or it will be the identifier associated with the receiver of the message (i.e. the request target for request or request fragment messages or the request client for reply or reply fragment messages). The value must equal Client if the value of **`target_context_id_valid`** in the CompleteEstablishContext was false, or the message has not yet been exchanged. It must equal Peer if the value of **`target_context_id_valid`** in the CompleteEstablishContext was true. The use of peer identifiers allows the recipient of the message to more efficiently find its security context. The values are defined as:

- Client. The context id is that of the association's client.
- Peer. The context id is that of the recipient of the message.
- Sender. The context id is that of the sender of the message. This is only used with the DiscardContext message when the sender of the DiscardContext message has no context and has received a message that it cannot process.

Message Definitions

MTEstablishContext

This message is passed by the client to the target when a new association is to be established. Its definition is:

```
struct EstablishContext {
    ContextId          client_context_id;
    sequence <octet>   initial_context_token;
};
```

- **`client_context_id`**. This is the client's identifier for the security association. It is passed by the target to the client with subsequent messages within the association. It enables the client to link the message with the appropriate security context.
- **`initial_context_token`**. This is the token required by the target to establish the security association. It contains a mechanism version number, mech type identifier, and mechanism-specific information required by the target to establish the context. It may be sent with a protected message (for example, if the client does not wish to authenticate the target).

MTCompleteEstablishContext

This message is returned by the target to indicate that the association has been established. It is sent as a reply to an establish context or continue establish context. It may be sent with a GIOP reply or reply fragment. Its definition is:

```
struct CompleteEstablishContext {
    ContextId          client_context_id;
    boolean            target_context_id_valid;
    ContextId          target_context_id;
    sequence <octet>   final_context_token;
};
```

- **`client_context_id`**. This is the client's identifier for the security association. It is returned by the target to the client to enable the client to link the message with the appropriate security context.
- **`target_context_id_valid`**. This indicates whether the target has supplied a **`target_context_id`** for use by the client. **`TRUE`** indicates that the following field is valid.
- **`target_context_id`**. The targets identifier for the association. It is passed by the client to the target with subsequent messages. It enables the target to associate a local identifier with the context to allow the target to identify the context efficiently.
- **`final_context_token`**. This is the token required by the client to complete the establishment of the security association. It may be zero length.

MTContinueEstablishContext

This message is used by the client or target during context establishment to pass further messages to its peer as part of establishing the context. It may be the response to an establish context or to another continue establish context and is defined as:

```
struct ContinueEstablishContext {
    ContextId          client_context_id;
    sequence <octet>   continuation_context_token;
};
```

- **`client_context_id`**. The client's identifier for the association. It is used by both client and target to identify the association during the establishment sequence.
- **`continuation_context_token`**. This is the security information required to continue establishment of the security association.

MTDiscardContext

This message is used to indicate to the receiver that the sender of the message has discarded the identified context. Once the message has been sent, the sender will not send further messages within the context. The message is used as a hint to enable contexts to be closed tidily. Its definition is:

```
struct DiscardContext {
    ContextIdDefn      message_context_id_defn;
    ContextId          message_context_id;
    sequence<octet>    discard_context_token;
};
```

- **`message_context_id_defn`**. The type of context identifier supplied in the **`message_context_id`** field.
- **`message_context_id`**. The context identifier to be used by the recipient of the message to identify the context to which the message applies.

- **discard_context_token**. A token to be used by the recipient of the message to identify which context needs to be discarded. Not all security mechanisms emit such tokens; in case no token is available, a zero-length octet string should be used.

MTMessageError

This message is used to indicate an error detected in attempting to establish an association either due to a message protocol error or a context creation error. The message is also used to indicate errors in use of the context.

```
struct MessageError {
    ContextIdDefn       message_context_id_defn;
    ContextId           message_context_id;
    long                major_status;
    long                minor_status;
};
```

- **message_context_id_defn**. The type of context identifier supplied in the **message_context_id** field.
- **message_context_id**. The context identifier to be used by the recipient of the message to identify the context to which the message applies. It is either the client's identifier for the context (type client) or the receiver of the messages identifier (type peer).
- **major_status**. The reason for rejecting the context. The values used are those defined by the GSS API (RFC 1508) for fatal error codes.
- **minor_status**. This field allows mechanism specific error status to further define the reason for rejecting the context. It is not defined further here.

MTMessageInContext

Once established messages are sent within the context using the MessageInContext message. Its definition is:

```
enum ContextTokenType {
    SecTokenTypeWrap,
    SecTokenTypeMIC
};

struct MessageInContext {
    ContextIdDefn       message_context_id_defn;
    ContextId           message_context_id;
    ContextTokenType    message_context_type;
    sequence <octet>    message_protection_token;
};
```

- **`message_context_id_defn`**. The type of context identifier supplied in the **`message_context_id`** field.
- **`message_context_id`**. The context identifier to be used by the recipient of the message to identify the context to which the message applies.
- **`message_context_type`**. An indicator on whether the protection token is a "Wrap" token (which includes the protected message text and is ordinarily used to provide confidentiality protection) or an "MIC" token (which does not include the protected message text and is used to provide only integrity protection).
- **`message_protection_token`**. The Wrap or MIC token for the message. This is a self-defining token which indicates how the message is protected. If the message is not protected, the token will be zero length.

For unprotected and integrity-protected messages, the token will be an MIC token, and the MessageInContext message will be followed by the higher level protocol message, which is being protected by the security context (i.e. GIOP message or message fragment). In this case, the length of the higher level protocol message is included in the **`message_size`** field of the MessageInContext message's SECIOP header.

For confidentiality-protected messages, the protected message text will be included in the **`message_protection_token`** (which will be a Wrap token) of the MessageInContext message, and no higher-level protocol messages will be transmitted within the security context described by the MessageInContext message. In this case, the value in the **`message_size`** field of the MessageInContext message's SECIOP header will represent the length of the MessageInContext message only.

SECIOP Protocol State Tables

Note that some mechanisms may start in state S3.

Table 15-11 Client State Table

	No context (SO)	**Context being created Message allowed (S1)**	**Context being created Message not allowed (S2)**	**Context created (S3)**
request context establish (client auth)	create context send establish context **S1**			
request context establish (target or mutual auth)	create context send establish context **S2**			
receive message error	send DiscardContext with the message sender's context_id **SO**	discard context **SO**	discard context **SO**	discard context **SO**

	No context (SO)	**Context being created Message allowed (S1)**	**Context being created Message not allowed (S2)**	**Context created (S3)**

receive continue establish context message	send DiscardContext with the message sender's context_id **S0**		update context state
			if ok
			  send continue establish context
			  S2
			else
			  send message error
			  S0	

receive complete establish context message	send DiscardContext with the message sender's context_id **S0**	complete context with target's context id
		if ok
		  S3
		else
		  delete context
		  send message error **S0**
			complete context with target's context id
			if ok
			  S3
			else
			  delete context
			  send message error **S0**	

request to send message in context		send message in context with client context id **S1**		send message in xontext with client or target context id **S3**

receive message in context	send DiscardContext with the message sender's context_id **S0**	process message
		if ok
		  S1
		else
		  if message decode error send message error **S1**
		else
		  send message error
		 S0
				process message
				if ok
				  S3
				else
				  if message decode error send message error **S3**
				else
				  send message error
				 S0

request to send discard context message		send discard context message delete context **S0**	send discard context message delete context **S0**	send discard context message delete context **S0**
receive discard context message		delete context **S0**	delete context **S0**	delete context **S0**

See Table 15-12 for the Target State information.

Table 15-12 Target State Table

	No context (SO)	**Context being created Message allowed (S1)**	**Context being created Message not allowed (S2)**	**Context created (S3)**
receive establish context message (client auth)	create context if ok send complete establish context **S3** else send message error **S0**			

receive extablish context message (target or mutual auth)	create context
	if ok
	if continuation
	send continue **S2**
	else
	send complete establish **S3**
	else
	send message error delete context **S0**			

receive message error	send DiscardContext with the message sender's context_id **SO**		delete context **SO**	delete context **SO**

receive continue establish context message	send DiscardContext with the message sender's context_id **S0**		update context
			if ok
			if continuation
			send continuation **S2**
			else
			send complete establish **S3**
			else
			send message error context **S0**	

request to send message in context				send message in xontext with peer context id **S3**

	No context (SO)	**Context being created Message allowed (S1)**	**Context being created Message not allowed (S2)**	**Context created (S3)**
receive message in context	send DiscardContext with the message sender's context_id **S0**	process message if ok **S1** else if message decode error send message error **S1** else send message error **S0**		process message if ok **S3** else if message decode error send message error **S3** else send message error **S0**
request to send discard context message			send discard context message delete context **S0**	send discard context message delete context **S0**
receive discard context message			delete context **S0**	delete context **S0**

15.8.5 DCE-CIOP with Security

This section describes how to provide secure interoperability between ORBs, which use the DCE Common Inter-ORB Protocol (DCE-CIOP). It describes how the DCE-CIOP transport layer should handle security (for example, how it should interpret the security components of the IOR profile when selecting DCE Security Services for a request and secure invocation).

Goals of Secure DCE-CIOP

The original goals of DCE-CIOP, documented in the CORBA 2.0 specification, are maintained and enhanced by Secure DCE-CIOP:

- Support multivendor, mission critical, enterprise-wide, secure ORB-based applications.
- Leverage services provided by DCE wherever appropriate.
- Allow efficient and straightforward implementation using public DCE APIs.
- Preserve ORB implementation freedom.

Secure DCE-CIOP achieves these goals by taking advantage of the integrated security services provided by DCE Authenticated RPC. It is not a goal of the Secure DCE-CIOP specification to support the use of arbitrary security mechanisms for protection of DCE-CIOP messages.

Secure DCE-CIOP Overview

Secure interoperability between ORBs using the DCE-CIOP transport relies on the DCE Security Services and the DCE Authenticated RPC run-time that utilizes those services.

The DCE Security Services (specified in the X/Open Preliminary Specification *X/Open DCE: Authentication and Security Services*), as employed by the DCE Authenticated RPC run-time (specified in the X/OPEN *CAE Specification C309* and the OSF *AES/Distributed Computing RPC Volume*), provide the following security features:

- Cryptographically secured mutual authentication of a client and target
- Ability to pass client identity and authorization credentials to the target as part of a request
- Protection against undetected, unauthorized modification of request data.
- Cryptographic privacy of data
- Protection against replay of requests and data

The RPC run-time provides the communication conduit for exchanging security credentials between communicating parties. It protects its communications from threats such as message replay, message modification, and eavesdropping.

The DCE-CIOP uses DCE RPC APIs to request security features for a given client-target communication binding. Subsequent DCE-CIOP messages on that binding flow over RPC and thus are protected at the requested levels.

This Secure DCE-CIOP specification defines the IOR Profile components required to support Secure DCE-CIOP. Each component is identified by a unique tag, and the encoding and semantics of the associated **`component_data`** are specified. Client secure association requirements, as indicated by client-side policy and target secure association requirements, as specified in the target IOR Profile security components are mapped to DCE Security Services. Finally, the use of DCE APIs to protect DCE-CIOP messages is described.

IOR Security Components for DCE-CIOP

The information necessary to invoke secure operations on objects using DCE-CIOP is encoded in an IOR in a profile identified by TAG_MULTIPLE_COMPONENTS. The **`profile_data`** for this profile is a CDR encapsulation (see “CDR Transfer Syntax” in Section 12.3 of the CORBA 2.0 specification) of the MultipleComponentProfile type, which is a sequence of TaggedComponent structures. These types are described in Chapter 3 of CORBA 2.0.

The Multiple Component Profile contains the tagged components required to support DCE-CIOP, described in Chapter 13 of the CORBA 2.0 specification, as well as the components required to support security for DCE-CIOP. The general security components are described in Security Components in the IOR under Section 15.8.4, CORBA Interoperable Object Reference with Security. The DCE-specific security component and semantics for the common security components are described here.

Although a conforming implementation of Secure DCE-CIOP is only required to generate and recognize the components defined here and in Chapter 13 of CORBA 2.0, the profile may also contain components used by other kinds of ORB transports and services. Implementations should be prepared to encounter profiles identified by TAG_MULTIPLE_COMPONENTS that do not support DCE-CIOP. Unrecognized components should be preserved but ignored. Although an implementation may choose to order the components in a profile in a particular way, other implementations are not required to preserve that order. Implementations must be prepared to handle profiles whose components appear in any order.

TAG_DCE_SEC_MECH

For a profile to support Secure DCE-CIOP, it must include exactly one TAG_DCE_SEC_MECH component. Presence of this component indicates support for the [non-GSSAPI] “DCE Security with Kerberos V5 with DES” mechanism type. The **`component_data`** field contains an authorization service identifier and an optional sequence of tagged components.

Future versions of DCE Security that require different information than what is provided by the **`component_data`** structure shown next are expected to be supported with a new component tag, rather than with revisions to the data structure associated with the TAG_DCE_SEC_MECH tag.

The DCE Security Mechanism component is defined by the following OMG IDL:

```
module DCE_CIOP {
    const IOP::ComponentId TAG_DCE_SEC_MECH = 103
    // CORBA IDL doesn't (yet) support const octet
    //
    // const octet DCEAuthorizationNone = 0;
    // const octet DCEAuthorizationName = 1;
    // const octet DCEAuthorizationDCE = 2;
    struct DCESecurityMechanismInfo {
        octet                     authorization_service;
        sequence<TaggedComponent> components;
    };
};
```

A TaggedComponent structure is built for the DCE Security mechanism component by setting the tag member to TAG_DCE_SEC_MECH, and setting the **`component_data`** member to a CDR encapsulation of a DCESecurityMechanismInfo structure.

authorization_service Field

The **`authorization_service`** field is used to indicate what authorization service is required by the target, and therefore must be supported by the authenticated RPC run-time for invocations on this IOR. Two authorization models are supported:

- DCEAuthorizationName and DCEAuthorizationDCE, with a third identifier.
- DCEAuthorizationNone, to indicate that no authorization is required.

See DCE RCP Authorization Services in Section 15.8.6, DCE-CIOP with Security, for details.

Components field

The components field contains a sequence of zero or more tagged components, none of which may appear more than once, from the following list of common security IOR components: TAG_ASSOCIATION_OPTIONS, and TAG_SEC_NAME.

Each of these components, defined in Security Components of the IOR in Section 15.8.4, CORBA Interoperable Object Reference with Security, may be present either in the components field of the DCESecurityMechanismInfo structure, or at the top level of the IOR profile. When one of these components appears at the top level of the profile, its data may be shared by other security mechanisms in the profile. When it appears in the nested components field of DCESecurityMechanismInfo, its data is available only to the DCE Security mechanism and overrides the data of an identically-tagged component, if present, at the top level of the profile.

TAG_ASSOCIATION_OPTIONS

The association options component, described in Security Components of the IOR in Section 15.8.4, CORBA Interoperable Object Reference with Security, contains flags indicating which protection and authentication services the target supports, and which it requires. This component is optional for Secure DCE-CIOP; defaults are used when the component is not present.

The way in which association options are interpreted for use with DCE security is reflected in Table 15-13, which shows how an association option is mapped to a DCE RPC protection level and authentication service.

Table 15-13 Association Option Mapping to DCE Security

Association Option	DCE RPC Protection Level	DCE RPC Authentication Service
NoProtection	rpc_c_protect_level_none	rpc_c_authn_none
Integrity	rpc_c_protect_level_pkt_integrity	rpc_c_authn_dce_secret
Confidentiality	rpc_c_protect_level_pkt_privacy	rpc_c_authn_dce_secret
DetectReplay	rpc_c_protect_level_pkt	rpc_c_authn_dce_secret
DetectMisordering	rpc_c_protect_level_pkt	rpc_c_authn_dce_secret
EstablishTrustInTarget	rpc_c_protect_level_connect	rpc_c_authn_dce_secret
EstablishTrustInClient	rpc_c_protect_level_connect	rpc_c_authn_dce_secret
tag not present	rpc_c_protect_level_default	rpc_c_authn_dce_secret

If the TAG_ASSOCIATION_OPTIONS component is not present, then the target is assumed both to support and to require**`rpc_c_protect_level_default`** and **`rpc_c_authn_dce_secret`**. (The value of **`rpc_c_protect_level_default`** is defined by the DCE implementation or by a site administrator.) See Behavior When TAG_ASSOCIATION_OPTIONS Not Present later in this section, for a description of how DCE security parameters are selected when this component is not present.

See DCE RPC Protection Levels and DCE RPC Authentication Services later in this section, for more details on the protection provided by the DCE authenticated RPC services.

target_supports field

When an association option is set in the **`target_supports`** field of the TAG_ASSOCIATION_OPTIONS **`component_data`**, it indicates that the target supports invocations which use Secure DCE-CIOP with the protection level and authentication service that correspond to the selected option, as shown in Table 15-13. Any or all of the association options may be set in the **`target_supports`** field. The options set in the **`target_supports`** field will be compared with client-side policy required options to determine if the target can support the client's requirements.

Although, for the DCE security mechanism, a single selected option may imply support for several other options (e.g., selection of the Integrity option implies support for DetectReplay, DetectMisordering, and EstablishTrustInClient), it is recommended that every supported option be explicitly set in the **`target_supports`** field to facilitate comparison with client requirements.

target_requires field

When an association option is set in the **`target_requires`** field of the TAG_ASSOCIATION_OPTIONS **`component_data`**, it indicates that the target requires invocations secured with at least the protection level and authentication service that correspond to the selected option, as shown in Table 15-13. Since DCE RPC supports a range of protection levels, each of which provides all the protection of the level below it and also some additional protection, selecting multiple **`target_requires`** options does not make sense. For DCE, no more than one option need be selected in the **`target_requires`** field.

If a TAG_ASSOCIATION_OPTIONS component is contained within the DCESecurityMechanismInfo structure, the **`target_requires`** field may conform to the DCE semantics (i.e. no more than one option selected). If other security mechanisms are sharing the TAG_ASSOCIATION_OPTIONS component, and perhaps using different rules for interpreting the target_requires field, then the **`target_requires`** field may have several options selected. The DCE Association Options Reduction algorithm, described later in this section, handles both cases and is used to select the appropriate DCE secure invocation services given a set of required association options.

The EstablishTrustInTarget option in the **`target_requires`** field is meaningless, and is therefore ignored.

TAG_SEC_NAME

The security name component contains the DCE principal name of the target. Generally, this is a global principal name that includes the name of the cell in which the target principal's account resides. If a cell-relative principal name (i.e., the cell prefix does not appear) is specified, the local cell is assumed. Cell-relative principal names are only appropriate for use in IORs that are consumed by clients in the same cell in which the target resides. When an IOR containing a cell-relative principal name in the TAG_SEC_NAME component crosses a cell boundary, the cell-relative principal name should be replaced with a global name.

The format of a "human-friendly" DCE principal name is described in Section 1.13 of the *X/Open DCE: Authentication and Security Services* specification [hereafter referred to as *X/Open DCE Security*]. It is a string containing a concatenated cell name and cell-relative principal name that looks like:

/.../cell-name/cell-relative-principal-name

For example, the principal with the cell-relative name "printserver" in the "mis.prettybank.com" cell has the global principal name:

/.../mis.prettybank.com/printserver

The **`component_data`** member of the TAG_SEC_NAME component is set to the string value of the DCE principal name. The string is represented directly in the sequence of octets, including the terminating NUL.

If the TAG_SEC_NAME component is not present, then a value of NUL is assumed, indicating that the client will depend on the DCE authenticated RPC run-time to retrieve the DCE principal name of the target, identified in the IOR by the DCE-CIOP string binding and binding name components. This case indicates that the client is not interested in authentication of the target identity.

DCE RPC Security Services

This section provides details about the protection provided by DCE Authenticated RPC authorization services, protection levels, and authentication services. See the **`rpc_binding_set_auth_info()`** man page in the *OSF DCE 1.1 Application Development Reference* for more information about using these protection parameters to secure an association between a client and target.

DCE RPC Authorization Services

This section describes the DCE authorization service indicated by the **`authorization_service`** member of the DCESecurityMechanismInfo structure in the **`component_data`** field of the TAG_DCE_SEC_MECH component.

DCEAuthorizationName indicates that the target performs authorization based on the client security name. The DCE RPC authorization service DCEAuthorizationName asserts the principal name (without cryptographic protection if the association option **NoProtection** is chosen, or with cryptographic protection otherwise).

DCEAuthorizationDCE indicates that the target performs authorization using the client's Privilege Attribute Certificate (for OSF DCE 1.0.3 or previous versions), or the client's Extended Privilege Attribute Certificate (for DCE 1.1). The authorization service DCEAuthorizationDCE asserts the principal name and appropriate authorization data (without cryptographic protection if the association option **NoProtection** is chosen, or with cryptographic protection otherwise).

DCEAuthorizationNone indicates that the target performs no authorization based on privilege information carried by the RPC run-time. This is valid only if the association option **NoProtection** is chosen.

The **`authorization_service`** identifiers defined here for Secure DCE-CIOP correspond to DCE RPC authorization service identifiers and are defined to have identical values.

Table 15-14 Relationship between Identifiers

Secure DCE-CIOP authorization_service	DCE RPC Authorization Service	Shared Value
DCEAuthorizationNone	rpc_c_authz_none	0
DCEAuthorizationName	rpc_c_authz_name	1
DCEAuthorizationDCE	rpc_c_authz_dce	2

DCE RPE Protection Levels

The meanings of the DCE RPC protection levels referenced in Table 15-14 are described next. For the purposes of evaluating the protection levels, it is interesting to remember that a single DCE-CIOP message is transferred over the wire in the body of one or more DCE RPC PDUs.

- **rpc_c_protect_level_none** indicates that no authentication or message protection is to be performed, regardless of the authentication service chosen. Depending on target policy, the client may be granted access as an unauthenticated principal.
- **rpc_c_protect_level_connect** indicates that the client and server identities are exchanged and cryptographically verified at the time the binding is set up between them. Strong mutual authentication and replay detection *for the binding setup only* is provided. There are no protection services per DCE RPC PDU.
- **rpc_c_protect_level_pkt** indicates that the **rpc_c_protect_level_connect** services are provided plus detection of misordering or replay of DCE RPC PDUs. There is no protection against PDU modification.
- **rpc_c_protect_level_pkt_integrity** offers the **rpc_c_protect_level_pkt** services plus detection of DCE RPC PDU modification.
- **rpc_c_protect_level_pkt_privacy** offers the **rpc_c_protect_level_pkt_integrity** services plus privacy of RPC arguments, which means the DCE-CIOP message in its entirety is privacy protected.
- **rpc_c_protect_level_default** indicates the default protection level, as defined by the DCE implementation or by a site administrator (should be one of the above defined values).

DCE RPC Authentication Services

The meanings of the DCE RPC authentication services referenced in Table 15-14 are described next.

- **rpc_c_authn_none** indicates no authentication. If this is selected, then no authorization, DCEAuthorizationNone, must be chosen as well.
- **rpc_c_authn_dce_secret** indicates the DCE shared-secret key authentication service.

Secure DCE-CIOP Operational Semantics

This section describes how the DCE-CIOP transport layer should provide security for invocation and locate requests.

During a request invocation, if the IOR components indicate support for the DCE-CIOP transport and the TAG_DCE_SEC_MECH component is present, then a Secure DCE-CIOP request can be made.

Deriving DCE Security Parameters from Association Options

The client-side secure invocation policy and the target-side policy expressed in the TAG_ASSOCIATION_OPTIONS component are used to derive the actual options using the method described in Determining Association Options in Section 15.8.4, CORBA Interoperable Object Reference with Security. These options are then reduced to a single **`required_option`** using the algorithm described in DCE Association Options Reduction Algorithm, next. The resultant **`required_option`** is used to select a DCE RPC protection level and authentication service using Table 15-13, Association Option Mapping to DCE Security. The derived protection level and authentication service are used to secure the association via the **`rpc_binding_set_auth_info()`** call (see Securing the Binding Handle to the Target, further in this section).

DCE Association Options Reduction Algorithm

The DCE Association Options Reduction algorithm is used to select a single association option, **`required_option`,** given the value required by client and target derived as described in Determining Association Options in Section 15.8.3, CORBA Interoperable Object Reference with Security. The resultant **`required_option`** indicates, via Table 15-13, the DCE protection level and authentication service to use for invocations.

