CHAPTER 1

Overview of the CORBA Security Features

This topic includes the following sections:

- The CORBA Security Features
- The CORBA Security Environment
- BEA Tuxedo Security SPIs

Notes: The BEA Tuxedo product includes environments that allow you to build both Application-to-Transaction Monitor Interfaces (ATMI) and CORBA applications. This topic explains how to implement security in a CORBA application. For information about implementing security in an ATMI application, see *Using Security in ATMI Applications*.

The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

The CORBA Security Features

Security refers to techniques for ensuring that data stored in a computer or passed between computers is not compromised. Most security measures involve proof material and data

encryption, where the proof material is a secret word or phrase that gives a user access to a particular program or system, and data encryption is the translation of data into a form that cannot be interpreted.

Distributed applications such as those used for electronic commerce (e-commerce) offer many access points for malicious people to intercept data, disrupt operations, or generate fraudulent input; the more distributed a business becomes, the more vulnerable it is to attack. Thus, the distributed computing software, or middleware, upon which such applications are built must provide security.

The CORBA security features of the BEA Tuxedo product lets you establish secure connections between client and server applications. It has the following features:

- Authentication of CORBA C++ applications to the BEA Tuxedo domain. Authentication can be accomplished using a standard username/password combination or the identity inside of the X.509 digital certificate provided to the server applications.
- Data integrity and confidentiality through Link-Level Encryption (LLE) or the Secure Sockets Layer (SSL) protocol. CORBA C++ applications can establish SSL sessions with a BEA Tuxedo domain. BEA Tuxedo client applications can use LLE or SSL to protect network traffic between bridges and domains.
- Security Service Provider Interfaces (SPIs) that can be used to integrate security mechanisms that provide authentication, authorization, auditing, and public key security features. Security vendors can use the SPIs to integrate third-party security offerings into the CORBA environment.
- A Public Key Infrastructure (PKI) that uses the SSL protocol and X.509 digital certificates to provide data privacy for messages sent over network links. In addition, a set of PKI SPIs are provided.

To access the full security features of the CORBA environment, you need to install a license that enable the use of the SSL protocol, LLE, and PKI. For information about installing the license for the security features, see the *Installing the BEA Tuxedo System*.

Note: *Using Security in CORBA Applications* describes the security features of the CORBA environment in the BEA Tuxedo product. For a complete description of using the security features in the ATMI environment in the BEA Tuxedo product, see *Using Security in ATMI Applications*.

Table 1-1 summarizes the features in the CORBA security features in the BEA Tuxedo product.

Table 1-1 CORBA Security Features

Security Features	Description	Service Provider Interface (SPI)	Default Implementation
Authentication	Proves the stated identity of users or system processes; safely remembers and transports identity information; and makes identity information available when needed.	Implemented as a single interface	Provides security at three levels: no authentication, application password, and certificate authentication.
Authorization	Controls access to resources based on identity or other information.	Implemented as a single interface	N/A
Auditing	Safely collects, stores, and distributes information about operating requests and their outcomes.	Implemented as a single interface	Default auditing security is implemented via the features of the user log (ULOG).
Link-Level Encryption	Uses symmetric key encryption to establish data privacy for messages moving over the network links that connect the machines in a CORBA application.	N/A	RC4 symmetric key encryption.

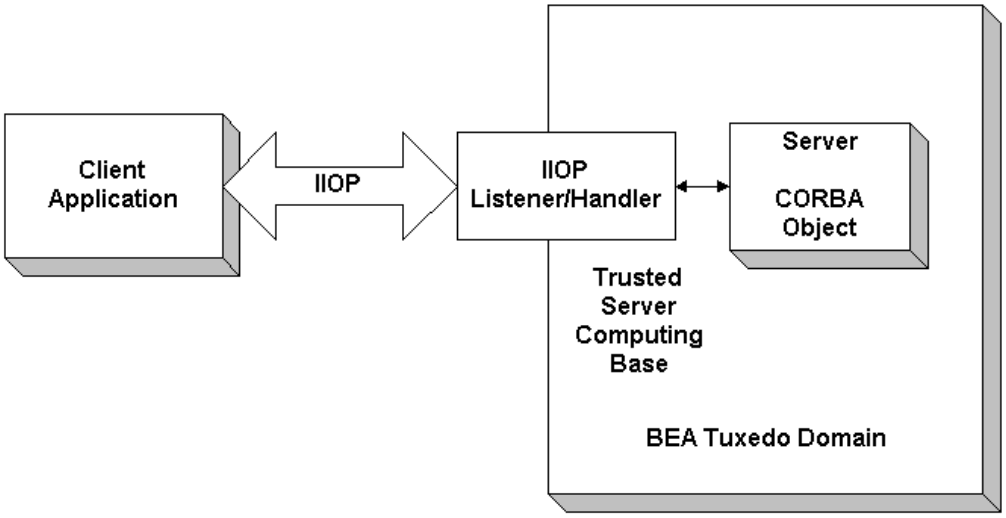
Table 1-1 CORBA Security Features (Continued)

Security Features	Description	Service Provider Interface (SPI)	Default Implementation
The Secure Sockets Layer (SSL) protocol	Uses asymmetric encryption to establish data privacy for messages moving over network links between BEA Tuxedo domains.	N/A	The SSL version 3.0 protocol.
Public key security	Uses public key (or asymmetric key) encryption to establish data privacy for messages moving over the network links between remote client applications and the IIOP Listener/Handler. Complies with SSL version 3.0 allowing mutual authentication based on X.509 digital certificates.	Implemented as the following interfaces: • Public key initialization • Key management • Certificate lookup • Certificate parsing • Certificate validation • Proof material mapping	Default public key security supports the following algorithms: • RSA for key exchange. • AES or DES and its variants RC2 and RC4 for bulk encryption. • MD5 and SHA for message digests.

The CORBA Security Environment

Direct end-to-end mutual authentication in a distributed enterprise middleware environment such as the BEA Tuxedo CORBA environment can be prohibitively expensive, especially when accomplished through security mechanisms optimized for long duration connections. It is not efficient for principals to establish direct network connections with each server application, nor is it practical to exchange and verify multiple authentication messages as part of processing each service request. Instead, CORBA applications in a BEA Tuxedo product implements a delegated trust authentication model as shown in Figure 1-1.

Figure 1-1 Delegated Trust Model



In a delegated trust model, principals (generally users of client applications) authenticate to a trusted system gateway process. In the case of the CORBA applications, the trusted system gateway process is the IIOP Listener/Handler. As part of successful authentication, security tokens are assigned to the initiating principal. A security token is an opaque data structure suitable for transfer between processes.

When a request from an authenticated principal reaches the IIOP Listener/Handler, the IIOP Listener/Handler attaches the principal's security tokens to the request and delivers the request to the target server application for authorization and auditing purposes.

In a delegated trust authentication model, the IIOP Listener/Handler trusts that the authentication software in the BEA Tuxedo domain will verify the identity of the principal and generates the appropriate security tokens. Server applications, in turn, trust that the IIOP Listener/Handler will attach the correct security tokens. Server applications also trust that any other server applications involved in the process of a request from a principal will safely deliver the security tokens.

A session is established between the initiating client application and the IIOP Listener/Handler in the following way:

1. When a client application wants to access an object within a BEA Tuxedo domain, the client application uses either a username and password or a X.509 digital certificate to authenticate over the connection with the IIOP Listener/Handler.

2. A security association called a security context is established between a principal and the IIOP Listener/Handler. This security context is used to control access to objects in the BEA Tuxedo domain.

   The IIOP Listener/Handler retrieves the authorization and auditing tokens from the security context. Together, the authorization and auditing tokens represent the principal's identity associated with the security context.

3. Once the authentication process is complete, the principal invokes an object in the BEA Tuxedo domain. The request is packaged into an IIOP request and forwarded to the IIOP Listener/Handler. The IIOP Listener/Handler associates the request with the previously established security context.

4. The IIOP Listener/Handler receives the request from the initiating principal.

   The protection of messages between the client application and the IIOP Listener/Handler is dependent on the security technology used in the CORBA application. The default behavior of the BEA Tuxedo product is to encrypt the authentication information but not to protect the message sent between the client application and the BEA Tuxedo domain. The message is sent in clear text. The SSL protocol can be used to protect the message. If the SSL protocol is configured to protect messages for integrity and confidentiality, the request is digitally signed and sealed (encrypted) before it is sent to the IIOP Listener/Handler.

