MTR 99W0000027

MITRE TECHNICAL REPORT

# Security Services Application Programming Interface (SS API) Developer's Security Guidance

**March 2000**

Amgad Fayad
Don Faatz

| | | | |
|---|---|---|---|
| **Sponsor:** | DISA | **Contract No.:** | DAAB07-99-C-C201 |
| **Dept. No.:** | G024 | **Project No.:** | 0700K6B0-CA |

MITRE

**Center for Integrated Intelligence Systems**
**McLean, Virginia**

MITRE Department Approval:

Kathryn M. Bitting
Department Head
Secure Distributed Computing

MITRE Project Approval:

William J. Quigley
Project Leader, K6Z0

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

| | | |
|---|---|---|
| 1. REPORT DATE<br>**MAR 2000** | 2. REPORT TYPE | 3. DATES COVERED<br>**00-03-2000 to 00-03-2000** |

4. TITLE AND SUBTITLE
**Security Services Application Programming Interface (SS API) Developer's Security Guidance**

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
**MITRE Corporation,202 Burlington Road,Bedford,MA,01730-1420**

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**The original document contains color images.**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | | **65** | |

**THIS PAGE INTENTIONALLY BLANK**

# Abstract

This document describes a specification of security services for distributed applications in the Defense Information Infrastructure (DII) Common Operating Environment (COE). Security services include identification and authentication, encryption, access control, and auditing. The security services are referred to as the COE security services API (COE SS API).

The document includes a high-level description of the COE SS API, a formal Java-based specification, a mapping from the specification to the C and Java programming languages, and sample applications to demonstrate how the COE SS API can be used.

KEYWORDS: Application Programming Interface, API, C, COE SS API, DII COE, Java, Security Services, DII COE

# Acknowledgments

We would like to thank the following people for their contributions:

Dr. Gregory Frazier, SAIC Corporation, for numerous contributions to this effort starting with requirements and design work.

Dr. Jackie Lawrence, INRI Corporation for his valuable input to the requirements and design phase.

Dr. Thom McVitti and Elia Kosnoski, JPL, for valuable input to the requirements phase.

Erik King, Patrick Cesard and Yong Choe, SAIC, for their implementation phase support.

LTC Alex Froede for his guidance.

Tom Gregg, and Russ Reopell, MITRE.

Carol R. Oakes for editing the content and style of this document.

# Table of Contents

# Executive Summary

DII COE applications are usually distributed applications. This document defines the COE SS API, a security services specification which can be used to provide identification and authentication, encryption, access control, and auditing to COE distributed applications.

In a distributed application with multiple tiers of clients and servers (n-tier applications), security services are needed at multiple levels. Point-to-point and End-to-end communications must be secured. In addition, multicast and file security must be secured.

This release of the COE SS API will provide a solution for the point-to-point problem. Future releases of the COE SS API will address the other components of n-tier applications.

The formal specification of the COE SS API is provided in Java in a generic fashion without utilizing Java-specific features. A mapping to the C, and Java, programming languages is provided along with sample application programs. The relationship between the COE SS API and other distributed application security mechanisms is also discussed.

**Section 1**

## 1.1 Purpose

In addition to their intended functionality, information systems need to exhibit other properties such as security. System security involves providing appropriate data integrity protection, data confidentiality protection, and accountability for actions taken while using the system. Some level of security can be provided without specific application contributions; however, in many cases application software needs to invoke security services explicitly to achieve the desired level of system security. The Defense Information Systems Agency (DISA) issued tasking to identify a set of security services application programming interfaces (APIs) and implementations for the Defense Information Infrastructure (DII) Common Operating Environment (COE). These APIs will be used by DII COE applications to contribute to system security. This document presents security guidance for developers of the DII COE (herein called the *COE SS API.*)

DII COE applications are typically distributed and they involve one or more of the interactions illustrated in Figure 1-1. A software client, acting on behalf of a user, interacts with a server. The server may interact with other servers or data sources such as a relational database management system. The application may need to provide security for any of these interactions. It may need to provide security for data manipulated by the client, the servers or the data sources. Finally, it may need to provide end-to-end security from the client, through the intermediate servers, to the data sources. To provide security, the application needs access to a variety of security services, such as identification and authentication, encryption, access control, and auditing. The COE SS API will provide applications with access to these services.

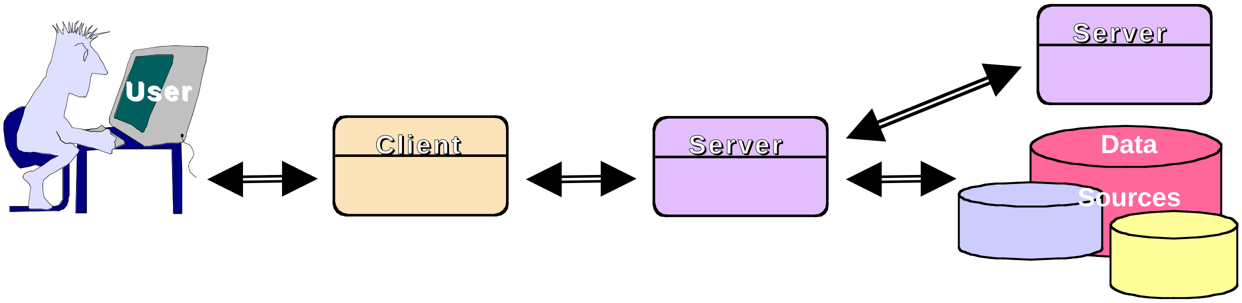

**Figure 1-1. Typical Distributed Application**

## 1.2 Scope

In selecting APIs and their associated implementations, three factors must be considered:

- Level of abstraction
- Location of security policy
- Implementation technology

The level of abstraction presented by an interface represents a tradeoff between ease of use and detailed control of implementation mechanisms. A high level of abstraction provides less control over the implementation mechanism, but reduces the application learning curve. A smaller learning curve reduces the potential for vulnerabilities introduced by the developer who is incorrectly using the security services.

In the COE SS API, the goal is to provide a level of abstraction for security services that matches the abstraction level of other services being used. For example, if the application developer uses an interface such as OpenConnection() to create an unsecure communications path to another server, a secure connection should use a similar interface, such as OpenSecureConnection(). That is, the developer should be able to interact with a secure service in the same way he/she interacts with the same service that does not provide security.

Security policy determines the level of protection needed in a particular system. While an application program may need to enforce portions of the security policy, the application should not dictate the policy because applications can be used in many different environments. Hence, it is usually inappropriate to code security policy directly into an application. For this reason, the COE SS API will place the system security policy inside the security services implementations.

Applications that use the COE SS API implementations will have the appropriate system policy enforced for them by the COE SS API implementations. Applications will be able to query the policy in effect through the API and raise the level of security protection. This process will ensure that the system security policy will be applied consistently across all applications, and that changes to the policy will not affect the applications themselves. The system policy specification is beyond the scope of this document.

A security services API implementation should be provided by current technology. This technology should have widespread use in order to have interoperability with other commercial technologies, such as World Wide Web-based products and office automation products.

The target COE version for the initial implementation of the COE SS API is COE 4.x. Since the C and Java programming languages are the languages used for developing applications in the COE, mappings to these two languages is provided.

## 1.3 Background

In researching potential APIs and implementations for the COE SS API, several products were considered: Northern Telecom's Entrust Toolkit, CyberSafe's TrustBroker, Netscape's Security Services (NSS), and Security Dynamics' SecureSight. While each of these products offers some of the needed security services, none provide a complete solution for the DII COE. Each product provides vendor-specific APIs to security services. Some of these products provide the Internet Engineering Task Force's (IETF) Generic Security Services API (GSS-API). However, each vendor extends the GSS API such that applications developed using one vendor's GSS-API implementation might not be compatible with an implementation of a similar product by another vendor. Also, GSS-API uses unfamiliar abstractions to application developers, thereby requiring a steep learning curve.

Based on this research and the cost of acquiring these products, the DISA DII COE Chief Engineer decided to pursue security services for the COE SS API using NSS. NSS uses the Secure Sockets Layer (SSL) protocol to provide mutual authentication, integrity, and confidentiality protection for data over point-to-point network connections. Numerous commercial products can interoperate with applications that use NSS due to the popularity of the SSL protocol.

In July 1998, a task was initiated to design a COE SS API that provides data integrity and data confidentiality protection for information exchanged between clients and servers using NSS. The COE SS API allows COE developers to write security-aware applications without providing security enforcement mechanisms. The security enforcement mechanisms will be provided in separate DII COE segments, called *bindings*. The COE SS API will mimic the unsecured sockets API to simplify the learning curve of COE application developers.

A COE SS API binding will consist of two parts (Figure 1-2), a thin isolation layer that maps security services to interface functions in the COE SS API and a security services product, such as NSS. The isolation layer will provide two distinct benefits: it will provide a vendor-independent interface definition and a location to incorporate the system security policy. Hence, when an application invokes a security service through the COE SS API, the isolation layer will first check the system security policy to determine which underlying security services are needed and then it will invoke those services in the security services product.

The COE SS API is specified using the Java programming language. As stated earlier, a mapping to the C and Java programming languages is provided. The Java specification differs from the Java mapping in that the mapping makes use of Java-specific features, while the Java specification is more generic, in order to facilitate defining mappings to other languages, such as C.

