Programming IceSSL in .NET

This page describes the .NET API for the IceSSL plug-in.

On this page:

The IceSSL Plugin Interface in C#

Applications can interact directly with the IceSSL plug-in using the native C# interface `IceSSL.Plugin`. A reference to a `Plugin` object must be obtained from the communicator in which the plug-in is installed:

C#

```
Ice.Communicator comm = // ...
Ice.PluginManager pluginMgr = comm.getPluginManager();
Ice.Plugin plugin = pluginMgr.getPlugin("IceSSL");
IceSSL.Plugin sslPlugin = (IceSSL.Plugin)plugin;
```

The `Plugin` interface supports the following methods:

C#

```
namespace IceSSL
{
    using System.Security.Cryptography.X509Certificates;

    abstract public class Plugin : Ice.Plugin
    {
        abstract public void
        setCertificates(X509Certificate2Collection certs);

        abstract public void
        setCertificateVerifier(CertificateVerifier verifier);

        abstract public CertificateVerifier
        getCertificateVerifier();

        abstract public void
        setPasswordCallback(PasswordCallback callback);

        abstract public PasswordCallback
        getPasswordCallback();
    }
}
```

The methods are summarized below:

- `setCertificates`
  This method is for advanced use cases and rarely used in practice.

- `setCertificateVerifier`
  `getCertificateVerifier`
  These methods install and retrieve a custom certificate verifier object that the plug-in invokes for each new connection. `getCertificateVerifier` returns null if a verifier has not been set.

- `setPasswordCallback`
  `getPasswordCallback`

These methods install and retrieve a password callback object that supplies IceSSL with passwords. `getPasswordCallback` returns null if a callback has not been set. Using `setPasswordCallback` is a more secure alternative to setting passwords in clear-text configuration files.

Obtaining SSL Connection Information in C#

You can obtain information about any SSL connection using the `getInfo` operation on a `Connection` object. It returns an `IceSSL.NativeConnectionInfo` class instance that derives from the Slice class `IceSSL::ConnectionInfo`. The Slice base class is defined as follows:

Slice

```
module Ice {
    local class ConnectionInfo {
        bool incoming;
        string adapterName;
    };

    local class IPConnectionInfo extends ConnectionInfo {
        string localAddress;
        int localPort;
        string remoteAddress;
        int remotePort;
    };
};

module IceSSL {
    local class ConnectionInfo extends Ice::IPConnectionInfo {
        string cipher;
        Ice::StringSeq certs;
    };
};
```

In turn, the C# class `NativeConnectionInfo` is defined as follows.

C#

```
public sealed class NativeConnectionInfo : ConnectionInfo
{
    public System.Security.Cryptography.X509Certificates.X509Certificate2[] nativeCerts;
}
```

Installing a Certificate Verifier in C#

A new connection undergoes a series of verification steps before an application is allowed to use it. The low-level SSL engine executes certificate validation procedures and, assuming the certificate chain is successfully validated, IceSSL performs additional verification as directed by its configuration properties. Finally, if a certificate verifier is installed, IceSSL invokes it to provide the application with an opportunity to decide whether to allow the connection to proceed.

The `CertificateVerifier` interface has only one method:

C#

```
namespace IceSSL
{
    public interface CertificateVerifier
    {
        bool verify(NativeConnectionInfo info);
    }
}
```

IceSSL rejects the connection if `verify` returns `false`, and allows it to proceed if the method returns `true`. The `verify` method receives a `NativeConnectionInfo` object that describes the connection's attributes.

The `nativeCerts` member of `NativeConnectionInfo` is an array of certificates representing the peer's certificate chain. The array is structured so that the first element is the peer's certificate, followed by its signing certificates in the order they appear in the chain, with the root CA certificate as the last element. This member is null if the peer did not present a certificate chain.

The `cipher` member is a description of the ciphersuite that SSL negotiated for this connection. The local and remote address information is provided in `localAddress` and `remoteAddr`, respectively. The `incoming` member indicates whether the connection is inbound (a server connection) or outbound (a client connection). Finally, if `incoming` is true, the `adapterName` member supplies the name of the object adapter that hosts the endpoint.

The following class is a simple implementation of a certificate verifier:

C#

```
using System.Security.Cryptography.X509Certificates;

class Verifier : IceSSL.CertificateVerifier
{
    public boolean
    verify(IceSSL.NativeConnectionInfo info)
    {
        if (info.nativeCerts != null)
        {
            X500DistinguishedName dn = info.nativeCerts[0].IssuerName;
            if (dn.Name.ToLower().Contains("zeroc"))
            {
                return true;
            }
        }
        return false;
    }
}
```

In this example, the verifier rejects the connection unless the string `zeroc` is present in the issuer's distinguished name of the peer's certificate. In a more realistic implementation, the application is likely to perform detailed inspection of the certificate chain.

Installing the verifier is a simple matter of calling `setCertificateVerifier` on the plug-in interface:

C#

```
IceSSL.Plugin sslPlugin = // ...
sslPlugin.setCertificateVerifier(new Verifier());
```

You should install the verifier before any SSL connections are established. An alternate way of installing the verifier is to define the `IceSSL.CertVerifier` property with the class name of your verifier implementation. IceSSL instantiates the class using its default constructor.

You can also install a certificate verifier using a custom plug-in to avoid making changes to the code of an existing application.

The Ice run time calls the `verify` method during the connection-establishment process, therefore delays in the `verify` implementation have a direct impact on the performance of the application. Do not make remote invocations from your implementation of `verify`.

Converting Certificates in C#

IceSSL offers the following convenience method to create a certificate object from a PEM-encoded string:

C#

```
namespace IceSSL
{
    using System.Security.Cryptography.X509Certificates;

    public sealed class Util
    {
        // ...

        public static X509Certificate2
        createCertificate(string certPEM);
    }
}
```

Given a string in the PEM format, `createCertificate` returns the equivalent `X509Certificate2` object.

See Also

- Using Connections
- Public Key Infrastructure
- Configuring IceSSL
- Advanced IceSSL Topics