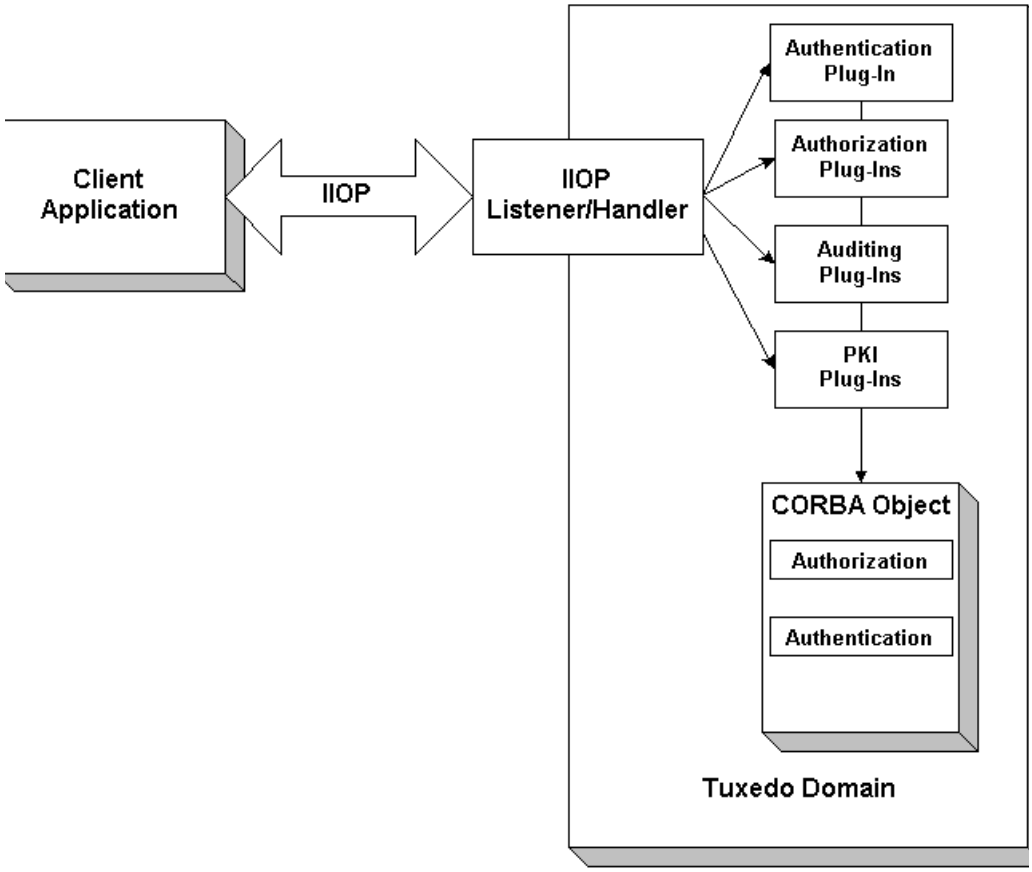
5. The IIOP Listener/Handler forwards the request along with the authorization and auditing tokens of the initiating principal to the appropriate server application.

6. When the request is received by the server application, the BEA Tuxedo system interrogates the forwarded tokens of the requesting principal to determine if the request should be processed or denied. The CORBA security features will, based on the decision of the authorization implementation, deny the processing of any request on an object for which the requesting principal has no permission to access.

BEA Tuxedo Security SPIs

As shown in Figure 1-2, the authentication, authorization, auditing, and public key security features available with the BEA Tuxedo product are implemented through a plug-in interface, which allows security plug-ins to be integrated into the CORBA environment. A security plug-in is a code module that implements a particular security feature.

Figure 1-2 Architecture for the BEA Tuxedo Security Service Provider Interfaces



The BEA Tuxedo product provides interfaces for the types of security plug-ins listed in Table 1-2.

Table 1-2 The BEA Tuxedo Security Plug-Ins

Plug-In	Description
Authentication	Allows communicating processes to mutually prove identification.
Authorization	Allows system administrators to control access to CORBA applications. Specifically, an administrator can use authorization to allow or disallow principals to use resources or services provided by a CORBA application.
Auditing	Provides a means to collect, store, and distribute information about operating requests and their outcomes. Audit-trail records may be used to determine which principals performed, or attempted to perform, actions that violated the configured security policies of a CORBA application. They may also be used to determine which operations were attempted, which ones failed, and which ones successfully completed.
Public key initialization	Allows public key software to open public and private keys. For example, gateway processes may need to have access to a specific private key in order to decrypt messages before routing them.
Key management	Allows public key software to manage and use public and private keys. Note that message digests and session keys are encrypted and decrypted using this interface, but no bulk data encryption is performed using public key cryptography. Bulk data encryption is performed using symmetric key cryptography.
Certificate lookup	Allows public key software to retrieve X.509v3 digital certificates for a given principal. Digital certificates may be stored using any appropriate certificate repository, such as Lightweight Directory Access Protocol (LDAP).

Table 1-2 The BEA Tuxedo Security Plug-Ins (Continued)

Plug-In	Description
Certificate parsing	Allows public key software to associate a simple principal name with an X.509v3 digital certificate. The parser analyzes a digital certificate to generate a principal name to be associated with the digital certificate.
Certificate validation	Allows public key software to validate an X.509v3 digital certificate in accordance with specific business logic.
Proof material mapping	Allows public key software to access the proof materials needed to open keys, provide authorization tokens, and provide auditing tokens.

The specifications for the SPIs are currently only available to third-party security vendors who have entered into a special agreement with BEA Systems, Inc. Customers who want to customize a security feature must contact one of these vendors or BEA Professional Services. For example, a BEA customer who wants a custom implementation of public key security must contact a third-party vendor who can provide the appropriate security plug-in or BEA Professional Services.

For more information about security plug-ins, including installation and configuration procedures, see your BEA account executive.

CHAPTER 2

Introduction to the SSL Technology

This topic includes the following sections:

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code

samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

The SSL Protocol

The Secure Sockets Layer (SSL) protocol allows you to integrate these essential features into your CORBA application:

- Confidentiality

  Confidentiality is the ability to keep communications secret from parties other than the intended recipient. It is achieved by encrypting data with strong algorithms. The SSL protocol provides a secure mechanism that enables two communicating parties to negotiate the strongest algorithm they both support and to agree on the keys with which to encrypt the data.

- Integrity

  Integrity is a guarantee that the data being transferred has not been modified in transit. The same handshake mechanism which allows the two parties to agree on algorithms and keys also allows the two ends of an SSL connection to establish shared data integrity secrets which are used to ensure that when data is received any modifications will be detected.

- Authentication

  Authentication is the ability to ascertain with whom you are speaking. By using digital certificates and public key security, CORBA client and server applications can each be authenticated to the other. This allows the two parties to be certain they are communicating with someone they trust. The SSL protocol provides a mechanism that can be used to authenticate principals to a BEA Tuxedo domain using X.509 digital certificates. The use of certificate authentication can be used as an alternative to password authentication.

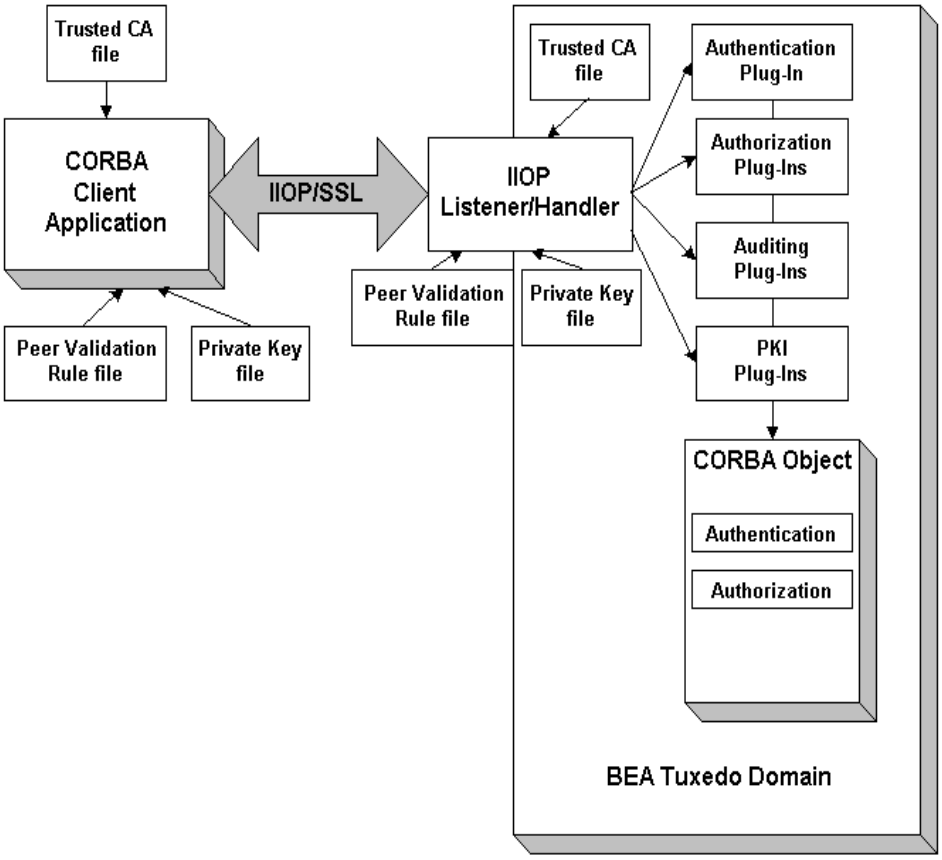
The SSL protocol provides secure connections by allowing two applications connecting over a network connection to authenticate the other's identity and by encrypting the data exchanged between the applications. When using the SSL protocol, the target always authenticates itself to the initiator. Optionally, if the target requests it, the initiator can authenticate itself to the target. Encryption makes data transmitted over the network intelligible only to the intended recipient. An SSL connection begins with a handshake during which the applications exchange digital

certificates, agree on the encryption algorithms to use, and generate encryption keys used for the remainder of the session.

The SSL protocol uses public key encryption for authentication. With public key encryption, a pair of asymmetric keys are generated for a principal or other entity such as the IIOP Listener/Handler or an application server. The keys are related such that the data encrypted with the public key can only be decrypted using the corresponding private key. Conversely, data encrypted with the private key can be decrypted only with the public key. The private key is carefully protected so that only the owner can decrypt messages. The public key, however, is distributed freely so that anyone can encrypt messages intended for the owner.

Figure 2-1 illustrates how the SSL protocol works in the CORBA security environment.

