OrbixSSL C++ Programmer's and Administrator's Guide

IONA Technologies PLC
September 2000

M 2 4 7 7

Contents

Part II

OrbixSSL Administration

Part IV

OrbixSSL C++ Reference

Part V

Appendices

Preface

OrbixSSL integrates Orbix, IONA Technologies' implementation of the CORBA standard, and the Secure Sockets Layer (SSL) version three protocol. This integration allows Orbix applications to communicate using SSL security.

This guide presents details of the integration between Orbix C++ Edition and SSL and explains how to add SSL security to Orbix applications.

Audience

This guide is aimed at system administrators who wish to set up a secure OrbixSSL environment and programmers who wish to develop Orbix C++ Edition applications that communicate using SSL security.

This guide does not assume that the reader has any knowledge of SSL security issues. This guide assumes that programmers have significant knowledge of Orbix programming.

Orbix documentation is periodically updated. New versions between releases are available at this site:

http://www.iona.com/docs/orbix/orbix33.html

If you need assistance with Orbix or any other IONA products, contact IONA at support@iona.com.

Comments on IONA documentation can be sent to doc-feedback@iona.com.

Organization of this Guide

This guide is divided into five parts:

Part I, "Introduction"

This part provides an overview of SSL security and describes how SSL integrates with Orbix. It then provides a tutorial example of how to add SSL security to an existing Orbix application. Read this part first.

Part II, "OrbixSSL Administration"

This part describes how system administrators can configure the use of SSL security in OrbixSSL applications. Both administrators and programmers should read this part.

Part III, "OrbixSSL Programming"

The part describes the OrbixSSL application programming interface (API) and how to implement common programming tasks using the API. This part is for programmers, but contains useful background information for administrators.

Part IV, "OrbixSSL C++ Reference"

This part provides a complete reference for OrbixSSL C++ programmers. It provides detailed information about the OrbixSSL API.

Part V, "Appendices"

This part provides supplemental information about OrbixSSL configuration and the SSL administration tools supplied with OrbixSSL.

Document Conventions

This document uses the following typographical and keying conventions:

`Constant width`	Constant width words or characters represent source code or system values you must use literally, such as commands, options, and path names.
Italic	Italic words in normal text represent *emphasis* and new terms.
	Italic words or characters in code and commands represent variable values you must supply, such as arguments or commands or path names for your particular system.

This guide uses the following keying conventions:

... . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

Part I

Introduction

1

An Introduction to OrbixSSL

OrbixSSL integrates Orbix with Secure Sockets Layer (SSL) security. Using OrbixSSL, distributed applications can transfer confidential data securely across a network.

An Overview of OrbixSSL

Secure Sockets Layer (SSL) provides data security for applications that communicate across networks. SSL is a transport layer security protocol layered between application protocols and TCP/IP.

Orbix applications communicate using the CORBA standard Internet Inter-ORB Protocol (IIOP) or IONA Technologies' proprietary Orbix protocol. These application-level protocols are layered above the transport-level protocol TCP/IP. OrbixSSL applications communicate using IIOP or the Orbix protocol layered above SSL. Figure 1.1 on page 4 illustrates how the SSL protocol layer integrates with Orbix communications.

All OrbixSSL components, including the Orbix daemon and Orbix utilities, and all OrbixSSL applications can communicate using SSL. OrbixSSL imposes few requirements on administrators and programmers who wish to support SSL communications in Orbix applications.

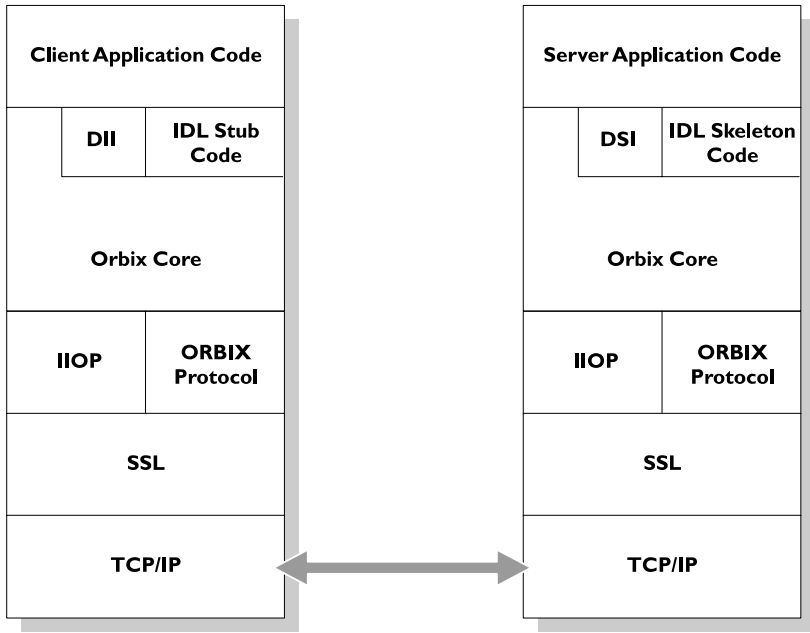

Figure 1.1: *The Role of SSL in Orbix Client/Server Communications*

OrbixSSL administrators use a single configuration file to configure a high-level security policy for a distributed system. OrbixSSL programmers develop standard Orbix applications that automatically communicate using SSL. The details of the SSL protocol are hidden, but programmers can use the OrbixSSL application programming interface (API) to customize SSL communications.

OrbixSSL applications can be configured to support any or all of the following options:

- IIOP
- IIOP over SSL

- Orbix Protocol
- Orbix Protocol over SSL

OrbixSSL acts as a dynamic upgrade to Orbix C++ Edition and Orbix Java Edition. Existing applications continue to work as before.

An Overview of SSL Security

SSL provides authentication, privacy, and integrity for communications across TCP/IP connections. Authentication allows an application to verify the identity of another application with which it communicates. Privacy ensures that data transmitted between applications can not be eavesdropped on or understood by a third party. Integrity allows applications to detect if data was modified during transmission.

Authentication in SSL

SSL uses Rivest Shamir Adleman (RSA) public key cryptography for authentication. In public key cryptography, each application has an associated public key and private key. Data encrypted with the public key can be decrypted only with the private key. Data encrypted with the private key can be decrypted only with the public key.

Public key cryptography allows an application to prove its identity by encoding data with its private key. As no other application has access to this key, the encoded data must derive from the true application. Any application can check the content of the encoded data by decoding it with the application's public key.

The SSL Handshake Protocol

Consider the example of two applications, a client and a server. The client connects to the server and wishes to send some confidential data. Before sending application data, the client must ensure that it is connected to the required server and not to an impostor.

When the client connects to the server, it confirms the server identity using the SSL handshake protocol. A simplified explanation of how the client executes this handshake in order to authenticate the server is as follows:

1. The client initiates the SSL handshake by sending the initial SSL handshake message to the server.
2. The server responds by sending its *certificate* to the client. This certificate verifies the server's identity and contains its public key.
3. The client extracts the public key from the certificate and encrypts a symmetric encryption algorithm session key with the extracted public key.
4. The server uses its private key to decrypt the encrypted session key which it will use to encrypt and decrypt application data passing to and from the client. The client will also use the shared session key to encrypt and decrypt messages passing to and from the server.

For a complete description of the SSL handshake, refer to the *Netscape Communications SSL V3.0* specification, available from `www.netscape.com`.

The SSL protocol permits a special optimized handshake in which a previously established session can be resumed. This has the advantage of not needing expensive public key computations. The SSL handshake also facilitates the negotiation of ciphers to be used in a connection.

The SSL protocol also allow the server to authenticate the client. Client authentication, which is supported by OrbixSSL, is optional in SSL communications.

As any application can have a public and private key pair, the transfer of the public key must be accompanied by additional information that proves the key is associated with the true server and not some other application. For this reason, the key is transmitted as part of a certificate.

Certificates in SSL Authentication

The public key is transmitted as part of a certificate. A certificate is used to ensure that the public key submitted is in fact the public key which belongs to the submitter. For the certificate to be acceptable to the client, it must have been digitally signed by a certification authority (CA) that the client explicitly trusts.

The International Telecommunications Union (ITU) recommendation X.509 defines a standard format for certificates. SSL authentication uses X.509 certificates to transfer information about an application's public key.

An X.509 certificate includes the following data:

- The name of the entity identified by the certificate.
- The public key of the entity.
- The name of the certification authority that issued the certificate.

The role of a certificate is to match an entity name to a public key. A CA is a trusted authority that verifies the validity of the combination of entity name and public key in a certificate. You must specify trusted CAs in order to use OrbixSSL.

According to the SSL protocol, it is unnecessary for applications to have access to all certificates. Generally, each application only needs to access its own certificate and the corresponding issuing certificates. Clients and servers supply their certificates to applications that they want to contact during the SSL handshake. The nature of the SSL handshake is such that there is nothing insecure in receiving the certificate from an as yet untrusted peer. The certificate will be checked to make sure that it has been digitally signed by a trusted CA and the peer will have to prove its identity during the handshake.

Privacy of SSL Communications

When a client authenticates a server, confidential data sent by the client can be encoded by the server's public key. It is only the actual server application that will be able to decode this data, using the corresponding private key.

Immediately after authentication, an SSL client application sends an encoded data value to the server. This unique session encoded value is a key to a symmetric cryptographic algorithm.

A symmetric cryptographic algorithm is an algorithm in which a single key is used to encode and decode data. Once the server has received such a key from the client, all subsequent communications between the applications can be encoded using the agreed symmetric cryptographic algorithm. This feature strengthens SSL security.

Examples of symmetric cryptographic algorithms used to maintain privacy in SSL communications are the Data Encryption Standard (DES) and RC4.

Integrity of SSL Communications

The authentication and privacy features of SSL ensure that applications can exchange confidential data that cannot be understood by an intermediary. However, these features do not protect against the modification of encrypted messages transmitted between applications.

To detect if an application has received data modified by an intermediary, SSL adds a message authentication code (MAC) to each message. This code is computed by applying a function to the message content and the secret key used in the symmetric cryptographic algorithm.

An intermediary cannot compute the MAC for a message without knowing the secret key used to encrypt it. If the message is corrupted or modified during transmission, the message content will not match the MAC. SSL automatically detects this error and rejects corrupted messages.

2

Getting Started with OrbixSSL

OrbixSSL provides SSL security for communications between components of your CORBA applications. This chapter shows you how to introduce SSL security to an existing application.

Using OrbixSSL, your CORBA applications benefit from the authentication, privacy, and integrity of SSL communications. When you create an OrbixSSL application, you must supply the information necessary to complete the authentication process. OrbixSSL then ensures the privacy and integrity of your communications without any intervention from you.

The SSL handshake, described in Chapter 1, enables components of your OrbixSSL application to authenticate each other. To ensure every SSL handshake completes successfully, each authenticated component must be able to access its certificate and private key.

There are two ways to provide this information to OrbixSSL applications. Administrators can use the OrbixSSL configuration file. Programmers can use the OrbixSSL application programming interface (API). This chapter uses some basic administration and some basic programming to show how you can add SSL security to an existing Orbix demonstration application.

Overview of the Application

The Orbix `demos` directory contains several demonstration programs, including a basic banking application, located in the `banksimple` subdirectory. In this application, an Orbix server creates a single object that implements the IDL interface `Bank`.

The server uses OrbixNames to associate a name with the `Bank` object. To begin communicating with the server, a client gets a reference to the `Bank` object from OrbixNames.

The client uses the `Bank` object to create `Account` objects. An `Account` object allows a client to manipulate a single bank account; for example, to query the balance of the account or deposit money in the account.

The IDL definitions for this application are as follows:

```
module BankSimple {
   typedef float CashAmount;

   interface Account;

   interface Bank {
      Account create_account (in string name);
      Account find_account (in string name);
   };

   interface Account {
      readonly attribute string name;
      readonly attribute CashAmount balance;
      void deposit (in CashAmount amount);
      void withdraw (in CashAmount amount);
   };
};
```

Running the Application without SSL

Without SSL, this application runs as follows:

1. The server gets a reference to OrbixNames. Implicitly, the server contacts the Orbix daemon.
2. The server uses OrbixNames to associate a name with the `Bank` object.

3. The client gets a reference to OrbixNames. Implicitly, the client contacts the Orbix daemon.
4. The client uses OrbixNames to get a reference to the `Bank` object.
5. The client calls operation `create_account()` on the `Bank` object. Implicitly, the client contacts the Orbix daemon over the connection that is already established. The client then contacts the server.

   The server processes the call to `create_account()` and returns a reference to an `Account` object.
6. The client calls operations on the `Account` object.

These steps are illustrated in Figure 2.1. When the application runs without SSL, all communications between parts of the application are insecure.

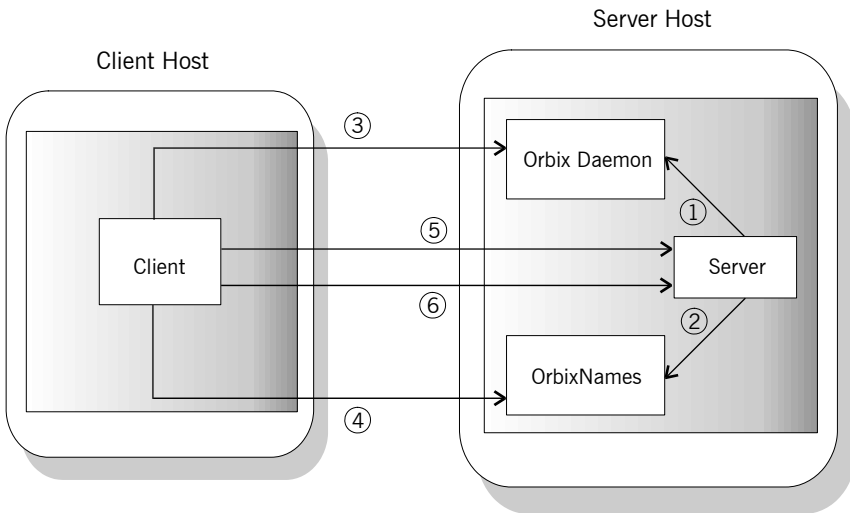

Figure 2.1: *Running the Banking Application*

Running the Application with SSL

When using SSL, each component of the application that acts as a server must be able to prove its identity. On first contact with another component, a server must be able to supply its certificate and encrypt messages with its private key. In this example, there are three servers: the bank server, the Orbix daemon, and the OrbixNames server.

With SSL, the application runs as shown in Figure 2.2:

1. The server gets a reference to OrbixNames. Implicitly, the server contacts the Orbix daemon.

   The Orbix daemon supplies its certificate to the server. The server uses this certificate to check the identity of the daemon.
2. The server uses OrbixNames to associate a name with the `Bank` object.

   OrbixNames supplies its certificate to the server. The server checks the identity of OrbixNames.
3. The client gets a reference to OrbixNames. Implicitly, the client contacts the Orbix daemon.

   The Orbix daemon supplies its certificate to the client. The client checks the identity of the Orbix daemon.
4. The client uses OrbixNames to get a reference to the `Bank` object.

   OrbixNames supplies its certificate to the client. The client checks the identity of OrbixNames.
5. The client calls operation `create_account()` on the `Bank` object. Implicitly, the client contacts the Orbix daemon over the secure connection that is already established. The client then contacts the server.

   The server supplies its certificate to the client. The client checks the identity of the server.

   The server processes the call to `create_account()` and returns a reference to an `Account` object.
6. The client calls operations on the `Account` object over a secure connection.

With SSL security, all the servers in the application can be identified and all communications between application components take place over secure connections.

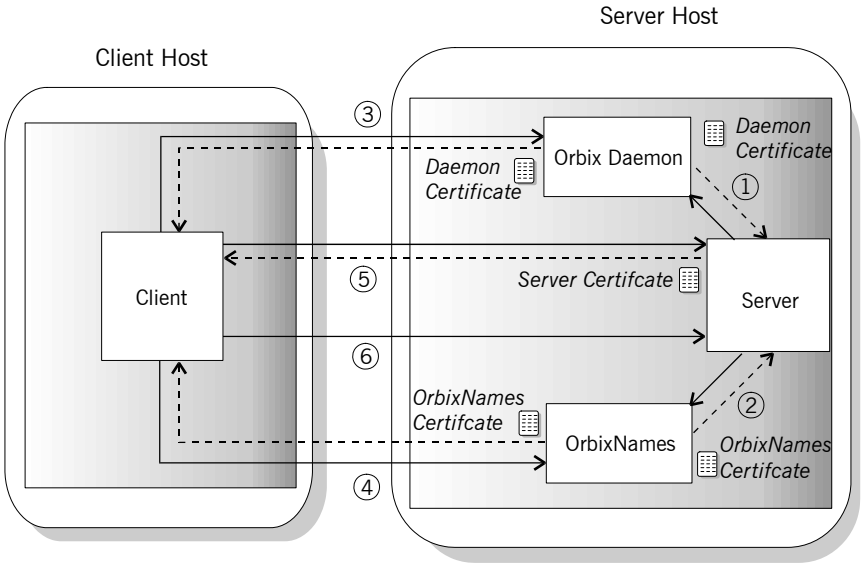

Figure 2.2: *Running the Banking Application with SSL Security*

To run this example, you must:

1. Provide each server with access to its certificate.
2. For each component that acts as a client, provide information about which certificates to accept.
3. Add OrbixSSL initialization code to the client and server programs.
4. Provide each server with access to its private key.

This chapter shows you how to implement steps 1 and 2 using OrbixSSL administration and steps 3 and 4 using the OrbixSSL API.

Modifying the Example Application

Before continuing with this chapter, go to the `demos` directory of your Orbix installation. In this directory, copy the subdirectory `banksimple` to `banksimplessl`. When this chapter instructs you to modify source files from the banking example, use the files in `banksimplessl`.

Providing Certificates for the Servers

In the banking application, the servers use demonstration certificates installed with OrbixSSL. Each certificate has a corresponding file in the OrbixSSL `certificates` directory. The certificates for the banking application are shown in Table 2.1.

Server	Certificate File
Bank	`demos/secure_bank_server`
OrbixNames	`services/orbix_names`
Orbix daemon	`services/orbix`

Table 2.1: *Demonstration Certificates Used by the Banking Application*

The `orbix` certificate is a general demonstration certificate for use with standard Orbix servers. The `secure_bank_server` certificate is a demonstration certificate specific to the bank server. Each of the demonstration certificates is signed by the OrbixSSL demonstration certificate authority (CA), called `demo_ca_1`.

WARNING: These certificates are completely insecure. Use them for OrbixSSL demonstration programs only. Do not use them in a deployed system. In a deployed system, you must create your own customized certificates for components of your application. The certificates for a deployed system should be signed by a CA that you can trust. Never trust the CA `demo_ca_1`. The process of creating and signing certificates is described in detail in Chapter 4 on page 51.

Using the OrbixSSL Configuration File

The OrbixSSL configuration file, `orbixssl.cfg`, enables you to specify how your applications use SSL. By default, this application is located in the `config` directory of your installation.

The OrbixSSL configuration file assigns values to OrbixSSL configuration variables. To enable SSL security, ensure that the configuration file includes the following setting:

```
OrbixSSL {
   IT_DISABLE_SSL = "FALSE";
};
```

If the value `OrbixSSL.IT_DISABLE_SSL` is set to `TRUE`, your system will not use SSL security.

Configuring All OrbixSSL Programs

Two OrbixSSL configuration variables allow a server to access its certificate:

- `IT_CERTIFICATE_PATH` specifies the directory in which the certificate file is stored in the file system.
- `IT_CERTIFICATE_FILE` specifies the name of the server's certificate file. Usually, you specify that this file is stored relative to the `IT_CERTIFICATE_PATH` directory.

The OrbixSSL configuration file uses the standard Orbix configuration syntax. By default, the variable `IT_CERTIFICATE_PATH` is set to the location of the OrbixSSL `certificates` directory, in the configuration scope `OrbixSSL`, for example:

```
OrbixSSL {
   IT_CERTIFICATE_PATH =
      "/opt/iona/OrbixSSL/certificates";
};
```

Variables set in the `OrbixSSL` configuration scope apply to all OrbixSSL applications, although you can override the values later in the configuration file.

Configuring a Single Program

To set the value of `IT_CERTIFICATE_FILE` for the banking server, append the following text to the file `orbixssl.cfg` on the server host:

```
Finance {
   BankingSystem {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "demos/secure_bank_server";
   };
};
```

The configuration scope `Finance.BankingSystem` is a custom scope for use by the banking server. You can create any number of custom scopes for your applications in `orbixssl.cfg`.

"Initializing OrbixSSL Configuration" on page 19 describes how you associate a specific configuration scope with an OrbixSSL program. The program then uses the settings defined in that scope. If a variable is not defined in the program scope, the program reads the variable setting from the scope `OrbixSSL`.

Configuring OrbixNames

To set the value of `IT_CERTIFICATE_FILE` for the OrbixNames server, append the following text to the file `orbixssl.cfg` on the server host:

```
OrbixNames {
   Server {
      IT_SECURITY_POLICY = "SECURE";
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "services/orbix_names";
   };
};
```

Configuring the Orbix Daemon

To set the value of IT_CERTIFICATE_FILE for the Orbix daemon, append the following text to the file orbixssl.cfg on the server host:

```
Orbix {
   orbixd {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "services/orbix";
   };
};
```

Specifying which Certificates to Accept

Every certificate is signed by a CA. When a client receives a certificate from a server, the client checks that the certificate is signed by a trusted CA. If the client trusts the CA, it accepts the certificate and connects to the server, otherwise it rejects the certificate.

When running an OrbixSSL application, you must specify a list of CAs that the application should trust. To do this, you first concatenate the certificate files for each trusted CA into a single file. You then use the OrbixSSL configuration variable IT_CA_LIST_FILE to specify the name and location of this file.

The banking example uses the insecure OrbixSSL demonstration CA, demo_ca_1. The CA certificate list file, which initially contains only the demo_ca_1 certificate, is located in the OrbixSSL ca_lists directory.

To specify that components of the banking example should accept certificates signed by demo_ca_1, add the following text to orbixssl.cfg on both the client and server hosts:

```
OrbixSSL {
   IT_CA_LIST_FILE = "OrbixSSL directory/
   ca_lists/demo_ca_list_1";
};
```

Replace *OrbixSSL directory* with the actual location of your OrbixSSL installation.

Initializing OrbixSSL

An OrbixSSL program initializes OrbixSSL using the OrbixSSL API. To get access to the OrbixSSL API, include the file `IT_SSL.h` in your programs:

```
#include <IT_SSL.h>
```

The OrbixSSL API contains a single initialization function that your OrbixSSL programs can call. This function is `IT_SSL::init()` and is defined as follows:

```
class IT_SSL {
public:
   virtual int init();
};
```

To call this function, use the globally available object `OrbixSSL`. For example, to initialize OrbixSSL in the banking client program, add the following code to the file `client.cxx`:

```
#include <IT_SSL.h>
...

int main (int argc, char *argv[]) {
   try {
      if (OrbixSSL.init() != IT_SSL_SUCCESS) {
         cout << "OrbixSSL initialization failed."
              << endl;
         return 1;
      }
      ...
   }
   ...
}
```

To initialize OrbixSSL in the banking server program, add the same code to the file `server.cxx`.

For OrbixSSL initialization to succeed, you must call the function `IT_SSL::init()` before your OrbixSSL program attempts to make any Orbix function calls. This includes calls to Orbix API functions that implicitly make remote calls, such as `CORBA::ORB::impl_is_ready()`.

Initializing OrbixSSL Configuration

As described in "Using the OrbixSSL Configuration File" on page 14, the example server uses the configuration scope `Finance.BankingSystem`. To specify that the server uses this scope, add the following code to `server.cxx`:

```
#include <IT_SSL.h>
...

int main (int argc, char *argv[]) {
   try {
      // Call IT_SSL::init().
      ...

      // Initialize configuration scope.
      if (OrbixSSL.initScope(
      "Finance.BankingSystem") != IT_SSL_SUCCESS)
         return 1;
   }
   ...
}
```

The OrbixSSL function `IT_SSL::initScope()` associates a custom scope in the OrbixSSL configuration file with you program.

Making Private Keys Available to Servers

By default, OrbixSSL expects the private key associated with a certificate to be appended to the certificate file. OrbixSSL expects the private key to be stored in encrypted Privacy Enhanced Mail (PEM) format; for example, all the OrbixSSL demonstration certificates have appended private keys in this format.

When a private key is encrypted in this way, you can access it only using a corresponding pass phrase. Specifying this private key pass phrase is a very important part of making a private key available to a server program.

Making a Private Key Available to a Server Program

The banking server uses the certificate file `secure_bank_server` in the OrbixSSL `certificates/demos` directory. This file has the associated private key appended, as expected by OrbixSSL.

When you run the server, it must supply its private key pass phrase to OrbixSSL. This allows OrbixSSL to read the private key and the server to encrypt data with this key, which is a critical part of SSL authentication.

The OrbixSSL API includes a single function that allows you to specify the pass phrase for your server. In the C++ API, this function is defined as:

```
class IT_SSL {
public:
   virtual int
      setPrivateKeyPassword (char* password);
   ...
};
```

In the banking example, you can complete the server application by calling this function. To do this, add this function call to the server file `server.cxx` as follows:

```
#include <IT_SSL.h>
...

int main (int argc, char *argv[]) {
   try {
      if (OrbixSSL.init() != IT_SSL_SUCCESS) {
         cout << "OrbixSSL initialization failed."
              << endl;
         return 1;
      }
      if (OrbixSSL.setPrivateKeyPassword
         ("demopassword") != IT_SSL_SUCCESS) {
         cout << "Private key pass phrase error."
              << endl;
         return 1;
      }
   }
   ...
}
```

In this example, the pass phrase is hard coded in the server program. In fact, this is insecure and useful only for demonstration purposes. In a deployed system, you must provide a secure mechanism for retrieving the server pass phrase. There are two fundamental approaches to this problem in OrbixSSL: an administrative approach, described in Chapter 5 on page 71 and a programmatic approach, described in Chapter 6 on page 81.

Making a Private Key Available to OrbixNames

Unlike an OrbixSSL server program, OrbixNames requires that the private key associated with a certificate is available in a separate file. The private key can also be appended to the certificate file, but OrbixNames ignores this appended key.

The OrbixNames demonstration certificate is associated with the private key file `orbix_names.jpk` in the OrbixSSL `certificates/services` directory. To specify this, add the following text to the OrbixSSL configuration file:

```
OrbixNames {
   Server {
      IT_PRIVATEKEY_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "services/orbix_names.jpk";
   };
};
```

This text assumes that you have already assigned the value of `IT_CERTIFICATE_PATH` in the `OrbixSSL` scope.

When you run the OrbixNames server, it requests that you input the pass phrase for its private key. Using the demonstration certificate, the pass phrase is `demopassword`.

Making a Private Key Available to the Orbix Daemon

As described in "Configuring the Orbix Daemon" on page 17, you can use the OrbixSSL configuration file to specify which certificate the Orbix daemon uses. When you run the Orbix daemon, it automatically uses the private key pass phrase associated with the demonstration certificate `orbix`. This pass phrase, `demopassword`, is established when you install OrbixSSL.

If you configure the daemon to use a different certificate, you must update the daemon executable with the pass phrase for the corresponding private key. To run the example described in this chapter, it is not necessary to do this.

To update the daemon, use the OrbixSSL `update` command. For example, on UNIX use the following command:

```
update orbixd "passphrase" 0
```

On Windows, use the following command:

```
update orbixd.exe "passphrase" 0
```

Review of the Development Steps

At this stage in the example, the steps outlined in "Running the Application with SSL" on page 12 are complete. It was necessary to:

1. Provide each server with access to its certificate.
2. For each component that acts as a client, provide information about which certificates to accept.
3. Add OrbixSSL initialization code to the client and server programs.
4. Provide each server with access to its private key.

To implement steps 1 and 2, you added configuration variables to the file `orbixssl.cfg`. To implement steps 3 and 4, you used the OrbixSSL API in the client file `client.cxx` and the server file `server.cxx`.

The remainder of this chapter shows you how to compile your modified banking example and how to run the application.

Compiling the Application

To use SSL security, your OrbixSSL program must be dynamically linked with the Orbix library and the OrbixSSL library. On each platform that Orbix supports, it provides two versions of the Orbix library: a single-threaded version and a

multi-threaded version. Similarly, OrbixSSL provides a single-threaded library and a multi-threaded library. Table 2.1 describes the OrbixSSL library names on UNIX and Windows platforms.

Platform	Single-Threaded	Multi-Threaded
UNIX	`libITtls`	`libITtlsmt`
Windows	`ITLSI.lib`	`ITLMI.lib`

Table 2.2: *OrbixSSL Library Names*

On Windows, the OrbixSSL libraries are import libraries for the associated dynamic link libraries (DLLs). On UNIX, the library file names can include additional information about the OrbixSSL version number and the C++ compiler associated with the libraries.

If you link your application with the single-threaded Orbix library, use the single-threaded OrbixSSL library also. Otherwise, use the multi-threaded OrbixSSL library.

To achieve this in the banking example, you must modify the makefile in your `banksimplessl` directory. The Orbix demonstration applications link with the multi-threaded Orbix library, so you must include the multi-threaded OrbixSSL library when linking the banking client and server programs.

On UNIX, edit the file `Makefile` in your `banksimplessl` directory. Add the multi-threaded OrbixSSL library to the client and server link lines as follows:

```
client: $(CLIENT_OBJS)
   $(C++) $(C++FLAGS) -o client $(CLIENT_OBJS) \
   $(LDFLAGS) $(ITORBIX) -lITtlsmt \
   $(ITDEM) $(ITNAM) $(SYSLIBS)

server: $(SERVER_OBJS)
   $(C++) $(C++FLAGS) -o server $(SERVER_OBJS) \
   $(LDFLAGS) $(ITORBIX) -lITtlsmt \
   $(ITDEM) $(ITNAM) $(SYSLIBS)
```

Compile the application using the `make` command.

On Windows, again edit the file `Makefile` in your `banksimplessl` directory. Add the multi-threaded OrbixSSL library to the client and server link lines as follows:

```
client.exe: $(CLIENT_OBJS)
   $(LINK) $(LINK_FLAGS_EXE) /OUT:$@ \
   $(CLIENT_OBJS) $(LINK_LIBS) ITLMI.lib

server.exe: $(SERVER_OBJS)
   $(LINK) $(LINK_FLAGS_EXE) /OUT:$@
$(SERVER_OBJS) $(LINK_LIBS) ITLMI.lib
```

Compile the application using the `nmake` command.

Running the Application

This section describes the steps required to run the server and client programs in the banking example.

Running the Server

To run the banking server, do the following on the server host:

1. Set the environment variable `IT_IONA_CONFIG_FILE` to the location of the Orbix configuration file, `iona.cfg`.
2. On UNIX, run the following `update` command to specify the location of the OrbixSSL configuration file, `orbixssl.cfg`:

   `update` *library OrbixSSL_directory* `2`

   Run this command for each of the OrbixSSL libraries, replacing *library* with the library file name and *OrbixSSL_directory* with the location of `orbixssl.cfg`.

   On Windows, set the environment variable `IT_SSL_CONFIG_PATH` to the location of `orbixssl.cfg`.
3. Set the environment variable that locates dynamic libraries, for example `PATH` on Windows, `LD_LIBRARY_PATH` on Solaris, or `SHLIB_PATH` on HP-UX, to include the Orbix `lib` directory.