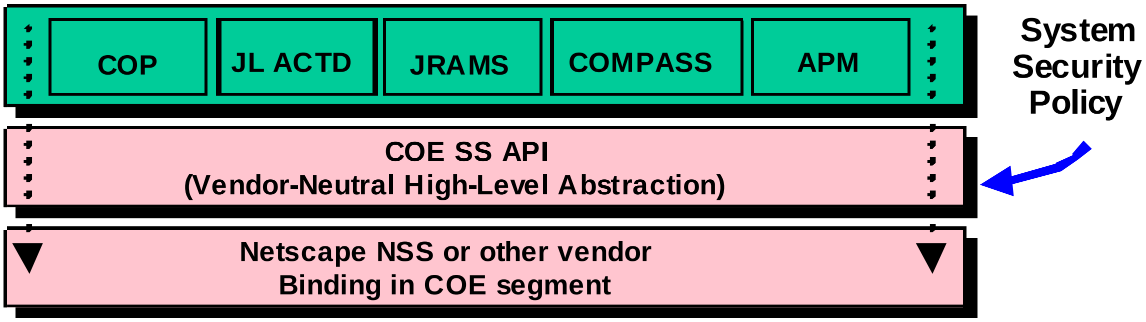

**Figure 1-2. Selected Solution**

## 1.4 Document Structure

Section 1 provides an introduction to the COE SS API and the document. Section 2 presents the elements of the COE SS API that have been defined to support secure point-to-point connections using NSS. Section 3 describes how COE SS API implementations can be incorporated in the DII COE. Appendices provide the Java specification for the COE SS API, mappings of the COE SS API to both the C and Java programming languages, sample C and Java programs, and a description of how the COE SS API relates to other middleware technologies.

Section 2

# Elements of the Security Services API

This section describes the basic elements of the COE SS API. Figure 2-1 illustrates these elements: the Security Context, the Credential, and the Secure Connection. These elements are used by security-aware client and server application programs to achieve secure communications, by creating *instances* of these elements. In addition, a System Security Policy, used by the binding, establishes the initial Security Context instance.

Section 2.4 describes additional services, which will be provided by the COE SS API in the future.

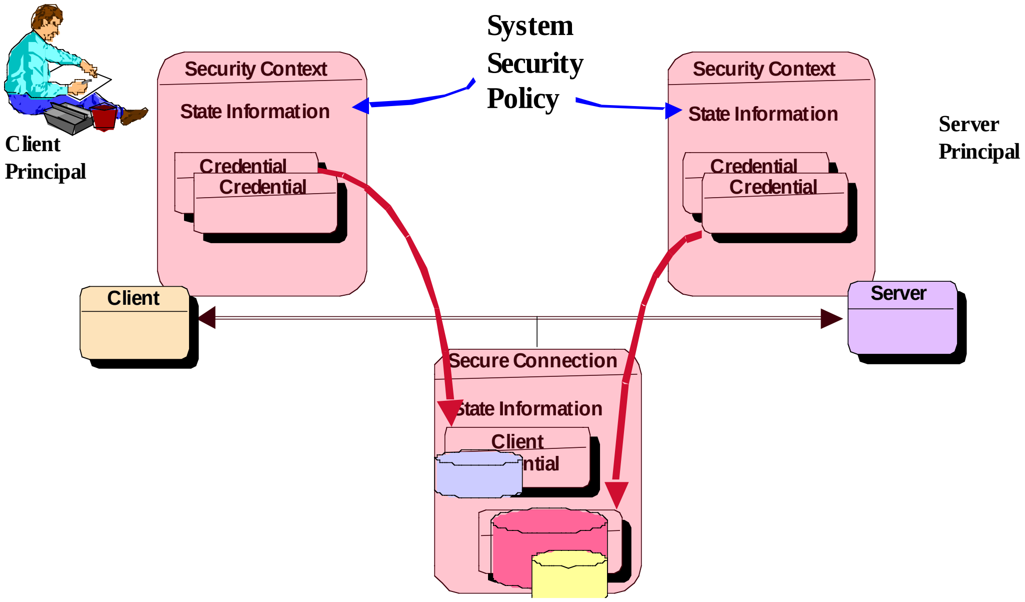

**Figure 2-1. Elements of the COE SS API**

## 2.1 Security Context

The Security Context is a data structure to be maintained by the COE SS API binding that contains information used to create Secure Connections. A system should have a default security policy that describes the level of security protection required between the clients and servers that operate with that system. This policy information is assumed to be encoded and available to the binding as the System Security Policy. This policy might state that all connections require integrity protection but not confidentiality protection.

When a program (client or server) creates its Security Context, the binding reads the System Security Policy and stores it in the context. If the program does not modify the context, this information is used when creating secure connections for the program. The System Security Policy may also contain binding-specific information such as the algorithms to use when providing integrity or confidentiality protection for communication.

The Security Context  provides methods that programs can use to modify information in the Security Context. For example, a program can increase the level of protection on its communication by modifying its Security Context to require confidentiality protection. Programs are not allowed to reduce the level of protection below that specified in the System Security Policy.[1]

## 2.2 Security Credentials

One critical element of the Security Context is the principal's credentials. Credentials are data structures used to identify the principal on whose behalf a program is executing. The actual structure of a credential depends on the technology used in the COE SS API binding. In an NSS binding, a credential would be a principal's public key certificate.  In a Kerberos binding, the credential might be a ticket granting ticket (TGT) or service ticket.

The COE SS API will not provide a method for creating credentials. This will be done either at login to the system or by the creation of the Security Context. Every binding must define a default credential that is used when establishing connections if no other credential is identified for use. If a binding provides multiple credentials, the COE SS API will provide methods to select which is used as the default and which should be used for a particular connection. Further, if permitted by the binding technology, the COE SS API will provide methods to change information in the credentials.

Every binding must provide two attributes for its credentials, an identifier and a principal's name. The identifier must uniquely identify each credential in a program's security context. The identifier can be used by programs to indicate which credential is to be used for a particular purpose. The principal's name, which may or may not be the same for each credential, identifies the principal who is being authenticated by the Security Credentials element.

[1] This document does not address the form or content of the System Policy as it is accessed only by the COE SS API binding and is not directly available to programs using the COE SS API. However, as part of the implementation of the first COE SS API binding, a standard approach to providing the System Policy to bindings should be defined.

## 2.3 Secure Connections

The COE SS API will provide methods to establish or accept connections. Servers will use the methods for accepting connections to wait for an incoming connection from a client. Clients will use methods for establishing a connection to initiate a new connection to a server. At the time the connection methods are invoked, the information from the Security Context of the client and the server will be copied to the Secure Connection element. Future changes in the Security Context of either end will not affect existing connections, but will be used by future connections.

The COE SS API will not address the behavior of the connection methods when the Security Contexts of the client and the server require different levels of protection. Resolution of this behavior may depend on binding technology capabilities and is, therefore, left as an issue for binding developers. However, several approaches can be envisioned and are used in existing systems. For example, the System Policy may specify how the binding will handle differences in protection requirements. The client and the server might negotiate a mutually acceptable level of protection. Failing any of these, connections might simply fail if the client and server cannot agree on an acceptable level of protection.

The COE SS API technology must implicitly assume that the existence of an environment in which programs can use multiple threads to handle blocking input/output (IO) operations. As such, most of the operations defined for Secure Connections are blocking operations. To accommodate existing software, such as the Common Operating Picture (COP) that uses the UNIX `select()` system call to poll multiple IO sources, bindings developers may need to provide a separate select-like mechanism as an add-on to the C programming language binding. This mechanism should not be used in newly developed programs. Also, environments that already have multiple threading support, such as those using Java, will not provide select-like functionality.

## 2.4 Other Services

In addition to providing point-to-point mutual authentication, data integrity, and data confidentiality, future versions of the COE SS API will need to provide other security services, such as secure storage and access control. The basic elements of the COE SS API (described herein as the Security Context and Security Credentials) provide information that many security services will need, such as policy information and identification of principals.

Using these basic elements, additional elements, such as Secure Storage, could be added as shown in Figure 2-2. Secure Storage would contain methods for accessing data stored in signed or sealed files. The System Security Policy would be extended to include a file-signing and encryption policy. The Security Credentials would be extended to include any cryptographic keys needed in signing or sealing a file. A principal would obtain access to a secure file in storage depending on the credential the principal presents and the System

Security Policy used when the file was encrypted. Other security services also could be supported.

