LibEST

3.2.0p

Generated by Doxygen 1.8.11

Contents

Chapter 1

Introduction to libEST

This document describes libEST, a library that implements the Enrollment over Secure Transport protocol (RFC 7030). This protocol is a replacement for SCEP. It allows X509 certificates to be provisioned on end-user devices and network infrastructure devices. Users of libEST are encouraged to read RFC 7030 prior to using the API. See the **Supported Features** (p. 2) section for more detailed information.

libEST provides a subset of the EST specification. The following message flows are provided:

- cacerts
- csrattrs
- simpleenroll
- simiplereenroll
- serverkeygen

This document is organized as follows. The first chapter provides background material on EST and an overview of libEST. The subsequent chapters provide a detailed reference to the libEST API and related functions. The reference material is created automatically (using the doxygen utility) from comments embedded in some of the C header files.

1.1 License and Disclaimer

libEST is distributed under the following license, which is included in the source code distribution. It is reproduced in the manual in case you attained the library from another source.

1.2 Supported Features

The EST specification defines requirements for enrolling X509 certifications for end-entities, Registration Authorities (RA), and Certificate Authorities (CA). There are six message flows described in the EST specification. libEST supports a subset of these message flows, but is designed to operate as a client, proxy, or server. libEST supports the following five EST message flows:

- cacerts
- csrattrs
- simpleenroll
- simiplereenroll
- serverkeygen

The following message flows are currently not supported:

- fullcmc

The user should be aware that it is possible to misuse this library, which may result in inadequate security. If you are implementing a feature using this library, you will want to read the Security Considerations section of the EST specification (RFC 7030). In addition, it is important that you read and understand the terms outlined in the **License and Disclaimer** (p. 1) section.

1.3 Installing and Building libEST

To install libEST, download the latest release of the distribution. The naming convention of the tarball is `libEST-A.B.gz`, where `A` is the major release number and `B` is the minor release number, and `gz` is the file extension. Users are encouraged to use the most recent release. Unpack the distribution and extract the source files; the directory into which the source files will go is named `libEST-A-B`.

libEST uses the GNU `autoconf` and `make` utilities[1]. In the `libest` directory, run the configure script and then make:

```
./configure [ options ]
make
make install
```

This package has been tested on both 32 and 64 bit Linux systems running CentOS and Ubuntu. The OpenSSL development package should be installed on the system. libEST is designed to work with the OpenSSL 1.0.2 API or above. You can use the –with-ssl-dir option with the ./configure command to specify the location of the OpenSSL installation on the host system.

1.3.1 Static Linking

By default, both the shared and statically linked version of libEST are built. If only the static version is to be used, the –disable-shared option should be used with ./configure. This will disbale the generation of the shared version of the libEST library, forcing the linker to use the statically linked version.

1.3.2 LibCoAP Library

libEST is dependent upon the libcoap library to communicate via CoAP. The libcoap library must first be obtained from Github (https://github.com/obgm/libcoap/releases). Cisco specific changes exist that must be applied to libcoap before it is built and installed. The patch containing the Cisco specific changes to libcoap is libcoap.cisco.xxx.patch (xxx set to the specific build version) and is found in the top level directory of the libEST code base.

```
tar -xvf libcoap-4.2.0-rc3.tar.gz
cd libcoap-4.2.0-rc3/libcoap
cp <install path to libEST code base>/libcoap.cisco.109.patch .
patch -i libcoap.cisco.109.patch -p 1
```

Libcoap needs to built using OpenSSL 1.1.0 or greater. The OpenSSL version used to build this dependency should be the same as the version used when building libEST. Once the Cisco specific changes have been applied to the libcoap source, perform the following commands to build and install:

```
export _OPENSSL_DIR=<OpenSSL 1.1.x+ install dir>
export CFLAGS="-I${_OPENSSL_DIR}/include"
export LDFLAGS="-L${_OPENSSL_DIR}/lib"
export PKG_CONFIG_PATH="${_OPENSSL_DIR}/lib/pkgconfig"
./configure [--prefix=<install-dir>] --disable-documentation --with-openssl\
 --with-cisco
make
make install
```

[1] BSD make may not work; if both versions of make are on your platform, you may have to invoke GNU make as `gmake`.

If the ./configure was not present, run the following command and rerun the above commands:

```
./autogen.sh
```

When configuring the libEST build, the –with-libcoap-dir option is used to point to the installed libcoap library to be used. If the prefix option was used to specify a new path to install to, make sure to add the libcoap lib folder to LD_LIBRARY_PATH so that it can be linked in at runtime. See the example application code for an example application using this feature's APIs.

1.3.3 Libcurl Library

libEST can be built with libcurl to allow libEST to be used in SOCKS/HTTP proxy mode. libEST has been built and tested with the following versions of libcurl:

- 7.60.0
- 7.62.0
- 7.64.0

Libcurl needs to built using OpenSSL 1.0.2 or greater. The OpenSSL version used to build this dependency should be the same as the version used to build libEST. To perform this build of libcurl on Linux download the libcurl tarball from the libcurl download page, expand it, and run the following commands:

```
export _OPENSSL_DIR=<OpenSSL 1.0.2+ install dir>
export CPPFLAGS="-I$_OPENSSL_DIR/include"
export LDFLAGS="-L$_OPENSSL_DIR/lib"
export LD\_LIBRARY\_PATH="$_OPENSSL_DIR/lib"
./configure [--prefix=<install-dir>] --with-ssl=${_OPENSSL_DIR}
make
make install
```

When configuring the libEST build, the –with-libcurl-dir option is used to point to the installed libcurl library to be used. If the prefix option was used to specify a new path to install to, make sure to add the libcurl lib folder to LD_LIBRARY_PATH so that it can be linked in at runtime.

1.3.4 URIParser Library

libEST can be built with uriparser to allow libEST to support the use of an URI path segment inside an EST request. libEST has been built and tested with the following versions of uriparser:

- 0.8.5
- 0.9.1

To build uriparser on Linux download the uriparser tarball from the uriparser download page, expand it, and run the following commands:

```
./configure [--prefix=<install-dir>]
make
make install
```

When configuring the libEST build, the –with-uriparser-dir option is used to point to the installed uriparser library to be used. If the prefix option was used to specify a new path to install to, make sure to add the uriparser lib folder to LD_LIBRARY_PATH so that it can be linked in at runtime.

1.3.5 Release builds

By default libEST includes debug symbols in the target binary. To perform a production release build that does not contain debug symbols, the following procedure can be used:

```
./configure [ options ]
make
make install-strip
```

1.3.6 Java build

NOTE: JEST is no longer supported. The code and the following instructions are included for teams that wish to use it as a starting point for developing their own JNI to LibEST.

libEST is used by JEST, which provides Java support for EST client-side operations. JEST requires the libEST libary (libest.so). JEST provides a Java wrapper class to invoke the client-side API available in libest.so. JEST consists of a native JNI library (libjest.so) and an associated Jar file containg the wrapper class. This allows your Java applications to leverage libEST (via JEST).

If you plan to use JEST for Java support, you will still need to build libEST. Use the `-enable-jni` option when configuring libEST. When this option is used, the libciscojest.so target is built in addition to the libest.so target. libjest.so is installed into the same directory as libest.so (/usr/local/est/lib by default). You will need to ensure the LD_LIBRARY_PATH is configured properly for your JVM to include the libEST and JEST native libraries.

The following procedure shows how to build and install libEST with Java support:

```
./configure --enable-jni
make
make install
```

The JEST Jar file is distributed separately from libEST. The Jar file should be included into your JVM using the CLASSPATH setting. Please refer to the JEST API documentation for using JEST within your Java application.

1.4 Example Code

Several example applications are included in the release tarball. These are located in the `/example` subdirectory.

Example Applications	Function tested
client-simple	Example EST client application using the easy provision API
client	Example EST client application using the granular API
server	Example EST server application using OpenSSL CA
proxy	Example EST proxy application acting as an RA

There are README files in each of the example subdirectories. The samples are built automatically when libEST is configured and compiled. When running the example server application, there is a script called createCA.sh that must be run once to create the root CA certificate and certificate chain that is required for running the EST example server. This script uses the host-resident copy of OpenSSL on the system to create a CA database. Your host-resident copy of OpenSSL should provide support for ECC operations.

Note, the example code is not suitable for including into commercial software. The examples are primarily used to test the libEST library and to show how the API should be used. The example code does not have all the appropriate error handling and robust design required for commercial software. For example, the OpenSSL CA used by the example EST server is not thread safe and provides no means to enforce certificate policy.

1.5 EST Overview

EST provides for a standards based mechanism to provision X509 certificates from either an RA or CA. An entity that desires to attain a valid X509 certificate bound to a trust point acts as an EST client and communicates with either an RA or a CA to request and receive an X509 certificate from a trusted CA. libEST provides both client-side and server-side capabilities to operate as any role in the EST enrollment process (end-entity, RA, or CA). It is important to note that libEST is not a CA. libEST is only the EST stack, which can be used by a CA to comply with RFC 7030.

