# IceSSL Properties

On this page:


- IceSSL.Alias
- IceSSL.CertAuthDir
- IceSSL.CertAuthFile
- IceSSL.CertFile
- IceSSL.CertVerifier
- IceSSL.CheckCertName
- IceSSL.CheckCRL
- IceSSL.Ciphers
- IceSSL.DefaultDir
- IceSSL.DH.bits
- IceSSL.EntropyDaemon
- IceSSL.FindCert.location.name
- IceSSL.ImportCert.location.name
- IceSSL.Keychain
- IceSSL.KeychainPassword
- IceSSL.KeyFile
- IceSSL.Keystore
- IceSSL.KeystorePassword
- IceSSL.KeystoreType
- IceSSL.Password
- IceSSL.PasswordCallback
- IceSSL.PasswordRetryMax
- IceSSL.Protocols
- IceSSL.Random
- IceSSL.Trace.Security
- IceSSL.TrustOnly
- IceSSL.TrustOnly.Client
- IceSSL.TrustOnly.Server
- IceSSL.TrustOnly.Server.AdapterName
- IceSSL.Truststore
- IceSSL.TruststorePassword
- IceSSL.TruststoreType
- IceSSL.VerifyDepthMax
- IceSSL.VerifyPeer


## IceSSL.Alias

### Synopsis

`IceSSL.Alias=`*`alias`* (Java)

### Description

Selects a particular certificate from the key store specified by `#IceSSL.Keystore`. The certificate identified by *`alias`* is presented to the peer request during authentication.

## IceSSL.CertAuthDir

### Synopsis

`IceSSL.CertAuthDir=`*`path`* (C++)

### Description

Specifies the `directory` containing the certificates of trusted certificate authorities. The directory must be prepared in advance using the OpenSSL utility `c_rehash`. The path name may be specified relative to the default directory defined by `#IceSSL.DefaultDir`.

## IceSSL.CertAuthFile

### Synopsis

`IceSSL.CertAuthFile=`*`file`* (C++, Ice Touch)

### Description

Specifies a file containing the certificate of a trusted certificate authority. The file name may be specified relative to the default directory defined by `#IceSSL.DefaultDir`.

### Platform Notes

C++

The certificate must be encoded using the PEM format.

Ice Touch

The certificate must be encoded using the DER format.

If `#IceSSL.DefaultDir` is also defined, IceSSL attempts to open the specified CA certificate file as `Resources/`*`DefaultDir/file`* in the application's resource bundle and as *`DefaultDir/file`* in the file system.

If `#IceSSL.DefaultDir` is not defined, IceSSL attempts to open the specified CA certificate file as `Resources/`*`file`* in the application's resource bundle and as *`file`* in the file system.

If this property is not defined, IceSSL looks for suitable CA certificates in the user's keychains and in the system keychain.

# IceSSL.CertFile

### Synopsis

`IceSSL.CertFile=`*`file`* (.NET, Ice Touch)
`IceSSL.CertFile=`*`file`*`[:`*`file`*`]` (C++ & Unix only)
`IceSSL.CertFile=`*`file`*`[;`*`file`*`]` (C++ & Windows only)

### Description

Specifies a file that contains the program's certificate, and may also contain the corresponding private key. The file name may be specified relative to the default directory defined by `#IceSSL.DefaultDir`.

### Platform Notes

C++

The private key is optional; if not present, the file containing the private key must be identified by `#IceSSL.KeyFile`. If a password is required, OpenSSL will prompt the user at the terminal unless the application has installed a password handler or supplied the password using `#IceSSL.Password`. The certificate must be encoded using the PEM format.

OpenSSL allows you to specify certificates for both RSA and DSA. To specify both certificates, separate the filenames using the platform's path separator character.

.NET

The file must use the PFX (PKCS#12) format and contain the certificate and its private key. The password for the file must be supplied using `#IceSSL.Password`.

Ice Touch

The file must use the PFX (PKCS#12) format and contain the certificate and its private key. The password for the file must be supplied using `#IceSSL.Password`. The certificate is imported into the keychain identified by the `#IceSSL.Keychain` property.