The association option names used in the following algorithm refer to options in the negotiated-required options set.

The DCE Association Options Reduction algorithm is expressed as follows.

```
If Confidentiality is set, then required_option = Confidentiality;
else if Integrity is set, then required_option = Integrity;
else if DetectReplay is set, OR
     if DetectMisordering is set,
     then required_option = DetectReplay;
     (alternatively, the same results are obtained with:
     then required_option = DetectMisordering;)
else if EstablishTrustInClient is set,
     then required_option = EstablishTrustInClient;
else required_option = NoProtection.
```

Behavior When TAG_ASSOCIATION_OPTIONS Not Present

As described earlier, if the TAG_ASSOCIATION_OPTIONS component is not present, then the target is assumed to support and require **`rpc_c_protect_level_default`** and **`rpc_c_authn_dce_secret`**. Since these protection parameters are not expressed as association options, the usual method of deriving a single **required_option** by combining client and target policy (see Determining Association Options in Section 15.8.3, CORBA Interoperable Object Reference with Security, and DCE Associations Options Reduction Algorithm, above) cannot be used. As an alternative, use the following method to derive the required DCE RPC protection level and authentication service:

- Translate the client-side secure invocation policy from a set of client supported association options to a single **`client_supported_option`** and from a set of client required association options to a single **`client_required_option`**, using in each case the algorithm described in DCE Association Options Reduction Algorithm.
- Using Table 15-13, Association Option Mapping to DCE Security, translate the **`client_supported_option`** and **`client_required_option`** to corresponding "supported" and "required" DCE RPC protection level/authentication service pairs.
- If the target principal is a member of the local cell, determine the target required protection level implied by **`rpc_c_protect_level_default`** by calling **`rpc_mgmt_inq_dflt_protect_level()`** passing **`rpc_c_authn_dce_secret`** as the **`authn_svc`** parameter. If the target principal is not a member of the local cell or if it's difficult to determine, then assume a target required protection level of **`rpc_c_protect_level_pkt_integrity`**.
- If the client supports **`rpc_c_authn_dce_secret`**, then choose the strongest protection level that both the client and target support and that does not exceed the strongest protection level required by either the client or target. If the client does not support **`rpc_c_authn_dce_secret`**, then choose **`rpc_c_authn_none`** and **`rpc_c_protect_level_none`**. Use the protection level and authentication service thus derived to secure the association between this client and target.

Securing the Binding Handle to the Target

The DCE-CIOP protocol engine acquires an **`rpc_binding_handle`** to the target using its normal procedure. The DCE_CIOP sets authentication and authorization information on that binding handle with the **`rpc_binding_set_auth_info()`** call using data from the IOR profile security components in the following way:

- The target security name string from the TAG_SEC_NAME component (or NUL, if the component is not present) is passed to **`rpc_binding_set_auth_info()`** via the **`server_princ_name`** parameter.
- If the TAG_ASSOCIATION_OPTIONS component is present in the IOR, see Deriving DCE Security Parameters from Association Options above to select a DCE RPC protection level and authentication service for this invocation.

  If the TAG_ASSOCIATION_OPTIONS component is not present in the IOR, see Behavior When TAG_ASSOCIATION_OPTIONS Not Present above to select a DCE RPC protection level and authentication service for this invocation.

  The selected protection level is passed to **`rpc_binding_set_auth_info()`** via the **`protect_level`** parameter. The selected authentication service is passed via the **`authn_svc`** parameter to **`rpc_binding_set_auth_info()`**.
- The **`auth_identity`** parameter is set to NUL to use the DCE default login context.

- The authorization service identifier from the **authorization_service** field of the DCESecurityMechanismInfo **component_data** is mapped to the corresponding DCE RPC authorization service identifier, which is then passed via the **authz_svc** parameter.

After a successful call to **rpc_binding_set_auth_info**(), the authenticated binding handle will be used by the DCE-CIOP protocol engine to make secure requests.

Appendix A Consolidated OMG IDL

A.1 Introduction

The OMG IDL for CORBA security is split into modules as follows:

- A module containing the common data types used by all other security modules.
- A module for application interfaces for each Security Functionality Levels 1 and 2. (Note that security-ready ORBs provide no real security functionality. Since they provide only one operation, and that is proposed to be on the ORB, they are included in Appendix B, Summary of CORBA 2 Core Changes, not here.)
- A module for Security Level 2 security policy administration.
- A module for non-repudiation, including the non-repudiation policy administration interface. This is the optional non-repudiation service.
- A module for the replaceable implementation Security Service, as described in Section 15.7, Implementor's Security Interfaces.

In addition, a number of extensions to existing CORBA modules are proposed for:

- Finding details of services in general, and in particular the security implementation.
- ORB Service/interceptor interfaces.
- The Object and Current interfaces for handling security (and management) information.
- Extensions for domain and policy handling.
- Secure interoperability using GIOP and DCE-CIOP.
- Core management-related interfaces.

The IDL changes for these modules are defined in Appendix B, Summary of CORBA 2 Core Changes.

A minimal security Management module is also included in Appendix B.

A.2 General Security Data Module

This subsection defines the OMG IDL for security data types common to the other security modules, which is the module *Security*. This module must be available with any ORB that claims to be Security Ready. The *Security* module depends on the *Time* module.

```
module Security {

    typedef string security_name;
    typedef sequence <octet> Opaque;
```

```
// extensible families for standard data types

struct ExtensibleFamily {
        unsigned short              family_definer;
        unsigned short              family;
};

// security association mechanism type

typedef          string      MechanismType;
struct           SecurityMechandName {
        MechanismType        mech_type;
        SecurityName         security_name;
};

typedef sequence<MechanismType> MechanismTypeList;
typedef sequence<SecurityMechandName> SecurityMechandNameList;

// security attributes

typedef unsigned long                    SecurityAttributeType;

// identity attributes; family = 0

const       SecurityAttributeType                 AuditId = 1;
const       SecurityAttributeType                 AccountingId = 2;
const       SecurityAttributeType                 NonRepudiationId = 3;

// privilege attributes; family = 1

const       SecurityAttributeType                 Public = 1;
const       SecurityAttributeType                 AccessId = 2;
const       SecurityAttributeType                 PrimaryGroupId = 3;
const       SecurityAttributeType                 GroupId = 4;
const       SecurityAttributeType                 Role = 5;
const       SecurityAttributeType                 AttributeSet    = 6;
const       SecurityAttributeType                 Clearance = 7;
const       SecurityAttributeType                 Capability = 8;

struct      AttributeType {
        ExtensibleFamily                      attribute_family;
        SecurityAttributeType                 attribute_type;
};
typedef sequence<AttributeType> AttributeTypeList;

struct SecAttribute {
        AttributeType                 attribute_type;
        Opaque                        defining_authority;
        Opaque                        value;
        // the value of this attribute; can be
        // interpreted only with knowledge of type
};

typedef sequence<SecAttribute> AttributeList;
```

```
          // Authentication return status

          enum AuthenticationStatus {
                    SecAuthSuccess,
                    SecAuthFailure,
                    SecAuthContinue,
                    SecAuthExpired
          };

          // Association return status

          enum AssociationStatus {
                    SecAssocSuccess,
                    SecAssocFailure,
                    SecAssocContinue
          };

          //        Authentication method
          typedef             unsigned long             AuthenticationMethod;

          // Credential types which can be set as Current default

          enum CredentialType {
                    SecInvocationCredentials,
                    SecOwnCredentials,
                    SecNRCredentials
          };

          // Declarations related to Rights
          struct Right {
                    ExtensibleFamily                        rights_family;
                    string                                  right;
          };

          typedef sequence <Right> RightsList;

          enum RightsCombinator {
                    SecAllRights,
                    SecAnyRight
          };

          // Delegation related
          enum DelegationState {
                    SecInitiator,
                    SecDelegate
          };

// pick up from TimeBase
typedef TimeBase::UtcT                                  UtcT;
typedef TimeBase::IntervalT                                       IntervalT;
typedef TimeBase::TimeT                                 TimeT;
```

```
// Security features available on credentials.
    enum SecurityFeature {
        SecNoDelegation,
        SecSimpleDelegation,
        SecCompositeDelegation,
        SecNoProtection,
        SecIntegrity,
        SecConfidentiality,
        SecIntegrityAndConfidentiality,
        SecDetectReplay,
        SecDetectMisordering,
        SecEstablishTrustInTarget
    };

    // Security feature-value
    struct SecurityFeatureValue {
        SecurityFeature                     feature;
        boolean                             value;
    };

    typedef sequence<SecurityFeatureValue>
                                        SecurityFeatureValueList;

    // Quality of protection which can be specified
    // for an object reference and used to protect messages
    enum QOP {
        SecQOPNoProtection,
        SecQOPIntegrity,
        SecQOPConfidentiality,
        SecQOPIntegrityAndConfidentiality
    };

    // Association options which can be administered
    // on secure invocation policy and used to
    // initialize security context

    typedef unsigned short                  AssociationOptions;

    const       AssociationOptions NoProtection = 1;
    const       AssociationOptions Integrity= 2;
    const       AssociationOptions Confidentiality = 4;
    const       AssociationOptions DetectReplay= 8;
    const       AssociationOptions DetectMisordering = 16;
    const       AssociationOptions EstablishTrustInTarget = 32;
    const       AssociationOptions EstablishTrustInClient = 64;

    // Flag to indicate whether association options being
    // administered are the “required” or “supported” set

    enum RequiresSupports {
        SecRequires,
        SecSupports
    };
```

```
// Direction of communication for which
// secure invocation policy applies
enum CommunicationDirection {
        SecDirectionBoth,
        SecDirectionRequest,
        SecDirectionReply
};

// AssociationOptions-Direction pair
struct OptionsDirectionPair {
        AssociationOptions                options;
        CommunicationDirection            direction;
};

typedef sequence<OptionsDirectionPair>
                                     OptionsDirectionPairList;

// Delegation mode which can be administered
enum DelegationMode {
        SecDelModeNoDelegation,        // i.e. use own credentials
        SecDelModeSimpleDelegation,    // delegate received
                                          credentials
        SecDelModeCompositeDelegation// delegate both;
};

// Association options supported by a given mech type

struct MechandOptions {
        MechanismType                mechanism_type;
        AssociationOptions           options_supported;
};

typedef sequence<MechandOptions> MechandOptionsList;

// Audit

struct AuditEventType {
        ExtensibleFamily             event_family;
        unsigned short               event_type;
};

typedef sequence<AuditEventType> AuditEventTypeList;

typedef unsigned long                SelectorType;

const      SelectorType         InterfaceRef = 1;
const      SelectorType         ObjectRef = 2;
const      SelectorType         Operation = 3;
const      SelectorType         Initiator = 4;
const      SelectorType         SuccessFailure = 5;
const      SelectorType         Time = 6;
```

```
        // values defined for audit_needed and audit_write are:
        // InterfaceRef: object reference
        // ObjectRef: object reference
        // Operation: op_name
        // Initiator: Credentials
        // SuccessFailure: boolean
        // Time: utc time on audit_write; time picked up from
        // environment in audit_needed if required

        struct      SelectorValue {
                SelectorType            selector;
                any                     value;
        };

        typedef sequence<SelectorValue> SelectorValueList;
};
```

A.3 Application Interfaces - Security Functionality Level 1

This subsection defines those interfaces available to application objects using only Security Functionality Level 1, and consists of a single module, *SecurityLevel1*. This module depends on the *CORBA* module, and on the *Security* and *Time* module. The interface *Current* is implemented by the ORB. Its interface is defined by the following PIDL.

```
module SecurityLevel1 {
        interface Current : CORBA::Current {            // PIDL
                Security::AttributeList get_attributes (
                        in      Security::AttributeTypeList             attributes
                );
        };
};
```

A.4 Application Interfaces - Security Functionality Level 2

This subsection defines the addition interfaces available to application objects using Security Functionality Level 2. There is one module, *SecurityLevel2*. This module depends on *CORBA* and *Security*. The interfaces are described in Section 15.5, Application Developer's Interfaces.

```
module SecurityLevel2 {
        // Forward declaration of interfaces
        interface PrincipalAuthenticator;
        interface Credentials;
        interface Object;
        interface Current;
```

```
// Interface PrincipalAuthenticator
	interface PrincipalAuthenticator {
		Security::AuthenticationStatus authenticate (
			in Security::AuthenticationMethod		method,
			in string					security_name,
			in Security::Opaque				auth_data,
			in Security::AttributeList			privileges,
			out Credentials					creds,
			out Security::Opaque				continuation_data,
			out Security::Opaque				auth_specific_data
		);

		Security::AuthenticationStatus continue_authentication (
			in	Security::Opaque			response_data,
			inout Credentials				creds,
			out Security::Opaque			continuation_data,
			out Security::Opaque			auth_specific_data
		);
	};

	// Interface Credentials
	interface Credentials {

		Credentials copy ();

		void set_security_features (
			in	Security::CommunicationDirection	direction,
			in	Security::SecurityFeatureValueList	security_features
		);

		Security::SecurityFeatureValueList
			get_security_features (
			in	Security::CommunicationDirection
								direction
		);

		boolean set_privileges (
			in	boolean				force_commit,
			in	Security::AttributeList		requested_privileges,
			out	Security::AttributeList		actual_privileges
		);

		Security::AttributeList get_attributes (
			in Security::AttributeTypeList		attributes
		);
		boolean is_valid (
			out	Security::UtcT			expiry_time
		);

		boolean refresh();
	};

	typedef sequence <Credentials> CredentialsList;
```

```
// RequiredRights Interface

interface RequiredRights{
        void get_required_rights(
                in Object                                       obj,
                in CORBA::Identifier                            operation_name,
                in CORBA::RepositoryId                          interface_name,
                out Security::RightsList                        rights,
                out Security::RightsCombinator                  rights_combinator
        );

        void set_required_rights(
                in string                                       operation_name,
                in CORBA::RepositoryId                          interface_name,
                in Security::RightsList                         rights,
                in Security::RightsCombinator                   rights_combinator
        );
};

// Interface Object derived from Object
// providing additional operations on objref at this
// security level.

interface Object : CORBA::Object {    // PIDL

        void override_default_credentials (
                in Credentials                          creds
        );

        void override_default_QOP (
                in      Security::QOP                   qop
        );

        Security::SecurityFeatureValueList get_security_features (
                in      Security::CommunicationDirection        direction
        );

        Credentials get_active_credentials();

        Security::MechanismTypeList             get_security_mechanisms();

        void override_default_mechanism(
                in Security::MechanismType mechanism_type
        );

        Security::SecurityMechandNameList       get_security_names ();
};
```

```
        // Interface Current derived from SecurityLevel1::Current
        // providing additional operations on Current at this
        // security level. This is implemented by the ORB

        interface Current : SecurityLevel1::Current { //PIDL

            void set_credentials (
                in     Security::CredentialType          cred_type,
                in     Credentials                       creds
            );

            Credentials get_credentials (
                in     Security::CredentialType          cred_type
            );

            readonly attribute CredentialsList received_credentials;

            readonly attribute Security::SecurityFeatureValueList
                                                      received_security_features;

            CORBA::Policy get_policy (
                in     CORBA::PolicyType                 policy_type
            );

            readonly attribute RequiredRights            required_rights_object;
            readonly attribute PrincipalAuthenticator    principal_authenticator;
        };

        // interface audit channel
        interface AuditChannel {
            void audit_write (
                in     Security::AuditEventType              event_type,
                in     CredentialsList                       creds,
                in     Security::UtcT                        time,
                in     Security::SelectorValueList           descriptors,
                in     Security::Opaque                      event_specific_data
            );
        };
        // interface for Audit Decision

        interface AuditDecision {
            boolean          audit_needed (
                in Security::AuditEventType                      event_type,
                in Security::SelectorValueList                   value_list
            );

            readonly attribute AuditChannel audit_channel;
        };

    };
```

A.5 Security Administration Interfaces

This section covers interfaces concerned with querying and modifying security policies, and comprises the module *SecurityAdmin*. The *SecurityAdmin* module depends on *CORBA*, *Security*, and *SecurityLevel2*. The interfaces are described in Section 15.6, Administrator's Interfaces. There are related interfaces for finding domain managers and policies. Since they are not security specific, they are included in Appendix B, Summary of CORBA 2 Core Changes, not here.

```
module SecurityAdmin {

    // interface AccessPolicy
    interface AccessPolicy : CORBA::Policy {
        Security::RightsList get_effective_rights (
            in SecurityLevel2::CredentialsList          cred_list,
            in Security::ExtensibleFamily               rights_family
        );
    };

    // interface DomainAccessPolicy
    interface DomainAccessPolicy : AccessPolicy {
        void grant_rights(
            in Security::SecAttribute                   priv_attr,
            in Security::DelegationState                del_state,
            in Security::ExtensibleFamily               rights_family,
            in Security::RightsList                     rights
        );

    void revoke_rights(
            in Security::SecAttribute                   priv_attr,
            in Security::DelegationState                del_state,
            in Security::ExtensibleFamily               rights_family,
            in Security::RightsList                     rights
        );
    void replace_rights (
            in Security::SecAttribute                   priv_attr,
            in Security::DelegationState                del_state,
            in Security::ExtensibleFamily               rights_family,
            in Security::RightsList                     rights
        );
        Security::RightsList get_rights (
            in Security::SecAttribute                   priv_attr,
            in Security::DelegationState                del_state,
            in Security::ExtensibleFamily               rights_family
        );
    };
```

```
// interface AuditPolicy

interface AuditPolicy : CORBA::Policy {
	void set_audit_selectors (
		in	CORBA::InterfaceDef			object_type,
		in	Security::AuditEventTypeList		events,
		in	Security::SelectorValueList		selectors
	);

	void clear_audit_selectors (
		in	CORBA::InterfaceDef			object_type,
		in	Security::AuditEventTypeList		events
	);

	void replace_audit_selectors (
		in	CORBA::InterfaceDef			object_type,
		in	Security::AuditEventTypeList		events,
		in	Security::SelectorValueList		selectors
	);

	Security::SelectorValueList get_audit_selectors (
		in	CORBA::InterfaceDef			object_type,
		in	Security::AuditEventTypeList		events,
		in	Security::SelectorValueList		selectors
	);

	void set_audit_channel (
		in	SecurityLevel2::AuditChannel		audit_channel
	);
};

// interface SecureInvocationPolicy
interface SecureInvocationPolicy : CORBA::Policy {

	void set_association_options(
		in CORBA::InterfaceDef				object_type,
		in Security::RequiresSupports			requires_supports,
		in Security::CommunicationDirection		direction,
		in Security::AssociationOptions			options
	);

	Security::AssociationOptions get_association_options(
		in CORBA::InterfaceDef				object_type,
		in Security::RequiresSupports			requires_supports,
		in Security::CommunicationDirection		direction
	);
};

// interface DelegationPolicy
interface DelegationPolicy : CORBA::Policy {
	void set_delegation_mode(
		in CORBA::InterfaceDef				object_type,
		in Security::DelegationMode			mode
	);
```

```
        Security::DelegationMode get_delegation_mode(
            in CORBA::InterfaceDef                  object_type
        );
    };
};
```

A.6 Application Interfaces for Non-repudiation

This subsection defines the optional application interface for non-repudiation. This module depends on *Security* and *CORBA*. The interfaces are described in Section 15.5, Application Developer's Interfaces.

```
module NRservice  {
    typedef Security::MechanismType                 NRmech;
    typedef Security::ExtensibleFamily              NRPolicyId;

    enum EvidenceType {
        SecProofofCreation,
        SecProofofReceipt,
        SecProofofApproval,
        SecProofofRetrieval,
        SecProofofOrigin,
        SecProofofDelivery,
        SecNoEvidence     // used when request-only token desired
    };

    enum NRVerificationResult {
        SecNRInvalid,
        SecNRValid,
        SecNRConditionallyValid
    };

    // the following are used for evidence validity duration
    typedef unsigned long DurationInMinutes;

    const       DurationInMinutes           DurationHour   = 60;
    const       DurationInMinutes           DurationDay    = 1440;
    const       DurationInMinutes           DurationWeek   = 10080;
    const       DurationInMinutes           DurationMonth = 43200;// 30 days
    const       DurationInMinutes           DurationYear   = 525600;//365 days

    typedef long  TimeOffsetInMinutes;

    struct NRPolicyFeatures {
        NRPolicyId              policy_id;
        unsigned long           policy_version;
        NRmech                  mechanism;
    };

    typedef sequence<NRPolicyFeatures> NRPolicyFeaturesList;
    // features used when generating requests
    struct RequestFeatures {
        NRPolicyFeatures        requested_policy;
        EvidenceType            requested_evidence;
        string                  requested_evidence_generators;
        string                  requested_evidence_recipients;
        boolean                 include_this_token_in_evidence;
    };

    struct EvidenceDescriptor {
        EvidenceType            evidence_type;
        DurationInMinutes       evidence_validity_duration;
        boolean                 must_use_trusted_time;
};

    typedef sequence<EvidenceDescriptor> EvidenceDescriptorList;

    struct AuthorityDescriptor {
        string              authority_name;
        string              authority_role;
        TimeOffsetInMinutes                 last_revocation_check_offset;
                // may be >0 or <0; add this to evid. gen. time to
                // get latest time at which mech. will check to see
                // if this authority's key has been revoked.
};

    typedef sequence<AuthorityDescriptor> AuthorityDescriptorList;

    struct MechanismDescriptor {
        NRmech                                          mech_type;
        AuthorityDescriptorList                         authority_list;
        TimeOffsetInMinutes                             max_time_skew;
                // max permissible difference between evid. gen. time
                // and time of time service countersignature
                // ignored if trusted time not reqd.
};

    typedef sequence<MechanismDescriptor> MechanismDescriptorList;

    interface NRCredentials {

        boolean set_NR_features (
            in      NRPolicyFeaturesList                requested_features,
            out     NRPolicyFeaturesList                actual_features
        );

    NRPolicyFeaturesList get_NR_features ();

    void generate_token (
        in      Security::Opaque            input_buffer,
        in      EvidenceType                generate_evidence_type,
        in      boolean                     include_data_in_token,
        in      boolean                     generate_request,
        in      RequestFeatures             request_features,
          in     boolean                     input_buffer_complete,
          out    Security::Opaque            nr_token,
          out    Security::Opaque            evidence_check
     );

     NRVerificationResult verify_evidence (
          in     Security::Opaque            input_token_buffer,
          in     Security::Opaque            evidence_check,
          in     boolean                     form_complete_evidence,
          in     boolean                     token_buffer_complete,
          out    Security::Opaque            output_token,
          out    Security::Opaque            data_included_in_token,
          out    boolean                     evidence_is_complete,
          out    boolean                     trusted_time_used,
          out    Security::TimeT             complete_evidence_before,
          out    Security::TimeT             complete_evidence_after
     );

     void get_token_details (
          in     Security::Opaque                  token_buffer,
          in     boolean                           token_buffer_complete,
          out    string                            token_generator_name,
          out    NRPolicyFeatures                  policy_features,
          out    EvidenceType                      evidence_type,
          out    Security::UtcT                    evidence_generation_time,
          out    Security::UtcT                    evidence_valid_start_time,
          out    DurationInMinutes                 evidence_validity_duration,
          out    boolean                           data_included_in_token,
          out    boolean                           request_included_in_token,
          out    RequestFeatures                   request_features
     );

     boolean form_complete_evidence (
          in     Security::Opaque                  input_token,
          out    Security::Opaque                  output_token,
          out    boolean                           trusted_time_used,
          out    Security::TimeT                   complete_evidence_before,
          out    Security::TimeT                   complete_evidence_after
     );
};

interface NRPolicy {

     void get_NR_policy_info   (
          out    Security::ExtensibleFamily                   NR_policy_id,
          out    unsigned long                                policy_version,
          out    Security::TimeT                              policy_effective_time,
          out    Security::TimeT                              policy_expiry_time,
          out    EvidenceDescriptorList                       supported_evidence_types,
          out    MechanismDescriptorList                      supported_mechanisms
     );
```

```
          boolean set_NR_policy_info (
               in      MechanismDescriptorList          requested_mechanisms,
               out     MechanismDescriptorList          actual_mechanisms
          );
     };
};
```

A.7 Security Replaceable Service Interfaces

This section defines the IDL interfaces to the Security objects, which should be replaced if there is a requirement to replace the Security services used for security associations (i.e. the Vault and Security Contexts, Access Decision, and Audit Decision). This section comprises the module *SecurityReplaceable*. This module depends on the *CORBA*, *Security,* and *SecurityLevel2* modules. The interfaces are described in Section 15.7, Implementor's Security Interfaces.

```
module SecurityReplacable {

     // Forward ref of Security Context object

     interface SecurityContext ;

     interface Vault {
          Security::AssociationStatus init_security_context (
               in SecurityLevel2::CredentialsList
                                                        creds_list,
               in Security::SecurityName                target_security_name,
               in Object                                target,
               in Security::DelegationMode              delegation_mode,
               in Security::OptionsDirectionPairList    association_options,
               in Security::MechanismType               mechanism,
               in Security::Opaque                      mech_data, //from IOR
               in Security::Opaque                      chan_binding,
               out Security::Opaque                     security_token,
               out SecurityContext                      security_context
          );

          Security::AssociationStatus accept_security_context (
               in SecurityLevel2::CredentialsList
                                                        creds_list,
               in Security::Opaque                      chan_bindings,
               in Security::Opaque                      in_token,
               out Security::Opaque                     out_token
          );

          Security::MechandOptionsList get_supported_mechs ();
     };
```

```
        interface SecurityContext {

            readonly attribute SecurityLevel2::CredentialsList
                                                                  received_credentials;
            readonly attribute Security::SecurityFeatureValueList
                                                            security_features ;

            Security::AssociationStatus continue_security_context (
                in      Security::Opaque                    in_token,
                out     Security::Opaque                    out_token
            );

            void protect_message (
                in      Security::Opaque                    message,
                in      Security::QOP                       qop,
                out     Security::Opaque                    text_buffer,
                out     Security::Opaque                    token
            );
            boolean reclaim_message (
                in      Security::Opaque                    text_buffer,
                in      Security::Opaque                    token,
                out     Security::QOP                       qop,
                out     Security::Opaque                    message
            );

            boolean is_valid (
                out Security::UtcT                          expiry_time
            );

            boolean refresh ();
        };

        interface AccessDecision {

            boolean access_allowed (
                in      SecurityLevel2::CredentialsList         cred_list,
                in      Object                                  target,
                in      CORBA::Identifier                       operation_name,
                in      CORBA::Identifier                       target_interface_name
            );
        };
};
```

The interfaces for interceptors are considered as CORBA core extensions, so the IDL for these is summarized in Appendix B, Summary of CORBA 2 Core Changes, not here.