The EST specification requires the use of TLS for transport. This allows all the entities participating in the enrollment process to verify the identity of each other. TLS also provides privacy on the communication channel between the EST client and RA/CA. Using the simple case of an EST client communicating with an EST enabled CA server, the following minimal setup is required prior to initiating the enrollment process:

- A server certificate must be provisioned for the EST enabled CA server. This allows the CA to identify itself to EST clients.
- The certificate chain used to sign the server certificate must be provided to libEST on the server side.
- The root certificate of the chain used to sign the server certificate should be pre-provisioned on the EST client. This allows the EST client to verify the identity of the EST enabled CA server[2]. If this root certificate can not be pre-provisioned, then a fingerprint of the EST server identity certificate can be used by the EST client to verify the EST server identity.

With the above setup requirements satisfied, the enrollment process consists of the following procedure:

- The EST client issues the /cacerts request to the server. During this process the EST client verifies the server identity using the pre-provisioned root certificate described above, or using a fingerprint of the server certificate.
- The EST client replaces the pre-provisioned trusted root certificates with the certificates provided by the EST enabled CA server in the /cacerts response. This new certificate chain is used for all future client-side EST operations. The client should persist this new trust chain for future use, such as certificate renewal.
- The EST client issues the /csrattrs request to the server. The server responds by providing X509 attributes that should be included in the forthcoming PKCS10 CSR request. The required X509 attributes are derived from the certificate policy configured on the CA.
- The EST client generates a PKCS10 CSR and includes any specific X509 attributes required by the server based on the response in the /csrattrs request.
- The EST client sends the PKCS10 CSR to the server using the /simpleenroll request. The EST enabled CA server will sign the CSR and return the X509 certificate to the client.

[2] This statement does not address the bootstrap enrollment process. The bootstrap method is supported by libEST, but requires additional configuration.

1.6 EST Client Operation

Applications that need to provision X509 certificates can utilize libEST in client mode. libEST provides an API that performs the steps in the enrollment process described in the **EST Overview** (p. 6) section. The first step when using the API is to create an EST context. Once created, the context is used for subsequent API calls into the libEST library. The EST specification allows for several authentication methods. This includes HTTP basic authentication, HTTP digest authentication, and certificate based authentication. The example client code provided with the libEST distribution tarball shows how to use the various capabilities in the libEST API, including the various authentication methods.

There are two methods for provisioning a certifice while operating in client mode. The first is the easy provision API, which provides a simplified API that encapsulates all the client-side EST operations into a single enroll operation. The second method is to use the granular API for performing the /cacerts, /csrattrs, and /simpleenroll operations individually. The easy provision API is limited in that the client can only specify the X509 Common Name value in the certificate request. If the CA certificate policy is configured such that additional X509 attributes are required to be supplied by the client, then the easy provision API should not be used.

1.6.1 Easy Provision API

The easy provision API provides a simplied entry point into libEST to enroll a certificate request with a CA. This API encapsulates the recommended steps as defined in RFC 7030, which includes:

- Attain a fresh copy of the EST server CA certificates.
- Request the CSR attributes from the EST server to determine if proof-of-possession is required.
- Issue the simple enroll request to the EST server.

The following procedure outlines the steps to utilize the easy provision API:

1. Invoke the est_apps_startup() helper function, which initializes the OpenSSL library. This may not be required if your application has already initialized OpenSSL elsewhere.
2. Invoke **est_client_init()** (p. 35) to attain an EST context. Use the pre-provisioned trusted root certificates.
3. Invoke **est_client_set_auth()** (p. 38) to specify the HTTP username and password to be used. Optionally, an existing X509 certificate can be used to identify the client to the EST server. For example, a previously issued certificate or a bootstrap certificate could be used. The HTTP username is optional when a client certificate is provided.
4. Invoke **est_client_set_server()** (p. 41) to specify the IP address of the EST server, which may be either an RA or CA.
5. Invoke **est_client_provision_cert()** (p. 36) to create a PKCS10 CSR and forward it to the EST server, expecting the EST server to return the signed X509 certificate.
6. Invoke **est_client_copy_enrolled_cert()** (p. 28) to copy the X509 cert provided by the EST server to local storage.
7. Invoke **est_client_copy_cacerts()** (p. 28) to retrieve the latest trust chain that was provided by the EST server. These certs should be persisted for future use.
8. Invoke **est_destroy()** (p. 20) to remove the EST context.

1.6.2 Full Provision API

While the easy provision API is easier to use, it does not allow the application to specify X509 attributes in the PKCS10 CSR other than the Common Name. Because the CA may be configured with policy requiring the certificate requestor to provide additional X509 attributes in the CSR, the easy provision API may not be suitable for your application. When this occurs, libEST provides a more granular API that allows your application to create the PKCS10 CSR itself, rather than having libEST create the CSR on behalf of your application. This allows your application to include any required X509 attributes in the CSR. The following steps outline the procedure for using the granular API:

1. Invoke the est_apps_startup() helper function, which initializes the OpenSSL library. This may not be required if your application has already initialized OpenSSL elsewhere.
2. Invoke **est_client_init()** (p. 35) to attain an EST context. Use the pre-provisioned trusted root certificates.
3. Invoke **est_client_set_auth()** (p. 38) to specify the HTTP username and password to be used. Optionally, an existing X509 certificate can be used to identify the client to the EST server. For example, a previously issued certificate or a bootstrap certificate could be used. The HTTP username is optional when a client certificate is provided.
4. Invoke **est_client_set_server()** (p. 41) to specify the IP address of the EST server, which may be either an RA or CA.
5. Invoke **est_client_get_cacerts()** (p. 34) and **est_client_copy_cacerts()** (p. 28) to retrieve the latest trust chain from the EST server.
6. Invoke **est_destroy()** (p. 20) to clean the EST context.
7. Invoke **est_client_init()** (p. 35) to attain a new EST context. Unlike step 2, we use the new CA certs just attained in step 5. Repeat steps 3 and 4 again too.
8. Invoke **est_client_get_csrattrs()** (p. 34) to retrieve the required X509 attributes from the CA.
9. Generate a public/private key pair if needed.
10. Using OpenSSL, generate a PKCS10 CSR and populate any required CSR attributes based on the response in step 8.
11. Invoke **est_client_enroll_csr()** (p. 32) to send the PKCS10 CSR to the CA and receive the X509 certificate.
12. Invoke **est_client_copy_enrolled_cert()** (p. 28) to copy the X509 cert provided by the EST server to local storage.
13. Invoke **est_destroy()** (p. 20) to remove the EST context.

The procedure outlined above describes the general use case for provisioning a certificate. Please be aware the EST specification allows for many variations. Other considerations include:

- Handling the retry-after response when the CA isn't configured for automatic approval.
- Manual verification of the CA server when trusted root certificates are not pre-provisioned on the EST client.
- Bootstrap use case where multiple trust anchors are used.
- The re-enrollment process to renew a certificate.
- Enable CRL checks when verifying the server identity.
- Provide thread locking mechanism required by OpenSSL.

1.7 EST Server Operation

The EST specification covers a broad spectrum of the OSI model. EST places requirements on TLS, HTTP, and the certificate authority (CA). libEST is not a CA. libEST only implements the TLS and HTTP layers as defined in RFC 7030. It is the responsibility of your application to bind the CA server to libEST. Additionally, libEST does not implement the TCP server required to accept incoming requests. Your application must initialize and manage the TCP sockets for incoming connections. Please reference EDCS-1265219 for a more detailed discussion on the architecture.

When implementing an EST server your application is responsible for binding the EST stack to a CA. The application is also responsible for opening the TCP sockets that will receive incoming connections. An overview of implementing a server is:

- Initialize an EST context. The application provides the server certificate and private key used to identify itself to any EST clients. The application also provides the trusted certificate chain to be used when responding to the /cacerts requests.
- Specify the callback function that libEST will use to process an incoming CSR from the client. This callback function is the glue that binds libEST to the CA. This function will forward the CSR to the CA and return the response to libEST. Your application must implement this callback function.
- Specify the callback function that libEST will invoke to verify the client identity when HTTP basic or digest authentication is used. This callback function would verify the user's identity and return a response to libEST. A typical use case would be to utilize a RADIUS server from within callback function. Your application is responsible for hooking into the desired authentication server, such as RADIUS, LDAP, OAuth, etc. libEST currently does not contain support for authentication protocols. In the future libEST may include built-in support for RADIUS, LDAP, etc.
- Open one or more listening TCP sockets. These sockets must be managed by the application. An example of doing this would be to write an NGINX module that utlized NGINX to manage the listening sockets, while forwarding incoming requests on these sockets to libEST. Another example would be to write a custom multi-threaded TCP server to accept incoming socket requests and forward them to libEST.

The following API calls would be invoked to implement an EST server. These represent the minimal configuration to implementing an EST server:

1. Invoke the est_apps_startup() helper function, which initializes the OpenSSL library.
2. Invoke **est_server_init()** (p. 57) to attain a new EST context. The server's certificate and private key are provided to this call.
3. Invoke **est_set_ca_enroll_cb()** (p. 63) to provide a callback function to libEST that is invoked by libEST when a PKCS10 CSR needs to be signed by the CA. Your application must implement this callback function to forward the CSR to the CA.
4. Invoke **est_set_csr_cb()** (p. 64) to provide a callback function to libEST that is invoked by libEST when requesting the CSR attributes that should be included by EST clients during the enrollment process.
5. Invoke **est_server_start()** (p. 61) to initialize the HTTPS services. This step is required to setup the TLS context within the HTTPS layer prior to receiving EST requests in step 7.
6. Open a TCP listening socket and wait for incoming requests.
7. Accept new incoming connections on the socket and invoke **est_server_handle_request()** (p. 70).