Figure 2-1 The SSL Protocol in the CORBA Security Environment



When using the SSL protocol in the CORBA security environment, the IIOP Listener/Handler authenticates itself to initiating principals. The IIOP Listener/Handler presents its digital certificate to the initiating principal. To successfully negotiate a SSL connection, the client application must then authenticate the IIOP Listener/Handler but the IIOP Listener/Handler will accept any client application into the SSL connection. This type of authentication is referred to as *server authentication*.

When using server authentication, the initiating client application is required to have digital certificates for certificate authorities that are to be trusted. The IIOP Listener/Handler must have

a private key and digital certificates that represents its identity. Server authentication is common on the Internet where customers want to create secure connections before they share personal data. In this case, the client application has a similar role to that of a Web browser.

With SSL version 3.0, principals can also authenticate to the IIOP Listener/Handler. This type of authentication is referred to as *mutual authentication*. In mutual authentication, principals present their digital certificates to the IIOP Listener/Handler. When using mutual authentication, both the IIOP Listener/Handler and the principal need private keys and digital certificates that represent their identity. This type of authentication is useful when you must restrict access to trusted principals only.

The SSL protocol and the infrastructure needed to use digital certificates is available in the BEA Tuxedo product by installing a license available in the product installation. For more information, see *Installing the BEA Tuxedo System*.

Digital Certificates

Digital certificates are electronic documents used to uniquely identify principals and entities over networks such as the Internet. A digital certificate securely binds the identity of a principal or entity, as verified by a trusted third party known as a certificate authority (CA), to a particular public key. The combination of the public key and the private key provides a unique identity to the owner of the digital certificate.

Digital certificates allow verification of the claim that a specific public key does in fact belong to a specific principal or entity. A recipient of a digital certificate can use the public key contained in the digital certificate to verify that a digital signature was created with the corresponding private key. If such verification is successful, this chain of reasoning provides assurance that the corresponding private key is held by the subject named in the digital certificate, and that the digital signature was created by that particular subject.

A digital certificate typically includes a variety of information, such as:

- The name of the subject (holder, owner) and other identification information required to uniquely identify the subject, such as the URL of the Web server using the digital certificate, or an individual's e-mail address.
- The subject's public key.
- The name of the certificate authority that issued the digital certificate.
- A serial number.

- The validity period (or lifetime) of the digital certificate (defined by a start date and an end date).

The most widely accepted format for digital certificates is defined by the ITU-T X.509 international standard. Thus, digital certificates can be read or written by any application complying with X.509. The PKI in the CORBA security environment recognizes digital certificates that comply with X.509 version 3, or X.509v3.

Certificate Authority

Digital certificates are issued by a certificate authority. Any trusted third-party organization or company that is willing to vouch for the identities of those to whom it issues digital certificates and public keys can be a certificate authority. When a certificate authority creates a digital certificate, the certificate authority signs it with its private key, to ensure the detection of tampering. The certificate authority then returns the signed digital certificate to the requesting subject.

The subject can verify the digital signature of the issuing certificate authority by using the public key of the certificate authority. The certificate authority makes its public key available by providing a digital certificate issued from a higher-level certificate authority attesting to the validity of the public key of the lower-level certificate authority. The second solution gives rise to hierarchies of certificate authorities. This hierarchy is terminated by a self-signed digital certificate known as the root key.

The recipient of an encrypted message can develop trust in the private key of a certificate authority recursively, if the recipient has a digital certificate containing the public key of the certificate authority signed by a superior certificate authority whom the recipient already trusts. In this sense, a digital certificate is a stepping stone in digital trust. Ultimately, it is necessary to trust only the public keys of a small number of top-level certificate authorities. Through a chain of digital certificates, trust in a large number of users' digital signatures can be established.

Thus, digital signatures establish the identities of communicating entities, but a digital signature can be trusted only to the extent that the public key for verifying the digital signature can be trusted.

Certificate Repositories

To make a public key and its identification with a specific subject readily available for use in verification, the digital certificate may be published in a repository or made available by other means. Certificate repositories are databases of digital certificates and other information

available for retrieval and use in verifying digital signatures. Retrieval can be accomplished automatically by directly requesting digital certificates from the repository as needed.

In the CORBA security environment, Lightweight Directory Access Protocol (LDAP) is used as a certificate repository. BEA Systems, Inc. does not provide or recommend any specific LDAP server. The LDAP server you choose should support the X.500 scheme definition and the LDAP version 2 or 3 protocol.

A Public Key Infrastructure

A Public Key Infrastructure (PKI) consists of protocols, services, and standards supporting applications of public key cryptography. Because the technology is still relatively new, the term PKI is somewhat loosely defined: sometimes PKI simply refers to a trust hierarchy based on public key digital certificates; in other contexts, it embraces digital signature and encryption services provided to end-user applications as well.

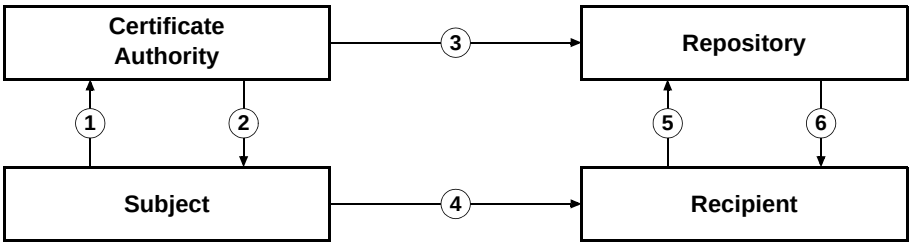
There is no single standard public key infrastructure today, though efforts are underway to define one. It is not yet clear whether a standard will be established or multiple independent PKIs will evolve with varying degrees of interoperability. In this sense, the state of PKI technology today can be viewed as similar to local and wide area (WAN) network technology in the 1980s, before there was widespread connectivity via the Internet.

The following services are likely to be found in a PKI:

- Key registration for issuing a new digital certificate for a public key.
- Certificate revocation for canceling a previously-issued digital certificate and private key.
- Key selection for obtaining a party's public key.
- Trust evaluation for determining whether a digital certificate is valid and which operations it authorizes.

Figure 2-2 shows the PKI process flow.

Figure 2-2 PKI Process Flow



1. The subject applies to a certificate authority for digital certificate.
2. The certificate authority verifies the identity of subject and issues a digital certificate.
3. The certificate authority or the subject publishes the digital certificate in a certificate repository such as LDAP.
4. The subject digitally signs an electronic message with the associated private key to ensure sender authenticity, message integrity, and nonrepudiation, and then sends message to recipient.
5. The recipient retrieves the sender's certificate from the certificate repository and then retrieves the public key from the certificate.

The BEA Tuxedo product does not provide the tools necessary to be a certificate authority. BEA Systems, Inc. recommends using a third-party certificate authority such as VeriSign or Entrust. By offering a Public Key SPI, BEA Systems, Inc. extends the opportunity to all BEA Tuxedo customers to use a PKI security solution with the PKI software from their vendor of choice. See "PKI Plug-ins" on page 3-22 for more information.

PKCS-5 and PKCS-8 Compliance

Informal but recognized industry standards for public key software have been issued by a group of leading communications companies, led by RSA Laboratories. These standards are called "Public-Key Cryptography Standards," or PKCS. The BEA Tuxedo product uses PKCS-5 and PKCS-8 to protect the private keys used with the SSL protocol.

- PKCS-5 is a specification of a format for using password-based encryption that uses DES to protect data.
- PKCS-8 is a specification of a format for storing private keys, including the ability to encrypt them with PKCS-5.

Supported Public Key Algorithms

Public key (or *asymmetric key*) algorithms are implemented through a pair of different but mathematically related keys:

- A public key (which is distributed widely) for verifying a digital signature or transforming data into a seemingly unintelligible form.
- A private key (which is always kept secret) for creating a digital signature or returning the data to its original form.

The public key security in the CORBA security environment also supports digital signature algorithms. Digital signature algorithms are simply public key algorithms used to provide digital signatures.

The BEA Tuxedo product supports the Rivest, Shamir, and Adelman (RSA) algorithm, the Diffie-Hellman algorithm, and Digital Signature Algorithm (DSA). With the exception of DSA, digital signature algorithms can be used for digital signatures and encryption. DSA can be used for digital signatures but not for encryption.

Supported Symmetric Key Algorithms

In symmetric key algorithms, the same key is used to encrypt and decrypt a message. The public key encryption system uses symmetric key encryption to encrypt a message sent between two communicating entities. Symmetric key encryption operates at least 1000 times faster than public key cryptography.

A block cipher is a type of symmetric key algorithm that transforms a fixed-length block of *plaintext* (unencrypted text) data into a block of *ciphertext* (encrypted text) data of the same length. This transformation takes place in accordance with the value of a randomly generated session key. The fixed length is called the block size.

The Public key security feature in the CORBA security environment supports the following symmetric key algorithms:

- DES-CBC (Data Encryption Standard for Cipher Block Chaining)

  DES-CBC is a 64-bit block cipher run in Cipher Block Chaining (CBC) mode. It provides 56-bit keys (8 parity bits are stripped from the full 64-bit key).

- Two-key triple-DES (Data Encryption Standard)

  Two-key triple-DES is a 128-bit block cipher run in Encrypt-Decrypt-Encrypt (EDE) mode. Two-key triple-DES provides two 56-bit keys (in effect, a 112-bit key).

For some time it has been common practice to protect and transport a key for DES encryption with triple-DES, which means that the input data (in this case the single-DES key) is encrypted, decrypted, and then encrypted again (an encrypt-decrypt-encrypt process). The same key is used for the two encryption operations.

