# Configuring IceSSL for Secure Transport

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 iOS and macOS applications.

On this page:



## Configuring IceSSL for macOS

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

```
Ice.Plugin.IceSSL=IceSSL:createIceSSL
IceSSL.DefaultDir=/opt/certs
IceSSL.CertFile=cert.pfx
IceSSL.CAs=ca.pem
IceSSL.Password=password
```

The `IceSSL.DefaultDir` property is a convenient way to specify the default location of your certificate files. The properties that follow it define the files containing the program's certificate with private key, and trusted CA certificate, respectively. Finally, the `IceSSL.Password` property specifies the password necessary to open `cert.pfx`.

> ⚠ 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 ^

### Keychain Examples for macOS

IceSSL imports the certificate specified by `IceSSL.CertFile` into a keychain. IceSSL uses the user's default keychain unless you choose a different one by defining the `IceSSL.Keychain` property:

```
IceSSL.Keychain=Test Keychain
```

If the specified keychain file does not exist, IceSSL will create it. The file name is opened relative to the user's current working directory unless an absolute path name is provided.

The user's default keychain is normally unlocked after logging into the system, so IceSSL doesn't usually require a password to import a certificate into this keychain. However, if your keychain is not unlocked automatically, or if you've selected a different keychain, you can supply a password using the `IceSSL.KeychainPassword` property:

```
IceSSL.KeychainPassword=password
```

> ⚠ 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.

If you don't define `IceSSL.KeychainPassword` and a password is required to open the keychain, macOS will prompt the user for the password.

To use a certificate that's already in a keychain, you can omit the `IceSSL.CertFile` and `IceSSL.Password` properties and use `IceSSL.FindCert` instead:

```
Ice.Plugin.IceSSL=IceSSL:createIceSSL
IceSSL.DefaultDir=/opt/certs
IceSSL.CAs=ca.pem
IceSSL.FindCert=Label:Client
```

Here we've instructed IceSSL to locate the certificate in the default keychain labeled `Client` and use that as the program's identity.

Back to Top ^

## ADH Example for macOS

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:createIceSSL
IceSSL.Ciphers=(DH_anon*)
IceSSL.DHParams=dhparams.der
IceSSL.VerifyPeer=0
```

The `IceSSL.Ciphers` property enables support for ADH. We also recommend setting `IceSSL.DHParams` with the name of a DER-encoded file containing pre-generated DH parameters.

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 IceSSL for iOS

With iOS applications, certificate files are loaded from the application's resource bundle. If the application's target platform is macOS, certificate files can also be loaded directly from the file system. Consider the following properties:

```
IceSSL.DefaultDir=certs
IceSSL.CAs=cacert.der
IceSSL.CertFile=cert.pfx
IceSSL.Password=password
```

The `IceSSL.DefaultDir` property is a convenient way to specify the location of your certificate files. Defining `IceSSL.DefaultDir` means IceSSL searches for certificate files relative to the specified directory. For the properties in the example above, IceSSL composes the pathnames `certs/cacert.der` and `certs/cert.pfx`. If `IceSSL.DefaultDir` is not defined, IceSSL uses the certificate file pathnames exactly as they are supplied.

As mentioned earlier, IceSSL has different semantics for locating certificate files depending on the target platform. For the iPhone and iPhone simulator, IceSSL attempts to open a certificate file in the application's resource bundle as `Resources/`*`DefaultDir/file`* if `IceSSL.DefaultDir` is defined, or as simply `Resources/`*`file`* otherwise. If the target platform is macOS and the certificate file cannot be found in the resource bundle, IceSSL also attempts to open the file in the file system as *`DefaultDir/file`* if a default directory is specified, or as simply *`file`* otherwise.

IceSSL requires that the CA certificate file specified by `IceSSL.CAs` use the DER format. The certificate file in `IceSSL.CertFile` must use the Personal Information Exchange (PFX, also known as PKCS#12) format and contain both a certificate and its corresponding private key. The `IceSSL.Password` property specifies the password used to secure the certificate file.

Back to Top ^

## Keychain Examples for iOS

IceSSL imports the certificate specified by `IceSSL.CertFile` into a keychain. IceSSL uses the `login` keychain by default unless you choose a different one by defining the `IceSSL.Keychain` property:

```
IceSSL.Keychain=Test Keychain
```

The `login` keychain is the user's default keychain, which is normally unlocked after logging into the system. IceSSL does not usually require a password to import a certificate into the `login` keychain. However, if your `login` keychain is not unlocked automatically, or if you have selected a different keychain, you can supply a password using the `IceSSL.KeychainPassword` property:

```
IceSSL.KeychainPassword=password
```

Back to Top ^

# IceSSL Diagnostics for macOS

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.

macOS also includes a TLS logging facility. To enable it, run the following command in Terminal:

```
$ sudo defaults write /Library/Preferences/com.apple.security SSLDebugScope -bool true
```

To see the log entries you must enable logging for warnings in syslog:

```
$ sudo syslog -c 0 -w
```

Then run syslog in 'wait' mode:

```
$ syslog -w
```

Back to Top ^

## See Also

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

Previous Next