OSF DCE SIG
Request For Comments: 9.0

P. Lin (IBM)
M. Leitch (IBM)

July 1992

EMBEDDED DCE SECURITY BRIDGE (EDB) APPLICATION PROGRAMMING INTERFACE

1. INTRODUCTION

This document describes the application programming interface (API) of the Embedded DCE Security Bridge (EDB), and its implementation on top of existing DCE security facilities.

The embedded DCE security bridge is intended to provide an API and a set of supporting functions to enable usage of DCE security by various communication mechanisms. The supporting functions are packaged in the key manager, which manages secret keys, conversation keys, and other related information about the client-server security interaction.

1.1. Design Principles

The EDB is designed with the following principles in mind:

(a) Encapsulate common functions used by all or many communication mechanism implementions.

(b) Insulate communication mechanism implementations from changes in the underlying DCE security facilities, e.g., support of new authentication or cryptographic algorithms.

(c) Minimize the number and complexity of API calls that a communication mechanism implemention needs to make to the EDB.

(d) Minimize other modifications required in the communication mechanism implementions.

(e) Maximize portability of applicable EDB functions to different operating system platforms that support DCE.

1.2. Characteristics of EDB

The EDB has the following characteristics:

(a) It includes administrative functions required by its callers, such as server key management and principal definition.

(b) It provides its callers with explicit knowledge of the progress of the authentication exchange, which enables the explicit control of the exchange that is needed in optimizing the performance of communication mechanisms.

(c) In the area of delegation, it closely follows the delegation proposal for DCE 1.1 [RFC 3.0].

(d) It directly supports the use of DCE principal names and privilege attribute certificates (PACs) as parameters, without requiring format conversion.

(e) It has been mapped to the existing DCE security facilities to verify the feasibility and practicality of its use in the DCE environment.

1.3. Implementation Notes

The abstract description of the EDB API given in this document is designed to be implementation-independent. Additionally, a suggested high-level implementation flow in the DCE environment is given — but this is always clearly marked as ‘‘*Notes on implementation on top of DCE security facilities.*’’

It is suggested that the EDB API could occupy an intermediate position in the DCE environment, below GSS-API and above actual security mechanisms. The EDB API would thus be available for security-knowledgeable, performance-conscious applications.

2. APPLICATION PROGRAMMING INTERFACE

The API functions for server principal definition, registration (with the EDB), authentication, communications integrity and communications confidentiality are described in separate sections below.

To facilitate portability to different platforms, usage of all functions except those identified as ‘‘for authorized callers only’’[1] is restricted indirectly. On multi-user operating systems, client credentials caches and key storage maintained by the key manager are protected by local system access control. Since each function accesses one or both of these data structures, the caller must have the required access permissions in order for a call to complete successfully.[2]

2.1. Server Principal Definition

2.1.1. Functions for general callers

2.1.1.1. Define server principal to DCE registry

(a) *Function name:* edb_define_principal

(b) *Input parameters:*

  (i) admin_userid — Local userid of administrator.[3]

  (ii) principal — Name of server principal to be defined.

  (iii) init_passwd — Initial password of server principal.

(c) *Output parameters:*

  (i) status — Completion status.

(d) *Description:* This function is intended to be called by an automated installation procedure running under the identity of an administrator to define a server principal to the DCE registry.

(e) *Notes on implementation on top of DCE security facilities:*

This function assumes that: (1) The DCE registry in the cell has been integrated with the local machine’s registry. (2) The administrator identified by admin_userid is a DCE principal who has privileges to create principals and accounts in the registry; since the DCE security server checks these privileges before allowing updates to the registry, edb_define_principal does not need to be an authorized call.

  (i) Obtain the system id (machine principal id) of the local system.

  (ii) Call the extended sec_idmap interface[4] to map the (local userid, system id) pair for the administrator to a DCE principal name. The mapping information is stored in the integrated DCE registry.

  (iii) Request a ticket-granting ticket (TGT) for the administrator principal from the DCE security server.

  (iv) Get the administrator’s key from the local registry.

  (v) Decrypt the TGT envelope with the key. The envelope contains, in addition to the TGT, an associated conversation key.