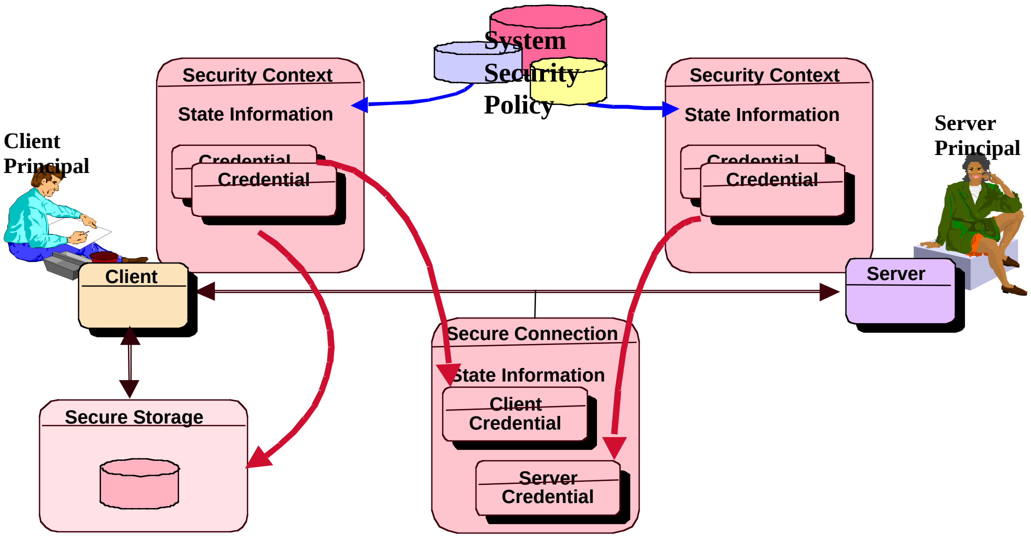

**Figure 2-2. Secure Storage**

Section 3

# COE SS API Implementation and Architecture

The COE kernel will ship with a simple default implementation of the COE SS API that allows security-aware applications to operate but may add little security. Specific security enforcement mechanisms will be added as infrastructure service segments, which will provide the appropriate COE SS API implementation.

Figure 3-1 illustrates this approach. The default implementation of the COE SS API will be provided as part of the COE kernel, thus allowing any applications built with the COE SS API to function, but without added security. As stated earlier, specific implementations of security services, (bindings), will be added as COE segments. When a binding segment is installed, applications built using the COE SS API will automatically use the provided security services. An initial binding to NSS will be provided with the COE.

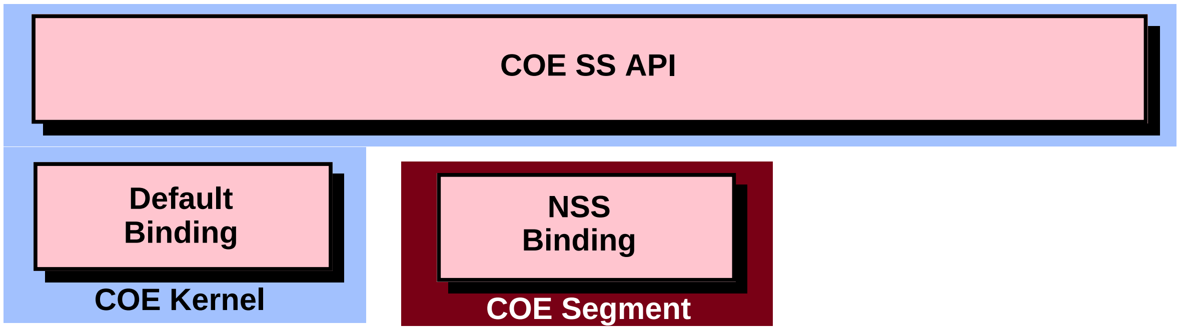

**Figure 3-1. Approach to Providing SS API Bindings**

In addition to bindings, mappings of the COE SS API to COE programming languages must be provided. A particular binding may implement only some of the defined language mappings. In the COE 4.x time frame, mappings to the Java and C programming languages are included in Appendixes B and D. In the future, mappings to other languages, such as C++, are likely to be needed.

The COE SS API was initially specified in Java. The Java language mapping, however, differs from the Java specification (Appendix A). This is because the Java specification does not make full use of Java-specific features to make the specification general enough to map to other languages. An attempt was made to specify the COE SS API in the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) Interface Definition Language (IDL) and take advantage of the language mappings defined from this IDL to implement languages such as Java and C. Unfortunately, the translation process from IDL generated a significant amount of CORBA-specific environment information. This approach was abandoned in favor of using Java as the specification language and manually generated mappings to the required languages.

The system security policy will be dependent on the particular binding. For example, some bindings will require listing allowable cryptographic algorithms that apply to the particular technology that the binding depends on.

Figure 3-2 shows where the COE SS API fits in the architecture that DISA calls the Security Services Architecture Framework (SSAF). The SSAF is an adaptation of a standard security framework called the Common Data Security Architecture (CDSA). The Open Group is responsible for maintaining the CDSA specification.

CDSA defines a low-level API layer called the Common Security Services Module (CSSM). The CSSM interfaces with lower-level security service modules. In addition, CSSM interfaces with a middleware layer (high-level APIs). As of this writing, this layer is not specified in detail by the Open Group. Recently, MITRE presented the COE SS API and the SSAF in an Open Group meeting in order to initiate an Open Group action item defining a standard high-level API for CDSA.

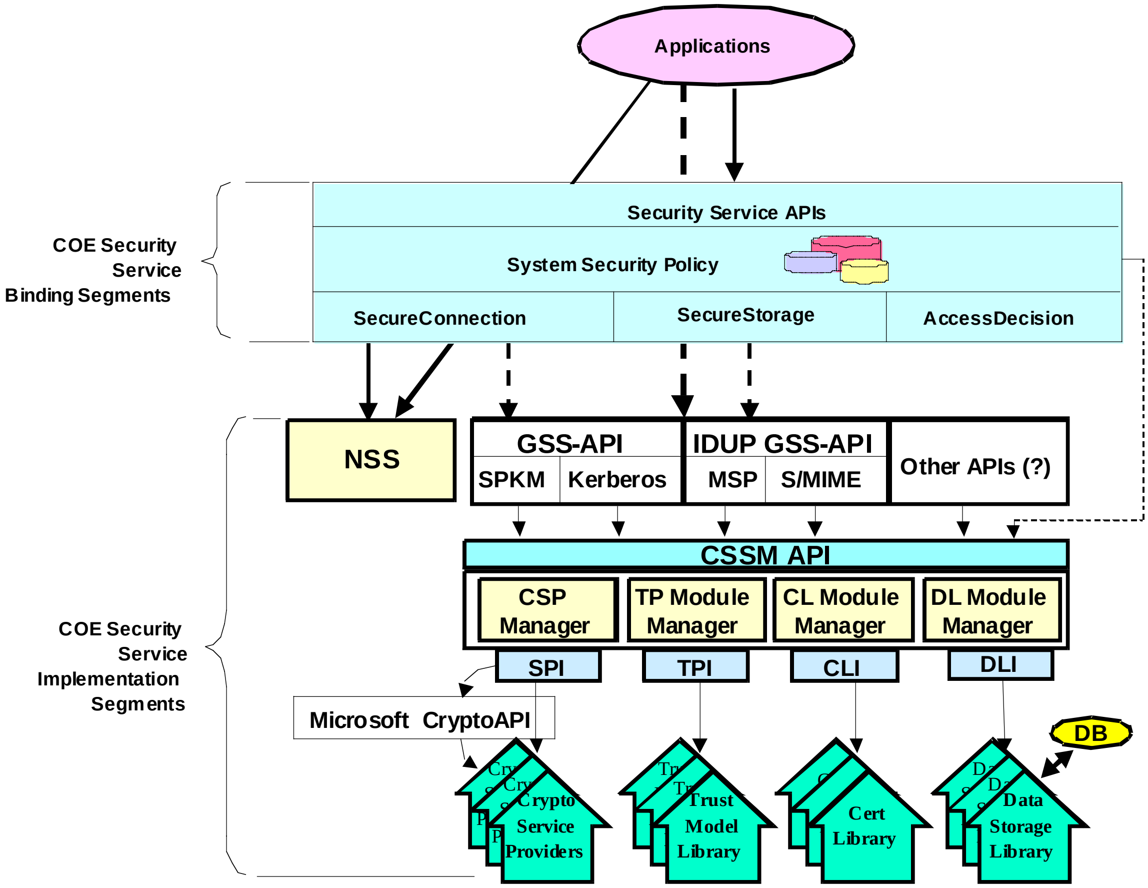

**Figure 3-2. SSAF Architecture**

**Appendix A**

# Java Specification for the COE SS API

## Class mil.disa.dii.kernel.securityServices.Credential

```
java.lang.Object
   |
   +----mil.disa.dii.kernel.securityServices.Credential
```

public class **Credential**

extends Object. Credential is the interface to the operator's identity. Credential objects have the ability to describe themselves, and a limited mechanism to allow the application to control how it can be used for authentication.

There are no constructor methods for credentials available to applications. Credentials are created or activated during authentication of the principal using an application. Hence, the credentials are either created by user login to the system or by `securityContext` constructor. There is an optional `copy` method that allows an application to make a copy of an existing credential before making modifications.

**See Also:**

SecurityContext

Method Index:

**copy**()

Make a copy of an existing credential.

**getCredentialIdentifier**()

Return a string that uniquely identifies this amongst all of the Credentials available to the application.

**getExpirationDate**()

Get the date/time at which the Credential did or will expire (cease to be a valid Credential).

**getNameOfPrincipal**()

Return a string that uniquely identifies the operator.

**getValidationDate**()

Get the date/time at which the Credential became or will become valid.

Methods:

### copy

public Credential copy() throws SecurityException

Make a copy of an existing credential.