If `#IceSSL.DefaultDir` is also defined, IceSSL attempts to open the specified certificate file as `Resources/`*`DefaultDir/file`* in the application's resource bundle and as *`DefaultDir/file`* in the file system.

If `#IceSSL.DefaultDir` is not defined, IceSSL attempts to open the specified certificate file as `Resources/`*`file`* in the application's resource bundle and as *`file`* in the file system.

# IceSSL.CertVerifier

### Synopsis

`IceSSL.CertVerifier=`*`classname`* (Java, .NET)

### Description

Specifies the name of a Java or .NET class that implements the `IceSSL.CertificateVerifier` interface for performing application-defined certificate verification.

## IceSSL.CheckCertName

### Synopsis

`IceSSL.CheckCertName=`*`num`*

### Description

If *`num`* is a value greater than zero, IceSSL attempts to match the server's host name as specified in the proxy endpoint against the common name component of the server certificate's subject name. If no match is found, IceSSL attempts to match the host name against the DNS and IP address fields of the server certificate's subject alternative name. The search does not issue any DNS queries but simply performs a case-insensitive string match. The server's certificate is accepted if its common name or any of its DNS or IP addresses matches the host name in the proxy endpoint. IceSSL skips this validation step if the server does not supply a certificate, or if the proxy endpoint does not include a host name and `Ice.Default.Host` is not defined. This property has no affect on a server's validation of a client's certificate. If no match is found, IceSSL aborts the connection attempt and raises an exception. If not defined, the default value is zero.

## IceSSL.CheckCRL

### Synopsis

`IceSSL.CheckCRL=`*`num`* (.NET)

### Description

If *`num`* is a value greater than zero, IceSSL checks the certificate revocation list to determine if the peer's certificate has been revoked. If so, IceSSL aborts the connection and raises an exception.

## IceSSL.Ciphers

### Synopsis

`IceSSL.Ciphers=`*`ciphers`* (C++, Java)

### Description

Specifies the cipher suites that IceSSL is allowed to negotiate. A cipher suite is a set of algorithms that satisfies the four requirements for establishing a secure connection: signing and authentication, key exchange, secure hashing, and encryption. Some algorithms satisfy more than one requirement, and there are many possible combinations.

### Platform Notes

C++

The value of this attribute is given directly to the OpenSSL library and is dependent on how OpenSSL was compiled. You can obtain a complete list of the supported cipher suites using the command `openssl ciphers`. This command will likely generate a long list. To simplify the selection process, OpenSSL supports several classes of ciphers. Classes and ciphers can be excluded by prefixing them with an exclamation point. The special keyword `@STRENGTH` sorts the cipher list in order of their strength, so that SSL gives preference to the more secure ciphers when negotiating a cipher suite. The `@STRENGTH` keyword must be the last element in the list. The classes are:

| | |
|---|---|
| `ALL` | Enables all supported cipher suites. This class should be used with caution, as it may enable low-security cipher suites. |
| `ADH` | Anonymous ciphers. |
| `LOW` | Low bit-strength ciphers. |
| `EXP` | Export-crippled ciphers. |

Here is an example of a reasonable setting:

```
ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH
```

This value excludes the ciphers with low bit-strength and known problems, and orders the remaining ciphers according to their strength. Note that no warning is given if an unrecognized cipher is specified.

Java

The property value is interpreted as a list of tokens delimited by white space. The plug-in executes the tokens in the order of appearance in order to assemble the list of enabled cipher suites. The table below describes the tokens:

| | |
|---|---|
| `NONE` | Disables all cipher suites. If specified, it must be the first token in the list. |
| `ALL` | Enables all supported cipher suites. If specified, it must be the first token in the list. This token should be used with caution, as it may enable low-security cipher suites. |
| *`NAME`* | Enables the cipher suite matching the given name. |
| `!`*`NAME`* | Disables the cipher suite matching the given name. |
| *`(EXP)`* | Enables cipher suites whose names contain the regular expression *EXP*. For example, the value `NONE (.*DH_anon.*)` selects only cipher suites that use anonymous Diffie-Hellman authentication. |
| `!(`*`EXP`*`)` | Disables cipher suites whose names contain the regular expression *EXP*. For example, the value `ALL !(.*DH_anon.*)` enables all cipher suites except those that use anonymous Diffie-Hellman authentication. |

If not specified, the plug-in uses the security provider's default cipher suites. Enable `#IceSSL.Trace.Security` to determine which cipher suites are enabled by default, or to verify your cipher suite configuration.