The procedure outlined above describes the general use case for implementing a simple EST server operating as either an RA or CA. The EST specification allows for many variations. libEST provides additional API entry points to customize the behavior of the server. Other considerations include:

- Specify the retry period when the CA is configured for manual approval.
- Enable CRL checks when verifying the client identity.
- Configuring either HTTP basic, HTTP digest, or no HTTP authentication.
- Disabling the proof-of-possession check.
- Implementing a multi-threaded EST server to allow for simultaneous processing of EST requests.
- Providing support for the EST /simplereenroll flow.
- Provide thread locking mechanism required by OpenSSL.
- Enable TLS single-use DH key generation.

1.8 EST Proxy Operation

libEST provides the ability to operate as an EST proxy. An EST proxy receives incoming EST requests and forwards outgoing EST requests to either an RA or CA. A typical use case for EST proxy mode would be to deploy the proxy on a VPN head-end device that is dual-homed on both a public and private network. This would allow EST requests orginating from VPN clients on the public Internet to be forwarded to an RA or CA residing on a private network.

Using libEST in proxy mode is similar to server mode operation. The application is responsible for implementing the TCP listener and managing the TCP connections. However, there is no need to integrate a CA with libEST when operating in proxy mode, which makes proxy mode suitable for operating as an RA. The following steps outline the minimal API calls required to implement an EST proxy:

1. Invoke the est_apps_startup() helper function, which initializes the OpenSSL library.
2. Invoke **est_proxy_init()** (p. 45) to attain a new EST context. The proxy server certificate and private key are provided to this call. This certificate is used to identify the proxy to both the client and to the RA/CA.
3. Invoke **est_proxy_set_server()** (p. 47) to specify the IP address and port of the RA/CA.
4. Invoke **est_proxy_start()** (p. 47) to initialize the HTTPS services. This step is required to setup the TLS context within the HTTPS layer prior to receiving EST requests in step 6.
5. Open a TCP listening socket and wait for incoming requests.
6. Accept new incoming connections on the socket and invoke **est_server_handle_request()** (p. 70).

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 file Struct Reference

Data Fields

- int **is_directory**
- time_t **modification_time**
- int64_t **size**
- FILE * **fp**
- const char * **membuf**

The documentation for this struct was generated from the following file:

- src/est/**est_server_http.c**

4.2 vec Struct Reference

Data Fields

- const char * **ptr**
- size_t **len**

The documentation for this struct was generated from the following file:

- src/est/**est_server_http.c**

Chapter 5

File Documentation

5.1 src/est/est.c File Reference

```
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include "est.h"
#include "est_locl.h"
#include "est_ossl_util.h"
#include "safe_mem_lib.h"
#include "safe_str_lib.h"
#include <execinfo.h>
```

Macros

- #define **MAX_FINISHED** 100

Functions

- void **est_log** (EST_LOG_LEVEL lvl, char ∗format,...)
- void **est_log_backtrace** (void)
- const char ∗ **est_get_version** (void)

  est_get_version() *(*p. *23) allows the application to retrieve the libEST version string. Returns a char∗ array containing the full version string value for the library.*
- int **est_get_api_level** (void)

  est_get_api_level() *(*p. *22) allows the application to retrieve the libEST API level. This is a numeric value that indicates the API level of the library. When new versions of libEST are released and the API changes, this value will be incremented. Applications can use this to determine which capabilities in the libEST library should or should not be attempted.*
- void **est_log_version** (void)
- EST_ERROR **est_init_logger** (EST_LOG_LEVEL lvl, void(∗loggerfunc)(char ∗, va_list))

  est_init_logger() *(*p. *23) allows the application to override the default log handler for EST logging messages.*

- void **est_enable_backtrace** (int enable)

  est_enable_backtrace() *(p. 21) allows the application to toggle whether the stack trace is displayed for WARNING and ERROR log messages coming from libEST.*
- X509_REQ * **est_read_x509_request** (unsigned char *csr, int csr_len, EST_CERT_FORMAT csr_format)

  est_read_x509_request() *(p. 24) is a helper function that reads a char* and converts it to an OpenSSL X509_REQ*. The char* data can be either PEM or DER encoded.*
- EVP_PKEY * **est_load_key** (unsigned char *key, int key_len, int format)

  est_load_key() *(p. 23) is a helper function that reads a char* and converts it to an OpenSSL EVP_PKEY*. The char* data can be either PEM or DER encoded.*
- EST_ERROR **est_load_ca_certs** (EST_CTX *ctx, unsigned char *raw, int size)
- EST_ERROR **est_load_trusted_certs** (EST_CTX *ctx, unsigned char *certs, int certs_len)
- EST_ERROR **est_set_ex_data** (EST_CTX *ctx, void *ex_data)

  est_set_ex_data() *(p. 24) sets the application specific data on the EST context.*
- void * **est_get_ex_data** (EST_CTX *ctx)

  est_get_ex_data() *(p. 22) retrieves the application specific data on the EST context.*
- EST_ERROR **est_destroy** (EST_CTX *ctx)

  est_destroy() *(p. 20) frees an EST context*
- int **est_base64_decode** (const char *src, char *dst, int dst_size)
- int **est_base64_encode** (const char *src, int actual_src_len, char *dst, int max_dst_len, int nl)
- char * **est_get_tls_uid** (SSL *ssl, int *uid_len, int is_client)
- void **est_hex_to_str** (char *dst, unsigned char *src, int len)
- EST_ERROR **est_enable_crl** (EST_CTX *ctx)

  est_enable_crl() *(p. 21) is used by an application to enable checking of a certificate revocation list when validating the client TLS peer certificate during the TLS handshake. When enabled, the ca_chain parameter provided to either* ***est_server_init()*** *(p. 57) or* ***est_client_init()*** *(p. 35) should contain both the trusted certificates along with the CRL entries. The CRL entries should be appened at the end.*
- EST_ERROR **est_is_challengePassword_present** (const char *base64_ptr, int b64_len, int *presence)
- EST_ERROR **est_asn1_parse_attributes** (const char *p, int len, int *pop_present)
- EST_ERROR **est_add_challengePassword** (const char *base64_ptr, int b64_len, char **new_csr, int *pop_len)
- int **est_X509_REQ_sign** (X509_REQ *csr, EVP_PKEY *pkey, const EVP_MD *md)

  est_X509_REQ_sign() *(p. 25) Sign an X509 certificate request using the digest and the key passed. Returns OpenSSL error code from X509_REQ_sign_ctx();*
- EST_ERROR **est_add_attributes_helper** (X509_REQ *req, int nid, void *string, int chtype)

  est_add_attributes_helper() *(p. 19) Add a NID and its character string to an X509_REQ as an attribute.*
- EST_ERROR **est_decode_attributes_helper** (char *csrattrs, int csrattrs_len, unsigned char **der, int *len)

  est_decode_attributes_helper() *(p. 19) Decode a base64 encoded string into DER format(ASN.1 hex).*
- EST_ERROR **est_get_attributes_helper** (unsigned char **der_ptr, int *der_len, int *new_nid)

  est_get_attributes_helper() *(p. 22) get attributes NID from a DER encoded string.*
- void **cleanse_auth_credentials** (EST_HTTP_AUTH_HDR *auth_cred)
- EST_OPERATION **est_parse_operation** (char *op_path)
- EST_ERROR **est_parse_uri** (char *uri, EST_OPERATION *operation, char **path_seg)
- EST_ERROR **est_store_path_segment** (EST_CTX *ctx, char *path_segment, int path_segment_len)
- int **est_strcasecmp_s** (char *s1, char *s2)
- size_t **est_strcspn** (const char *str1, const char *str2)
- size_t **est_strspn** (const char *str1, const char *str2)
- char * **skip_quoted** (char **buf, const char *delimiters, const char *whitespace, char quotechar)
- char * **skip** (char **buf, const char *delimiters)
- EST_ERROR **est_enable_performance_timers** (EST_CTX *ctx)

est_enable_performance_timers() *(p. 21) is used by an application to enable the use and output of the libest performance timers for the given est context. When enabled these timers will output logs that can be parsed to give you performance metrics of different areas of libest as they are executed.*

- EST_ERROR **est_disable_performance_timers** (EST_CTX *ctx)

  est_disable_performance_timers() *(p. 20) is used by an application to disable the use and output of the libest performance timers for the given est context.*
- int **start_timer** (EST_TIMER *timer, EST_CTX *ctx, char *tag)
- void **start_http_req_timer** (EST_TIMER *timer, EST_CTX *est_ctx, EST_OPERATION op)
- int **stop_timer** (EST_TIMER *timer)
- int **stop_timer_with_id** (EST_TIMER *timer, char *id)
- void **null_timer** (EST_TIMER *timer)
- unsigned char **is_same_time** (struct timeval *time1, struct timeval *time2)
- unsigned char **is_started** (EST_TIMER *timer)
- unsigned char **is_stopped** (EST_TIMER *timer)
- unsigned char **is_running** (EST_TIMER *timer)

Variables

- const char * **EST_ERR_STRINGS** [ ]

5.1.1 Function Documentation

5.1.1.1 EST_ERROR est_add_attributes_helper (X509_REQ * *req,* int *nid,* void * *string,* int *chtype*)

est_add_attributes_helper() (p. 19) Add a NID and its character string to an X509_REQ as an attribute.