4. Run the Orbix daemon, using the following command:

   ```
   orbixd
   ```

5. Register the OrbixNames server in the Implementation Repository. For example, using `putit` as follows:

   ```
   putit NS OrbixNames directory/server
   ```

   Replace *OrbixNames directory* with the full path of the directory in which the OrbixNames server is located.

6. Register the banking server in the Implementation Repository with server name `IT_Demo/banksimple/Bank`. For example, you can do this using the following commands:

   ```
   mkdirit IT_Demo
   mkdirit IT_Demo/banksimple
   putit IT_Demo/banksimple/Bank server directory/server
   ```

   Replace *server directory* with the full path of the directory in which your server is located.

7. Run the OrbixNames server, using the following command:

   ```
   ns -I nsior.ref
   ```

   This causes the server to write its IOR to a file named `nsior.ref`. You must use this file when running clients of OrbixNames.

8. Run the OrbixNames server again, using the following command:

   ```
   ns -secure
   ```

   When OrbixNames requests a pass phrase, enter the string `demopassword`.

9. Edit the Orbix configuration file included in `iona.cfg`. Add the following text to this file:

   ```
   Common {
      Services {
         NameService = "OrbixNames IOR";
      };
   };
   ```

   Replace *OrbixNames IOR* with the full IOR string contained in `nsior.ref`. If you paste this string from `nsior.ref`, ensure that no additional characters are added, such as line breaks.

10. Set up the naming context `IT_Demo.banksimple` in the Naming Service. For example, you can do this using the following commands:

```
putnewncns IT_Demo
putnewncns IT_Demo.banksimple
```

11. Run the banking server as follows:

```
server -bindns -timeout 60000
```

The server should now be running as a secure SSL server.

Running the Client

When the server is running, do the following on the client host:

1. Set the environment variable `IT_IONA_CONFIG_FILE` to the location of the Orbix configuration file, `iona.cfg`.
2. On UNIX, run the following `update` command to specify the location of the OrbixSSL configuration file, `orbixssl.cfg`:

   `update` *library OrbixSSL_directory* `2`

   Run this command for each of the OrbixSSL libraries, replacing *library* with the library file name and *OrbixSSL_directory* with the location of `orbixssl.cfg`.

   On Windows, set the environment variable `IT_SSL_CONFIG_PATH` to the location of `orbixssl.cfg`.
3. Edit the Orbix configuration file that is been included in `iona.cfg`. Add the following text to this file:

```
Common {
   Services {
      NameService = "OrbixNames IOR";
   };
};
```

   Replace *OrbixNames IOR* with the full IOR string contained in `nsior.ref` on the server host.
4. Set the environment variable that locates dynamic libraries, for example `PATH` on Windows, `LD_LIBRARY_PATH` on Solaris, or `SHLIB_PATH` on HP-UX, to include the Orbix `lib` directory.
5. Run the banking client as follows:

```
client
```

The application now runs as in the normal Orbix banking example. However, all communications between components of the application take place over SSL connections. During server authentication, OrbixSSL takes responsibility for checking the validity of certificates.

Part II

OrbixSSL Administration

3

Defining a Security Policy

Each installation of OrbixSSL includes a configuration file that allows you to specify how your applications use SSL security. This chapter describes how you can configure SSL security for each of your applications.

Defining a security policy means configuring your OrbixSSL applications to achieve the level of security required by your system. The OrbixSSL configuration file includes security settings that enable you to specify the location of certificates, which certificates programs should use, which certificates they should accept, and so on. You can apply OrbixSSL configuration settings to all your programs simultaneously, or to individual programs.

This chapter begins with an overview of OrbixSSL configuration. It then describes each of the configuration tasks required to define a comprehensive security policy.

Overview of OrbixSSL Configuration

The OrbixSSL configuration file, `orbixssl.cfg`, defines the security policy for your system. This file allows you to assign values to a set of OrbixSSL configuration variables. These variables specify how your applications use SSL security.

Using the OrbixSSL Configuration File

When you install OrbixSSL, the configuration file is located in the Orbix `config` directory. To define a security policy for your system, you must do the following on each host:

1. Add the required configuration variables to the file.
2. Put the file in a location where all OrbixSSL applications on the host can read it. Ensure that the file is a local file.
3. Restrict write access to a single trusted user. For example, on UNIX only the superuser `root` should be able to modify this file.
4. Include `orbixssl.cfg` in the Orbix configuration file `iona.cfg`.
5. On Windows, set the environment variable `IT_SSL_CONFIG_PATH` to the location of `orbixssl.cfg`.
6. On UNIX, run the `update` command on each OrbixSSL library:

   `update` *library SSL_config_directory* `2`

   Replace *library* with the library file name and *SSL_config_directory* with the location of `orbixssl.cfg`.

Applications read the OrbixSSL configuration file only on startup. If you change the settings in the file, applications must be restarted to read the new settings.

This chapter shows you how to assign configuration values and describes some of the most commonly used variables. Appendix B on page 201 provides a complete list of the OrbixSSL configuration variables and, where appropriate, their default values.

Assigning Values to Configuration Variables

The OrbixSSL configuration file uses the standard Orbix configuration syntax, described in the *Orbix C++ Administrator's Guide*. This syntax allows you to assign values to variables within configuration scopes. For example, in the Orbix configuration file `iona.cfg`, variables that are common to several IONA products are defined in the standard scope `Common`.

In a configuration file, the characters `{...};` delimit a configuration scope. For example, you could assign the value of the most basic OrbixSSL configuration variable, `IT_DISABLE_SSL`, in the `OrbixSSL` scope as follows:

```
OrbixSSL {
   IT_DISABLE_SSL = "FALSE";
};
```

If `OrbixSSL.IT_DISABLE_SSL` is set to `TRUE`, no application in your system can communicate using SSL. The default value for this variable is `FALSE`.

In OrbixSSL, the `OrbixSSL` scope enables you to configure SSL security for all your programs simultaneously. If a configuration variable value is the default for all programs, assign it in the `OrbixSSL` scope.

OrbixSSL also enables you to customize SSL support to meet the requirements of individual programs. You can override a configuration variable value, or assign a value to a new configuration variable in a scope that applies to a single program only.

By default, each server scope is defined with the standard scope `OrbixSSL.ServerNames`. For example, the default application scope for a server called `Bank` is `OrbixSSL.ServerNames.Bank`:

```
OrbixSSL {
   ServerNames {
      Bank {
         ...
      };
   };
};
```

If an application includes calls to OrbixSSL functions, you must define your own custom scope for that application. The OrbixSSL API enables the programmer to specify which scope the program uses. For example, the server `Bank` could use a custom scope, such as `Finance.BankingSystem`:

```
Finance {
   BankingSystem {
      ...
   };
};
```

If the programmer specifies that a program uses this scope, the program reads from the scope `OrbixSSL` any settings not defined in the custom scope. It ignores settings in the default application scope `OrbixSSL.ServerNames.Bank`.

To specify that a program uses a custom scope, a C++ programmer calls the method `IT_SSL::initScope()`, described on page 158. Use custom scopes for all servers that include OrbixSSL code. Use the default server scope `OrbixSSL.ServerNames` only for existing servers that do not contain any OrbixSSL code.

Including the OrbixSSL Configuration File

To include `orbixssl.cfg` in the file `iona.cfg`, use the `include` directive as follows:

```
# iona.cfg
ssl_dir = "SSL config directory";

include ssl_dir + "orbixssl.cfg";
```

The value *SSL config directory* should be the location of `orbixssl.cfg` in the local file system.

Configuring Server Authentication

Before running an OrbixSSL application, you must do the following to ensure that server authentication succeeds:

- Specify which certificate each server should use.
- Specify the private key pass phrase for each server.
- Specify which certificates each client should accept.

This section describes how to specify which certificate a server should use and which certificates a client should accept, using the OrbixSSL configuration file. To specify the private key pass phrase for a server, administrators can use the administration mechanism described in Chapter 5 on page 71, or programmers can use the OrbixSSL API.

For the purposes of SSL communications, a server is any Orbix program that can accept operation calls. This includes Orbix servers and clients that accept callbacks.

Specifying the Location of Certificates

To specify the location of your certificate files, add the following variable to `orbixssl.cfg` on the server host:

```
OrbixSSL {
   IT_CERTIFICATE_PATH = "certificate directory";
};
```

In most cases, only a single directory on each host contains certificates for OrbixSSL applications. Consequently, you usually assign the value of `IT_CERTIFICATE_PATH` in the `OrbixSSL` scope.

To specify the certificate that an application should use, set the variable `IT_CERTIFICATE_FILE` in `orbixssl.cfg`. Set this variable in the application scope, for example:

```
Finance {
   BankingSystem {
      IT_CERTIFICATE_FILE =
      OrbixSSL.IT_CERTIFICATE_PATH +
      "server certificate file name";
   };
};
```

Each Orbix service, such as OrbixNames or OrbixEvents, has its own configuration scope. For example, to set the value of `IT_CERTIFICATE_FILE` for OrbixNames, use the `OrbixNames.Server` scope:

```
OrbixNames {
   Server {
      IT_CERTIFICATE_FILE =
      OrbixSSL.IT_CERTIFICATE_PATH +
      "OrbixNames certificate file name";
   };
};
```

To set the value of `IT_CERTIFICATE_FILE` for the Orbix daemon, use the configuration scope `Orbix.orbixd`:

```
Orbix {
   orbixd {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "daemon certificate file name";
   };
};
```

If you change the certificate associated with the Orbix daemon, you must run the OrbixSSL `update` command to provide the corresponding private key pass phrase to the daemon executable.

Running the OrbixSSL Update Utility

Orbix executable files, such as the Orbix daemon and Orbix utilities, include embedded information about the pass phrase associated with their private keys. If you change the private key associated with these files, you must modify the embedded information using the OrbixSSL `update` utility.

In the same way, the OrbixSSL libraries contain an embedded private key pass phrase and the location of the OrbixSSL configuration file. You can update both these values by running `update` on the library files.

When updating a pass phrase, the `update` command takes the following form:

```
update filename passphrase {0 | 1}
```

If the file specified by *filename* is an executable, the final argument should be `0`. If the file is a library, the final argument should be `1`.

When updating the OrbixSSL libraries with the location of the OrbixSSL configuration file, the command takes the following form:

```
update filename location 2
```

You can also use the `update` utility to change embedded information in files associated with Orbix services, such as OrbixNames or OrbixManager. Consult the documentation associated with the service for more information.

Specifying Certificates to Accept

The program that receives a certificate must validate it to ensure the identity of the server. OrbixSSL does some basic validation, and the programmer can add more. To enable OrbixSSL to do this basic validation, you provide some information about which certificates your programs should accept.

The OrbixSSL CA certificate list file includes certificates that identify each CA your applications accept. As described in Chapter 2, "Getting Started with OrbixSSL", to specify the location of this file, you set the variable `IT_CA_LIST_FILE`, for example:

```
OrbixSSL {
   IT_CA_LIST_FILE =
      IT_CERTIFICATE_PATH + "CA list file name";
};
```

Usually, the value of IT_CA_LIST_FILE is the same for all applications on a single host.

Each X.509 certificate is signed by a CA. A CA certificate, included in the list file, can in turn be signed by another CA. This process is known as *certificate chaining*.

To ensure security of your OrbixSSL application, it is often necessary to limit the maximum number of certificates in a chain. To limit the maximum chain depth for each of your applications, assign a value to the variable IT_MAX_ALLOWED_CHAIN_DEPTH in the OrbixSSL scope, for example:

```
OrbixSSL {
   IT_MAX_ALLOWED_CHAIN_DEPTH = "2";
};
```

You can then assign a maximum chain depth for a specific application in the application scope, using the variable IT_DEFAULT_MAX_CHAIN_DEPTH:

```
Finance {
   BankingSystem {
      IT_DEFAULT_MAX_CHAIN_DEPTH = "1";
   };
};
```

The value for a specific application cannot be greater than the common maximum chain depth, specified by OrbixSSL.IT_MAX_ALLOWED_CHAIN_DEPTH. A chain depth of one indicates that a certificate can be signed by one trusted CA only. A chain depth of two indicates that the CA certificate can in turn be signed by a trusted CA. If any CA in the chain is trusted, the application certificate is considered valid by OrbixSSL.

Configuring Client Authentication

Some secure applications, for example Internet banking systems, require that clients can identify themselves to servers. These applications use an extended SSL handshake, in which the server validates the client certificate. Client authentication is optional in SSL security.

To specify that servers should authenticate clients by default, add the following to `orbixssl.cfg`:

```
OrbixSSL {
   IT_AUTHENTICATE_CLIENTS = "TRUE";
};
```

You can then override this default value for a particular server, if necessary:

```
Finance {
   BankingSystem {
      IT_AUTHENTICATE_CLIENTS = "FALSE";
   };
};

OrbixNames {
   IT_AUTHENTICATE_CLIENTS = "FALSE";
};
```

Similarly, you could set the default value of `IT_AUTHENTICATE_CLIENTS` to `FALSE` and override it for servers that should authenticate clients.

Using `IT_AUTHENTICATE_CLIENTS`, you can enable or disable client authentication for a server. However, the server programmer can also enable or disable client authentication using the OrbixSSL API. The API overrides your configuration settings.

In some cases, you might wish to enforce client authentication for a server and prevent the API from overriding your configuration. To do this, use the variable `IT_SERVERS_MUST_AUTHENTICATE_CLIENTS`, for example:

```
OrbixSSL {
   IT_SERVERS_MUST_AUTHENTICATE_CLIENTS = "TRUE";
};
```

This setting forces all servers to authenticate clients.

Securing the Orbix Daemon

The Orbix daemon process is an important element of an Orbix system. This process is responsible for managing the Implementation Repository and activating Orbix servers in response to operation calls from clients. Because it activates server processes, it is imperative that you ensure the security of the daemon.

As part of your security policy, OrbixSSL allows you to specify how the daemon process should communicate with your OrbixSSL programs. In particular, you can specify:

- Whether the daemon accepts SSL communications, non-SSL communications, or both.
- Whether the daemon authenticates clients.

This section describes how you address each of these issues using the OrbixSSL configuration file.

Configuring Orbix Daemon Communications

Some OrbixSSL systems contain only applications that communicate securely. Others contain some secure and some insecure applications. When securing the Orbix daemon, you must specify which types of communication the daemon should accept.

OrbixSSL defines four Orbix daemon types:

- *Secure daemon*. This type of daemon communicates using SSL only.
- *Restricted semi-secure daemon*. This type of daemon supports SSL communication, and permits only a restricted set of operations to insecure clients.
- *Semi-secure daemon*. This type of daemon supports SSL and non-SSL communication.
- *Insecure daemon*. This type of daemon does not support SSL communication.

A secure daemon does not accept communications from insecure applications and consequently prevents insecure clients from launching servers in your system. This daemon type is the most secure configuration and is recommended for systems in which all legitimate Orbix applications can communicate using SSL.

A restricted semi-secure Orbix daemon accepts communications from secure applications and insecure applications. However, insecure clients of the daemon can, by default, only cause servers to be launched.

A semi-secure Orbix daemon accepts communications from secure and insecure applications. This is useful for systems in which insecure Orbix applications coexist with secure applications and you wish to place no restrictions on insecure communications with the daemon.

An insecure daemon is recommended only for systems in which SSL security is completely disabled.

Specifying the Orbix Daemon Type

To specify which type of daemon should run on a particular host, add the following variable to `oribxssl.cfg` on that host:

```
OrbixSSL {
   IT_DAEMON_POLICY = "daemon type";
};
```

The legal values for *daemon type* correspond to the four available types of daemon:

- SECURE_DAEMON
- RESTRICTED_SEMI_SECURE_DAEMON
- SEMI_SECURE_DAEMON
- INSECURE_DAEMON

By default, the Orbix daemon uses the value SECURE_DAEMON. However, if IT_DISABLE_SSL is set to TRUE, the daemon type is INSECURE_DAEMON.

The INSECURE_DAEMON and SEMI_SECURE_DAEMON settings mean that insecure clients can connect to the daemon and call any operation on the daemon. This is not desirable in most situations.

Checking the Orbix Daemon Type

When you start the Orbix daemon, it displays a string describing its communication configuration. This string can tell you whether the daemon is using SSL security or not as follows:

- `[orbixd: Server "IT_daemon" is now available to the network]`
  `[Configuration SSL-TCP/10666/Orbix-XDR]`

  SSL-TCP means that the daemon is fully secure and will only accept secure connections.

- `[orbixd: Server "IT_daemon" is now available to the network]`
  `[Configuration TCP/10666/Orbix-XDR]`

  TCP means that the daemon is insecure and will not accept or initiate secure connections.

- `[orbixd: Server "IT_daemon" is now available to the network]`
  `[Configuration [SSL] TCP/10666/Orbix-XDR]`

  [SSL] TCP means that the daemon is capable of accepting secure and insecure connections.

A similar communications string is displayed for OrbixSSL servers depending on their security capabilities.

Configuring a Restricted Semi-Secure Daemon

The Orbix daemon is an Orbix server program that implements the IDL interface `IT_daemon`, as described in the *Orbix C++ Programmer's Reference*. A restricted semi-secure Orbix daemon accepts calls from insecure clients to a limited set of IDL operations on this interface. To specify which operations the daemon should accept, use the configuration variable `IT_DAEMON_UNRESTRICTED_METHODS`.

For example, to allow insecure clients to call only the operations `_IT_PING()`, `listServers()`, `listActiveServers()`, `getIIOPDetails()`, and `getImplementationDetails()`, add the following to `orbixssl.cfg`:

```
OrbixSSL {
   IT_DAEMON_UNRESTRICTED_METHODS = "_IT_PING,
      listServers, listActiveServers,
      getIIOPDetails, getImplementationDetails";
};
```

If you do not set the value of `IT_DAEMON_UNRESTRICTED_METHODS`, a restricted semi-secure daemon accepts calls to the operations `_IT_PING()`, `getIIOPDetails()`, and `getImplementationDetails()`. Consequently, a restricted semi-secure daemon allows an insecure client only to launch and locate servers.

Configuring the Orbix Daemon to Authenticate Clients

The configuration variable `IT_DAEMON_AUTHENTICATES_CLIENTS` determines whether the daemon enforces client authentication for all clients that attempt to connect to it. This includes Orbix utilities, such as `pingit` or `lsit`, and clients or servers that contact the daemon directly.

Set the value of the variable `IT_DAEMON_AUTHENTICATES_CLIENTS` in the `OrbixSSL` scope, for example:

```
OrbixSSL {
   IT_DAEMON_AUTHENTICATES_CLIENTS = "TRUE";
};
```

The default value for this variable depends on the current value of IT_DAEMON_POLICY, as shown in Table 3.1.

Daemon Policy	Daemon Authenticates Clients
SECURE_DAEMON	TRUE
RESTRICTED_SEMI_SECURE_DAEMON	TRUE
SEMI_SECURE_DAEMON	FALSE
INSECURE_DAEMON	FALSE

Table 3.1: *Default Values for Daemon Authentication of Clients*

If the Orbix daemon authenticates clients, programs that communicate with it must be able to supply certificates. This includes any applications that communicate with servers and the Orbix daemon utilities, such as putit, lsit, and mkdirit. To specify the certificate for these utilities, use the configuration scope Orbix.utilities in orbixssl.cfg:

```
Orbix {
   utilities {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "utilities certificate file name";
   };
};
```

Securing the Orbix Interface Repository

The Orbix Interface Repository is an Orbix server program that provides runtime information about IDL interfaces available in your system. Before running the Interface Repository, it is important to specify what type of communications it supports. To do this, set the configuration variable IT_ORBIX_BIN_SERVER_POLICY in the OrbixSSL configuration file as follows:

```
OrbixSSL {
   IT_ORBIX_BIN_SERVER_POLICY = "policy type";
};
```

Replace *policy type* with one of the following values:

`SECURE_SERVER`	The server supports only secure communications.
`SEMI_SECURE_SERVER`	The server supports both secure and insecure communications.
`INSECURE_SERVER`	The server supports only insecure communications.

If the Interface Repository server policy is `SECURE_SERVER` or `SEMI_SECURE_SERVER`, you must specify which certificate the Interface Repository server uses. To do this, use the `Orbix.utilities` scope in `orbixssl.cfg`, as for the Orbix utilities:

```
Orbix {
   utilities {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "utilities certificate file name";
   };
};
```

Currently, the Interface Repository server must use the same certificate as the Orbix utilities.

Securing the Orbix Services

Each of the Orbix services, such as OrbixNames or OrbixManager, has an associated configuration scope in the OrbixSSL configuration file. For example, OrbixManager uses the following scope:

```
OrbixManager {
   ...
};
```

Each of the services requires specific settings in the OrbixSSL configuration file and may require additional action to fully enable SSL. For details of how to run a service in secure system, consult the documentation associated with that service.

Configuring Ciphers

OrbixSSL allows you to specify which ciphers should be used for SSL encryption. Two configuration variables determine these ciphers:

`IT_CIPHERSUITES`	The value of this configuration variable determines the default list of ciphers that an OrbixSSL application uses. A space separated list of the possible values is given in order of preference.
`IT_ALLOWED_CIPHERSUITES`	This variable defines an additional list of ciphers that a program can specify using the API method `IT_SSL::specifyCipherSuite()`.

The possible values for these configuration cipher variables are:

```
SSLV3_RSA_WITH_RC4_128_SHA
SSLV3_RSA_WITH_RC4_128_MD5
SSLV3_RSA_WITH_3DES_EDE_CBC_SHA
SSLV3_RSA_WITH_DES_CBC_SHA

SSLV3_RSA_EXPORT_WITH_DES40_CBC_SHA
SSLV3_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSLV3_RSA_EXPORT_WITH_RC4_40_MD5
```

All of these values comprise the following components:

- Specification of the key exchange algorithm.

  RSA certificates are useful for key exchanges as RSA is a widely used public-key algorithm that can be used for either encryption or digital signing.

- Specification of the cipher to be used.

  Permitted ciphers are taken from the following list: `RC2`, `RC4`, `DES`, `3DES_EDE`, `CBC`.

- Specification of the hash algorithm to be used.

  Permitted hashes include `MD5` and `SHA`.

Only specific combinations of these options are available as listed, and one combination is referred to as a *cipher suite*.

OrbixSSL Session Caching Configuration

SSL session caching allows the reuse of information previously agreed between a client and server thus enabling faster subsequent reconnection. This can significantly increase server throughput if clients repeatedly reconnect to the server. The `IT_CACHE_OPTIONS` configuration variable offers the following options for controlling the use of SSL session caching in OrbixSSL applications:

`NO_SESSION_CACHING`	This variable means that OribxSSL clients and servers will not use SSL session caching. That is, they cannot accept re-used SSL session IDs proffered by SSL clients, and will not offer to resume previously established SSL sessions when contacting servers for a second or subsequent time.
`CACHE_CLIENT`	This variable means that OrbixSSL client programs will cache any sessions that are successfully established with servers. However, if subsequent attempts are made to reconnect to the server, then the initial session will be offered for reuse to the server. Whether the session is actually reused or not depends on the server's policy with respect to session caching. This applies to servers when they are acting as clients as well as pure clients.
`CACHE_SERVER`	This variable means that servers of OrbixSSL will cache any sessions that are successfully established with clients. If subsequent attempts are made to reconnect by clients, then the previously established session that is being offered by the client will be accepted provided that it has not been flushed from the OrbixSSL session cache.

It is important to note that for an OrbixSSL cache to be reused, SSL session caching has to be enabled for clients and servers. This applies to clients when they are receiving callbacks as well as to pure clients.

Providing IORs with SSL Information

When a non-Orbix client wants to obtain a server IOR from the Orbix daemon by means of IIOP, it is necessary to provide that IOR with SSL information. You can do this by means of the `putit` utility:

This is the full `putit` command syntax:

```
putit [-v] [-h <host>] [-per-client | -per-client-pid]
[ [-shared | -unshared] [-marker <marker>] ]
[ -j | -java [-classpath <classpath> | -addpath <path> ] ]
[ -oc <ORBclass> -os <ORBSingletonClass>] [ -jdk2]
| [-per-method [-method <method>] ]
[-port <iiop portnumber>]
[ -n <number of servers> ] [ -l ]
[ -ssl_secure | -ssl_semi_secure  [-ssl_client_auth]  [-
ssl_support_null_enc | -ssl_support_null_enc_only]  [-
ssl_support_null_auth | -ssl_support_null_auth_only] ]
<serverName> [ <commandLine> | -persistent ]
```

The ssl parameters are described in Table 3.2. To use them, you must specify either `-ssl_secure` or `-ssl_semi_secure` first.

putit Flag	Description
`-ssl_client_auth`	Indicates that the server authenticates clients.
`-ssl_support_null_enc`	This indicates that the NULL encryption SSL ciphersuites (which do not support confidentiality) are supported by the server.
`-ssl_support_null_enc_only`	This indicates that only the server supports the NULL encryption SSL ciphersuites..
`-ssl_secure`	This is the minimal flag needed to indicate that the server is SSL enabled. If this flag or –ssl_semi_secure are not supplied then the server is insecure and no SSL related data should be written to the IR. One of these two flags must be supplied before any other SSL flag is acceptable. An error should be presented to the user if they are not.

Table 3.2: *putit SSL Parameters*

putit Flag	Description
-ssl_semi_secure	This indicates a SEMI_SECURE server policy. If this flag or -ssl_secure are not supplied to putit then the policy is INSECURE and no SSL related stuff should be written to the IR. One of these two flags must be supplied before any other SSL flag is acceptable. An error should be presented to the user if they are not.
-ssl_support_null_auth	This flag indicates that the server support null authentication. OrbixSSL servers do not currently support this.
-ssl_support_null_auth_only	This flag indicates that the server support null authentication. OrbixSSL servers do not currently support this.

Table 3.2: *putit SSL Parameters*

Using the putit SSL Parameters

There are four groups of SSL parameters. If you want to use them, you must use one from Group 1, followed by one or none from each of the other three groups:

Group 1

```
-ssl_secure
-ssl_semi_secure
```

Group 2

```
-ssl_support_null_enc
-ssl_support_null_enc_only
```

Group 3

```
-ssl_support_null_auth
-ssl_support_null_auth_only
```

Group 4

```
-ssl_client_auth
```

As OrbixSSL supports per server process security policy settings, those settings specified by putit apply to all objects created by the server.

The most common use cases are:

```
Putit -ssl_secure demo/grid grid.exe
Putit -ssl_secure -ssl_client_auth demo/grid grid.exe
Putit -ssl_semi_secure demo/grid grid.exe
```

The following might be less common:

```
Putit -ssl_semi_secure -ssl_client_auth demo/grid grid.exe
```

4

Managing Certificates

SSL authentication uses X.509 certificates. This chapter explains how you can create X.509 certificates that identify your OrbixSSL applications.

An X.509 certificate binds a name to a public key value. The role of a certificate is to guarantee that the public key can be used to verify the identity contained in the X.509 certificate.

Authentication of a secure application depends on the integrity of the public key value in the application's certificate. If an impostor replaced the public key with its own public key, it could impersonate the true application and gain access to secure data.

To prevent this form of attack, all certificates must be signed by a *certification authority* (CA). A CA is a trusted node that confirms the integrity of the public key value in a certificate.

A CA signs a certificate by adding its digital signature to the certificate. A digital signature is a message encoded with the CA's private key. The CA's public key is made available to applications by distributing a certificate for the CA. Applications verify that certificates are validly signed by decoding the CA's digital signature with the CA's public key.

Most of the demonstration certificates supplied with OrbixSSL are signed by the CA `demo_ca_1`. This CA is completely insecure because anyone can access its private key. To secure your system, you must create new certificates signed by a trusted CA. This chapter describes the certificates required by an OrbixSSL application and shows you how to create those certificates.

Creating Certificates for an Application

To set up a fully secure OrbixSSL system, you must generate a full set of certificates for the secure components of your system, such as server, authenticated clients, the Orbix daemon, Orbix services, and so on. There are three steps required to do this:

1. Set up a CA that you can trust.
2. Use the CA to create signed certificates.
3. Deploy the signed certificates.

If a component of your application must prove its identity during SSL authentication, that component requires a certificate signed by your chosen CA. In a secure system, this always includes the Orbix daemon, the Orbix utilities, the Orbix services, and your server programs. If you use client authentication, your clients also require certificates.

Overview of the OrbixSSL Demonstration Certificates

The OrbixSSL `certificates` directory contains a set of demonstration certificates that enable you to run the OrbixSSL example applications. The certificates contained in the `certificates` directory are described in Table 4.1.

Certificate	Description
`ca/demo_ca_1` `ca/demo_ca_2`	Contains the certificates for the example CAs `demo_ca_1` and `demo_ca_2`. The CA list file, `demo_ca_list_1`, in the OrbixSSL `ca_lists` directory, includes the certificate for `demo_ca_1`. Programs that set the value of `IT_CA_LIST_FILE` to this file accept only certificates signed by `demo_ca_1`.

Table 4.1: *Demonstration Certificates Supplied with OrbixSSL*

Certificate	Description
demos/bad_guy demos/bank_customer_1 demos/bank_customer_2 demos/secure_bank_server demos/demo_client demos/demo_client_ca2 demos/demo_server demos/demo_server_ca2 ...	Example certificates used in the OrbixSSL demonstration programs. These programs are contained in the OrbixSSL demos directory. These certificates are signed by demo_ca_1, with the exception of those with _ca2 appended to the file name, which are signed by demo_ca_2. In the demonstration programs, the certificate bad_guy is used to represent a certificate for which the security has been compromised. This certificate is included in the certificate revocation list (CRL) crl.pem in the OrbixSSL crl directory. Refer to "Managing Certificate Revocation Lists" on page 68 for information about CRLs.
services/kdm_client services/kdm_server	Example certificates used by the server key distribution mechanism (KDM) server and the KDM utilities, for example putkdm. Refer to Chapter 5 for further details.
services/orbix services/orbix_events services/orbix_manager services/orbix_names services/orbix_ots services/orbix_trader	Example certificates used by Orbix services and standard Orbix executable files, such as the Orbix daemon, the Orbix utilities, and the Interface Repository server.

Table 4.1: *Demonstration Certificates Supplied with OrbixSSL*

The remainder of this chapter describes the steps involved in setting up a CA and signing certificates. As an example, it then shows you how to replace the demonstration certificates in the OrbixSSL certificates directory with your own, secure certificates.

Choosing a Certification Authority

A CA must be trusted to keep its private key secure. When setting up an OrbixSSL system, it is important to choose a suitable CA, make the CA certificate available to all applications, and then use the CA to sign certificates for your applications.

There are two types of CA available. A *commercial CA* is a company that signs certificates for many systems. A *private CA* is a trusted node that you set up and use to sign certificates for your system only.

Commercial Certification Authorities

There are several commercial CAs available. The mechanism for signing a certificate using a commercial CA depends on which CA you choose.

An advantage of commercial CAs is that they are often trusted by a large number of people. If your applications are designed to be available to systems external to your organization, use a commercial CA to sign your certificates. If your applications are for use within an internal network, a private CA might be appropriate.

Before choosing a CA, examine the certificate signing policies of some commercial CAs and, if your applications are designed to be available on an internal network only, review the potential costs of setting up a private CA.

Private Certification Authorities

If you wish to take responsibility for signing certificates for your system, set up a private CA. To set up a private CA, you require access to a software package that provides utilities for creating and signing certificates. Several packages of this type are available.

One software package that allows you to set up a private CA is SSLeay. SSLeay is an implementation of SSL developed by Eric Young of CryptSoft Pty. Ltd. The SSLeay package includes basic command line utilities for generating and signing certificates and these utilities are available with every installation of OrbixSSL.