A.8 Secure Inter-ORB Protocol (SECIOP)

The SECIOP module holds structure declarations related to the layout of message fields in the secure inter-ORB protocol. This module does not depend on any other module.

```
module SECIOP {

    const       IOP::ComponentId          TAG_GENERIC_SEC_MECH = 12;

    const       IOP::ComponentId          TAG_ASSOCIATION_OPTIONS = 13;

    const       IOP::ComponentId          TAG_SEC_NAME = 14;

    struct TargetAssociationOptions{
                Security::AssociationOptions              target_supports;
                Security::AssociationOptions              target_requires;
    };

    struct GenericMechanismInfo {
                sequence <octet>                        security_mechanism_type;
                sequence <octet>                          mech_specific_data;
                sequence <IOP::TaggedComponent>           components;
    };

    enum MsgType {
                MTEstablishContext,
                MTCompleteEstablishContext,
                MTContinueEstablishContext,
                MTDiscardContext,
                MTMessageError,
                MTMessageInContext
    };

    struct ulonglong {
                unsigned long low;
                unsigned long high;
    };

    typedef ulonglong ContextId;

    enum ContextIdDefn {
                Client,
                Peer,
                Sender
    };

    struct EstablishContext {
                ContextId                 client_context_id;
                sequence <octet>          initial_context_token;
    };
```

```
        struct CompleteEstablishContext {
                ContextId               client_context_id;
                boolean                 target_context_id_valid;
                ContextId               target_context_id;
                sequence <octet>        final_context_token;
        };

        struct ContinueEstablishContext {
                ContextId               client_context_id;
                sequence <octet>        continuation_context_token;
        };

        struct DiscardContext {
                ContextIdDefn           message_context_id_defn;
                ContextId               message_context_id;
                sequence <octet>        discard_context_token;
        };

        struct MessageError {
                ContextIdDefn           message_context_id_defn;
                ContextId               message_context_id;
                long                    major_status;
                long                    minor_status;
        };

        enum ContextTokenType {
                SecTokenTypeWrap,
                SecTokenTypeMIC
        };

        struct MessageInContext {
                ContextIdDefn           message_context_id_defn;
                ContextId               message_context_id;
                ContextTokenType        message_context_type;
                sequence <octet>        message_protection_token;
        };
};
```

A.9 Values for Standard Data Types

A number of data types in this specification allow an extensible set of values, so the user can add values as required to meet his own security policies. However, if all users defined their own values, portability and interoperability would be seriously restricted.

Therefore, some standard values for certain data types are defined. These include the values that identify:

- Security attributes (privilege and other attribute types)
- Rights families
- Audit event families and types
- Security mechanism types as used in the IOR (and Vault, etc.)

Rights families and audit event families are defined as an *ExtensibleFamily* type. This has a family definer value registered with OMG and a family id defined by the family definer. Security attribute types also have family definers. Family definers with values 0 - 7 are reserved for OMG. The family value 0 is used for defining standard types (e.g., of security attributes).

A.9.1 Attribute Types

Section 15.5, Application Developer's Interfaces, defines an attribute structure for privilege and other attributes. This includes:

- A family, as previously described.
- An attribute type. Users may add new attribute types. Two standard OMG families are defined: the family of privilege attributes (family = 1), and the family of other attributes (family = 0). Types in these families are listed in the following table.
- An optional defining authority. This indicates the authority responsible for defining the value within the attribute type. Some policies demand that multiple sources of values for a given attribute type be supported (e.g. a policy accepting attribute values defined outside the security domain). These policies give rise to a risk of value clashes. The defining authority field is used to separate these values. When not present (i.e. length = 0), the value defaults to the name of the authority that issued the attribute.
- An attribute value. The attribute value is defined as a sequence<octet>, which someone who understands that attribute type can decipher.

Table A-1 Attribute Values

Attribute	**Type Value**	**Meaning**
Privilege Attributes (family = 1)		**All privilege attributes are used for access control**
Public	1	The principal has no authenticated identity
AccessId	2	The identity of the principal used for access control
PrimaryGroupId	3	The primary group to which the principal belongs
GroupId	4	A group to which the principal belongs
Role	5	A role the principal takes
AttributeSet	6	An identifier for a set of related attributes, which a user or application can obtain
Clearance	7	The principal's security clearance
Capability	8	A capability
Other Attributes (family = 0)		
AuditId	1	The identity of the principal used for auditing
AccountingId	2	The id of the account to be charged for resource use
NonRepudiationId	3	The id of the principal used for non-repudiation

A.9.2 Rights Families and Values

Administration is simplified by defining rights that provide access to a set of operations, so the administrator only needs to know what rights are required, rather than the semantics of particular operations.

Rights are grouped into families. Only one rights family is defined in this specification. The family definer is OMG (value 0) and the family id is CORBA (value 1). Other families may be added by vendors or users.

Three values are specified for the standard CORBA rights family.

Table A-2 CORBA Rights Family Values

Right	**Meaning**
get	Used for any operation on the object that does not change its state
set	For operations on an object that changes its state
manage	For operations on the attributes of the object, not its state

A.9.3 Audit Event Families and Types

Events, like rights, are grouped into families as defined in Section 15.5, Application Developer's Interfaces.

Only one event family is defined in this specification. This has a family definer of OMG (value 0) and family of SYSTEM (value 1) and is used for auditing system events. All events of this type are audited by the object security services, or the underlying security services they use. Some of these events must be audited by secure object systems conforming to SecurityFunctionality Level 1 (though in some cases, the event may be audited by underlying security services). Other event types are identified so that, if produced, a standard record is generated, so that audit trails from different systems can more easily be combined. System audit events are specified in Table A-3.

Table A-3 System Audit Events

Event Type	Whether Mandatory	Meaning and Event Specific Data
Principal authentication	Yes	Authentication of principals, either via the principal authentication interface or underlying security services
Session authentication	Yes	Security association/peer authentication
Authorization	Yes	Authorization of an object invocation (normally using an Access Decision object)
Invocation	No	Object invocation (i.e. the request/reply)
Security environment state change	No	Change to the security environment for this client or object (e.g. set_security_features, override_default_credentials)
Policy change	Yes	Change to a security policy (using the administrative interfaces in Section 15.6, Administrator's Interfaces)
Object creation	No	Creation of an object
Object destruction	No	Destruction of an object
Non-repudiation	No	Generation or verification of evidence

Application audit policies are expected to use application audit families.

A.9.4 Security Mechanisms

The security specification allows use of different mechanisms for security associations. These are used in the Interoperable Object Reference and also on the interface to the Vault.

No values for these are defined in this version of the specification. However, values will be defined in response to the Out-of-the-Box Interoperability RFP. Values will be registered by OMG as described in Appendix H, Interoperability Guidelines.

Appendix B Summary of CORBA 2 Core Changes

B.1 Introduction

In a secure object environment, security must be pervasive and automatically enforced, so that it cannot be bypassed. Both clients (which may or may not be objects) and target objects require a secure environment in which security policies will be enforced.

The CORBA security specification requires a number of changes to the CORBA Core to provide this security. Where possible, the changes proposed are made general, so future services can make use of them, rather than being specific to security.

This appendix describes the changes needed to the CORBA Core. It also specifies one change to the Transaction Service to have it use a general mechanism for obtaining the initial reference to the `Current` pseudo-object.

B.2 Finding What Security Facilities Are Supported

This specification provides an operation, **`get_service_information`**, which can be used to find what security facilities are supported by this implementation (i.e. what security functionality level and options), and also some details about the mechanism and policy options.

The **`get_service_information`** operation could be used for information about other CORBA facilities and services, so is not security-specific, though only security details are specified.

The specific changes required in the CORBA module appear in Section B.9.1, CORBA Module Changes to Support Security Level 1.

B.3 Extension to the Use of Current

The Transaction Service introduced a `Current` interface to allow an application to demarcate and manage the transaction associated with the current thread of activity (i.e. the execution context of the client or target object).

This specification generalizes this use of `Current` so it can be used to handle other information associated with the execution context at both client and target objects. In particular, it associates security information, such as credentials, with `Current` and provides means for accessing it.

The Current object in the environment may provide both Transaction and Security operations, depending on the implementation.

For security, there are two new interfaces: `SecurityLevel1::Current` and `SecurityLevel2::Current`, which the `Current` pseudo-object in a secure object system supports. The pseudo-OMG IDL for these are presented in Appendix A, Consolidate OMG IDL.

The mechanism for obtaining a reference to the `Current` object is provided by the new **`get_current`** operation of the ORB. The details of changes that need to be made to CORBA and `CosTransactions` to incorporate this general mechanism are in Section B.9.4, Changes to Support the Current Pseudo-Object. A single operation is added to the ORB interface:

```
Current get_current ( );
```

Return Value

An object reference to the `Current` pseudo-object.

B.4 Extensions to Object Interfaces for Security

In a similar manner, a secure object system extends the existing `CORBA::Object` interface which is implicitly supported by all objects, with the operations in the `SecurityLevel1::Object` and `SecurityLevel2::Object` interface. As with most of the operations in the existing `CORBA::Object` interface, the additional security functions operate locally on the object reference and are not implemented as invocations on the target. See Interfaces in Section 15.5.5, Object Reference, for details of operations provided by `SecurityLevel1::Object` and `SecurityLevel2::Object.`

Note that at a client in a secure environment, the object reference of objects that are not themselves in a secure environment will still contain the **`SecurityLevel1::Object`** or **`SecurityLevel2::Object`** operations (depending on the level of security supported by the ORB), since object reference operations are implemented by the client ORB. Security-aware applications will access these security-specific operations by using the `SecurityLevel1::Object` or the `SecurityLevel2::Object` interface instead of the vanilla `CORBA::Object` interface. Others will transparently continue to use the usual `CORBA::Object` interface, and still be provided the level of security appropriate for security-unaware applications.

B.5 Extensions to CORBA for Domains and Policies

In a secure object system, all objects should be subject to policy. The CORBA security specification therefore specifies policy domains, where each domain has a domain manager and a number of associated security policies.

Both the applications and ORB need to be able to find what policies apply so they can enforce them. Administrative applications need to be able to find the domain managers, and hence the policy objects, so they can administer the policies.

Domain managers, and the way of finding policies associated with them, are not security-specific. Therefore, the **`get_policy`** and **`get_domain_managers`** operations needed to support this (see Section 15.6, Administrator's Interfaces) are proposed as extensions to the standard CORBA Object interface, rather than as part of the security service specific Object interfaces. (Note that this specification does not specify interfaces for managing membership of domains, as this is assumed to be done by a Management or Collections service.)

Ensuring that all objects are subject to security policy also affects the way objects are created. When objects are created, they must automatically be made members of domains, and so subject to the security policies for those domains.

Many applications, even those that create other objects, are often unaware of security, so these applications should not have to take any special action to ensure that the newly created object is subject to policy.

Therefore, `BOA::create` must be extended as described in the Implementor's View of Secure Object Creation in Section 15.4.5, Security Object Models. This change does not affect the definition of the `BOA::create` interface; rather it has implications for its implementation. As previously noted, domains and policy mechanisms are not specific to security. The specific changes to the CORBA module are in Section B.9.2, CORBA Module Changes to Support Security Level 2.

B.6 Further Definition of ORB Services

This section gives an enhanced definition of the ORB Services, which were introduced to CORBA 2 as part of the Interoperability specification. This enhanced definition is required to support the ORB Services replaceability conformance option and covers the Interceptor interfaces used to implement security functions during invocation. It does not specify how ORB service implementations are registered with the ORB, nor their relationship with specific object adaptors, since this can and should be addressed by the generic ORB technology adoption process.

B.6.1 ORB Core and ORB Services

The ORB Core is defined in the CORBA architecture as "that part of the ORB which provides the basic representation of objects and the communication of requests." ORB Services, such as the Security Services, are built on this core and extend the basic functions with additional qualities or transparencies, thereby presenting a higher-level ORB environment to the application.

The function of an ORB service is specified as a transformation of a given message (a request, reply, or derivation thereof). A client may generate an object request, which necessitates some transformation of that request by ORB services (for example, Security Services may protect the message in transit by encrypting it).

B.6.2 Interceptors

An interceptor is responsible for the execution of one or more ORB services. Logically, an interceptor is interposed in the invocation (and response) path(s) between a client and a target object. When several ORB services are required, several interceptors may be used.

Two types of interceptors are defined in this specification:

- Request-level interceptors, which execute the given request.
- Message-level interceptors, which send and receive messages (unstructured buffers) derived from the requests and replies.

Interceptors provide a highly flexible means of adding portable ORB Services to a CORB-compliant object system. The flexibility derives from the capacity of a binding between client and target object to be extended and specialized to reflect the mutual requirements of client and target. The portability derives from the definition of the interceptor interface in OMG IDL.

The kinds of interceptors available are known to the ORB. Interceptors are created by the ORB as necessary during binding, as described next.

B.6.3 Client-Target Binding

The Security architecture builds upon the ORB Interoperability architecture in considering the selection of ORB Services as part of the process of establishing a binding between a client and a target object.

A binding provides the context for a client communicating with a target object via a particular object reference. The binding determines the mechanisms that will be involved in interactions such that compatible mechanisms are chosen and client and target policies are enforced. Some requirements, such as auditing or access control, may be satisfied by mechanisms in one environment, while others, such as authentication, require cooperation between client and target. Binding may also involve reserving resources in order to guarantee the particular qualities of service demanded.

Although resolution of mechanisms and policies involves negotiation between the two parties, this need not always involve physical interactions between the parties as information about the target can be encoded in the object reference, allowing resolution of the client and target requirements to take place in the client. The outcome of the negotiation can then be sent with the request, for example, in the GIOP service context. Where there is an issue of trust, however, the target must still check that this outcome is valid.

The binding between client and target at the application level can generally be decomposed into bindings between lower-level objects. For example, the agreement on transport protocol is an agreement between two communications endpoints, which will generally not have a one-to-one correspondence to application objects. The overall binding therefore includes a set of related sub-bindings which may be shared, and also potentially distributed among different entities at different locations.

B.6.4 Binding Model

No object representing the binding is made explicitly visible since the lifetime of such an object is not under the control of the application, an existing binding potentially being discarded, and a new one made without the application being aware of the fact.

Instead, operations that will affect how a client will interact with a target are provided on the Object interface and allow a client to determine how it will interact with the target denoted by that object reference. On the target side, the binding to the client may be accessed through the Current interface. This indirect arrangement permits a wide range of implementations that trade-off the communication and retention of binding information in different ways.

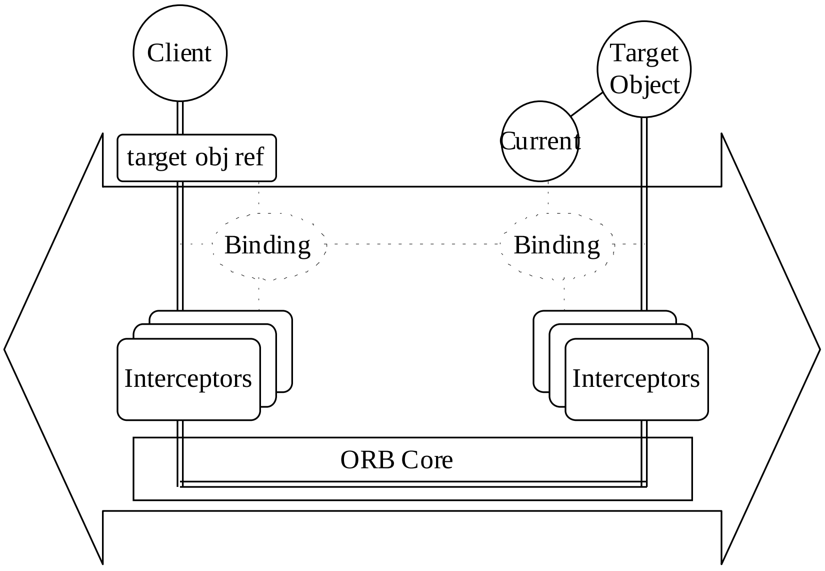

Figure B-1 Binding Model

The action of establishing a binding is generally implicit, occurring no later than the first invocation between client and target. It may be necessary for a client to establish more than one binding to the same target object, each with different attributes (for example, different security features). In this case, the client can make a copy of the object reference using `Object::duplicate` and subsequently specify different attributes for that reference.

The scope of attributes associated with an object reference is that of the object reference instance, i.e. the attributes are *not* copied if the object reference is used as an argument to another operation or copied using `Object::duplicate`. If an object reference is an *inout* argument, the attributes will still be associated with the object reference after the call if the reference still denotes the same object, but not otherwise.

B.6.5 Establishing the Binding and Interceptors

An ORB maintains a list of interceptors, which it supports, and when these are called. Note that at the client, when handling the request, the request-level interceptors are always called before the message level ones, while at the target the message-level ones are called first.

When the ORB needs to bind an object reference, it refers to the characteristics of the target object and relates this to the types of interceptor it supports. From this it determines the appropriate type of interceptor to handle the request and creates it, passing the object reference in the call. (No separate interceptor initialization operation is used. The `client_invoke/target_invoke` or `send_message/receive_message` calls are used both for the first invocation and for subsequent ones.)

When an interceptor is created, it performs its bind time functions. These may involve getting the policies that apply to the client (and have not been overridden by the client) and to the target. This could involve communicating with the target, for example, a secure invocation interceptor setting up a security association. Note that the ORB Core itself is unaware of service-specific policies. In addition to performing its specific functions, the interceptor must continue the request by invoking object(s) derived from the given object reference.

The interceptors themselves maintain per-binding information relevant to the function they perform. This information will be derived from:

- The policies that apply to the client and target object because of the domains to which they belong, for example the access policies, default quality of protection.
- Other static properties of the client and target object such as the security mechanisms and protocols supported.
- Dynamic attributes, associated with a particular execution context or invocation (for example, whether a request must be protected for confidentiality).

If the relevant client or target environment changes, part or all of a binding may need to be reestablished. For example, the secure invocation interceptor may detect that the invocation credentials have changed and therefore needs to establish a new security association using the new credentials. If the binding cannot be reestablished, an exception is raised to the application, indicating the cause of the problem.

Similarly, at the target, the ORB will create an instance of each interceptor needed there. A single interceptor handles both requests and replies at the client (or target), as these share context information.

B.6.6 Using Interceptors

When a client performs an object request, the ORB Core uses the binding information to decide which interceptors provide the required ORB Services for this client and target as described in Section 15.7.3, Security Interceptors.

Request-Level Interceptors

Request-level interceptors could be used for services such as transaction management, access control, or replication. Services at this level process the request in some way. For example, they may transform the request into one or more lower-level invocations or make checks that the request is permitted. The request-level interceptors, after performing whatever action is needed at the client (or target), reinvoke the (transformed) request using the Dynamic Invocation Interface (DII) `CORBA::Request::invoke`. The interceptor is then stacked until the invocation completes, when it has an opportunity to perform further actions, taking into account the response before returning.

Interceptors can find details of the request using the operations on the request as defined in the Dynamic Skeleton interface in CORBA 2. This allows the interceptor to find the target object[1], operation name, context, parameters, and (when complete) the result.

If the interceptor decides not to forward the request, for example, the access control interceptor determines that access is not permitted, it indicates the appropriate exception and returns.

When the interceptor resumes after an inner request is complete, it can find the result of the operation using the **`result`** operation on the Request pseudo-object, and check for exceptions using the **`exception`** operation, etc. before returning.

Message-Level Interceptors

When remote invocation is required, the ORB will transform the request into a message that can be sent over the network. Message-level interceptors operate on messages in general without understanding how these messages relate to requests (for example, the message could be just a fragment of a request). Note that the message interceptors may achieve their purpose not by (just) transforming the given message, but by communicating using their own message (for example, to establish a secure association). Fragmentation and message protection are possible message-level interceptors.

`send_message` is always used when sending a message, so is used by the client to send a request (or part of a request), and by the target to send a reply.

When a client message-level interceptor is activated to perform a **`send_message`** operation, it transforms the message as required, and calls a **`send`** operation to pass the message on to the ORB and hence to its target. Unlike invoke operations, **`send`** operations may return to the caller without completing the operation. The interceptor can then perform other operations if required before exiting. The client interceptor may next be called either using **`send_message`** to process another outgoing message, or using **`receive_message`** to process an incoming message.

A target message-level interceptor also supports **`send_message`** and **`receive_message`** operations, though these are obviously called in a different order from the client side.

B.6.7 Interceptor Interfaces

Two interceptor interfaces are specified, both used only by the ORB:

- **RequestInterceptor** for operations on request-level interceptors. Two operations are supported:
  - **`client_invoke`** for invoking a request-level interceptor at the client.
  - **`target_invoke`** for invoking a request-level interceptor at the target.
- **MessageInterceptor** for operations on message-level interceptors. Two operations are supported:

1.It is assumed that the target object reference is available, as this is described in the C++ mapping for DSI, though not yet in the OMG IDL.

- **send_message** for sending a message from the client to the target or the target to the client.
- **receive_message** for receiving a message.

Request-level interceptors operate on a representation of the request itself as used in the CORBA Dynamic Invocation and Skeleton interfaces. (It is assumed that the Request pseudo-object defined in the Dynamic Invocation interface is compatible with the ServerRequest pseudo-object in the Dynamic Skeleton interface, and so supports operations such as **op_name**, which returns the name of the operation being invoked.)

Client and Target Invoke

These invoke a request-level interceptor at the client or target. Both operations have identical parameters and return values.

```
interface RequestInterceptor: Interceptor  // PIDL
{
    void client_invoke (
        inout   Request    request);

    void target_invoke (
        inout   Request    request);
};
```

Parameters

request	The request being invoked. This is a pseudo-object as defined in the Dynamic Invocation Interface. After invocation, output parameters and the associated result and exceptions may have been updated.