  (vi) Request a PTGT for the administrator principal from the DCE security server.

  (vii) Decrypt the PTGT envelope with the TGS conversation key.

---

1. Authorized callers are trusted by the local operating system and have special privileges not available to general callers.
2. The implementation should perform appropriate clean-up for calls that do not complete successfully.
3. The name by which the administrator is known on the local system.
4. Provided as part of the integration of the local registry with the DCE registry.

(viii) Set up a login context for the administrator.

(ix) Define the server principal to the DCE registry.

(x) Define an account for the server principal in the DCE registry, supplying the initial password.

2.2. Registration with EDB

2.2.1. Functions for general callers

2.2.1.1. Register a general client

(a) *Function name:* edb_register

(b) *Input parameters:*

(i) caller_name — Name of calling client.

(c) *Output parameters:*

(i) caller_token — Caller's token for interactions with key manager.

(ii) status — Completion status.

(d) *Description:* This function is called by a client to register itself with the key manager at start-up time.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Set up storage for conversation keys and related information.

(ii) The key manager returns a token for the caller to use in subsequent interactions.

2.2.1.2. De-register a general client

(a) *Function name:* edb_deregister

(b) *Input-output parameters:*

(i) caller_token — Caller's token for interactions with key manager.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a client to de-register itself with the key manager at shut-down time.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Erase storage set aside for the caller's conversation keys and related information.

(ii) Invalidate the caller's token.

2.2.2. Functions for authorized callers only

Although all functions in this section are indicated as authorized calls, edb_register_server and edb_deregister_server can be implemented as unauthorized calls on platforms that do not support the distinction between general and authorized callers.

2.2.2.1. Register a server or trusted client

(a) *Function name:* edb_register_server

(b) *Input parameters:*

(i) caller_name — Name of calling server or trusted client.

(ii) init_passwd — Initial password for server or trusted client.

(c) *Output parameters:*

(i) caller_token — Caller's token for interactions with key manager.

(ii) status — Completion status.

(d) *Description:* This function is called by a server or a trusted client (one that logs users in to DCE using edb_trusted_login) to register itself with the key manager at start-up time, and start up automatic key management.

(e) *Notes on implementation on top of DCE security facilities:*

This function assumes that the initial password supplied has also been entered in the DCE registry by an administrator.

(i) Check that the caller is authorized.

(ii) Set up storage for conversation keys and related information.

(iii) Pass the initial password to the key manager. The key manager converts this password to the initial server key and stores the key.

(iv) The key manager will log in to DCE as the caller and obtain its key expiration time. This is the next time at which the caller's key will be automatically updated.

(v) The key manager returns a token for the caller to use in subsequent interactions.

2.2.2.2. De-register a server or trusted client

(a) *Function name:* edb_deregister_server

(b) *Input-output parameters:*

(i) caller_token — Caller's token for interactions with key manager.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a server or a trusted client (one that logs users in to DCE using edb_trusted_login) to de-register itself with the key manager at shut-down time.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Check that the caller is authorized.

(ii) Erase storage set aside for the caller's conversation keys and related information.

(iii) Terminate automatic key management for the caller.

(iv) Invalidate the caller's token.

2.2.2.3. Register multiple callers with key manager

(a) *Function name:* edb_register_multiple

(b) *Input parameters:*

(i) list_length — Length of caller_name_list, c_or_s_list, init_passwd_list, caller_token_list and status_list.

(ii) caller_name_list — List of client or server names.

(iii) c_or_s_list — List of flags: = 1 if server or trusted client, = 0 otherwise.

(iv) init_passwd_list — List of initial passwords (an element in this list is null if the corresponding element in c_or_s_list = 0).

(c) *Output parameters:*

(i) caller_token_list — List of caller tokens for interactions with key manager.

(ii) status_list — List of completion statuses.

(d) *Description:* This function is called by an authorized caller to register a list of clients and servers with the key manager at start-up time.

(e) *Notes on implementation on top of DCE security facilities:*

  (i) Check that the caller is authorized.

  (ii) For each caller in caller_name_list:

    [a] If the corresponding element of c_or_s_list = 1 (server or trusted client), invoke edb_register_server with the corresponding elements of init_passwd_list, caller_token_list and status_list as parameters.