Parameters

req	an X509_REQ structure used for the CSR request
nid	NID to be added as an attribute
string	pointer to the NID string if needed
chtype	type of string used with this NID

Returns

EST_ERROR

This function is used to add a CSR attribute to a CSR request by the EST client.

5.1.1.2 EST_ERROR est_decode_attributes_helper (char * *csrattrs,* int *csrattrs_len,* unsigned char ** *der,* int * *len*)

est_decode_attributes_helper() (p. 19) Decode a base64 encoded string into DER format(ASN.1 hex).

Parameters

csrattrs	pointer to a base64 encoded string
csrattrs_len	base64 string length
der_ptr	pointer to a pointer to store the DER encoded string
der_len	pointer to store the DER string length

Returns

EST_ERROR

This function is used decode a base64 encoded CSR attributes string into DER format. It also performs range checking on the input parameters.

5.1.1.3 EST_ERROR est_destroy (EST_CTX * *ctx*)

est_destroy() (p. 20) frees an EST context

Parameters

ctx	Pointer to an EST context

Returns

EST_ERROR

This function is used to release all the memory allocated under the EST_CTX*. This should be called last after performing EST operations using the context.

5.1.1.4 EST_ERROR est_disable_performance_timers (EST_CTX * *ctx*)

est_disable_performance_timers() (p. 20) is used by an application to disable the use and output of the libest performance timers for the given est context.

Parameters

ctx	Pointer to the EST context

The libest performance timers is disabled by default.

Returns

EST_ERROR.

5.1.1.5 void est_enable_backtrace (int *enable*)

est_enable_backtrace() (p. 21) allows the application to toggle whether the stack trace is displayed for WARNING and ERROR log messages coming from libEST.

Parameters

enable	Set to zero to disable stack traces, non-zero to enable stack traces through the logging facility.

This function allows an application to enable stack traces, which may be useful for troubleshooting the libEST library. Stack traces are disabled by default. Call this function with a non-zero argument to enable stack traces for both WARNING and ERROR log messages. This setting is global to the library and will impact all contexts.

Returns

void.

5.1.1.6 EST_ERROR est_enable_crl (EST_CTX * *ctx*)

est_enable_crl() (p. 21) is used by an application to enable checking of a certificate revocation list when validating the client TLS peer certificate during the TLS handshake. When enabled, the ca_chain parameter provided to either **est_server_init()** (p. 57) or **est_client_init()** (p. 35) should contain both the trusted certificates along with the CRL entries. The CRL entries should be appened at the end.

Parameters

ctx	Pointer to the EST context

CRL checking is disabled by default. This function must be called after invoking **est_server_init()** (p. 57) or **est_client_init()** (p. 35) and prior to performing any EST operations. Therefore, there is no 'disable' version of this method.

Returns

EST_ERROR.

5.1.1.7 EST_ERROR est_enable_performance_timers (EST_CTX * *ctx*)

est_enable_performance_timers() (p. 21) is used by an application to enable the use and output of the libest performance timers for the given est context. When enabled these timers will output logs that can be parsed to give you performance metrics of different areas of libest as they are executed.

Parameters

ctx	Pointer to the EST context

The libest performance timers is disabled by default.

Returns

EST_ERROR.

5.1.1.8 int est_get_api_level (void)

est_get_api_level() (p. 22) allows the application to retrieve the libEST API level. This is a numeric value that indicates the API level of the library. When new versions of libEST are released and the API changes, this value will be incremented. Applications can use this to determine which capabilities in the libEST library should or should not be attempted.

Returns

int

5.1.1.9 EST_ERROR est_get_attributes_helper (unsigned char ** *der_ptr,* int * *der_len,* int * *new_nid*)

est_get_attributes_helper() (p. 22) get attributes NID from a DER encoded string.

Parameters

der_ptr	pointer to a pointer of DER encoded string
der_len	pointer to the DER encoded string length
new_nid	pointer to storage for NID, if found

Returns

EST_ERROR

This function is used to find the next NID in a DER encoded string. If no NID is found before reaching the end of the string, then new_nid returned as zero and EST_ERR_BAD_ASN1_HEX.

5.1.1.10 void* est_get_ex_data (EST_CTX * *ctx*)

est_get_ex_data() (p. 22) retrieves the application specific data on the EST context.

Parameters

ctx	Pointer to an EST context

Returns

void∗

This function is used to attain a reference to the application specific data on the EST_CTX structure. This data should have been set by invoking **est_set_ex_data()** (p. 24) earlier. Otherwise it will return NULL.

5.1.1.11 const char∗ est_get_version (void)

est_get_version() (p. 23) allows the application to retrieve the libEST version string. Returns a char∗ array containing the full version string value for the library.

Returns

const char∗

5.1.1.12 EST_ERROR est_init_logger (EST_LOG_LEVEL *lvl,* void(∗)(char ∗, va_list) *loggerfunc*)

est_init_logger() (p. 23) allows the application to override the default log handler for EST logging messages.

Parameters

lvl	Sets the desired logging level to EST_LOG_LEVEL
loggerfunc	Sets the callback function to handle logging

This function allows an application that uses EST to provide a function for logging EST messages. EST provides a default handler that sends messages to stderr. Applications may desire to send messages to syslog or some other logging facility. An application would provide a function pointer using this method to intercept and handle EST log messages. This setting is global to the library and will impact all contexts.

Returns

EST_ERROR.

5.1.1.13 EVP_PKEY∗ est_load_key (unsigned char ∗ *key,* int *key_len,* int *format*)

est_load_key() (p. 23) is a helper function that reads a char∗ and converts it to an OpenSSL EVP_PKEY∗. The char∗ data can be either PEM or DER encoded.

Parameters

key	This is the char∗ that contains the PEM or DER encoded key pair.
key_len	This is the length of the key char∗. DER encoded data may contain zeros, which requires the length to be provided by the application layer.
key_format	This parameter specifies the encoding method of the key char∗ that was provided. Set this to either EST_FORMAT_PEM or EST_FORMAT_DER.

This function converts a PEM or DER encoded char* to the OpenSSL EVP_PKEY* structure. This function will return NULL if the PEM/DER data is corrupted or unable to be parsed by the OpenSSL library. This function will allocate memory for the EVP_PKEY data. You must free the memory in your application when it's no longer needed by calling EVP_PKEY_free().

Returns

EVP_PKEY*

5.1.1.14 X509_REQ* est_read_x509_request (unsigned char * *csr,* int *csr_len,* EST_CERT_FORMAT *csr_format*)

est_read_x509_request() (p. 24) is a helper function that reads a char* and converts it to an OpenSSL X509_REQ*. The char* data can be either PEM or DER encoded.

Parameters

csr	This is the char* that contains the PEM or DER encoded X509 CSR.
csr_len	This is the length of the csr char*. DER encoded data may contain zeros, which requires the length to be provided by the application layer.
csr_format	This parameter specifies the encoding method of the csr char* that was provided. Set this to either EST_CERT_FORMAT_PEM or EST_CERT_FORMAT_DER.

This function converts a PEM or DER encoded char* to the OpenSSL X509_REQ structure. This function will return NULL if the PEM/DER data is corrupted or unable to be parsed by the OpenSSL library. This function will allocate memory for the X509_REQ data. You must free the memory in your application when it's no longer needed by calling X509_REQ_free().

Returns

X509_REQ*

5.1.1.15 EST_ERROR est_set_ex_data (EST_CTX * *ctx,* void * *ex_data*)

est_set_ex_data() (p. 24) sets the application specific data on the EST context.

Parameters

ctx	Pointer to an EST context
ex_data	Pointer to application specific data that will be passed through to the EST callbacks.

Returns

EST_ERROR

This function is used to link application specific data to the EST_CTX structure. This can be used by an application to bind application specific data to an EST operation. libEST does not use the application specific data. The *ex_data

pointer is passed back to the application when libEST invokes the enroll, re-enroll, CSR attributes, and HTTP auth callbacks.

libEST will not free the memory referenced by the *ex_data parameter when **est_destroy()** (p. 20) is invoked. The application is responsible for releasing its application specific data.