- RC2 (Rivest's Cipher 2)

  RC2 is a variable key-size block cipher.

  – RC4 (Rivest's Cipher 4)

  RC4 is a variable key-size block cipher with a key size range of 40 to 128 bits. It is faster than DES and is exportable with a key size of 40 bits. A 56-bit key size is allowed for foreign subsidiaries and overseas offices of United States companies. In the United States, RC4 can be used with keys of virtually unlimited length, although the public key security in the CORBA security environment restricts the key length to 128 bits.

- AES-256-CBC (Advanced Encryption Standard for Cipher Block Chaining)

  AES-256-CBC is a 128-bit block cipher run in Cipher Block Chaining (CBC) mode. It provides 256-bits keys

Customers of the BEA Tuxedo product cannot expand or modify this list of algorithms.

Supported Message Digest Algorithms

The CORBA security environment supports the MD5 and SHA-1 (Secure Hash Algorithm 1) message digest algorithms. Both MD5 and SHA-1 are well known, one-way hash algorithms. A one-way hash algorithm takes a message and converts it into a fixed string of digits, which is referred to as a *message digest* or *hash value*.

MD5 is a high-speed, 128-bit hash; it is intended for use with 32-bit machines. SHA-1 offers more security by using a 160-bit hash, but is slower than MD5.

Supported Cipher Suites

A cipher suite is a SSL encryption method that includes the key exchange algorithm, the symmetric encryption algorithm, and the secure hash algorithm used to protect the integrity of the communication. For example, the cipher suite `RSA_WITH_RC4_128_MD5` uses RSA for key exchange, RC4 with a 128-bit key for bulk encryption, and MD5 for message digest.

The CORBA security environment supports the cipher suites described in Table 2-1.

Table 2-1 SSL Cipher Suites Supported by the CORBA Security Environment

Cipher Suite	Key Exchange Type	Symmetric Key Strength
SSL_RSA_WITH_RC4_128_SHA	RSA	128
SSL_RSA_WITH_RC4_128_MD5	RSA	128
SSL_RSA_WITH_DES_CDC_SHA	RSA	56
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA	40
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	40
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA	40
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	Diffie-Hellman	40
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	Diffie-Hellman	40
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	112
SSL_RSA_WITH_NULL_SHA	RSA	0
SSL_RSA_WITH_NULL_MD5	RSA	0

Standards for Digital Certificates

The CORBA security environment supports the digital certificates that conform to the X.509v3 standard. The X.509v3 standard specifies the format of digital certificates. BEA recommends obtaining certificates from a certificate authority such as Verisign or Entrust.

CHAPTER 3

Fundamentals of CORBA Security

This topic includes the following sections:

- Link-Level Encryption
- Password Authentication
- The SSL Protocol
- Certificate Authentication
- Using an Authentication Plug-in
- Authorization
- Auditing
- PKI Plug-ins
- Commonly Asked Questions About the CORBA Security Features

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

Link-Level Encryption

Link-Level Encryption (LLE) establishes data privacy for messages moving over the network links. The objective of LLE is to ensure confidentiality so that a network-based eavesdropper cannot learn the content of BEA Tuxedo system messages or CORBA application-generated messages. It employs the symmetric key encryption technique (specifically, RC4), which uses the same key for encryption and decryption.

When LLE is being used, the BEA Tuxedo system encrypts data before sending it over a network link and decrypts it as it comes off the link. The system repeats this encryption/decryption process at every link through which the data passes. For this reason, LLE is referred to as a point-to-point facility.

LLE can be used to encrypt communication between machines and/or domains in a CORBA application..

Note: LLE cannot be used to protect connections between remote CORBA client applications and the IIOP Listener/Handler.

There are three levels of LLE security: 0-bit (no encryption), 56-bit (Export), and 128-bit (Domestic). The Export LLE version allows 0-bit and 56-bit encryption. The Domestic LLE version allows 0, 56, and 128-bit encryption.

How LLE Works

LLE works in the following way:

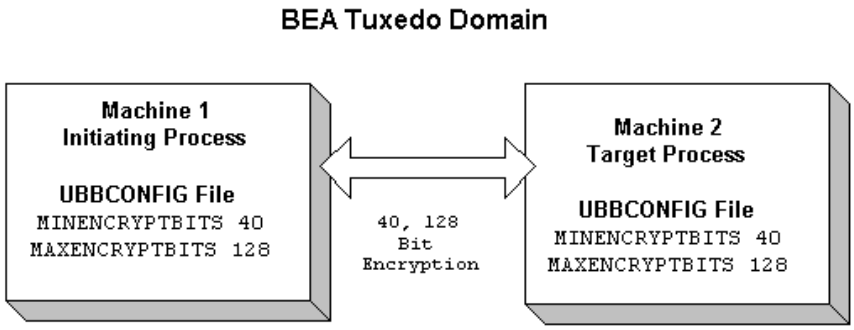
1. The system administrator sets parameters for any processes that want to use LLE to control the encryption strength.
   - The first configuration parameter is the minimum encryption level that a process will accept. It is expressed as a key length: 0, 56, or 128 bits.
   - The second configuration parameter is the maximum encryption level a process can support. It also is expressed as a key length: 0, 56, or 128 bits.

   For convenience, the two parameters are denoted as (`min`, `max`). For example, the values (56, 128) for a process mean that the process accepts at least 56-bit encryption but can support up to 128-bit encryption.
2. An initiator process begins the communication session.
3. A target process receives the initial connection and starts to negotiate the encryption level to be used by the two processes to communicate.

4. The two processes agree on the largest common key size supported by both.
5. The configured maximum key size parameter is reduced to agree with the installed software's capabilities. This step must be done at link negotiation time, because at configuration time it may not be possible to verify a particular machine's installed encryption package.
6. The processes exchange messages using the negotiated encryption level.

Figure 3-1 illustrates these steps.

Figure 3-1 How LLE Works



Encryption Key Size Negotiation

When two processes at the opposite ends of a network link need to communicate, they must first agree on the size of the key to be used for encryption. This agreement is resolved through a two-step process of negotiation.

1. Each process identifies its own *min-max* values.
2. Together, the two processes find the largest key size supported by both.

Determining min-max Values

When either of the two processes starts up, the BEA Tuxedo system (1) checks the bit-encryption capability of the installed LLE version by checking the LLE licensing information in the

`lic.txt` file and (2) checks the LLE *`min-max`* values for the particular link type as specified in the two configuration files. The BEA Tuxedo system then proceeds as follows:

- If the configured *`min-max`* values accommodate the installed LLE version, then the local software assigns those values as the *`min-max`* values for the process.
- If the configured *`min-max`* values do not accommodate the installed LLE version, for example, if the Export LLE version is installed but the configured *`min-max`* values are (0, 128), then the local software issues a run-time error; link-level encryption is not possible at this point.
- If there are no *`min-max`* values specified in the configurations for a particular link type, then the local software assigns 0 as the minimum value and assigns the highest bit-encryption rate possible for the installed LLE versions as the maximum value, that is, (0, 128) for the Domestic LLE version.

Finding a Common Key Size

After the *`min-max`* values are determined for the two processes, the negotiation of key size begins. The negotiation process need not be encrypted or hidden. Once a key size is agreed upon, it remains in effect for the lifetime of the network connection.

Table 3-1 shows which key size, if any, is agreed upon by two processes when all possible combinations of *`min-max`* values are negotiated. The header row holds the *`min-max`* values for one process; the far left column holds the *`min-max`* values for the other.

Table 3-1 Interprocess Negotiation Results

	(0, 0)	**(0, 56)**	**(0, 128)**	**(56, 56)**	**(56, 128)**	**(128, 128)**
(0, 0)	0	0	0	ERROR	ERROR	ERROR
(0, 56)	0	56	56	56	56	ERROR
(0, 128)	0	56	128	56	128	128
(56, 56)	ERROR	56	56	56	56	ERROR
(56, 128)	ERROR	56	128	56	128	128
(128, 128)	ERROR	ERROR	128	ERROR	128	128

WSL/WSH Connection Timeout During Initialization

The length of time a Workstation client can take for initialization is limited. By default, this interval is 30 seconds in an application not using LLE, and 60 seconds in an application using LLE. The 60-second interval includes the time needed to negotiate an encrypted link. This time limit can be changed when LLE is configured by changing the value of the `MAXINITTIME` parameter for the Workstation Listener (WSL) server in the `UBBCONFIG` file, or the value of the `TA_MAXINITTIME` attribute in the `T_WSL` class of the `WS_MIB`(5).

Development Process

To use LLE in a CORBA application, you need to install a license that enables the use of LLE. For information about installing the license, see *Installing the BEA Tuxedo System*.

The implementation of LLE is an administrative task. The system administrators for each CORBA application set *`min-max`* values in the `UBBCONFIG` file that control encryption strength. When the two CORBA applications establish communication, they negotiate what level of encryption to use to exchange messages. Once an encryption level is negotiated, it remains in effect for the lifetime of the network connection.

Password Authentication

The CORBA security environment supports a password mechanism to provide authentication to existing CORBA applications and to new CORBA applications that are not prepared to deploy a full Public Key Infrastructure (PKI). When using password authentication, the applications that initiate invocations on CORBA objects authenticate themselves to the BEA Tuxedo domain using a defined username and password.