    [b] Otherwise, invoke edb_register with the corresponding elements of caller_token_list and status_list as parameters.

2.2.2.4. De-register multiple callers with key manager

(a) *Function name:* edb_deregister_multiple

(b) *Input-output parameters:*

  (i) list_length — Length of caller_token_list and status_list.

  (ii) caller_token_list — List of caller tokens for interactions with key manager.

(c) *Output parameters:*

  (i) status_list — List of completion statuses.

(d) *Description:* This function is called by an authorized caller to de-register a list of clients and servers with the key manager at shut-down time.

(e) *Notes on implementation on top of DCE security facilities:*

  (i) Check that the caller is authorized.

  (ii) For each caller in caller_name_list:

    [a] If the caller is a server or trusted client, invoke edb_deregister_server with the corresponding elements of caller_token_list and status_list as parameters.

    [b] Otherwise, invoke edb_deregister with the corresponding elements of caller_token_list and status_list as parameters.

2.3. Authentication

In the EDB API, the words ‘‘ticket’’ and ‘‘authenticator’’ are used as generic terms for (full) authentication token and light-weight authentication token respectively.

Note that the format of the authenticator and authentication reply generated by the API functions in this section can be dependent on the specific communication mechanism calling the EDB.

Notes on implementation on top of DCE security facilities: For the edb_get_delegation, edb_receive_delegation and edb_get_client_ticket functions, the privilege ticket-granting ticket (PTGT) is forwarded because the client does not know which servers will be in the chain beyond the first one. This scheme is suitable where the servers involved are trusted to fully assume the client’s identity. Note that this scheme closely parallels the design for delegation under authenticated RPC in OSF DCE 1.1, which is sketched out in [RFC 3.0]. It is proposed that the case in which trust in intermediate servers is not warranted be addressed in conjunction with delegation in OSF DCE 1.1, since the support for this case requires changes to the privilege server. These changes should not affect the EDB API, however.

2.3.1. Calling sequences

2.3.1.1. Client on workstation to server on host

For the ‘‘client on workstation to server on host’’ configurations, edb_get_ticket and edb_parse_ticket are the only functions required to authenticate the client to the server. The other two functions, edb_build_auth_reply and edb_parse_auth_reply, are used to authenticate the server to the client, in order to achieve mutual authentication.

A typical end-to-end calling sequence without mutual authentication would be:

(a) edb_get_ticket

A client gets a ticket to authenticate itself to the server.

(b) edb_parse_ticket

The server parses the ticket from the client.

A typical end-to-end calling sequence with mutual authentication would be:

(a) edb_get_ticket

A client gets a ticket to authenticate itself to the server.

(b) edb_parse_ticket

The server parses the ticket from the client.

(c) edb_build_auth_reply

The first server builds an authentication reply to authenticate itself to the client.

(d) edb_parse_auth_reply

The client parses the authentication reply from the first server.

2.3.1.2. Server on host to server on host

The functions edb_get_delegation, edb_receive_delegation and edb_get_client_ticket are intended for the ‘‘server on host to server on host’’ configuration.

A typical end-to-end calling sequence with delegation would be:

(a) edb_get_ticket

A client gets a ticket to authenticate itself to the first server in a chain.

(b) edb_parse_ticket

The first server parses the ticket from the client.

(c) edb_build_auth_reply

The first server builds an authentication reply to authenticate itself to the client.

(d) edb_parse_auth_reply

The client parses the authentication reply from the first server.

(e) edb_get_delegation

The client gets a delegation token to forward to the first server.

(f) edb_receive_delegation

The first server receives the delegation token from the client.

(g) edb_get_client_ticket

The first server gets a ticket on behalf of the client to the second server in the chain.

Note that mutual authentication is included because the client may wish to verify the identity of the server before forwarding its delegation token. If it is acceptable to forgo this verification, then the calling sequence can be simplified to the following, in which the client gets both the ticket for the first server and its delegation token and forwards the two items to the server together.

(a) edb_get_ticket

A client gets a ticket to authenticate itself to the first server in a chain.