**Returns:**

a copy of the Credential.

**Throws:** `SecurityException`

thrown if the binding does not support delegation.

### getCredentialIdentifier

public String getCredentialIdentifier()

Return a string that uniquely identifies this amongst all of the Credentials available to the application.

**Returns:**

the "name" of the Credential.

### getNameOfPrincipal

public String getNameOfPrincipal()

Return a string that uniquely identifies the operator. This is the string that is made available to the process at the other end of a connection established using this Credential.

**Returns:**

the "name" of the operator.

**See Also:**

getOtherPrincipalName

### getExpirationDate

public Date getExpirationDate()

Get the date/time at which the Credential did or will expire (cease to be a valid Credential).

**Returns:**

the date/time that the Credential did/will expire.

**See Also:**

Date, getValidationDate

### getValidationDate

public Date getValidationDate()

Get the date/time at which the Credential became or will become valid.

**Returns:**

the date/time that the Credential did/will become valid.

**See Also:**

Date, getExpirationDate, setExpirationDate

## Class mil.disa.dii.kernel.securityServices.SecurityContext

```
java.lang.Object
   |
   +----mil.disa.dii.kernel.securityServices.SecurityContext
```

public class **SecurityContext**

extends Object. The SecurityContext is the interface to security services. It gives the application access to attributes of the security services (allowing applications to specify and/or learn the level of service) and to credentials available to the application. It is also the source of `SecureConnection` objects.

**See Also:**

Credential, SecureConnection

Constructor Index:

**SecurityContext**(boolean)

Create a new `SecurityContext`.

**SecurityContext**(String, boolean)

Create a new `SecurityContext`, prompting for operator authentication if necessary.

**SecurityContext**(String, String, boolean)

Create a new `SecurityContext` using the operator authentication passed in as arguments.

Method Index:

**acceptConnection**()

Accept a secure connection from another process.

**acceptConnection**(Credential)

Accept a secure connection to another process.

**establishConnection**(String, int)

Establish a secure connection to another process.

**establishConnection**(String, int, Credential)

Establish a secure connection to another process.

**getAvailableCredentials**()

Get the credentials that are available to the process.

**getBindingName**()

Return a string that describes the security implementation.

**getDefaultCredential**()

Get the Credential that is the default when establishing connections.

**getEncryption**()

Return the *real* encryption state.

**listen**(int)

Listen for connection requests from clients.

**setDefaultCredential**(Credential)

Set the Credential that will be used to establish future connections when a Credential is not specified for that specific connection.

**setEncryption**(boolean)

Specify whether communication connections created via the SecurityContext will encrypt communication.

Constructors:

**SecurityContext**

public SecurityContext(boolean allowAnonymous) throws SecurityException

Create a new `SecurityContext`. If authentication cannot be obtained from the environment, throw a `SecurityException`. (This constructor will not prompt the operator for authentication.)

**Parameters:**

allowAnonymous - true allows anonymous credentials.

**Throws:** SecurityException

Thrown if the operator's authentication cannot be obtained from the environment or if the constructor fails for any reason.

**SecurityContext**

public SecurityContext(String prompt,

boolean allowAnonymous) throws SecurityException

Create a new `SecurityContext`, prompting for operator authentication if necessary. If authentication cannot be obtained from the environment, prompt the operator for authentication (e.g. name and/or password). The number of tries given to the operator is established by the binding and the local security administrator. Applications that catch the `SecurityException` thrown by this method should not call the method again to give the operator another chance to authenticate. This constructor can be used by clients or any machine that has an operator present (in order to type in the name and/or password.)

**Parameters:**

prompt - If the binding prompts the operator to enter authentication information (e.g. name/password), this string will accompany the prompt.

allowAnonymous - true allows anonymous credentials.

**Throws:** SecurityException

Thrown if the operator fails to authenticate or if the constructor fails for any reason.

### SecurityContext

```
 public SecurityContext(String username,
                        String passsword,
                        boolean allowAnonymous) throws SecurityException
```

Create a new `SecurityContext` using the operator authentication passed in as arguments. The application passes in the password that is used to authenticate the operator. Prompt the operator for authentication (e.g. name and password). The number of tries given to the operator is established by the binding and the local security administrator. Applications that catch the `SecurityException` thrown by this method should not call the method again to give the operator another chance to authenticate. This constructor can be used by servers or any machine that does not have an oprator present.

**Parameters:**

username - the name used to authenticate the operator.

password - the password used to authenticate the operator.

allowAnonymous - true allows anonymous credentials.

**Throws:** SecurityException

Thrown if authentication fails.

Methods:

### getAvailableCredentials

```
 public Credential[] getAvailableCredentials()
```

Get the credentials that are available to the process. The credentials are returned in an array. If no credentials are available, `null` is returned.

**Returns:**

an array of credentials.

**See Also:**

Credential

### setEncryption

public void setEncryption(boolean useEncryption)

Specify whether communication connections created via the SecurityContext will encrypt communication. Note that no exception is thrown. The application does not know whether/how the underlying security implementation supports encryption. And it does not need to know - the application will function correctly without this knowledge (in contrast to the Credential `setExpirationDate` and `setDelegatable` methods).

**Parameters:**

useEncryption - set to TRUE if encryption is desired.

### getEncryption

public boolean getEncryption()

Return the *real* encryption state. In other words, if the application specifies that encryption is to be used, but the security binding does not support encryption, `getEncryption` will return FALSE. Conversely, if this application specifies to not use encryption, but the system default is to encrypt all communication, `getEncryption` will return TRUE.

**Returns:**

TRUE if communication is to be encrypted, FALSE if not.

### getBindingName

public String getBindingName()

Return a string that describes the security implementation.

**Returns:**

the name of this binding between the security APIs and security implementation.

### setDefaultCredential

public void setDefaultCredential(Credential defaultCredential) throws SecurityException

Set the Credential that will be used to establish future connections when a Credential is not specified for that specific connection. Note that every instance of the SecurityContext has a binding-specific default Credential prior to a call to this method (assuming that *any* Credential is available). An exception is thrown if the Credential cannot be used as the default. For example, if the Credential is not one of the Credentials listed in a call to `getAvailableCredentials`.

**Parameters:**

defaultCredential - The Credential to use as a default. Specify `null` if an anonymous connection is to be the default.

**Throws:** SecurityException

Thrown if the Credential cannot be used as the default credential.

**See Also:**

Credential, getDefaultCredential

### getDefaultCredential

public Credential getDefaultCredential()

Get the Credential that is the default when establishing connections. `null` is returned if there are no credentials available.

**Returns:**

The Credential to use as a default.

**See Also:**

Credential, setDefaultCredential

### establishConnection

```
public SecureConnection establishConnection(String host,
                                            int port) throws SecurityException, IOException
```

Establish a secure connection to another process. Takes a hostname and port number as arguments. Opens a secure connection to the specified port, using the current `SecurityContext` state and default `Credential` as the connection parameters. Note that the other process participating in the connection must use `acceptConnection` for the connection to be successfully established.

**Parameters:**

host - The DNS name or IP address of the machine to establish the connection to.

port - The socket port to establish the connection to.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

acceptConnection

### establishConnection

public SecureConnection establishConnection(String host, int port,

Credential credential) throws SecurityException, IOException

Establish a secure connection to another process. Takes a hostname, port number and a Credential as arguments. Opens a secure connection to the specified port, using the current SecurityContext state as the connection parameters. Note that the other process participating in the connection must use `acceptConnection` for the connection to be successfully established.

**Parameters:**

host - The DNS name or IP address of the machine to establish the connection to.

port - The socket port to establish the connection to.

credential - The credential to use for this connection.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

acceptConnection

### listen

public void listen(int port) throws SecurityException, IOException

Listen for connection requests from clients. Takes port number as argument. Note that the other process participating in the connection must use `establishConnection`. Also, the server must call `acceptConnection` after listen

**Parameters:**

port - The socket port to monitor for connections.

**Returns:**

void

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

acceptConnection

### acceptConnection

public SecureConnection acceptConnection() throws SecurityException, IOException

Accept a secure connection from another process. Returns a `SecureConnection` to another process that communicates with the port being listened on using the same Security Services binding. The current `SecurityContext` state and default `Credential` are used as the connection parameters. Note that the other process participating in the connection must use `establishConnection` for the connection to be successfully established.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

establishConnection

### acceptConnection

public SecureConnection acceptConnection(Credential credential) throws SecurityException, IOException

Accept a secure connection to another process. Takes a Credential as an argument. Returns a `SecureConnection` to another process that communicates with the port being listened on using the same Security Services binding. The current SecurityContext state is used as the connection parameters. Note that the other process participating in the connection must use `establishConnection` for the connection to be successfully established.

**Parameters:**

credential - The credential to use for this connection.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

establishConnection

## Class mil.disa.dii.kernel.securityServices.SecureConnection

```
java.lang.Object
  |
  +----mil.disa.dii.kernel.securityServices.SecureConnection
```

public class **SecureConnection**

extends Object. The SecureConnection is the interface to the state associated with a specific connection. This object is created as a result of establishing a connection via the SecurityContext. It provides methods to query the connection state (e.g. the identity of the principle at the other end of the connection, the credential used to establish the connection, the level of service provided by the connection, etc.), and methods to write data to and read data from the connection. No methods are provided to change the state of the connection.