Send and Receive Message

These invoke a message-level interceptor to send and receive messages. Both operations have identical parameters and return values.

```
interface MessageInterceptor: Interceptor
{
    void send_message (
        in      Object              target,
        in      Message             msg);
    void receive_message (
        in      Object              target,
        in      Message             msg,);
};
```

Parameters

target	The target object reference. Note: The target here may not be the same as seen by the application. For example, a replication request-level interceptor may send the request to more than one underlying object.
msg	The message to be handled by this interceptor.

B.6.8 Interface Changes Required for Interceptors

Use of binding and interceptors requires extra interfaces on the target object reference to get components (e.g. from the multicomponent profiles in the IOR). It is assumed that these will be specified by the CORBA 2 (revision) task force, since this group is developing the general form of the multicomponent profile structure.

B.7 Further Definition of ORB Interoperability

This specification describes the use of and extensions to the CORBA 2.0 interoperability protocol and Interoperable Object Reference (IOR) to allow secure interoperability between ORBs. Additional tags are defined in IOR Security Components of the DCE-CIOP in Section 15.8.5, DCE-CIOP with Security, for security information in the IOR. Extra messages are added to the IOP/IIOP protocol for protected messages and replies and are defined in Section 15.8.4, Secure Inter-ORB Protocol (SECIOP). These are designed to be able to fit with GIOP fragmentation proposals also being considered. These security extensions can be used with a range of different security mechanisms for security associations.

This submission describes TAGs for security for use in multicomponent profiles. Modifications to the CORBA 2.0 IOR specification to support this are being discussed by the Interoperability Revision Task Force, and have also been discussed with the security submitters.

Appendix I, Further ORB Interoperability, contains a description of possible modifications to CORBA 2 for this, but the definitive version of such changes will come from the Revision Task Force.

The security submitters therefore require the Interoperability Revision Task Force to define the modifications needed. This should result in multicomponent profiles, which will be used both by IIOP- or IIOP-derived protocols and DCE-CIOP.

This specification maintains strict message format compatibility with the IIOP protocol as defined in CORBA 2.0. It also maintains compatibility with existing unsecured implementations of DCE-CIOP.

B.8 Implications of Assurance

The ORB must function correctly, enforcing security policy on object invocation, object creation, etc. as defined in this specification. It must do this to the level of assurance specified in its Conformance Statement (see Appendix F, Conformance Statement). It must also meet other assurance requirements defined there such as preventing interference between objects to the required extent.

B.9 Enhancements to the CORBA Module

The enhancements to the CORBA Core previously discussed requires the following modifications to the CORBA module.

B.9.1 CORBA Module Changes to Support Security Level 1

The following additions and changes to the CORBA module are necessary for the Security Level 1 conformance point

New Data Types Added to the CORBA Module

The following data types need to be inserted into the CORBA module preceding the declaration of the ORB interface.

```
module CORBA {

typedef unsigned short ServiceType ;

const ServiceType Security = 1 ;
// other Service types to be defined

typedef unsigned long ServiceOption ;

const ServiceOption   SecurityLevel1 = 1;
const ServiceOption   SecurityLevel2 = 2;
const ServiceOption   NonRepudiation = 3;
const ServiceOption   SecurityORBServiceReady = 4;
const ServiceOption   SecurityServiceReady = 5;
const ServiceOption   ReplaceORBServices = 6 ;
const ServiceOption   ReplaceSecurityServices = 7;
const ServiceOption   StandardSecureInteropability = 8;
const ServiceOption   DCESecureInteroperability = 9;

// Service details supported by the implementation

typedef unsigned long ServiceDetailType;

// security mech type(s) supported for secure associations

const  ServiceDetailType  SecurityMechanismType = 1;
// privilege types supported in standard access policy

const   ServiceDetailType  SecurityAttribute = 2;

    struct  ServiceDetail {
        ServiceDetailType     service_detail_type;
        sequence <octet>      service_detail;
    };

    struct ServiceInformation {
        sequence <ServiceOption>  service_options;
        sequence <ServiceDetail>  service_details;
    };
};
```

Extensions to the ORB Interface

The operation **get_service_information** needs to be appended to the list of operations in the ORB interface.

```
module CORBA {
    interface ORB {
        boolean get_service_information (
            in      ServiceType           service_type,
            out     ServiceInformation    service_information
        );
    };
};
```

The specific change consists of adding the lines

```
boolean get_service_information (
            in      ServiceType           service_type,
            out     ServiceInformation    service_information
        );
```

to the list of operations in the definition of the ORB interface on page 7-2 in *CORBA V2.0 July 1995*. The associated addition of data types and interfaces must precede the declaration of the ORB interface in the CORBA module.

B.9.2 CORBA Module Changes to Support Security Level 2

The following additions and changes to the CORBA module are necessary for the Security Level 2 conformance point.

New Data Types Added to the CORBA Module

The following data types need to be added to the CORBA module for this conformance level.

```
module CORBA {
    enum PolicyType {
        SecClientInvocationAccess,
        SecTargetInvocationAccess,
        SecApplicationAccess,
        SecClientInvocationAudit,
        SecTargetInvocationAudit,
        SecApplicationAudit,
        SecDelegation,
        SecClientSecureInvocation,
        SecTargetSecureInvocation,
        SecNonRepudiation,
        SecConstruction
    };
};
```

New Interfaces Added to the CORBA Module

The following segment of OMG IDL needs to be inserted into the CORBA module preceding the definition of the Object interface.

```
module CORBA
{
    // Interfaces to support the basic management infrastructure
    interface Policy {
    // Features common to all Policies
    };

    interface DomainManager {

        // get policies for objects in this domain
        Policy get_domain_policy (
            in PolicyType                 policy_type
        );
    typedef sequence<DomainManager> DomainManagerList;
    };

    interface ConstructionPolicy : Policy{
        void make_domain_manager(
            in InterfaceDef               object_type
        );
    };
};
```

Extensions to the Object Interfaces

The operations in the OMG IDL block shown next need to be appended to the list of operations in the definition of the Object interface in the CORBA module.

```
module CORBA {
    interface Object {
    // operations to facilitate basic management infrastructure
        Policy get_policy (
            in PolicyType                policy_type
        );
        DomainManagerList get_domain_managers();
    };
};
```

The specific changes are on page 7-3 of *CORBA V2.0 July 1995*. Append the following lines to the list of operations in the definition of Object interface.

```
        Policy get_policy (
            in PolicyType                policy_type
        );
        DomainManagerList get_domain_managers();
```

Add the corresponding documentation for these operations from Section 15.6.2 of this document to page 7-3 of *CORBA V2.0 July 1995*.

B.9.3 CORBA Module Changes for Replaceability Conformance

The following additions and changes to the CORBA module are necessary for supporting the Interceptor mechanism to satisfy the ORB Services Replaceability conformance option.

New Interfaces Added to the CORBA Module

The following new interfaces need to be added to the CORBA module to support this conformance option.

The message-level interceptor has a Message parameter, which is a pseudo-object (see the Request pseudo-object used on the message interface). This pseudo-object comprises an ordered sequence of octets. The operations for accessing it should be aligned with the operations for operating on collections as expected to be defined for the Collections Service technology adoption process.

```
module CORBA {
    interface Interceptor { // PIDL
        // Generic interceptor operations (management etc.)
    };

    interface RequestInterceptor: Interceptor { // PIDL
        void client_invoke (
            inout Request                request
        );
        void target_invoke (
            inout Request                request
        );
    };
    interface MessageInterceptor: Interceptor { // PIDL
        void send_message (
            in Object                    target,
            in Message                   msg
        );
        void receive_message (
            in Object                    target,
            in Message                   msg
        );
    };
};
```

Add corresponding documentation for these operations from Section B.6, Further Definition of ORB Services, to the appropriate section of *CORBA V2.0 July 1995*.

B.9.4 Changes to Support the Current Pseudo-Object

The CORBA module changes and additions described here are necessary for supporting Security Replaceability and Security Level 2. The changes to Transaction service are not necessary from the perspective of meeting any security requirements, but is highly recommended for maintaining uniformity of mechanisms and interfaces.

New Interface Added to the CORBA Module

```
module CORBA
{
    // interface for the Current pseudo-object
    interface Current {            // PIDL
    };
};
```

Extensions to the ORB Interfaces

The following extension needs to be made to the ORB interface.

```
module CORBA {
    interface ORB {
        Current get_current ();
    };
};
```

The specific change consists of adding

```
        Current get_current ( );
```

to the definition of the ORB interface on page 7-2 in *CORBA V2.0 July 1995*. The associated addition of data types and interfaces must precede the declaration of the ORB interface in the CORBA module.

Transaction Service Changes

The following change needs to be made to the Transaction Service to make it compatible with and able to use the **`ORB::get_current`** operation. The change is to be made in *CORBAservices: Common Object Services Specification, Rev. Ed. March 31, 1995, OMG Document Number 95-3-31.*

On page 10-19, change the first line of the OMG IDL in the box from

```
    interface Current {
```

to

```
    interface Current : CORBA::ORB::Current {
```

B.9.5 CORBA Module Deprecated Interfaces

`SecurityLevel2::Credentials` is the preferred interface for retrieving information about the identity of callers in CORBA Security conformant ORB implementations; the use of `CORBA::get_principal` is deprecated, and it is anticipated that this interface will be eliminated in a future CORBA revision.

Appendix C Relationship to Other Services

C.1 Introduction

This appendix describes the relationship between Object Services and Common Facilities and the security architecture components, if they are to participate in a consistent, secure object system.

C.2 General Relationship to Object Services and Common Facilities

In general, Object Services and Common Facilities, like any application objects, may be unaware of security, and rely on the security enforced automatically on object invocations. As for application objects, access to their operations can be controlled by access policies as described in Section 15.3, Security Reference Model, Section 15.5, Application Developer's Interfaces, and elsewhere.

An Object Service or Common Facility needs to be aware of security if it needs to enforce security itself. For example, it may need to control access to functions and data at a finer granularity than at object invocation, or need to audit such activities. The way it can do this is described in Section 15.4, Security Architecture. Existing Object Services should be reviewed to see if such access control and auditing is required.

If an Object Service or Common Facility is required to be part of a more secure system, some assurance of its correct functioning, if security relevant, is needed, even if it is not responsible for enforcing security itself. See Appendix E, Guidelines for a Trustworthy System, for guidelines on this matter.

Where an Object Service is called by an ORB service as part of object invocation in a secure system, there is a need to ensure security of all the information involved in the invocation. This requires ORB Services to be called in the order required to provide the specified quality of protection. For example, the Transaction Service must be invoked first to obtain the transaction context information before the whole message is protected for integrity and/or confidentiality.

In the following sections, we provide an initial estimation of the relationship between Security Service and other existing services and facilities.

C.3 Relationship with Specific Object Services

C.3.1 Transaction Service

This specification builds on the definition of Current introduced by the Transaction Service to provide information about the current execution context. It also specifies a general ORB operation for applications to get hold of an object reference to the Current pseudo-object (see Appendix B, Summary of CORBA 2 Core Changes).

In order to have the Transaction Service use the proposed mechanism, the definition of the `CosTransactions::Current` interface needs to be modified so that it is derived from `CORBA::ORB::Current`. The necessary change is presented in Section B.9.4, Changes to Support the Current Pseudo-Object.

C.3.2 Naming Service

For security, the object must be correctly identified wherever it is within the distributed object system. The Naming Service must do this successfully in an environment where an object name is unique within a naming context, and name spaces are federated. (However, to provide the required proof of identity, objects, and/or the gatekeepers which give access to them will be authenticated using a separate Authentication Service.) See Section E.6.2, Basis of Trust, for additional information about the relationship between security and names.

C.3.3 Event Service

The implementation of a Security Audit Service may involve the use of Event Service objects for the routing of both audits and alarms.

However, this is only possible if the Event Service itself is secure in that it protects the audit trail from modification and deletion. It must also be able to guard against recursion if it audits its own activities.

C.3.4 Persistent Object Service

No explicit use is made of this service. Audit trails may be saved using this service, in which case the implementation of the Persistent Object Service must ensure that data stored and retrieved through it is not tampered with by unauthorized entities. If it is used in the implementation of Security Service or by a secure application, it must follow the guidelines in Appendix E, Guidelines for a Trustworthy System.

C.3.5 Time Service

The Security Service uses the data types for time, timestamps, and time intervals as defined by the Time Service, so that applications can readily use the Time Service defined interfaces to manipulate the time data that the Security Service uses. The interfaces of Security Service do not explicitly pass any interfaces defined in the Time Service.

C.3.6 Other Services

The other services are not used explicitly. If any of them are used in the implementation of Security Service or by a secure application, it must be verified that the service used follows the guidelines in Appendix E, Guidelines for a Trustworthy System.

C.4 Relationship with Common Facilities

Because Management Services have been identified as Common Facilities in the Object Management Architecture, only minimal, security-specific administration interfaces are specified here. When Common Facilities Management services are specified, they will need to take into account the need for security management and administration identified in this specification. Also, such management services will themselves need to be secure.

This specification adds certain basic interfaces to CORBA, which form the basis for the minimal policy administration related interfaces and functionality that has been provided. Future management facilities are expected to build upon this foundation.

Appendix D Conformance Details

D.1 Introduction

Conformance to CORBA Security covers:

- **Main security functionality**. There are two possible levels.
    - *Level 1*: This provides a first level of security for applications unaware of security, and for those that have limited requirements to enforce their own security in terms of access controls and auditing.
    - *Level 2*: This provides more security facilities, and allows applications to control the security provided at object invocation. It also includes administration of security policy, allowing applications administering policy to be portable.
- **Security Functionality Options.** These are functions expected to be required in several ORBs, so are worth including in this specification, but are not generally required enough to form part of one of the main security functionality levels previously specified. There is only one such option in the specification.
    - *Non-Repudiation*: This provides generation and checking of evidence so that actions cannot be repudiated.
- **Security Replaceability**. This specification is designed to allow security policies to be replaced. The additional policies must also conform to this specification. This includes, for example, new Access Polices. Security Replaceability specifies if and how the ORB fits with different security services. There are two possibilities.
    - *ORB Services replaceability*: The ORB uses interceptor interfaces to call on object services, including security ones. It must use the specified interceptor interfaces and call the interceptors in the specified order. An ORB conforming to this does not include any significant security-specific code, as that is in the interceptors.
    - *Security Service replaceability*: The ORB may or may not use interceptors, but all calls on security services are made via the replaceability interfaces specified in Section 15.7, Implementor's Security Interfaces. These interfaces are positioned so that the security services do not need to understand how the ORB works, so they can be replaced independently of that knowledge.

  An ORB that supports one or both of these replaceability options may be Security Ready (i.e. support no security functionality itself, but be ready to have security added, or may support Security Functionality Level 1 or 2).

  Note: Some replaceability of the security mechanism used for secure associations may still be provided if the implementation uses some standard generic interface for security services such as GSS-API.
- **Secure Interoperability**: Possibilities are
    - *Secure Interoperability - Standard:* An ORB supporting this can generate/use security information in the IOR and can send/receive secure requests to/from other ORBs using the GIOP/IIOP protocol with the security (SECIOP)

enhancements defined in Section 15.8, Security and Interoperability, providing they can both use the same underlying security mechanism and algorithms for security associations.

- *Standard plus DCE-CIOP Option:* As for Standard, but secure DCE-CIOP is also supported.

If the ORB does not conform to one of these, it will not use the GIOP security enhancements, and so will interoperate securely only in an environment-specific way.

The conformance statement required for a CORBA Security conformant implementation is defined in Appendix F, Conformance Statement. Appendix F includes a checklist, which can be completed to show what the ORB conforms to; it is reproduced next. A main security functionality level must always be specified. Functional Options, Security Replaceability, and Security Interoperability should be indicated by checking the boxes corresponding to the function supported by the ORB.

Main Functionality Level		**Functional Options**	**Security Replaceability**				**Security Interoperability**	
1	2	Non Repudiation	ORB Services	Security Services	Security Ready - ORB Services	Security Ready - Security Services	Standard	Standard + DCE-CIOP

D.2 Security Functionality Level 1

Security Functionality Level 1 is the level to which all OMG-compliant security implementations must conform. It provides:

- A level of security functionality available to applications unaware of security. (It will, of course, also provide this functionality to applications aware of security.) This level includes security of the invocation between client and target object, simple delegation of client security attributes to targets, ORB-enforced access control checks, and auditing of security-relevant system events.
- An interface through which a security-aware application can retrieve security attributes, which it may use to enforce its own security policies (e.g. to control access to its own attributes and operations).

D.2.1 Security Functionality Required

An ORB supporting Level 1 security functionality must provide the following security features for all applications, whether they are security-aware or not.

- Allow users and other principals to be authenticated, though this may be done outside the object system.
- Provide security of the invocation between client and target object including:
  - Establishment of trust between them, where needed. At Level 1, this may be supported by ORB level security services or can be achieved in any other secure way. For example, it could use secure lower-layer communications. Mutual authentication need not be supported.
  - Integrity and/or confidentiality of requests and responses between them.
  - Control of whether this client can access this object. At this level, access controls can be based on "sets" of subjects and "sets" of objects. Details of the Access Policy and how this is administered are not specified.
- At an intermediate object in a chain of calls, the ability to be able to either delegate the incoming credentials or use those of the intermediate object itself.
- Auditing of the mandatory set of system's security-relevant events specified in Appendix A, Consolidated OMG IDL. In some cases, the events to be audited may occur, and be audited, outside the object system (for example, in underlying security services). In this case, the conformance statement must identify the product responsible for generating the record of such an event (or choice of product, for example, when the ORB is portable to different authentication services).

  At this level, auditing of object invocations need not be selectable. However, it must be possible to ensure that certain events are audited (see Section A.9, Values for Standard Data Types, for the list of mandatory events).

For security aware applications, it must also:

- Make the privileges of authenticated principals available to applications for use in application access control decisions.

These facilities require the ORB and security services to be initialized correctly. For example, the Current object at the client must be initialized with a reference to a credentials object for the appropriate principal.

D.2.2 Security Interfaces Supported

Security interfaces available to applications may be limited to:

- `get_service_information` providing security options and details (see Section 15.5.2, Finding Security Features).
- `get_attributes` on Current (see Interfaces under Section 15.5.6, Security Operations on Current).

No administrative interfaces are mandatory at this level.

D.2.3 Other Security Conformance

An ORB providing Security Functionality Level 1 may also conform to other security options. For example, it may also:

- Support some of the Security Functionality Options specified in Section D.4, Security Functionality Options.
- Provide security replaceability using either of the replaceability options.
- Provide secure interoperability, though in this case, will need to provide security associations at the ORB level (not lower-layer communications) as the protocol assumes security tokens are at this level.

D.3 Security Functionality Level 2

This is the functionality level that supports most of the application interfaces defined in Section 15.5, Application Developer's Interfaces, and the administrative interfaces defined in Section 15.6, Administrator's Interfaces. It provides a competitive level of security functionality for most situations.

D.3.1 Security Functionality Required

An ORB that supports Security Functionality Level 2 supports the functionality in Security Level 1 previously defined, and also:

- Principals can be authenticated outside or inside the object system.
- Security of the invocation between client and target objects is enhanced.
  - Establishment of trust and message protection can be done at the ORB level, so security below this (for example, in the lower layer communications) is not required (though may be used for some functions).
  - Further integrity options can be requested (e.g. replay protection and detection of messages out of sequence) but need not be supported.
  - The standard DomainAccessPolicy is supported for control of access to operations on objects.
  - Selective auditing of methods on objects is supported.
- Applications can control the options used on secure invocations. It can:
  - Choose the quality of protection of messages required (subject to policy controls).
  - Change the privileges in credentials.
  - Choose which credentials are to be used for object invocation.
  - Specify whether these can just be used at the target (e.g. for access control) or whether they can also be delegated to further objects.

- No further delegation facilities are mandatory, but the application can request "composite" delegation, and the target can obtain all credentials passed, in systems that support this. Note that "composite" here just specifies that both received credentials and the intermediate's own credentials should be used. It does not specify whether this is done by combining the credentials or linking them.
- Administrators can specify security policies using domain managers and policy objects as specified in Section 15.6, Administrator's Interfaces. The security policy types supported at Level 2 are all those defined in Section 15.6 except non-repudiation. The standard policy management interfaces for each of the Level 2 policies is supported.
- Applications can find out what security policies apply to them. This includes policies they enforce themselves (e.g. which events types to audit) and some policies the ORB enforces for them (e.g. default qop, delegation mode).
- ORBs (and ORB Services, if supported) can find out what security policies apply to them. They can then use these policy objects to make decisions about what security is needed (check if access is permitted, check if auditing is required) or get the information needed to enforce policy (get QOP, delegation mode, etc.) depending on policy type.

As at Level 1, these facilities require the ORB and security services to be initialized correctly.

D.3.2 Security Interfaces Supported

Interfaces supported at this level are:

- All application interfaces defined in Section 15.5, Application Developer's Interfaces (except those in Section 15.5.11, Non-repudiation).
- All security policy administration interfaces defined in Section 15.6, Administrator's Interfaces (except those for the non-repudiation policy).

Note that some of these interfaces may return a NO-IMPLEMENT exception, as not ORBs conforming to Level 2 Security need implement all possible values of all parameters. This will happen when:

- A privilege attribute is requested of a type that is not supported (attribute types supported are defined in Appendix A, Consolidated OMG IDL).
- A delegation mode is requested, which is not supported.
- A communication direction for association options is requested, which is not supported.

D.3.3 Other Security Conformance

An ORB providing Security Functionality Level 2 may also conform to other security options. For example, it may also:

- Support some of the Security Functionality Options specified in Section D.5, Security Replaceability.
- Provide security replaceability, using either of the replaceability options.
- Provide secure interoperability.

D.4 Security Functionality Options

An ORB may also conform to optional security functionality defined in this specification. Only one optional facilities is specified: non-repudiation.

Also, some requirements on conformance of additional facilities are specified.

D.4.1 Non-repudiation

Security Functionality

An ORB conforming to this must support the non-repudiation facilities for generating and verifying evidence described in The Model as Seen by Applications in Section 15.4.5, Security Object Models. Note that these use NRCredentials, which may be the same as the credentials used for other security facilities. Where non-repudiation is supported, the credentials acquired from the environment or generated by the authenticate operation must be able to support non-repudiation.

Security Interfaces Supported

The following interfaces must be supported. All are available to applications. They are:

- `set_/get_NR_features` as defined in Section 15.5.11, Non-repudiation.
- `generate_token`, `verify_evidence`, form complete evidence and get token details as defined in Section 15.5.11.
- Use of `set/get_credentials` on Current specifying the type of credentials to be used is NRCredentials.
- NR policy object with associated interfaces as in Section 15.6.7, Non-repudiation Policy Management.

Fit with Other Security Conformance

Non-repudiation requires use of credentials; thus it can only be used with ORBs, which support some of the interfaces defined in Security Functionality level 2. However, conformance to all of Security Functionality Level 2 is not a prerequisite for conformance to the non-repudiation security functionality option.

Secure interoperability as defined in Section D.6, Secure Interoperability, is not affected by non-repudiation. The evidence may be passed on an invocation as a parameter to a request, but the ORB need not be aware of this.

The current specification does not specify interoperability of evidence (i.e. one non-repudiation service handling evidence generated by another).

D.4.2 Conformance of Additional Policies

This specification is designed to allow security policies to be replaced. The additional policies must also conform to some of the interfaces in this specification if they are used to replace the standard policies automatically enforced on object invocation.

The case described next is for the addition of a new Access Policy which can be used for controlling access to objects automatically, replacing the standard DomainAccessPolicy.

Clearly, other policies can be replaced. For example, the audit policy could be replaced by one that used different selectors, or the delegation policy could be replaced by one that supported more advanced features.

Additional Access Policies

A new Access Policy, which is to be enforced automatically at invocation time, should be supported by providing a new Access Policy object. This must support the **`access_allowed`** operation defined in Access Decision Object under Section 15.7.4, Implementation-Level Security Object Interfaces, so that it can be called automatically by the ORB to check if access is allowed.