(b) edb_get_delegation

The client gets a delegation token to forward to the first server.

(c) edb_parse_ticket

The first server parses the ticket from the client.

(d) edb_receive_delegation

The first server receives the delegation token from the client.

(e) edb_get_client_ticket

The first server gets a ticket on behalf of the client to the second server in the chain.

2.3.1.3. Client on host to server on workstation

The last two functions (edb_trusted_login, edb_trusted_logout) are intended for the ‘‘client on host to server on workstation’’ configuration.

Typical calling sequences for this configuration are the same as those described in the sections on ‘‘Client on workstation to server on host’’ and ‘‘Server on host to server on host’’, except that edb_trusted_login is used to log the client or the user on whose behalf it acts into DCE, in order to provide the login context needed by these calling sequences, and edb_trusted_logout is used to purge the login context.

2.3.2. Functions for general callers

2.3.2.1. Get ticket

(a) *Function name:* edb_get_ticket

(b) *Input parameters:*

  (i) caller_token — Token for calling client.

  (ii) server_name — Name of target server.

  (iii) ticket_req — = 1 if both ticket and authenticator are required, = 0 if only authenticator is required.

(c) *Input-output parameters:*

  (i) conv_token — Conversation identifier (null if ticket_req = 1).

(d) *Output parameters:*

  (i) ticket — Ticket for target server (null if ticket_req = 0, unless the previous ticket has expired).

  (ii) authent — Authenticator.

  (iii) status — Completion status.

(e) *Description:* This function is called by a client to get a ticket, an authenticator and a conversation key for a target server. There is an option to get an authenticator only.

(f) *Notes on implementation on top of DCE security facilities:*

This function assumes that the calling client, or the user on whose behalf it acts, has logged in to DCE[5] so that a login context is available.

(i) Get the current login context.

(ii) If only an authenticator is required:

[a] Get the conversation key for the target server.

[i] If there is an unexpired credential (ticket and conversation key) in the client's credentials cache, the conversation key will be returned.

[ii] Otherwise, a credential (ticket and new conversation key) will be obtained from the DCE security server.

[b] Build an authenticator. If a new conversation key was returned by the previous step, it is included in the authenticator as the subkey, so that it will be used for subsequent communication between the client and the target server.

[c] Encrypt the authenticator in the conversation key identified by conv_token.

[d] Pass the new conversation key to the key manager for storage under conv_token.

(iii) Otherwise:

[a] Get a credential (ticket and conversation key) for the target server.

[i] If there is an unexpired credential in the client's credentials cache, it will be returned.

[ii] Otherwise, a credential will be obtained from the DCE security server. Pass the conversation key to the key manager for storage and get the conv_token returned by the key manager.

[b] Build an authenticator.

[c] Encrypt the authenticator in the conversation key.

2.3.2.2. Parse ticket

(a) *Function name:* edb_parse_ticket

(b) *Input parameters:*

(i) caller_token — Token for calling server.

(ii) ticket — Ticket received (null if only the authenticator needs to be verified).

(iii) authent — Authenticator.

(c) *Input-output parameters:*

(i) conv_token — Conversation identifier (supplied as input if only the authenticator needs to be verified).

(d) *Output parameters:*

(i) userid — Local userid.[6]

(ii) principal — DCE principal name.

(iii) pac — Privilege attribute certificate.

5. Via dce_login, the DCE sec_login API (see [OSF]), or edb_trusted_login.

6. A name known to the local operating system.

(iv) status — Completion status.

(e) *Description:* This function is called by a server to validate an incoming ticket and the associated authenticator, and obtain the client identity and conversation key. There is an option to verify an authenticator only.

(f) *Notes on implementation on top of DCE security facilities:*

(i) If only the authenticator needs to be verified:

[a] Get the conversation key identified by conv_token from the key manager.

[b] Decrypt and verify the authenticator.

[c] Extract the DCE principal name from the authenticator.

[d] If there is a subkey included in the authenticator, pass it to the key manager for storage under conv_token.

(ii) Otherwise:

[a] Get the server's key from the key manager.

[b] Decrypt the ticket.

[c] Check that the ticket has not expired.