**See Also:**

SecurityContext

---

Method Index:

**close**()

Close the connection.

**getBindingName**()

Return a string that describes the security implementation associated with this connection.

**getEncryption**()

Indicate whether encryption is being used by this connection.

**getMyCredential**()

Return the `Credential` that was used to create this connection.

**getOtherPrincipalName**()

Return the name of the principal at the other end of the connection.

**read**()

Read a "block" of data from the connection and return it as a byte array.

**write**(byte[])

Write a "block" of data (an array of bytes) to the connection.

Methods:

### getMyCredential

public Credential getMyCredential()

Return the `Credential` that was used to create this connection.

**Returns:**

the `Credential` used to create this connection. If the connection was made with no Credential, return `null`.

### getOtherPrincipalName

public String getOtherPrincipalName()

Return the name of the principal at the other end of the connection. The value of this name should be equal to what is returned by a call to the `getNameOfPrincipal` method of the credential object used by the other principal to establish this connection.

**Returns:**

the name of the principal at the other end of the connection.

**See Also:**

getNameOfPrincipal

### getEncryption

public boolean getEncryption()

Indicate whether encryption is being used by this connection.

**Returns:**

`true` if communication is to be encrypted, `false` if not.

### getBindingName

public String getBindingName()

Return a string that describes the security implementation associated with this connection.

**Returns:**

the name of this binding between the security APIs and security implementation.

### read

public byte[] read() throws SecurityException, IOException

Read a "block" of data from the connection and return it as a byte array. Note that this is a blocking read.

**Returns:**

An array of bytes that is the data that was transmitted by the other principal.

**Throws:** SecurityException

thrown if encryption is on and the data cannot be decrypted, validation is on and the data cannot be validated, or any other binding-specific security failure occurs.

**Throws:** IOException

thrown if an IO failure occurs.

**See Also:**

write

### write

public void write(byte data[]) throws SecurityException, IOException

Write a "block" of data (an array of bytes) to the connection. The existence of a reliable transport protocol is assumed. Thus, if the transport protocol indicates a successful transmission, then so will `write`. If the corresponding `read` fails, it is up to the application protocol to NACK.

**Parameters:**

An - array of bytes that is the data to be transmitted.

**Throws:** SecurityException

thrown if encryption is on and the data cannot be decrypted, validation is on and the data cannot be validated, or any other binding-specific security failure occurs.

**Throws:** IOException

thrown if an IO failure occurs.

**See Also:**

read

### close

public void close() throws SecurityException, IOException

Close the connection. All pending outgoing data is flushed. All pending incoming data is dropped. Any `read` or `write` invocations subsequent to the `close` will result in `IOException` being thrown.

**Throws:** SecurityException

thrown if an error occurs.

**Throws:** IOException

thrown if the connection cannot be closed (perhaps because it was already closed).

## Class mil.disa.dii.kernel.securityServices.SecurityException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----mil.disa.dii.kernel.securityServices.SecurityException
```

public class **SecurityException**

extends Exception. This is the exception class thrown by the classes in the mil.disa.dii.kernel.securityServices package when a security violation has occurred (e.g. failure to authenticate).

Constructor Index:

**SecurityException**()

**SecurityException**(String)

Constructors:

**SecurityException**

public SecurityException()

**SecurityException**

public SecurityException(String msg)

## Class mil.disa.dii.kernel.securityServices.IllegalValueException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----mil.disa.dii.kernel.securityServices.IllegalValueException
```

public class **IllegalValueException**

extends Exception. This is the exception class used when an illegal value is passed to a parameterized service in the mil.disa.dii.kernel.securityServices package.

Constructor Index:

**IllegalValueException**()

**IllegalValueException**(String)

Constructors:

**IllegalValueException**

public IllegalValueException()

**IllegalValueException**

public IllegalValueException(String msg)

**Appendix B**

# Java Programming Language Mapping

## Class mil.disa.dii.kernel.securityServices.SecurityFactory

```
java.lang.Object
   |
   +----mil.disa.dii.kernel.securityServices.SecurityFactory
```

---

public class **SecurityFactory**

extends Object. Instantiates objects that implement the securityService interfaces.

---

Constructor Index:

**SecurityFactory**()

Method Index

**getBindingName**()

Return a string that describes the security implementation.

**makeSecurityContext**(String, boolean)

Create a new `SecurityContext`.

**makeSecurityContext**(String, boolean, String)

Create a new `SecurityContext`, prompting for operator authentication if necessary.

**makeSecurityContext**(String, boolean, String, String)

Create a new `SecurityContext` using the operator authentication passed in as arguments.

Constructors:

**SecurityFactory**

public SecurityFactory()

Methods:

**getBindingName**

public static String getBindingName()

Return a string that describes the security implementation.

**Returns:**

the name of this binding between the security APIs and security implementation.

**makeSecurityContext**

public static SecurityContext makeSecurityContext(String path,
boolean allowAnonymous) throws SecurityException

Create a new `SecurityContext`. If authentication cannot be obtained from the environment, throw a `SecurityException`. (This constructor will not prompt the operator for authentication.)

**Parameters:**

allowAnonymous - If `false` and authentication cannot be obtained, throw a SecurityException. If `true` and authentication cannot be obtained, return a SecurityContext whose only credential is ANONYMOUS.

**Throws:** SecurityException

Thrown if the operator's authentication cannot be obtained from the environment or if the constructor fails for any reason.

**makeSecurityContext**

public static SecurityContext makeSecurityContext(String path,
boolean allowAnonymous,
String prompt) throws SecurityException

Create a new `SecurityContext`, prompting for operator authentication if necessary. If authentication cannot be obtained from the environment, prompt the operator for authentication (e.g. name and/or password). The number of tries given to the operator is established by the binding and the local security administrator. Applications that catch the `SecurityException` thrown by this method should not call the method again to give the operator another chance to authenticate.

**Parameters:**

allowAnonymous - If `false` and authentication cannot be obtained, throw a SecurityException. If `true` and authentication cannot be obtained, return a SecurityContext whose only credential is ANONYMOUS.

prompt - If the binding prompts the operator to enter authentication information (e.g. name/password), this string will accompany the prompt.

**Throws:** SecurityException

Thrown if the operator fails to authenticate or if the constructor fails for any reason.

### makeSecurityContext

```
 public static SecurityContext makeSecurityContext(String path,
                                      boolean allowAnonymous,
                                      String username,
                                      String password) throws SecurityException
```

Create a new `SecurityContext` using the operator authentication passed in as arguments. The application passes in the password that is used to authenticate the operator. prompt the operator for authentication (e.g. name and password). The number of tries given to the operator is established by the binding and the local security administrator. Applications that catch the `SecurityException` thrown by this method should not call the method again to give the operator another chance to authenticate.

**Parameters:**

allowAnonymous - If `false` and authentication cannot be obtained, throw a SecurityException. If `true` and authentication cannot be obtained, return a SecurityContext whose only credential is ANONYMOUS.

username - the name used to authenticate the operator.

password - the password used to authenticate the operator.

**Throws:** SecurityException

Thrown if authentication fails.

---

## Interface mil.disa.dii.kernel.securityServices.SecurityContext

public interface **SecurityContext.** The SecurityContext is the interface to security services. It gives the application access to attributes of the security services (allowing applications to specify and/or learn the level of service) and to credentials available to the application. It is also the source of `SecureConnection` objects.

Note that some of the methods specified in this class throw the `MethodNotImplementedException` exception. These are "optional" APIs for the interface. Any application developers writing code that calls these methods must understand the implications to their code if a method throws that exception.

**See Also:**

Credential, SecureConnection

Method Index:

**acceptConnection**()

Accept a secure connection from another process.

**acceptConnection**(Credential)

Accept a secure connection to another process.

**establishConnection**(String, int)

Establish a secure connection to another process.

**establishConnection**(String, int, Credential)

Establish a secure connection to another process.

**getAvailableCredentials**()

Get the credentials that are available to the process.

**getDefaultCredential**()

Get the Credential that is the default when establishing connections.

**getEncryption**()

Return the *real* encryption state.

**getSequencing**()

Return the *real* sequencing state.

**getValidation**()

Return the *real* validation state.

**listen**(int)

Directs the context to listen to the specified port.

**setDefaultCredential**(Credential)

Set the Credential that will be used to establish future connections when a Credential is not specified for that specific connection.

**setEncryption**()

Specify whether communication connections created via the SecurityContext will encrypt communication.

**setSequencing**()

Specify whether communication connections created via the SecurityContext will use sequencing to detect missing or duplicated messages.

**setValidation**()

Specify whether communication connections created via the SecurityContext will integrity seal communication.

Methods:

**getAvailableCredentials**

public abstract Credential[] getAvailableCredentials()

Get the credentials that are available to the process. The credentials are returned in an array. If no credentials are available, `null` is returned.

**Returns:**

an array of credentials.

**See Also:**

Credential

**setEncryption**

public abstract void setEncryption()

Specify whether communication connections created via the SecurityContext will encrypt communication. Note that no exception is thrown. The application does not know whether/how the underlying security implementation supports encryption. And it does not need to know - the application will function correctly without this knowledge (in contrast to the Credential `setExpirationDate` and `setDelegatable` methods).