5.1.1.16 int est_X509_REQ_sign (X509_REQ * *csr,* EVP_PKEY * *pkey,* const EVP_MD * *md*)

est_X509_REQ_sign() (p. 25) Sign an X509 certificate request using the digest and the key passed. Returns OpenSSL error code from X509_REQ_sign_ctx();

Parameters

csr	an X509_REQ structure to be signed
pkey	key to sign the request with
md	the signing digest to be used

Returns

int

5.1.2 Variable Documentation

5.1.2.1 const char* EST_ERR_STRINGS[ ]

Initial value:

```
= {
    "EST_ERR_NONE",
}
```

5.2 src/est/est_client.c File Reference

```
#include <string.h>
```

```
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/types.h>
#include <openssl/ssl.h>
#include <openssl/cms.h>
#include <openssl/rand.h>
#include "est.h"
#include "est_locl.h"
#include "est_ossl_util.h"
#include "jsmn.h"
#include "safe_mem_lib.h"
#include "safe_str_lib.h"
#include <openssl/x509v3.h>
#include <openssl/asn1.h>
```

Macros

- #define **SLEEP**(x) sleep(x)
- #define **SSL_EXDATA_INDEX_INVALID** -1
- #define **HOST_NOMATCH** 0
- #define **HOST_MATCH** 1

Functions

- int **est_client_set_cert_and_key** (SSL_CTX ∗ctx, X509 ∗cert, EVP_PKEY ∗key)
- int **est_client_send_enroll_request_internal** (EST_CTX ∗ctx, SSL ∗ssl, BUF_MEM ∗bptr, unsigned char ∗new_key, int ∗key_len, unsigned char ∗pkcs7, int ∗pkcs7_len, int est_op)
- int **est_client_send_enroll_request** (EST_CTX ∗ctx, SSL ∗ssl, BUF_MEM ∗bptr, unsigned char ∗pkcs7, int ∗pkcs7_len, int reenroll)
- EST_ERROR **est_client_send_keygen_request** (EST_CTX ∗ctx, SSL ∗ssl, BUF_MEM ∗bptr, unsigned char ∗new_key, int ∗key_len, unsigned char ∗pkcs7, int ∗pkcs7_len)
- EST_ERROR **est_client_connect** (EST_CTX ∗ctx, SSL ∗∗ssl)
- void **est_client_disconnect** (EST_CTX ∗ctx, SSL ∗∗ssl)
- EST_ERROR **est_client_set_uid_pw** (EST_CTX ∗ctx, const char ∗uid, const char ∗pwd)
- EST_ERROR **est_client_enroll_internal** (EST_CTX ∗ctx, char ∗cn, int ∗pkcs7_len, int ∗pkcs8_len, EVP_PKEY ∗new_public_key, EST_OPERATION est_op)
- EST_ERROR **est_client_server_keygen_enroll** (EST_CTX ∗ctx, char ∗cn, int ∗pkcs7_len, int ∗pkcs8_len, E↩ VP_PKEY ∗new_public_key)

  est_client_server_keygen_enroll() *(*p. *37) performs the simple enroll request with the EST server*
- EST_ERROR **est_client_enroll** (EST_CTX ∗ctx, char ∗cn, int ∗pkcs7_len, EVP_PKEY ∗new_public_key)

  est_client_enroll() *(*p. *31) performs the simple enroll request with the EST server*
- EST_ERROR **est_client_provision_cert** (EST_CTX ∗ctx, char ∗cn, int ∗pkcs7_len, int ∗ca_cert_len, EVP_P↩ KEY ∗new_public_key)

est_client_provision_cert() *(p. 36) performs the full sequence of EST operations to enroll a new certificate using a trusted message flow.*

- EST_ERROR **est_client_reenroll** (EST_CTX ∗ctx, X509 ∗cert, int ∗pkcs7_len, EVP_PKEY ∗priv_key)

  est_client_reenroll() *(p. 36) performs a re-enroll request with the EST server using an existing X509 certificate.*
- EST_ERROR **est_client_enroll_csr** (EST_CTX ∗ctx, X509_REQ ∗csr, int ∗pkcs7_len, EVP_PKEY ∗priv_key)

  est_client_enroll_csr() *(p. 32) performs the simple enroll request with the EST server using a PKCS10 CSR provided by the application layer.*
- EST_ERROR **est_client_server_keygen_enroll_csr** (EST_CTX ∗ctx, X509_REQ ∗csr, int ∗pkcs7_len, int ∗pkcs8_len, EVP_PKEY ∗priv_key)

  est_client_server_keygen() performs the simple enroll with server side key generation request with the EST server
- EST_ERROR **est_client_copy_enrolled_cert** (EST_CTX ∗ctx, unsigned char ∗pkcs7)

  est_client_copy_enrolled_cert() *(p. 28) passes back the client certificate that was previously obtained from the EST server by the call to* ***est_client_enroll()*** *(p. 31).*
- EST_ERROR **est_client_copy_server_generated_key** (EST_CTX ∗ctx, unsigned char ∗pkcs8)

  est_client_copy_server_generated_key() *(p. 30) passes back the server generated private key that was previously obtained from the EST server by the call to* ***est_client_enroll()*** *(p. 31).*
- EST_ERROR **est_client_get_cacerts** (EST_CTX ∗ctx, int ∗ca_certs_len)

  est_client_get_cacerts() *(p. 34) performs a CAcerts GET request to the EST server*
- EST_ERROR **est_client_copy_cacerts** (EST_CTX ∗ctx, unsigned char ∗ca_certs)

  est_client_copy_cacerts() *(p. 28) copies the previously retrieved CA certificates to the application's buffer.*
- EST_ERROR **est_client_get_csrattrs** (EST_CTX ∗ctx, unsigned char ∗∗csr_data, int ∗csr_len)

  est_client_get_csrattrs() *(p. 34) performs the CSR attributes request to the EST server.*
- EST_ERROR **est_client_enable_srp** (EST_CTX ∗ctx, int strength, char ∗uid, char ∗pwd)

  est_client_enable_srp() *(p. 30) is used by an application to enable TLS-SRP as the transport, which is used in place of traditional TLS. TLS-SRP allows for secure transport when an X.509 certificate is not available or when a trust anchor is not available.*
- EST_ERROR **est_client_set_auth** (EST_CTX ∗ctx, const char ∗uid, const char ∗pwd, X509 ∗client_cert, EV↩ P_PKEY ∗private_key)

  est_client_set_auth() *(p. 38) is used by an application to set up the authentication parameters to be used.*
- EST_ERROR **est_client_set_auth_cred_cb** (EST_CTX ∗ctx, auth_credentials_cb callback)

  est_client_set_auth_cred_cb() *(p. 39) is used by an application to register its callback function.*
- EST_ERROR **est_client_enable_basic_auth_hint** (EST_CTX ∗ctx)

  est_client_enable_basic_auth_hint() *(p. 30) is used by an application to reduce overhead at the TCP and TLS layers when the client knows that the EST server is using HTTP Basic Authentication.*
- EST_CTX ∗ **est_client_init** (unsigned char ∗ca_chain, int ca_chain_len, EST_CERT_FORMAT cert_format, int(∗cert_verify_cb)(X509 ∗, int))

  est_client_init() *(p. 35) is used by an application to create a context in the EST library. This context is used when invoking other functions in the client API.*
- EST_ERROR **est_client_set_server** (EST_CTX ∗ctx, const char ∗server, int port, char ∗path_segment)

  est_client_set_server() *(p. 41) is called by the application layer to specify the address/port of the EST server. It must be called after* ***est_client_init()*** *(p. 35) and prior to issuing any EST commands.*
- EST_ERROR **est_client_set_proxy** (EST_CTX ∗ctx, EST_CLIENT_PROXY_PROTO proxy_proto, const char ∗proxy_server, unsigned short int proxy_port, unsigned int proxy_auth, const char ∗username, const char ∗password)

  est_client_set_proxy() *(p. 40) is called by the application layer to specify the proxy to the EST server. It must be called after* ***est_client_init()*** *(p. 35) and prior to issuing any EST commands.*
- EST_ERROR **est_client_set_sign_digest** (EST_CTX ∗ctx, int nid)

  est_client_set_sign_digest() *(p. 41) is called by the application layer to specify the hash algorithm used to sign the P↩ KCS10 CSR during the enroll operation. It must be called after* ***est_client_init()*** *(p. 35) and prior to issuing any EST commands.*

- EST_ERROR **est_client_copy_retry_after** (EST_CTX *ctx, int *retry_delay, time_t *retry_time)

  est_client_copy_retry_after() *(p. 29) copies the retry after value stored in this client context.*
- EST_ERROR **est_client_force_pop** (EST_CTX *ctx)

  est_client_force_pop() *(p. 33) is used by an application to enable the proof-of-possession generation at the EST client. This proves that the EST client that sent the CSR to the server/proxy is in possession of the private key that was used to sign the CSR. This binds the TLS session ID to the CSR.*
- EST_ERROR **est_client_unforce_pop** (EST_CTX *ctx)

  est_client_unforce_pop() *(p. 42) is used by an application to disable the proof-of-possession generation at the EST client. Please see the documentation for* ***est_client_force_pop()*** *(p. 33) for more information on the proof-of-possession check.*
- EST_ERROR **est_client_set_read_timeout** (EST_CTX *ctx, int timeout)

  est_client_set_read_timeout() *(p. 40) is used by an application to set timeout value of read operations. After the EST client sends a request to the EST server it will attempt to read the response from the server. This timeout value limits the amount of time the client will wait for the response.*
- int **est_client_get_last_http_status** (EST_CTX *ctx)

  est_client_get_last_http_status() *(p. 35) is used by an application to get the HTTP status code returned by the EST server for the most recent operation.*

Variables

- int **e_ctx_ssl_exdata_index** = SSL_EXDATA_INDEX_INVALID

5.2.1 Function Documentation

5.2.1.1 EST_ERROR est_client_copy_cacerts (EST_CTX * *ctx*, unsigned char * *ca_certs*)

est_client_copy_cacerts() (p. 28) copies the previously retrieved CA certificates to the application's buffer.

Parameters

ctx	Pointer to the current EST context.
ca_certs	Pointer to the buffer into which the retrieved CA certificates are to be copied.