# IceSSL.DefaultDir

### Synopsis

`IceSSL.DefaultDir=`*`path`*

### Description

Specifies the default directory in which to look for certificate, key, and key store files. See the descriptions of the relevant properties for more information.

# IceSSL.DH.*bits*

### Synopsis

`IceSSL.DH.`*`bits`*`=`*`file`* (C++)

### Description

Specifies a *`file`* containing Diffie Hellman parameters whose key length is *`bits`*, as shown in the following example:

```
IceSSL.DH.1024=dhparams1024.pem
```

IceSSL supplies default parameters for key lengths of 512, 1024, 2048, and 4096 bits, which are used if no user-defined parameters of the desired key length are specified. The file name may be specified relative to the default directory defined by `#IceSSL.DefaultDir`. The parameters must be encoded using the PEM format.

# IceSSL.EntropyDaemon

### Synopsis

`IceSSL.EntropyDaemon=`*`file`* (C++)

### Description

Specifies a Unix domain socket for the entropy gathering daemon, from which OpenSSL gathers entropy data to initialize its random number generator.

## IceSSL.FindCert.*location*.*name*

### Synopsis

`IceSSL.FindCert.`*`location`*`.`*`name`*`=`*`criteria`* (.NET)

### Description

Queries the certificate repository for matching certificates and adds them to the application's collection of certificates that are used for authentication. The value for *`location`* must be `LocalMachine` or `CurrentUser`.

The *`name`* corresponds to the .NET enumeration `StoreName` and may be one of the following values: `AddressBook`, `AuthRoot`, `CertificateAuthority`, `Disallowed`, `My`, `Root`, `TrustedPeople`, `TrustedPublisher`. It is also possible to use an arbitrary value for *`name`*.

The value for *`criteria`* may be `*`, in which case all of the certificates in the store are selected. Otherwise, *`criteria`* must be one or more *`field`*`:`*`value`* pairs separated by white space. The valid field names are described below:

| | |
|---|---|
| `Issuer` | Matches a substring of the issuer's name. |
| `IssuerDN` | Matches the issuer's entire distinguished name. |
| `Serial` | Matches the certificate's serial number. |
| `Subject` | Matches a substring of the subject's name. |
| `SubjectDN` | Matches the subject's entire distinguished name. |
| `SubjectKeyId` | Matches the certificate's subject key identifier. |
| `Thumbprint` | Matches the certificate's thumbprint. |

The field names are case-insensitive. If multiple criteria are specified, only certificates that match all criteria are selected. Values must be enclosed in single or double quotes to preserve white space.

Multiple occurrences of the property are allowed, but only one query is possible for each location/name combination. The certificates from all queries are combined to form the certificate collection, including a certificate loaded using `#IceSSL.CertFile`. Here are some sample queries:

```
IceSSL.FindCert.LocalMachine.My=issuer:verisign serial:219336
IceSSL.FindCert.CurrentUser.Root=subject:"Joe's Certificate"
```

A server requires a certificate for authentication purposes, therefore IceSSL selects the first certificate in the accumulated collection. This is normally the certificate loaded via `#IceSSL.CertFile`, if that property was defined. Otherwise, one of the certificates from `IceSSL.FindCert.`*`location`*`.`*`name`* is selected. Since IceSSL does not guarantee the order in which it evaluates `IceSSL.FindCert.`*`location`*`.`*`name`* properties, it is recommended that the criteria select only one certificate.

## IceSSL.ImportCert.*location*.*name*

`IceSSL.ImportCert.`*`location`*`.`*`name`*`=`*`file`*`[;`*`password`*`]` (.NET)

### Description

Imports the certificate in *`file`* into the specified certificate store. The value for *`location`* must be `LocalMachine` or `CurrentUser`. The *`name`* corresponds to the .NET enumeration `StoreName` and may be one of the following values: `AddressBook`, `AuthRoot`, `CertificateAuthority`, `Disallowed`, `My`, `Root`, `TrustedPeople`, `TrustedPublisher`. It is also possible to use an arbitrary value for *`name`*, which adds a new store to the repository. If you are importing a trusted CA certificate, it must be added to `AuthRoot` or `Root`.

