Configuring IceSSL for Java

Previous Next

After installing IceSSL, an application typically needs to define a handful of additional properties to configure settings such as the location of certificate and key files. This page provides an introduction to configuring the plug-in for Java applications.

On this page:

Configuring Keystores for Java

IceSSL uses Java's native format for storing keys and certificates: the keystore.

A keystore is represented as a file containing key pairs and associated certificates, and is usually administered using the `keytool` utility supplied with the Java run time. Keystores serve two roles in Java's SSL architecture:

1. A keystore containing a key pair identifies the peer and is usually closely guarded.
2. A keystore containing public certificates represents the identities of trusted peers and can be freely shared. These keystores are also referred to as "truststores" when they are used to store only trusted certificate chains.

A single keystore file can fulfill both of these purposes.

Java supports a pluggable architecture for keystore implementations in which a system property selects a particular implementation as the default keystore type. IceSSL uses the default keystore type unless otherwise specified.

A password is assigned to each key pair in a keystore, as well as to the keystore itself. IceSSL must be provided with the password for the key pair, but the keystore password is optional. If a keystore password is specified, it is used only to verify the keystore's integrity. IceSSL requires that all of the key pairs in a keystore have the same password.

Our first example shows the properties that are sufficient in many situations:

```
Ice.Plugin.IceSSL=IceSSL:com.zeroc.IceSSL.PluginFactory
IceSSL.DefaultDir=/opt/certs
IceSSL.Keystore=keys.jks
IceSSL.Truststore=ca.jks
IceSSL.Password=password
```

IceSSL resolves the filenames defined in its configuration properties as follows:

1. Attempt to open the file as a class loader resource. This is especially useful for deploying applications with special security restrictions, such as applets.
2. Attempt to open the file in the local file system.
3. If `IceSSL.DefaultDir` is defined, prepend its value and try steps 1 and 2 again. The `IceSSL.DefaultDir` property is a convenient way to specify the default location of your keystore and truststore files.

The `IceSSL.Password` property specifies the password of the key pair.

It is a security risk to define a password in a plain text file, such as an Ice configuration file, because anyone who can gain read access to your configuration file can obtain your password. IceSSL also supports alternate ways to supply a password.

Back to Top ^

DSA Example for Java

Java supports both RSA and DSA keys. No additional properties are necessary to use DSA:

```
Ice.Plugin.IceSSL=IceSSL:com.zeroc.IceSSL.PluginFactory
IceSSL.DefaultDir=/opt/certs
IceSSL.Keystore=dsakeys.jks
IceSSL.Truststore=ca.jks
IceSSL.Password=password
```

Back to Top ^

ADH Example for Java

The following example uses ADH (the Anonymous Diffie-Hellman cipher). ADH is not a good choice in most cases because, as its name implies, there is no authentication of the communicating parties, and it is vulnerable to man-in-the-middle attacks. However, it still provides encryption of the session traffic and requires very little administration and therefore may be useful in certain situations. The configuration properties shown below demonstrate how to use ADH:

```
Ice.Plugin.IceSSL=IceSSL:com.zeroc.IceSSL.PluginFactory
IceSSL.DefaultDir=/opt/certs
IceSSL.Ciphers=NONE (DH_anon.*AES)
IceSSL.VerifyPeer=0
```

The `IceSSL.Ciphers` property enables support for ADH, which is disabled by default. Furthermore, this setting enables only those ADH ciphersuites that support AES encryption and eliminates other, lower-strength ciphersuites that may not be supported by recent JVMs.

The `IceSSL.VerifyPeer` property changes the plug-in's default behavior with respect to certificate verification. Without this setting, IceSSL rejects a connection if the peer does not supply a certificate (as is the case with ADH).

Back to Top ^

Configuring Ciphersuites for Java

A ciphersuite represents a particular combination of encryption, authentication and hashing algorithms. You can configure the ciphersuites that the underlying SSL engines are allowed to negotiate during handshaking with a peer. By default, IceSSL uses the underlying engine's default ciphersuites, but you can define a property to customize the ciphersuite list. Normally the default configuration is chosen to eliminate relatively insecure ciphersuites such as ADH, which is why it needs to be explicitly enabled as we saw in the example above.

IceSSL for Java interprets the value of `IceSSL.Ciphers` as a sequence of expressions that filter the selected ciphersuites using name and pattern matching. If the property is not defined, the Java security provider's default ciphersuites are used. The following table defines the valid expressions that may appear in the property value.

Expression	Description
`NONE`	Disables all ciphersuites. If specified, it must appear first.
`ALL`	Enables all supported ciphersuites. If specified, it must appear first. This expression should be used with caution, as it may enable low-security ciphersuites.
`NAME`	Enables the ciphersuite matching the given name.
`!`*`NAME`*	Disables the ciphersuite matching the given name.
`(EXP)`	Enables ciphersuites whose names contain the regular expression *EXP*.
`!(EXP)`	Disables ciphersuites whose names contain the regular expression *EXP*.

To determine the set of enabled ciphersuites, the plug-in begins with a list of ciphersuite names containing the default set as determined by the security provider. The expressions in the property value add and remove ciphersuites from this list and are evaluated in the order of appearance. For example, consider the following property definition:

```
IceSSL.Ciphers=NONE (RSA.*AES) !(EXPORT)
```

The expressions in this property have the following effects:

- `NONE` clears the list of enabled ciphersuites.
- `(RSA.*AES)` is a regular expression that enables ciphersuites whose names contain the string "RSA" followed by "AES", meaning ciphersuites using RSA authentication and AES encryption.
- `!(EXPORT)` is a regular expression that disables any of the selected ciphersuites whose names contain the string "EXPORT", meaning ciphersuites having export-quality strength.

As another example, this property adds anonymous Diffie-Hellman to the default set of ciphersuites and disables export ciphersuites:

```
IceSSL.Client.Ciphers=(DH_anon) !(EXPORT)
```

Finally, this example selects only one ciphersuite:

```
IceSSL.Client.Ciphers=NONE SSL_RSA_WITH_RC4_128_SHA
```

We recommend setting `IceSSL.Trace.Security=1` when experimenting with ciphersuite configurations. Pay special attention to the log output to verify the ciphersuites that IceSSL has enabled.

Back to Top ^

IceSSL Diagnostics for Java

You can use two configuration properties to obtain more information about the plug-in's activities. Setting `IceSSL.Trace.Security=1` enables the plug-in's diagnostic output, which includes a variety of messages regarding events such as ciphersuite selection, peer verification and trust evaluation. The other property, `Ice.Trace.Network`, determines how much information is logged about network events such as connections and packets. Note that the output generated by `Ice.Trace.Network` also includes other transports such as TCP and UDP.

You can also use a system property that displays a great deal of information about SSL certificates and connections, including the ciphersuite that is selected for use by each connection. For example, the following command sets the system property that activates the diagnostics:

```
$ java -Djavax.net.debug=ssl MyProgram
```

Back to Top ^

See Also

- Public Key Infrastructure
- Using IceSSL
- Programming IceSSL
- Advanced IceSSL Topics
- IceSSL.*