[d] Decrypt the authenticator with the conversation key in the ticket. Verify the authenticator to ascertain that the ticket is not a replay.[7]

[e] Extract the privilege attribute certificate (PAC) from the ticket.

[f] Extract the (local userid, system id) pair from the PAC and check that the system id matches the id of the local system.[8]

[g] Translate the principal UUID in the PAC to the corresponding principal name.

[h] Pass the conversation key to the key manager for storage.

[i] If a subkey is included in the authenticator, it is the actual conversation key.

[ii] Otherwise, extract the conversation key from the ticket.

[i] The key manager returns a conv_token.

2.3.2.3. Build authentication reply

(a) *Function name:* edb_build_auth_reply

(b) *Input parameters:*

(i) caller_token — Token for calling server.

(ii) principal_name — DCE principal name associated with target client.

(iii) conv_token — Conversation identifier.

(c) *Output parameters:*

(i) auth_reply — Authentication reply.

(ii) status — Completion status.

---

7. It is assumed that a replay cache is not required.

8. The local userid is typically a character string that represents the user on the local system. It identifies the user for the purposes of access control and other system functions.

(d) *Description:* This function is called by a server to build an authentication reply used to complete a mutual authentication exchange with a target client.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Build an authentication reply.

(ii) Get the conversation key identified by conv_token from the key manager.

(iii) Encrypt the authentication reply.

2.3.2.4. Parse authentication reply

(a) *Function name:* edb_parse_auth_reply

(b) *Input parameters:*

(i) caller_token — Token for calling client.

(ii) auth_reply — Authentication reply received.

(iii) conv_token — Conversation identifier.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a client to validate an incoming authentication reply from a target server.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Get the conversation key identified by conv_token from the key manager.

(ii) Decrypt the authentication reply.

(iii) Verify the authentication reply.

2.3.2.5. Get delegation token

(a) *Function name:* edb_get_delegation

(b) *Input parameters:*

(i) caller_token — Token for calling client.

(ii) login_context — Pointer to client's login context.

(iii) server_name — Name of target server.

(iv) conv_token — Conversation identifier.

(c) *Output parameters:*

(i) deleg_env — Envelope containing delegation token and associated conversation key between client and delegation server.

(ii) status — Completion status.

(d) *Description:* This function is called by a client to get a delegation token and the associated conversation key between the client and the delegation token server (DS key) to be forwarded to a target server. Typically, the client would have completed a mutual authentication exchange with the server before forwarding the delegation token.

(e) *Notes on implementation on top of DCE security facilities:*

This function assumes that the calling client, or the user on whose behalf it acts, has logged in to DCE[9] so that a login context is available.

(i) Get a PTGT for the cell of the target server, together with the associated PS key.

[a] If there is an unexpired PTGT in the client's credentials cache that satisfies the requirements, it will be returned along with the associated PS key.

[b] Otherwise, a PTGT and PS key will be obtained from the DCE security server.

(ii) Get the conversation key identified by conv_token from the key manager.

(iii) Encrypt the PTGT and PS key to give the PTGT envelope.

2.3.2.6. Receive delegation token

(a) *Function name:* edb_receive_delegation

(b) *Input parameters:*

(i) caller_token — Token for calling server.

(ii) deleg_env — Envelope containing delegation token and associated conversation key to the delegation token server.

(iii) conv_token — Conversation identifier.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a server to store an incoming delegation token and the associated conversation key to the delegation token server (DS key).

(e) *Notes on implementation on top of DCE security facilities:*

(i) Get the conversation key identified by conv_token from the key manager.

(ii) Decrypt the PTGT envelope to give the PTGT and the associated PS key.

(iii) Pass the PTGT and PS key to the key manager for storage. They will be stored with the conversation key identified by conv_token.

2.3.2.7. Get ticket for client

(a) *Function name:* edb_get_client_ticket

(b) *Input parameters:*

(i) caller_token — Token for calling server.

(ii) conv_token — Conversation identifier.

(iii) server_name — Name of target server.

(iv) ticket_req — = 1 if both ticket and authenticator are required, = 0 if only authenticator is required.