The *`password`* is optional; it is only necessary if the certificate file also contains a private key or uses a secure storage format such as PFX.

The file name and password may be enclosed in single or double quotes if necessary. The file name may be specified relative to the default directory defined by `#IceSSL.DefaultDir`.

Importing a certificate into `LocalMachine` requires administrator privileges, while importing into `CurrentUser` may cause the platform to prompt the user for confirmation.

# IceSSL.Keychain

### Synopsis

`IceSSL.Keychain=`*`name`* (Ice Touch)

### Description

Specifies the name of keychain in which to import the certificate identified by `#IceSSL.CertFile`. If not defined, the keychain named `login` is used by default. Note that this property is only relevant for the iPhone simulator and Mac OS X targets.

# IceSSL.KeychainPassword

### Synopsis

`IceSSL.KeychainPassword=`*`password`* (Ice Touch)

### Description

Specifies the password for the keychain identified by `#IceSSL.Keychain`. If not defined, IceSSL attempts to open the keychain without a password. Note that this property is only relevant for the iPhone simulator and Mac OS X targets.

# IceSSL.KeyFile

### Synopsis

`IceSSL.KeyFile=`*`file`* (C++)

### Description

Specifies a file containing the private key associated with the certificate identified by `#IceSSL.CertFile`. The file name may be specified relative to the default directory defined by `#IceSSL.DefaultDir`. The key must be encoded using the PEM format.

# IceSSL.Keystore

### Synopsis

`IceSSL.Keystore=`*`file`* (Java)

### Description

Specifies a key store file containing certificates and their private keys. If the key store contains multiple certificates, you should specify a particular one to use for authentication using `#IceSSL.Alias`. IceSSL first attempts to open the file as a class loader resource and then as a regular file. If the file cannot be found in the file system, IceSSL attempts to open the file relative to the directory specified by `#IceSSL.DefaultDir`. The format of the file is determined by `#IceSSL.KeystoreType`.

If this property is not defined, the application will not be able to supply a certificate during SSL handshaking. As a result, the application may not be able to negotiate a secure connection, or might be required to use an anonymous cipher suite.

# IceSSL.KeystorePassword

### Synopsis

`IceSSL.KeystorePassword=`*`password`* (Java)

### Description

Specifies the password used to verify the integrity of the key store defined by `#IceSSL.Keystore`. The integrity check is skipped if this property is not defined.

It is a security risk to use a plain-text password in a configuration file.

# IceSSL.KeystoreType

### Synopsis

`IceSSL.KeystoreType=`*`type`* (Java)

### Description

Specifies the format of the key store file defined by `#IceSSL.Keystore`. Legal values are `JKS` and `PKCS12`. If not defined, the JVM's default value is used (normally `JKS`).

# IceSSL.Password

### Synopsis

`IceSSL.Password=`*`password`*

### Description

Specifies the password necessary to decrypt the private key. It is a security risk to use a plain-text password in a configuration file.

### Platform Notes

C++

This property supplies the password that was used to secure the private key contained in the file defined by `#IceSSL.CertFile` or `#IceSSL.KeyFile`. If this property is not defined and you have not installed a password callback object, OpenSSL will prompt the user for a password if one is necessary.

Java

This property supplies the password that was used to secure the private key contained in the key store defined by `#IceSSL.Keystore`. All of the keys in the key store must use the same password.

.NET

This property supplies the password that was used to secure the file defined by `#IceSSL.CertFile`.

Ice Touch

This property supplies the password that was used to secure the file defined by `#IceSSL.CertFile`.

# IceSSL.PasswordCallback

### Synopsis

`IceSSL.PasswordCallback=`*`classname`* (Java, .NET)

### Description

Specifies the name of a Java or .NET class that implements the `IceSSL.PasswordCallback` interface. Using a password callback is a more secure alternative to specifying a password in a plain-text configuration file.

# IceSSL.PasswordRetryMax

### Synopsis

`IceSSL.PasswordRetryMax=`*`num`* (C++)

### Description

Specifies the number of attempts IceSSL should allow the user to make when entering a password. If not defined, the default value is 3.