This policy object should be associated with a domain, and be specified as a client or target policy as for the standard Access Policy. The policy object should include administrative interfaces to allow the policy to be administered, but this need not (normally cannot) conform to the administrative interface defined for the standard policy.

D.5 Security Replaceability

This specifies how an ORB can fit with security services, which may not come from the same vendor as the ORB. As explained above, there are two levels where this can be done (apart from any underlying APIs used by an implementation).

D.5.1 Security Features Replaceability

Conformance to this allows security features to be replaced.

If it is provided without conformance to the ORB Service replaceability option (see Section D.5.2, ORB Services Replaceability), it requires the ORB to have a reasonable understanding of security, handling credentials, etc. and knowing when and how to call on the right security services.

Support for this replaceability option requires an ORB (or the ORB Services it uses) to use the implementation-level security interfaces as defined in Section 15.7, Implementor's Security Interfaces. This includes:

- The Vault, Security Context, Access Decision, Audit and Principal Authentication objects defined in Section 15.7.4, Implementation-Level Security Object Interfaces.
- This also includes the CORBA changes defined in Appendix B, Summary of CORBA 2 Core Changes.

D.5.2 ORB Services Replaceability

Conformance to this allows an ORB to know little about security except which interceptors to call in what order. This is intended for ORBs, which may use different ORB services from different vendors, and require these to fit together. It therefore provides a generic way of calling a variety of ORB Services, not just security ones. It also assumes that any of these services may have associated policies, which control some of their actions.

Support for this replaceability option requires an ORB to:

- Use the Interceptor interfaces defined in Section B.6 to call security interceptors defined in Section 15.7.3, Security Interceptors, in the order defined there.
- Use the `get_policy` interfaces (and the associated security policy interfaces such as `access_allowed`, `audit_needed` defined in Section 15.7.4, Implementation-Level Security Object Interfaces, for access control and audit and also `get_association_options` and `get_delegation_mode` defined in Section 15.6.6, Secure Invocation and Delegation Policies, for association options, quality of protection of messages, and delegation).

D.5.3 Security Ready for Replaceability

An ORB is Security Ready for Replaceability if it does not provide any security functionality itself, but does support one of the security replaceability options.

Security Functionality Required

An ORB that is Security Ready does not have to provide any security functionality, though must correctly respond to a request for the security features supported.

Security Interfaces Supported

- **`get_service_information`** operation providing security options and details (see Section 15.5.2, Finding Security Features).
- **`get_current`** operation to obtain the Current object for the execution context (see Section B.3, Extension to the Use of Current).

Other Security Conformance

An ORB that is Security Ready for replaceability supports one of the replaceability options. This should be done in such a way that the ORB can work without security, but can take advantage of security services when they become available. So it calls on the replaceability interfaces correctly (using dummy routines to replace security services when these are needed, but not available).

The ORB may also conform to secure interoperability, meaning it can transmit security tokens and handle protected messages returned by security interceptors and/or services in accordance with the secure interoperability security conformance option.

D.6 Secure Interoperability

The definition of secure interoperability in this document specifies that a conformant ORB can:

- Generate, and take appropriate action on, Interoperable Object References (IORs), which include security tags as specified in Section 15.8, Security and Interoperability.
- Transmit and receive the security tokens needed to establish security associations, and also the protected messages used for protected requests and responses once the association has been established according to the protocol defined in Section 15.8.

Note that a Security Ready ORB (i.e. with no built-in security functionality) may, by additions of appropriate security services, conform to secure interoperability.

The current security specification does *not* mandate a particular security mechanism for security associations (or the associated set of cryptographic algorithms they use), so for ORBs to interoperate securely, they must choose to use the same mechanism, algorithms, etc. (or use a bridge between them, if available). A future specification is expected to cover standard security mechanisms and algorithms.

D.6.1 Secure Interoperability - Standard

An ORB that conforms to this must support the security-enhanced IOR defined in Section 15.8, Security and Interoperability, and also GIOP/IIOP protocol with the SECIOP enhancements as defined in Section 15.8. (This is in line with CORBA 2 interoperability, where all interoperable ORBs must support the IOR and GIOP/IIOP.)

As for CORBA 2, this may be done by immediate bridges or half bridges. (However, use of half bridges implies more complex trust relationships, which some systems may not be able to support.) This allows a large range of security mechanisms to be used.

D.6.2 Secure Interoperability with DCE-CIOP Option

An ORB that conforms to this must conform to Standard Secure Interoperability using GIOP/IIOP as described in Section D.6.1, and also support secure interoperability using DCE-CIOP as defined in Section 15.8, Security and Interoperability.

The only security mechanism supported is DCE Security. Any version of DCE up to and including DCE 1.1 is supported; the DCE interfaces and protocols are specified in *X/Open Application Environment Specification for Distributed Computing*.

Appendix E Guidelines for a Trustworthy System

E.1 Introduction

This appendix provides some general guidelines for helping ORB implementors produce a trustworthy system. The intention is to have all information related to trustworthiness and assurance in this appendix, to explain how the specification has taken into account the requirements for assurance, and also to show how conformant implementations can have different levels of assurance.

The remainder of the introduction first provides the rationale for including these guidelines in the specification, and then gives some background on trustworthiness and assurance. Section E.2, Protecting Against Threats, describes the threats and countermeasures relevant to a CORBA security implementation. Sections E.3 through E.6 provide the architecture and implementation guidelines for each security object model described in Section 15.4, Security Architecture.

E.1.1 Purpose of Guidelines

The security standards proposed in this specification have been deliberately chosen to allow flexibility in the security features, which can be provided. The specification can support significantly different security policies and mechanisms for security functions such as access control, audit and authentication. However, there is an overall security model which applies whatever the security policy. This is described in the earlier sections of the document.

There is also flexibility in the level of security assurance, which can be provided, conforming to this model and these standards. This appendix describes the trustworthiness issues underlying the security model and interfaces described earlier in the document, and provides implementation guidance on what components of the architecture need to be trusted and why. Note that trust requirements assume conformance to all of the security models, including the implementor's view, as the implementation affects trustworthiness. If a CORBA security implementation conforms to the security features replaceability level, but not the ORB services one, any requirements on ORB services will apply to the ORB. Trustworthiness will also depend on several other implementation choices, such as the particular security technology used.

E.1.2 Trustworthiness

Before an enterprise places valuable business assets within an IT system, enterprise management must decide whether the assets will be adequately protected by the system. Management must be convinced that the particular system configuration is sufficiently *trustworthy* to meet the security needs of the enterprise environment. Security trustworthiness is thus the ability of a system to protect resources from exposure to misuse through malicious or inadvertent means.

The basis for trust in distributed systems differs from host-centric stand-alone systems largely for two reasons. First, the assignment of trust in a distributed system is not isolated to a single global system mechanism. Second, the degree of trust in elements of distributed systems (particularly distributed *object* systems) may change dynamically over time, whereas in host-centric systems trustworthiness is typically static. In many cases, trust in distributed systems must be seen in the context of mutual suspicion.

E.1.3 Assurance

Assurance is a qualitative measure of trustworthiness; assurance is the confidence that a system meets enterprise security needs. The qualitative nature of assurance means that enterprises may have different assurance guidelines for an equivalent level of confidence in security. Some organizations may need extensive evaluation criteria, while other organizations need very little evidence of trustworthiness.

It is necessary to set a context by which CORBA developers and end-users of the CORBA Security specification may evaluate the level of security to meet their needs. A single overall trust model that underlies the security reference model and architecture (as described elsewhere in this specification) can set this context for closed systems, but it is unlikely that a single trust model exists for the diversity of open distributed systems likely to populate the distributed object technology world.

To support a balanced approach, assurance arguments should be assembled from a set of system building blocks. Concepts of system composition and integration should allow the assurance analysis to be tailored to specific user requirements. Assurance evidence should be carefully packaged to best support enterprise decision-makers during the security trade-off process.

The security object models defined by the CORBA Security specification are the basis for the necessary building blocks. The trust guidelines described in Section, Guidelines for Structural Model, provide constraints on how these components may relate.

The relationship between assurance and security provides the foundation for the overall security model. The key characteristic is balance. Balanced assurance promotes the use of assurance arguments and evidence appropriate to the level of risk in the system components.

Basic system building blocks, such as those in the CORBA Security specification previously noted, are critical to developing balanced assurance. For example, confidentiality is of most importance to a classified intelligence or military system, whereas data integrity may be of more importance in a computer patient record system. The former relies on assurance in the underlying operating system, where the latter focuses security in application software.

E.2 Protecting Against Threats

An enterprise needs to protect its assets against perceived threats using appropriate security measures. This document addresses security in distributed object systems, so focuses on the threats to assets, software, and data, in such systems.

An enterprise may want to assess the risk of a security breach occurring, against the damage which will be done if it does occur. The enterprise can then decide the best trade-off between the cost of providing protection from such threats and any performance degradation this causes, against the probability of loss of assets. This specification allows options in how security is provided to counter the threats. However, it is expected that many enterprises will not undertake a formal risk assessment, but rely on a standard level of protection for most of their assets, as identified by industry or government criteria. This section describes CORBA-specific security goals, the main distributed system threats, and protection against them. The discussion does not emphasize generic issues of threats and countermeasures, but instead concentrates on issues that are unique to the CORBA security architecture.

E.2.1 Goals of CORBA Security

The overall goals of the CORBA security architecture were described in Section 15.1, Introduction to Security. CORBA security is based on the four fundamental objectives of any secure system:

- Maintain confidentiality of data and/or system resources.
- Preserve data and/or system integrity.
- Maintain accountability.
- Assure data/system availability.

Many of the goals described in Section 15.1 are relevant to any IT system that is targeted at large-scale applications. However, some security goals described are specific to the CORBA security architecture. These goals deserve special attention because they surface potential threats that may not be encountered in typical architectures. CORBA-specific security goals include:

- Providing security across a heterogeneous system where different vendors may supply different ORBs.
- Providing purely object-oriented security interfaces.
- Using encapsulation to promote system integrity and to hide the complexity of security mechanisms under simple interfaces.
- Allowing polymorphic implementations of objects based on different underlying mechanisms.
- Ensuring object invocations are protected as required by the security policy.
- Ensuring that the required access control and auditing is performed on object invocation.

The discussion of the architecture and implementation guidelines in Section E.3, Guidelines for Structural Model, addresses the mechanisms used to ensure these CORBA-specific security goals, as well as many other generic security issues.

E.2.2 Threats

The CORBA security model needs to take into account all potential threats to a distributed object system. It must be possible to set a security policy and choose security services and mechanisms that can protect against the threats to the level required by a particular enterprise.

A security *threat* is a potential system misuse that could lead to a failure in achieving the system security goals previously described. Section 15.1, Introduction to Security, provided an overview of security threats in a distributed object system. These threats and related attacks include:

- **Information compromise** - the deliberate or accidental disclosure of confidential data (e.g., masquerading, spoofing, eavesdropping).
- **Integrity violations** - the malicious or inadvertent modification or destruction of data or system resources (e.g., trapdoor, virus).
- **Denial of service** - the curtailment or removal of system resources from authorized users (e.g., network flooding).
- **Repudiation of some action** - failure to verify the actual identity of an authorized user and to provide a method for recording the fact (e.g., audit modification).
- **Malicious or inadvertent misuse** - active or passive bypassing of controls by either authorized or unauthorized users (e.g., browsing, inference, harassment).

The threats described above give rise to a wide variety of attacks. Most if not all the threats that pertain to host-centric systems are pertinent to distributed systems. Furthermore, it appears likely that the wide distribution of resources and mediation in truly distributed systems will not only exacerbate the strain on host-centric security services and mechanisms in use today on client/server systems, but also engender new forms of threat.

Threats may be of different strengths. For example, accidental misuse of a system is easier to protect against than malicious attacks by a skilled hacker. This specification does not attempt to counter all threats to a distributed system. Those that should be countered by measures outside the scope of this specification include:

- Denial of service, which may be caused by flooding the communications with traffic. It is assumed that the underlying communications software deals with this threat.
- Traffic analysis.
- Inclusion of rogue code in the system, which gives access to sensitive information. (This affects the build and change control process.)

E.2.3 Vulnerabilities of Distributed Object-Oriented Systems

Vulnerabilities are system weaknesses that leave the system open to one or more of the threats previously described. Information systems are subject to a wide range of vulnerabilities, a number of which are compounded in distributed systems. These vulnerabilities often result from deliberate or unintentional trade-offs made in system design and implementation, usually to achieve other more desirable goals such as increased performance or additional functionality.

Classes of vulnerabilities include:

- An authorized user of the system gaining access to some information which should be hidden from that user, but has not been properly protected (e.g., access controls have not been properly set up or the store occupied by one object has not been cleared out when another reuses the space).
- A user masquerading as someone else, and so obtaining access to whatever that user is authorized to do, resulting in actions being attributed to the wrong person. In a distributed system, a user may delegate his rights to other objects, so they can act on his behalf. This adds the threat of rights being delegated too widely, again, causing a threat of unauthorized access.
- Controls that enforce security being bypassed.
- Eavesdropping on a communication line giving access to confidential data.
- Tampering with communication between objects: modifying, inserting, and deleting items.
- Lack of accountability due, for example, to inadequate identification of users.

System data as well as business data must be protected. For example:

- If a principal's credentials are successfully obtained by an unauthorized user, they could be used to masquerade as that principal.
- If the security sensitive information in the security context between client and target object is available to an unauthorized user, confidential messages can be read, and it may be possible to modify and resend integrity-protected messages or send false messages without this being detected.

As described earlier, system threats and vulnerabilities are compounded by the complexities of distributed object-based systems. Some of the inherent characteristics of distributed object systems that make them particularly vulnerable include:

- **Dynamic Systems** -- Distributed object systems are always changing. New components are constantly being added, deleted, and modified. Security policies also may be dynamically modified as enterprises change. Dynamic systems are inherently complex, and thus security may be difficult to ensure. For example, in a large distributed object system it will be difficult to update a security policy atomically. While an administrator installs a new policy on some parts of the system, other parts of the system still may be using the old version of the policy. These potential inconsistencies in policy enforcement could lead to a security failure.
- **Mutual Suspicion** -- In a large distributed system, some system components will not trust others. Mistrust could occur at many layers within the architecture: principals, objects, administrators, ORBs, and operating systems may all have varying degrees of trustworthiness. In this environment, there is always the

potential to inadvertently place unjustified trust in some system component, thus exposing a vulnerability. Although there are many mechanisms (e.g., cryptographic authentication) to ensure the identity of a remote component, the system security architecture must be carefully structured to ensure that these checks are always performed.

- **Multiple Policy Domains** -- Distributed object systems that interconnect many enterprises are likely to require many different security policy domains, each one enforcing the security requirements of its organization. There is no single security policy and enforcement mechanism that is appropriate for all businesses. As a result, security policies must be able to address interactions across policy domain boundaries. Defining the appropriate policies to enforce across domains may be a difficult job. Mismatched policies could lead to vulnerabilities.
- **Layering of Security Mechanisms** -- Distributed object systems are highly layered, and the security mechanisms for those systems will be layered as well. Complex, potentially nondeterministic interactions at the boundary of the layers is another area for vulnerabilities to occur. A hardware error, for example, could cause security checking code in the ORB to be bypassed, thus violating the policy. The complexity of the layering is further compounded in systems where security enforcement is widely distributed; that is, there is no clear security perimeter containing only a small amount of simple functionality.
- **Complex Administration** -- Finally, large geographically distributed object systems may be difficult to administer. Security administration requires the cooperation of all the administrators, who even may be mutually suspicious. All of the issues listed above lead to complex, error-prone administration. An innocent change to a principal’s access rights, for example, could expose a serious vulnerability.

E.2.4 Countermeasures

Some threats are common across most distributed secure systems, so should be countered by standard security features of any OMA-compliant secure systems. However, the level of protection against these threats may vary. Complete protection is almost impossible to achieve. Most enterprises will want a balance between a level of protection against threats which are important to them, and the cost in performance and use of other resources of providing that level of protection.

A number of measures exist for countering or mitigating the effects of the above threats/attacks. Countering these threats requires the use of the security object models described in this specification. Relevant features of the object models include the following:

- Authentication of principals proves who they are, so it is possible to check what they should be able to do. This check can be performed at both client and target object, as the client principal’s credentials can be passed to the server.
- Authentication between clients and target objects allows them to check that they are communicating with the right entities.

- Security associations can protect the integrity of the security information in transit between client and target object (e.g., credentials, keys) to prevent theft and replay, and keep the keys used for protecting business data confidential.
- Business data can be integrity-protected in transit so any tampering is detected using the message protection ORB services. (This includes detecting extra or missing messages, and messages out of sequence.)
- Unauthorized access to objects is protected using access controls.
- Misuse of the system can be detected using auditing.
- Segregating (groups of) applications from each other and security services from applications can prevent unauthorized access between them.
- Bypassing of security controls is deterred by use of a Trusted Computing Base (TCB), where security is automatically enforced during object invocation.

Assurance arguments and evidence are frequently founded on the concept of a TCB, which mediates security by segregating the security-relevant functions into a security kernel or reference monitor.

A traditional monolithic TCB approach is not suitable for the open, multiuser, multiple environment situations in which most CORBA users reside. In many cases, for example, secure interoperability of CORBA applications and ORBs may be based on mutual suspicion. TCB scalability issues also argue against typical TCB approaches. Given the complexity of distributed systems, it is not clear whether centralized access mediation is possible in the presence of distributed data and program logic.

Traditional TCB approaches also do not adequately address application security requirements, particularly for many commercial applications. Applications common to the CORBA world such as general purpose DBMSs, financial accounting, electronic commerce, or horizontal common facilities will have many security requirements in addition to those that can be enforced by a central underlying TCB.

Despite the limitations of the traditional TCB, we use the concept of a *distributed TCB* in the assurance discussions of the next section. The concept of a distributed TCB is the collection of objects and mechanisms that must be trusted so that end-to-end security between client and target object is maintained. However, note that depending on the assurance requirements of a particular CORBA security architecture, sensitive data may still be handled by "untrusted" ORB code. Thus, our informal use of the distributed TCB concept may not correspond to other existing models for network TCBs, particularly for minimal assurance commercial CORBA security applications.

E.3 Guidelines for Structural Model

This section provides architecture and implementation guidelines for the structural model of the CORBA security architecture described in Section 15.4, Security Architecture. The security functions provided in the model and the basis for trust are described.

E.3.1 Security Functions

Figure E-1 outlines interactions during a normal use of the system. It gives a simple case, where the application is unaware of security except for calling a security service such as audit. The security interactions include those seen by application objects and secure object system implementors.

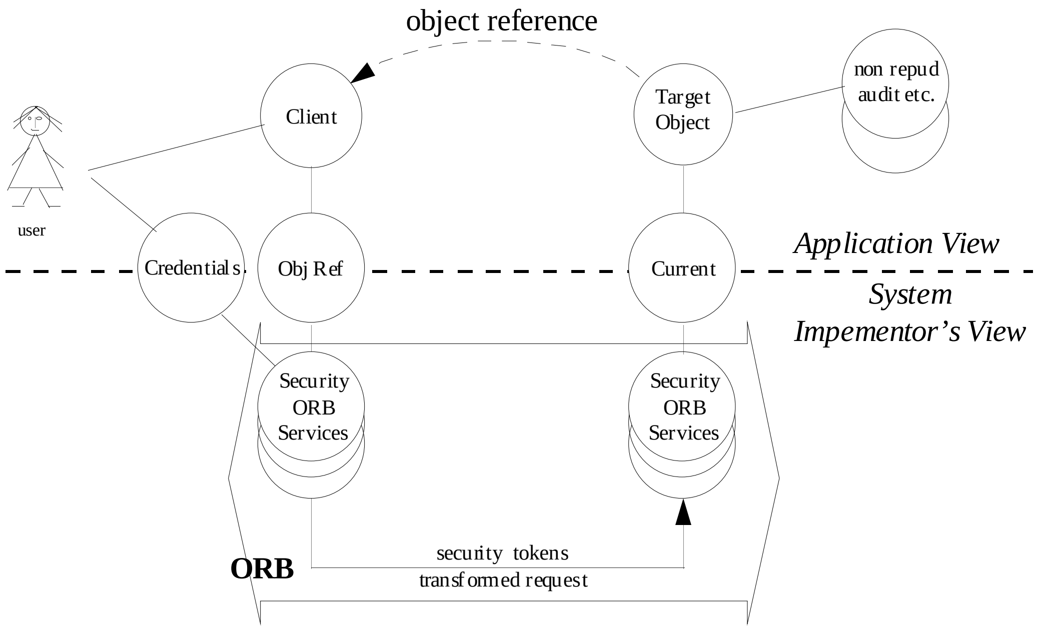

Figure E-1 Normal System Interactions

This diagram is the basis for the discussions of security functions in each of the security object models described next.

E.3.2 Basis of Trust

Enterprise management is responsible for setting the overall security policies and ensuring system enforcement of the policies.

The system developer and systems integrators must provide a system that supports the required level of assurance in the core security functionality. Generally application developers cannot be expected to be aware of all the threats to which the system will be subject, and to put the right countermeasures in place.

Higher levels of security may require the code enforcing it to be formally evaluated according to security criteria such as those of the US TCSEC or European ITSEC.

Distributed Trusted Computing Base

The key security functionality in the system is enforced transparently to the application objects so that it can be provided for application objects, which are security unaware. This key functionality is contained in the distributed TCB of the system. It is therefore responsible for ensuring that:

- Users cannot invoke objects unless they have been authenticated (unless the security policy supports unauthenticated, guest access for some services).
- Security policies on access control, audit, and security association are enforced on object invocation. This includes policies for message protection, both confidentiality (ensuring confidential data cannot be read) and integrity (ensuring any corruption of data in transit is detected).
- A principal's credentials are automatically transferred on object invocation if required, so the access control and other security policies can be enforced at the server object.
- Application objects which do not trust each other cannot interfere with each other.
- The security policy between different security policy domains is suitably mediated.
- The security mechanisms themselves cannot be tampered with.
- The security policy data cannot be changed except by authorized administrators.
- The system cannot be put into an undefined or insecure state as a result of the operation of nonprivileged code.

The distributed TCB also needs to provide the required information so that applications can enforce their own security policies in a way that is consistent with the domain security policy.

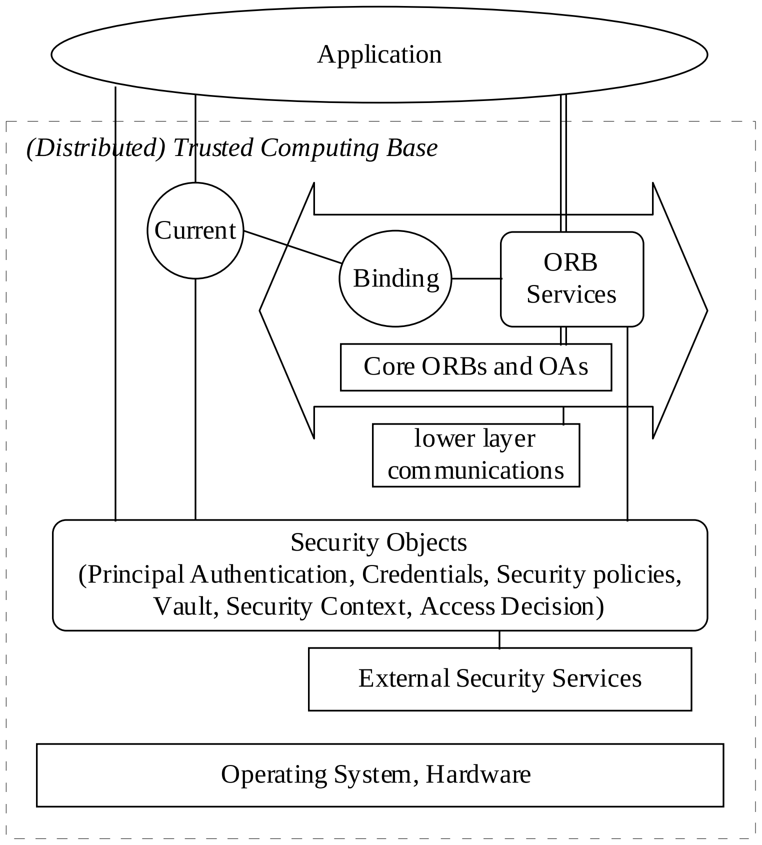
Application
(Distributed) Trusted Computing Base
Current
Binding
ORB Services
Core ORBs and OAs
lower layer communications
Security Objects (Principal Authentication, Credentials, Security policies, Vault, Security Context, Access Decision)
External Security Services
Operating System, Hardware

Figure E-2 Distributed TCB

The TCB in an OMA-compliant secure system is normally distributed and includes components as follows.