Returns

EST_ERROR

est_client_copy_cacerts() (p. 28) copies the most recently retrieved CA certificates from the EST server. Once these CA certificates are copied to the application's buffer pointed to by ca_certs they are removed from the EST client context.

Once the CA certificates are retrieved by the application, the EST client library must be reset. When this reset is performed, the CA certificates retrieved in this est_client_copy_cacerts call should be passed into the EST client initialization function as the explicit TA database.

5.2.1.2 EST_ERROR est_client_copy_enrolled_cert (EST_CTX * *ctx*, unsigned char * *pkcs7*)

est_client_copy_enrolled_cert() (p. 28) passes back the client certificate that was previously obtained from the EST server by the call to **est_client_enroll()** (p. 31).

Parameters

ctx	Pointer to an EST context
pkcs7	Pointer to a pointer that will point to the buffer that contains the newly enrolled client certificate.

Returns

EST_ERROR

est_client_copy_enrolled_cert() (p. 28) copies the previously obtained client certificate from the EST context to the application's buffer. Once this client certificate is copied out of the context it is removed from the context.

5.2.1.3 EST_ERROR est_client_copy_retry_after (EST_CTX * *ctx,* int * *retry_delay,* time_t * *retry_time*)

est_client_copy_retry_after() (p. 29) copies the retry after value stored in this client context.

Parameters

ctx	Pointer to the current EST context.
retry_delay	Pointer to the integer where the retry-after delay secs value is copied. If the server sent a retry-after in delay seconds format then it will be passed here. If it did not, then this value will be zero.
retry_time	Pointer to the time_t where the retry-after time date value is copied. If the server sent a retry-after in time and date string format then this string is converted into a time_t value and passed up in this parameter. This value will only be set if the server sent a time and date string response, otherwise, this value is set to zero.

Returns

EST_ERROR

When a response is received from the EST server the headers are checked to see if the server has included a Retry-↩ After header, indicating that this request currently cannot be processed. If a Retry-After HTTP header is included in the received response from the server the delay value is saved in the context and an EST error code is given to the application on this request indicating that the client must retry the request at a later time.

The value specified by the server can be in one of two basic formats, a string version of a integer value that represents the number of seconds the client must wait before retrying the request, and a string containing a date and time when the client can retry the request. The date and time string can be in any format specified in RFC 2616. If the second delay value is sent it is converted into an integer and saved in the EST context and if the date time string value is sent it is converted into a time_t value and saved into the EST context. The application must then call **est_client_copy_retry↩ _after()** (p. 29) to obtain the amount of time to wait before retrying the request. **est_client_copy_retry_after()** (p. 29) copies the current retry-after value from the client context and returns it to the application. Only one of the two return values will be set with a non-zero value.

NOTE: The processing of a Retry-After value in time/date format is currently not supported. The EST Client will always return only a retry delay value in seconds.

5.2.1.4 EST_ERROR est_client_copy_server_generated_key (EST_CTX * *ctx,* unsigned char * *pkcs8*)

est_client_copy_server_generated_key() (p. 30) passes back the server generated private key that was previously obtained from the EST server by the call to **est_client_enroll()** (p. 31).

Parameters

ctx	Pointer to an EST context
pkcs8	Pointer to a pointer that will point to the buffer that contains the server generated private key.

Returns

EST_ERROR

est_client_copy_server_generated_key() (p. 30) copies the previously obtained server-generated key from the EST context to the application's buffer. Once this key is copied out of the context it is removed from the context.

5.2.1.5 EST_ERROR est_client_enable_basic_auth_hint (EST_CTX * *ctx*)

est_client_enable_basic_auth_hint() (p. 30) is used by an application to reduce overhead at the TCP and TLS layers when the client knows that the EST server is using HTTP Basic Authentication.

Parameters

ctx	Pointer to EST context for a client session

Normally libEST will send an anonymous HTTP request when doing the initial request from the EST server. This function allows an application to improve performance by sending the HTTP Basic Auth header in the initial request sent to the EST server. This eliminates the need for the server to send the HTTP authentication challenge response, which eliminates a round-trip between the EST client and server. This function should be called immediately after invoking **est_client_set_auth()** (p. 38).

Precautions should be taken by your application to ensure this hint is only enabled when it is known that the EST server is configured for HTTP Basic Authentication. If the EST server is configured for HTTP Digest Authentication, then enabling this hint will cause the EST transaction to fail.

Returns

EST_ERROR EST_ERR_NONE - Success.

5.2.1.6 EST_ERROR est_client_enable_srp (EST_CTX * *ctx,* int *strength,* char * *uid,* char * *pwd*)

est_client_enable_srp() (p. 30) is used by an application to enable TLS-SRP as the transport, which is used in place of traditional TLS. TLS-SRP allows for secure transport when an X.509 certificate is not available or when a trust anchor is not available.

Parameters

ctx	EST context obtained from the **est_client_init()** (p. 35) call.
strength	Specifies the SRP strength to use.
uid	char buffer containing the user id to be used as the SRP user name.
pwd	char buffer containing the password to be used as the SRP password.

This function allows an application to enable TLS-SRP cipher suites, which is another form for TLS. This could be used when the EST client does not have an X.509 certificate to identify itself to the EST server. It can also be used by the EST client when a trust anchor is not available to authenticate the EST server identity. The EST server must support TLS-SRP when using this API.

This function must be invoked after **est_client_init()** (p. 35) and prior to issuing any EST commands..

All string parameters are NULL terminated strings.

Returns

EST_ERROR.

5.2.1.7 EST_ERROR est_client_enroll (EST_CTX * *ctx*, char * *cn*, int * *pkcs7_len*, EVP_PKEY * *new_public_key*)

est_client_enroll() (p. 31) performs the simple enroll request with the EST server

Parameters

ctx	Pointer to an EST context
cn	Pointer to the Common Name value to be used in the enrollment request.
pkcs7_len	Pointer to an integer to hold the length of the PKCS7 buffer.
new_public_key	Pointer an EVP_PKEY structure that holds the client's key pair to be used in the simple enroll request . The public key is included in the Certificate Signing Request (CSR) sent to the CA Server, and the private key is used to sign the request.

Returns

EST_ERROR

est_client_enroll() (p. 31) connects to the EST server, builds a simple enroll request using the Common Name passed in cn, establishes a SSL/TLS connection to the EST server that was configured with the previous call to **est_client_set_server()** (p. 41), and sends the simple enroll request. The response is stored in the EST context and the length is passed back to the application through the pkcs7_len parameter of this function. The application can then allocate a correctly sized buffer and call **est_client_copy_enrolled_cert()** (p. 28) to retrieve the new client certificate from the context.

5.2.1.8 EST_ERROR est_client_enroll_csr (EST_CTX * *ctx,* X509_REQ * *csr,* int * *pkcs7_len,* EVP_PKEY * *priv_key*)

est_client_enroll_csr() (p. 32) performs the simple enroll request with the EST server using a PKCS10 CSR provided by the application layer.

Parameters

ctx	Pointer to an EST context
csr	Pointer to the PKCS10 CSR data, which is defined as an OpenSSL X509_REQ.
pkcs7_len	Pointer to an integer to hold the length of the PKCS7 buffer.
priv_key	Pointer to the private key that will be used to sign the CSR, or NULL

Returns

EST_ERROR

est_client_enroll_csr() (p. 32) connects to the EST server, establishes a SSL/TLS connection to the EST server that was configured with the previous call to **est_client_set_server()** (p. 41), and sends the simple enroll request. The application layer must provide the PKCS10 CSR that will be enrolled. If the priv_key argument given is not NULL, then the CSR should not need to be signed by the private key. However, the CSR must contain everything else that is required, including the public key. If the private key is provided with an already signed CSR, then the EST library will re-sign the CSR.

The enroll response is stored in the EST context and the length is passed back to the application through the pkcs7_len parameter of this function. The application can then allocate a correctly sized buffer and call **est_client_copy_↩ enrolled_cert()** (p. 28) to retrieve the new client certificate from the context.

Unless the CSR is not already signed, which is indicated by a NULL priv_key, the application must provide a pointer to the private key used to sign the CSR. This is required by the EST library in the event that the EST server has requested the proof-of-possession value be included in the CSR. The EST library will automatically include the proof-of-posession value and sign the CSR again.

Be aware that the X509_REQ data passed to this function must be valid. Passing corrupted CSR data may result in a system crash. libEST utilizes the OpenSSL ASN decoding logic to read the X509_REQ data. OpenSSL does not perform safety checks on the X509_REQ data when parsing. If your application is reading externally generated PEM or DER encoded CSR data, then please use the **est_read_x509_request()** (p. 24) helper function to convert the PEM/DER CSR into a valid X509_REQ pointer.

5.2.1.9 EST_ERROR est_client_force_pop (EST_CTX * *ctx*)

est_client_force_pop() (p. 33) is used by an application to enable the proof-of-possession generation at the EST client. This proves that the EST client that sent the CSR to the server/proxy is in possession of the private key that was used to sign the CSR. This binds the TLS session ID to the CSR.

Note, if the CSR attributes configured on the server require PoP checking, then there is no need to call this function to enable PoP. The PoP will be enabled automatically under this scenario when the CSR attributes are requested from the server/proxy.