# IceSSL.Protocols

### Synopsis

`IceSSL.Protocols=`*`list`* (C++, Java, .NET)

### Description

Specifies the protocols to allow during SSL handshaking. Legal values are `SSL3` and `TLS1`. You may also specify both values, separate by commas or white space. If this property is not defined, the platform's default is used.

# IceSSL.Random

### Synopsis

`IceSSL.Random=`*`filelist`* (C++, Java)

### Description

Specifies one or more files containing data to use when seeding the random number generator. The file names should be separated using the platform's path separator (a colon on Unix and a semicolon on Windows). The file names may be specified relative to the default directory defined by `#IceSSL.DefaultDir`.

In Java, IceSSL first attempts to open the files as class loader resources and then as regular files.

# IceSSL.Trace.Security

### Synopsis

`IceSSL.Trace.Security=`*`num`* (C++, Java, .NET)

### Description

The SSL plug-in trace level:

| | |
|---|---|
| `0` | No security tracing (default). |
| `1` | Display diagnostic information about SSL connections. |

# IceSSL.TrustOnly

### Synopsis

`IceSSL.TrustOnly=`*`ENTRY`*`[;`*`ENTRY`*`;...]` (C++, Java, .NET)

### Description

Identifies trusted and untrusted peers. This family of properties provides an additional level of authentication by using the peer certificate's distinguished name (DN) to decide whether to accept or reject a connection.

Each *`ENTRY`* in the property value consists of relative distinguished name (RDN) components, formatted according to the rules in RFC 2253. Specifically, the components must be separated by commas, and any component that contains a comma must be escaped or enclosed in quotes. For example, the following two property definitions are equivalent:

```
IceSSL.TrustOnly=O="Acme, Inc.",OU=Sales
IceSSL.TrustOnly=O=Acme\, Inc.,OU="Sales"
```

Use a semicolon to separate multiple entries in a property:

```
IceSSL.TrustOnly=O=Acme\, Inc.,OU=Sales;O=Acme\, Inc.,OU=Marketing
```

By default, each entry represents an acceptance entry. A `!` character appearing at the beginning of an entry signifies a rejection entry. The order of the entries in a property is not important.

After the SSL engine has successfully completed its authentication process, IceSSL evaluates the relevant `IceSSL.TrustOnly` properties in an attempt to find an entry that matches the peer certificate's DN. For a match to be successful, the peer DN must contain an exact match for all of the RDN components in an entry. An entry may contain as many RDN components as you wish, depending on how narrowly you need to restrict access. The order of the RDN components in an entry is not important.

The connection semantics are described below:

1. IceSSL aborts the connection if any rejection or acceptance entries are defined and the peer does not supply a certificate.
2. IceSSL aborts the connection if the peer DN matches any rejection entry. (This is true even if the peer DN also matches an acceptance entry.)
3. IceSSL accepts the connection if the peer DN matches any acceptance entry, or if no acceptance entries are defined.

Our original example limits access to people in the sales and marketing departments:

```
IceSSL.TrustOnly=O=Acme\, Inc.,OU=Sales;O=Acme\, Inc.,OU=Marketing
```

If it later becomes necessary to deny access to certain individuals in these departments, you can add a rejection entry and restart the program:

```
IceSSL.TrustOnly=O=Acme\, Inc.,OU=Sales; O=Acme\, Inc.,OU=Marketing; !O=Acme\, Inc.,CN=John Smith
```

While testing your trust configuration, you may find it helpful to set the `#IceSSL.Trace.Security` property to a non-zero value, which causes IceSSL to display the DN of each peer during connection establishment.

This property affects incoming and outgoing connections. IceSSL also supports similar properties that affect only incoming connections or only outgoing connections.

# IceSSL.TrustOnly.Client

## Synopsis

`IceSSL.TrustOnly.Client=`*`ENTRY`*`[;`*`ENTRY`*`;...]` (C++, Java, .NET)
`IceSSL.TrustOnly.Client=`*`ID`* (Ice Touch)

## Description

Identifies trusted and untrusted peers for outgoing (client) connections. The entries defined in this property are combined with those of `#IceSSL.TrustOnly`.