The following levels of password authentication are provided:

- None—indicates that no password or access checking is performed in the CORBA application.
- Application Password—indicates that users are required to supply a domain password in order to access the CORBA application.
- User Authentication—indicates that users are required to supply an application password as well as the domain password in order to access the CORBA application.
- ACL—indicates that authorization is used in the CORBA application and access control checks are performed on interfaces, queue names, and event names. If an associated ALC is not found for a user, it is assumed that access is granted.

- Mandatory ACL—indicates that authorization is used in the CORBA application and access control checks are performed on interfaces, queue names, and event names. The value of Mandatory ACL is similar to ACL, but permission is denied if an associated ACL is not found for the user.

When using Password authentication, you have the option of using the `Tobj::PrincipalAuthenticator::logon()` or the `SecurityLevel2::PrincipalAuthenticator::authenticate()` methods in your client application.

If you use password authentication, the SSL protocol can be used to provide confidentiality and integrity to communication between applications. For more information, see "The SSL Protocol" on page 3-9.

How Password Authentication Works

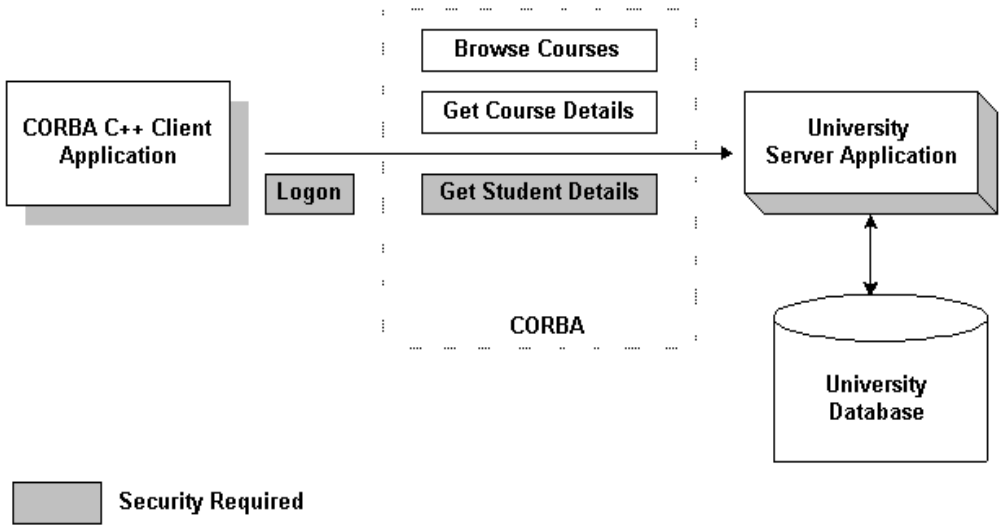
Password authentication works in the following way:

1. The initiating application accesses the BEA Tuxedo domain in one of the following ways:
   - Through the CORBA Interoperable Naming Service (INS) Bootstrapping mechanism. Use this mechanism if you are using a client ORB from another vendor. For more information about using CORBA INS, see the *CORBA Programming Reference* in the BEA Tuxedo online documentation
   - The BEA Bootstrapping mechanism. Use this mechanism if you are using BEA CORBA client applications.
2. The initiating application obtains credentials for the user. The initiating application must provide proof material to be used by the BEA Tuxedo domain to authenticate the user. This proof material consists of the name of the user and a password.
   - The initiating application creates the security context using a `PrincipalAuthenticator` object. The request for authentication is sent to the IIOP Listener/Handler. The proof material in the authentication request is securely relayed to the authentication server, which verifies the supplied information.
   - If the verification succeeds, the BEA Tuxedo system constructs a `Credentials` object that is used by all future invocations. The `Credentials` object for the user is associated with the `Current` object that represents the security context.
3. The initiating application invokes a CORBA object in the BEA Tuxedo domain using an object reference. The request is packaged into an IIOP request and is forwarded to the IIOP Listener/Handler that associates the request with the previously established security context.

4. The IIOP Listener/Handler receives the request from the initiating application.
5. The IIOP Listener/Handler forwards the request, along with the credentials of the initiating application, to the appropriate CORBA object.

Figure 3-2 illustrates these steps.

Figure 3-2 How Password Authentication Works



Development Process for Password Authentication

Defining password authentication for a CORBA application includes administration and programming steps. Table 3-2 and Table 3-3 list the administration and programming steps for password authentication. For a detailed description of the administration steps for password authentication, see "Configuring Authentication" on page 7-1. For a complete description of the programming steps, see "Writing a CORBA Application That Implements Security" on page 9-1.

Table 3-2 Administration Steps for Password Authentication

Step	Description
1	Set the SECURITY parameter in the UBBCONFIG file to APP_PW, USER_AUTH, ACL, or MANDATORY_ACL.
2	If you defined the SECURITY parameter as USER_AUTH, ACL, or MANDATORY_ACL, configure the authentication server (AUTHSRV) in the UBBCONFIG file.
3	Use the tpusradd and tpgrpadd commands to define lists of authorized users and groups including the IIOP Listener/Handler.
4	Use the tmloadcf command to load the UBBCONFIG file. When the UBBCONFIG file is loaded, the system administrator is prompted for a password. The password entered at this time becomes the password for the CORBA application.

Table 3-3 Programming Steps for Password Authentication

Step	Description
1	Write application code that uses the Bootstrap object to obtain a reference to the SecurityCurrent object or CORBA INS to obtain a reference to a PrincipalAuthenticator object in the BEA Tuxedo domain.
2	Write application code that obtains the PrincipalAuthenticator object from the SecurityCurrent object.
3	Write application code that uses the Tobj::PrincipalAuthenticator::logon() or SecurityLevel2::PrincipalAuthenticator::authenticate() operation to establish a security context with the BEA Tuxedo domain.
4	Write application code that prompts the user for the password defined when the UBBCONFIG file is loaded.

The SSL Protocol

The BEA Tuxedo product provides the industry-standard SSL protocol to establish secure communications between client and server applications. When using the SSL protocol, principals use digital certificates to prove their identity to a peer.

The default behavior of the SSL protocol in the CORBA security environment is to have the IIOP Listener/Handler prove its identity to the principal who initiated the SSL connection using digital certificates. The digital certificates are verified to ensure that each of the digital certificates has not been tampered with or expired. If there is a problem with any of the digital certificates in the chain, the SSL connection is terminated. In addition, the issuer of a digital certificate is compared against a list of trusted certificate authorities to verify the digital certificate received from the IIOP Listener/Handler has been signed by a certificate authority that is trusted by the BEA Tuxedo domain.

Like LLE, the SSL protocol can be used with password authentication to provide confidentiality and integrity to communication between the client application and the BEA Tuxedo domain. When using the SSL protocol with password authentication, you are prompted for the password of the IIOP Listener/Handler defined by the `SEC_PRINCIPAL_NAME` parameter when you enter the `tmloadcf` command.

How the SSL Protocol Works

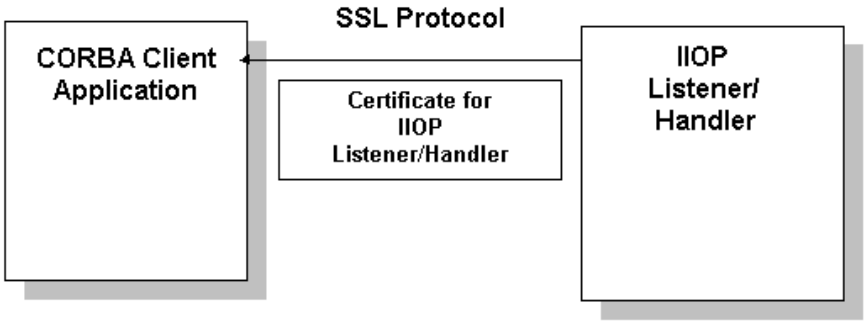
The SSL protocol works in the following manner:

1. The IIOP Listener/Handler presents its digital certificate to the initiating application.
2. The initiating application compares the digital certificate of the IIOP Listener/Handler against its list of trusted certificate authorities.
3. If the initiating application validates the digital certificate of the IIOP Listener/Handler, the application and the IIOP Listener/Handler establish an SSL connection.

   The initiating application can then use either password or certificate authentication to authenticate itself to the BEA Tuxedo domain.

Figure 3-3 illustrates how the SSL protocol works.

Figure 3-3 How the SSL Protocol Works in a CORBA Application



Requirements for Using the SSL Protocol

To use the SSL protocol in a CORBA application, you need to install a license that enables the use of the SSL protocol and PKI. For information about installing the license for the security features, see *Installing the BEA Tuxedo System*.

The implementation of the SSL protocol is flexible enough to fit into most public key infrastructures. The BEA Tuxedo product requires that digital certificates are stored in an LDAP-enabled directory. You can choose any LDAP-enabled directory service. You also need to choose the certificate authority from which to obtain digital certificates and private keys used in a CORBA application. You must have an LDAP-enabled directory service and a certificate authority in place before using the SSL protocol in a CORBA application.

Development Process for the SSL Protocol

Using the SSL protocol in a CORBA application is primarily an administration process. Table 3-5 lists the administration steps required to set up the infrastructure required to use the SSL protocol and configure the IIOP Listener/Handler for the SSL protocol. For a detailed description of the administration steps, see "Managing Public Key Security" on page 4-1 and "Configuring the SSL Protocol" on page 6-1.

Once the administration steps are complete, you can use either password authentication or certificate authentication in your CORBA application. For more information, see "Writing a CORBA Application That Implements Security" on page 9-1.

Note: If you are using the BEA CORBA C++ ORB as a server application, the ORB can also be configured to use the SSL protocol. For more information, see "Configuring the SSL Protocol" on page 6-1.