(c) *Output parameters:*

(i) ticket — Client's ticket for target server (null if ticket_req = 0, unless the previous ticket has expired).

(ii) authent — Authenticator.

---

9. Via dce_login, the DCE sec_login API (see [OSF]), or edb_trusted_login.

(iii) status — Completion status.

(d) *Description:* This function is called by a server on behalf of one its clients to get a ticket and an authenticator for another target server. There is an option to get an authenticator only.

(e) *Notes on implementation on top of DCE security facilities:*

(i) If only an authenticator is required:

[a] Get the client ticket and conversation key identified by conv_token from the key manager.

[b] If the client ticket has expired:

[i] Get the client PTGT and PS key identified by conv_token from the key manager.

[ii] Get a credential (ticket and new conversation key) for the target server, by presenting the client PTGT to the DCE security server. Note that the PS key is used to encrypt the envelope containing the PTGT before it is sent to the DCE security server.

[iii] Pass the client ticket to the key manager for storage under conv_token.

[c] Build an authenticator that includes the conversation key identified by conv_token as the subkey. This conversation key is used for subsequent communication between the calling server and the target server.

[d] Encrypt the authenticator in the new conversation key.

(ii) Otherwise:

[a] Get the conversation key, client PTGT, and PS key identified by conv_token from the key manager.

[b] Get a credential (ticket and new conversation key) for the target server, by presenting the client PTGT to the DCE security server. Note that the PS key is used to encrypt the envelope containing the PTGT before it is sent to the DCE security server.

[c] Pass the client ticket to the key manager for storage under conv_token.

[d] Build an authenticator that includes the conversation key identified by conv_token as the subkey. This conversation key is used for subsequent communication between the calling server and the target server.

[e] Encrypt the authenticator in the new conversation key.

2.3.3. Functions for authorized callers only

2.3.3.1. Trusted login

(a) *Function name:* edb_trusted_login

(b) *Input parameters:*

(i) caller_token — Token for calling client.

(ii) userid — Local userid.[10]

(c) *Output parameters:*

(i) login_context — Pointer to user's login context.

(ii) status — Completion status.

---

10. A name known to the local operating system.

(d) *Description:* This function is called by a trusted client to log a user in to DCE.

(e) *Notes on implementation on top of DCE security facilities:*

This function assumes that: (1) The DCE registry in the cell has been integrated with the local machine's registry. (2) The calling client has registered with the key manager.

(i) Check that the caller is authorized.

(ii) Obtain the system id (machine principal id) of the local system.

(iii) Call the extended sec_idmap interface[11] to map the (local userid, system id) pair to a DCE principal name. The mapping information is stored in the integrated DCE registry.

(iv) Request a ticket-granting ticket (TGT) for the principal from the DCE security server.

(v) Get the local user's key from the local registry.

(vi) Decrypt the TGT envelope with the key. The envelope contains, in addition to the TGT, an associated conversation key.

(vii) Request a PTGT for the principal from the DCE security server.

(viii) Decrypt the PTGT envelope with the TGS conversation key.

(ix) Set up a login context for the user.

2.3.3.2. Trusted logout

(a) *Function name:* edb_trusted_logout

(b) *Input parameters:*

(i) caller_token — Token for calling client.

(c) *Input-output parameters:*

(i) login_context — Pointer to user's login context.

(d) *Output parameters:*

(i) status — Completion status.

(e) *Description:* This function is called by a trusted client to log a user out of DCE.

(f) *Notes on implementation on top of DCE security facilities:*

(i) Purge the user's login context.

(ii) Purge the user's credentials cache.

2.4. Communications Integrity

2.4.1. Functions for general callers

2.4.1.1. Build verifier

(a) *Function name:* edb_build_verifier

(b) *Input parameters:*

(i) caller_token — Token for calling client or server.

(ii) length — Length of data.

---

11. Provided as part of local registry integration.

(iii) data — Pointer to data to be sent.

(iv) conv_token — Conversation identifier.

(v) verifier — Pointer to buffer for verifier.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a client or a server to build an integrity verifier for data to be sent.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Compute an MD5 digest over the input data.

(ii) Get the conversation key identified by conv_token from the key manager.