To set up a private CA using OrbixSSL, do the following:

1. Choose a suitable host to act as CA.
2. Install OrbixSSL on the CA host.
3. Use the SSLeay utilities to create a certificate and private key for the CA.
4. Copy the CA certificate and private key to the required directories on the CA host.

When you complete these steps, you can use the SSLeay utilities to sign application certificates for your system.

Choosing a Host for a Private Certification Authority

Choosing a host is an important step in setting up a private CA. The level of security associated with the CA host determines the level of trust associated with certificates signed by the CA.

If you are setting up a CA for use in the development and testing of OrbixSSL applications, use any host that the application developers can access. However, when you create the CA certificate and private key, do not make the CA private key available on hosts where security-critical applications run.

If you are setting up a CA to sign certificates for applications that you are going to deploy, make the CA host as secure as possible. For example, take the following precautions to secure your CA:

- Do not connect the CA to a network.
- Restrict all access to the CA to a limited set of trusted users.
- Protect the CA from radio-frequency surveillance using an RF-shield.

When you choose a suitable host to act as the CA host, install OrbixSSL and use the SSLeay utilities to create the CA certificate and private key.

Creating a Self-Signed Certificate and Private Key

A self-signed certificate is a CA certificate in which the issuer and subject of the certificate are identical. It acts as the final authority in a certificate chain. To create a self-signed certificate and private key for your CA, use the SSLeay utility `ssleay` to run the command `req` as follows:

```
ssleay req -config ssleay_config_file -days 365
-out ca_cert_file.pem -new -x509
```

The utility `ssleay` is located in the OrbixSSL `bin` directory. Replace *ssleay_config_file* with the fully qualified name of the SSLeay configuration file `ssleay.cnf`. By default, OrbixSSL installs this file in the `config` directory of your Orbix installation.

The `req` command requests information that identifies the CA, including your organization name, organization address, and so on. This information comprises the CA's *distinguished name*.

This command also asks you to specify a pass phrase with which `req` will encrypt the private key for the CA. Note the pass phrase and guard it carefully.

The `req` command outputs two files. The first output file is `ca_cert_file.pem`, which contains the CA certificate in Privacy Enhanced Mail (PEM) format. The second output file is named `privkey.pem` (this default filename can be overridden using the `-keyout` option) and contains the encrypted private key for your CA in PEM format.

Note: The integrity of your private CA depends on the security of the pass phrase used to encrypt the CA's private key and the integrity of the CA's private key file. These should be available only to trusted users of the CA.

An Example of Creating a Self-Signed Certificate and Private Key

Consider the example of creating a certificate and private key for a CA to be used in signing certificates within the finance department of IONA Technologies.

If the ssleay.cnf file is installed in the default directory, run req as follows:

```
ssleay req -config ssleay config file -days 365
   -X509 -new -out demo_ca_1 -keyout demo_ca_1.pk
```

The req command begins by generating the private key for your CA. req prompts you to enter a pass phrase, which is used to encrypt the private key:

```
Generating a 512bit private key
..............+++++
............+++++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

The default ssleay.cnf file supplied with OrbixSSL configures the key length to 512 bits. This should be increased to 1024 bits for most live systems. When using 1024 bit keys, the initial SSL handshake is a number of times slower than for 512 bit keys, but the level of security obtained is very much greater.

The req command continues by requesting identification information for your CA:

```
Country Name (2 letter code) []: IE
State or Province Name (full name) []: Co. Dublin
Locality Name (eg, city) []: Dublin
Organization Name (eg, company) []: IONA Technologies
Organizational Unit Name (eg, section) []: Finance
Common Name (eg, YOUR name) []: Gordon Brown
Email Address []: gbrown@iona.com
```

The input for these identification fields should clearly identify the individual or group responsible for controlling the CA.

As a result of this operation, the req command outputs two files in the local directory. The CA certificate file is called demo_ca_1. The CA private key file is called demo_ca_1.pk.

Installing the Certificate and Private Key Files

To prepare the CA to sign certificates, do the following:

1. Ensure that the CA certificate file name matches the `certificate` value in the `ssleay.cnf` file.
2. On the CA host, copy the CA certificate file to the *root certificate directory*. To locate this directory, consult the `dir` entry in `ssleay.cnf`.
3. Ensure that the name of the CA private key file matches the `private_key` value in the `ssleay.cnf` file.
4. On the CA host, copy the private key file to the directory specified by the `private_key` entry in `ssleay.cnf`.

When you complete these steps, the CA is ready to sign application certificates.

Publishing a Certification Authority Certificate

To authenticate a certificate signed by a CA, an application requires access to the CA's own certificate.

To install a CA certificate on an OrbixSSL application host, add the CA certificate to the file specified by the `IT_CA_LIST_FILE` variable in the OrbixSSL configuration file on that host.

Certificates Signed by Multiple Certification Authorities

A CA certificate may be signed by another CA. For example, an application certificate may be signed by the CA for the finance department of IONA Technologies, which in turn is signed by a commercial CA.

This system of signing certificates is known as *certificate chaining*. An application can accept a signed certificate if the CA certificate for any CA in the signing chain is available in the certificate file in the local root certificate directory.

To limit the length of certificate chains accepted by your applications, add the following settings to your `orbixssl.cfg` file:

- `IT_DEFAULT_MAX_CHAIN_DEPTH`

  This configuration variable determines the default length of certificate chains which will be accepted by OrbixSSL clients and servers.

- `IT_MAX_ALLOWED_CHAIN_DEPTH`

  This configuration variable determines the maximum length of certificate chains which will be accepted by OrbixSSL for all OrbixSSL clients and servers that are using the security policy file.

Refer to "Configuring Server Authentication" on page 35 for more information about these configuration variables. Applications can also limit the maximum chain depth that they accept by using `IT_SSL::setMaxChainDepth()`.

Signing Application Certificates

If using a commercial CA, you must follow the CA's procedures for obtaining signed certificates.

If using a private CA, you can sign application certificates for use in your system. The process for generating a signed certificate is as follows:

1. An individual or group responsible for an application generates a *certificate signing request* (CSR).
2. The CSR is submitted to the CA for signing.
3. The CA signs and returns the new certificate.
4. The certificate file is copied to the OrbixSSL certificates directory on the host in which the application runs.

When this process is complete, the OrbixSSL application can use the signed certificate to prove its identity to other applications.

Generating a Certificate Signing Request

To generate a certificate signing request (CSR), run the SSLeay command `req` as follows:

```
ssleay req -config ssleay config file -days 365
   -new -out csr_file.pem
```

The `req` command requests information that identifies your application. This information includes the components of the distinguished name for your organization.

This command also asks you to specify a pass phrase which `req` will use to encrypt the private key for your application. Note the pass phrase and guard it carefully.

The `req` command outputs two files. The first output file is *csr_file*`.pem`, which contains the CSR for your application. The second output file is `privkey.pem` and contains the application private key.

The file *csr_file*`.pem` should now be transferred to the CA for signing.

An Example of Generating a Certificate Signing Request

Consider the example of generating a CSR for an OrbixSSL server application with server name `Bank`. Run `req` as follows:

```
ssleay req -config ssleay config file -days 365
    -new -out Bank-csr.pem
```

The `req` command begins by generating a private key for your application:

```
Generating a 512 bit private key
....+++++
.........+++++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

The private key is encrypted using a pass phrase that you supply.

The req command continues by requesting identification information for your certificate:

```
Country Name (2 letter code) []:IE
State or Province Name (full name) []: Co. Dublin
Locality Name (eg, city) []: Dublin
Organization Name (eg, company) []: IONA Technologies PLC
Organizational Unit Name (eg, section) []: Finance
Common Name (eg, YOUR name) []: CORBA Server:Bank
Email Address []: info@iona.com
```

Your organization should define a clear policy for the format and content of the identification fields added to your application certificates. Enter the requested fields according to this policy.

Signing a Certificate

To sign a certificate, run the ca command as follows:

```
ssleay ca -config ssleay config file -days 365
   -in csr_file.pem > certname.pem
```

The ca command displays the identification information contained in the CSR. It is critically important that you check that this information is correct with respect to the application for which the CSR was generated.

The ca command asks you if you wish to sign the application certificate. If you sign the certificate, the ca command outputs the certificate in PEM format to the file *certname*.pem. This *certname*.pem file is supplied to the originator of the certificate request.

To return the certificate to the person who issued the CSR, copy the file to disk and transfer this file from disk to a location accessible to that person. This certificate file can then be copied to the certificates directory on the application host. To locate this directory, consult the certs value in the local ssleay.cnf file.

Upon receipt of the certificate, the originator of the request concatenates the output file *certname*.pem with the private key file privkey.pem, produced by the req command. On UNIX, this is as follows:

```
cat certname.pem privkey.pem > cert_file
```

On Windows NT, it is:

```
copy certname.pem privkey.pem cert_file
```

The concatenated file now contains the application certificate and encrypted private key.

An Example of Signing a Certificate

Consider the example CSR described in "An Example of Generating a Certificate Signing Request" on page 60. Sign this certificate by running `ca` (on the CA host) as follows:

```
ssleay ca -config ssleay config file
-days 365 -in Bank-csr.pem -out Bank-cert.pem
```

The output from this command begins by requesting the pass phrase used to encode the CA private key:

```
Enter PEM pass phrase:
```

If you enter the correct pass phrase, `ca` displays the identification information contained in the CSR:

```
Check that the request matches the signature
Signature ok

The Subjects Distinguished Name is as follows
countryName               :PRINTABLE:'IE'
stateOrProvinceName       :PRINTABLE:'Co. Dublin'
localityName              :PRINTABLE:'Dublin'
organizationName          :PRINTABLE:'IONA Technologies PLC'
organizationalUnitName    :PRINTABLE:'Finance'
commonName                :PRINTABLE:'CORBA Server:Bank'
emailAddress              :IA5STRING:'info@iona.com'

Certificate is to be certified until Dec 12 14:11:12 1998
GMT (365 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Check that the identification information contained in the CSR is correct in accordance with the security policy of your organization. If the information is correct, sign the certificate and commit the operation when prompted.

This command produces a signed application certificate in the file `Bank-cert.pem`. Concatenate this file with the private key file produced by the `req` command. On UNIX, this is as follows:

```
cat Bank-cert.pem privkey.pem > Bank.pem
```

On Windows NT, it is:

```
copy Bank-cert.pem privkey.pem Bank.pem
```

Copy the file `Bank.pem` to the certificates directory on the host on which the `Bank` server runs.

Example of Creating Certificates with SSLeay

In Chapter 2, "Getting Started with OrbixSSL", the banking demonstration uses SSL security. However, this demonstration is not secure because it uses the OrbixSSL demonstration certificates. To make this demonstration secure, you must replace the demonstration certificates with certificates that are signed by a trusted CA.

To replace the demonstration certificates:

1. On the secure CA host, add the OrbixSSL `bin` directory to your path.
2. In any directory, create a new subdirectory, named `newcerts`, to store your new certificates.
3. In the Orbix `config` directory, edit the file `ssleay.cnf`. Change the value of the `dir` setting to the absolute path of your `newcerts` directory. For example:

   ```
   # ssleay.cnf
   dir = /iona/newcerts
   ...
   ```

4. Change directory to `newcerts`.
5. In the directory `newcerts`, create the following subdirectories to store your new versions of the demonstration certificates described in "Overview of the OrbixSSL Demonstration Certificates" on page 52:

   ```
   ca
   demos
   services
   ```

6. In directory `newcerts`, initialize two files called `serial` and `index.txt`.

   On UNIX:

   ```
   echo "01" > serial
   touch index.txt
   ```

   On Windows:

   ```
   echo 01 > serial
   echo 2>index.txt
   ```

7. Create a new self-signed CA and private key:

   ```
   ssleay req -x509 -new -config
   Orbix config dir/ssleay.cnf -days 365 -out ca/NewCA
   -keyout ca/NewCA.pk
   ```

   This command prompts you for a pass phrase for the CA private key and details of the CA distinguished name:

   ```
   Using configuration from /iona/ssleay.cnf
   Generating a 512 bit RSA private key
   ....+++++
   .+++++
   writing new private key to 'NewCA.pk'
   Enter PEM pass phrase:
   Verifying password - Enter PEM pass phrase:
   -----
   You are about to be asked to enter information
   that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a
   Distinguished Name or a DN.
   There are quite a few fields but you can leave
   some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) []:IE
   State or Province Name (full name) []:Co. Dublin
   Locality Name (eg, city) []:Dublin
   Organization Name (eg, company) []:IONA
   Technologies PLC
   Organizational Unit Name (eg, section) []:Finance
   Common Name (eg, YOUR name) []:Gordon Brown
   Email Address []:gbrown@iona.com
   ```

Note: The security of the CA depends on the security of the private key file and private key pass phrase used in this step.

8. Add the CA certificate to the file specified in the configuration variable `IT_CA_LIST_FILE` on each host that runs secure applications. Do not copy the CA private key to these hosts.
9. In the Orbix `config` directory, edit the file `ssleay.cnf`. Change the values of the `certificate` and `private_key` settings to the location of the files `NewCA` and `NewCA.pk` respectively. In addition, change the value of `new_certs_dir`, `database` and `serial`, if necessary. For example:

   ```
   # ssleay.cnf
   dir = /iona/newcerts
   certs = $dir
   certificate = $certs/ca/NewCA
   private_key = $certs/ca/NewCA.pk
   new_certs_dir = $certs
   database = $certs/index.txt
   serial = $certs/serial
   ...
   ```

   You are now ready to sign certificates with your new CA.
10. Create a new CSR for the `orbix` certificate, which is used by the Orbix daemon:

    ```
    ssleay req -new -config Orbix config dir/ssleay.cnf
    -days 365 -out ./orbix_csr.pem -keyout
    ./orbix_pk.pem
    ```

    This command prompts you for a pass phrase for the daemon's private key and information about the certificate distinguished name:

    ```
    Using configuration from /iona/ssleay.cnf
    Generating a 512 bit RSA private key
    .+++++
    .+++++
    writing new private key to './orbix_pk.pem'
    Enter PEM pass phrase:
    Verifying password - Enter PEM pass phrase:
    -----
    You are about to be asked to enter information
    that will be incorporated
    into your certificate request.
    ```

```
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave
some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IE
State or Province Name (full name) []:Co. Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) []:IONA
Technologies PLC
Organizational Unit Name (eg, section) []:Systems
Common Name (eg, YOUR name) []:Orbix
Email Address []:info@iona.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:IONA
```

Some of the entries in the CSR distinguished name must be the same as those used in the CA certificate. These entries depend on the CA policy section of the file `ssleay.cnf`. Refer to Appendix C for more information.

11. Sign the `orbix` CSR:

```
ssleay ca -config Orbix config dir/ssleay.cnf
-days 365 -in Orbix_csr.pem -out orbix.pem
```

This command requires the pass phrase for the private key associated with CA `NewCA`:

```
Using configuration from ../ssleay.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'IE'
stateOrProvinceName   :PRINTABLE:'Co. Dublin'
localityName          :PRINTABLE:'Dublin'
organizationName      :PRINTABLE:'IONA
Technologies PLC'
organizationalUnitName:PRINTABLE:'Systems'
```

```
commonName            :PRINTABLE:'Orbix Binary
Certificate'
emailAddress          :IA5STRING:'info@iona.com'
Certificate is to be certified until May 24
13:06:57 2000 GMT (365 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit?
[y/n]y
Write out database with 1 new entries
Data Base Updated
```

To sign the certificate successfully, you must enter the CA private key pass phrase used in step 7.

12. Concatenate the certificate and private key files. On UNIX, do the following:

    ```
    cat orbix.pem orbix_pk.pem > services/orbix
    ```

    On Windows, use the following command:

    ```
    copy orbix.pem orbix_pk.pem services\orbix
    ```

13. Copy the output file to each host that runs OrbixSSL applications.
14. If you change the certificate and private key associated with an Orbix executable or one of the Orbix services, it is important to run the OrbixSSL `update` command to register the pass phrase associated with the new private key.

    On UNIX, to register the pass phrase used in step 10 with the Orbix daemon, run `update` as follows:

    ```
    update orbixd passphrase 0
    ```

    On Windows, run this command as follows:

    ```
    update orbixd.exe passphrase 0
    ```

    Run this command on each host that runs OrbixSSL servers and uses the new Orbix daemon certificate.

15. Repeat steps 10 to 13, creating the other certificates described in "Overview of the OrbixSSL Demonstration Certificates" on page 52.

If you develop Java applications using OrbixSSL, you must convert the private key associated with each application certificate to the encrypted format required by OrbixSSL Java applications. To do this, use the utility `keyenc`, as described in the *OrbixSSL Java Programmer's and Administrator's Guide*.

Managing Certificate Revocation Lists

In some cases, CAs revoke existing certificates. For example, this can happen when a replacement certificate is issued to correct an error in a previous one, or when the security of the corresponding private key has been compromised.

A certificate revocation list (CRL) is a file, issued by a CA, that contains a list of certificates that are no longer valid, even though they have not yet expired. OrbixSSL supports CRLs. When checking the validity of a certificate, OrbixSSL implicitly checks the current CRL issued by the CA that signed the certificate. If the certificate has been revoked, OrbixSSL rejects it.

Obtaining Certificate Revocation Lists

How you obtain a CRL depends on which CAs your system uses. Commercial CAs have distinct procedures for the issuing of CRLs. If you use the SSLeay utilities to set up a private CA, your CA can issue CRLs using the SSLeay `ca` command with the `-gencrl` flag.

Each CRL is defined in a single file. Each file includes information identifying the issuer and a list of certificates that are no longer valid. The list contains the signature number of each revoked certificate and the date on which the certificate was revoked. A serial number is a unique identifier contained in every X.509 certificate.

The OrbixSSL `crl` directory contains an example CRL issued by the demonstration CA `demo_ca_1`. The demonstration application in the OrbixSSL `demos/crl` directory uses this CRL. The CRL contains the serial number of the demonstration certificate `bad_guy` and the application illustrates how OrbixSSL rejects this revoked certificate.

Using Certificate Revocation Lists

To instruct OrbixSSL to use CRLs, add the following setting to the OrbixSSL configuration file:

```
OrbixSSL {
   IT_CRL_ENABLED = "TRUE";
};
```

You must then specify the location of the CRLs in your file system. For example, the OrbixSSL demonstration CRLs are stored in the OrbixSSL `crl` directory. To specify this CRL location, add the following to the OrbixSSL configuration file:

```
OrbixSSL {
   IT_CRL_REPOSITORY =
      "OrbixSSL directory/crl";
};
```

Specifying the Update Period for CRLs

When you start an OrbixSSL program, OrbixSSL reads the CRLs from file and stores them in memory. By default, OrbixSSL does not read the information from the CRL files again.

Using the OrbixSSL configuration file, you can instruct OrbixSSL to refresh the CRL information stored in memory at regular intervals. To do this, use the configuration variable `IT_CRL_UPDATE_INTERVAL`. This variable takes a numeric value, measured in seconds.

For example, to instruct OrbixSSL to read the CRL information every sixty seconds, add the following to the configuration file:

```
OrbixSSL {
   IT_CRL_UPDATE_INTERVAL = "60";
};
```

5

Managing Pass Phrases

Every server secured with OrbixSSL has an associated certificate and private key. To access its private key, and use it to encrypt messages, a server must retrieve the associated pass phrase. This chapter shows you how to use OrbixSSL administration to supply pass phrases to servers.

As described in Chapter 2, "Getting Started with OrbixSSL", a programmer can use the OrbixSSL API to specify the pass phrase associated with the private key of any OrbixSSL program. For example, the programmer might request the pass phrase from the user and then supply this to OrbixSSL.

One problem with this approach is that many OrbixSSL servers are launched automatically by the Orbix daemon. Ideally, such servers would not require user intervention to obtain a pass phrase.

For this reason, OrbixSSL provides an administrative solution to the problem of providing private key pass phrases to servers. The OrbixSSL server *key distribution mechanism* (KDM) is a utility that enables you to supply pass phrases to servers at runtime.

Using a Central Repository for Servers

The OrbixSSL server key distribution mechanism (KDM) allows an administrator to maintain a database of servers and their associated private key pass phrases. When the Orbix daemon launches an OrbixSSL server, OrbixSSL uses the KDM to retrieve the pass phrase.

This section describes the KDM in detail. It explains how the KDM works, how you can maintain the database of server pass phrases, and how you can replace the KDM with other key distribution systems.

Overview of the Key Distribution Mechanism

The KDM is a single process that runs on each server host in your secure system. The KDM stores an encrypted repository of server names and their associated pass phrases. When a client connects to an OrbixSSL server, the Orbix daemon uses the KDM to provide the correct pass phrase to the server.

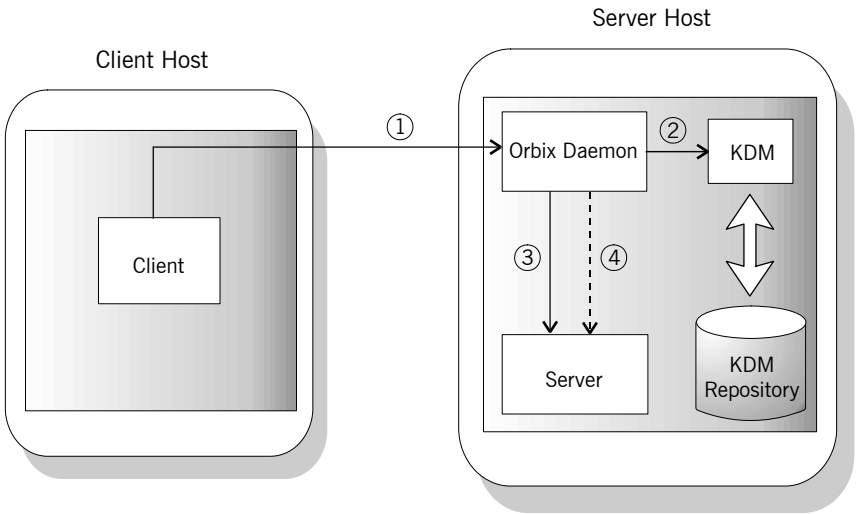

Figure 5.1: *Role of the Key Distribution Mechanism*

As shown in Figure 5.1, the following events happen when a client connects to a server that uses the KDM:

1. The client contacts the Orbix daemon on the server host.
2. The Orbix daemon requests security details for the server from the KDM.
3. The Orbix daemon launches the server.
4. The Orbix daemon sends the pass phrase to the server.

All communications between the Orbix daemon and the KDM use SSL security. To ensure that only the Orbix daemon has access to server pass phrases, the KDM always uses client authentication. If another process requests a pass phrase from the KDM, this authentication fails.

Communications between the Orbix daemon and the server is secure. This ensures that an external process cannot read the server pass phrase when the daemon transfers it to the server process.

Configuring the Key Distribution Mechanism

Before running the KDM, add the following settings to the OrbixSSL configuration file on your server host:

```
OrbixSSL {
   IT_KDM_ENABLED = "TRUE";
   IT_KDM_REPOSITORY = "repository directory";
   IT_KDM_SERVER_PORT = "server port";
};

KDM {
   server {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "KDM server cert file";
   };
   putkdm {
      IT_CERTIFICATE_FILE =
         OrbixSSL.IT_CERTIFICATE_PATH +
         "KDM client cert file";
   };
};
```

These configuration settings do the following:

`OrbixSSL.IT_KDM_ENABLED`	Enables the KDM. If the value of this variable is `TRUE`, all servers on the host use the KDM. Otherwise, no servers use the KDM.
`OrbixSSL.IT_KDM_REPOSITORY`	Specifies the absolute path of the directory in which the KDM stores its database of pass phrases. The user that runs the KDM should have full read and write access to this directory.
`OrbixSSL.IT_KDM_SERVER_PORT`	Specifies the port number on which the KDM listens for incoming communications. You can use any available port for this value.
`KDM.server.IT_CERTIFICATE_FILE`	Specifies the certificate file that the KDM server should use to prove its identity. If you are using the OrbixSSL demonstration certificates, set this variable to the file `services/kdm_server` in the OrbixSSL `certificates` directory.
`KDM.putkdm.IT_CERTIFICATE_FILE`	Specifies the certificate file that the KDM utility `putkdm` should use to prove its identity to the KDM server. If you are using the OrbixSSL demonstration certificates, set this variable to the file `services/kdm_client` in the OrbixSSL `certificates` directory.

Configuring Client Authentication

To ensure that the KDM supplies accepts pass phrases from the `putkdm` utility only and supplies pass phrases to the Orbix daemon only, the KDM server always uses client authentication. To configure client authentication, add the following setting to the OrbixSSL configuration file:

```
OrbixSSL {
   IT_KDM_CLIENT_COMMON_NAMES =
      "Orbix daemon CN, putkdm CN";
};
```

Replace *Orbix daemon CN* with the common name from the Orbix daemon certificate. Replace *putkdm CN* with the common name from the certificate used by `putkdm`. For example, if you are using the OrbixSSL demonstration certificates, the required values are as follows:

```
OrbixSSL {
   IT_KDM_CLIENT_COMMON_NAMES =
      "Orbix, KDM Client";
};
```

If you have replaced the demonstration certificates, as described in Chapter 4, these common names must be the same as those you entered when creating your Orbix daemon and `putkdm` certificates.

Configuring the Transfer of a Pass Phrase to a Server

When the Orbix daemon transfers a pass phrase to a server, it uses one of two communication methods: the server environment or an operating system pipe. Using either method, the pass phrase is encrypted and the transfer is secure.

By default, the Orbix daemon transfers the pass phrase in the server environment. To enable the use of operating system pipes, set the following value in the OrbixSSL configuration file:

```
OrbixSSL {
   IT_KDM_PIPES_ENABLED = "TRUE";
};
```

Running the Key Distribution Mechanism

The KDM is an OrbixSSL server that the Orbix daemon contacts using an IDL interface. The KDM server executable is called `kdm` and is located in the `bin` directory of your installation.

Although the KDM is an OrbixSSL server, it is unlike a normal server in one respect: you run the KDM before running the Orbix daemon. To run the KDM:

1. Add the OrbixSSL `bin` directory to your path.
2. Run the following command:

   ```
   kdm
   ```
3. The KDM requests the pass phrase associated with its certificate.

   If the KDM server uses the demonstration certificate `services/kdm_server`, enter `demopassword` as the pass phrase. If the KDM uses another certificate, enter the pass phrase for the associated private key.

Maintaining the Database

Before the Orbix daemon launches a server that uses the KDM, you must ensure that the server has a corresponding entry in the KDM database. To add an entry to the database, use the `putkdm` command:

```
putkdm server_name pass_phrase
```

The server name must match the name used to register the server in the Implementation Repository. The private key pass phrase must be at least six characters in length.

Verifying the Integrity of Server Executables

As an optional feature, the KDM allows you to ensure that the Orbix daemon only supplies pass phrases to the correct server executables. This prevents a malicious user from replacing a server executable with another program.

To support this feature, OrbixSSL provides a command-line utility, called `ccsit`, that takes a server executable file as input and outputs a *cryptographic checksum* based on the contents of the file. If the file is changed, the checksum becomes invalid.

Before running the `ccsit` utility, add the following settings to the OrbixSSL configuration file:

```
OrbixSSL {
   IT_CHECKSUMS_ENABLED = "TRUE";
   IT_CHECKSUMS_REPOSITORY = "checksums directory";
};
```

Replace *checksums directory* with a directory that can contain the checksums created by `ccsit`. In a production system, limit write access to your checksums directory to a single trusted user.

To register a checksum for a server, run the `ccsit` utility as follows:

```
ccsit server_file server_name
```

Replace *server_file* with the fully qualified name of the server executable. Replace *server_name* with the name used to register the server in the Implementation Repository.

Using the Key Distribution Mechanism

When the Orbix daemon launches a server and supplies its pass phrase using the KDM, it is not necessary for the server to call the API function `IT_SSL::setPrivateKeyPassword()`. If the server calls this function, it overrides the value supplied by the KDM. For information about how to write server code that uses the KDM when available, but supplies a password explicitly when the KDM is not available, refer to "Setting the Private Key Pass Phrase" on page 84.

Part III

OrbixSSL Programming

6

Programming with OrbixSSL

This chapter introduces the OrbixSSL application programming interface (API). It describes the main features of the API and how you can use it to customize SSL support in your applications.

The OrbixSSL C++ API is a set of C++ classes that provides you with access to the features of OrbixSSL when developing your applications. The API enables you to:

- Initialize OrbixSSL.
- Specify whether a program can make calls to secure servers, insecure servers, or both.
- Specify whether a program can accept calls from secure clients, insecure clients, or both.
- Read and write OrbixSSL configuration values.
- Read certificates and private key pass phrases from files.
- Configure the cipher suites used in SSL encryption.
- Customize certificate validation.

This chapter describes how to use the API to achieve some of these tasks. Part IV of this guide provides a complete reference for all the C++ classes in the OrbixSSL API. Refer to this part for more information about classes and methods introduced in this chapter.

Overview of the OrbixSSL API

The OrbixSSL API is defined in the header file `IT_SSL.h`, located in the `include` directory of your OrbixSSL installation. To access the API in an OrbixSSL program, include this file:

```
#include <IT_SSL.h>
...
```

A program that uses the API must be linked with the OrbixSSL dynamic library, as described on page 22

The class `IT_SSL`, defined in `IT_SSL.h`, provides the core features of the OrbixSSL API. A globally available instance of this class, named `OrbixSSL`, provides access to its methods. For example, to call the method `IT_SSL::init()`, use the `OrbixSSL` object as follows:

```
#include <IT_SSL.h>
...