**Parameters:**

useEncryption - set to TRUE if encryption is desired.

### getEncryption

public abstract boolean getEncryption()

Return the *real* encryption state. In other words, if the application specifies that encryption is to be used, but the security binding does not support encryption, `getEncryption` will return FALSE. Conversely, if this application specifies to not use encryption, but the system default is to encrypt all communication, `getEncryption` will return TRUE.

**Returns:**

TRUE if communication is to be encrypted, FALSE if not.

### setValidation

public abstract void setValidation()

Specify whether communication connections created via the SecurityContext will integrity seal communication. Note that no exception is thrown. The application does not know whether/how the underlying security implementation supports validation. And it does not need to know - the application will function correctly without this knowledge (in contrast to the Credential `setExpirationDate` and `setDelegatable` methods).

**Parameters:**

useValidation - set to TRUE if validation is desired.

### getValidation

public abstract boolean getValidation()

Return the *real* validation state. In other words, if the application specifies that validation is to be used, but the security binding does not support validation, `getValidation` will return FALSE. Conversely, if this application specifies to not use validation, but the system default is to encrypt all communication, `getValidation` will return TRUE.

**Returns:**

TRUE if communicated data is to be validated, FALSE if not.

### setSequencing

public abstract void setSequencing()

Specify whether communication connections created via the SecurityContext will use sequencing to detect missing or duplicated messages. Note that no exception is thrown. The application does not know whether/how the underlying security implementation supports sequencing. And it does not need to know - the application will function correctly without this knowledge (in contrast to the Credential `setExpirationDate` and `setDelegatable` methods).

**Parameters:**

useSequencing - set to TRUE if sequencing is desired.

### getSequencing

public abstract boolean getSequencing()

Return the *real* sequencing state. In other words, if the application specifies that sequencing is to be used, but the security binding does not support sequencing, `getSequencing` will return FALSE. Conversely, if this application specifies to not use sequencing, but the system default is to encrypt all communication, `getSequencing` will return TRUE.

**Returns:**

TRUE if communicated data is to be sequenced, FALSE if not.

### setDefaultCredential

public abstract void setDefaultCredential(Credential defaultCredential) throws SecurityException

Set the Credential that will be used to establish future connections when a Credential is not specified for that specific connection. Note that every instance of the SecurityContext has a binding-specific default Credential prior to a call to this method (assuming that *any* Credential is available). An exception is thrown if the Credential cannot be used as the default. For example, if the Credential is not one of the Credentials listed in a call to `getAvailableCredentials`.

**Parameters:**

defaultCredential - The Credential to use as a default. Specify `null` if an anonymous connection is to be the default.

**Throws:** SecurityException

Thrown if the Credential cannot be used as the default credential.

**See Also:**

Credential, getDefaultCredential

### getDefaultCredential

public abstract Credential getDefaultCredential()

Get the Credential that is the default when establishing connections. `null` is returned if there are no credentials available.

**Returns:**

The Credential to use as a default.

**See Also:**

Credential, setDefaultCredential

### establishConnection

public abstract SecureConnection establishConnection(String host,
int port) throws SecurityException, IOException

Establish a secure connection to another process. Takes a hostname and port number as arguments. Opens a secure connection to the specified port, using the current `SecurityContext` state and default `Credential` as the connection parameters. Note that the other process participating in the connection must use `acceptConnection` for the connection to be successfully established.

**Parameters:**

host - The DNS name or IP address of the machine to establish the connection to.

port - The socket port to establish the connection to.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

acceptConnection

### establishConnection

public abstract SecureConnection establishConnection(String host,
int port,
Credential credential) throws SecurityException,
IOException

Establish a secure connection to another process. Takes a hostname, port number and a Credential as arguments. Opens a secure connection to the specified port, using the current SecurityContext state as the connection parameters. Note that the other process participating in the connection must use `acceptConnection` for the connection to be successfully established.

**Parameters:**

host - The DNS name or IP address of the machine to establish the connection to.

port - The socket port to establish the connection to.

credential - The credential to use for this connection.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

acceptConnection

### acceptConnection

public abstract SecureConnection acceptConnection() throws SecurityException, IOException

Accept a secure connection from another process. Takes port number as argument. Listens for connections to the port, and returns a `SecureConnection` to another process that communicates with that port using the same Security Services binding. The current `SecurityContext` state and default `Credential` are used as the connection parameters. Note that the other process participating in the connection must use `establishConnection` for the connection to be successfully established.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if the other principal cannot be communicated with or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

establishConnection

### acceptConnection

public abstract SecureConnection acceptConnection(Credential credential) throws SecurityException, IOException

Accept a secure connection to another process. Takes port number and a Credential as arguments. Listens for connections to the port, and returns a `SecureConnection` to another process that communicates with that port using the same Security Services binding. The current SecurityContext state is used as the connection parameters. Note that `listen` must be invoked before `acceptConnection`. Note that the other process participating in the connection must use `establishConnection` for the connection to be successfully established.

**Parameters:**

credential - The credential to use for this connection.

**Returns:**

the `SecureConnection` interface that represents this connection.

**Throws:** SecurityException

thrown if the other principal fails to authenticate in accordance with the binding, system and instance parameters or if the other principal rejects this process's authentication.

**Throws:** IOException

thrown if `listen` has not been called, the other principal cannot be communicated with, or if the other principal does not speak the protocol specific to the security binding.

**See Also:**

establishConnection, listen

### listen

public abstract void listen(int port) throws SecurityException, IOException

Directs the context to listen to the specified port. This creates within the SecurityContext an SSLServerSocket that listens to the specified port. This method must be invoked before any calls to `acceptConnection`.

**Parameters:**

port - The socket port to monitor for connections.

**Throws:** SecurityException

thrown if the principal fails to initialize the SSL server socket with appropriate security settings

**Throws:** IOException

thrown if there is a system error associated with creating the server socket. e. g.) unavailable system resources/port

---

## Interface mil.disa.dii.kernel.securityServices.Credential

public interface **Credential.** Credential is the interface to the operator's identity. Credential objects have the ability to describe themselves, and a limited mechanism to allow the application to control how it can be used for authentication.

**See Also:**

getAvailableCredentials

---

Method Index:

**getCredentialIdentifier**()

Return a string that uniquely identifies this amongst all of the Credentials available to the application.

**getExpirationDate**()

Get the date/time at which the Credential did or will expire (cease to be a valid Credential).

**getNameOfPrincipal**()

Return a string that uniquely identifies the operator.

**getValidationDate**()

Get the date/time at which the Credential became or will become valid.

**isAnonymous**()

Methods:

**isAnonymous**

public abstract boolean isAnonymous()

**getCredentialIdentifier**

public abstract String getCredentialIdentifier()

Return a string that uniquely identifies this amongst all of the Credentials available to the application.

**Returns:**

the "name" of the Credential.

**getNameOfPrincipal**

public abstract String getNameOfPrincipal()

Return a string that uniquely identifies the operator. This is the string that is made available to the process at the other end of a connection established using this Credential.

**Returns:**

the "name" of the operator.

**See Also:**

getOtherPrincipalName

**getExpirationDate**

public abstract Date getExpirationDate()

Get the date/time at which the Credential did or will expire (cease to be a valid Credential).

**Returns:**

the date/time that the Credential did/will expire.

**See Also:**

Date, getValidationDate

**getValidationDate**

public abstract Date getValidationDate()

Get the date/time at which the Credential became or will become valid.

**Returns:**

the date/time that the Credential did/will become valid.

**See Also:**

Date, getExpirationDate

## Interface mil.disa.dii.kernel.securityServices.SecureConnection

public interface **SecureConnection.** The SecureConnection is the interface to the state associated with a specific connection. This object is created as a result of establishing a connection via the SecurityContext. It provides methods to query the connection state (e.g. the identity of the principle at the other end of the connection, the credential used to establish the connection, the level of service provided by the connection, etc.), and methods to write data to and read data from the connection. No methods are provided to change the state of the connection.

**See Also:**

SecurityContext

Method Index:

**close**()

Close the connection.

**getEncryption**()

Indicate whether encryption is being used by this connection.

**getInputStream**()

Return an input stream to read from the connection.

**getMyCredential**()

Return the `Credential` that was used to create this connection.

**getOtherPrincipalName**()

Return the name of the principal at the other end of the connection.

**getOutputStream**()

Return an output stream to write to the connection.

**getSequencing**()

Indicate whether sequencing is being used by this connection.

**getValidation**()

Indicate whether data signing/validation is being used by this connection.

Methods:

### getMyCredential

public abstract Credential getMyCredential()

Return the `Credential` that was used to create this connection.

**Returns:**

the `Credential` used to create this connection. If the connection was made with no Credential, return `null`.

### getOtherPrincipalName

public abstract String getOtherPrincipalName()

Return the name of the principal at the other end of the connection. The value of this name should be equal to what is returned by a call to the `getNameOfPrincipal` method of the credential object used by the other principal to establish this connection.

**Returns:**

the name of the principal at the other end of the connection.

**See Also:**

getNameOfPrincipal

### getEncryption

public abstract boolean getEncryption()

Indicate whether encryption is being used by this connection.