- The distributed core ORBs and associated Object Adapters
  Core ORBs are trusted to function correctly and call the ORB Security Services correctly in the right order, but do not need to understand what these do.
  Object Adapters are trusted to utilize the operating system facilities to provide the required protection boundaries between components in line with the security policy.
- The associated ORB Services
  ORB Services other than security are trusted similarly to the ORB. ORB Security Services are used to provide the required security on object invocation.
- Related objects
  ORB Services use objects such as the binding and Current to find which security is required.
- Security objects
  Security objects include those available to applications such as Principal Authentication and Credentials and those called by security interceptors (Vault,

Security Context, Access Decision, and Security Audit). These are trusted to function correctly to enforce security in line with the security policy and other requirements.

- Any external security services used by the security services, as part of enforcing the security policy.
- The supporting operating systems.
  These are trusted to ensure that objects (in different trust domains) cannot interfere with each other (using protection domains). The security services should also ensure that the security information driving the security policy (such as the credentials and security contexts) is adequately protected from the application objects using such features.
- Optionally, lower layer communications software. However, this does not generally need to be particularly secure (at least for normal commercial security) as protection of data in transit is done by the security association and message protection interceptors, which are independent of the underlying communication software.

A distributed system may be split into domains, which have different security policies. These domains may include ORBs and ORB Services with different levels of trust. Trust between domains needs to be established, and an interdomain policy between them enforced. The ORB security services (and external security services that these call) to provide this interdomain working are part of the distributed TCB. Note, therefore, that the parts of this TCB in different domains may have different levels of trust.

Note that application objects may enforce their own security polices, if these are consistent with the policy of the security domain. However, failure to enforce these securely will affect only the applications concerned and any other application objects that trusted them to perform this function.

Protection Boundaries

The general approach is to establish **protection boundaries** around groups of one or more components, which are said to belong to a corresponding **protection domain**. Components belonging to a protection domain are assumed to trust each other, and interactions between them need not be protected from each other, whereas interactions across boundaries may be subject to controls. Protection Boundaries and Domains are a lower level concept than Environment Domains; they are the fundamental protection mechanism on which higher levels are built.

At a minimum, it must be possible to create protection boundaries between:

- Application components that do not trust each other.
- Components that support security services and other components.
- Components that support security services and each other.

Controlled Communications

As well as providing protection boundaries, it is necessary to provide a controlled means of allowing particular components to interact across protection boundaries (for example, an application invoking a Security Object (explicitly), or an interceptor (implicitly).

It must not be possible for applications to bypass security services which enforce security policies. It is therefore necessary to ensure that the components supporting those services are always invoked when required. This is achieved by using both protection boundaries and controlled communications to ensure that client requests (and server responses) are routed via the components (interceptors and Security Objects), which implement the security services.

Figure E-3 illustrates the segregation of components implementing security services into separate protection domains from application components; the only means of communication between components is via controlled communication paths.

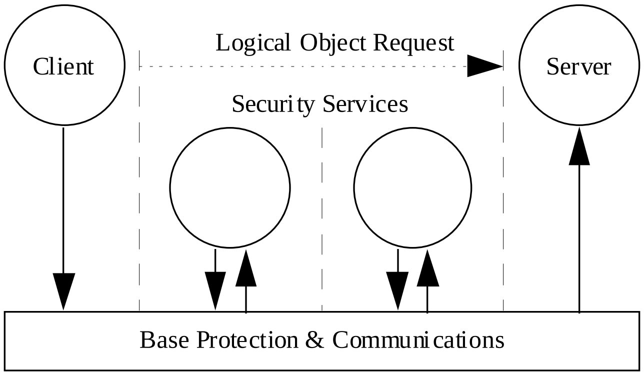

Figure E-3 Base Protection and Communications

In implementation terms, components could, for example, be executed in separate processes, with process boundaries acting as protection boundaries. Alternatively, security services could be executed in-process with (i.e. in the same address space as) corresponding client and server application components, provided that they are adequately protected from each other -- for example, by hardware-supported multilevel access control mechanisms).

Figure E-4 shows two examples of protection boundaries. In the first example, the boundaries between components might be process boundaries. In the second example, ORB and security components might be protected from applications by memory protection mechanisms (e.g. kernel and user spaces) and client and server components might be protected from each other by physical separation.

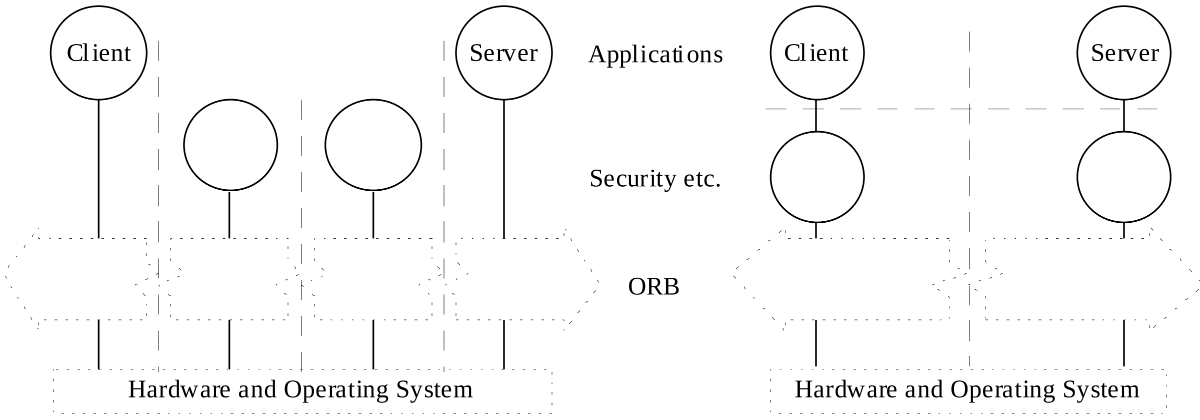

Figure E-4 Protection Boundaries

E.3.3 Construction Options

For some systems, the TCB in domains of the distributed system may need to meet security evaluation criteria for both functionality and assurance (in the correctness and effectiveness of the security functionality) as defined in TCSEC, ITSEC, or other security evaluation criteria.

The split into components previously described allows a choice over the way the system is constructed to meet different requirements for assurance and performance.

This section describes three options for how the system may be constructed, as follows:

- A commercial system where all applications are generated using trusted tools.
- A commercial system with limited security requirements.
- A higher security system.

Note: These are just examples to show the type of flexibility provided by the security model. It is not expected that any implementation will provide all the options implied by these.

Example Using Trusted Generation Tools and ORBs

If all applications are generated using trusted tools, applications can be trusted not to interfere with other components in the same environment. Therefore there is no need to provide protection boundaries between different application objects or between application objects and the underlying ORB.

If the ORB and ORB Services are also trusted, there may need be no need to provide a protection boundary between the ORB and the underlying security services and objects. It may well be acceptable to run them all in the same process, relying on the trust between the components, rather than more rigidly enforced boundaries.

However, if the application generation tools and the ORB are less trusted than the security services, then there may need to be a protection boundary to prevent access to security-sensitive information in the Credentials, Security Context, and Vault objects.

Commercial System with Limited Security Requirements

Some systems may not contain very sensitive business information, so enterprises may not be prepared to pay for a high level of security. They may also know that the probability of serious malicious attempts to break the system is low, and decide that protecting against such attempts is not worth the cost. They may also choose not to sacrifice performance for better levels of security.

In many systems, applications are generated using tools that are not particularly trusted. For example, using a C compiler, it would be possible to write an application that can read, or even alter, any information within the same protection domain. Theoretically, providing good security implies putting protection boundaries between each application object, and between applications and the ORB and Security Services.

The security model allows environment domains to be defined, where enforcement of policy can be achieved by means local to the environment. For example, objects in the same identity domain can share a security identity. Applications belonging to environment domains may trust each other not to interfere with each other, and so can be put in the same protection domain.

It may also be acceptable to run (part of) the ORB in the same protection domain as the application objects. This assumes that an interface boundary between applications and the ORB is sufficient protection from accidental damage (the probability of an application corrupting an ORB being low in a commercial system). Even if the application does corrupt the ORB, damage is limited, as the ORB does not handle security-sensitive data.

In some commercial systems, it may also be acceptable to run some of the security services in the same protection domain as the application and ORB. The chance of these being accidentally (or maliciously) corrupted may be low, so it may be acceptable to risk a failure to enforce the access control policy because the Access Decision object is corrupt.

However, it will often be desirable to protect the state information of security objects, which contain very sensitive security information from the applications.

Higher Security System

In a security system requiring high assurance, different security policies may be used. For example, label-based access controls may be used and these may be mandatory (set under administrator's controls) and not changeable by application objects.

Stronger protection boundaries are also likely to be needed, allowing:

- Individual applications to be protected from each other. Even if environment domains are used, the size of the domain is likely to be smaller.
- The ORB and ORB Services to be protected from the application.

- The core security objects, which contain security-sensitive information such as keys to be protected from applications and ORBs, etc.
- Particular secure objects (e.g. the Access Decision objects) to be separate from others, as they may have been written by someone less trusted than those who wrote, for example, the Security Context objects.

E.3.4 Integrity of Identities (Trojan Horse Protection)

In traditional procedural systems, protecting the integrity of an identity is straightforward; programs are stored in files, which are protected against modification by operating system access control mechanisms. When invoked, programs run inside a process whose address space is protected by operating system memory protection mechanisms. Programs load code in fairly predictable ways.

Since this specification does not mandate which entities have identities, implementors have a wide variety of choices; identities may be associated, for example, with the following:

- Object instances
- Servers
- Object adaptors
- Address spaces

If identities are associated with object instances, precautions are necessary to prevent object instance code from being modified by other code (which may have no identity, or a different identity) in the instance's address space.

Servers may permit dynamic instantiation of previously unknown classes into their address spaces. This makes it difficult to determine what code is running under an identity if identities are associated with servers; this in turn makes it difficult to determine whether a server identity can be "trusted." Identified servers must therefore be provided with some way of controlling what code can run under their identities.

Observing the following guidelines will help to ensure integrity of identities.

- Code running under one identity must not be permitted to modify code running under another identity without passing an authorization check.
- It must be possible for an identified "entity" to control which code runs within the scope of its identity.

E.4 Guidelines for Application Interface Model

This section provides architecture and implementation guidelines for the application interface model of the CORBA security architecture described in Section 15.4, Security Architecture. The security functions provided in the model and the basis for trust are described.

E.4.1 Security Functions

Logging onto the System

When a user or other principal wants to use a secure object system, it authenticates itself and obtains credentials. These contain its certified identity and (optionally) privilege attributes, and also controls where and when they can be used. This principal information is integrity-protected and it should be possible to ascertain what security service certified them.

Walkthrough of Secure Object Invocation

The following is a walkthrough of what happens when a client invokes a target object.

- The client invokes the object using its object reference. The ORB Security Services are transparent to the client and application object and use the security information with the object reference and the security policy to decide on the security facilities required. There are separate ORB Services for security associations, message protection, and access control on object invocation, but the audit service can be called by any or none of these according to security policy.

  The client and target object establish the required level of trust in each other, transmitting security tokens to each other to provide the required degree of proof. For example, they may or may not require mutual authentication. It is expected that most security mechanisms will provide options here, though the details of how they do this, and the form of tokens used, is mechanism dependent.

  The principal's credentials are normally passed from client to target object transparently. These should be protected in transit from theft and replay as well as for integrity of the information itself (though some security mechanisms may not support this). The Vault object will validate these, checking that it trusts who certified them, as well as whether they are still intact.

  Different ORB services may be called at the target end. For example, access control is normally called at the server, rather than the client.
- Once the security association has been established between client and target object, the request can be passed using the message protection interceptor to protect it. This should be able to provide integrity and/or confidentiality protection. It should also be able to provide continuous authentication, as the messages will be protected using keys only known to this client and server (or the trust group for the target object).
- The application object may also call security services for access control and audit. These will use the security information available from the environment to identify the initiating principal and its privileges.

- This application object may now act as a client, and call further objects. It may delegate the client's credentials or use its own (or use both). However, there may be constraints on whether the client's credentials can be delegated. For example, a particular principal's credentials may be constrained to particular groups of objects.

E.4.2 Basis of Trust

Users have some trust in application objects, and application objects have some trust in other objects. Both may:

- Trust application objects to perform the business functions.
- Have limited trust in some applications, or domains of the distributed system, so restrict which of their privilege attributes are available to these objects.
- Want to restrict the extent that their credentials can be propagated at all.
- Have to prove their identity to the system so it can enforce access on their behalf, unless they are only going to access publicly available services.

Both users and applications trust the underlying system to enforce the system security policy, and therefore protect their information from unauthorized access and corruption.

E.5 Guidelines for Administration Model

This section provides architecture and implementation guidelines for the administration model of the CORBA security architecture described in Section 15.4, Security Architecture. The security functions provided in the model and the basis for trust are described.

E.5.1 Security Functions

Object and Object Reference Creation

When an object is created in a secure object system, the security attributes associated with it depend on the security policies associated with its domain and object type, though the object may be permitted to change some of these. These attributes control what security is enforced on object invocation (or example, whether access control is needed and, if so, the Access Decision object to be used; the minimum quality of protection required).

The object reference for a such an object is extended to include some security information. For example, it may contain:

- An extended identity. This includes the object identity as normal in an object reference. However, it will also contain the identity of the trust domain, if the object belongs to one. Small objects, which are dynamically created and do not need to be protected from each other, will normally share a trust domain. There could also be a node identity.

- Security policy attributes required by the object when invoked by a client such as the minimum quality of protection of data in transit.
- The security technology it supports. It may also contain some mechanism-specific information such as its public key, if public key technology is being used, and particular algorithms used.

Much of the information is just “hints” about which security is required, and will be verified by the ORB services supporting the target object, so does not need protecting.

E.5.2 Basis of Trust

Authorization Policy Information

Domain objects may store policy information inside their own encapsulation boundaries, or they may store it elsewhere (for example, authorization policy information could be encapsulated in the state data of the protected objects themselves, or it could be stored in a procedural Access Control Manager whose interfaces are accessible to Domain objects). Wherever authorization policy information is stored, it must be protected against modification by unauthorized users.

Authorization policy information must be modifiable only by authorized administrators.

Audit Policy Information and Audit Logs

Audit policy information is security sensitive and must be protected against unauthorized modification. Audit logs are security sensitive and may contain private information; they should be viewed and changed only by authorized auditors.

- Audit policy information must be modifiable only by authorized audit administrators.
- Audit logs must be protected against unauthorized examination and modification.

E.6 Guidelines for Security Object Implementation Model

This section provides architecture and implementation guidelines for the security object implementation model of the CORBA security architecture described in Section 15.4, Security Architecture. The security functions provided in the model and the basis for trust are described.

E.6.1 Security Functions

The distributed core ORBs, object adapters, ORB security services, and security objects provide the underlying implementation to support the application and administration interfaces.

E.6.2 Basis of Trust

Target Object Identities

CORBA objects do not have unique identities; for this reason, when objects that are not associated with a human user authenticate themselves in a secure CORBA system, they use "security names." Successful authentication to a target object indicates that it possesses the authentication data (perhaps a cryptographic key), which is presumed to be known only to the legitimate owner of the security name. An object's security name may be included in references to that object as a "hint." The question "how do applications know that the security-name hint is reliable?" naturally arises.

The answer is as follows:

- If the EstablishTrustinTarget security feature is specified, then the security services defined in this specification will authenticate the target security name found in the target object reference. The semantics of this authentication operation include an assumption that the security name in the reference corresponds to an identity that the user is willing to trust to provide the target object's implementation. There is no way for the security services to test this assumption.
- If your implementation provides a trusted source of object references, then everything will work properly. If you do not have a source of trusted object references, the specification provides a **`get_security_names`** operation on the object reference through which applications can retrieve the target's security name and perform any tests, which may help satisfy them of its validity.

CORBA object references can circulate very widely; for example, they can be "stringified" and then (potentially) copied onto a piece of paper. Implementations with very high integrity requirements could ensure that references are trustworthy by providing a trustworthy service that generates references and cryptographically signs the contents, including the target security name.

Assumptions about Security Association Mechanisms

Implementation of a secure CORBA system requires use of security mechanisms to enforce the security with the required degree of protection against the threats. For example, cryptographic keys are normally used in implementing security, for functions such as authenticating users and protecting data in transit between objects. However, different security mechanisms may use different types of cryptographic technology (e.g. secret or public key) and may use it in different ways when, for example, protecting data in transit. These cryptographic keys have to be managed, and again, the way this is done is mechanism specific.

A full analysis of how well an implementation counters the threats requires knowledge of the security mechanisms used. However, this specification does not dictate that a particular mechanism is used.

It does assume that the security mechanisms used for authentication and security associations can provide the relevant security countermeasures listed in Section E.2.4, Countermeasures. These are expected to be provided by a number of security mechanisms, which will be available for protecting secure object systems. Therefore, the analysis of threats and the trust model assume this facility level.

It would be possible to use a security mechanism that does not provide some of these facilities (for example, mutual authentication, or even to switch this off to improve performance in systems that can provide it). However, if such a system is used, it will be vulnerable to more threats.

Invoking Special Objects

Some of the objects described in this document are "pseudo" objects, which bypass the normal invocation process and therefore are not subject to the security enforced by the ORB services. The *Current* object (used, for example, by the target object to obtain security information about the client) is of this type. Protection of these objects is provided by other means, for example, using protection boundaries previously described.

Isolating Security Mechanisms

Figure E-5 depicts how security functionality and trust is distributed throughout the architecture.

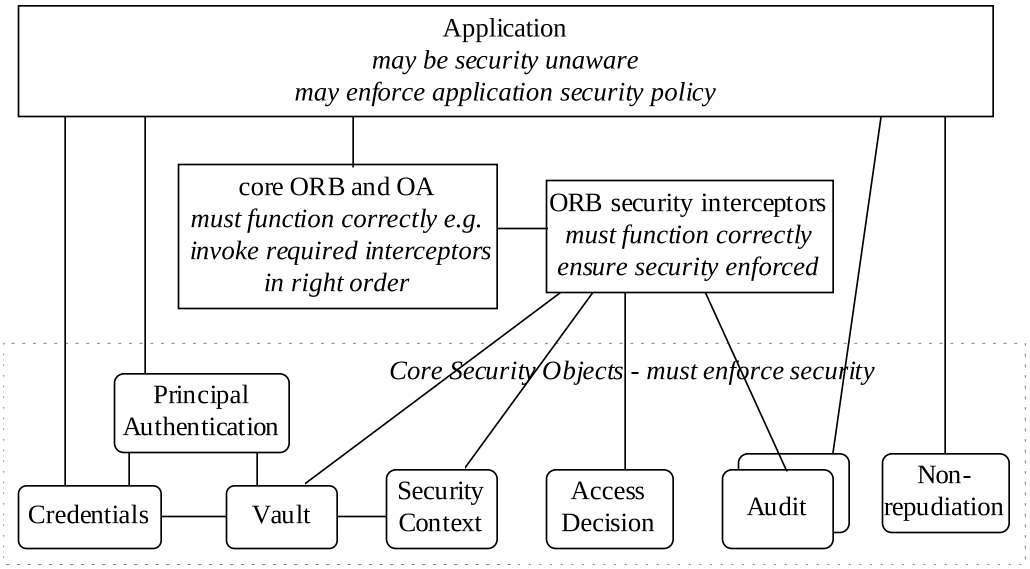

Figure E-5 Distribution of Security Functionality and Trust

The split of security objects is designed to reduce (as much as possible) the amount of security-sensitive information, which must be visible to applications and ORBs.

- Only log-in applications (where provided) need to handle secrets such as passwords, and then only briefly during authentication.

- Cryptographic keys and other security-sensitive information about principals are held with Credentials objects. References to Credentials objects are visible to applications so they can invoke operations on them to, for example, reduce privileges in the credentials before calling an object. However, no operations on the Credentials provide visibility of security information such as keys.
- Security information used to protect application data in transit between objects is held in Security Context objects, which are not visible to applications at all. (Target applications can ask for attributes associated with an incoming invocation using the Current object.)

Security objects such as Credentials, Security Context, and Access Decision objects are also not used directly by the core ORB, only by the security interceptors. Therefore the core ORB needs to be trusted to call the interceptors correctly in the right order, but does not need to understand security or have access to the security-sensitive information in them.

The split also is intended to isolate components which may be replaced to change security policy or security mechanisms. For example, to replace the access control policy, the Access Decision objects need to be changed. However, the access control interceptor will remain responsible for finding and invoking the right Access Decision object. To replace the security mechanisms for security association, only the Vault and associated Security Context objects need to be replaced.

Integrity of the ORB and Security Service Objects

Security in a CORBA environment depends on the correct operation of the ORB and Security Services. In order for these mechanisms to operate correctly, the following rules must be followed.

- The ORB and Vault code must not be modifiable by unauthorized users or processes.
- The ORB must protect all messages, according to policy, using the message protection interfaces.
- The ORB must always check the client's authorization before dispatching a client's message to a protected object.

Safeguarding the Object Environment

To guard against unauthorized modification of the ORB and security services, implementors should use Operating System protection mechanisms to isolate the ORB and Security Service objects from untrusted applications and user code.

Note that some modifications of ORB or Vault code may not compromise system integrity. For example, in a CORBA implementation, which relies on third-party authentication and does not share Vault or ORB objects between processes, corruption of the client-side Vault (or ORB) by user-written code may not compromise system security. (This is because the client-side ORB and Vault in a third-party-based system may, depending upon the implementation, contain only information that the user is entitled to know and change anyway. In this case, nothing the user can do to information on his machine will enable him to deceive the third-party authentication server about his identity and credentials.)

Safeguarding the Dispatching Mechanism

To ensure that the ORB always checks the client's authorization before dispatching a client's message to a protected object, ORB implementors should follow one of the following rules.

- Eliminate "direct dispatching" mechanisms (which permit clients to dispatch messages directly to target objects without going through the ORB).
- Permit "direct dispatching" only after checking authorization and issuing "restricted object references" to client objects. A "restricted object reference" is one that grants access only to those methods of the target object, which the client is authorized to invoke.

Safeguarding Information in Shared Vault Objects

Vault objects encapsulate identity-specific, security-sensitive information (for example, cryptographic keys associated with Security Context objects). If code owned by one principal can penetrate a Vault object and examine or modify another principal's information, security can be compromised.

In an implementation that does not permit sharing of Vault objects by multiple identities, this problem does not arise. However, if Vault objects are accessible to and encapsulate information about multiple identities, the following guidelines should be observed:

- Do not permit a Vault object, which encapsulates one principal's Security Contexts, to exist in the same address space as code running under a different principal's identity.
- If a Vault object contains Security Contexts for two different principals, ensure that no principal is able to obtain or use another principal's Security Contexts.

Appendix F Conformance Statement

F.1 Introduction

A secure object system, like any secure system, should not only provide security functionality, but should also provide some assurance of the correctness and effectiveness of that functionality.

Each OMG-compliant secure or security ready implementation must therefore include in its documentation a conformance statement describing:

- The product's supported security functionality levels and options, security replaceability, and security interoperability, as described in Appendix D, Conformance Details.
- The vendor's assurance argument that demonstrates how effectively the product provides its specified security functionality and security policies.
- Constraints on the use of the product to ensure security conformance.

The vendor provides the conformance statement so that a potential product user can make an informed decision on whether a product is appropriate for a particular application. Ordinary descriptive documentation is not required as part of an OMG-compliant product. However, because the CORBA security specification provides a general security framework rather than a single model, there are many different kinds of secure ORB implementations that conform to the framework. For example, some systems may have greater flexibility and support customized security policies, while other systems may come with a single built-in policy. Some systems may strive for a high level of security assurance, while others provide minimal assurance. The conformance statement will help the user understand the security features provided by the product.