OrbixSSL.init();
```

Many methods in the OrbixSSL API return a status value, of type `int`, that indicates whether the method is successful. For example, if an API call is successful, it returns `IT_SSL_SUCCESS`. Otherwise, it returns an error code.

For example, when calling the method `IT_SSL::init()`, you can check for success as follows:

```
#include <IT_SSL.h>
...

if (OrbixSSL.init() != IT_SSL_SUCCESS) {
   // Decide to continue or exit.
}
```

The available error codes are defined in `IT_SSL.h`. Each error code name begins with the string `IT_SSL_ERR_`.

Most OrbixSSL code is transferable from application to application. For example, customized certificate validation is often identical for a group of related servers. Where possible, keep OrbixSSL API code separate from your main application code. In addition, factor this code and place it in a shared library file. This enables you to upgrade all applications easily if you wish to avail of new features added to a future version of the OrbixSSL API.

Initializing OrbixSSL

The method `IT_SSL::init()` initializes SSL support in an OrbixSSL program. All OrbixSSL programs must call this method, for example:

```
#include <IT_SSL.h>
...

int main () {
   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   ...
}
```

All the initialization methods described in this section must be called before any remote communications take place using Orbix.

Initializing the Configuration Scope

After a call to `IT_SSL::init()`, OrbixSSL reads its configuration file to determine the required settings for the program. If your program is a client, OrbixSSL reads only the settings in scope `OrbixSSL`. However, you can instruct OrbixSSL to also read the values in a custom scope by calling the method `IT_SSL::initScope()`. For example, if the client custom scope is `Clients.BankClient`, call this method as follows:

```
#include <IT_SSL.h>
...

int main () {
   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   if (OrbixSSL.initScope("Clients.BankClient") !=
   IT_SSL_SUCCESS)
      return 1;
   ...
}
```

If your program is a server, OrbixSSL reads the values in scope `OrbixSSL` and then reads the values in the scope associated with the server. By default, the server scope is defined within `OrbixSSL.ServerNames`, as described in "Configuring Server Authentication" on page 35. However, if you call `IT_SSL::initScope()`, OrbixSSL uses your custom scope instead.

All servers that include OrbixSSL API calls should use a custom configuration scope and call `IT_SSL::initScope()`.

Setting the Private Key Pass Phrase

If an application has an associated certificate and private key, it must supply the private key pass phrase to OrbixSSL. This includes all OrbixSSL servers and all authenticated clients.

Setting the Pass Phrase for a Client

In an authenticated client, you supply the pass phrase by calling the method `IT_SSL::setPrivateKeyPassword()`. For example, if the pass phrase is `password`, call this method as follows:

```
#include <IT_SSL.h>
...

int main () {
   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   if (OrbixSSL.setPrivateKeyPassword("password")
   != IT_SSL_SUCCESS)
      return 1;

   ...
}
```

However, this code is not secure, because it is possible to examine the strings embedded in an executable file. For this reason, you should not hard code the pass phrase. Instead you should use some mechanism to retrieve the pass phrase and supply it as a parameter, of type `const char *`, to `IT_SSL::setPrivateKeyPassword()`. For example, the program could request the user to enter the password at runtime.

Setting the Pass Phrase for a Server

There are two ways to supply the pass phrase for a server private key: using the KDM, as described in Chapter 5, or using `IT_SSL::setPrivateKeyPassword()`. A call to `IT_SSL::setPrivateKeyPassword()` overrides a pass phrase provided by the KDM.

Typically, a server checks the availability of a pass phrase from the KDM before calling `IT_SSL::setPrivateKeyPassword()`. To do this, call `IT_SSL::hasPassword()` as follows:

```
#include <IT_SSL.h>
...

int main () {
   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   if (OrbixSSL.hasPassword() != IT_SSL_SUCCESS) {
      if (OrbixSSL.setPrivateKeyPassword
      ("password") != IT_SSL_SUCCESS)
         return 1;
   }

   ...
}
```

If the server is launched manually, or the KDM is not running, `IT_SSL::hasPassword()` returns a false value and the server calls `IT_SSL::setPrivateKeyPassword()`. Otherwise, the server accepts the pass phrase supplied by the KDM and continues processing.

Requesting the Pass Phrase from a User

There are many ways to request a pass phrase from a user. To make this task convenient, OrbixSSL provides a function, called `IT_SSL::getPassword()` that requests a password from the user and reads it into a variable of type `char *` in your program. This function must use a console to request user input, but is useful because it disables the console character echo when the user enters text.

For example, to use `IT_SSL::getPassword()` in a server application, you could do the following:

```
#include <IT_SSL.h>
...

int main () {
   char* password;

   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   if (OrbixSSL.hasPassword() != IT_SSL_SUCCESS) {
      password =
         OrbixSSL.getPassword("Enter password:");
      if (OrbixSSL.setPrivateKeyPassword
      (password) != IT_SSL_SUCCESS) {
         delete[] password;
         return 1;
      }
   }

   ...
}
```

Specifying which Certificate to Use

There are two ways to specify which certificate a program uses to identify itself: using the OrbixSSL configuration file, as described in "Configuring Server Authentication" on page 35, or using the method `IT_SSL::setSecurityName()`. Calling this method has the same effect as setting the value of the configuration variable `IT_CERTIFICATE_FILE` in the custom configuration scope for the program.

For example, if a program uses the certificate `Bank`, call `IT_SSL::setSecurityName()` as follows:

```
#include <IT_SSL.h>
...

int main () {
   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   if (OrbixSSL.hasPassword() != IT_SSL_SUCCESS) {
      if (OrbixSSL.setPrivateKeyPassword
      ("password") != IT_SSL_SUCCESS)
         return 1;
   }

   if (OrbixSSL.setSecurityName("Bank") !=
   IT_SSL_SUCCESS)
      return 1;

   ...
}
```

If you call `IT_SSL::setPrivateKeyPassword()`, you must call it before calling `IT_SSL::setSecurityName()`, as shown in this example.

If the variable `IT_CERTIFICATE_PATH` is set in the configuration file, OrbixSSL searches in that directory for the certificate specified by `IT_SSL::setSecurityName()`. In addition, a call to `IT_SSL::setSecurityName()` always overrides a corresponding `IT_CERTIFICATE_FILE` value set in the configuration file.

Configuring OrbixSSL Application Types

Orbix defines two general application types: clients, which call IDL operations on CORBA objects, and servers, which contain those objects. However, these roles are sometimes reversed. For example, in many applications, servers make callbacks to objects located in clients.

In OrbixSSL, it is important to be aware that all programs can potentially act as clients and servers. For each program, OrbixSSL allows you to specify an *invocation policy*. This policy determines whether the program uses SSL when connecting to a server and whether it uses SSL when it accepts connection attempts from clients. An invocation policy is a combination of these two independent settings.

Possible settings for making connections are:

- Only make connections to servers using SSL.
- Only make connections to servers without using SSL.
- Make connections using SSL, but allow insecure connections to specified interfaces or servers.
- Make connections to servers using SSL or without using SSL, as required.

Possible setting for accepting connection attempts are:

- Accept only connection attempts that use SSL.
- Accept only connection attempts that do not use SSL.
- Accept either connection attempts that use SSL or attempts that do not. In this case, the client determines whether to use SSL.

This chapter describes how you set the invocation policy for an OrbixSSL program and how programs interact based on their policy settings.

Choosing Invocation Policies

The most secure OrbixSSL system architecture is one in which all applications connect using SSL. If SSL security is available to all applications in your system, you should ensure that each application has a fully secure policy for making and accepting connections. This is the default setting for an OrbixSSL application.

The least secure system architecture is one in which no applications use SSL security. It is unlikely that your OrbixSSL system will consist of only insecure applications, but it may be acceptable for some of your applications to interact without using SSL.

For example, in a secure system it is sometimes necessary to accommodate existing applications that cannot communicate over SSL. In this case, your system could consist of a combination of fully secure applications, fully insecure applications, and applications that combine secure communications with insecure communications.

Setting an Invocation Policy

To specify the invocation policy for a program, call the method `IT_SSL::setInvocationPolicy()`. This method is defined as follows:

```
class IT_SSL {
public:
   virtual int setInvocationPolicy(int pol);
   ...
};
```

The parameter `pol` specifies which invocation policy the application should use. This integer is a bitwise `OR` combination of the values defined in the enumerated type `IT_SSLInvocationOptions`. These values are:

```
IT_SECURE_ACCEPT
IT_INSECURE_ACCEPT
IT_INSECURE_CONNECT
IT_SECURE_CONNECT
IT_SPECIFIED_INSECURE_CONNECT
IT_SPECIFIED_SECURE_CONNECT
```

The values IT_SECURE_ACCEPT and IT_INSECURE_ACCEPT determine how the program behaves when accepting connection attempts from clients. The other values determine how the program behaves when establishing connections to servers.

For example, to specify that a program should be able to accept both secure and insecure connection attempts, but should establish only secure connections with servers, do the following:

```
#include <IT_SSL.h>
...

int main () {
   if (OrbixSSL.init() != IT_SSL_SUCCESS)
      return 1;

   if (OrbixSSL.setInvocationPolicy(
   IT_SECURE_ACCEPT | IT_INSECURE_ACCEPT |
   IT_SECURE_CONNECT) != IT_SSL_SUCCESS)
      return 1;

   ...
}
```

How Invocation Policies Affect OrbixSSL Communications

Table 6.1 describes the set of client and target invocation policies that communicate successfully and indicates the type of communications associated with each case. The first column of this table indicates the client policy of the application that calls an IDL operation, the second column indicates the target policy of the application that receives this operation call.

Client Policy	Target Policy	Resulting Communications
IT_SECURE_CONNECT	IT_SECURE_ACCEPT	Secure.
IT_SECURE_CONNECT	IT_SECURE_ACCEPT IT_INSECURE_ACCEPT	Secure.
IT_SECURE_CONNECT	IT_INSECURE_ACCEPT	N/A.
IT_SPECIFIED_INSECURE_CONNECT	IT_SECURE_ACCEPT	Secure.
IT_SPECIFIED_INSECURE_CONNECT	IT_SECURE_ACCEPT IT_INSECURE_ACCEPT	Secure unless explicitly specified by client.
IT_SPECIFIED_INSECURE_CONNECT	IT_INSECURE_ACCEPT	Insecure only if explicitly specified by client; otherwise N/A.
IT_SPECIFIED_SECURE_CONNECT	IT_SECURE_ACCEPT	Secure only if explicitly specified by client; otherwise N/A.
IT_SPECIFIED_SECURE_CONNECT	IT_SECURE_ACCEPT IT_INSECURE_ACCEPT	Insecure unless explicitly specified by client; otherwise secure.
IT_SPECIFIED_SECURE_CONNECT	IT_INSECURE_ACCEPT	Insecure unless explicitly specified by client; otherwise N/A.
IT_INSECURE_CONNECT	IT_SECURE_ACCEPT	N/A.
IT_INSECURE_CONNECT	IT_SECURE_ACCEPT IT_INSECURE_ACCEPT	Insecure.
IT_INSECURE_CONNECT	IT_INSECURE_ACCEPT	Insecure.

Table 6.1: *How Programs with Different Invocation Policies Communicate*

Limitations Imposed by Incompatible Invocation Policies

Because of incompatible security capabilities, limitations exist on the interaction between some programs. For example, an insecure client cannot communicate with a fully secure server. Such instances have the value N/A in the communications column of Table 6.1 on page 91.

If a secure client attempts to communicate securely with an insecure target, for example by resolving a reference to an object in the target program, the client application receives an SSL_FAILURE exception or a COMM_FAILURE exception.

If an insecure client attempts to communicate with a fully secure target, the client receives a NO_PERMISSION exception, or a COMM_FAILURE exception.

Specifying Exceptions to an Invocation Policy

If your program has a client policy of IT_SPECIFIED_INSECURE_CONNECT, it can make insecure calls only to specified interfaces or servers. To specify the list of interfaces, the client must call the function IT_SSL::specifySecurityForInterfaces(). To specify the list of servers, the client must call IT_SSL::specifySecurityForServers().

Similarly, if your program has a client policy of IT_SPECIFIED_SECURE_CONNECT, it can make secure calls only to specified interfaces or servers. The functions IT_SSL::specifySecurityForInterfaces() and IT_SSL::specifySecurityForServers() also allow a client to specify these interfaces and servers. Refer to "Class IT_SSL" on page 149 for more information.

It is important to limit use of IT_SPECIFIED_INSECURE_CONNECT or IT_SPECIFIED_SECURE_CONNECT, because it is not difficult for a program to change the server name or interface that it uses. If a client passes sensitive data to a server, it should always use IT_SECURE_CONNECT. If a client does not pass sensitive data to a server, but the server passes sensitive data to the client, the server should force the client to connect using SSL.

Configuring OrbixSSL

The OrbixSSL configuration file, described in Chapter 3, specifies the security policy for each of your applications. An OrbixSSL program can override some of the configuration values in this files using the OrbixSSL API.

Table 6.2 describes the configuration variables that you can read or write using the OrbixSSL API and the associated API functions for these variables. Refer to "Class IT_SSL" on page 149 for more information about each function.

Configuration Variable	Associated Functions
IT_AUTHENTICATE_CLIENTS_BY_DEFAULT	IT_SSL::getClientAuthentication() IT_SSL::setClientAuthentication()
IT_CA_LIST_FILE	IT_SSL::getCAListFile()
IT_CACHE_OPTIONS	IT_SSL::getCacheOptions() IT_SSL::setCacheOptions()
IT_CERTIFICATE_FILE	IT_SSL::getSecurityName() IT_SSL::setSecurityName()
IT_CIPHERSUITES	IT_SSL::getNegotiatedCipherSuite() IT_SSL::specifyCipherSuites()
IT_CRL_REPOSITORY	IT_SSL::getCRLDir()
IT_DEFAULT_MAX_CHAIN_DEPTH	IT_SSL::getMaxChainDepth() IT_SSL::setMaxChainDepth()
IT_INSECURE_REMOTE_INTERFACES	IT_SSL::specifySecurityForInterfaces()
IT_INVOCATION_POLICY	IT_SSL::getInvocationPolicy() IT_SSL::setInvocationPolicy()
IT_SECURE_REMOTE_INTERFACES	IT_SSL::specifySecurityForInterfaces()

Table 6.2: *Read and Write Functions for OrbixSSL Configuration Variables*

Logging OrbixSSL Trace Information

Security trace information can be added to OrbixSSL programs by using the following configurtion variables that can only be set in the environment:

`IT_SSL_TRACE_LEVEL`	When this variable is set to `1`, programs affected by the variable output diagnostic information about the peer certificate chain during SSL authentication.
`IT_SSL_TRACEFILE`	If you require additional trace information, set this variable to the full path name of the file that you want this information to be written to. This file should be associated with only one running process.

The variable `IT_SSL_TRACEFILE` has a large effect on performance. Set this variable only if diagnostic information is required. Once set, it causes the trace file to grow quickly.

7

Validating Certificates

During SSL authentication, OrbixSSL checks the validity of an application's certificate. This chapter describes how OrbixSSL validates a certificate and how you can use the OrbixSSL API to introduce additional validation to your applications.

The OrbixSSL API allows you to define functions that implement custom validation of certificates. During SSL authentication, OrbixSSL validates a certificate and then passes it to your custom validation function for examination. This functionality is very important in systems that log information about certificates or have application-specific requirements for the contents of each certificate.

An X.509 certificate contains information about the supplier and the CA that issued the certificate. The structure of a certificate is specified in Abstract Syntax Notation One (ASN.1), a standard syntax for describing messages that can be sent or received on a network.

OrbixSSL provides a set of C++ classes that enable you to extract the information from a certificate without a detailed understanding of the corresponding ASN.1 definitions. When writing your certificate validation functions, you use these classes to examine the certificate contents.

Overview of Certificate Validation

Figure 7.1 shows a server sending its certificate to a client during an SSL handshake. OrbixSSL code at the server reads the certificate from file and transmits it as part of the handshake. OrbixSSL code at the client reads the certificate from the network, checks the validity of its contents, and either accepts or rejects the certificate.

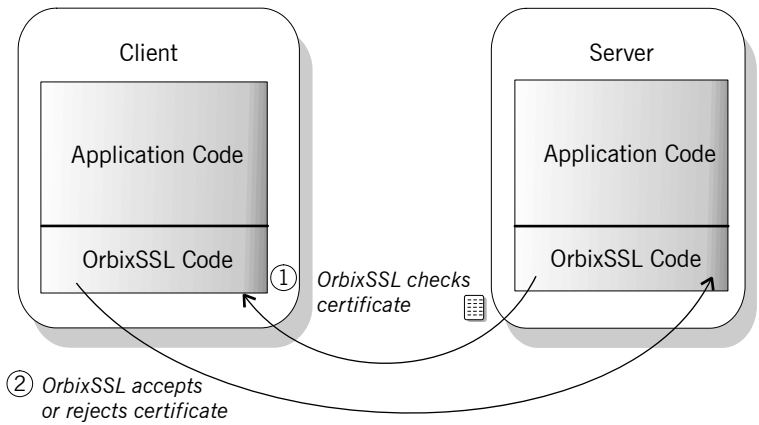

Figure 7.1: *OrbixSSL Validating a Certificate*

The default certificate validation in OrbixSSL checks:

- That the certificate is a validly constructed X.509 certificate.
- That the signature is correct for the certificate.
- That the certificate chain is validly constructed, consisting of the peer certificate plus valid issuer certificates up to the maximum allowed chain depth.
- That the certificate has not been revoked by the issuer. This check takes place only if enabled by OrbixSSL configuration.

For some applications, it is necessary to introduce additional validation. For example, your client programs might check that each server uses a specific, expected certificate.

Using OrbixSSL, you can register a function that carries out extra validation on certificates. When OrbixSSL receives a certificate, it validates it in the usual way and then passes it to your custom validation function, with an error code indicating whether the default validation succeeded or failed. You can then use the OrbixSSL API to examine the full contents of the certificate and instruct OrbixSSL whether to accept or reject it.

Figure 7.2 illustrates how a custom validation function interacts with OrbixSSL code during an SSL handshake.

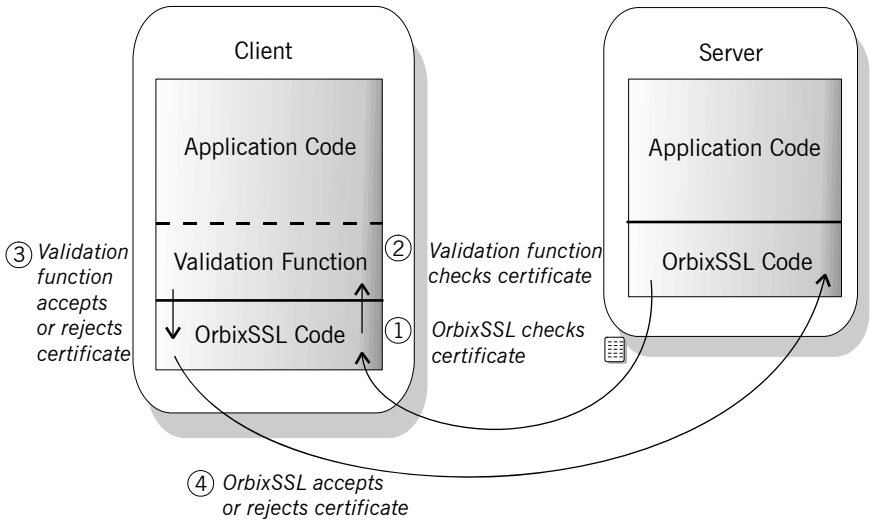

Figure 7.2: *Using a Custom Validation Function*

Introducing Additional Validation

OrbixSSL allows you to register two functions for additional certificate validation: one for validating certificates received from servers, and another for validating certificates received from clients. These two types of certificate often require different validation at the application level.

To register a function for server certificate validation, call the function `IT_SSL::setValidateServerCallback()` on the `OrbixSSL` object. This function is defined as:

```
class IT_SSL {
public:
   virtual void setValidateServerCallback(
   IT_ValidateX509CertCB cb);
   ...
};
```

To register a function for server certificate validation, call the function `IT_SSL::setValidateClientCallback()` on the `OrbixSSL` object. This function is defined as:

```
class IT_SSL {
public:
   virtual void setValidateClientCallback(
   IT_ValidateX509CertCB cb);
   ...
};
```

A certificate validation function must have the following signature:

```
int function_name(IT_CertValidity ok,
                  IT_X509CertChain& peerChain);
```

When OrbixSSL calls your validation function, it supplies two parameters. The first parameter is of type `IT_CertValidity`. This parameter indicates whether the default certificate validation succeeded or failed. It takes one of the following values:

`IT_SSL_VALID_YES`	Indicates that the default certificate validation succeeded.
`IT_SSL_VALID_NO`	Indicates the default certificate validation failed, and the application must reject the certificate.
`IT_SSL_VALID_NO_APP_DECISION`	Indicates the default certificate validation failed, but the application can chose whether to accept or reject the certificate.

The second parameter is of type `IT_X509CertChain&`. This parameter provides access to the full certificate chain. "Examining the Contents of a Certificate" on page 100 describes how you use this parameter to examine the contents of the peer certificate.

Your custom validation function must return an `int` value. If this return value is `IT_SSL_VALID_NO`, OrbixSSL rejects the certificate. If the return value is `IT_SSL_VALID_YES`, OrbixSSL accepts the certificate. The return value has no effect if the first parameter passed to the function is `IT_SSL_VALID_NO`.

Examining the Contents of a Certificate

The role of a certificate is to associate an identity with a public key value. In more detail, a certificate includes:

- X.509 version information.
- A *serial number* that uniquely identifies the certificate.
- A *common name* that identifies the supplier.
- The *public key* associated with the common name.
- The name of the user who created the certificate, which is known as the *subject name*.
- Information about the certificate issuer.
- The signature of the issuer.
- Information about the algorithm used to sign the certificate.
- Some optional X.509 version three extensions. For example, an extension exists that distinguishes between CA certificates and end-entity certificates.

The second parameter to your custom validation function, of type `IT_X509CertChain&`, provides access to the certificate chain received by OrbixSSL. Class `IT_X509CertChain` is defined in `IT_SSL.h` as follows:

```
class IT_X509CertChain {
public:
   ...

   virtual unsigned int numCerts();
   virtual int getCert(unsigned int pos,
   IT_X509Cert& ret);
   virtual int getErrorInfo(IT_CertError& ret);
   virtual int getCurrentCert(IT_X509Cert& ret);
   virtual int getCurrentDepth();
};
```

The function `IT_X509CertChain::numCerts()` indicates the number of certificates in the certificate chain. For example, if the peer certificate is signed by a single, self-signed CA, this function returns a value of two. The function

`IT_X509CertChain::getCert()` returns a certificate from a particular position in the chain, starting at one. Repeated calls to `IT_X509CertChain::getCurrentCert()` iterate through the certificate chain.

When you call `IT_X509CertChain::getCert()` or `IT_X509CertChain::getCurrentCert()`, you receive an object of type `IT_X509Cert` that represents the required certificate. Class `IT_X509Cert` is defined in `IT_SSL.h` as follows:

```
class IT_X509Cert {
public:
   ...

   virtual int getVersion(unsigned int& ver);
   virtual int getSerialNumber(IT_IntegerData& i);
   virtual int getIssuer(IT_AVAList& l);
   virtual int getSubject(IT_AVAList& l);
   virtual int getExtensions(IT_ExtensionList& e);
   virtual int getNotBefore(IT_UTCTime& t);
   virtual int getNotAfter(IT_UTCTime& t);
   virtual int getSignatureAlgorithm(IT_OID& oid);
   virtual int length(IT_Format f);
   virtual int convert(char *buf, IT_Format f);
};
```

Part IV of this guide provides detailed information about the member functions of this class. These member functions return C++ types corresponding to the ASN.1 types of the certificate contents. For example, `IT_X509Cert::getVersion()` returns an unsigned integer value that indicates the X.509 version number in use. In accordance with the X.509 standard, a value of `0` corresponds to version one, `1` corresponds to version two, and `2` corresponds to version three.

Working with Distinguished Names

An X.509 certificate uses ASN.1 *distinguished name* structures to store information about the certificate issuer and subject. A distinguished name consists of a series of attribute value assertions (AVAs). Each AVA associates a value with a field from the distinguished name.

For example, the distinguished name for a certificate issuer could be represented in string format as follows:

```
/C=IE/ST=Co. Dublin/L=Dublin/O=IONA/OU=PD/CN=IONA
```

In this example, AVAs are separated by the `/` character. The first field in the distinguished name is `C`, representing the country of the issuer, and the corresponding value is the country code `IE`. This example distinguished name contains six AVAs.

When you call the functions `IT_X509Cert::getIssuer()` or `IT_X509Cert::getSubject()`, OrbixSSL returns the corresponding distinguished name as an object of type `IT_AVAList`. Class `IT_AVAList` is defined as follows:

```
class IT_AVAList {
public:
     virtual int convert(char* buf, IT_Format f);
     virtual int getAVA(unsigned int pos,
        IT_AVA& retAVA);
     virtual int getAVAByOID(IT_OID oid,
        IT_AVA& retAVA);
     virtual int getAVAByOIDTag(IT_OID_Tag oid,
        IT_AVA& retAVA);
     virtual unsigned int getNumAVAs();
     virtual int length(IT_Format f);
};
```

To retrieve a particular AVA from a distinguished name, use the `IT_AVAList` object that represents the name. Each AVA in a distinguished name has an associated ASN.1 object identifier (OID).

You can retrieve a particular field using any one of the following three functions:

`getAVA()`	Returns an AVA from a particular position in the distinguished name. To use this, you must understand the contents of the distinguished name that you receive.
`getAVAByOID()`	Returns an AVA associated with a particular OID. To use this, you must know the OID of the field you require.
`getAVAByOIDTag()`	Returns an AVA associated with a particular OID, but uses the tags defined in type `IT_OIDTag` instead of the actual OID. Using this method, you can access some of the commonly required distinguished name fields without knowing the corresponding OIDs or positions in the distinguished name.

Each of these functions returns an object of type `IT_AVA`. You can then use the functions in class `IT_AVA` to convert the AVA to a number of different formats, such as string format or DER format, and retrieve the associated OID. Refer to class `IT_AVA` on page 113 for more details.

Working with X.509 Extensions

Some X.509 version three certificates include extensions. These extensions can contain several different types of information. If you wish to extract information from the extensions included in a certificate, call `IT_X509Cert::getExtensions()` on the certificate object.

This function returns an object of type of type `IT_ExtensionList`. This class is defined as follows:

```
class IT_ExtensionList {
public:
   virtual int convert(char* buf, IT_Format f);
   virtual unsigned int getNumExtensions();
   virtual int getExtension(int pos,
      IT_Extension& retExt);
   virtual int getExtensionByOID(IT_OID oid);
```

```
        virtual int getExtensionByOIDTag(
           IT_OID_Tag oid);
        virtual int length(IT_Format f);
     };
```

Like AVAs, each possible extension is associated with an ASN.1 OID. Given a list of extensions, you can retrieve the extension you require using any one of the following three functions:

`getExtension()`	Returns an extension from a particular position in the extension list. To use this, you must understand the list of extensions included in the certificate.
`getExtensionByOID()`	Returns an extension associated with a particular OID. To use this, you must know the OID of the extension you require. Use this function only when the extension you require is not available from `getExtensionByOIDTag()`.
`getExtensionByOIDTag()`	Returns an extension associated with a particular OID, but uses the tags defined in type `IT_OIDTag` instead of the actual OID. Using this method, you can access some extensions without knowing the corresponding OIDs or positions in the extension list.

Each of these functions returns an object of type `IT_Extension`. You can then use the functions in class `IT_Extension` to convert the extension information to a number of different formats, such as string format or DER format, and retrieve the associated OID. Refer to class `IT_Extension` on page 129 for more details.

Example of a Certificate Validation Function

This section describes a simple validation function, registered in an OrbixSSL client, that prints the common name (CN) of a server to which the client connects. The code for this function is as follows:

```
int example_val_func(int ok, IT_X509CertChain& PeerCertChain) {
   int ret = ok;

   // Checks only the peer certificate. The current chain
   // depth is zero for this certificate.
   if(ok==IT_SSL_VALID_YES && PeerCertChain.getCurrentDepth()==0){
      char *buf = NULL;
      int len;
      IT_X509Cert x;
      IT_AVAList aval;
      IT_AVA ava;

      PeerCertChain.getCurrentCert(x);

      // Get details about the subject.
1     x.getSubject(aval);

      // Get the common name from the subject details.
2     aval.getAVAByOIDTag(IT_OIDT_commonName, ava);

3     if ((len = ava.length(IT_FMT_STRING)) <= 0)
            return 0;
      buf = new char[len];
4     if (buf && (ava.convert(buf, IT_FMT_STRING)==
                  IT_SSL_SUCCESS))
         cout << "The common name (CN) of the servers cert
         is:" << buf << endl;
      delete[] buf;
   }

   return ret;
}
```

You can register this function using the following call:

```
OrbixSSL.setValidateServerCertCallback(example_val_func);
```

The code is explained as follows:

1. The `IT_X509Cert::getSubject()` function returns the subject's distinguished name field from an X.509 certificate.
2. A call to `IT_AVAList::getAVAByOIDTag()` extracts the common name field from the subject name. The common name field is the name of the entity for whom the certificate was issued.
3. A call `IT_AVA::length()` gets the amount of memory required to store the common name.
4. A call `IT_AVA::convert()` returns the common name in the supplied buffer.

The validation function is called once for each certificate in the peer certificate chain. However, you can restrict the function to just examining the peer certificate (that is, the server's actual certificate) by checking if the current depth in the chain is zero, as shown in this example.

Using Certificate Revocation Lists

As described in "Managing Certificate Revocation Lists" on page 68, you can configure OrbixSSL to include Certificate Revocation List (CRL) checking when it validates certificates. This means that each time OrbixSSL checks the validity of a certificate, it examines the CRL associated with the certificate CA to ensure that the issuer has not revoked the certificate.

Using the OrbixSSL API, you can also access CRLs directly. For example, you can use the API to check the contents of CRLs located in a directory other than the CRL directory that you have configured OrbixSSL to use.

OrbixSSL represents a group of CRLs as an object of type `IT_CRL_List`. This class is defined as follows:

```
class IT_CRL_List {
public:
   ...

   virtual int add(const IT_X509_CRL_Info& aCRL);
   virtual int remove
      (const IT_AVAList& lstIssuer);
```

```
    virtual int find(const IT_AVAList& lstIssuer,
       CORBA(Boolean)& bFound,
       IT_X509_CRL_Info& aCRL) const;
    virtual int openCRLFiles(const char* szCRLDir);
    virtual int PollForUpdates(
       CORBA(Boolean)& bUpdated);
};
```

To examine a list of CRLs:

1. Create an object of type `IT_CRL_List`, for example:

   ```
   IT_CRL_List extraCRLs = new IT_CRL_List();
   ```

2. On this object, call the function `IT_CRL_List::openCRLFiles()`, specifying the location in which your CRLs are stored:

   ```
   extraCRLs.openCRLFiles("/local/crl");
   ```

3. Call the function `IT_CRL_List::find()` to access the CRL associated with a particular CA. This function returns the CRL as an object of type `IT_X509_CRL_Info`.

In addition to accessing a CRL from the list, you can use the functions `IT_CRL_List::add()` and `IT_CRL_List::remove()` to modify the list contents. However, these functions affect only the copy of the list stored in memory and not the files associated with the CRLs.

Examining the Contents of a Certificate Revocation List

OrbixSSL represents each CRL in the CRL list as an object of type `IT_X509_CRL_Info`. This class is defined as follows:

```
class IT_X509_CRL_Info {
public:
   ...

   virtual int getSignatureAlgorithm(IT_OID& oid)
      const;
   virtual int getVersion(unsigned int& nVer)
      const;
   virtual int getIssuer
      (IT_AVAList& lstIssuer) const;
   virtual int getLastUpdate(IT_UTCTime& t) const;
   virtual int getNextUpdate(IT_UTCTime& t) const;
```

```
        virtual int getRevokedCerts
           (IT_X509_RevokedList& r) const;
        virtual int find(const IT_IntegerData&
           nSerialNum, CORBA(Boolean)& bFound,
           IT_X509_Revoked& r) const;
        virtual int getExtensions(IT_ExtensionList& e)
           const;
        virtual int fromDER(const char** pData);
        virtual int openFile(const char* file,
           IT_Format fmt);
     };
```

This class provides information about the CRL issuer, the CRL version number, when the CRL was last updated, and when the next update is expected. It also allows you to access the contents of the CRL.

To retrieve information about the revoked certificates, call the function `IT_X509_CRL_Info::getRevokedCerts()`. This function returns the revoked certificate information as an object of type `IT_X509_Revoked_List`:

```
class IT_X509_RevokedList {
public:
   ...

   virtual unsigned int getCount() const;
   virtual int getRevoked(int nPos,
      IT_X509_Revoked& r) const;
};
```

Given an object of this type, call the function `IT_X509_Revoked_List::getCount()` to determine the number of revoked certificates in the list and use `IT_X509_Revoked_List::getRevoked()` to access information about an individual revoked certificate.

Function `IT_X509_Revoked_List::getRevoked()` returns the revoked certificate information as an object of type `IT_X509_Revoked`:

