to this is to ensure that self-signed certificates are long-lived and renew them as needed.

Example

```
<TrustEngine type="ExplicitKey"/>
```