Parameters

ctx	Pointer to the EST context

This function may be called at any time.

Returns

EST_ERROR.

5.2.1.10 EST_ERROR est_client_get_cacerts (EST_CTX * *ctx,* int * *ca_certs_len*)

est_client_get_cacerts() (p. 34) performs a CAcerts GET request to the EST server

Parameters

ctx	Pointer to an EST context
ca_certs_len	Pointer to an integer to hold the length of the CA certs buffer

Returns

EST_ERROR

est_client_get_cacerts() (p. 34) connects to the EST server, builds a CA certs request, and sends the GET CA certs request. The response is placed in a buffer allocated and maintained by the EST client library and a pointer to this buffer is returned to the calling application. The returned CA certs are in base64 encoded DER format and is stored in a NULL terminated string buffer.

Once the CA certificates are retrieved from the EST server, the ET Client library must be reset. The retrieved CA certificates should now be passed into the EST client initialization function as the explicit TA database.

5.2.1.11 EST_ERROR est_client_get_csrattrs (EST_CTX * *ctx,* unsigned char ** *csr_data,* int * *csr_len*)

est_client_get_csrattrs() (p. 34) performs the CSR attributes request to the EST server.

Parameters

ctx	Pointer to EST context for a client session
csr_data	Pointer to a buffer that is to hold the returned CSR attributes
csr_len	Pointer to an integer that is to hold the length of the CSR attributes buffer

Returns

EST_ERROR

est_client_get_csrattrs() (p. 34) connects to the EST server, sends the CSR attributes request to the server, saves aways the returned CSR attribute data, and then disconnects from the EST server.

API Change Note: In order to allow more descriptive errors to be supplied to the client, the returned EST_ERROR codes that are returned from this function have changed. In the past when no CSR attributes were found on the server and a HTTP 204 (No Content) or a HTTP 404 (Not Found) code were received, this function would return an EST_ERR_NONE

indicating no error as the client should be able to continue to operating without the csr attributes. Releases of libEST after 3.1.0 will now return the error code EST_ERR_HTTP_NO_CONTENT, and EST_ERR_HTTP_NOT_FOUND when the client recieves a 204 and a 404 response code respectively. Client code that utilizes this function may need to be updated to handle these error code as a success cases. For examples take a look at the updated example code in estclient.c.

5.2.1.12 int est_client_get_last_http_status (EST_CTX * *ctx*)

est_client_get_last_http_status() (p. 35) is used by an application to get the HTTP status code returned by the EST server for the most recent operation.

Parameters

ctx	Pointer to the EST context

This can be called after an EST operation, such as an enroll operation. This function will return the most recent HTTP status code received from the EST server. Normally, a status of 200 would be returned by the EST server to indicate a successful operation. However, if the operation failed for some reason, the HTTP status code may be useful to understand the reason for failure. For instance, the EST server would return a HTTP status of 401 if the EST client was not authorized. Please see RFC 2616 for a description of the various HTTP status codes.

Returns

int value representing the HTTP status code, or NULL if the a NULL EST context was provided.

5.2.1.13 EST_CTX* est_client_init (unsigned char * *ca_chain,* int *ca_chain_len,* EST_CERT_FORMAT *cert_format,* int(*)(X509 *, int) *cert_verify_cb*)

est_client_init() (p. 35) is used by an application to create a context in the EST library. This context is used when invoking other functions in the client API.

Parameters

ca_chain	Required char buffer containing CA certificates as raw byte data, to be used for authenticating the EST server
ca_chain_len	length of ca_chain char buffer.
cert_format	defines the format of the certificates that will be passed down during this instantiation of the EST client library. Currently, the only value accepted is EST_CERT_FORMAT_PEM
cert_verify_cb	A pointer to a function in the EST client application that is called when a received server identity certificate has failed verification from the SSL code. This function takes as input two parameters, a pointer to the X509 structure containing the server's certificate, and a integer value set to the OpenSSL defined error for this certificate. This callback function returns a 0 if the server's identity certificate has been rejected, and any other value if it has been approved.

This function allows an application to initialize an EST client context. The application must provide the local CA certificates (ca_chain/ca_chain_len) to use for client operation. The certificates provided must be in the format specified

by the cert_format parameter. Currently, only PEM encoded certificates are supported. The length parameters for the certificates (ca_chain_len) are to be used when DER formatted certificates are passed. The CA certificates may contain CRL entries that will be used when authenticating the certificates received from the server.

Returns

EST_CTX. If error, NULL.

5.2.1.14 EST_ERROR est_client_provision_cert (EST_CTX * *ctx,* char * *cn,* int * *pkcs7_len,* int * *ca_cert_len,* EVP_PKEY * *new_public_key*)

est_client_provision_cert() (p. 36) performs the full sequence of EST operations to enroll a new certificate using a trusted message flow.

Parameters

ctx	Pointer to an EST context
cn	Pointer to the Common Name value to be used in the enrollment request.
pkcs7_len	Pointer to an integer to hold the length of the PKCS7 certificate returned from the RA or CA.
ca_cert_len	Pointer to an integer to hold the length of the buffer that will hold the new trusted CA certificates.
new_public_key	Pointer an EVP_PKEY structure that holds the client's key pair to be used in the simple enroll request . The public key is included in the Certificate Signing Request (CSR) sent to the CA Server, and the private key is used to sign the request.

Returns

EST_ERROR

est_client_provision_cert() (p. 36) connects to the EST server, retrieves the latest trusted CA certificates from the server, retrieves the CSR attributes from the server, and sends the simple enroll request to the server to provision a new certificate from the RA or CA. This is a convenience function that is equivalent to invoking the following three functions in order:

est_client_get_cacerts() (p. 34) **est_client_get_csrattrs()** (p. 34) **est_client_enroll()** (p. 31)

This function takes a Common Name (CN) as the only entity identifier that will be used in the CSR. If additional X509 attributes or extensions are required because the EST server is enforcing the presence of all the CSR attributes, then this function should not be used to provision a certificate. The **est_client_enroll_csr()** (p. 32) function should be used when additional X509 attributes are to be included in the enroll request.

The provisioning response is stored in the EST context and the length is passed back to the application through the pkcs7_len parameter of this function. The application can then allocate a correctly sized buffer and call **est_client_↩ copy_enrolled_cert()** (p. 28) to retrieve the new client certificate from the context.

The provisioning response also includes the latest copy of the trusted CA certificates from the EST server. These should be persisted locally by the application for future use. The ca_cert_len argument will contain the length of the certificates, which can then be retrieved by invoking **est_client_copy_cacerts()** (p. 28).

5.2.1.15 EST_ERROR est_client_reenroll (EST_CTX * *ctx,* X509 * *cert,* int * *pkcs7_len,* EVP_PKEY * *priv_key*)

est_client_reenroll() (p. 36) performs a re-enroll request with the EST server using an existing X509 certificate.

Parameters

ctx	Pointer to an EST context
cert	Pointer to the X509 certificate, which is defined as an OpenSSL X509.
pkcs7_len	Pointer to an integer to hold the length of the PKCS7 buffer.
priv_key	Pointer to the private key that will be used to sign the CSR.

Returns

EST_ERROR

est_client_reenroll() (p. 36) connects to the EST server, establishes a SSL/TLS connection to the EST server that was configured with the previous call to **est_client_set_server()** (p. 41), and sends the re-enroll request. The application layer must provide the X509 certificate that will be enrolled. This certificate should have previously been enrolled with the CA. The application also needs to provide the private key associated with the public key in the X509 certificate. This private key is required to sign the CSR that is generated from the X509 certificate.

The enroll response is stored in the EST context and the length is passed back to the application through the pkcs7_len parameter of this function. The application can then allocate a correctly sized buffer and call **est_client_copy_↩ enrolled_cert()** (p. 28) to retrieve the new client certificate from the context.

The application must provide a pointer to the private key used to sign the CSR. This is required by the EST library in the event that the EST server has requested the proof-of-possession value be included in the CSR. The EST library will automatically include the proof-of-posession value and sign the CSR again.

Be aware that only the public key and subject name from the X509 certificate are included in the re-enroll request sent to the EST server. The CA is responsible for re-applying any X509 extensions that are to be issued with the renewed certificate.

5.2.1.16 EST_ERROR est_client_server_keygen_enroll (EST_CTX * *ctx*, char * *cn*, int * *pkcs7_len*, int * *pkcs8_len*, EVP_PKEY * *new_public_key*)

est_client_server_keygen_enroll() (p. 37) performs the simple enroll request with the EST server

Parameters

ctx	Pointer to an EST context
cn	Pointer to the Common Name value to be used in the enrollment request.
pkcs7_len	Pointer to an integer to hold the length of the PKCS7 buffer.
pkcs8_len	Pointer to an integer to hold the length of the PKCS8 buffer.
new_public_key	Pointer an EVP_PKEY structure that holds the client's key pair to be used in the simple enroll request . The public key is included in the Certificate Signing Request (CSR) sent to the CA Server, and the private key is used to sign the request.