(iii) Encrypt the digest using DES in cipher block chaining mode to give the verifier.

2.4.1.2. Check verifier

(a) *Function name:* edb_check_verifier

(b) *Input parameters:*

(i) caller_token — Token for calling client or server.

(ii) length — Length of data.

(iii) data — Pointer to data received.

(iv) verifier — Verifier received.

(v) conv_token — Conversation identifier.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a client or a server to check the integrity of data received.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Compute an MD5 digest over the input data.

(ii) Get the conversation key identified by conv_token from the key manager.

(iii) Encrypt the digest using DES in cipher block chaining mode to give a verifier.

(iv) Compare this verifier with the verifier received.

2.5. Communications Confidentiality

2.5.1. Functions for general callers

2.5.1.1. Encrypt data

(a) *Function name:* edb_encrypt_data

(b) *Input parameters:*

(i) caller_token — Token for calling client or server.

(ii) length — Length of data.

(iii) data — Pointer to data to be encrypted.

(iv) conv_token — Conversation identifier.

(v) enc_length — Length of buffer for encrypted data.

(vi) enc_data — Pointer to buffer for encrypted data.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a client or a server to encrypt data to be sent. It also builds a verifier for integrity checking purposes.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Compute a CRC-32 checksum[12] over the encrypted data.

(ii) Get the conversation key identified by conv_token from the key manager.

(iii) Encrypt the checksum followed by the input data using DES in cipher block chaining mode to give enc_data.

2.5.1.2. Decrypt data

(a) *Function name:* edb_decrypt_data

(b) *Input parameters:*

(i) caller_token — Token for calling client or server.

(ii) length — Length of data.

(iii) data — Pointer to data to be decrypted.[13]

(iv) conv_token — Conversation identifier.

(v) dec_length — Length of buffer for decrypted data.

(vi) dec_data — Pointer to buffer for decrypted data.

(c) *Output parameters:*

(i) status — Completion status.

(d) *Description:* This function is called by a client or a server to decrypt data received and check its integrity.

(e) *Notes on implementation on top of DCE security facilities:*

(i) Get the conversation key identified by conv_token from the key manager.

(ii) Decrypt the data received to give dec_data.

(iii) Separate the checksum from the body of the data.

(iv) Compute a CRC-32 checksum[14] over the body of the data.

(v) Compare the checksum with the checksum received.

APPENDIX A. GLOSSARY

(a) *Automatic key management*

Automatic renewal of a server's key when it expires. This involves generating a new key randomly, storing it in both local key storage and the DCE registry. Each expired key is retained in local key storage for a period of time sufficient for all tickets that could have encrypted with that key to expire.

---

12. A CRC-32 checksum is considered to be acceptable since it is more difficult to tamper with encrypted data.
13. Since this data was generated by edb_encrypt_data, it includes a checksum for integrity checking purposes.
14. A CRC-32 checksum is considered to be acceptable since it is more difficult to tamper with encrypted data.

(b) *Client*

A process that makes use of a network service on behalf of a user. Note that in some cases a server may itself be a client of some other server (e.g., a print server may be a client of a file server).

(c) *Envelope*

An encrypted data structure.

(d) *Principal*

A client, server or machine instance that participates in secure network communication. Principals are represented by entries in the DCE registry.

REFERENCES

[OSF] *OSF DCE Version 1.0 DCE Application Development Guide*, Revision 1.0, December 31, 1991.

[RFC 3.0] Joe Pato, ‘‘Extending the DCE Authorization Model to Support Practical Delegation (Extended Summary)’’, June 1992.

AUTHORS’ ADDRESSES

Ping Lin
Distributed Software Products
IBM Canada Laboratory
Stn. 2G, 1150 Eglinton Avenue East
Toronto, Ontario
CANADA M3C 1H7
Internet email: plin@torolab5.vnet.ibm.com
Telephone: +1-416-448-3942

Mark Leitch
Distributed Software Products
IBM Canada Laboratory
Stn. 2G, 1150 Eglinton Avenue East
Toronto, Ontario
CANADA M3C 1H7
Internet email: mleitch@torolab5.vnet.ibm.com
Telephone: +1-416-448-4431