**Returns:**

`true` if communication is to be encrypted, `false` if not.

### getValidation

public abstract boolean getValidation()

Indicate whether data signing/validation is being used by this connection.

**Returns:**

`true` if communicated data is to be validated, `false` if not.

### getSequencing

public abstract boolean getSequencing()

Indicate whether sequencing is being used by this connection. Sequencing allows the detection of duplicated or missing messages.

**Returns:**

`true` if communicated data is being sequenced, `false` if not.

### getInputStream

public abstract InputStream getInputStream() throws IOException

Return an input stream to read from the connection.

**Returns:**

The input stream that reads from the connection.

**Throws:** IOException

Thrown if an IO error occurs.

### getOutputStream

public abstract OutputStream getOutputStream() throws IOException

Return an output stream to write to the connection.

**Returns:**

The output stream that writes to the connection.

**Throws:** IOException

Thrown if an IO error occurs.

### close

public abstract void close() throws IOException

Close the connection. All pending outgoing data is flushed. All pending incoming data is dropped. All resources held by the connection are released.

**Throws:** IOException

thrown if the connection cannot be closed (perhaps because it was already closed).

**Appendix C**

# Sample Java Programs

## Sample Java Client

```
import mil.disa.dii.kernel.securityServices.*;
import java.io.*;
public class coeclient{
      public static void main(String[] argv) {
            String message;
            try {
            SecurityContext s =
SecurityFactory.makeSecurityContext("triton_db",true,argv[0],"admin123");
            SecureConnection a = s.establishConnection("calypso",8888);

            OutputStream o = a.getOutputStream();
            System.out.print ("To "+ a.getOtherPrincipalName()+": ");
            BufferedReader kbi = new BufferedReader (new InputStreamReader
(System.in));
            o.write(kbi.readLine().getBytes());
            o.flush();

            byte[] b = new byte[100];
            InputStream i = a.getInputStream();
            int tmpi = i.read(b);
            System.out.println ("Received from server: "+ new String(b));

            a.close();
            } catch (Exception e) {System.out.println(e.toString());}
      }
}
```

## Sample Java Server

```
import mil.disa.dii.kernel.securityServices.*;
import java.io.*;
public class coeserver {

public static void main (String[] argv) {
      SecurityContext s=null;
      SecurityContext t=null;
      String incoming = "nothing";
      byte[] b = new byte[64];

      try {
```

```
      s = SecurityFactory.makeSecurityContext("calypso_db",true,"Enter
password" );
      Credential c=s.getDefaultCredential();
      System.out.println ("Current Security Context Using: " +
c.getCredentialIdentifier()+ c.getExpirationDate());
      s.listen(9999);
      } catch (Exception e) {System.out.println ("Error "+e.toString());}


      while (true) {
      try {
      SecureConnection a = s.acceptConnection();
      System.out.println("connected to: " + a.getOtherPrincipalName());
      InputStream i = a.getInputStream();
      int tmpi = i.read(b);
      incoming=new String (b);
      System.out.println ("Received: "+ incoming);

      OutputStream o = a.getOutputStream();
      o.write (incoming.getBytes());

      i.close();
      o.close();
      } catch (Exception e) {System.out.println ("Error
Communicating"+e.toString());};
      }
}
}
```

**Appendix D**

# C Language Mapping

```
/*-------------------------------------------------------
 * coesecserv.h
 *-------------------------------------------------------*/
#ifdef NT
#include    <time.h>
#else
#include    <sys/time.h>
#endif

#ifdef      __coesecserv_h

#else
#define     __coesecserv_h
typedef     int               COEBoolean;

typedef     void              *SecContextHandle;
typedef     void              *CredentialHandle;
typedef     void              *SecConnectionHandle;

/*-------------------------------------------------------
 * The secExcep structure is used to emulate exceptions
 * in java. On each call that could generate an exception,
 * a pointer to a secExcep structure is provided. If
 * the value of secErrno is anything other than
 * NoException, the operation generated an exception.
 *-------------------------------------------------------*/
struct      secExcep{
      int   secErrno;
      char  *secDescription;
};
typedef     struct secExcep         SecExcep;

#define     NoException             0
#define     IllegalValueException         1
#define     SecurityException       2

/*-------------------------------------------------------
 * Security Context
 *-------------------------------------------------------*/

enum  whichparam1 {CONTEXT_A1, CONTEXT_B1, CONTEXT_C1};
// enum is used to identify which parameter list is to be used:
// CONTEXT_A1: SecExcep*, COEBoolean allowAnonymous
// CONTEXT_B1: char* prompt, SecExcep* e, COEBoolean allowAnonymous
// CONTEXT_C1: char* name, char* passwd, SecExcep* e, COEBoolean
allowAnonymous

SecContextHandle  newSecurityContext(const whichparam1, ...);

CredentialHandle  *getAvailableCredentials(SecContextHandle cx,
      SecExcep *e);

void  setEncryption(COEBoolean useEncryption);
COEBoolean  getEncryption(SecContextHandle cx, SecExcep *e);

char  *getBindingName(SecContextHandle cx, SecExcep *e);

void  setDefaultCredential(SecContextHandle cx, COEBoolean flag,
      SecExcep *e);
CredentialHandle  getDefaultCredential(SecContextHandle cx,
      SecExcep *e);

/*-------------------------------------------------------
 * Client side establishConnection() methods
 *-------------------------------------------------------*/

enum  whichparam2 {CONTEXT_A2, CONTEXT_B2};
// enum is used to identify which parameter list is to be used:
// CONTEXT_A2: SecContextHandle cx, char *host, int port, SecExcep *e
// CONTEXT_B2: SecContextHandle cx, char *host, int port, CredentialHandle
cr,
//             SecExcep *e

SecConnectionHandle     establishConnection(const whichparam2, ...);

/*-------------------------------------------------------
 * Server side listenConnection() method
 *-------------------------------------------------------*/

void  listenConnection(SecContextHandle cx, int port, SecExcep *e);

/*-------------------------------------------------------
 * Server side acceptConnection() methods
 *-------------------------------------------------------*/

enum  whichparam3 {CONTEXT_A3, CONTEXT_B3};
// enum is used to identify which parameter list is to be used:
// CONTEXT_A3: SecContextHandle cx, SecExcep *e
// CONTEXT_B3: SecContextHandle cx, CredentialHandle cr, SecExcep *e

SecConnectionHandle     acceptConnection(const whichparam3, ...);

/*-------------------------------------------------------
 * Credential
 *-------------------------------------------------------*/

char  *getCredentailIdentifier(CredentialHandle cr, SecExcep *e);
char  *getNameOfPrincipal(CredentialHandle cr, SecExcep *e);

struct tm   getExpirationDate(CredentialHandle cr, SecExcep *e);

struct tm   getValidationDate(CredentialHandle cr, SecExcep *e);

CredentialHandle  copyCredential(CredentialHandle cr, SecExcep *e);


/*-------------------------------------------------------
 * Secure Connection
 *-------------------------------------------------------*/

CredentialHandle  getMyCredential(SecConnectionHandle ch,
      SecExcep *e);

CredentialHandle  getOtherPrincipalCredential(
      SecConnectionHandle ch,SecExcep *e);

char *getBindingName(SecConnectionHandle ch,
      SecExcep *e);

COEBoolean  getEncryption(SecConnectionHandle ch, SecExcep *e);

int   writeSecureConnection(SecConnectionHandle ch, char *buffer,
      int bufsiz, SecExcep *e);
int   readSecureConnection(SecConnectionHandle ch,
      char *buffer, int bufsiz, SecExcep *e);
void  closeSecureConnection(SecConnectionHandle ch, SecExcep *e);

#endif
```