```
class IT_X509_Revoked {
public:
   ...

   virtual int getSerialNumber
      (IT_IntegerData& serialNum) const;
   virtual int getRevocationDate(IT_UTCTime& t)
      const;
   virtual int getExtensions
      (IT_ExtensionList& e) const;
   virtual int getSequence(int& n) const;
};
```

In a CRL, each certificate is identified by its serial number. The function `IT_X509_Revoked::getSerialNumber()` returns this identifier, which you can check against the serial number of an `IT_X509Cert` object. Call `IT_X509::getSerialNumber()` to retrieve the serial number from an object of this type.

For more information about the OrbixSSL CRL support classes, refer to Part IV of this guide.

Part IV

OrbixSSL C++ Reference

Class IT_AVA

Synopsis As described in Chapter 7, "Validating Certificates", an `IT_AVAList` is an abstraction of a distinguished name from a certificate. An `IT_AVAList` consists of a number of `IT_AVA` objects.

Individual `IT_AVA` objects represent an element of the distinguished name such as the common name field (CN) or organization unit (OU). You can retrieve a desired `IT_AVA` object can using the `IT_AVAList` class.

`IT_AVA` objects can be converted to a number of different forms such as string format or DER format. For more information on these formats, refer to `IT_AVAList::convert()` on page 117 and `IT_AVAList::length()` on page 121.

C++

```
class IT_AVA {
public:
        virtual int convert(char* buf, IT_Format f);
        virtual int length(IT_Format f);
        virtual int OID(IT_OID& retOID);
        virtual int getSet();
};
```

IT_AVA::convert()

Synopsis

```
virtual int convert(char* buf, IT_Format f);
```

Description This function fills the supplied buffer with the requested format of data corresponding to the contents of the AVA object. For example, given an `IT_AVA` object that is part of a subject `IT_AVAList`, you could obtain the string form of the common name component of a distinguished name by using `MyAVA.convert(buf, IT_FMT_STRING)`.

Parameters

`buf`	The user supplied buffer that must be of sufficient size to hold the requested conversion. To find the required length of buffer for a particular type, call `IT_AVA::length()`.
`f`	The format of the required conversion. The following `IT_Format` values are supported: `IT_FMT_DER`. In this format, `buf` contains a sequence of bytes corresponding to the DER encoding of the AVA. This option is typically only used by applications that require special processing of the DER data. `IT_FMT_STRING`. In this format, `buf` contains a null-terminated sequence of characters corresponding to the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields. `IT_FMT_HEX_STRING`. In this format, `buf` contains a formatted hexadecimal dump of the DER data of the AVA. `IT_FMT_INTERNAL`. In this format, `buf` contains the value of a variable of the SSLeay data type `X509_NAME_ENTRY *`.

WARNING: This value provides access to low-level SSL Toolkit data structures, and is non-portable. Code that uses this feature will not work if the underlying SSL toolkit is changed. `IT_FMT_INTERNAL` allows direct access to the low-level SSL toolkit data representation of this AVA, thus allowing the user to directly call the toolkit API. Take extreme care if using this option.

Return Value Returns the number of bytes required to store the information associated with this AVA in the requested format. Returns `-1` if the required conversion is not supported.

IT_AVA::length()

Synopsis

```
virtual int length(IT_Format f);
```

Description This function is used to calculate how much storage is required to hold the result of a call to `IT_AVA::convert()` for a particular `IT_Format` value. Refer to `IT_AVA::convert()` for a list of the supported `IT_Format` values.

Parameters

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`

`IT_FMT_STRING`

`IT_FMT_HEX_STRING`

`IT_FMT_INTERNAL`

Return Value Returns the number of bytes required to store the result of the conversion. Returns `-1` if the required conversion is not supported.

IT_AVA::OID()

Synopsis

```
virtual int OID(IT_OID& retOID);
```

Description This function obtains the `IT_OID` structure which represents the object identifier for this AVA.

Parameters

`retOID` The `IT_OID` variable that is to be updated.

Return Value Returns `IT_SSL_SUCCESS` if `IT_OID` structure is successfully obtained. Returns `IT_SSL_ERR_INVALID_OPERATION` if the `IT_AVA` object has not yet been initialized.

See Also

`T_OID_Tag`
`IT_AVAList::getAVAByOID()`
`IT_Extension::OID()`
`IT_ExtensionList::getExtensionByOID()`
`IT_OID`

IT_AVA::getSet()

Synopsis

```
virtual int getSet();
```

Description

This function obtains the set that an AVA belongs to in an `AVAList`. It is required only in the rare case where you expect to parse certificates that have AVA sets with a cardinality greater than one in the `RelativeDistinguishedName`. Normally, there is only one element in each set. Most OrbixSSL developers never use this function.

Return Value

Returns the set that an AVA belongs to in an `AVAList`.

Class IT_AVAList

Synopsis

An `IT_AVA_List` consists of a number of `IT_AVA` objects and is an abstraction of the distinguished name fields in a certificate. This class provides a number of methods for obtaining individual `IT_AVA` objects.

A distinguished name is composed of a number of Attribute Value Assertions (AVAs). Each `IT_AVA` instance represents one component of the distinguished name. `IT_AVA` instances may be selected from an `IT_AVAList` using `IT_OID_Tag` values as keys, or by using an integer array that represents the ASN.1 object identifier. It is also possible to iterate over the list.

C++

```
class IT_AVAList {
public:
      virtual int convert(char* buf, IT_Format f);
      virtual int getAVA(unsigned int pos, IT_AVA& retAVA);
      virtual int getAVAByOID(IT_OID oid, IT_AVA& retAVA);
      virtual int getAVAByOIDTag(IT_OID_Tag oid, IT_AVA& retAVA);
      virtual unsigned int getNumAVAs();
      virtual int length(IT_Format f);
};
```

See Also

`IT_AVA`
`IT_OID_Tag`

IT_AVAList::convert()

Synopsis

```
virtual int convert (char* buf, IT_Format f);
```

Description

This function fills the supplied buffer with the requested format of data corresponding to the contents of the `AVAList` object. For example, given an `IT_AVAList` object corresponding to the subject field, you can obtain the DER form of the name by calling `MyAVAList.convert(buf, IT_FMT_DER)`.

Parameters

`buf` A user-supplied buffer that must be of sufficient size to hold the requested conversion. To find the required length of buffer for a particular conversion type, call `IT_AVAList::length()`.

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`. In this format, `buf` contains a sequence of bytes corresponding to the DER encoding of the AVA. This option is typically used only by applications that require special processing of the DER data.

`IT_FMT_STRING`. In this format, `buf` contains a null-terminated sequence of characters corresponding to a printable string which contains the text values of the AVAs concatenated together. Each AVA element is preceded by the short name description of the AVA. For example, `"/C=IE/ST=Co. Dublin/L=Dublin/O=IONA Technologies PLC./OU=PD/CN=IONA Technologies Test CA/Email=info@iona.com"`.

`IT_FMT_HEX_STRING`. In this format, `buf` contains a null-terminated string which is a formatted hexadecimal dump of the DER data of the AVA.

`IT_FMT_INTERNAL`. In this format, `buf` contains the value of a variable of the SSLeay data type `X509_NAME *`.

WARNING: This value provides access to low-level SSL Toolkit data structures, and is non-portable. Code that uses this feature will not work if the underlying SSL toolkit is changed. `IT_FMT_INTERNAL` allows direct access to the low-level SSL toolkit data representation of this AVA, thus allowing the user to directly call the toolkit API. Take extreme care if using this option.

Return Value Returns an array of bytes that store the result of the conversion. Returns `NULL` if the required conversion is not supported.

IT_AVAList::getAVA()

Synopsis

```
virtual int getAVA(unsigned int pos, IT_AVA& retAVA);
```

Description This function obtains the AVA at the specified index.

Parameters

`pos`	The specified index position. The index ranges in value from `0` to the number of elements in the list minus `1`.
`retAVA`	The AVA object to be updated.

Return Value Returns `IT_AVA_SUCCESS` if the AVA is successfully returned at the specified index.

Returns `IT_SSL_ERR_INVALID_PARAMETER` if the index position is invalid.

Returns `IT_SSL_ERR_INVALID_OPERATION` if the `IT_AVAList` is not initialized.

Returns `IT_SSL_ERR_AVA_NOT_PRESENT` if the specified AVA does not exist.

See Also

```
IT_AVAList::getAVAByOID()
IT_AVAList::getAVAByOIDTag()
```

IT_AVAList::getAVAByOIDTag()

Synopsis

```
virtual int getAVAByOIDTag(IT_OID_Tag t, IT_AVA& retAVA);
```

Description This function obtains the `IT_AVA` element of the `IT_AVAList` which corresponds to the requested `IT_OID_Tag` value.

Parameters

`t`	The `IT_OID_Tag` value which identifies the AVA you want to retrieve.
`retAVA`	The AVA object to be updated.

Return Value Returns `IT_SSL_SUCCESS` if the `IT_AVA` element of the `IT_AVAList` is successfully returned.

Returns `IT_SSL_ERR_AVA_NOT_PRESENT` if the `IT_AVA` element of the `IT_AVAList` is not found.

Returns `IT_SSL_ERR_INVALID_OPERATION` if the `IT_AVAList` is not initialized.

See Also
```
enum IT_OID_Tag
IT_AVA::OID()
IT_AVAList::getAVAByOID()
struct IT_OID
```

IT_AVAList::getAVAByOID()

Synopsis
```
virtual int getAVAByOID(int* seq, unsigned int n, IT_AVA& retAVA);
```

Description This function obtains the `IT_AVA` element of the `IT_AVAList` which has the requested object identifier.

Parameters

`seq`	An array of integer values.
`n`	The number of elements in the array.
`retAVA`	The `IT_AVA` object to be updated.

Return Value Returns `IT_SSL_SUCCESS` if the `IT_AVA` element of the `IT_AVAList` is successfully returned.

Returns `IT_SSL_ERR_AVA_NOT_PRESENT` if the `IT_AVA` element of the `IT_AVAList` is not found.

Returns `IT_SSL_ERR_INVALID_OPERATION` if the `IT_AVAList` is not initialized.

See Also
```
enum IT_OID_Tag
IT_AVA::OID()
IT_AVAList::getAVAByOIDTag()
struct IT_OID
```

IT_AVAList::getNumAVAs()

Synopsis
```
virtual unsigned int getNumAVAs();
```

Description This function obtains the number of AVA elements contained in this `IT_AVAList`.

Return Value Returns the number of AVA elements.

See Also class `IT_AVA`

IT_AVAList::length()

Synopsis

```
virtual int length(IT_Format f);
```

Description

This function is used to calculate how much storage is required to hold the result of a call to `IT_AVAList::convert()` for a particular `IT_Format` value. Refer to `IT_AVAList::convert()` for a list of the supported `IT_Format` values.

Parameters

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`

`IT_FMT_STRING`

`IT_FMT_HEX_STRING`

`IT_FMT_INTERNAL`

For more information, refer to `IT_AVAList::convert()` on page 117.

Return Value

Returns the number of bytes required to store the result of the conversion. Returns `-1` if the required conversion is not supported.

See Also

`IT_AVAList::convert()`

Struct IT_CertError

Synopsis Some API functions use the structure `IT_CertError` to return information gathered during certificate chain processing.

C++

```
struct IT_CertError {
public:
   int depth;
   int error;
   int externalError;
   int externalErrorDepth;
   int externalErrorSet;
};
```

Description The structure `IT_CertError` contains the following fields:

`depth`	The depth in the certificate chain at which point the error was encountered.
`error`	The error code that OrbixSSL has associated with the certificate chain during validation of the certificate.
`externalErrorSet`	For diagnostic purposes, OrbixSSL provides direct access to the error code returned by the underlying SSL toolkit. This field is set to `1` if an external SSL toolkit error code is available. If `externalErrorIsSet` is `1`, you can examine the `externalError` and `externalErrorDepth` fields to get more details about the error returned by the toolkit.
`externalError`	This field contains the SSL toolkit's internal error code. Examine this field only if the value of `externalErrorIsSet` is `1`.
`externalErrorDepth`	This field contains the depth in the peer certificate chain at which the external error was encountered.

See Also

```
IT_X509CertChain::getErrorInfo()
IT_SSL::setClientCertValidationCB()
IT_SSL::setServerCertValidationCB()
```

Class IT_CRL_List

Synopsis

This class represents a list of the certificate revocation lists (CRLs) available to an OrbixSSL program. CRLs are described in Chapter 4, "Managing Certificates" and Chapter 7, "Validating Certificates".

Using class `IT_CRL_List`, you can open the CRLs located in any directory on your host, check for the presence of a specified certificate issuer in the CRLs, and add or remove CRLs.

C++

```
class IT_CRL_List {
public:
   IT_CRL_List();
   virtual ~IT_CRL_List();

   virtual int add(const IT_X509_CRL_Info& aCRL);
   virtual int remove(const IT_AVAList& lstIssuer);
   virtual int find(const IT_AVAList& lstIssuer,
      CORBA(Boolean)& bFound, IT_X509_CRL_Info& aCRL) const;
   virtual int openCRLFiles(const char* szCRLDir);
   virtual int PollForUpdates(CORBA(Boolean)& bUpdated);
};
```

See Also

```
IT_SSL::getCRLDir()
IT_X509_CRL_Info
IT_X509_Revoked
IT_X509_RevokedList
```

IT_CRL_List::add()

Synopsis

```
virtual int add(const IT_X509_CRL_Info& aCRL);
```

Description

This function adds a new CRL to the existing CRLs stored on the host. The CRL is represented as an object of type `IT_X509_CRL_Info`. The CRL is not written to file; it is represented in memory only.

Parameters This function takes the following parameter:

`aCRL`	An object that contains information about the CRL to be added.

Return Value Returns a non-zero value if it succeeds in adding the CRL. Otherwise, it returns zero.

IT_CRL_List::find()

Synopsis

```
virtual int find(const IT_AVAList& lstIssuer,
   CORBA(Boolean)& bFound, IT_X509_CRL_Info& aCRL) const;
```

Description This function locates the CRL issued by a specified CA.

Parameters This function takes the following parameters:

`lstIssuer`	An object of type `IT_AVAList` containing the distinguished name that identifies the issuer.
`bFound`	A boolean value. Set to `true` if the CA has an associated CRL. Otherwise, this value is set to `false`.
`aCRL`	An object that represents the CRL associated with the CA.

Return Value Returns a non-zero value if it succeeds in finding the CRL. Otherwise, it returns zero.

IT_CRL_List::openCRLFiles()

Synopsis

```
virtual int openCRLFiles(const char* szCRLDir)
```

Description This function opens the CRL files contained in a specified directory. You must call this function before using the other functions in this class.

Parameters This function takes the following parameter:

`szCRLDir`	The directory that contains the CRL files.

Return Value Returns a non-zero value if it succeeds in opening the CRL files. Otherwise, it returns zero.

IT_CRL_List::PollForUpdates()

Synopsis

```
virtual int PollForUpdates(CORBA(Boolean)& bUpdated);
```

Description Checks the open CRL files to determine if the CRLs have been updated since they were last read. If the files have been updated, the updates are read into memory.

Parameters This function takes the following parameter:

`bUpdated`	This boolean value indicates whether the files have been updated. It is set to `true` if they have been updated. Otherwise, it is set to `false`.

Return Value Returns a non-zero value if it succeeds in polling the files. Otherwise, it returns zero.

IT_CRL_List::remove()

Synopsis

```
virtual int remove(const IT_AVAList& lstIssuer);
```

Description This function removes an existing CRL from the list of CRLs stored on the host. The CRL is represented as an object of type `IT_X509_CRL_Info`. The update is not written to file; it is represented in memory only.

Parameters This function takes the following parameter:

`lstIssuer`	An object of type `IT_AVAList` containing the distinguished name that identifies the issuer associated with the CRL to be removed.

Return Value Returns a non-zero value if it succeeds in removing the CRL. Otherwise, it returns zero.

Class IT_Extension

Synopsis

The `IT_Extension` and `IT_ExtensionList` classes provide the OrbixSSL developer with an interface to any X.509 version three extensions that an X.509 certificate can contain. `IT_X509Cert::getExtensions()` enables you to obtain an `IT_ExtensionList` object that has a number of member functions for retrieving individual extensions.

The `IT_Extension` class provides an interface to accessing the data for one particular extension. Using the `IT_Extension::convert()` and `IT_Extension::length()` member functions, the data can be converted into a number of representations. Use of the `IT_Extension` and `IT_ExtensionList` classes is analogous to the use of the `IT_AVA` and `IT_AVAList` classes.

C++

```
class IT_Extension {
public:
        virtual int convert(char* buf, IT_Format f);
        virtual int critical();
        virtual int length(IT_Format f);
        virtual int OID(IT_OID& retOID);
};
```

IT_Extension::convert()

Synopsis

```
virtual int convert(char* buf, IT_Format f);
```

Description

This function fills the supplied buffer with the requested format of data corresponding to the contents of the `IT_Extension` object.

Parameters

`buf` The user supplied buffer that must be of sufficient size to hold the requested conversion. To find the required length of buffer for a particular conversion type, call the `IT_Extension::length()` function.

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`. In this format, `buf` contains a sequence of bytes corresponding to the DER encoding of the extension. This option is typically only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. In this format, `buf` contains a null terminated sequence of characters corresponding to the actual data contained in the extension. This data has not been modified in any way, and may include non printable characters if present in the actual extension data. This is just a regular 'C' string for printable string fields.

`IT_FMT_HEX_STRING`. In this format, `buf` contains a formatted hexadecimal dump of the DER data of the extension.

`IT_FMT_INTERNAL` where `buf` will contain the value of a variable of the SSLeay data type `X509_EXTENSION *`.

WARNING: This value provides access to low-level SSL Toolkit data structures, and is non-portable. Code that uses this feature will not work if the underlying SSL toolkit is changed. `IT_FMT_INTERNAL` allows direct access to the low-level SSL toolkit data representation of this AVA, thus allowing the user to directly call the toolkit API. Take extreme care if using this option.

Return Value Returns an array of bytes that store the result of the conversion. Returns `NULL` if the required conversion is not supported.

IT_Extension::critical()

Synopsis `virtual int critical();`

Description This function determines whether or not this extension has been designated as critical.

Return Value Returns the integer value of the critical field in the extension.

IT_Extension::length()

Synopsis `virtual int length(IT_Format f);`

Description This function is used to calculate how much storage is required to hold the result of a call to `IT_Extension::convert()` for a particular `IT_Format` value.

Parameters

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`

`IT_FMT_STRING`

`IT_FMT_HEX_STRING`

`IT_FMT_INTERNAL`

For more information, refer to `IT_Extension::convert()` on page 129.

Return Value Returns the number of bytes required to store the result of the conversion. Returns `-1` if the required conversion is not supported.

IT_Extension::OID()

Synopsis `virtual int OID(IT_OID& retOID);`

Description This function obtains the `IT_OID` structure that represents the object identifier for this extension.

Parameters

`retOID` The `IT_OID` variable that is to be updated.

Return Value Returns `IT_SSL_SUCCESS` if the `IT_OID` element of the `IT_Extension` is successfully returned.

Returns `IT_SSL_ERR_INVALID_OPERATION` if the `IT_OID` element of the `IT_Extension` is invalid.

See Also
`IT_OID_Tag`
`IT_ExtensionList::getAVAByOID()`
`IT_OID`

Class IT_ExtensionList

Synopsis

The `IT_Extension` and `IT_ExtensionList` classes provide you with an interface to any X.509 version three extensions. `IT_X509Cert::getExtensions()` is used to obtain an `IT_ExtensionList` object that has a number of member functions for retrieving individual `IT_Extension` objects.

The `IT_Extension` class provides an interface to accessing the data for one particular extension. Use of the `IT_Extension` and `IT_ExtensionList` classes is analogous to the use of the `IT_AVA` and `IT_AVAList` classes.

C++

```
class IT_ExtensionList {
public:
      virtual int convert(char* buf, IT_Format f);
      virtual unsigned int getNumExtensions();
      virtual int getExtension(int pos, IT_Extension& retExt);
      virtual int getExtensionByOID(IT_OID oid);
      virtual int getExtensionByOIDTag(IT_OID_Tag oid);
      virtual int length(IT_Format f);
};
```

IT_ExtensionList::convert()

Synopsis

```
virtual int convert (char* buf, IT_Format f);
```

Description

This function fills the supplied buffer with the requested format of data corresponding to the contents of the `IT_ExtensionList` object.

Note: Generally `convert()` is called on the individual extensions. This function is not commonly used.

Parameters

`buf`	The user-supplied buffer that must be of sufficient size to hold the requested conversion. Call `IT_Extension::length()` to find the required length of buffer for a particular conversion type.
`f`	The format of the required conversion. The following `IT_Format` value is supported: `IT_FMT_INTERNAL`. In this format, `buf` contains the value of a variable of the SSLeay data type `X509_Extension *`.

WARNING: This value provides access to low-level SSL Toolkit data structures, and is non-portable. Code that uses this feature will not work if the underlying SSL toolkit is changed. `IT_FMT_INTERNAL` allows direct access to the low-level SSL toolkit data representation of this AVA, thus allowing the user to directly call the toolkit API. Take extreme care if using this option.

Return Value Returns an array of bytes that store the result of the conversion. Returns `NULL` if the required conversion is not supported.

See Also `IT_Extension::length()`

IT_ExtensionList::getExtension()

Synopsis

```
virtual int getExtension(int pos, IT_Extension& retExt);
```

Description This function obtains the extension at the specified index in the list.

Parameters

`pos`	The index position of the required extension in this list.
`retExt`	The `IT_Extension` object to be updated.

Return Value Returns `IT_SSL_SUCCESS` if the extension is successfully retrieved.

Returns `IT_SSL_ERR_INVALID_OPERATION` if the extension list is not initialized.

Returns IT_SSL_ERR_EXTENSION_NOT_PRESENT if the specified extension does not exist.

IT_ExtensionList::getExtensionByOID()

Synopsis

```
virtual int getExtensionByOID
                (int* seq, unsigned int n, IT_Extension& retExt);
```

Description This function obtains the IT_Extension element of the IT_ExtensionList which has the requested object identifier.

Parameters

seq	An array of integers representing the ASN.1 object identifier.
n	The number of elements in the array.
retExt	The IT_Extension object to be updated.

Return Value Returns IT_SSL_SUCCESS if the IT_Extension element of the IT_ExtensionList is successfully returned.

Returns IT_SSL_ERR_EXTENSION_NOT_PRESENT if the IT_Extension element of the IT_ExtensionList is not found.

Returns IT_SSL_ERR_INVALID_OPERATION if the IT_Extension element of the IT_ExtensionList is invalid.

See Also

```
IT_OID_Tag
IT_Extension::OID()
IT_ExtensionList::getExtension()
IT_OID
```

IT_ExtensionList::getExtensionByOIDTag()

Synopsis

```
virtual int getExtensionListByOIDTag
                (IT_OID_Tag oid, IT_Extension& retExt);
```

Description This function obtains the IT_Extension element of the IT_ExtensionList, which corresponds to the supplied IT_OID_Tag value.

Parameters

`oid`	The `IT_OID_Tag` variable which identifies the extension we want to retrieve.
`retExt`	The `IT_Extension` object to be updated.

Return Value Returns `IT_SSL_SUCCESS` if the `IT_Extension` element of the `IT_ExtensionList` is successfully returned.

Returns `IT_SSL_ERR_EXTENSION_NOT_PRESENT` if the `IT_Extension` element of the `IT_ExtensionList` is not found.

Returns `IT_SSL_ERR_INVALID_OPERATION` if the `IT_Extension` element of the `IT_ExtensionList` is invalid.

See Also
```
enum IT_OID_Tag
IT_Extension::OID()
IT_ExtensionList::getExtension()
struct IT_OID
```

IT_ExtensionList::getNumExtensions()

Synopsis
```
virtual unsigned int getNumExtensions();
```

Description This function obtains the number of extensions in this list.

Return Value Returns the number of extensions in this list.

IT_ExtensionList::length()

Synopsis
```
virtual int length(IT_Format f);
```

Description This function is used to calculate how much storage is required to hold the result of a call to `IT_ExtensionList::convert()` for a particular `IT_Format` value. Refer to `IT_ExtensionList::convert()` for a list of the supported `IT_Format` values.

Parameters

`f` The following `IT_Format` value is supported:

`IT_FMT_INTERNAL`

Refer to `IT_ExtensionList::convert()` on page 133, for more information.

Return Value Returns the number of bytes required to store the result of the conversion. Returns `-1` if the required conversion is not supported.

See Also `IT_ExtensionList::convert()`

Class IT_IntegerData

Synopsis Some OrbixSSL functions, such as `IT_X509Cert::getSerialNumber()`, return ASN.1 integers as `out` parameters. Class `IT_IntegerData` is the OrbixSSL abstraction for an ASN.1 integer.

OrbixSSL uses this class because some ASN.1 integers might be too big to be represented by the C++ `long` data type. Class `IT_IntegerData` allows you to determine the category of the integer and to access the DER data, if necessary.

C++

```
class IT_IntegerData {
public:
   virtual int convert (char* buf, IT_format f);
   virtual int getLong (long& retLong);
   virtual int length (IT_format f);
};
```

IT_IntegerData::convert()

Synopsis

```
virtual int convert (char* buf, IT_Format f);
```

Description This function fills the supplied buffer with the requested format of data corresponding to the contents of the `IT_IntegerData` object.

Parameters

`buf` The user-supplied buffer that must be of sufficient size to hold the requested conversion. To find the required length of buffer for a particular conversion type, call `IT_IntegerData::length()`.

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`. In this format, `buf` contains a sequence of bytes corresponding to the DER encoding of the ASN.1 integer. This option is typically only used by applications that require special processing of the DER data.

`IT_FMT_HEX_STRING`. In this format, `buf` contains a formatted hexadecimal dump of the DER encoding of the ASN.1 integer.

`IT_FMT_INTERNAL`. In this format, `buf` contains the value of a variable of the SSLeay data type `ASN1_INTEGER *`.

WARNING: This value provides access to low-level SSL Toolkit data structures, and is non-portable. Code that uses this feature will not work if the underlying SSL toolkit is changed. `IT_FMT_INTERNAL` allows direct access to the low-level SSL toolkit data representation of this AVA, thus allowing the user to directly call the toolkit API. Take extreme care if using this option.

Return Value Returns the requested format of data. Returns `NULL` if the required conversion is not supported.

IT_IntegerData::getLong()

Synopsis

```
virtual int getLong(long& retLong);
```

Description This function obtains the `long` associated with this ASN.1 integer. It is important to examine the return value of this function.

Parameters

`retLong` The `long` that is to be updated with the value of the ASN.1 integer.

Return Value Returns `1` if the value can fit in `long`. Returns `0` otherwise.

If 0 is returned, do not use the value retLong. For large numbers, the application must use IT_IntegerData::convert() to obtain the DER data which can then be processed by the application.

IT_IntegerData::length()

Synopsis `virtual int length(IT_Format f);`

Description This function is used to calculate the storage required to hold the result of a call to IT_IntegerData::convert() for a particular IT_Format value. Refer to IT_IntegerData::convert() for a list of the supported IT_Format values.

Parameters

`f` The format of the required conversion. The following IT_Format values are supported:

IT_FMT_DER

IT_FMT_HEX_STRING

IT_FMT_INTERNAL

For more information, refer to IT_IntegerData::convert() on page 139.

Return Value Returns the number of bytes required to store the result of the conversion. Returns -1 if the required conversion is not supported.

See Also IT_IntegerData::convert()

Struct IT_OID

Synopsis

This structure is used by OrbixSSL to hold information identifying an ASN.1 object. An ASN.1 object identifier is a sequence of integer values used to identify certificate components. ASN.1 is the low-level format in which X.509 certificates are stored. This structure holds information by maintaining an array of integers that map onto the ASN.1 sequence of integers which correspond to an object identifier (OID).

OrbixSSL handles object identifiers as follows:

1. It provides an enumerated type `IT_OID_Tag` which has values for a number of common objects. For example, `IT_OIDT_commonName` identifies the common name (CN) component of a subject field in a certificate. Use of this enumerated type is sufficient for most OrbixSSL developer requirements.
2. If the desired OIDs are not listed in the enumerated values for type `IT_OID_Tag`, you can directly supply the sequence of integers that represent the OID.

An explanation of the fields of struct `IT_OID` follows:

`tag` This field contains the value of the enumerated type `IT_OID_Tag` which represents the object. When OrbixSSL API functions return `IT_OID` structures, they will supply this field if possible as well as always returning values for the `OID` and `OIDLen` fields. If the value for this field is `IT_OIDT_UNDEF`, this means that either no object has been specified or that the OID has been directly specified using the other fields of the structure.

`OID` This array of integer values corresponds to the ASN.1 sequence of integers which represents an ASN.1 object identifier. This field should not be examined unless the `OIDLen` field of the same structure is > `0`.

`OIDLen` This field represents the number of elements in the `OID` array field above.

If the value of the tag field is `IT_OIDT_UNDEF` and the value of `OIDLen` is `0`, no OID has been specified.

An example of an IT_OID struct representing the common name field of a name component in a certificate could have the following values:

(1)
```
tag = IT_OIDT_commonName
OIDLen = 0
```

(2)
```
tag = IT_OIDT_commonName
OIDLen = 3
OID = {0x55, 0x04, 0x03}
```

(3)
```
tag = IT_OIDT_UNDEF
OIDLen = 3
OID = {0x55, 0x04, 0x03}
```

An OrbixSSL developer will normally use the tag value where possible because it is easy to use. However, it is also faster to look up an IT_X509Extension or an IT_AVA using a supplied tag value rather than just the sequence of integers. When passing IT_OID objects to OrbixSSL API functions, it is not necessary to specify the integer sequence if the tag value has been specified.

C++

```
struct IT_OID {
      IT_OID_Tag tag;
      unsigned int OID[MAX_OID_CARDINALITY];
      unsigned int OIDLEN;
};
```

See Also

IT_OID_Tag
IT_AVA::OID()
IT_AVAList::getAVAByOID()
IT_Extension::OID()
IT_ExtensionList::getExtensionByOID()

Enum IT_OIDTag

Synopsis The values of this enumerated data type are used to represent an ASN.1 object identifier (OID). Access to certificate components using the `IT_OIDTag` is faster than using the raw sequence of integers that correspond to the underlying ASN.1 OID value.