Table 3-4 Administration Steps for the SSL Protocol

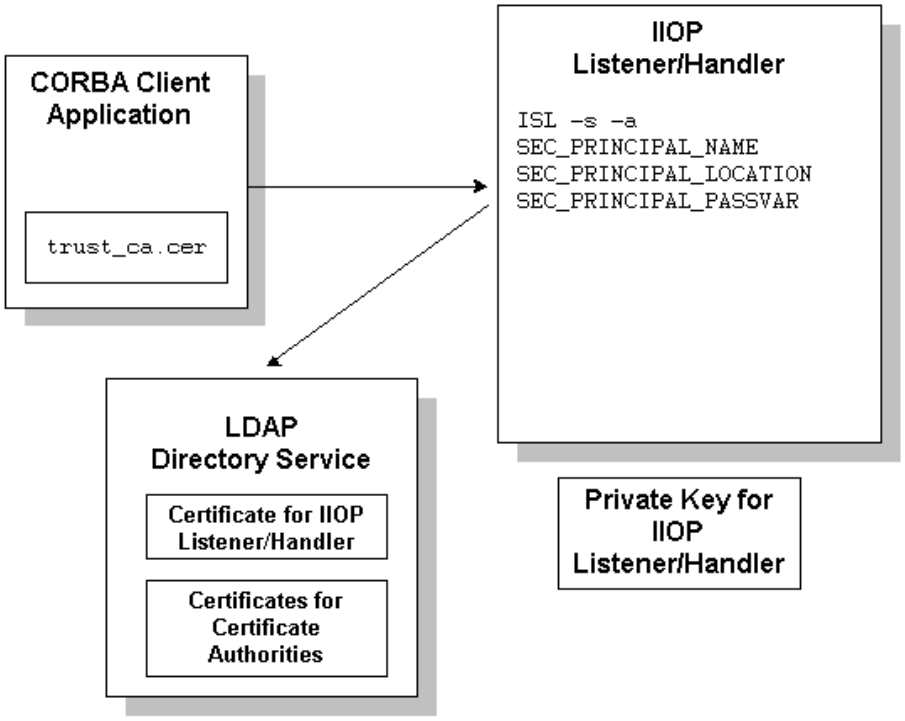
Step	Description
1	Set up an LDAP-enabled directory service. You will be prompted for the name of the LDAP server during the installation of the BEA Tuxedo product.
2	Install the license for the SSL protocol.
3	Obtain a digital certificate and private key for the IIOP Listener/Handler from a certificate authority.
4	Publish the digital certificates for the IIOP Listener/Handler and the certificate authority in the LDAP-enabled directory service.
5	Define the `SEC_PRINCIPAL_NAME`, `SEC_PRINCIPAL_LOCATION`, and `SEC_PRINCIPAL_PASSVAR` parameters for the ISL server process in the `UBBCONFIG` file.
6	Set the `SECURITY` parameter in the `UBBCONFIG` file to `NONE`.
7	Define a port for secure communication on the IIOP Listener/Handler using the `-S` option of the ISL command.
8	Create a Trusted Certificate Authority file (`trust_ca.cer`) that defines the certificate authorities trusted by the IIOP Listener/Handler.
9	Use the `tmloadcf` command to load the `UBBCONFIG` file.
10	Optionally, create a Peer Rules file (`peer_val.rul`) for the IIOP Listener/Handler.
11	Optionally, modify the LDAP Search filter file to reflect the directory hierarchy in place in your enterprise.

If you use the SSL protocol with password authentication, you need to set the `SECURITY` parameter in the `UBBCONFIG` file to desired level of authentication and if appropriate, configure

the Authentication Server (AUTHSRV). For information about the administration steps for password authentication, see "Password Authentication" on page 3-5.

Figure 3-4 illustrates the configuration of a CORBA application that uses the SSL protocol.

Figure 3-4 Configuration for Using the SSL Protocol in a CORBA Application



Certificate Authentication

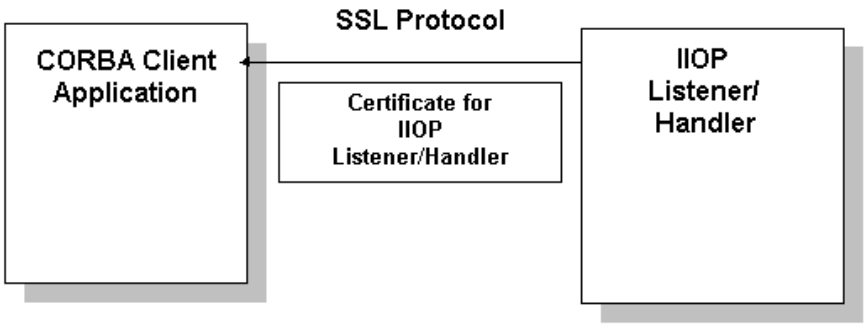
Certificate authentication requires that each side of an SSL connection proves its identity to the other side of the connection. In the CORBA security environment, the IIOP Listener/Handler presents its digital certificate to the principal who initiated the SSL connection. The initiator then provides a chain of digital certificates that are used by the IIOP Listener/Handler to verify the identity of the initiator.

Once a chain of digital certificates is successfully verified, the IIOP Listener/Handler retrieves the value of the distinguished name from the subject of the digital certificate. The CORBA security environment uses the e-mail address element of the subject's distinguished name as the identity of the principal. The IIOP Listener/Handler uses the identity of the principal to impersonate the principal and establish a security context between the initiating application and the BEA Tuxedo domain.

Once the principal has been authenticated, the principal that initiated the request and the IIOP Listener/Handler agree on a cipher suite that represents the type and strength of encryption that they both support. They also agree on the encryption key and synchronize to start encrypting all subsequent messages.

Figure 3-5 provides a conceptual overview of the certificate authentication.

Figure 3-5 Certificate Authentication



Commonly, X.509 V3 CA certificates are required to contain the Basic Constraints extension, marked as being from a Certificate Authority (CA), and marked as a critical extension (see IETF RFC 2459). Ensuring that V3 CA certificates protects against non-CA certificates from masquerading as intermediate CA certificates.

For more information, please refer to the following URL:

`http://www.ietf.org/rfc/rfc2459.txt`

Note: This default behavior will not check Basic Constraints on X.509 V1 and V2 certificates, as these versions of X.509 certificates do not support certificate extensions.

There is a mechanism provided to control the level of enforcement that will be performed in order to avoid problems with some customer's applications:

The mechanism is used by setting the value of the environment variable TUX_SSL_ENFORCECONSTRAINTS. The levels of enforcement are as follows:

0

This level disables the enforcement entirely. This is not recommended as a solution unless you really have no other choice.

For example, a customer has purchased certificates from a commercial CA and the chain does not pass the new checks. Most current commercial CA certificates should work under the default level 1 setting.

```
TUX_SSL_ENFORCECONSTRAINTS=0
```

1

This level is the default. No checking is performed on V1 or V2 certificates in the certificate chain. The Basic Constraints for V3 CA certificates are checked and the certificates are verified to be CA certificates.

```
TUX_SSL_ENFORCECONSTRAINTS=1
```

2

This level does the same checking as level 1, and additionally enforces two more requirements:

- All CA certificates in the certificate chain must be V3 certificates.
- The Basic Constraints extensions of the CA certificates must be marked as "critical" in accordance with IETF RFC 2459.

  This is not the default setting because a number of current commercially available V3 CA certificates do not mark the Basic Constraints as critical.

```
TUX_SSL_ENFORCECONSTRAINTS=2
```

How Certificate Authentication Works

Certificate authentication works in the following manner:

1. The initiating application accesses the BEA Tuxedo domain in one of the following ways:
   - Through the CORBA INS Bootstrapping mechanism. Use this mechanism if you are using a client ORB from another vendor. For more information about using CORBA INS, see *CORBA Programming Reference* in the BEA Tuxedo online documentation.

  – The BEA Bootstrapping mechanism. Use this mechanism if you are using the BEA client ORB.

2. The initiating application instantiates the Bootstrap object with a URL in the form of `corbaloc://host:port` or `corbalocs://host:port` and controls the requirement for protection by setting attributes on the `SecurityLevel2::Credentials` object returned as a result of the `SecurityLevel2::PrincipalAuthenticator::authenticate` operation.

Note: You can also use the `SecurityLevel2::Current::authenticate()` method to secure the bootstrapping process and specify that certificate authentication is to be used.

3. The initiating application obtains the digital certificates and the private key of the principal. Retrieval of this information may require proof material to be supplied to gain access to the principal's private key and certificate. The proof material typically is a pass phrase rather than a password.

   The security context is established as result of a `SecurityLevel2::PrincipalAuthenticator::authenticate()` method.

   The IIOP Listener/Handler receives and validates the application's digital certificate as part of the authentication process.

4. If the verification succeeds, the BEA Tuxedo system constructs a `Credentials` object. The `Credentials` object for the principal represents the security context for the current thread of execution.

5. The initiating application invokes a CORBA object in the BEA Tuxedo domain using an object reference.

6. The request is packaged into an IIOP request and is forwarded to the IIOP Listener/Handler that associates the request with the established security context.