Some products will undergo an independent formal security evaluation (such as ones meeting the ITSEC or TCSEC). The OMG security conformance statement does not take the place of a formal evaluation, but may refer to formal assurance documentation, if it exists. When formal evaluations are not required (often the case in commercial systems), it is expected that the product's security conformance statement along with supporting product documentation will provide an adequate description of security functionality and assurance.

F.2 Conformance Template Overview

The following template specifies the contents for CORBA security conformance statements. Guidelines for using this template are provided in Section, Conformance Guidelines.

CORBA Security Conformance Statement

<date>
<product identification>
<vendor identification>

1. Introduction

1.1 Summary of Security Conformance

1.2 Scope of Product

1.3 Security Overview

2. Security Conformance

2.1 Main Security Functionality Level

2.2 Security Functionality Options

2.3 Security Replaceability

2.4 Secure Interoperability

3. Assurance

3.1 Philosophy of Protection

3.2 Threats

3.3 Security Policies

3.4 Security Protection Mechanisms

3.5 Environmental Support

3.6 Configuration Constraints

3.7 Security Policy Extensions

4. Supplemental Product Information

F.3 Conformance Guidelines

The guidelines in this section are intended to help the ORB implementor determine which information belongs in each section of the conformance statement. The statement will often be accompanied by product documentation to provide some of the information needed.

1. Introduction

1.1 Summary of Security Conformance

This section should give a summary of the security conformance provided by the product. The summary is in the form of a table with boxes that are ticked to show the relevant conformance.

Main Functionality Level		**Functional Options**	**Security Replaceability**				**Security Interoperability**	
1	2	Non-repudiation	ORB Services	Security Services	Security Ready - ORB Services	Security Ready - Security Services	Standard	Standard + DCE-CIOP

For the main security functionality level, one of the boxes must be selected (either Level 1 or Level 2), though note that an ORB can be just Security Ready, so does not support either of the main security functionality levels. For security functionality options, security replaceability, and secure interoperability, the appropriate boxes should be selected.

1.2 Scope of Product

This section should define what security components this product offers. Examples are:

- ORB plus all security services needed to support it plus other object services fitting with it and meeting the assurance criteria.
- Security-ready ORB.
- Security Services, which can be used with a security-ready ORB.

1.3 Security Overview

This section should give an overview of the product's security features.

2. Security Conformance

2.1 Main Security Functionality Level

This section should define which main security functionality level this product supports, Level 1 or Level 2.

This should also include any qualifications on that support. For example, any interpretation of the CORBA security specification and how it is supported, any bells and whistles around the published interfaces, and any limitations on support for this level.

As in the conformance level descriptions, the description should be divided into:

- The security functionality provided by the product
- The application developer's interfaces
- The administrative interfaces

2.2 Security Functionality Options

This section should define which functionality options are provided, in particular the support for non-repudiation.

For non-repudiation, as this is a published interface in this specification, it should be accompanied by a qualification statement if needed, as for the main security functionality level.

2.3 Security Replaceability

This section should define whether the product supports replaceability of security services, ORB services, or neither.

This should also include any qualifications on that support. For example, any interpretation of the CORBA security specification and how it is supported, any bells and whistles around the published interfaces, and any limitations on support for this conformance option.

2.4 Secure Interoperability

This section should define whether the product supports standard secure interoperability, standard and DCE-CIOP interoperability, or neither. As with the previous sections, qualifications of the support, interpretations of the CORBA specification, and limitations should be included as needed.

3. Assurance

If the product already has supporting assurance documentation (for example, because it is being formally evaluated), much of this section may be satisfied by references to such documentation. Appendix E, Guidelines for a Trustworthy System, provides general discussions of many of the topics described here, particularly the basis of trust needed for each of the architecture object models.

3.1 Philosophy of Protection

Overview of supported security policies, security mechanisms and supporting mechanisms.

3.2 Threats

Description of specific threats intended to be addressed by the system security policy, as well as those not addressed.

3.3 Security Policies

Description of any predefined policies, including

- Classes of entities (such as clients, objects) controlled by security policy
- Modes of access (conditions that allow active entities to access objects)
- Use of domains (policy, trust, technology)
- Requirements for authentication of principal, client and target objects
- Requirements for trusted path between principals, clients, ORBs, and target objects
- Delegation model
- Security of communications
- Accountability requirements (audit, non-repudiation)
- Environmental assumptions of the policy (e.g. classes of users, LAN/WAN, physical protection)

3.4 Security Protection Mechanisms

- Rationale for approach
- Identification of components, which must function properly for security policies to be enforced
- Description of mechanisms used to enforce security policy
- How protection mechanisms are distributed in the architecture
- Why security mechanisms (such as access control) are always invoked and tamper-proof

3.5 Environmental Support

- How the underlying environment (such as operating systems, generation tools, hardware, network services, time services, security technology) are used in providing assurance
- How installation tools ensure secure configuration
- How security management and administration maintains secure configuration

3.6 Configuration Constraints

Constraints to ensure that system security assurance is preserved, for example:

- Requirements on use and development of: clients, target objects, legacy software
- Limitations on interoperability
- Required software and hardware configuration

3.7 Security Policy Extensions

- Supported security policy extensions, if applicable
- Limitations of extensions
- Requirements imposed on developers to ensure trustworthiness of policy extensions
- Supported interactions and compositions of security policies

4. Supplemental Product Information

Supplemental product information is included at the vendor's discretion. It can be used to describe, for example:

- Additional security features, not covered by the CORBA Security specification
- The impact of security mechanisms on existing applications

Appendix G Facilities Not in This Specification

G.1 Introduction

Security in CORBA systems is a big subject, which affects many parts of the Object Management Architecture. It was therefore decided to phase the specification in line with the priorities agreed as part of the security evaluation criteria by the Security Working Group prior to the production of this specification.

This specification therefore includes the core security facilities and the security architecture to allow further facilities to be added. Priority has been given to those requirements most needed by commercial systems. Even with these limitations, the size of the specification is larger than desirable for OMG members to review easily or for vendors to implement.

Some of the facilities omitted from this specification are agreed to be required in some secure CORBA systems, and so are expected to be added later, using the usual OMG process of RFPs to request their specification.

This appendix lists those security facilities which are not included in the specification, but left to later specifications, which may be in response to further RFPs for Object Services or Common Facilities.

G.2 Interoperability Limitations between Unlike Domains

Secure interoperability is included in this specification. This allows applications running under different ORBs in different domains to interoperate providing that:

- Both support and can use the same security mechanisms (and algorithms, etc.) for authentication and secure associations (an ORB may support a choice of security mechanisms).
- Use of these between the domains will not contravene any government regulations on the use of cryptography.
- The security policies they support are consistent -- for example, use the same types for privileges which can be understood in both places.

Limitations in the specification which affect this type of interoperability are:

- The standard policies defined do not include specifying different policies when a client communicates with different domains (though it is possible to define specific policies to do this).
- There is no specification of the mapping policies required to translate attributes when crossing a domain boundary where these policies are inconsistent, and how these must be positioned, for example, to allow delegation of the mapped attributes. Again, such mapping policies are not prevented.
- In general, there is no specification of how federated policies are implemented.

- There is no specification of gateways to handle interoperability between security mechanisms. It is expected that only limited interoperability between particular security mechanisms will ever be provided, so this is not expected to be the subject of an RFP in the foreseeable future.

G.3 Nonsession-Oriented SECIOP Protocol

The SECIOP protocol defined in Section 15.8, Security and Interoperability, assumes that all underlying security mechanisms are session-oriented. The current specification does not support security mechanisms, which encapsulate key distribution and other security context management information in a single message along with the data being protected (examples of such mechanisms include those accessed through the proposed internet IDUP-GSS-API interface). Changes to the SECIOP protocol would be required to support non-session-oriented protocols.

G.4 Mandatory Security Mechanisms

The current specification does not mandate any particular security mechanism which all secure ORBs must implement. This is because the submitters did not think it was possible to specify out-of-the-box interoperability adequately in the timescale of this submission.

G.5 Specific Security Policies

This specification includes some standard types of security policies for security functionality such as access control, audit, and security of invocations. These are aimed at general commercial users. Some enterprises may require other types of policies, for example, support of mandatory access controls. Where there is a sufficient market for such policies, new policies may be defined, providing they fit with the replaceability interfaces defined in this specification.

G.6 Other Audit Services

This specification only contains limited audit facilities, which allow audit records of security relevant events to be collected. It does not include:

- Filtering of records after generation to further reduce the size of the audit trail.
- Routing audit records to a collection point for consolidation and analysis or routing some as alarms to security administrators. (However, routing may be done using the OMG Event Service, if that is secure enough.)
- Audit reporting or analysis tools to use the audit trails to track down problems.

G.7 Management

This specification contains only the management interfaces, which are essential for security policy management. It specifies how to obtain and use security policy objects. However, it does not contain:

- All facilities for handling domains, policies other than those required for security policy administration. This is to avoid unnecessary conflict with System Management proposals.
- Management of some aspects of security. For example, it does not specify how to create and install permanent keys, as this is implementation specific.

G.8 Reference Restriction

This specification requires the movement of credentials to delegate access rights from one object to another. Another technique of access rights delegation restricts the use of an object reference according to a set of criteria. This approach, know as reference restriction, is under study by a number of vendors, but is not ready for standardization at this time. The criteria used to restrict references could include:

- Whether an object has the right to assert certain privileges, such as act on behalf of a principal, act on behalf of a group of principals, act in a particular role, act with a particular clearance, etc.
- Whether the object reference has been limited to use within a given time interval.
- Whether a particular method can be used by an object holding the object reference.

Various techniques for restricting object references have been developed. Some use cryptographic methods, while others store state in the object associated with the restricted reference, allowing the object to decide if a method request meets the restricted reference use criteria.

It is anticipated that vendors will explore this type of access rights delegation and move towards the standardization of an interface supporting it in a submission to a future RFP.

G.9 Target Control of Message Protection

In the current specification, message protection can be specified by policy administration at both the client and the target object.

Requesting an operation on an object may result in many other objects being invoked. The CORBA security specification in this document allows an intermediate object in such a chain of objects to delegate received credentials to the next object in the chain (subject to policy). However, the current specification does not allow the application to control when and where these credentials are used. A later specification may provide such controls to ride the default quality of protection selectively. Therefore, it could cause some messages to have different qualities of protection during a security association.

The target has no equivalent interface to request the quality of protection for a particular response. There are cases where this could be useful.

A future security specification should consider adding control of quality of protection by the target for individual responses.

G.10 Advanced Delegation Features

Requesting an operation on an object may result in many other objects being invoked. The CORBA security specification in this document allows an intermediate object in such a chain of objects to delegate received credentials to the next object in the chain (subject to policy).

However, the current specification does not allow the application to control when and where these credentials are used.

A later specification may provide such controls.

If so, it is expected that a **`set_controls`** operation on the Credentials object will be added to enable the application to set the controls, and a matching **`get_controls`** operation to enable it to see what controls apply (see the **`set_privileges`** and **`get_attributes`** operations defined in Interfaces under Section 15.5.4, Credentials).

The **`set_controls`** operation would allow the application to specify a set of required control values such as delegation mode (allowing for richer forms of delegation), restrictions on where the credentials may be used and/or delegated, and validity period.

Note: These operations were not included in the specification because of concerns about portability of applications using them. Current delegation implementations use a wide variety of delegation controls, and some use similar controls in semantically different ways. Further implementation experience and investigation may make it possible to define a portable, standard set.

G.11 Reconciling Policy for Overlapping and Hierarchical Domains

This specification does not require support for overlapping or hierarchical security policy domains. However, it is possible to implement both using the interfaces provided.

Recall from Section 15.6, Administrator's Interfaces, that the DomainAccessPolicy for each domain defines which rights are *granted* to subjects when they attempt to access objects in the domain. In order to make an access decision, the AccessDecision logic also needs to know which rights are *required* to execute the operations of an object, which is a member of the relevant domain. The RequiredRights interface provides this information; the AccessDecision object will probably use this interface in most implementations.

A RequiredRights instance can be queried to determine which rights a user must be granted in order to be allowed to invoke an object's operations. The intended use of DomainAccessPolicy and RequiredRights objects by the AccessDecision object is illustrated next, in Figure G-1.

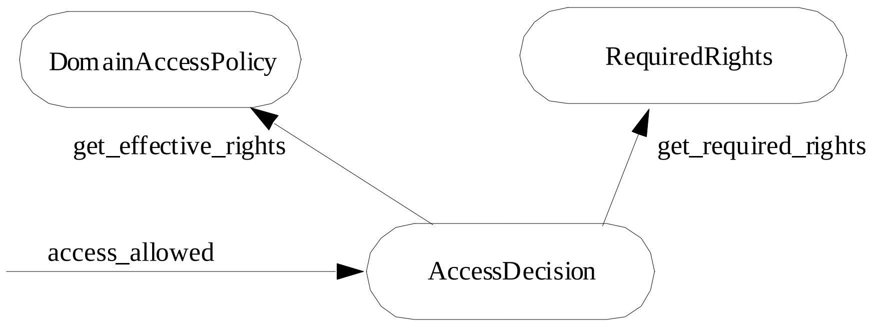

Figure G-1 Intended Use by AccessDecision

AccessDecision retrieves the relevant policy object by calling `get_domain_managers` on the target object reference, and then calling `get_domain_policy(access)` on the returned domain manager (assuming for purposes of this example that there is only one). It then calls `get_effective_rights` on the returned policy object. DomainAccessPolicy calls `get_required_rights` on `RequiredRights` and compares the returned list of required rights with the effective rights. If all required rights have been granted, it grants the access.

Figure G-2 illustrates how the specification could be implemented to support overlapping access policy domains (i.e. to allow an object to be a member of more than one domain, such that each domain has an access policy and all domains' access policies are applied). In the diagram, the AccessDecision object must have logic to combine the policies asserted by the various AccessPolicy objects (which may involve evaluating which AccessPolicy object's policy takes precedence over the others). Note that the AccessDecision object knows the target object reference, because it is passed as an input parameter to the **`access_allowed`** operation.

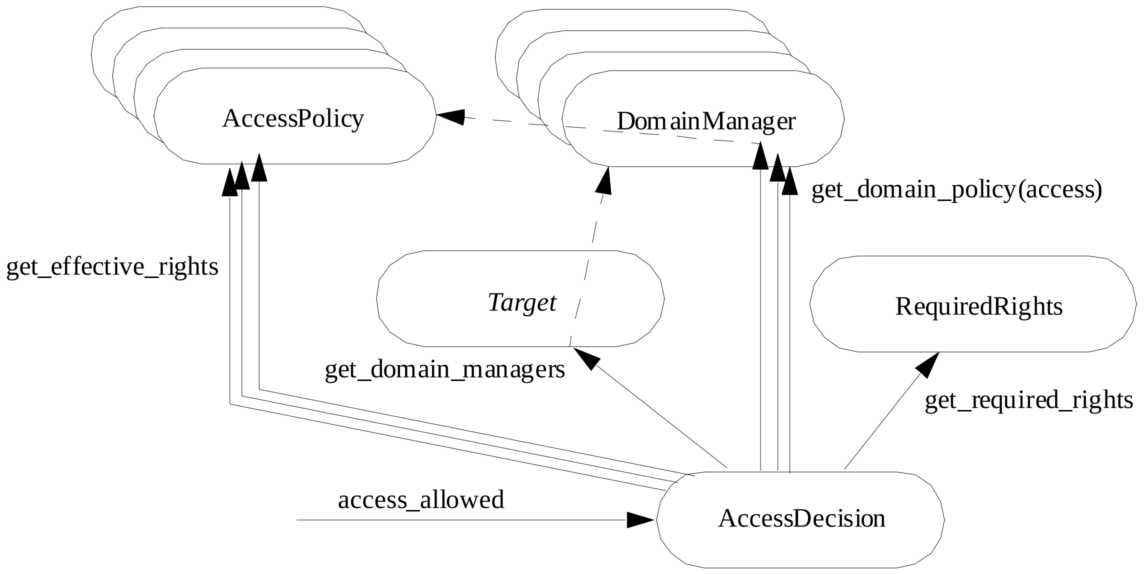

Figure G-2 Supporting Overlapping Access Policy Domains

Hierarchical domains can be handled in a similar way as illustrated in Figure G-3 (note that once again the AccessDecision object's implementation is responsible for reconciling the various retrieved policies).

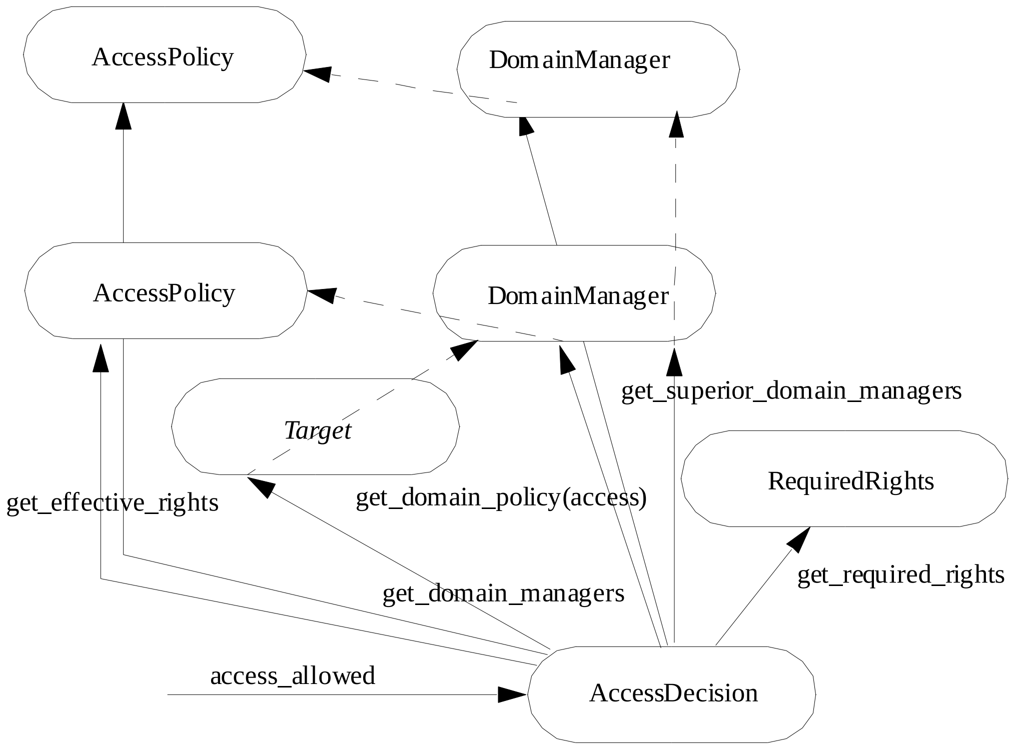

Figure G-3 Hierarchical Domains

G.12 Capability-Based Access Control

Capability-based systems store access policy information in tokens, which are passed from sender to receiver along with a message, rather than in tables associated with target objects or domains. In such systems, the DomainAccessPolicy object will generally not be used in resolving target-side access control checks. Instead, a CapabilityAccessPolicy object might be returned from a call to `object::get_policies` in a capability-based system. This object could retrieve the granted rights from the capability (which will be associated with the requester's credentials), illustrated in Figure G-4.

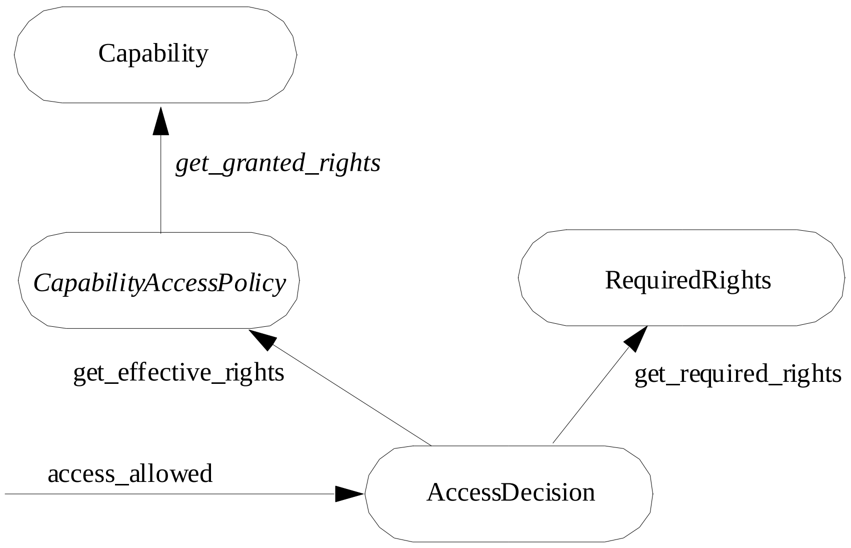

Figure G-4 Retrieving Granted Rights

Note that neither the CapabilityAccessPolicy interfaces nor the Capability interfaces are defined in this specification (the **`get_granted_rights`** call to the capability in the previous diagram is printed in italics, to indicate that no IDL is provided for it in this specification). The diagram assumes that CapabilityAccessPolicy inherits the **`get_effective_rights`** operation from `AccessPolicy`.

G.13 Non-repudiation Services

This specification contains Non-repudiation Services for evidence handling. It is anticipated that future service offerings could include data protection processing and the specification of a delivery service. In addition, it is expected that policy processing interfaces will emerge to cover the broad range of non-repudiation policy coverage within the service.

It is anticipated that the data protection and delivery service functions will be reaching a level of maturity within other standards domains (such as IETF and ISO SC27), which should allow a richer definition of these services to be enabled in future revisions of this specification.

The absence of these services in this specification means that application writers and manipulators will need to consult local implementation practice for the correct course of action to be taken when writing or porting their software.

This specification also does not include a standard format of evidence token for interoperability. In the future, a token format based on public key certificates may be specified.

Appendix H Interoperability Guidelines

H.1 Introduction

This appendix includes:

- Guidelines for defining Security Mechanism TAGs in Interoperable Object References (IORs)
- Examples of the secure inter-ORB protocol, SECIOP

H.2 Guidelines for Mechanism TAG Definition in IORs

Section 15.8, Security and Interoperability, defined a prototype TAG definition for security association mechanisms. This appendix provides guidelines that specifiers of mechanism TAGs (called authors here) should follow.

In addition to registering TAGs with the OMG, authors must lodge a document that explains how the mechanism (and its associated options) is mapped to this standard. Its document should:

- Identify the "security mechanism tagged component" being described. It may be either:
  - A new component TAG for the mechanism with a set of options it can have (for example, a separate TAG for each combination of mechanism and algorithm),

  or

  - Use TAG_GENERIC_SEC_MECH and specify the mechanism OID (for use in the `security_mechanism_type` field) being described by this specification.

  It may not be both.
- Specify the scope implied by the above mechanism identifier. This should not exceed:
  - Security association mechanism
  - Negotiation protocols
  - Cryptographic algorithms
  - Authentication method (e.g. public key)
- For the first example under the first bullet, describe the format, contents, and encoding of the `component_data` field for the TAG-specific components. For the second example under the first bullet, describe the format, contents, and encoding of the data in the `mech_specific_data` and components fields of the TAG specific components. In each case, this may include:
  - Allocating new component TAGs and describing the format, contents, and encoding of their data.
  - Specifying the use of these new tagged components, as well as other predefined tagged components within TAG-specific components.

  - Specifying the use of these new tagged components, as well as other predefined tagged components that may or should appear at the top level of the multicomponent profile.
- Describe a model that should be followed when defining future extensions or variations using the same mechanism.
- The author must define either by reference to another document, or explicitly, the format of the context tokens used by the mechanism in the SECIOP protocol.

H.3 SECIOP Examples

H.3.1 Mutual Authentication

In this example, the client wishes to authenticate the identity of the target (in addition to the targets requirement to authenticate the client) before it is prepared to send a request to the target.

The client sends an EstablishContext message to the target containing the client's context id for the association, and the token required by the target to authenticate it and define the options chosen by the client for the association. The target verifies the client's token and generates the token required by the client to authenticate the target. The target sends this token (along with the client's context id for the association and its own) to the client in a CompleteEstablishContext message. When the client receives this message, it authenticates the target using the token supplied by the target and establishes the peer id as part of the context.

