IceSSL

Security is an important consideration for many distributed applications, both within corporate intranets as well as over untrusted networks, such as the Internet. The ability to protect sensitive information, ensure its integrity, and verify the identities of the communicating parties is essential for developing secure applications. With those goals in mind, Ice includes the IceSSL *plug-in* that provides these capabilities using the Transport Layer Security (TLS) protocol.

Although security is an optional component of Ice, it is not an afterthought. The IceSSL plug-in integrates easily into existing Ice applications, in most cases requiring nothing more than configuration changes. Naturally, some additional effort is required to create the necessary security infrastructure for an application, but in many enterprises this work will have already been done.



IceSSL is available for C++, Java and .NET applications. Objective-C, Python, PHP, Ruby and Swift applications can use IceSSL for C++ via configuration.

On this page:

- Overview of TLS
- Public Key Infrastructure
- Requirements

Overview of TLS

The Transport Layer Security (TLS) cryptographic protocol is the de facto standard for secure network communication. Its support for authentication, non-repudiation, data integrity, and strong encryption makes it the logical choice for securing Ice applications.

TLS is the protocol that enables Web browsers to conduct secure transactions and therefore is one of the most commonly used protocols for secure network communication. You do not need to know the technical details of the TLS protocol in order to use IceSSL successfully (and those details are outside the scope of this text). However, it would be helpful to have a high-level understanding of how the protocol works and the infrastructure required to support it.

TLS provides a secure environment for communication (without sacrificing too much performance) by combining a number of cryptographic techniques:

- public key encryption
- symmetric (shared key) encryption
- message authentication codes
- digital certificates

When a client establishes a TLS connection to a server, a *handshake* is performed. During a typical handshake, digital certificates that identify the communicating parties are validated, and symmetric keys are exchanged for encrypting the session traffic. Public key encryption, which is too slow to be used for the bulk of a session's data transfer, is used heavily during the handshaking phase. Once the handshake is complete, TLS uses message authentication codes to ensure data integrity, allowing the client and server to communicate at will with reasonable assurance that their messages are secure.

Public Key Infrastructure

Security requires trust, and public key cryptography by itself does nothing to establish trust. TLS addresses the issue of trust using Public Key Infrastructure (PKI), which binds public keys to identities using certificates. A certificate *issuer* creates a certificate for an entity, called the *subject*. The subject is often a person, but it may also be a computer or a specific application. The subject's identity is represented by a *distinguished name*, which includes information such as the subject's name, organization and location. A certificate alone is not sufficient to establish the subject's identity, however, as anyone can create a certificate for a particular distinguished name.

In order to authenticate a certificate, we need a third-party to guarantee that the certificate belongs to the subject described by the distinguished name. This third party, called a Certificate Authority (CA), expresses this guarantee by using its own private key to sign the subject's certificate. Combining the CA's certificate with the subject's certificate forms a *certificate chain* that provides TLS with most of the information it needs to authenticate the remote peer. In many cases, the chain contains only the aforementioned two certificates, but it is also possible for the chain to be longer when the *root* CA issues a certificate that the subject may use to sign other certificates. Regardless of the length of the chain, this scheme can only work if we trust that the root CA has sufficiently verified the identity of the subject before issuing the certificate.

An implementation of the TLS protocol also needs to know which root CAs we trust. An application supplies that information as a list of certificates representing the trusted root CAs. With that list in hand, the TLS implementation authenticates a peer by obtaining the peer's certificate chain and examining it carefully for validity. If we view the chain as a hierarchy with the root CA certificate at the top and the peer's certificate at the bottom, we can describe TLS's validation activities as follows:

- The root CA certificate must be self-signed and be present among the application's trusted CA certificates.
- All other certificates in the chain must be signed by the one immediately preceding it.
- The certificates must not be expired or revoked.

Copyright 2018, ZeroC, Inc.

These tests certify that the chain is valid, but applications often require the chain to undergo more intensive scrutiny to determine whether the chain is trustworthy.

Commercial CAs exist to supply organizations with a reliable source of certificates, but in many cases a private CA is completely sufficient. You can create and manage your CA using freely-available tools, and in fact Ice includes a collection of utilities that simplify this process.

Depending on your implementation language, it may also possible to avoid the use of certificates altogether; encryption is still used to obscure the session traffic, but the benefits of authentication are sacrificed in favor of reduced complexity and administration.

Requirements

Integrating IceSSL into your application often requires no changes to your source code, but does involve the following administrative tasks:

- creating a public key infrastructure (if necessary)
- configuring the IceSSL plug-in
- modifying your application's configuration to install the IceSSL plug-in and use secure connections

The remainder of this discussion concentrates on plug-in configuration and programming.

Topics

- Using IceSSL
- Configuring IceSSL
- Programming IceSSL
- Advanced IceSSL Topics
- Setting up a Certificate Authority

Copyright 2018, ZeroC, Inc.