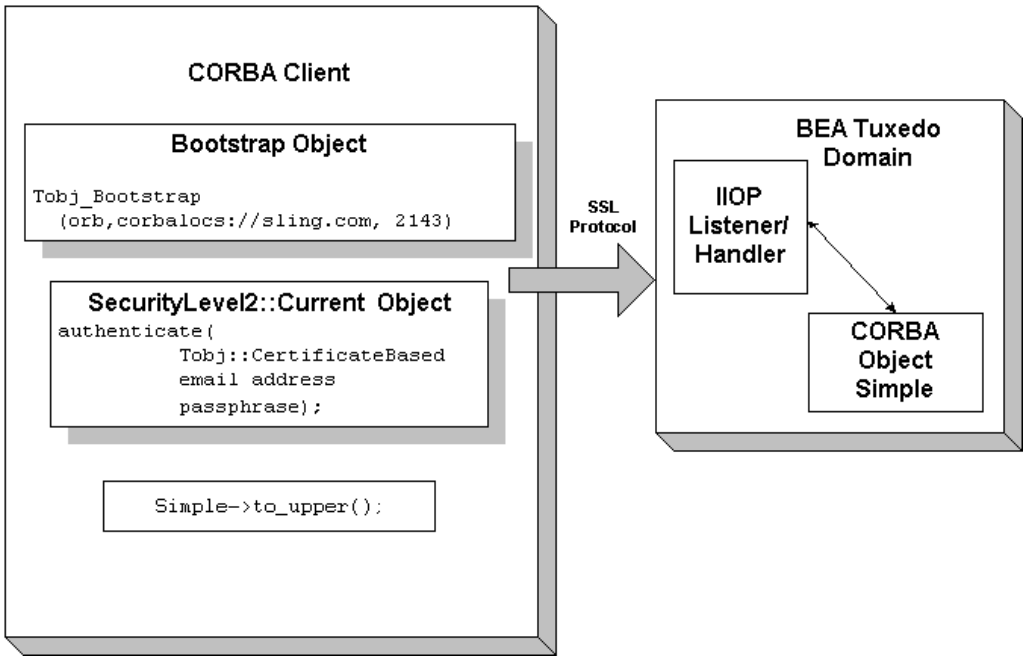
7. The request is digitally signed and encrypted before it is sent to the IIOP Listener/Handler. The BEA Tuxedo system performs the signing and sealing of requests.

8. The IIOP Listener/Handler receives the request from the initiating application. The request is decrypted.

9. The IIOP Listener/Handler retrieves the e-mail component of the subjectDN of the principal's and uses that as the identity of the user.

10. The IIOP Listener/Handler forwards the request, along with the associated tokens of the principal, to the appropriate CORBA object.

Figure 3-6 How Certificate Authentication Works



Development Process for Certificate Authentication

To use certificate authentication in a CORBA application, you need to install a license that enables the use of the SSL protocol and PKI. For information about installing the license, see *Installing the BEA Tuxedo System*.

Using certificate authentication in a CORBA application includes administration and programming steps. Table 3-5 and Table 3-6 list the administration and programming steps for certificate authentication. For a detailed description of the administration steps, see "Managing Public Key Security" on page 4-1 and "Configuring the SSL Protocol" on page 6-1.

Table 3-5 Administration Steps for Certificate Authentication

Step	Description
1	Set up an LDAP-enabled directory service. You will be prompted for the name of the LDAP server during the installation of the BEA Tuxedo product.
2	Install the license for the SSL protocol.
3	Obtain a digital certificate and private key for the IIOP Listener/Handler from a certificate authority.
4	Obtain digital certificates and private keys for the CORBA client applications from a certificate authority.
5	Store the private key files for the CORBA client applications and the IIOP Listener/Handler in the Home directory of the user or in `$TUXDIR/udataobj/security/keys`.
6	Publish the digital certificates for the IIOP Listener/Handler, the CORBA applications, and the certificate authority in the LDAP-enabled directory service.
7	Define the `SEC_PRINCIPAL_NAME`, `SEC_PRINCIPAL_LOCATION`, and `SEC_PRINCIPAL_PASSVAR` for the ISL server process in the `UBBCONFIG` file.
8	Set the `SECURITY` parameter in the `UBBCONFIG` file to `USER_AUTH`, `ACL`, or `MANDATORY_ACL`.
9	Configure the Authentication Server (`AUTHSRV`) in the `UBBCONFIG` file.
10	Use the `tpusradd` and `tpgrpadd` commands to define the authorized Users and Groups of your CORBA application.
11	Define a port for SSL communication on the IIOP Listener/Handler using the `-S` option of the ISL command.
12	Enable certificate authentication in the IIOP Listener/Handler using the `-a` option of the ISL command.
13	Create a Trusted Certificate Authority file (`trust_ca.cer`) that defines the certificate authorities trusted by the IIOP Listener/Handler.
12	Create a Trusted Certificate Authority file (`trust_ca.cer`) that defines the certificate authorities trusted by the CORBA client application.

Table 3-5 Administration Steps for Certificate Authentication (Continued)

Step	Description
13	Use the `tmloadcf` command to load the `UBBCONFIG` file. You will be prompted for the password of the IIOP Listener/Handler defined in the `SEC_PRINCIPAL_NAME` parameter.
14	Optionally, create a Peer Rules file (`peer_val.rul`) for both the CORBA client application and the IIOP Listener/Handler.
15	Optionally, modify the LDAP Search filter file to reflect the directory hierarchy in place in your enterprise.

Figure 3-7 illustrates the configuration of a CORBA application that uses certificate authentication.

Figure 3-7 Configuration for Using Certificate Authentication in a CORBA Application

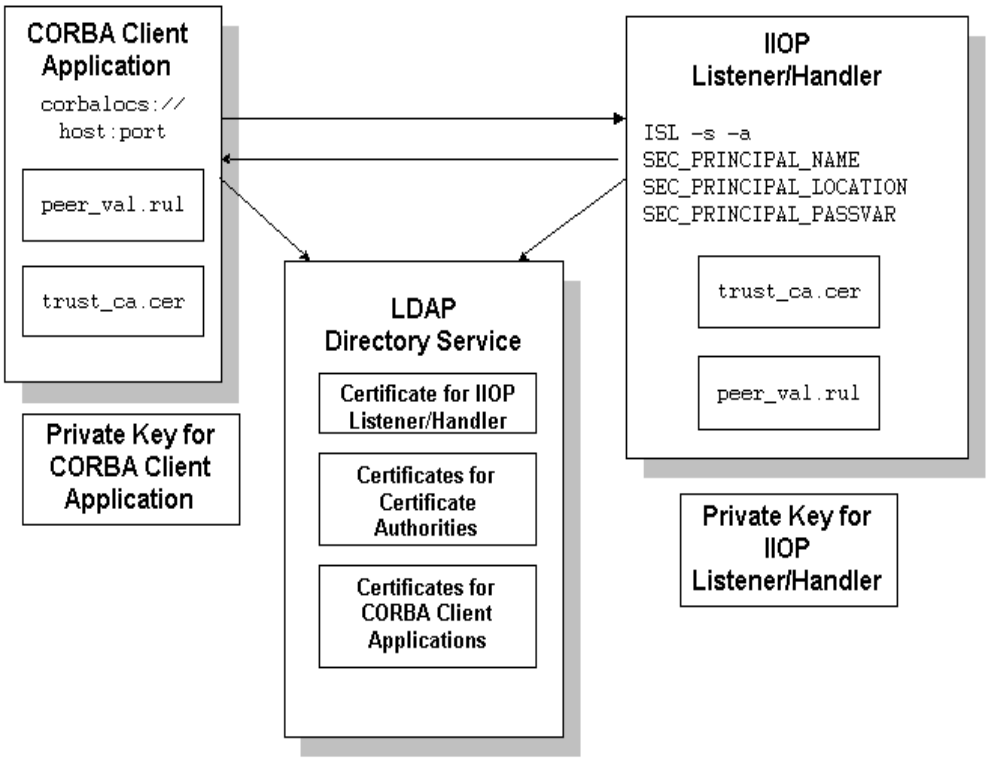


Table 3-6 lists the programming steps for using certificate authentication in a CORBA application. For more information, see "Writing a CORBA Application That Implements Security" on page 9-1.

Table 3-6 Programming Steps for Certificate Authentication

Step	Description
1	Write application code that uses the `corbaloc` or `corbalocs` URL address formats of the Bootstrap object. Note that the CommonName in the Distinguished Name of the certificate of the IIOP Listener/Handler must match exactly the host name provided in the URL address format. For more information on the URL address formats, see "Using the Bootstrapping Mechanism" on page 9-1. You can also use the CORBA INS bootstrap mechanism to object a reference to a PrincipalAuthenticator object in the BEA Tuxedo domain. For more information about using CORBA INS, see the *CORBA Programming Reference*.
2	Write application code that uses the `authenticate()` method of the `SecurityLevel2::PrincipalAuthenticator` interface to perform authentication. Specify `Tobj::CertificateBased` for the method argument and the pass phrase for the private key as the `auth_data` argument for `Security::Opaque`.

Using an Authentication Plug-in

The BEA Tuxedo product allows the integration of authentication plug-ins into a CORBA application. The BEA Tuxedo product can accommodate authentication plug-ins using various authentication technologies, including shared-secret password, one-time password, challenge-response, and Kerberos. The authentication interface is based on the generic security service (GSS) application programming interface (API) where applicable and assumes authentication plug-ins have been written to the GSSAPI.

If you chose to use an authentication plug-in, you must configure the authentication plug-in in the registry of the BEA Tuxedo system. For more detail about the registry, see "Configuring Security Plug-ins" on page 8-1.

For more information about an authentication plug-ins, including installation and configuration procedures, see your BEA account executive.

Authorization

Authorization allows system administrators to control access to CORBA applications. Specifically, an administrator can use authorization to allow or disallow principals to use resources or services provided by a CORBA application.