```
enum IT_OID_Tag {
   IT_OIDT_UNKNOWN = 0,

      IT_OIDT_rsadsi,
      IT_OIDT_pkcs,
      IT_OIDT_md2,
      IT_OIDT_md5,
      IT_OIDT_rc4,
      IT_OIDT_rsaEncryption,
      IT_OIDT_md2WithRSAEncryption,
      IT_OIDT_md5WithRSAEncryption,
      IT_OIDT_pbeWithMD2AndDES_CBC,
      IT_OIDT_pbeWithMD5AndDES_CBC,

      IT_OIDT_X500,
      IT_OIDT_X509,
      IT_OIDT_commonName,
      IT_OIDT_countryName,
      IT_OIDT_localityName,
      IT_OIDT_stateOrProvinceName,
      IT_OIDT_organizationName,
      IT_OIDT_organizationalUnitName,
      IT_OIDT_rsa,

      IT_OIDT_pkcs7,
      IT_OIDT_pkcs7_data,
      IT_OIDT_pkcs7_signed,
      IT_OIDT_pkcs7_enveloped,
      IT_OIDT_pkcs7_signedAndEnveloped,
      IT_OIDT_pkcs7_digest,
      IT_OIDT_pkcs7_encrypted,
      IT_OIDT_pkcs3,
```

```
IT_OIDT_dhKeyAgreement,
IT_OIDT_des_ecb,
IT_OIDT_des_cfb64,
IT_OIDT_des_cbc,
IT_OIDT_des_ede,
IT_OIDT_des_ede3,
IT_OIDT_idea_cbc,
IT_OIDT_idea_cfb64,
IT_OIDT_idea_ecb,

IT_OIDT_rc2_cbc,
IT_OIDT_rc2_ecb,
IT_OIDT_rc2_cfb64,
IT_OIDT_rc2_ofb64,
IT_OIDT_sha,
IT_OIDT_shaWithRSAEncryption,
IT_OIDT_des_ede_cbc,
IT_OIDT_des_ede3_cbc,
IT_OIDT_des_ofb64,
IT_OIDT_idea_ofb64,

IT_OIDT_pkcs9,
IT_OIDT_pkcs9_emailAddress,
IT_OIDT_pkcs9_unstructuredName,
IT_OIDT_pkcs9_contentType,
IT_OIDT_pkcs9_messageDigest,
IT_OIDT_pkcs9_signingTime,
IT_OIDT_pkcs9_countersignature,
IT_OIDT_pkcs9_challengePassword,
IT_OIDT_pkcs9_unstructuredAddress,
IT_OIDT_pkcs9_extCertAttributes,

IT_OIDT_netscape,
IT_OIDT_netscape_cert_extension,
IT_OIDT_netscape_data_type,
IT_OIDT_des_ede_cfb64,
IT_OIDT_des_ede3_cfb64,
IT_OIDT_des_ede_ofb64,
IT_OIDT_des_ede3_ofb64,
IT_OIDT_sha1,
IT_OIDT_sha1WithRSAEncryption,
IT_OIDT_dsaWithSHA,
IT_OIDT_dsa,
```

```
        IT_OIDT_pbeWithSHA1AndRC2_CBC,
        IT_OIDT_pbeWithSHA1AndRC4,
        IT_OIDT_dsaWithSHA1,
        IT_OIDT_netscape_cert_type,
        IT_OIDT_netscape_base_url,
        IT_OIDT_netscape_revocation_url,
        IT_OIDT_netscape_ca_revocation_url,
        IT_OIDT_netscape_renewal_url,
        IT_OIDT_netscape_ca_policy_url,
        IT_OIDT_netscape_ssl_server_name,
        IT_OIDT_netscape_comment,
        IT_OIDT_netscape_cert_sequence,

        IT_OIDT_desx_cbc,
        IT_OIDT_ld_ce,
        IT_OIDT_subject_key_identifier,
        IT_OIDT_key_usage,
        IT_OIDT_private_key_usage_period,
        IT_OIDT_subject_alt_name,
        IT_OIDT_issuer_alt_name,

        IT_OIDT_basic_constraints,
        IT_OIDT_crl_number,
        IT_OIDT_certificate_policies,
        IT_OIDT_authority_key_identifier,
        IT_OIDT_bf_cbc,
        IT_OIDT_bf_ecb,
        IT_OIDT_bf_cfb64,
        IT_OIDT_bf_ofb64,
        IT_OIDT_mdc2,
        IT_OIDT_mdc2WithRSA,
};
```

See Also

IT_AVAList::getAVAByOIDTag()
IT_ExtensionList::getExtensionByOIDTag()
IT_OID

Class IT_SSL

Synopsis The `IT_SSL` class is the main OrbixSSL API interface consisting of basic API member functions, specific security policy and configuration member functions, and member functions for the custom retrieval of certificates and private keys.

```
class IT_SSL {
public:
   // Toolkit initialization functions.
   virtual int init();
   virtual int initScope(const char* scope);
   virtual char *getInitErrorString();

   // Basic API member functions.
   virtual int setSecurityName(const char *name);
   virtual char *getSecurityName();
   virtual void setValidateServerCertCallback
      (IT_ValidateX509CertCB cb);
   virtual voidsetValidateClientCertCallback
      (IT_ValidateX509CertCB cb);
   virtual int getPeerCert(CORBA(Object_ptr) obj,
      IT_X509Cert& PeerCert);
   virtual int getPeerCert(CORBA(Request)* req,
      IT_X509Cert& PeerCert);
   virtual int getPeerCert(int fd, IT_X509Cert& PeerCert);
   virtual int getApplicationCert(IT_X509Cert& cert);
   virtual int setPrivateKeyPassword(char *password);
   virtual char *getPassword(const char *prompt);
   virtual int setInvocationPolicy(int pol);
   virtual int getInvocationPolicy();
   virtual int specifySecurityForInterfaces
      (IT_CommsSecuritySpec *SpecList, unsigned int n);
   virtual int specifySecurityForServers
      (IT_CommsSecuritySpec *SpecList, unsigned int n);
   virtual IT_SSLCipherSuite getNegotiatedCipherSuite
      (CORBA(Object_ptr) obj);
   virtual IT_SSLCipherSuite getNegotiatedCipherSuite
      (CORBA(Request)* req);
```

```
    virtual IT_SSLCipherSuite getNegotiatedCipherSuite(int fd);

    // Policy and configuration member functions.
    virtual int specifyCipherSuites
       (const IT_SSLCipherSuites& suite, const unsigned int n,
       IT_SSLCipherSuites& SetSuite, unsigned int& retn);
    virtual int setCacheOptions(const int opts);
    virtual int getCacheOptions();
    virtual int setMaxChainDepth(unsigned int depth);
    virtual int getMaxChainDepth();
    virtual int setClientAuthentication(int f);
    virtual int getClientAuthentication();

    // Custom retrieval of certs and private keys member functions.
    virtual int setRSAPrivateKeyFromDER
       (char *PrivateKey, unsigned int len);
    virtual int setRSAPrivateKeyFromFile(char *file, IT_Format f);
    virtual int setX509CertFromDER
       (char *derCert, unsigned int len);
    virtual int setX509CertFromFile(const char *FileName,
       IT_Format f);
    virtual char *getCRLDir();
    virtual const char *getCAListFile(void) const;
    virtual int hasPassword(void) const;

    // The default scope used within init().
    static const char *DEFAULT_CONFIG_SCOPE;
};
```

IT_SSL::getApplicationCert()

Synopsis

```
int getApplicationCert(IT_X509Cert& cert);
```

Description This function obtains the certificate associated with the current application.

Parameters

`cert` The application certificate object.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in obtaining the application certificate. Otherwise, it returns an error code.

IT_SSL::getCacheOptions()

Synopsis `int getCacheOptions();`

Description This function obtains the current setting for the OrbixSSL cache options.

Return Value Returns the current setting for the OrbixSSL cache.

See Also `IT_SSL::setCacheOptions()`

IT_SSL::getClientAuthentication()

Synopsis `int getClientAuthentication();`

Description This function is used to determine whether the application is configured to authenticate clients.

Return Value This function returns `1` to signify that clients will be authenticated. Returns `0` otherwise.

See Also `IT_SSL::setClientAuthentication()`

IT_SSL::getCRLDir()

Synopsis `char *getCRLDir();`

Description This function returns the directory in which the application expects certificate revocation lists (CRLs) to be stored.

Return Value Returns the CRL directory for the application, if set using the `IT_CRL_REPOSITORY` configuration variable.

IT_SSL::getErrorString()

Synopsis `char* getErrorString();`

Description This method returns a description of an initialization error, if available.

IT_SSL::getInvocationPolicy()

Synopsis

```
int getInvocationPolicy();
```

Description This function obtains the invocation policy setting for an OrbixSSL application. The invocation policy for an OrbixSSL application specifies whether clients support or require SSL for incoming and outgoing connections.

Return Value Returns the current invocation policy value.

See Also `IT_SSL::setInvocationPolicy()`

IT_SSL::getMaxChainDepth()

Synopsis

```
int getMaxChainDepth();
```

Description This function returns the maximum allowed depth of the certificate chain for this application. The maximum certificate chain length acceptable to OrbixSSL clients and servers using the policy file is set by `IT_MAX_ALLOWED_CHAIN_DEPTH` or `IT_DEFAULT_MAX_CHAIN_DEPTH` during configuration. Applications can change the maximum certificate chain length by calling `IT_SSL::setMaxChainDepth()`. However, they may only set the length of the certificate chain to be less than or equal to `IT_MAX_ALLOWED_CHAIN_DEPTH`.

Return Value Returns a numeric value specifying the maximum length of the certificate chain.

See Also `IT_SSL::setMaxChainDepth()`

IT_SSL::getNegotiatedCipherSuite()

Synopsis

```
IT_SSLCipherSuite getNegotiatedCipherSuite
                            (CORBA(Object_ptr) obj);
```

Description This function allows OrbixSSL applications to query the specified cipher that was chosen for connection to the specified peer. It does this by returning the cipher suite associated with the specified remote object (`obj`).

Parameters

`obj` A remote object.

Return Value Returns the chosen `IT_SSLCipherSuite` value.

Returns IT_SSL_ERR_NO_CONNECTION to indicate that the object selected is not remote.

Returns IT_SSL_ERR_NO_CIPHER to indicate that the connection to the peer is insecure. In such a case, there is no associated cipher.

See Also IT_SSL::specifyCipherSuites()

IT_SSL::getNegotiatedCipherSuite()

Synopsis

```
IT_SSLCipherSuite getNegotiatedCipherSuite
                  (CORBA (Object(Request)* req);
```

Description This function allows OrbixSSL applications to query the specified cipher that was chosen for connection to the specified peer. It does this by requesting the cipher suite associated with the specified connection (req).

Parameters

req	A request object sent across the connection.

Return Value Returns the chosen IT_SSLCipherSuite value.

Returns IT_SSL_ERR_NO_CONNECTION to indicate that the connection selected is not remote.

Returns IT_SSL_ERR_NO_CIPHER to indicate that the connection to the peer is insecure. In such a case, there is no associated cipher.

See Also IT_SSL::specifyCipherSuites()

IT_SSL::getNegotiatedCipherSuite()

Synopsis

```
IT_SSLCipherSuite getNegotiatedCipherSuite(int fd);
```

Description This function allows OrbixSSL applications to query the specified cipher that was chosen for connection to the specified peer. It does this by returning the cipher suite associated with the file descriptor (fd) for a particular connection.

Parameters

fd	The file descriptor for a particular connection.

Return Value Returns the chosen `IT_SSLCipherSuite` value.

Returns `IT_SSL_ERR_NO_CONNECTION` to indicate that the file descriptor is invalid.

Returns `IT_SSL_ERR_NO_CIPHER` to indicate that the connection to the peer is insecure. In such a case, there is no associated cipher.

See Also `IT_SSL::specifyCipherSuites()`

IT_SSL::getPassword()

Synopsis

```
char* getPassword(const char* prompt);
```

Description This function requests the user to input a password and returns the password as a string. This is a console-based function. When it requests the user to enter the password, this function disables console echoing to prevent the password displaying on the user's screen.

Parameters

`prompt`	A prompt to display when asking the user to input a password.

Return Value Returns the string entered by the user.

IT_SSL::getPeerCert()

Synopsis

```
int getPeerCert(CORBA(Object_ptr)& obj, IT_X509Cert& PeerCert);
```

Description This function allows OrbixSSL applications to query peer certificates. It retrieves the peer certificate information associated with a remote object (`obj`) and returns this information in the supplied `IT_X509Cert` object.

Parameters

`obj`	A remote object.
`PeerCert`	The `IT_X509Cert` object to be updated with the peer certificate information.

Return Value Returns `IT_SSL_ERR_NO_CONNECTION` to indicate that the object selected is not remote.

Returns `IT_SSL_ERR_INSECURE_CONNECTION` to indicate that the connection to the peer is insecure. This means that no certificate is available.

Returns `IT_SSL_ERR_NO_CERT_AVAILABLE` to indicate that the connection to the peer is secure but no certificate is available. For example, this can occur when client authentication is disabled. Secure servers will always have certificates.

See Also `class IT_X509Cert`

IT_SSL::getPeerCert()

Synopsis

```
int getPeerCert(CORBA(Request)& req, IT_X509Cert& PeerCert);
```

Description This function allows an OrbixSSL application to request the certificate of a peer. It retrieves the peer certificate information associated with the specified connection (`req`) and returns this information in the supplied `IT_X509Cert` object.

Parameters

`req`	A specified connection.
`PeerCert`	The `IT_X509Cert` object to be updated with the peer certificate information.

Return Value Returns `IT_SSL_ERR_NO_CONNECTION` to indicate that the specified connection is invalid.

Returns `IT_SSL_ERR_INSECURE_CONNECTION` to indicate that the connection to the peer is insecure. This means that no certificate is available.

Returns `IT_SSL_ERR_NO_CERT_AVAILABLE` to indicate that the connection to the peer is secure but no certificate is available. This can occur when client authentication is disabled. Secure servers always have certificates.

See Also `IT_X509Cert`

IT_SSL::getPeerCert()

Synopsis

```
int getPeerCert(int fd, IT_X509Cert& PeerCert);
```

Description This function allows OrbixSSL applications to query peer certificates. It retrieves the peer certificate information associated with the file descriptor (`fd`) for a particular connection and returns this information in the supplied `IT_X509Cert` object.

Parameters

`fd`	The file descriptor for a particular connection.
`PeerCert`	The `IT_X509Cert` object to be updated with the peer certificate information.

Return Value Returns `IT_SSL_ERR_NO_CONNECTION` to indicate that the file descriptor is invalid.

Returns `IT_SSL_ERR_INSECURE_CONNECTION` to indicate that the connection to the peer is insecure. This means that no certificate is available.

Returns `IT_SSL_ERR_NO_CERT_AVAILABLE` to indicate that the connection to the peer is secure but no certificate is available. This can occur when client authentication is disabled. Secure servers will always have certificates.

See Also

```
class IT_X509Cert
CORBA::Object::_fd()
CORBA::Request::descriptor()
```

IT_SSL::getSecurityName()

Synopsis

```
const char* getSecurityName();
```

Description This function returns the security name which the application is currently using. Refer to `IT_SSL::setSecurityName()` on page 165 for a detailed explanation of what the security name means.

Return Value Returns the security name string.

See Also
```
IT_SSL::SetRSAPrivateKeyFromFile()
IT_SSL::SetRSAPrivateKeyFromDER()
IT_SSL::setSecurityName()
```

IT_SSL::hasPassword()

Synopsis
```
int hasPassword(void) const;
```

Description If called in a server, this function indicates whether or not the server has received a private key pass phrase from the server key distribution mechanism (KDM). If the server has not yet received a pass phrase, you should call `IT_SSL::setPrivateKeyPassword()`.

Return Value Returns `IT_SSL_SUCCESS` if the server has received a pass phrase. Otherwise, it returns an error.

See Also
```
IT_SSL::getPassword()
IT_SSL::setPrivateKeyPassword()
```

IT_SSL::init()

Synopsis
```
int init();
```

Description This function must be called by the application before any communications take place. It initializes the OrbixSSL component.

Return Value Returns `IT_SSL_SUCCESS` if initialization of the OrbixSSL component is successful.

Returns `IT_SSL_ERR_SECURITY_INACTIVE` if OrbixSSL is either not available in the runtime environment or disabled. For example, this could occur if a security policy file does not exist or if `IT_DISABLE_SSL` is set to `TRUE` in the security policy file.

Returns `IT_SSL_ERR_VAR_CERT_DIR` if the certificate directory specified in the OrbixSSL configuration file is invalid.

Returns `IT_SSL_ERR_VAR_CA` if an invalid CA is specified.

Returns `IT_SSL_ERR_UNKNOWN_CONFIG_VAR` if an invalid configuration variable is specified in the OrbixSSL configuration file.

Returns `IT_SSL_ERR_NO_CONFIG_VAL_SPEC` if an OrbixSSL configuration variable is missing a value.

Returns `IT_SSL_ERR_BAD_CONFIG_VAL` if an invalid configuration value is specified.

Note: You can obtain a string describing the error by calling `IT_SSL::getInitErrorString()`.

See Also `IT_SSL::getInitErrorString()`

IT_SSL::initScope()

Synopsis

```
virtual int initScope(const char* scope);
```

Description This function instructs OrbixSSL to read the configuration settings for the application from a custom scope in the OrbixSSL configuration file. Configuration variables not specified in the custom scope are read from the scope `OrbixSSL`.

Return Value Returns `IT_SSL_SUCCESS` if initialization of the OrbixSSL component is successful. Otherwise, it returns an error.

See Also `IT_SSL::getInitErrorString()`

IT_SSL::setCacheOptions()

Synopsis

```
int setCacheOptions(IT_SSLCacheOptions opts);
```

Description This function sets the OrbixSSL cache option settings.

Parameters

opts	This parameter is the bitwise OR combination of the following flags: IT_SSL_CACHE_OFF. This option disables session caching. If this flag is specified, it must be the only flag used. IT_SSL_CACHE_CLIENT. This option enables caching for clients. IT_SSL_CACHE_SERVER. This option enables caching for servers.

Return Value Returns IT_SSL_SUCCESS or an error code indicating the failure reason.

See Also IT_SSL::getCacheOptions()

IT_SSL::setClientAuthentication()

Synopsis

```
int setClientAuthentication(int f);
```

Description This function is used by an application to specify whether client authentication should be performed or not. This function is primarily used by servers, but can be used by clients to enforce client authentication on any non bi-directional callbacks that they receive.

Parameters

f	Setting this parameter to 1 signifies that client authentication should be performed. Setting this parameter to 0 signifies that client authentication should not be performed.

Return Value This function returns 1 if the value is allowed by the security policy. Returns 0 otherwise.

See Also IT_SSL::getClientAuthentication()

IT_SSL::setInvocationPolicy()

Synopsis

```
int setInvocationPolicy(int pol);
```

Description This function is used by an OrbixSSL application to set the invocation policy for an application. The invocation policy for an OrbixSSL application controls whether the application supports or requires SSL communications for incoming or outgoing connections. Applications have separate control with respect to using OrbixSSL security to invoke operations and with respect to using OrbixSSL security to receive operation invocations. By default, an OrbixSSL application only allows secure incoming and outgoing connections.

Note: Clients can be servers when they receive callbacks; servers can also be clients of other servers, for example, the Orbix daemon.

Parameters

`pol` An integer value which is the bitwise `OR` combination of the `IT::SSL_InvocationOptions` flags detailed below:

```
IT_SECURE_ACCEPT
IT_INSECURE_ACCEPT
IT_SPECIFIED_INSECURE_CONNECT
IT_INSECURE_CONNECT
IT_SECURE_CONNECT
IT_SPECIFIED_SECURE_CONNECT
```

The options are explained as follows:

- `IT_SECURE_ACCEPT`

  This option means that the server accepts SSL connections. If the `IT_INSECURE_ACCEPT` option is not also specified, it accepts only SSL connections and rejects non-SSL connections. It rejects non-SSL connections by sending a `NO_PERMISSION` exception to the initiator and closing the connection. In this case, an `SSL_FAILURE` exception is generated at the server application.

- `IT_INSECURE_ACCEPT`

  This option means that the server is capable of accepting connections from non-SSL clients. If `IT_SECURE_ACCEPT` and `IT_INSECURE_ACCEPT` are both specified, the server serves both secure and insecure clients. This type of server offers optional connection authentication, privacy and

integrity to clients that wish to avail of it. It should not be specified for servers whose services are regarded as sensitive and to which access should be restricted.

- `IT_SECURE_CONNECT`

  This means the client is capable of initiating SSL connections. If this connect option is set, your client will only connect securely to servers and will reject insecure servers. In this case, an `SSL_FAILURE` exception will be thrown.

- `IT_SPECIFIED_INSECURE_CONNECT`

  For some secure client applications, it may be too restrictive to allow only secure connections to all servers; there may be one server (or a few) that you need to contact without using SSL. When this option is chosen, attempts to connect through specified insecure interfaces or to specified insecure servers will be allowed. For more information, refer to `IT_SSL::specifySecurityForInterfaces()` on page 170 and `IT_SSL::specifySecurityForServers()` on page 171.

- `IT_SPECIFIED_SECURE_CONNECT`

  This option means that the client will generally try to communicate insecurely with all servers, except when connecting through explicitly specified secure interfaces, or explicitly specified secure servers. When this option is specified, the client will additionally attempt to use SSL when the server's IOR indicates that it requires SSL.

Note: Currently, this is only possible if the client has an IOR from a server which contains the `TAG_SSL_SEC_TRANS` struct indicating that the server supports or requires SSL. OrbixSSL automatically includes this tag in IORs that are generated by SSL servers.

- `IT_INSECURE_CONNECT`

  This option means that your client is capable of initiating insecure connections and that the client side of the application has no security requirements.

Return Value Returns `IT_SSL_SUCCESS` if successful in specifying security for an OrbixSSL application.

Returns IT_SSL_ERR_INVALID_OPT_COMBO if an illegal combination of flags has been specified (for example, more than one CONNECT flag).

Returns IT_SSL_ERR_POLICY_DISALLOWS if the settings chosen are disallowed by the security policy.

See Also

```
IT_SSL::setClientAuthentication()
IT_SSL::specifyCipherSuites()
IT_SSL::specifySecurityForInterfaces()
IT_SSL::specifySecurityForServers()
```

IT_SSL::setMaxChainDepth()

Synopsis

```
int setMaxChainDepth(unsigned int depth);
```

Description

This function allows individual applications to set the length of the certificate chain. The maximum certificate chain length acceptable to OrbixSSL clients and servers using the policy file is set by IT_MAX_ALLOWED_CHAIN_DEPTH or IT_DEFAULT_MAX_CHAIN_DEPTH during configuration. Applications can only set the length of the certificate chain to be less than or equal to IT_MAX_ALLOWED_CHAIN_DEPTH.

Parameters

depth	Numeric value specifying the acceptable maximum certificate chain length.

Return Value

Returns IT_SSL_SUCCESS to accept numeric value specifying the maximum certificate chain length. Returns IT_SSL_ERR_USING_PRIVATE_KEY otherwise.

See Also

```
IT_SSL::getMaxChainDepth()
```

IT_SSL::setPrivateKeyPassword()

Synopsis

```
int setPrivateKeyPassword(char* password);
```

Description

This function sets the pass phrase for the private key of an OrbixSSL application. The private key for an OrbixSSL C++ application is encrypted in PEM format with a secret pass phrase and stored in the application certificate file. The private key pass phrase is generally chosen by the system administrator when creating the application certificate signing request (CSR).

An application needs to supply the pass phrase that protects the private key. If your private key is encrypted and you are not explicitly supplying your own private key using either `setRSAPrivatekeyFromDER` or `setRSAPrivateKeyFromFile`, call this function before calling `IT_SSL::setSecurityName()`.

Parameters

`password`	A null-terminated string containing the pass phrase that was used to encrypt the private key. If you use the SSLeay utilities to create certificate requests, this corresponds to the pass phrase you enter when executing the `ssleay req` command.

Return Value Returns `IT_SSL_SUCCESS` if the pass phrase is accepted. Otherwise, it returns an error code indicating the failure reason.

See Also
`IT_SSL::setRSAPrivateKeyFromFile()`
`IT_SSL::setRSAPrivateKeyFromDER()`
`IT_SSL::setSecurityName()`

IT_SSL::setRSAPrivateKeyFromDER()

Synopsis

```
int setRSAPrivateKeyFromDER(char* PrivateKey, unsigned int len);
```

Description `setRSAPrivateKeyFromDER()` is a member function allowing you to directly supply private keys to Orbix. Private keys are used by OrbixSSL applications for authentication purposes.

Given a PEM format private key file, you can convert it into the DER format using the following command line:

```
ssleay rsa -in MyPrivateKeyFile.pem -inform PEM
-outform DER -out MyPrivateKeyFile.der
```

Parameters

`PrivateKey`	`PrivateKey` points to a user supplied buffer of length `len` that contains the DER format private key.
`len`	Length of the buffer.

Return Value Returns `IT_SSL_SUCCESS` if private key is successfully supplied. Returns `IT_SSL_ERR_USING_PRIVATE_KEY` otherwise.

See Also
```
IT_SSL::setPrivateKeyPassword()
IT_SSL::setRSAPrivateKeyFromFile()
```

Note: You can supply a private key directly and still use `setSecurityName()`. However, if supplying a certificate directly, `setSecurityName()` cannot be used as you already have a certificate. For more information, refer to `IT_SSL::setX509CertFromDER()` on page 167.

IT_SSL::setRSAPrivateKeyFromFile()

Synopsis
```
int setRSAPrivateKeyFromFile(char* file, IT_Format f);
```

Description `setRSAPrivateKeyFromFile()` is a member function allowing you to supply private keys directly to Orbix. Private keys are used by OrbixSSL applications for authentication purposes. If the private key is encrypted, you must call `setPrivateKeyPassword()` before calling this function.

Parameters

`file`	The filename of the private key file.
`f`	Format of the file. For example: `IT_FMT_PEM` (PEM format). `IT_FMT_DER` (DER encoding).

Return Value Returns `IT_SSL_SUCCESS` if successful in supplying a private key. Returns `IT_SSL_ERR_USING_PRIVATE_KEY` (that is, private key file was read but could not be used), or `IT_SSL_FAILURE` otherwise.

See Also
```
IT_SSL::setPrivateKeyPassword()
IT_SSL::setRSAPrivateKeyFromFile()
```

IT_SSL::setSecurityName()

Synopsis

```
int setSecurityName(const char* name);
```

Description

This function is used to associate a particular certificate and private key with an OrbixSSL client or server application. OrbixSSL includes a certificate and private key retrieval mechanism.

Note: It is also possible for OrbixSSL developers to implement their own mechanism for retrieving certificates and private keys by supplying the private keys and certificates directly to OrbixSSL from memory or from file.

The parameter `name` is a string identifying the certificate to use. This string corresponds to the path of the certificate file relative to the directory specified by the `IT_CERTIFICATE_PATH` configuration variable. It is mapped onto the operating system's directory structure. (Without changing the application code, future versions of OrbixSSL may change this underlying mapping.)

For example, for an unmodified OrbixSSL installation, consider the following call:

```
OrbixSSL.setSecurityName("demo/demoserver")
```

This causes the application to use the certificate file `demoserver`, which is located in the OrbixSSL `demo` subdirectory of the OrbixSSL `certs` directory. The default location of the certificate directory is the `certs` subdirectory of the OrbixSSL installation location.

The certificate file specified by the `securityName` parameter must be in PEM format. If the certificate is in DER format, you can use the SSLeay utility `x509` to convert to PEM format. This certificate file can optionally contain the PEM format private key of the certificate, appended to the end of the PEM certificate. This private key is typically encrypted using the triple DES algorithm and a pass phrase unique to the certificate. Leaving the private key unencrypted is strongly discouraged.

Note: It is important to note that any person who gains access to unencrypted private keys would as a result be able to impersonate the entity for which the certificate was issued.

For OrbixSSL to use the private key, it needs to be aware of the pass phrase that was used to protect it. To make SSL aware of the pass phrase, call `IT_SSL::setPrivateKeyPassword()` before calling `IT_SSL::setSecurityName()`. It is not necessary to supply a pass phrase if the private key is not encrypted. This practice, however, is not encouraged. If the certificate file does not contain the private key, the application must supply OrbixSSL with the private key before calling `IT_SSL::setSecurityName()`. For more information, refer to `IT_SSL::setRSAPrivateKeyFromDER()` on page 163 and `IT_SSL::setRSAPrivateKeyFromFile()` on page 164.

Parameters

`name`	The security name string that identifies the certificate to use.

Return Value Returns `IT_SSL_SUCCESS` if the certificate has been successfully loaded.

Returns `IT_SSL_ERR_USING_PRIVATE_KEY` if the private key file was read but could not be used.

Returns `IT_SSL_ERR_LOADING_CERT` if unable to load the X.509 certificate.

See Also

```
IT_SSL::getSecurityName()
IT_SSL::setPrivateKeyPassword()
IT_SSL::setRSAPrivateKeyFromDER()
IT_SSL::setRSAPrivateKeyFromFile()
IT_SSL::setX509CertFromDer()
IT_SSL::setX509CertFromFile()
```

IT_SSL::setValidateClientCertCallback()

Synopsis

```
void setValidateClientCertCallback(IT_ValidateX509CertCB cb);
```

Description This function is used to validate client certificates. The user can register functions to process server or client certificates separately, or the same function for both if desired.

Parameters

`cb`	A pointer to the user-supplied function which is used to validate peer certificates.

See Also
```
IT_SSL::getPeerCert()
IT_SSL::setValidateServerCertCallback()
IT_ValidateX509CertCB
```

IT_SSL::setValidateServerCertCallback()

Synopsis
```
void setValidateServerCertCallback(IT_ValidateX509CertCB cb);
```

Description This function is used to validate server certificates.

Parameters

`cb`	A pointer to the user-supplied function which is used to validate peer certificates.

See Also
```
IT_SSL::getPeerCert()
IT_SSL::setValidateClientCertCallback()
IT_ValidateX509CertCB
```

IT_SSL::setX509CertFromDER()

Synopsis
```
int setX509CertFromDER(char* derCert, unsigned int len);
```

Description The `setSecurityName()` member function is used to automatically retrieve and use specified certificates. However, this function allows an OrbixSSL application to supply the certificate associated with the application for authentication purposes. You must call either `setRSAPrivateKeyFromDER()` or `setRSAPrivateKeyFromFile()` before using this function.

Parameters

`derCert`	A pointer to a user supplied buffer containing the DER bytes representing the X.509 certificate to be used.
`len`	The number of bytes in the supplied buffer.

Return Value Returns `IT_SSL_SUCCESS` if certificate is successfully supplied. Returns `IT_SSL_ERR_FAILURE` otherwise.

See Also
```
IT_SSL::setRSAPrivateKeyFromDER()
IT_SSL::setRSAPrivateKeyFromFile()
IT_SSL::setX509CertFromFile()
```

Note: The function `IT_SSL::setRSAPrivateKeyFromFile()` must be called before calling either `IT_SSL::setX509CertFromDER()` or `IT_SSL::setX509CertFromFile()`, if setting the certificate yourself.

IT_SSL::setX509CertFromFile()

Synopsis
```
int setX509CertFromFile(const char* FileName, IT_Format f);
```

Description The `setSecurityName()` member function automatically retrieves and uses the specified certificate. However, this function allows you to supply the certificate that an OrbixSSL application uses for authentication purposes. The certificate is contained in the supplied file. You must call either `setRSAPrivateKeyFromDER()` or `setRSAPrivateKeyFromFile()` before using this function.

Parameters

`FileName`	The filename where the certificate is held.
`f`	Format of the file. For example: `IT_FMT_PEM` (PEM format). `IT_FMT_DER` (DER encoding).

Return Value Returns `IT_SSL_SUCCESS` if the certificate is successfully supplied. Returns `IT_SSL_ERR_FAILURE` or `IT_SSL_ERR_USING_PRIVATE_KEY` otherwise.

See Also
```
IT_SSL::setRSAPrivateKeyFromDER()
IT_SSL::setRSAPrivateKeyFromFile()
IT_SSL::setX509CertFromDER()
```

Note: The function `IT_SSL::setRSAPrivateKeyFromFile()` must be called before calling either `IT_SSL::setX509CertFromDER()` or `IT_SSL::setX509CertFromFile()`, if setting the certificate yourself.

IT_SSL::specifyCipherSuites()

Synopsis