**Appendix E**

# Sample C Programs

## Sample C Client

```
/*-------------------------------------------------------------------------
-----
Name: client.c
Description: Sample client application using the COE Security Services API
Author: Patrick O. Cesard
Date: 11/11/98
Version: 1.0
--------------------------------------------------------------------------
----*/
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "coesecserv.h"

#ifdef _WINDOWS
#include <conio.h>
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

int main(int argc, char *argv[]) {

      SecContextHandle  sxh                   = NULL;
      SecConnectionHandle sch                 = NULL;
      CredentialHandle  Credential        = NULL;
      CredentialHandle* Credential_List   = NULL;
      char*                   Identifier      = NULL;
      char*                   Name            = NULL;
      char*                   HostName        = NULL;
      char*                   CertDBPwd       = NULL;
      char*                   Nickname        = NULL;
      COEBoolean              Allow_Anonymous = TRUE;
      struct tm               ExpireDate      = {0,0,0,0,0,0,0,0,0};
      struct tm               ValidDate       = {0,0,0,0,0,0,0,0,0};
      int                           PortNum                = 9999;
      const                   DataSize        = 20;
      char*                   CertDir                = "c:\\client_db"; //
Certificate DB path
      int                           i                      = 0;
      char                    ProgName[10];
      char                    Buffer[20];
      char                    ch;
      SecExcep                e;

      // Read in the command line parameters
      if (argc == 3) {
            HostName = argv[1];                  // Host to connect to
            PortNum = atoi(argv[2]);       // Port number to use
            Nickname = NULL;               // No certificate specified so
anonymous user
      }
      else if (argc >= 4) {
            HostName = argv[1];                  // Host to connect to
            PortNum = atoi(argv[2]);       // Port number to use
            Nickname = argv[3];                  // Certificate nickname
      }
      else {
            strcpy(ProgName, argv[0]);
            printf("Usage: %s <host> <port> [<nickname>]\n\n", ProgName);
            exit(1);
      }

      // Initialize exception reporting
      e.secErrno = NoException;
      e.secDescription = NULL;

      // Create a security context
      sxh = newSecurityContext_C(CertDir, Allow_Anonymous, Nickname,
CertDBPwd, &e);

      // Check for errors
      if (e.secErrno != NoException) {
            printf("Error: %s\n", e.secDescription);
            secServShutdown();
            exit(1);
      }

      printf("\nCreated a new security context\n");

      // Retrieve all available credentials
      Credential_List = getAvailableCredentials(sxh, &e);
      printf("\nRetrieved all credentials...\n");

      // Go thru list and print useful info out
      for (i = 0; (Credential = Credential_List[i]) != NULL; i++) {
            // Get credential identifier
            Identifier = getCredentialIdentifier(Credential, &e);
            printf("\nCredential identifier is: %s\n", Identifier);

            // Get credential principal's name
            Name = getNameOfPrincipal(Credential, &e);
            printf("Credential principal name is: %s\n", Name);

            // Get credential expiration date
            ExpireDate = getExpirationDate(Credential, &e);
            printf("Credential expiration date is: %s",
asctime(&ExpireDate));

            // Get credential validation date
            ValidDate = getValidationDate(Credential, &e);
            printf("Credential validation date is: %s",
asctime(&ValidDate));

      }

      // Establish a connection with server
      sch = establishConnection_C(sxh, HostName, PortNum, &e);

      // Check for errors
      if (e.secErrno != NoException) {
            printf("Error: %s\n", e.secDescription);
            secServShutdown();
            exit(1);
      }

      printf("\nEstablished a connection!\n");
      printf("\nReady to communicate with server\n");
      printf("Press . to end communication\n");

      while (TRUE) {

            // Read in data to sent over
            i = 0;
            while ( ((ch = getchar()) != '\n') && (ch != EOF) )
                  Buffer[i++] = (char)ch;
            Buffer[i] = 0;

            // Send data to server
            writeSecureConnection(sch, Buffer, 20, &e);
            printf("Client sent to server: %s\n", Buffer);

            // Empty data buffer
            Buffer[0] = 0;

            // Read server response
```

```
        readSecureConnection(sch, Buffer, 20, &e);
        printf("Client read from server: %s\n", Buffer);

        // Check for end of communication
        if (Buffer[0] == '.')
            break;

        // Empty data buffer
        Buffer[0] = 0;
    }

    // Close the connection
    closeSecureConnection(sch, &e);
    printf("\nClosed the connection\n");

    // Shut down security services
    secServShutdown();
    printf("Shut down security services\n");

    return(0);
}
```

## Sample C Server

```
/*-------------------------------------------------------------------------
-----
Name: server.c
Description: Sample server application using the COE Security Services
API.
Author: Patrick O. Cesard
Date: 11/18/98
Version: 1.0
---------------------------------------------------------------------------
----*/
#include <stdio.h>
#include <string.h>
#include "coesecserv.h"

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

int main(int argc, char *argv[]) {
      SecContextHandle  sxh                      = NULL;
      SecConnectionHandle sch                    = NULL;
      CredentialHandle  crh                      = NULL;
      CredentialHandle* crhh               = NULL;
      ConnectionStatus  status                   = NULL;
      char*                   Identifier         = NULL;
      char*                   Name               = NULL;
      char*                   HostName           = NULL;
      char*                   certname           = NULL;
      char*                   certDBpwd          = NULL;
      int                           PortNum                  = 9999; //
Port number to use
      COEBoolean              allow_anonymous = TRUE;
      char*                   user_prompt        = "Please enter the DB
password:";
      char*                   certdir                  = "c:\\server_db";
// Certificate DB path
      const                   DataSize           = 20;
      char                    progName[10];
      char                    Buffer[20];
      SecExcep                e;

      // Read in command line arguments, if provided
      if (argc >= 3) {
            PortNum = atoi(argv[1]);       // Port number to use
            certname = argv[2];                  // Certificate nickname
      }
      else {
            strcpy(progName, argv[0]);
            printf("Usage: %s <port> <nickname>\n\n", progName);
            exit(1);
      }

      // Initialize exception reporting
      e.secErrno = NoException;
      e.secDescription = NULL;

      // Create a security context
      sxh = newSecurityContext_C(certdir, allow_anonymous, certname,
certDBpwd, &e);
      printf("\nA security context was created\n");

      // Check for errors
      if (e.secErrno != NoException) {
            printf("Error: %s\n", e.secDescription);
            secServShutdown();
            exit(1);
      }
      // Create a rendez-vous socket
      listenConnection_A(sxh, PortNum, &e);

      if (e.secErrno != NoException) {
            printf("Error: %s\n", e.secDescription);
            secServShutdown();
            exit(1);
      }


      while (TRUE) {

            printf("\nWaiting for a connection...\n\n");

            // Accept a connection
            sch = acceptConnection_A(sxh, &e);

            if (e.secErrno != NoException) {
                  printf("Error: %s\n", e.secDescription);
                  secServShutdown();
                  exit(1);
            }

            printf("\nA connection was made!\n\n");

            // Get the status on the secure connection
            status = getSecureConnStatus(sch, &e);

            if (status->cipher)
                  printf("Cipher used is %s, key size is %d, secret key
size is %d\n", status->cipher, status->key_size, status->secret_key_size);

            if (status->security_on)
                  printf("Security is ON\n");
            else
                  printf("Security is OFF\n");

            if (status->issuer && status->subject)
                  printf("Client cert issued by %s,\n for %s\n", status-
>issuer, status->subject);

            // Get the client credential name
            Name = getOtherPrincipalName(sch, &e);

            printf("Client name is: %s\n", Name);

            printf("\n");


            // Loop until a '.' is received
            while (TRUE) {

                  // Read data on the secure connection
                  readSecureConnection(sch, Buffer, 20, &e);

                  if (e.secErrno != NoException) {
                        printf("Error: %s\n", e.secDescription);
                        break;
                  }

                  printf("Server received from client: %s\n", Buffer);

                  // Write data on the secure connection
                  writeSecureConnection(sch, Buffer, 20, &e);

                  if (e.secErrno != NoException) {
                        printf("Error: %s\n", e.secDescription);
                        break;
                  }

                  printf("Server sent to client: %s\n", Buffer);

                  if (Buffer[0] == '.') {

                        // Close this secure connection
                        closeSecureConnection(sch, &e);

                        break;
                  }

                  // Empty the data buffer
                  Buffer[0] = 0;

            } // Current connection loop

      } // Accept new connection loop


      printf("Shut down security services\n");

      // Shut down security services
      secServShutdown();
}
```

**Appendix F**

# Relationship of COE SS API to Other Technologies

The Secure Connection element of the COE SS API was designed for use with distributed applications that use sockets as the communications mechanism. Hence, it satisfies the security needs of current DII COE applications. However, this architecture may cause people to ask the question, “Will the COE SS API support future applications built using distributed middleware?”

The answer to this question depends on the particular COE SS API security services one wants to use and the particular middleware. However, it is unlikely that the Secure Connection element will support such applications. One perspective on middleware is that it sits in the middle between the application and the socket communications mechanism. For example OMG’s CORBA, Microsoft Corporation’s Component Object Model (COM) and Remote Procedure Calls (RPCs) involve a significant layer of software between the application and the socket. As such, the Secure Connection element, which will operate directly on a socket, cannot be used to provide security.

Each of these techniques has its own approach to providing the equivalent security provided by the Secure Connection element. CORBA has the CORBA Security Specification, which defines “invocation security” for providing mutual authentication, integrity, and confidentiality protection for CORBA method invocations. COM uses a secure RPC mechanism to provide similar protections.

Should the COE SS API define a Secure Storage element for access to signed and/or sealed files, CORBA, COM, and RPC applications could take advantage of the service because these middleware technologies do not intervene between an application and its access to file storage. Hence, they could use the COE SS API Secure Storage element.

# Glossary

| | |
|---|---|
| **API**: | Application Programming Interface |
| **APM** | Accounts Profile Manager |
| **COE** | Common Operating Environment |
| **COM** | Component Object |
| **COMPASS** | Common Operational Modeling, Planning, and Simulation Strategy |
| **COP** | Common Operating Picture |
| **DII** | Defense Information Infrasture |
| **DISA** | Defense Information Systems Agency |
| **GSS-API** | Generic Security Services API |
| **I/O** | input/ouput |
| **IETF** | Internet Engineering Task Force |
| **JL ACTD** | Joint Logistics Advanced Concept Technology Demonstration |
| **JRAMS** | Joint Readiness Automated Management System |
| **NSS** | Netscape's Security Services |
| **RPC** | Remote Procedure Call |
| **SS** | Security Services |
| **SSAF** | Security Services Architecture Framework |
| **TGT** | ticket granting ticket |