## Platform Notes

Ice Touch

For an outgoing connection to succeed, the peer certificate's subject key identifier must match the property value exactly. The property value is formatted as a series of hexadecimal values separated by colons or spaces, as shown in the example below:

```
C2:E8:D3:33:D7:83:99:6E:08:F7:C2:34:31:F7:1E:8E:44:87:38:57
```

Since this value is specific to a single certificate authority, this property is valid only when used in conjunction with `IceSSL.CertAuthFile`.

# IceSSL.TrustOnly.Server

### Synopsis

`IceSSL.TrustOnly.Server=`*`ENTRY`*`[;`*`ENTRY`*`;...]` (C++, Java, .NET)

### Description

Identifies trusted and untrusted peers for incoming ("server") connections. The entries defined in this property are combined with those of `#IceSSL.TrustOnly`. To configure trusted and untrusted peers for a particular object adapter, use `#IceSSL.TrustOnly.Server.AdapterName`.

# IceSSL.TrustOnly.Server.*AdapterName*

### Synopsis

`IceSSL.TrustOnly.Server.`*`AdapterName`*`=`*`ENTRY`*`[;`*`ENTRY`*`;...]` (C++, Java, .NET)

### Description

Identifies trusted and untrusted peers for incoming (server) connections to the object adapter *`AdapterName`*. The entries defined in this property are combined with those of `#IceSSL.TrustOnly` and `#IceSSL.TrustOnly.Server`.

# IceSSL.Truststore

### Synopsis

`IceSSL.Truststore=`*`file`* (Java)

### Description

Specifies a key store file containing the certificates of trusted certificate authorities. IceSSL first attempts to open the file as a class loader resource and then as a regular file. If the file cannot be found in the file system, IceSSL attempts to open the file relative to the directory specified by `#IceSSL.DefaultDir`. The format of the file is determined by `#IceSSL.TruststoreType`.

If this property is not defined, IceSSL uses the value of `#IceSSL.Keystore` by default. If no truststore is specified and the keystore does not contain a valid certificate chain, the application will not be able to authenticate the peer's certificate during SSL handshaking. As a result, the application may not be able to negotiate a secure connection, or might be required to use an anonymous cipher suite.

# IceSSL.TruststorePassword

### Synopsis

`IceSSL.TruststorePassword=`*`password`* (Java)

### Description

Specifies the password used to verify the integrity of the key store defined by `#IceSSL.Truststore`. The integrity check is skipped if this property is not defined.

It is a security risk to use a plain-text password in a configuration file.

# IceSSL.TruststoreType

### Synopsis

`IceSSL.TruststoreType=`*`type`* (Java)

### Description

Specifies the format of the key store file defined by `#IceSSL.Truststore`. Legal values are `JKS` and `PKCS12`. If not defined, the default value is `JKS`.

# IceSSL.VerifyDepthMax

### Synopsis

`IceSSL.VerifyDepthMax=`*`num`* (C++, Java, .NET)

### Description

Specifies the maximum depth of a trusted peer's certificate chain, including the peer's certificate. A value of zero accepts chains of any length. If not defined, the default value is `2`.

# IceSSL.VerifyPeer

### Synopsis

`IceSSL.VerifyPeer=`*`num`* (C++, Java, .NET)

### Description

Specifies the verification requirements to use during SSL handshaking. The legal values are shown in the table below. If this property is not defined, the default value is `2`.

| | |
|---|---|
| `0` | For an outgoing connection, the client verifies the server's certificate (if an anonymous cipher is not used) but does not abort the connection if verification fails. For an incoming connection, the server does not request a certificate from the client. |
| `1` | For an outgoing connection, the client verifies the server's certificate and aborts the connection if verification fails. For an incoming connection, the server requests a certificate from the client and verifies it if one is provided, aborting the connection if verification fails. |
| `2` | For an outgoing connection, the semantics are the same as for the value `1`. For an incoming connection, the server requires a certificate from the client and aborts the connection if verification fails. |

### Platform Notes

.NET

This property has no effect on outgoing connections, since .NET always uses the semantics of value `2`. For an incoming connection, the value `0` has the same semantics as the value `1`.