The CORBA security environment supports the integration of authorization plug-ins. Authorization decisions are based in part on the user identity represented by an authorization token. Authorization tokens are generated during the authentication process so coordination between the authentication plug-in and the authorization plug-in is required.

If you chose to use an authorization plug-in, you must configure the authorization plug-in the registry of the BEA Tuxedo system. For more detail about the registry, see "Configuring Security Plug-ins" on page 8-1.

For more information about authorization plug-ins, including installation and configuration procedures, see your BEA account executive.

Auditing

Auditing provides a means to collect, store, and distribute information about operating requests and their outcomes. Audit-trail records may be used to determine which principals performed, or attempted to perform, actions that violated the configured security policies of a CORBA application. They may also be used to determine which operations were attempted, which ones failed, and which ones successfully completed.

The current implementation of the auditing feature supports the recording of logon failures, impersonation failures, and disallowed operations into the `ULOG` file. In the case of disallowed operations, the value of the parameters to the operation are not provided because there is no way to know the order and data types of the parameter for an arbitrary operation. Audit entries for logon and impersonation include the identity of the principal attempting to be authenticated. For information about setting up the `ULOG` file, see *Setting Up a BEA Tuxedo Application*.

You can enhance the auditing capabilities of your CORBA application by using an auditing plug-in. The BEA Tuxedo system will invoke the auditing plug-in at predefined execution points, usually before an operation is attempted and then when potential security violations are detected or when operations are successfully completed. The actions taken to collect, process, protect, and distribute auditing information depend on the capabilities of the auditing plug-in. Care should be taken with the performance impact of audit information collection, especially successful operation audits, which may occur at a high rate.

Auditing decisions are based partly on user identity, which is stored in an auditing token. Because auditing tokens are generated by the authentication plug-in, providers of authentication and auditing plug-ins need to ensure that these plug-ins work together.

The purpose of an auditing request is to record an event. Each auditing plug-in returns one of two responses: `success` (the audit succeeded and the event was logged) or `failure` (the audit failed

and the event was not logged the event). An auditing plug-in is called once before the operation is performed and once after the operation completes.

- The preoperation audit allows the auditing of both attempts to call an operation, and also allows storage of input data for the postoperation check.
- The postoperation audit reports the status of the completion of an operation. For failure status, the postoperation audit is called to report a potential security violation. Usually this type of report is issued when a preoperation or postoperation authorization check fails or when some other potential security attack is detected.

Multiple implementations of the auditing plug-in can be used in a CORBA application. Using multiple authorization plug-ins causes more than one preoperation and postoperation auditing operation to be performed.

When using multiple auditing plug-ins, all the plug-ins are placed under a single master auditing plug-in. Each subordinate authorization plug-in returns `SUCCESS` or `FAILURE`. If any plug-in fails the operation, the auditing master plug-in determines the outcome to be `FAILURE`. Other error returns are also considered `FAILURE`. Otherwise, `SUCCESS` is the outcome.

In addition, a BEA Tuxedo system process may call an auditing plug-in when a potential security violation occurs. (Suspicion of a security violation arises when a preoperation or postoperation authorization check fails or when an attack on security is detected.) In response, the auditing plug-in performs a postoperation audit and returns whether the audit succeeded.

The auditing process is somewhat different for users of the auditing feature provided by the BEA Tuxedo product and users of auditing plug-ins. The default auditing feature does not support preoperation audits. If the default auditing feature receives a preoperation audit request, it returns immediately and does nothing.

If you chose to use an auditing plug-in other than the default auditing plug-in, you must configure the auditing plug-in the registry of the BEA Tuxedo system. For more detail about the registry, see “Configuring Security Plug-ins” on page 8-1.

For more information about auditing plug-ins, including installation and configuration procedures, see your BEA account executive.

PKI Plug-ins

The BEA Tuxedo product provides a PKI environment which includes the SSL protocol and the infrastructure needed to use digital certificates in a CORBA application. However, you can use the PKI interfaces to integrate a PKI plug-in that supplies custom message-based digital signature

and message-based encryption to your CORBA applications. Table 3-7 describes the PKI interfaces.

Table 3-7 PKI Interfaces

PKI Interface	Description
Public key initialization	Allows public key software to open public and private keys. For example, gateway processes may need to have access to a specific private key in order to decrypt messages before routing them.
Key management	Allows public key software to manage and use public and private keys. Note that message digests and session keys are encrypted and decrypted using this interface, but no bulk data encryption is performed using public key cryptography. Bulk data encryption is performed using symmetric key cryptography.
Certificate lookup	Allows public key software to retrieve X.509v3 digital certificates for a given principal. Digital certificates may be stored using any appropriate certificate repository, such as Lightweight Directory Access Protocol (LDAP).
Certificate parsing	Allows public key software to associate a simple principal name with an X.509v3 digital certificate. The parser analyzes a digital certificate to generate a principal name to be associated with the digital certificate.
Certificate validation	Allows public key software to validate an X.509v3 digital certificate in accordance with specific business logic.
Proof material mapping	Allows public key software to access the proof materials needed to open keys, provide authorization tokens, and provide auditing tokens.

The PKI interfaces support the following algorithms:

- Public key algorithms: Rivest, Shamir, and Adelman (RSA) and Digital Signature Algorithm (DSA)

- Symmetric key algorithms:
  - Data Encryption Standard for Cipher Block Chaining (DES-CBC)
  - Two-key triple-DES
  - Rivest's Cipher 4 (RC4)
- Message digest algorithms:
  - Message Digest 5 (MD5)
  - Secure Hash Algorithm 1 (SHA-1)

If you chose to use a PKI plug-in, you must configure the PKI plug-in in the registry of the BEA Tuxedo system. For more detail about the registry, see "Configuring Security Plug-ins" on page 8-1.

For more information about PKI plug-ins, including installation and configuration procedures, see your BEA account executive.

Commonly Asked Questions About the CORBA Security Features

The following sections answer some of the commonly asked questions about the CORBA security features.

Do I Have to Change the Security in an Existing CORBA Application?

The answer is no. If you are using security interfaces from previous versions of the WebLogic Enterprise product in your CORBA application there is no requirement for you to change your CORBA application. You can leave your current security scheme in place and your existing CORBA application will work with CORBA applications built with BEA Tuxedo 8.0 or later.

For example, if your CORBA application consists of a set of server applications which provide general information to all client applications which connect to them, there is really no need to implement a stronger security scheme. If your CORBA application has a set of server applications which provide information to client applications on an internal network which provides enough security to detect sniffers, you do not need to implement the additional security features.

Can I Use the SSL Protocol in an Existing CORBA Application?

The answer is yes. You may want to take advantage of the extra security protection provided by the SSL protocol in your existing CORBA application. For example, if you have a CORBA server application which provides stock prices to a specific set of client applications, you can use the SSL protocol to make sure the client applications are connected to the correct CORBA server application and that they are not being routed to a fake CORBA server application with incorrect data. A username and password is sufficient proof material to authenticate the client application. However, by using the SSL protocol, the message request/reply information can be protected as an additional level of security.

The SSL protocol offers CORBA applications the following benefits:

- Protection of the entire conversation including the initial bootstrapping process. The SSL protocol protects against Man-In-The-Middle attacks, replay attacks, tampering, and sniffing.
- Even if you only use the default settings, the SSL protocol provides signed and sealed protection since the default encryption settings are a minimum of 56 bits by default.
- Client verification of the connected IIOP Listener/Handler using the digital certificate of the IIOP Listener/Handler. The client application can then apply additional security rules to restrict access to the client application by the IIOP Listener/Handler. This protection also applies to IIOP Listener/Handlers connecting to remote server applications when using callback objects.

To use the SSL protocol in a CORBA application, set up the infrastructure to use digital certificates, change the command-line options on the ISL server process to use the SSL protocol, and configure a port for secure communications on the IIOP Listener/Handler. If your existing CORBA application uses password authentication, you can use that code with the SSL protocol. If your CORBA C++ client application does not already catch the `InvalidDomain` exception when resolving initial references to the Bootstrap object and performing authentication, write code to handle this exception. For more information, see “PKI Plug-ins” on page 3-22.

When Should I Use Certificate Authentication?

You might be ready to migrate your existing CORBA application to use Internet connections between the CORBA application and Web browsers and commercial Web servers. For example, users of your CORBA application might be shopping over the Internet. The users must be confident that:

- They are in fact communicating with the server at the online store and not an impostor that mimics the store's server to get credit card information.
- The data exchanged between the user of the CORBA application and the online store will be unintelligible to network eavesdroppers.
- The data exchanged with the online store will arrive unaltered. An instruction to order $500 worth of merchandise must not accidently or maliciously become a $5000 order.

In these situations, the SSL protocol and certificate authentication offer CORBA applications the maximum level of protection. In addition to the benefits achieved through the use of the SSL protocol, certificate authentication offers CORBA applications:

- IIOP Listener/Handler verification of the client application that initiates a request using the digital certificate of the client application. In addition, the IIOP Listener/Handler can apply additional rules which restrict access to the client application based on the identity established by the digital certificate. A remote ORB acting as a server application can also be configured to allow mutual authentication and verify the identity of a client application based on a digital certificate.
- Inside the BEA Tuxedo domain, the client application can still have a BEA Tuxedo username and password. The IIOP Listener/Handler maps the identity defined in a digital certificate to a BEA Tuxedo username and password thus allowing existing CORBA applications to have an identity in native CORBA server applications.

For more information, see "PKI Plug-ins" on page 3-22.