Having completed the establishment of the context, the client sends the request as part of a MessageInContext message, which includes the target's context identifier and the integrity token for the message. When the target receives the message, it identifies the context by its identifier, checks the integrity of the message with the token, and passes the message to GIOP. When the reply is returned, it is sealed for integrity and returned to the client in a SECIOP MessageInContext with the client identifier for the context and the generated integrity token.

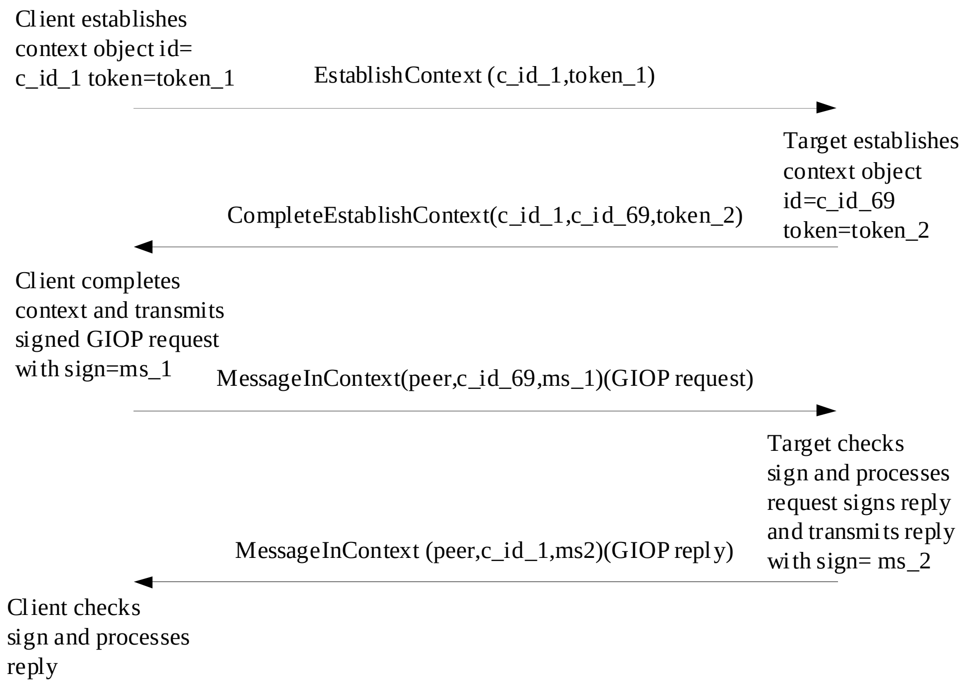

Figure H-1 Mutual Authentication

H.3.2 Confidential Message with Context Establishment

This example describes how context establishment is combined with the transmission of a confidentiality protected message when the client does not wish to authenticate the target before passing it a message.

The client establishes its context object with identifier c_id_1. This identifier is included with the token (token_1) in an EstablishContext message. The GIOP request is transformed into the message seal (ms_1) and sent with the client's context identifier in a MessageInContext.

When the target receives the message, it first processes the EstablishContext message, authenticating the client and allowing the target to create its context object. It then unseals the message in ms_1 and passes it to GIOP.

When GIOP sends the reply, SECIOP adds a CompleteEstablishContext message to the MessageInContext message, which protects the reply, to enable the target to return its context identifier to the client. When the client receives the message, it first completes its view of the context (adding the targets id to the state for the context). It can then unseal the reply from ms_2 and passes the reply message up the protocol stack.

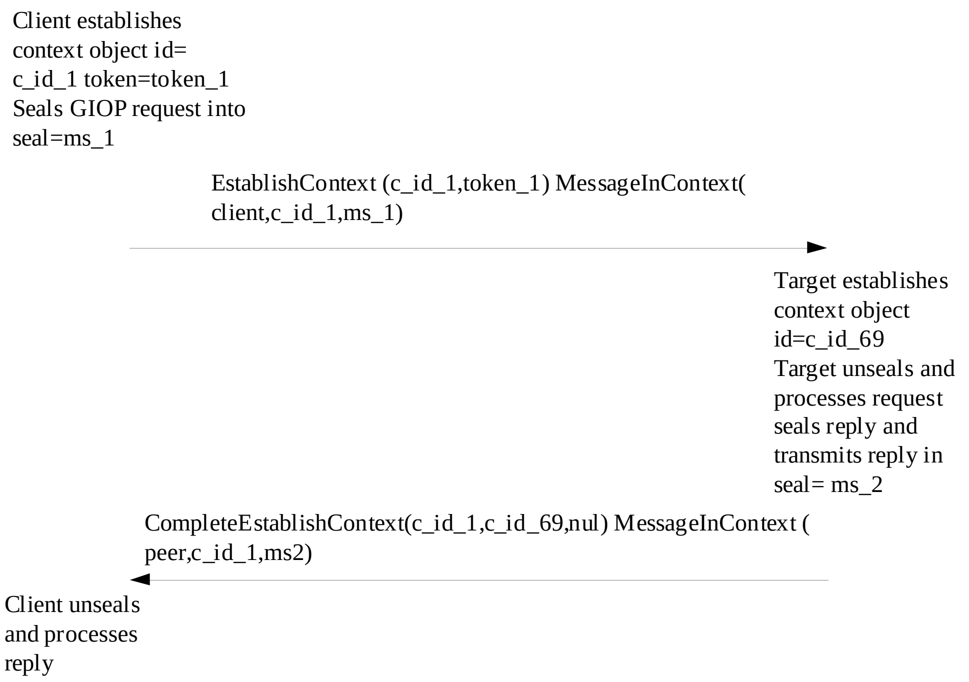

Figure H-2 Confidential Message with Context Establishment

H.3.3 Fragmented GIOP Request with Context Establishment

In this example, the security context is established as part of the processing of a fragmented GIOP request (note that the current GIOP protocol does not support fragmentation, but this example indicates the independence of SECIOP from the current GIOP protocol and explains how the SECIOP protocol would handle a fragmented GIOP request). The sequence described reflects the requirement of the target to authenticate the client's privileges.

The client establishes its context object (with id c_id_1) and passes this identifier with the authentication token in an EstablishContext message. As the client does not require authenticating the target, this message is sent with a MessageInContext message with the integrity sign (ms_1) and the GIOP fragment (as the message field of the MessageInContext).

When the target receives the messages, it authenticates the client using token_1. It then creates a context object with c_id_69, and then processes the MessageInContext, checking the integrity of the message using sign ms_1. Having checked the message, it passes the fragment up the protocol stack.

The client sends the final fragment as a MessageInContext with sign ms_2, but as the target has not yet passed its identifier for the context to the client, the client uses its own identifier for the context.

The target finds its context object from the client's identifier (c_id_1) and checks the integrity of the message. It then passes the final fragment up the protocol stack to GIOP.

GIOP now has a complete request and can invoke the object (subject to the access decision function).

GIOP generates a single fragment reply, which is passed to the SECIOP protocol machine. The reply is sent within a MessageInContext with sign ms_3. In addition, a CompleteEstablishContext message is generated to allow the target to pass its identifier for the context (c_id_69) to the client for use in future messages.

The client receives the message and updates its context object to record the target's context identifier. It then checks the integrity of the MessageInContext and passes the reply up the protocol stack (to GIOP).

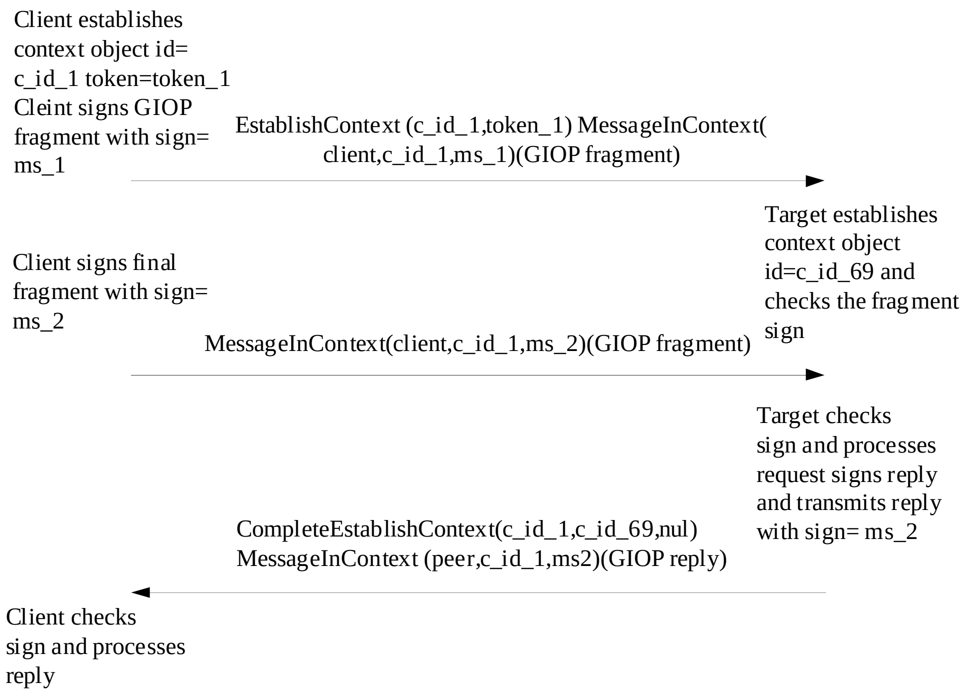
Client establishes context object id= c_id_1 token=token_1 Cleint signs GIOP fragment with sign= ms_1
EstablishContext (c_id_1,token_1) MessageInContext(client,c_id_1,ms_1)(GIOP fragment)
Target establishes context object id=c_id_69 and checks the fragment sign
Client signs final fragment with sign= ms_2
MessageInContext(client,c_id_1,ms_2)(GIOP fragment)
Target checks sign and processes request signs reply and transmits reply with sign= ms_2
CompleteEstablishContext(c_id_1,c_id_69,nul) MessageInContext (peer,c_id_1,ms2)(GIOP reply)
Client checks sign and processes reply

Figure H-3 Fragmented GIOP Request with Context Establishment

Appendix I Glossary

I.1 Definitions

***absolute time**:* Time accurate within a known margin of error.

***access control**:* The restriction of access to resources to prevent its unauthorized use.

***access control information** (ACI):* Information about the initiator of a resource access request, used to make an access control enforcement decision.

***access control list**:* A list of entities, together with their access rights, which are authorized to have access to a resource.

***access decision function**:* The function which is evaluated in order to make an access control enforcement decision. The inputs to an access decision function include the requester's access control information (q.v.), the resource's control information, and context data.

***ADO**: Access Decision Object:* The CORBA security object which implements access decision functions.

***accountability**:* The property that ensures that the action of an entity may be traced uniquely to the entity.

***active threat**:* The threat of a deliberate unauthorized change to the state of a system.

***adjudicator**:* An authority that resolves disputes among parties in accordance with a policy. In CORBA security, an adjudicator evaluates non-repudiation evidence in order to resolve disputes.

anonymous user: A user of the system operating under a distinguished "public" identity corresponding to no specific user.

***assurance**:* 1. Justified confidence in the security of a system. 2. Development, documentation, testing, procedural, and operational activities carried out to ensure that a system's security services do in fact provide the claimed level of protection.

***asymmetric key**:* One half of a key pair used in an asymmetric ("public-key") encryption system. Asymmetric encryption systems have two important properties: (i) the key used for encryption is different from the one used for decryption (ii) neither key can feasibly be derived from the other.

***audit**:* See *security audit.*

***audit event**:* The data collected about a system event for inclusion in the system audit log.

***audit trail**:* See *security audit trail.*

***authentication**:* The verification of a claimant's entitlement to use a claimed identity and/or privilege set.

***authentication information**:* Information used to establish a claimant's entitlement to a claimed identity (a common example of authentication information is a password).

***authorization**:* The granting of authority, which includes the granting of access based on access rights.

***availability**:* The property of being of being accessible and usable upon demand by an authorized user.

***call chain**:* The series of client to target object calls required to complete an operation. Used in this specification in conjunction with delegation.

***certification authority**:* A party trusted to vouch for the binding between names or identities and public keys. In some systems, certification authorities generate public keys.

***ciphertext**:* The result of applying encryption to input data; encrypted text.

***cleartext**:* Intelligible data; text which has not been encrypted or which has been decrypted using the correct key. Also known as "plaintext".

***confidentiality**:* The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

***conformance level**:* A graduated sequence of defined sets of functionality defined by the CORBA Security specification. An implementation must implement at least one of these defined sets of functionality in order to claim conformance to CORBA Security.

conformance option: A defined set of functionality which implementations may optionally provide in order to claim CORBA Security conformant functionality over and above the minimum required by the defined conformance levels.

***conformance statement**:* A written document describing the conformance levels and conformance options to which an implementation of the OMG CORBA Security specification conforms.

***control attributes**:* The set of characteristics which restrict when and where privileges can be invoked or delegated.

***counter-measures**:* Action taken in response to perceived threats.

***credentials**:* Information describing the security attributes (identity and/or privileges) of a user or other principal. Credentials are claimed through authentication or delegation (q.v.) and used by access control (q.v.).

***current object**:* An object representing the current execution context; CORBA Security associates security state information, including the credentials of the active principal, with the current object.

***DAC**:* Discretionary Access Control - an access control policy regime wherein the creator of a resource is permitted to manage its access control policy information.

***data integrity**:* The property that data has not been undetectably altered or destroyed in an unauthorized manner or by unauthorized users.

DCE: Distributed Computing Environment (of OSF).

***DCE CIOP**:*DCE Common Inter-ORB Protocol - the protocol specified in the OMG CORBA 2.0/ Interoperability specification which uses the DCE RPC for interoperability.

decipherment: Generation of cleartext from ciphertext by application of a cryptographic algorithm with the correct key.

decryption: See *decipherment.*

delegation: The act whereby one user or principal authorizes another to use his (or her or its) identity or privileges, perhaps with restrictions.

denial of service: The prevention of authorized access to resources or the delaying of time-critical operations.

digital signature: Data appended to, or a cryptographic transformation of. a data unit that allows a recipient of the data unit to prove the source and integrity of the data against forgery, e.g. by the recipient.

domain: A set of objects sharing a common characteristic or abiding by a common set of rules. CORBA Security defines several types of domains, including security policy domains, security environment domains, and security technology domains.

domain manager: A CORBA Security object through whose interfaces the characteristics of a security policy domain are administered.

encipherment: Generation of ciphertext from corresponding cleartext by application of a cryptographic algorithm and a key.

encryption: See *encipherment.*

ESIOP: Environment-Specific Inter-ORB Protocol (specified in the OMG CORBA 2.0/ Interoperability specification).

evidence: Data generated by the CORBA Security Non-Repudiation service to prove that a specific principal initiated a specific action.

evidence token: A data structure containing CORBA Security Non-Repudiation evidence.

federated domains: Separate domains whose policy authorities have agreed to a set of shared policies governing access by users from one domain to resources in another.

GSS-API: Generic Security Services- Application Programming Interface - specified by RFC 1508 issued by the Internet IETF. An update to this interface is near completion as this is written, and it is anticipated that RFC 1508 will be superseded by a revised specification soon.

GIOP: General Inter-ORB Protocol (specified in the OMG CORBA 2.0/ Interoperability specification.)

group: A CORBA Security privilege attribute. Many users (and other principals) may be assigned the same group attribute; this allows administrators to simplify security administration by granting rights to groups rather than to individual principals.

granularity: The relative fineness or coarseness by which a mechanism may be adjusted.

hierarchical domains: A set of domains together with a precedence hierarchy defining the relationships among their policies.

identity: A security attribute with the property of uniqueness; no two principals' identities may be identical. Principals may have several different kinds of identities, each unique (for example, a principal may have both a unique audit identity and a unique access identity). Other security attributes (e.g. groups, roles, etc...) need not be unique.

immediate invoker: In a delegated call chain, the client from which an object directly receives a call.

impersonation: The act whereby one principal assumes the identity and privileges of another principal without restrictions and without any indication visible to recipients of the impersonator's calls that delegation has taken place.

initiator: The first principal in a delegation "call chain"; the only participant in the call chain which is not the recipient of a call.

integrity: In security terms, the property that a system always faithfully and effectively enforces all of its stated security policies.

interceptor: An object which provides one or more specialized services, at the ORB invocation boundary, based upon the context of the object request,. The OMG CORBASecurity specification define the security interceptors.

intermediate: An object in a delegation "call chain" which is neither the initiator or the ultimate (final) target.

IETF: Internet Engineering Task Force. Reviews an issues Internet standards.

IIOP: Internet Interoperable Object Protocol (specified in the OMG CORBA 2.0/ Interoperability specification).

IOR: Interoperable Object Reference - a data structure specified in the OMG CORBA 2.0/ Interoperability specification.

ITSEC: Information Technology Security Evaluation Criteria (of ECSC-EEC-EAEC). Harmonized Criteria.

MAC: Mandatory Access Control - an access control regime wherein resource access control policy information is always managed by a designated authority, regardless of who creates the resources.

mechanism: A specific implementation of security services, using particular algorithms, data structures, and protocols.

message protection: Security protection applied to a message to protect it against unauthorized access or modification in transit between a client and a target.

mutual authentication: The process whereby each of two communicating principals authenticates the other's identity. Frequently this is a prerequisite for the establishment of a secure association between a client and a target.

Non-Repudiation: The provision of evidence which will prevent a participant in an action from convincingly denying his responsibility for the action.

ORB Core: The functionality provide by the CORBA Object Request Broker which provides the basic representations of objects and the communication of requests.

***ORB Services**:* Elements of functionality provided transparently to applications by the CORBA Object Request Broker in response to the implicit context of an object request.

***ORB technology domain**:* A set of objects or entities that share a common ORB implementation technology.

***originator**:* The entity in an object request which creates the request.

***passive threat**:* The threat of unauthorized disclosure of information without changing the state of the system.

***physical security**:* The measures used to provide physical protection of resources against deliberate and accidental threats.

***POSIX**:* Portable Open System Interfaces (for) UNIX - A set of standardized interfaces to UNIX systems specified by IEEE Standard 1003.

***principal**:* A user or programmatic entity with the ability to use the resources of a system.

***privacy**:* 1. See *confidentiality*. 2. The right of individuals to control or influence what information related to them may be collected and stored and by whom that information may be disclosed.

***private key**:* In a public-key (asymmetric) cryptosystem, the component of a key pair which is not divulged by its owner.

***privilege**:* A security attribute (q.v.) which need not have the property of uniqueness, and which thus may be shared by many users and other principals. Examples of privileges include groups, roles, and clearances.

***proof of delivery**:* Non-repudiation evidence demonstrating that a message or data has been delivered.

***proof of origin**:* Non-repudiation evidence identifying the originator of a message or data.

***proof of receipt**:* Non-repudiation evidence demonstrating that a message or data has been received by a particular party.

***protection boundary**:* The domain boundary within which security services provide a known level of protection against threats.

***PDU**:* Protocol Data Unit. The data fields of a protocol message, as distinguished from the protocol header and trailer fields.

***proof of submission**:* Non-repudiation evidence demonstrating that a message or data has been submitted to a particular principal or service.

***public key**:* In a public-key (asymmetric) cryptosystem, the component of a key pair which is revealed.

***public-key cryptosystem**:* An encryption system which uses an asymmetric-key (q.v.) cryptographic algorithm.

***QOP**:* Quality of Protection. The type and strength of protection provided by a message-protection service.

RPC: Remote Procedure Call.

replaceability: The quality of an implementation which permits substitution of one security service for another semantically similar service.

repudiation: Denial by one of the entities involved in an action of having participated in all or part of the action.

RFP: Request for Proposal. An OMG procedure for soliciting technology from OMG members.

right: A named value conferring the ability to perform actions in a system. Access control policies grant rights to principals (on the basis of their security attributes); in order to make an access control decision, access decision functions compare the rights granted to a principal against the rights required to perform an operation.

rights type: A defined set of rights.

role: A privilege attribute representing the position or function a user represents in seeking security authentication. A given human being may play multiple roles and therefore require multiple role privilege attributes.

RSA: An asymmetric encryption algorithm invented by Ron Rivest, Adi Shamir, and Len Adelman.

seal: To encrypt data for the purpose of providing confidentiality protection.

secret-key cryptosystem: A cryptosystem which uses a symmetric-key (q.v.) cryptographic algorithm.

secure time: A reliable Time service that has not been compromised, and whose messages can be authenticated by their recipients.

security association: The shared security state information which permits secure communication between two entities.

security attributes: Characteristics of a subject (user or principal) which form the basis of the system's policies governing that subject.

security audit: The facility of a secure system which records information about security-relevant events in a tamper-resistant log. Often used to facilitate an independent review and examination of system records and activities in order to test for adequacy of system controls, to ensure compliance with established policy and operational procedures, to detect breaches in security, and to recommend changes in control, policy and procedures.

security features: Operational information which controls the security protection applied to requests and responses in a CORBA Security conformant system.

security context: The CORBA Security object which encapsulates the shared state information representing a security association.

security policy: The data which defines what protection a system's security services must provide. There are many kinds of security policy, including access control policy, audit policy, message protection policy, non-repudiation policy, etc.

***security policy domain**:* A domain whose objects are all governed by the same security policy. There are several types of security policy domain, including access control policy domains and audit policy domains.

***security service**:* Code that implements a defined set of security functionality. Security services include Access Control, Audit, Non-repudiation, and others.

***security technology domain**:* A set of objects or entities whose security services are all implemented using the same technology.

***subject**:* An active entity in the system; either a human user principal or a programmatic principal.

***symmetric key**:* The key used in a symmetric ("secret-key") encryption system. In such systems, the same key is used for encryption and decryption.

***tagged profile**:* The data element in an IOR which provides the profile information for each protocol supported.

***target**:* The final recipient in a delegation "call chain." The only participant in such a call chain which is not the originator of a call.

***target ACI**:* The Access Control Information for the target object.

target object: The recipient of a CORBA request message.

***threat**:* A potential violation of security.

***traced delegation**:* Delegation wherein information about the initiator and all intervening intermediates is available to each recipient in the call chain, or to the authorization subsystem controlling access to each recipient.

***trust model**:* A description of which components of the system and which entities outside the system must be trusted, and what they must be trusted for, if the system is to remain secure.

***trusted code**:* Code assumed to always perform some specified set of operations correctly.

***TCB**:* Trusted Computing Base. The portion of a system which must function correctly in order for the system to remain secure. A TCB should be tamper-proof and its enforcement of policy should be noncircumventable. Ideally a system's TCB should also be as small as possible, to facilitate analysis of its integrity.

***TCSEC**:* Trusted Computer System Evaluation Criteria (a U.S. Department of Defense Standard specifying requirements for secure systems).

***unauthenticated principal**:* A user or other principal who has not authenticated any identity or privilege.

***UNO**:* Universal Networked Objects (an OMG Specification, now obsolete).

***UTC**:* Coordinated Universal Time.

***unsecure time**:* Time obtained from an unsecure time services.

***UTO**:* Universal Time Object.

***user**:* A human being using the system to issue requests to objects in order to get them to perform functions in the system on his behalf.

***user sponsor**:* The interactive user interface to the system which acts as the authenticating authority (e.g. validating passwords) which validate the identity of a user.

***vault**:* The CORBA Security object which creates security context objects.

***X/Open**:* X/Open Company Ltd., U.K.

I.2 References

The following sources were used in the preparation of this glossary:

Applied Cryptography, 2nd edition *by Bruce Schneier, John Wiley and Sons, New York, 1996.*

ISO Standard 7498-2*, "Information Processing Systems -- Open Systems Interconnection -- Basic Reference Model -- Part 2:Security Architecture", International Standards Organization,1989.*

ECMA TR/46 *"Security in Open Systems: A Security Framework", European Computer Manufacturers Association, 1988.*

ITSEC "Information Technology Security Evaluation Criteria" *European Commission, 1991.*

DoD Standard 5200.28-STD *"Department of Defense Trusted Computer System Evaluation Criteria", US Department of Defense, 1985.*

X/Open Snapshot: *"Distributed Security Framework: Company Review Draft", X/Open Company Ltd.,U.K. 1994.*

Computer Related Risks*: Peter G. Neuman, The ACM Press, 1995*