```
int specifyCipherSuites(const IT_SSLCipherSuites& suite,
                        const unsigned int n, IT_SSLCipherSuites&
                        SetSuite, unsigned int& retn);
```

Description

An application uses this function to customize the set of ciphers that it can use. Specification of the desired set of ciphers to be used is supplied in the `suite` parameter. This function operates within the constraints of the lists of ciphers allowed by the Security Policy configuration file which specifies that any ciphers that are *preferred* or *allowed* can be used.

Ciphers that the application will actually use are returned in the `SetSuite` parameter. If all ciphers specified are disallowed, then the previously existing set of ciphers is used. The set of ciphers for the application is automatically initialized to the `IT_PREFERRED_CIPHERS` list at application start up. This means that this function need not be called by an application unless it wants to restrict or expand its cipher suite.

Parameters

`suite`	A list of ciphers. The application attempts to specify the ciphers according to preference.
`n`	The number of ciphers in `suite`.
`SetSuite`	The list of ciphers that the application will actually use.
`retn`	The number of ciphers in `SetSuite`.

Return Value

If all ciphers specified in the `suite` parameter are allowed, `IT_SSL_SUCCESS` is returned and `SetSuite` is populated accordingly with these ciphers.

If all ciphers specified in the `suite` parameter are disallowed, `IT_SSL_ERR_NO_CIPHERS_ALLOWED` is returned. In this case the previously existing set of ciphers in `SetSuite` is used.

If some, but not all of the ciphers specified in the `suite` parameter are allowed, the ones that are allowed are set in `SetSuite` and `IT_SSL_ERR_NOT_ALL_CIPHERS_ALLOWED` is returned.

See Also

`IT_SSL::getNegotiatedCipherSuite()`

IT_SSL::specifySecurityForInterfaces()

Synopsis

```
int specifySecurityForInterfaces
      (IT_CommsSecuritySpec* SpecList, unsigned int n);
```

Description

This function allows clients and servers acting as clients to explicitly specify particular security requirements for servers. For example, if an otherwise secure client wishes to connect to an insecure server, it can do so using this function.

This function is only applicable when a connection to a server is initially being established. Once a connection to a server has been established, this connection can be used to access other interfaces in that server without reference to the list of specified interfaces. The main use anticipated for this member function is to provide a means to explicitly allow insecure connections to be established through a specified insecure interface.

The structure `IT_CommsSecuritySpec` datatype holds the following members:

```
struct IT_CommsSecuritySpec {
      char* id;
      IT_SecCommsCategory CommsCat;
};
```

The parameter `id` specifies the name of the target interface. The enumerated datatype `CommsCat` indicates whether the interface should be associated with secure or insecure communications. This type is defined as follows:

```
enum IT_SecCommsCategory {
      IT_COMMS_CAT_INSECURE,
      IT_COMMS_CAT_SECURE
};
```

Parameters

`SpecList`	Specifies an array of `IT_CommsSecuritySpec` structures that holds the name of the server in the `id` parameter, and the `CommsCat` parameter.
`n`	Specifies the number of items in the array.

Return Value

Returns `IT_SSL_SUCCESS` if the security setting for the interface is accepted. Returns `IT_SSL_ERR_POLICY_DISALLOWS` if the OrbixSSL configuration file disallows this option.

See Also

`IT_SSL::SpecifySecurityForServers()`

IT_SSL::specifySecurityForServers()

Synopsis

```
int specifySecurityForServers
      (IT_CommsSecuritySpec* SpecList, unsigned int n);
```

Description

This function allows clients and servers acting as clients to explicitly specify particular security requirements for servers. For example, if an otherwise secure client wishes to connect to an insecure server, they may do so using this function.

The structure `IT_CommsSecuritySpec` holds the following members:

```
struct IT_CommsSecuritySpec {
      char* id;
      IT_SecCommsCategory CommsCat;
};
```

The parameter `id` specifies the target server name. The enumerated datatype `CommsCat` indicates whether the server should be associated with secure or insecure communications. It holds the following identifiers that you can assign to servers:

```
enum IT_SecCommsCategory {
      IT_COMMS_CAT_INSECURE,
      IT_COMMS_CAT_SECURE
};
```

Parameters

`SpecList`	Specifies an array of `IT_CommsSecuritySpec` structures that holds the name of the server in the `id` parameter, and the `CommsCat` parameter.
`n`	Specifies the number of items in the array.

Return Value

Returns `IT_SSL_SUCCESS` if security setting for interface is accepted. Returns `IT_SSL_ERR_POLICY_DISALLOWS` if the OrbixSSL configuration file disallows this option.

See Also

`IT_SSL::SpecifySecurityForInterfaces()`

Struct IT_UTCTime

Synopsis

The `IT_UTCTime` structure holds a null-terminated `UTCTime` format string. The `UTCTime` type denotes a "coordinated universal time" or Greenwich Mean Time (GMT) value. A `UTCTime` value includes the local time precise to either minutes or seconds, and an offset from GMT in hours and minutes. It takes any of the following forms:

1. YYMMDDhhmmZ
2. YYMMDDhhmm+hh'mm'
3. YYMMDDhhmm-hh'mm'
4. YYMMDDhhmmssZ
5. YYMMDDhhmmss+hh'mm'
6. YYMMDDhhmmss-hh'mm'

The elements in these time formats are:

- *YY* is the least significant two digits of the year.
- *MM* is the month (01 to 12).
- *DD* is the day (01 to 31).
- *hh* is the hour (00 to 23).
- *mm* are the minutes (00 to 59).
- *ss* are the seconds (00 to 59).
- *Z* indicates that local time is GMT, + indicates that the local time is later than GMT, and - indicates that the local time is earlier than GMT.
- *hh* is the absolute value of the offset from GMT in hours.
- *mm* is the absolute value of the offset from GMT in minutes.

The `UTCTime` type is used for signing times in a PKCS signing-time attribute and for certificate validity periods in the X.509 `Validity` type.

Note: The ISO defines time fields in X.509 certificates as `UTCTime` if the date is before 2051, and as `GeneralisedTime` for later years. If your application is going to do any specific validation of the time fields, it should be aware that 00 for a year means the year 2000.

C++

```
struct IT_UTCTime {
   char TimeString[MAX_UTCTIME_LEN+1];
};
```

Typedef IT_ValidateX509CertCB

Synopsis

```
typedef int (*IT_ValidationX509CertCB)
      (IT_CertValidity ok, IT_X509CertChain& PeerCertChain);
```

Description

`IT_ValidateX509CertCB` is the typedef that defines the user-supplied function passed to `IT_SSL::setValidateClientCertCallback()` or `IT_SSL::setValidateServerCertCallback()`. This function determines whether an OrbixSSL application accepts certificates it receives.

An example function declaration which matches this typedef would be the following:

```
int myValidationFunc(IT_CertValidity ok,
IT_X509CertChain& chain);
```

Parameters

`ok`

Specifies whether OrbixSSL thinks the certificate supplied is valid or not. The values for the `ok` parameter are as follows:

`IT_VALID_YES` indicates that OrbixSSL thinks the certificate is valid. To override OrbixSSL's opinion of the certificate, the application can return `IT_VALID_NO`. Typically, this would be the case if some application level access control checking indicated that the peer was not authorized to connect to this application.

`IT_SSL_VALID_NO_APP_DECESION` indicates that OrbixSSL thinks the certificate is invalid. To override OrbixSSL's opinion of the peer certificate chain and perform its own checking, the application can return `1`. For more information about the nature of the error the application detected, the application can call `IT_X509CertChain::getErrorInfo()`.

`IT_SSL_VALID_NO` indicates that OrbixSSL will not accept the certificate even if the application returns `1`. This could happen if the certificate was in violation of parameters specified by the security policy file, for example if the maximum allowed chain depth is exceeded.

`PeerCertChain`	This parameter is used to obtain information about the peer certificate and its issuer certificates. This information is needed to allow the application to do access checking and logging.

Return Value Returns `1` to indicate acceptance of a certificate. Returns `0` to indicate rejection of a certificate.

See Also
```
IT_X509CertChain()
IT_SSL::setValidateClientCertCallback()
IT_SSL::setValidateServerCertCallback()
IT_X509CertChain::getErrorInfo()
```

Class IT_X509_CRL_Info

Synopsis

A Certificate Revocation List (CRL) is a list of certificates that are no longer valid, even though they have not yet reached their expiry dates. CAs issue CRLs to revoke certificates when the security of those certificates has been compromised or they are no longer in use. Each certificate in the CRL is identified by its unique serial number.

The class `IT_X509_CRL_Info` provides a C++ interface to a CRL.

C++

```
class IT_X509_CRL_Info {
public:
   IT_X509_CRL_Info();
   IT_X509_CRL_Info( const IT_X509_CRL_Info& crl );
   virtual ~IT_X509_CRL_Info();
   virtual IT_X509_CRL_Info& operator=(
      const IT_X509_CRL_Info& crl );

   virtual int getSignatureAlgorithm(IT_OID& oid) const;
   virtual int getVersion(unsigned int& nVer) const;
   virtual int getIssuer(IT_AVAList& lstIssuer) const;
   virtual int getLastUpdate(IT_UTCTime& t) const;
   virtual int getNextUpdate(IT_UTCTime& t) const;
   virtual int getRevokedCerts(IT_X509_RevokedList& r) const;
   virtual int find(const IT_IntegerData& nSerialNum,
      CORBA(Boolean)& bFound, IT_X509_Revoked& r) const;
   virtual int getExtensions(IT_ExtensionList& e) const;
   virtual int fromDER(const char** pData);
   virtual int openFile(const char* file, IT_Format fmt);
};
```

See Also

`IT_CRL_List`
`IT_X509_Revoked`
`IT_X509_RevokedList`

IT_X509_CRL_Info::find()

Synopsis

```
virtual int find(const IT_IntegerData& nSerialNum,
   CORBA(Boolean)& bFound, IT_X509_Revoked& r) const;
```

Description This function allows you to check if a specified certificate is included in the CRL.

Parameters

`nSerialNum`	The serial number of the certificate to be checked.
`bFound`	A boolean value that indicates whether the certificate was found in the CRL. This value is set to `true` if the certificate was found. Otherwise it is set to `false`.
`r`	An object that represents information about the entry in the CRL associated with the specified certificate.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

IT_X509_CRL_Info::fromDERFile()

Synopsis

```
virtual int fromDER(const char** pData);
```

Description Converts CRL information stored in a file in DER format to an `IT_X509_CRL_Info` object.

Parameters This function takes the following parameter:

`pData`	The name of the file containing the CRL information in DER format.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

IT_X509_CRL_Info::getExtensions()

Synopsis `virtual int getExtensions(IT_ExtensionList& e) const;`

Description Returns any X.509 version three extensions that the CRL includes.

Parameters This function takes the following parameter:

`e`	The list of extensions included in the CRL.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

IT_X509_CRL_Info::getIssuer()

Synopsis `virtual int getIssuer(IT_AVAList& lstIssuer) const;`

Description This function returns a distinguished name that identifies the CA that issued the CRL.

Parameters This function takes the following parameter:

`lstIssuer`	Distinguished name identifying the CA that issued the CRL.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

IT_X509_CRL_Info::getLastUpdate()

Synopsis `virtual int getLastUpdate(IT_UTCTime& t) const;`

Description This function returns the time at which the CRL was last updated.

Parameters This function takes the following parameter:

`t`	The time of the last CRL update, represented as an `IT_UTCTime` object.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

See Also `IT_UTCTime`

IT_X509_CRL_Info::getNextUpdate()

Synopsis `virtual int getNextUpdate(IT_UTCTime& t) const;`

Description This function returns the time at which the CA will next update the CRL.

Parameters

`t` The time of the next CRL update, represented as an `IT_UTCTime` object.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

See Also `IT_UTCTime`

IT_X509_CRL_Info::getRevokedCerts()

Synopsis `virtual int getRevokedCerts(IT_X509_RevokedList& r) const;`

Description This function provides access to the revoked certificate information stored in the CRL. The entries in the CRL are returned as an `IT_X509_RevokedList` object.

Parameters

`r` An `IT_X509_RevokedList` object containing information about each entry in the CRL.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in checking the contents of the CRL. Otherwise, it returns an error.

See Also `IT_X509_RevokedList`

IT_X509_CRL_Info::getSignatureAlgorithm()

Synopsis `virtual int getSignatureAlgorithm(IT_OID& oid) const;`

Description This function returns information about the algorithm used to sign the CRL.

Parameters

`oid`	The ASN.1 object identifier associated with the algorithm used to sign the CRL.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in getting the CRL signing algorithm. Otherwise, it returns an error.

See Also `IT_OID`

IT_X509_CRL_Info::getVersion()

Synopsis

```
virtual int getVersion(unsigned int& nVer) const;
```

Description This function returns the X.509 version associated with the CRL.

Parameters This function takes the following parameter:

`nVer`	This parameter specifies which version of X.509 the CRL uses. In accordance with the X.509 specification, a value of `0` indicates version one, a value of `1` indicates version two, and a value of `2` indicates version three.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in getting the X.509 version number. Otherwise, it returns an error.

IT_X509_CRL_Info::openFile()

Synopsis

```
virtual int openFile(const char* file, IT_Format fmt);
```

Description This function allows you to create an `IT_X509_CRL_Info` object that represents a CRL stored in a file.

Parameters

`file`	The name of the file that contains the CRL.
`fmt`	The format in which the CRL is stored in the file. For example: `IT_FMT_PEM` (PEM format). `IT_FMT_DER` (DER encoding).

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in creating the CRL from file. Otherwise, it returns an error.

Class IT_X509_Revoked

Synopsis This class represents a single entry in a Certificate Revocation List (CRL). Each entry specifies a certificate that is no longer valid. An entry includes the certificate serial number and the date at which the certificate was revoked. An entry can also include X.509 version three extensions.

C++

```
class IT_X509_Revoked {
public:
   IT_X509_Revoked();
   IT_X509_Revoked(const IT_X509_Revoked& r);
   virtual ~IT_X509_Revoked();
   IT_X509_Revoked& operator=(const IT_X509_Revoked& r);

   virtual int getSerialNumber(IT_IntegerData& serialNum) const;
   virtual int getRevocationDate(IT_UTCTime& t) const;
   virtual int getExtensions(IT_ExtensionList& e) const;
   virtual int getSequence(int& n) const;
};
```

See Also `IT_CRL_List`
`IT_X509_CRL_Info`
`IT_X509_RevokedList`

IT_X509_CRL_Revoked::getExtensions()

Synopsis

```
virtual int getExtensions(IT_ExtensionList& e) const;
```

Description If an entry in a CRL includes any X.509 version three extensions, you can use this function to retrieve them.

Parameters This function takes the following parameter:

`e`	A list of the extensions included in the revocation record.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in returning the extensions. Otherwise, it returns an error.

See Also `IT_ExtensionList`

IT_X509_CRL_Revoked::getRevocationDate()

Synopsis `virtual int getRevocationDate(IT_UTCTime& t) const;`

Description This function returns the date at which the certificate was revoked.

Parameters This function takes the following parameter:

`t` An `IT_UTCTime` object that represents the certificate revocation date.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in returning the revocation date. Otherwise, it returns an error.

See Also `IT_UTCTime`

IT_X509_CRL_Revoked::getSequence()

Synopsis `virtual int getSequence(int& n) const;`

Description This function returns the position of the revocation record within the CRL from which it was retrieved.

Parameters This function takes the following parameter:

`n` The sequence number of the record in the CRL.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in getting the sequence number. Otherwise, it returns an error.

IT_X509_CRL_Revoked::getSerialNumber()

Synopsis `virtual int getSerialNumber(IT_IntegerData& serialNum) const;`

Description This function returns the serial number that uniquely identifies the revoked certificate.

Parameters This function takes the following parameter:

`serialNum` The certificate serial number.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in returning the serial number. Otherwise, it returns an error.

Class IT_X509_RevokedList

Synopsis This class represents a list of revoked certificate entries extract from a Certificate Revocation List (CRL). Each individual record in this list is stored as an `IT_X509_Revoked` object.

C++

```
class IT_X509_RevokedList {
public:
   IT_X509_RevokedList() { pImpl = NULL; }
   IT_X509_RevokedList( const IT_X509_RevokedList& r );
   virtual ~IT_X509_RevokedList();
   virtual IT_X509_RevokedList& operator=(
      const IT_X509_RevokedList& r);

   virtual unsigned int getCount() const;
   virtual int getRevoked(int nPos, IT_X509_Revoked& r) const;
};
```

See Also

```
IT_CRL_List
IT_X509_CRL_Info
IT_X509_Revoked
```

IT_X509_RevokedList::getCount()

Synopsis

```
virtual unsigned int getCount() const;
```

Description This function returns the number of revoked certificates in the list.

Return Value Returns the number of revoked certificates as an `unsigned int` value.

IT_X509_RevokedList::getRevoked()

Synopsis

```
virtual int getRevoked(int nPos, IT_X509_Revoked& r) const;
```

Description This function returns the revocation record from a specified position in the list.

Parameters

`enPos`	The position in the list at which the required certificate entry is located. The first record is at position zero. The last record is at one below the return value of `IT_X509_RevokedList::getCount()`.
`r`	The certificate revocation record from the specified position.

Return Value This function returns `IT_SSL_SUCCESS` if it succeeds in returning the required record. Otherwise, it returns an error.

Class IT_X509Cert

Synopsis This class provides a high-level interface to an X.509 certificate. A number of member functions are provided to obtain information contained in the certificate. This class, along with other certificate interface classes, shields the OrbixSSL developer from having to know about the low-level details such as the encoding of X.509 certificates; although access to low-level DER information is provided if required.

C++

```
class IT_X509Cert {
public:
   virtual int convert(char* buf, IT_Format f);
   virtual int getExtensions(IT_ExtensionList& e);
   virtual int getIssuer(IT_AVAList& l);
   virtual int getSerialNumber(IT_IntegerData& i);
   virtual int getSubject(IT_AVAList& l);
   virtual int getVersion(unsigned int& ver);
   virtual int length(IT_Format f);
   virtual int getNotAfter(IT_UTCTime& t);
   virtual int getNotBefore(IT_UTCTime& t);
};
```

IT_X509Cert::convert()

Synopsis `virtul int convert(char* buf, IT_Format f);`

Description This function fills the supplied buffer with the requested format of data corresponding to the contents of the X.509 certificate that the `IT_X509Cert` object represents.

Parameters

`buf` The user-supplied buffer that must be of sufficient size to hold the requested conversion. To find the required length of buffer for a particular conversion type, call `IT_X509Cert::length()`.

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`. In this format, `buf` contains a sequence of bytes corresponding to the DER encoding of the X.509 certificate. Typically, you require this option only if you use your own certificate parsing routines.

`IT_FMT_HEX_STRING`. In this format, `buf` contains a null-terminated string which is a formatted hex dump of the DER data of the certificate.

`IT_FMT_INTERNAL`. In this format, `buf` contains the value of a variable of the SSLeay data type `X509 *`.

WARNING: This value provides access to low-level SSL Toolkit data structures, and is non-portable. Code that uses this feature will not work if the underlying SSL toolkit is changed. `IT_FMT_INTERNAL` allows direct access to the low-level SSL toolkit data representation of this AVA, thus allowing the user to directly call the toolkit API. Take extreme care if using this option.

Return Value Returns `IT_SSL_SUCCESS` if the conversion was successful. Otherwise, it returns the error code `IT_SSL_ERR_INVALID_PARAM`.

See Also `IT_X509Cert::length()`

IT_X509Cert::getExtensions()

Synopsis

```
virtual int getExtensions(IT_ExtensionList& e);
```

Description This function retrieves the list of X.509 version three extensions the certificate can include. Individual extensions may be retrieved from the returned `IT_ExtensionList` as `IT_Extension` instances.

Parameters

e	The IT_ExtensionList object to be updated.

Return Value Returns a list of extensions.

IT_X509Cert::getIssuer()

Synopsis

```
virtual int getIssuer(IT_AVAList& retAVAList);
```

Description This function retrieves the distinguished name of the certificate issuer as an IT_AVAList instance. Individual components of the distinguished name (for example, the common name or the organization name) can be retrieved from the IT_AVAList instance.

Parameters

retAVAList	The IT_AVAList object to be updated.

Return Value Returns IT_SSL_SUCCESS or an error code indicating the failure reason.

See Also IT_AVA
IT_AVAList
IT_Extension
IT_ExtensionList

IT_X509Cert::getSerialNumber()

Synopsis

```
virtual int getSerialNumber(IT_IntegerData& i);
```

Description This function obtains the serial number of the certificate.

Parameters

i	The supplied IT_IntegerData object. This object is initialized with the serial number data field of the X.509 certificate.

Return Value Returns IT_SSL_SUCCESS or an error code indicating the failure reason.

See Also class IT_IntegerData

IT_X509Cert::getSubject()

Synopsis

```
virtual int getSubject(IT_AVAList& retAVAList);
```

Description This function retrieves the distinguished name corresponding to the subject field of this certificate as an `IT_AVAList` instance. Individual components of the distinguished name (for example, common name or organization name) can be retrieved from the `IT_AVAList` instance.

Parameters

`retAVAList`	The `AVAList` object to be updated with the subject information.

Return Value Returns `IT_SSL_SUCCESS` or an error code indicating the error reason.

See Also

```
IT_AVA
IT_AVAList
IT_Extension
IT_ExtensionList
```

IT_X509Cert::getVersion()

Synopsis

```
virtual int getVersion(unsigned int& ver);
```

Description This function obtains the version number of the X.509 certificate.

Parameters

`ver`	X.509 version certificate number. In accordance with the X.509 specification, a value of `0` indicates version one, a value of `1` indicates version two, and a value of `2` indicates version three.

Return Value Returns `IT_SSL_SUCCESS` if an X.509 version of the certificate is successfully returned. Otherwise, it returns `IT_SSL_FAILURE`.

IT_X509Cert::length()

Synopsis

```
virtual int length(IT_Format f);
```

Description This function is used to calculate how much storage is required to hold the result of a call to `IT_X509Cert::convert()` for a particular `IT_Format` value. Refer to `IT_X509Cert::convert()` for a list of the supported `IT_Format` values.

Parameters

`f` The format of the required conversion. The following `IT_Format` values are supported:

`IT_FMT_DER`

`IT_FMT_HEX_STRING`

`IT_FMT_INTERNAL`

Return Value Returns the number of bytes required to store the result of the conversion; returns minus `1` if the required conversion is not supported.

IT_X509Cert::getNotAfter

Synopsis

```
virtual int getNotAfter(IT_UTCTime& t);
```

Description This function is used to extract the `notAfter` field from an X.509 certificate. This field is used in determining the date validity of a certificate in conjunction with the `notBefore` field. A certificate can be specified as not being valid until after some point in the future. The field is returned in the supplied `IT_UTCTime` variable which is passed as a parameter to the function.

Parameters

`t` The `IT_UTCTime` object to be updated with the `notAfter` field of this X.509 certificate.

Return Value Returns `IT_SSL_SUCCESS` if successful. Otherwise, it returns an error code indicating the failure reason.

See Also `IT_X509Cert::getNotBefore()`
`IT_UTCTime`

IT_X509Cert::getNotBefore()

Synopsis

```
virtual int getNotBefore(IT_UTCTime& t);
```

Description

This function is used to extract the `notBefore` field from an X.509 certificate. This field is used in determining the date validity of a certificate in conjunction with the `notAfter` field. A certificate can be specified as not being valid until some point in the future. The field is returned in the supplied `IT_UTCTime` variable which is passed as a parameter to the function.

Parameters

`t` The `notBefore` field in an X.509 certificate.

Return Value

Returns `IT_SSL_SUCCESS` if successful, or an error code indicating the failure reason.

See Also

`IT_X509Cert::getNotAfter()`
`IT_UTCTime`

Class IT_X509CertChain

Synopsis An instance of this class is supplied as a parameter to a certificate validation function. Using this class, you can obtain each certificate in a certificate chain. The first certificate in the chain is the certificate associated with the application that supplied the chain. This is referred to as the *peer certificate*. Each subsequent certificate is the issuer certificate for the previous one.

```
class IT_X509CertChain {
public:
      IT_X509CertChain();

      virtual unsigned int numCerts();
      virtual int getCert(unsigned int pos, IT_X509Cert& ret);
      virtual int getErrorInfo(IT_CertError& retErr);
      virtual int getCurrentCert(IT_X509Cert& ret);
      virtual int getCurrentDepth();
};
```

See Also `IT_SSL::setValidateClientCertCallback()`
`IT_SSL::setValidateServerCertCallback()`
`IT_ValidateX509CertCB`

IT_X509CertChain::getCert()

Synopsis
```
virtual int getCert(unsigned int pos, IT_X509Cert& ret);
```

Description This function obtains the certificate at the specified index in the chain.

Parameters

`pos`	The index position in the chain of the required certificate. The index ranges in value from `0` to the number of certificates minus `1`.
`ret`	The certificate that is returned.

Return Value Returns `IT_SSL_SUCCESS` or an error code indicating the failure reason.

See Also
```
IT_SSL::getCurrentCert()
IT_SSL::getCurrentDepth()
IT_SSL::getErrorInfo()
IT_SSL::setClientCertCallback()
IT_SSL::setServerCertCallback()
```

IT_X509CertChain::getCurrentCert()

Synopsis
```
virtual int getCurrentCert(IT_X509Cert& ret);
```

Description This function returns the current certificate in the certificate chain that is being validated. Functionally, this is equivalent to `getCert(getCurrentDepth())`.

Parameters

`ret`	The certificate currently being validated.

Return Value Returns `IT_SSL_SUCCESS` if it succeeds in returning the certificate. Otherwise, it returns an error code indicating the failure reason.

See Also
```
IT_SSL::getCert()
IT_SSL::getCurrentDepth()
IT_SSL::getErrorInfo()
IT_SSL::setValidateClientCertCallback()
IT_SSL::setValidateServerCertCallback()
```

IT_X509CertChain::getCurrentDepth()

Synopsis
```
virtual int getCurrentDepth();
```

Description This function returns a value between `0` and the number of certificates minus one to indicate which certificate is currently being validated in the certificate chain.

Return Value Returns `IT_SSL_SUCCESS` or an error code indicating the failure reason.

See Also
```
IT_SSL::getCert()
IT_SSL::getCurrentCert()
IT_SSL::setValidateServerCertCallback()
IT_SSL::setValidateClientCertCallback()
```

IT_X509CertChain::getErrorInfo()

Synopsis

```
virtual int getErrorInfo(IT_CertError& retErr);
```

Description This function allows you to get error information associated with an `IT_X509CertChain`. This may be useful during certificate validation, for example when a value of `IT_SSL_VALID_NO` or `IT_SSL_VALID_NO_APP_DECESION` is supplied to the certificate validation function registered by your application.

Parameters

`retErr`	The `IT_CertError` structure that contains the error information.

Return Value Returns `IT_SSL_SUCCESS` or an error code indicating the failure reason.

See Also

```
IT_SSL::getCert()
IT_SSL::getCurrentCert()
IT_SSL::getCurrentDepth()
IT_SSL::setClientCertValidationCB()
IT_SSL::setServerCertValidationCB()
IT_SSL::setValidateServerCertCallback()
IT_SSL::setValidateClientCertCallback()
ITCertError
IT_ValidateX509CertCB
```

IT_X509CertChain::numCerts()

Synopsis

```
virtual unsigned int numCerts();
```

Description This function returns the number of certificates in the chain represented by the `IT_X509CertChain` object.

Return Value Returns the number of certificates in the chain.

Part V

Appendices

Appendix A
Security Recommendations

Some general recommendations for increasing the security of OrbixSSL applications are as follows:

- Use SSL security for every application where possible. This means specifying `SECURE_DAEMON` as your daemon policy, and using the default invocation policy for all OrbixSSL applications. Under these conditions, no unauthorized applications can access your servers or be accessed by your applications.
- Replace the demonstration certificates that are installed with OrbixSSL. These must be replaced by a set of certificates and private keys that have been securely generated. Refer to Chapter 4 on page 51 for more information.

  You should also change the pass phrases used to protect private keys. Do not reuse the pass phrases that were used for the example private keys.
- Do not set the `IT_ENABLE_DEFAULT_CERT` configuration variable, and do not issue a default certificate for live systems.

  The use of a default certificate is generally not appropriate in a production system because access to the dynamic library of the OrbixSSL version installed on the system would allow any client to use the default certificate, even a client from another machine. The OrbixSSL dynamic libraries in effect contain the default pass phrase that protects the private key of the default certificate. The default value for the configuration variable `IT_ENABLE_DEFAULT_CERT` is `FALSE`.
- If your application requires some interoperability with insecure applications, only allow specifically listed servers and interfaces to be contacted insecurely by your clients. Use secure callbacks for clients wherever possible as this is the default setting for OrbixSSL.

- Where it is necessary for remote insecure clients to contact OrbixSSL servers that are capable of accepting secure and insecure connections, set the daemon policy to `RESTRICTED_SEMI_SECURE_DAEMON` (instead of `SEMI_SECURE_DAEMON`).
- The OrbixSSL installation modifies the existing Orbix binaries so that they can use the Orbix binary certificate for authentication purposes. The permissions on these binaries are readable only by `root`, but executable by everybody. Do not change the permissions to be readable by everybody.
- Use the 128 bit or triple DES cipher suites exclusively where possible. The extra time taken to perform the more secure bulk cipher computations does not impact the overall performance of OrbixSSL applications significantly.

  If some applications require export-level cryptography for interoperability with other applications, use the `IT_ALLOWED_CIPHERS` configuration variable and set the use of the export ciphers in the required applications explicitly. This is preferable to adding the export ciphers to all applications (by adding export ciphers to the `IT_PREFERRED_CIPHERS` configuration variable) which would effectively result in reducing the security of all applications that did not explicitly specify their own requirements.

  The security of an SSL application is only as strong as the weakest cipher suite that it is prepared to support. Consider the presence of stronger cipher suites as an optional service for more discerning applications that wish to communicate with your application.
- An RSA key size of at least 1024 bits is recommended for most secure applications. 1024 bit keys are significantly slower to use than 512 bit keys but they greatly increase the security of systems. The use of SSL session caching helps to minimize the number of public key computations.

Appendix B
OrbixSSL Configuration Variables

The OrbixSSL configuration file, `orbixssl.cfg`, uses configuration variables to specify a security policy for your applications. The following OrbixSSL configuration variables are available:

- `IT_ALLOWED_CIPHERSUITES`

  This configuration variable defines a list of ciphers, supplemental to those defined by `IT_CIPHERSUITES`, that applications can optionally choose to support. Refer to "Configuring Ciphers" on page 46 for more information.

- `IT_AUTHENTICATE_CLIENTS`

  Setting this value to `TRUE` enforces client authentication in all servers affected by the configuration scope. This value does not override `IT_SERVERS_MUST_AUTHENTICATE_CLIENTS`.

- `IT_BIDIRECTIONAL_IIOP_BY_DEFAULT`

  This configuration variable is principally supplied to facilitate the migration of single-threaded Orbix programs that are the recipients of callbacks. Single-threaded clients using the IIOP protocol run the risk of encountering deadlock if callbacks are used. The use of bidirectional IIOP, however, resolves this issue.

  Setting this configuration variable to `TRUE` enables bidirectional IIOP support in all OrbixSSL servers and clients. This is directly equivalent to calling the Orbix function `supportBidirectionalIIOP()` with a true parameter value. For more information on the use of bidirectional IIOP, refer to the *Orbix C++ Programmer's Guide*.

  OrbixSSL calls `supportBidirectionalIIOP()` when the application calls `IT_SSL::init()`. After this call, calling `supportBidirectionalIIOP()` overrides the configuration setting.

- IT_CA_LIST_FILE

  This configuration variable gives the fully qualified file name of the file that contains certificates of all trusted certificate authorities (CAs). OrbixSSL uses this file when validating a certificate. The certificates of all trusted CAs are concatenated into this single file, in PEM format.

- IT_CACHE_OPTIONS

  This variable configures the use of SSL session caching in OrbixSSL programs. Refer to "OrbixSSL Session Caching Configuration" on page 47 for more information.

- IT_CERTIFICATE_FILE

  This variable specifies the fully qualified file name of the certificate associated with a program. Usually, this is specified relative to IT_CERTIFICATE_PATH, for example:

  ```
  IT_CERTIFICATE_FILE =
     IT_CERTIFICATE_PATH + "demo/demo_server";
  ```

- IT_CERTIFICATE_PATH

  This configuration variable gives the location of the directory used to store certificate files.

- IT_CHECKSUMS_ENABLED

  This configuration variable enables the use of cryptographic checksums for servers that use the server key distribution mechanism (KDM). Refer to "Verifying the Integrity of Server Executables" on page 76 for more information.

- IT_CHECKSUM_REPOSITORY

  This configuration variable specifies the location in which OrbixSSL stores checksums calculated for servers that use the KDM. Refer to "Verifying the Integrity of Server Executables" on page 76 for more information.

- IT_CIPHERSUITES

  This configuration variable determines the default list of ciphers that an OrbixSSL application uses. Refer to "Configuring Ciphers" on page 46 for more information.

- IT_CRL_ENABLED

  Setting this variable to TRUE instructs OrbixSSL to check certificate revocation lists (CRLs) during authentication.

- IT_CRL_REPOSITORY

  This variable specifies the fully qualified path of the directory used to store CRLs.

- IT_CRL_UPDATE_INTERVAL

  This variable specifies the time period, in seconds, between checking the CRLs for updates.

- IT_DAEMON_AUTHENTICATES_CLIENTS

  This variable controls whether the Orbix daemon authenticates SSL-enabled clients or not. For more information, refer to "Configuring the Orbix Daemon to Authenticate Clients" on page 43.

- IT_DAEMON_POLICY

  This variable specifies the types of communication accepted by the Orbix daemon. Refer to "Configuring a Restricted Semi-Secure Daemon" on page 43 for more information.

- IT_DAEMON_UNRESTRICTED_METHODS

  This variable applies only when IT_DAEMON_POLICY is RESTRICTED_SEMI_SECURE_DAEMON. It specifies a comma-separated list of insecure operations supported by this type of daemon. Refer to "Configuring a Restricted Semi-Secure Daemon" on page 43 for more information.

- IT_DEFAULT_MAX_CHAIN_DEPTH

  This configuration variable sets the maximum chain depth accepted by programs affected by the configuration scope. This value does not override IT_MAX_ALLOWED_CHAIN_DEPTH.

- IT_DISABLE_SSL

  Setting this variable to TRUE disables SSL security in all programs affected by the configuration scope. The default value for this variable is FALSE.

- IT_ENABLE_DEFAULT_CERT

  This configuration variable adds SSL security to programs that contain no OrbixSSL code. It allows these applications to use the OrbixSSL demos/default certificate. Take extreme care when setting this value to TRUE. It can disrupt running applications and is completely insecure. This variable is useful only for testing purposes, for example to quickly identify an application's performance characteristics using SSL.

- IT_FILTER_BAD_CONNECTS_BY_DEFAULT

  When set to TRUE, this variable has the effect of calling the Orbix function filterBadConnectAttempts() with a true parameter value, in all affected OrbixSSL programs. Orbix applications that do not call this function terminate unless they correctly handle an exception thrown when a client connection attempt fails. For example, this would be the case if a secure server was contacted by an insecure client. This configuration variable has no effect when set to FALSE, which is the default value.

  OrbixSSL calls filterBadConnectAttempts() when the application calls IT_SSL::init(). After calling IT_SSL::init(), you can override the configuration value by calling filterBadConnectAttempts().

- IT_INSECURE_REMOTE_INTERFACES

  This variable allows you to specify the list of remote interfaces that a program with invocation policy IT_SPECIFIED_INSECURE_CONNECT can contact without using SSL. Refer to "Configuring OrbixSSL Application Types" on page 88 for more information.

- IT_INSECURE_SERVERS

  This variable allows you to specify the list of remote servers that a program with invocation policy IT_SPECIFIED_INSECURE_CONNECT can contact without using SSL. Refer to "Configuring OrbixSSL Application Types" on page 88 for more information.

- IT_INVOCATION_POLICY

  This variable sets the invocation policy associated with an application. It accepts a comma-separated list of the policy settings described in "Configuring OrbixSSL Application Types" on page 88.

- IT_KDM_CLIENT_COMMON_NAMES

  The server key distribution mechanism (KDM) always uses client authentication. Only the Orbix daemon and the utility putkdm should be able to communicate with the KDM directly. This variable allows you to specify the common names used by the daemon and putkdm, so that the KDM can authenticate them successfully.

- IT_KDM_ENABLED

  Setting this variable to TRUE enables use of the KDM in all OrbixSSL servers.

- IT_KDM_PIPES_ENABLED

  When the Orbix daemon transfers a pass phrase from the KDM to a server, it can do so in one of two ways: using the server environment, or using an operating system pipe. Setting this value to TRUE enables the use of pipes. The default value is FALSE.

- IT_KDM_REPOSITORY

  If using the KDM, you must set this value to the absolute path of the directory in which the KDM stores information about private key pass phrases for servers.

- IT_KDM_SERVER_PORT

  This variable specifies the port number on which the KDM server listens for incoming communications.

- IT_MAX_ALLOWED_CHAIN_DEPTH

  This configuration variable sets the absolute maximum chain depth that programs can choose to accept. This value limits the possible values that you can set for IT_DEFAULT_MAX_CHAIN_DEPTH.

- IT_ORBIX_BIN_SERVER_POLICY

  This configuration variable allows you to control the communications used by server binaries installed with Orbix.

- IT_SECURE_REMOTE_INTERFACES

  This variable allows you to specify the list of remote interfaces that a program with invocation policy IT_SPECIFIED_SECURE_CONNECT can contact without using SSL. Refer to "Configuring OrbixSSL Application Types" on page 88 for more information.

- IT_SECURE_SERVERS

  This variable allows you to specify the list of remote servers that a program with invocation policy IT_SPECIFIED_SECURE_CONNECT can contact without using SSL. Refer to "Configuring OrbixSSL Application Types" on page 88 for more information.

- IT_SERVERS_MUST_AUTHENTICATE_CLIENTS

  Setting this value to TRUE forces client authentication in all servers affected by the configuration scope. This value cannot be overridden by IT_AUTHENTICATE_CLIENTS.

Appendix C
SSLeay Utilities

OrbixSSL ships a version of the `ssleay` program that is available with Eric Young's SSLeay 0.8.1b package. SSLeay is a publicly available implementation of the SSL protocol. Consult the `SSLeay.cpr` file that is provided with OrbixSSL for information about the copyright terms of SSLeay.

The `ssleay` program consists of a large number of utilities that have been combined into one program. This appendix describes how you use the `ssleay` program with OrbixSSL when managing X.509 certificates and private keys.

A number of examples using `ssleay` commands are described in Chapter 4, "Managing Certificates". Read Chapter 4 before consulting this appendix.

This appendix describes four `ssleay` utility commands:

`x509`	Manipulates X.509 certificates.
`req`	Creates and manipulates certificate signing requests, and self-signed certificates.
`rsa`	Manipulates RSA private keys.
`ca`	Implements a Certification Authority (CA).

Using SSLeay Utilities

An `ssleay` utility command line takes the following form:

```
ssleay command arguments
```

For example:

```
ssleay x509 -in OrbixCA -text
```

Each command is individually described in this appendix. To get a list of the arguments associated with a particular command, use the `-help` option as follows:

```
ssleay command -help
```

For example:

```
ssleay x509 -help
```

The x509 Utility Command

In OrbixSSL the `x509` utility command is mainly used for:

- Printing text details of certificates you wish to examine.
- Converting certificates to different formats.

The options supported by the ssleay `x509` utility command are as follows:

```
-inform arg       - input format - default PEM
                  (one of DER, NET or PEM)
-outform arg      - output format - default PEM
                  (one of DER, NET or PEM
-keyform arg      - private key format - default PEM
-CAform arg       - CA format - default PEM
-CAkeyform arg    - CA key format - default PEM
-in arg           - input file - default stdin
 -out arg          - output file - default stdout
-serial           - print serial number value
-hash             - print serial number value
-subject          - print subject DN
```

```
-issuer           - print issuer DN
-startdate        - notBefore field
-enddate          - notAfter field
-dates            - both Before and After dates
-modulus          - print the RSA key modulus
-fingerprint      - print the certificate fingerprint
 -noout           - no certificate output
-days arg         - How long till expiry of a signed certificate
                  - def 30 days
-signkey arg      - self sign cert with arg
-x509toreq        - output a certification request object
-req              - input is a certificate request, sign and
                  output
-CA arg           - set the CA certificate, must be PEM format
-CAkey arg        - set the CA key, must be PEM format. If
                   missing it is assumed to be in the CA file
-CAcreateserial   - create serial number file if it does not
                  exist
-CAserial         - serial file
-text             - print the certificate in text form
-C                - print out C code forms
-md2/-md5/-sha1/  - digest to do an RSA sign with
-mdc2
```

Using the x509 Utility Command

To print the text details of an existing PEM-format X.509 certificate, use the `x509` utility command as follows:

```
ssleay x509 -in MyCert.pem -inform PEM -text
```

To print the text details of an existing DER-format X.509 certificate, use the `x509` utility command as follows:

```
ssleay x509 -in MyCert.der -inform DER -text
```

To change a certificate from PEM format to DER format, use the `x509` utility command as follows:

```
ssleay x509 -in MyCert.pem -inform PEM -outform
DER -out MyCert.der
```

The req Utility Command

The `req` utility command is used to generate a self-signed certificate or a certificate signing request (CSR). A CSR contains details of a certificate to be issued by a CA. When creating a CSR, the `req` command prompts you for the necessary information from which a certificate request file and an encrypted private key file are produced. The certificate request is then submitted to a CA for signing.

If the `-nodes` (no DES) parameter is not supplied to `req`, you are prompted for a pass phrase which will be used to protect the private key.

Note: It is important to specify a validity period (using the `-days` parameter). If the certificate expires, applications that are using that certificate will not be authenticated successfully.

The options supported by the ssleay `req` utility command are as follows:

-inform arg	input format - one of DER TXT PEM
-outform	arg output format - one of DER TXT PEM
-in arg	inout file
-out arg	output file
-text	text form of request
-noout	do not output REQ
-verify	verify signature on REQ
-modulus	RSA modulus
-nodes	do not encrypt the output key
-key file	use the private key contained in file
-keyform arg	key file format
-keyout arg	file to send the key to
-newkey rsa:bits	generate a new RSA key of 'bits' in size
-newkey dsa:file	generate a new DSA key, parameters taken from CA in 'file'
-[digest]	Digest to sign with (md5, sha1, md2, mdc2)
-config file	request template file
-new	new request
-x509	output an x509 structure instead of a certificate req. (Used for creating self signed certificates)
-days	number of days an x509 generated by -x509 is valid for
-asn1-kludge	Output the 'request' in a format that is wrong but some CA's have been reported as requiring [It is now always turned on but can be turned off with -no-asn1-kludge]

Using the req Utility Command

To create a self signed certificate with an expiry date a year from now, the `req` utility command can be used as follows to create the certificate `CA_cert.pem` and the corresponding encrypted private key file `CA_pk.pem`:

```
ssleay req -config ssl_conf_path_name -days 365
-out CA_cert.pem -new -x509 -keyout CA_pk.pem
```

This following command creates the certificate request `MyReq.pem` and the corresponding encrypted private key file `MyEncryptedKey.pem`:

```
ssleay req -config ssl_conf_path_name -days 365
-out MyReq.pem -new -keyout MyEncryptedKey.pem
```

The rsa Utility Command

The `rsa` command is a useful utility for examining and modifying RSA private key files. Generally RSA keys are stored encrypted with a symmetric algorithm using a user-supplied pass phrase. The SSLeay `req` command prompts the user for a pass phrase in order to encrypt the private key. By default, `req` uses the triple DES algorithm. The `rsa` command can be used to change the password that protects the private key and to convert the format of the private key. Any `rsa` command that involves reading an encrypted `rsa` private key will prompt for the PEM pass phrase used to encrypt it.

The options supported by the ssleay `rsa` utility command are as follows:

`-inform arg`	`input format - one of DER NET PEM`
`-outform arg`	`output format - one of DER NET PEM`
`-in arg`	`inout file`
`-out arg`	`output file`
`-des`	`encrypt PEM output with cbc des`
`-des3`	`encrypt PEM output with ede cbc des using 168 bit key`
`-text`	`print the key in text`
`-noout`	`do not print key out`
`-modulus`	`print the RSA key modulus`

Using the rsa Utility Command

Converting a private key to PEM format from DER format involves using the `rsa` utility command as follows:

```
ssleay rsa -inform DER -in MyKey.der -outform PEM
-out MyKey.pem
```

Changing the pass phrase which is used to encrypt the private key involves using the `rsa` utility command as follows:

```
ssleay rsa -inform PEM -in MyKey.pem -outform PEM
-out MyKey.pem -des3
```

Removing encryption from the private key (which is not recommended) involves using the `rsa` command utility as follows:

```
ssleay rsa -inform PEM -in MyKey.pem -outform PEM
-out MyKey2.pem
```

Note: Do not specify the same file for the `-in` and `-out` parameters, because this may corrupt the file.

The ca Utility Command

You can use the `ca` command to create X.509 certificates by signing existing signing requests. It is imperative that you check the details of a certificate request before signing. Your organization should have a policy with respect to the issuing of certificates. Before implementing CAs, refer to Chapter 4 for more information.

The `ca` command is used to sign certificate requests thereby creating a valid X.509 certificate which can be returned to the request submitter. It can also be used to generate Certificate Revocation Lists (CRLS). For information on the `ca -policy` and `-name` options, refer to "The SSLeay configuration file" on page 216.

To create a new CA using the ssleay `ca` utility command, two files (`serial` and `index.txt`) need to be created in the location specified by the SSLeay configuration file that you are using.

The options supported by the SSLeay `ca` utility command are as follows:

```
-verbose          - Talk alot while doing things
-config file      - A config file
-name arg         - The particular CA definition to use
-gencrl           - Generate a new CRL
-crldays days     - Days is when the next CRL is due
-crlhours hours   - Hours is when the next CRL is due
-days arg         - number of days to certify the certificate for
-md arg           - md to use, one of md2, md5, sha or sha1
-policy arg       - The CA 'policy' to support
-keyfile arg      - PEM private key file
-key arg          - key to decode the private key if it is
                    encrypted
-cert             - The CA certificate
-in file          - The input PEM encoded certificate request(s)
-out file         - Where to put the output file(s)
-outdir dir        - Where to put output certificates
-infiles....      - The last argument, requests to process
```

```
-spkac file       - File contains DN and signed public key and
                  challenge
-preserveDN       - Do not re-order the DN
-batch            - Do not ask questions
-msie_hack         - msie modifications to handle all thos
                  universal strings
```

Note: Most of the above parameters have default values as defined in `ssleay.cnf`.

Using the ca Utility Command

Converting a private key to PEM format from DER format involves using the `ca` utility command as shown in the following example. To sign the supplied CSR `MyReq.pem` to be valid for 365 days and create a new X.509 certificate in PEM format, use the `ca` utility as follows:

```
ssleay ca -config ssl_conf_path_name -days 365
-in MyReq.pem -out MyNewCert.pem
```

The SSLeay configuration file

A number of SSLeay commands (for example, `req` and `ca`) take a `-config` parameter that specifies the location of the SSLeay configuration file. This section provides a brief description of the format of the configuration file and how it applies to the `req` and `ca` commands. An example configuration file is listed at the end of this section.

The `ssleay.cnf` configuration file consists of a number of sections that specify a series of default values which are used by the SSLeay commands.

[req] Variables

The `req` section contains the following settings:

```
default_bits = 1024
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
```

The `default_bits` setting is the default RSA key size that you wish to use. Other possible values are 512, 2048, 4096.

The `default_keyfile` value is default name for the private key file created by `req`.

The `distinguished_name` value specifies the section in the configuration file that defines the default values for components of the distinguished name field. The `req_attributes` variable specifies the section in the configuration file that defines defaults for certificate request attributes.

[ca] Variables

You can configure the file ssleay.cnf to support a number of CAs that have different policies for signing CSRs. The -name parameter to the ca command specifies which CA section to use. For example:

```
ssleay ca -name MyCa ...
```

This command refers to the CA section [MyCa]. If -name is not supplied to the ca command, the CA section used is the one indicated by the default_ca variable. In the "Example ssleay.cnf File" on page 219, this is set to CA_default (which is the name of another section listing the defaults for a number of settings associated with the ca command). Multiple different CAs can be supported in the configuration file, but there can be only one default CA.

Possible [ca] variables include the following:

```
dir:            The location for the CA database

The database is a simple text database containing
the following tab separated fields

status:         A value of 'R' - revoked, 'E'
                -expired or 'V' valid
issued date:    When the certificate was
                certified
revoked date:   When it was revoked, blank if not
                revoked
serial number:  The certificate serial number
certificate:    Where the certificate is located
CN:             The name of the certificate

The serial field should be unique as should the
CN/status combination. The ca program checks these
at startup.

certs:          This is where all the previously
                issued certificates are kept
```

[policy] Variables

The policy variable specifies the default policy section to be used if the `-policy` argument is not supplied to the `ca` command. The CA policy section of a configuration file identifies the requirements for the contents of a certificate request which must be met before it is signed by the CA.

There are 2 policies defined in the "Example ssleay.cnf File" on page 219: `policy_match` and `policy_anything`.

Consider the following value:

```
countryName = match
```

This means that the country name must match the CA certificate.

Consider the following value:

```
organisationalUnitName  = optional
```

This means that the `organisationalUnitName` does not have to be present.

Consider the following value:

```
commonName = supplied
```

This means that the `commonName` must be supplied in the certificate request.

The `policy_match` section of the example `ssleay.cnf` file specifies the order of the attributes in the generated certificate as follows:

```
countryName
stateOrProvinceName
organizationName
organizationalUnitName
commonName
emailAddress
```

Example ssleay.cnf File

```
####################################################################
# SSLeay example configuration file.
# This is mostly used for generation of certificate requests.
####################################################################
[ ca ]
default_ca    = CA_default      # The default ca section
####################################################################

[ CA_default ]

dir       =/opt/iona/OrbixSSL1.0c/certs # Where everything is kept

certs               = $dir # Where the issued certs are kept
crl_dir             = $dir/crl # Where the issued crl are kept
database            = $dir/index.txt # database index file
new_certs_dir       = $dir/new_certs # default place for new certs
certificate         = $dir/CA/OrbixCA # The CA certificate
serial              = $dir/serial # The current serial number
crl                 = $dir/crl.pem # The current CRL
private_key         = $dir/CA/OrbixCA.pk # The private key
RANDFILE            = $dir/.rand # private random number file
default_days        = 365 # how long to certify for
default_crl_days    = 30 # how long before next CRL
default_md          = md5 # which message digest to use
preserve            = no # keep passed DN ordering

# A few different ways of specifying how closely the request should
# conform to the details of the CA

policy                  = policy_match

# For the CA policy [policy_match]

countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

```
# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types

[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
default_bits            = 1024
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
attributes              = req_attributes

[ req_distinguished_name ]
countryName             = Country Name (2 letter code)
countryName_min         = 2
countryName_max         = 2
stateOrProvinceName     = State or Province Name (full name)
localityName            = Locality Name (eg, city)
organizationName        = Organization Name (eg, company)
organizationalUnitName  = Organizational Unit Name (eg, section)
commonName              = Common Name (eg. YOUR name)
commonName_max          = 64
emailAddress            = Email Address
emailAddress_max        = 40

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min   = 4
challengePassword_max   = 20
unstructuredName        = An optional company name
```

Appendix D
Performance Characteristics of the SSL Protocol

This appendix describes the performance characteristics of the SSL toolkit that OrbixSSL C++ uses, in order to give an understanding of the nature of the performance characteristics of SSL applications. This information is provided by the author of SSLeay, Eric Young of CryptSoft Pty Ltd.

The program used to generate the results detailed in this chapter is a version of `ssl/ssltest.c` which is the SSLeay SSL protocol testing program. It is a single process that talks to both sides of the SSL protocol using a non-blocking memory buffer interface.

The results displayed in this chapter were generated using SSLeay 0.8.2 on a pentium pro 200, running Linux. They give an indication of the SSL protocol and encryption overheads.

The results are as follows:

```
proto cipher         number      bytes    new         reuse
SSLv2 RC4-MD5        1000 x          1   12.83s       0.70s
SSLv3 NULL-MD5       1000 x          1   14.35s       1.47s
SSLv3 RC4-MD5        1000 x          1   14.46s       1.56s
SSLv3 RC4-MD5 1024   1000 x          1   51.93s       1.62s
SSLv3 RC4-SHA        1000 x          1   14.61s       1.83s
SSLv3 DES-CBC-SHA    1000 x          1   14.70s       1.89s
SSLv3 DES-CBC3-SHA   1000 x          1   15.16s       2.16s
SSLv2 RC4-MD5        1000 x       1024   13.72s       1.27s
SSLv3 NULL-MD5       1000 x       1024   14.79s       1.92s
SSLv3 RC4-MD5 1024   1000 x       1024   52.58s       2.29s
SSLv3 RC4-SHA        1000 x       1024   15.39s       2.67s
SSLv3 DES-CBC-SHA    1000 x       1024   16.45s       3.55s
SSLv3 DES-CBC3-SHA   1000 x       1024   18.21s       5.38s
SSLv2 RC4-MD5        1000 x      10240   18.97s       6.52s
SSLv3 NULL-MD5       1000 x      10240   17.79s       5.11s
SSLv3 RC4-MD5        1000 x      10240   20.25s       7.90s
SSLv3 RC4-MD5 1024   1000 x      10240   58.26s       8.08s
SSLv3 RC4-SHA        1000 x      10240   22.96s      11.44s
SSLv3 DES-CBC-SHA    1000 x      10240   30.65s      18.41s
SSLv3 DES-CBC3-SHA   1000 x      10240   47.04s      34.53s
SSLv2 RC4-MD5        1000 x     102400   70.22s      57.74s
SSLv3 NULL-MD5       1000 x     102400   43.73s      31.03s
SSLv3 RC4-MD5        1000 x     102400   71.32s      58.83s
SSLv3 RC4-MD5 1024   1000 x     102400  109.66s      59.20s
SSLv3 RC4-SHA        1000 x     102400   95.88s      82.21s
SSLv3 DES-CBC-SHA    1000 x     102400  173.22s     160.55s
SSLv3 DES-CBC3-SHA   1000 x     102400  336.61s     323.82s
```

Interpreting the Data

The first two columns identify the protocol and cipher used.

The next column (*number*) is the number of connections being made. The column *bytes* is the number of bytes exchanged between the client and server side of the protocol. This is the number of bytes that the client sends to the server, and the server returns. Because this all happens in one process, the data is encrypted, decrypted, encrypted and then decrypted again; it is a round trip of

that number of bytes. Because one process performs both the client and server sides of the protocol, this number is multiplied by four to generate the number of bytes encrypted, decrypted, and for which a MAC is calculated.

The `new` column lists the time value. That is, the number of seconds that have elapsed since initiation of a full SSL handshake. The `reuse` column lists the cost of one full handshake; the rest is session-id reuse.

For a server and client using RC4-MD5 and a 512bit server key, with no session-id reuse and a transfer size of 10240 bytes, a pentium pro 200 running linux can handle the SSLv3 protocol overheads of about 49 connections a second; and 126 with session-id reuse.

In comparison, using `s_server` and `s_client` (simple TCP programs) on the same machine, about 44 new connections a second can be handled; and 49 with reuse. Using a 1024 bit key, this drops to 16 new connections; and 49 with reuse. In theory, the limit is 17. TCP problems begin to diminish with larger CPU burdens. (When calculating these results, clients and servers were all on the same host.)

Important points to be aware of when interpreting this data:

- The *new* column is for 1000 full SSL handshakes. The *reuse* column is for 1 full ssl handshake and 999 with session-id reuse. The RSA overheads for each exchange are one public and one private operation, but the protocol/MAC/cipher cost is similar in both the client and server.
- The time measured is user time, but these numbers are not exact.
- Remember this is the cost of both client and server sides of the protocol.
- The TCP round trip latencies, while slowing individual connections, has a minimal impact on throughput.
- In the test program, instead of sending a 102400 byte buffer, an 8K buffer is sent until the required number of bytes is processed.
- The SSLv3 connections are SSLv2-compatible SSLv3 headers.
- A 512 bit server key is used except where noted.
- No server key verification is performed on the client side of the protocol. This would slow things down very little.
- The library used is SSLeay 0.8.x. Using different cipher libraries or SSL implementations will modify these times.

- The normal measuring system consisted of commands as follows:

```
time ./ssltest -num 1000 -bytes 102400 -cipher
DES-CBC-SHA -reuse
```

Performance of Ciphers in SSLeay

The general cipher performance results for the same platform are as follows (under Windows NT, they are significantly higher because Visual C is a much better x86 compiler than `gcc`):

```
SSLeay 0.8.2a 04-Sep-1997
built on Fri Sep  5 17:37:05 EST 1997
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long)
idea(int)
blowfish(ptr2)
C flags:gcc -DL_ENDIAN -DTERMIO -O3 -fomit-frame-pointer -m486
The 'numbers' are in 1000s of bytes per second processed.

type     bytes       64 bytes    256 bytes   1024 bytes  8192 bytes

md2        131.02k     368.41k     500.57k     549.21k     566.09k
mdc2       535.60k     589.10k     595.88k     595.97k     594.54k
md5       1801.53k    9674.77k   17484.03k   21849.43k   23592.96k
sha       1261.63k    5533.25k    9285.63k   11187.88k   11913.90k
sha1      1103.13k    4782.53k    7933.78k    9472.34k   10070.70k
rc4      10722.53k   14443.93k   15215.79k   15299.24k   15219.59k

des cbc   3286.57k    3827.73k    3913.39k    3931.82k    3926.70k
des ede3  1443.50k    1549.08k    1561.17k    1566.38k    1564.67k
idea cbc  2203.64k    2508.16k    2538.33k    2543.62k    2547.71k
rc2 cbc   1430.94k    1511.59k    1524.82k    1527.13k    1523.33k

blowfish cbc   4716.07k   5965.82k   6190.17k   6243.67k
6234.11k

                   sign       verify
rsa  512 bits      0.0100s    0.0011s
rsa 1024 bits      0.0451s    0.0012s
rsa 2048 bits      0.2605s    0.0086s
rsa 4096 bits      1.6883s    0.0302s
```

Appendix E
Troubleshooting OrbixSSL

This is a checklist to help you make sure that OrbixSSL is installed and configured correctly:

- Ensure that your application works without OrbixSSL, by disabling all OrbixSSL calls in the application. If the application does not work, OrbixSSL is not causing the problem.
- Check whether your application works using the Default Cert mechanism provided by OrbixSSL. Disable all OrbixSSL calls in the application and specify `IT_ENABLE_DEFAULT_CERT TRUE` in the `itssl.cfg` OrbixSSL policy file. If the application now works, any problem is likely to be caused by either OrbixSSL code in the application, or by the certificate or private key that your application is using.

The rest of the suggestions in this appendix assume that your OrbixSSL code is not disabled.

- Insure that `IT_SSL::init()` is called and the return value checked. Also ensure that the return value of all OrbixSSL functions is carefully examined.
- Set `export IT_SSL_TRACE_LEVEL=1`

  This will give some high level handshake information.
- Set `IT_SSL_TRACEFILE` to point to a debug file for a process. The process can now write additional very detailed SSL debug information to this file. Set `IT_SSL_TRACEFILE` to a different file for each process, so that the output of two processes are not confused.

- Check that the certificates, private keys and passwords are correct. For example:

  ```
  ssleay x509 -in MyCert -text
  ```

  This should display the text details of the certificate.

  ```
  ssleay rsa -in MyKey -text
  ```

  This should display the text details of the private key, if the private key is encrypted (which it normally should be). You are asked for a pass-phrase –input the pass-phrase that the OrbixSSL application is attempting to use to decrypt the private key.

- Investigate whether the `ssleay s_client` or `ssleay s_server` utilities provided with OrbixSSL can communicate using the same certificates and keys that they are trying to use with the OrbixSSL applications. If this is not the case then there is a problem with the keys, certificates, or pass-phrases. The customer should recheck them. For example:

  ```
  ssleay s_client -ssl3 -host SomeHost
  -port SomeServerPort -CAfile SomeCAFile
  -cert SomeClientCert -debug

  ssleay s_server -accept MyServerPort -ssl3 -CAfile
  SomeCAFile -cert SomeClientCert -debug -Verify 2
  ```

  The argument `-Verify` enforces client authentication. It is followed by an integer that determines the maximum chain depth allowed. You can also use `-verify` can be instead of `-Verify` which will not reject the connection if a client cert is not available.

  If `ssleay_server` is interrupted the port number it was using can become unavailable for a period of time. Simply use another port when trying again. The `ssleay s_client` port parameter must change to match.

  There is no support for SSL Version 2.0 in OrbixSSL. It supports SSL Version 3.0 only. It does not issue or accept Version 2.0 `hello` messages. This behavior can be simulated in `ssleay s_client` and `ssleay s_server` by the use of the `-ssl3` parameter shown above.

  You can also use `ssleay s_client` and `ssleay s_server` can be used to establish SSL connections with OrbixSSL servers. For example, you can specify the OrbixSSL server port to `ssleay s_client`, and it then attempts to handshake with the OrbixSSL server.

You can also use `s_server` to simulate an OrbixSSL server by running it on the SSL port specified in the IOR that an OrbixSSL client uses. Use `IORDump` see the port.

- If you are an experienced progammer, examine the output of operating system diagnostic tools such as `truss` (Solaris) or `trace` (HP-UX) for the client, server and daemon separately.

Summary of Useful Output to Gather

If you have problems with OrbixSSL and must make a support call, he following can be very helpful:

- Separate files for the Daemon, client and server of the following output having specified `IT_SSL_TRACE_LEVEL=1`:

  The `stdout` and `stderr` (for example, `&` on Unix)

  ```
  daemon.out
  client.out
  server.out
  ```

- Separate `IT_SSL_TRACE_FILE` output for the daemon, client and server:

  ```
  daemon.log
  client.log
  server.log
  ```

- Separate `truss` (or `trace`) output for the daemon, client and server. For Multi-threaded applications use `trace -l` on Solaris to show the system calls per thread.

  ```
  daemon.trc
  client.trc
  server.trc
  ```

- The OrbixSSL Security config file `itssl.cfg`
- The root CA file that is referenced by `itssl.cfg`
- If appropriate the certificates and private key files with passwords can be useful, in order to attempt to reproduce the problem exactly.

Note: Do not send us the password and private keys for a Live system!

- If possible the complete source for a minimal test case.
- If this is not possible then include the excerpts of the client and server programs which make OrbixSSL calls.
- A core dump, and a text stack trace, if the problem causes the program to dump core.

Index

K

L

M

N

O

P

R

S