Returns

EST_ERROR

est_client_server_keygen_enroll() (p. 37) connects to the EST server, builds a simple enroll request using the Common Name passed in cn. (It is built with the provided key, but the keygen logic on the server will properly populate a CSR with the newly server-generated key.) Then establishes an SSL/TLS connection to the EST server that was configured with the previous call to **est_client_set_server()** (p. 41), and sends the request. The response is stored in the EST context and the cert and key lengths are passed back to the application through the pkcs7_len and pkcs8_len parameters of this function. The application can then allocate a correctly sized buffer and call **est_client_copy_enrolled_cert()** (p. 28) to retrieve the new client certificate from the context, or **est_client_copy_server_generated_key()** (p. 30) to retrieve the new key from the context.

5.2.1.17 EST_ERROR est_client_server_keygen_enroll_csr (EST_CTX * *ctx,* X509_REQ * *csr,* int * *pkcs7_len,* int * *pkcs8_len,* EVP_PKEY * *priv_key*)

est_client_server_keygen() performs the simple enroll with server side key generation request with the EST server

Parameters

ctx	Pointer to an EST context
csr	Pointer to the PKCS10 CSR data, which is defined as an OpenSSL X509_REQ.
pkcs7_len	Pointer to an integer to hold the length of the PKCS7 buffer.
pkcs8_len	Pointer to an integer to hold the length of the PKCS8 buffer.
priv_key	Pointer to the private key that will be used to sign the CSR, or NULL

Returns

EST_ERROR

est_client_server_keygen_enroll() (p. 37) connects to the EST server, builds an enroll request using the CSR passed in csr. (It is built with the provided key, but the keygen logic on the server will properly populate a CSR with the newly server-generated key.) Then establishes an SSL/TLS connection to the EST server that was configured with the previous call to **est_client_set_server()** (p. 41), and sends the request. The response is stored in the EST context and the cert and key lengths are passed back to the application through the pkcs7_len and pkcs8_len parameters of this function. The application can then allocate a correctly sized buffer and call **est_client_copy_enrolled_cert()** (p. 28) to retrieve the new client certificate from the context, or **est_client_copy_server_generated_key()** (p. 30) to retrieve the new key from the context.

5.2.1.18 EST_ERROR est_client_set_auth (EST_CTX * *ctx,* const char * *uid,* const char * *pwd,* X509 * *client_cert,* EVP_PKEY * *private_key*)

est_client_set_auth() (p. 38) is used by an application to set up the authentication parameters to be used.

Parameters

ctx	EST context obtained from the **est_client_init()** (p. 35) call.
uid	char buffer containing the user id to be used for basic and digest based authentication
pwd	char buffer containing the password to be used for basic and digest based authentication
client_cert	char buffer containing the client application certificate.
private_key	Private key that can be used with the client cert

This function allows an application to provide the information required for authenticating the EST client with the EST server. Until this call is made, the only accepted request is the GET CA Certs. If the user id is provided, a password must also be provided.

The application may pass the private key (pkey_raw/pkey_len) to be used for signing requests to the server, otherwise, only basic or digest based authentication will be performed on the TLS session for these requests. If the private key is passed, it must contain the private key that matches the public key contained in the client_cert parameter.

All string parameters are NULL terminated strings.

Returns

EST_ERROR. If error, NULL.

5.2.1.19 EST_ERROR est_client_set_auth_cred_cb (EST_CTX * *ctx,* auth_credentials_cb *callback*)

est_client_set_auth_cred_cb() (p. 39) is used by an application to register its callback function.

Parameters

ctx	EST context obtained from the **est_client_init()** (p. 35) call.
auth_credentials_cb	Function pointer to the application layer callback

The registered callback function is used by the EST client library to obtain authentication credentials. The application can provide authentication credentials during initialization if they are available, such as the userid and password used with HTTP basic authentication. During the processing of a request, the EST client library will call this application callback in the event that it does not have the authentication credentials that are being requested by the EST server.

The callback function definition must match the following function prototype,

int (*auth_credentials_cb)(EST_HTTP_AUTH_HDR *auth_credentials);

auth_credentials - A pointer to a EST_HTTP_AUTH_HDR structure. The structure is provided by the EST library and the callback function fills in the specific credentials being requested. These credential values must be passed in the format in which they will be sent to the server, that is, the EST client library will perform no reformatting of these credentials. Ownership of the memory holding these credential values is transferred from the application layer to the EST library when the application layer returns these values to the EST library. This allows the EST library to free up this memory as soon as it is done using these values.

The return value from the callback must be one of the following values:

EST_HTTP_AUTH_CRED_SUCCESS - If the callback was able to provide the requested credentials. EST_HTTP_A↩ UTH_CRED_NOT_AVAILABLE - If the callback could not provide the requested credentials.

The auth_credentials_cb parameter can be set to NULL to reset the callback function.

All string parameters are NULL terminated strings.

Returns

EST_ERROR. EST_ERR_NONE - Success. EST_ERR_NO_CTX

5.2.1.20 EST_ERROR est_client_set_proxy (EST_CTX * *ctx,* EST_CLIENT_PROXY_PROTO *proxy_proto,* const char * *proxy_server,* unsigned short int *proxy_port,* unsigned int *proxy_auth,* const char * *username,* const char * *password*)

est_client_set_proxy() (p. 40) is called by the application layer to specify the proxy to the EST server. It must be called after **est_client_init()** (p. 35) and prior to issuing any EST commands.

Parameters

ctx	Pointer to EST context for a client session
proxy_proto	Proxy protocol
proxy_server	Name of the proxy server to connect to. The ASCII string representing the name of the server is limited to 254 characters (EST_MAX_SERVERNAME_LEN)
port	TCP port on the proxy server to connect
proxy_auth	Proxy authentication method
username	Username to use to authenticate with the proxy
password	Password to use to authenticate with the proxy

Returns

EST_ERROR EST_ERR_NONE - Success. EST_ERR_NO_CTX - NULL value passed for EST context EST_E↩ RR_INVALID_SERVER_NAME - NULL value passed for EST server name, or server name string too long EST_↩ ERR_INVALID_PORT_NUM - port num to proxy server is invalid EST_ERR_CLIENT_NOT_INITIALIZED - Called before **est_client_init()** (p. 35) EST_ERR_INVALID_PARAMETERS - NULL value passed for either username or password OR username or password is too long EST_ERR_CLIENT_PROXY_MODE_NOT_SUPPORTED - client proxy mode is only supported when built with libcurl support EST_ERR_INVALID_CLIENT_PROXY_PROTOCOL - An invalid proxy protocol has been specified

est_client_set_proxy error checks its input parameters and then stores the proxy information into the EST context.

NOTE: HTTP proxy tunnelling is not supported by libEST in server mode. If configuring libEST in client mode to communicate with libEST in server mode, then EST_CLIENT_PROXY_HTTP_NOTUNNEL must be specified for the proxy protocol.

5.2.1.21 EST_ERROR est_client_set_read_timeout (EST_CTX * *ctx,* int *timeout*)

est_client_set_read_timeout() (p. 40) is used by an application to set timeout value of read operations. After the EST client sends a request to the EST server it will attempt to read the response from the server. This timeout value limits the amount of time the client will wait for the response.

Parameters

ctx	Pointer to the EST context
timeout	Integer value representing the read timeout in seconds. The minimum value is EST_SSL_READ_TIMEOUT_MIN and the maximum value is EST_SSL_READ_TIMEOUT_MAX.

Returns

EST_ERROR.

5.2.1.22 EST_ERROR est_client_set_server (EST_CTX * *ctx,* const char * *server,* int *port,* char * *path_segment*)

est_client_set_server() (p. 41) is called by the application layer to specify the address/port of the EST server. It must be called after **est_client_init()** (p. 35) and prior to issuing any EST commands.

Parameters

ctx	Pointer to EST context for a client session
server	Name of the EST server to connect to. The ASCII string representing the name of the server is limited to 254 characters
port	TCP port on the EST server to connect
path_segment	A string containing the optional path segment to be added to the URI. If not used, set to NULL.

Returns

EST_ERROR EST_ERR_NONE - Success. EST_ERR_NO_CTX - NULL value passed for EST context ES↩ T_ERR_INVALID_SERVER_NAME - NULL value passed for EST server name, or server name string too long EST_ERR_CLIENT_NOT_INITIALIZED - Called before **est_client_init()** (p. 35) EST_ERR_INVALID_PORT_N↩ UM - Invalid port number input, less than zero or greater than 65535

est_client_set_server error checks its input parameters and then stores both the hostname and port number into the EST context.

5.2.1.23 EST_ERROR est_client_set_sign_digest (EST_CTX * *ctx,* int *nid*)

est_client_set_sign_digest() (p. 41) is called by the application layer to specify the hash algorithm used to sign the PKCS10 CSR during the enroll operation. It must be called after **est_client_init()** (p. 35) and prior to issuing any EST commands.

Parameters

ctx	Pointer to EST context for a client session
nid	This is the NID value defined in the OpenSSL header file obj_mac.h for the desired digest to use for signing.

Returns

EST_ERROR EST_ERR_NONE - Success. EST_ERR_NO_CTX - NULL value passed for EST context EST_E↩ RR_INVALID_DIGEST - An unsupported NID was provided.

libEST supports SHA1, SHA224, SHA256, SHA384, and SHA512 digests. SHA256 is the default digest to use for signing. There's no need to invoke this function unless another digest is desired. The supported NID values are: NID_sha1 NID_sha224 NID_sha256 NID_sha384 NID_sha512

5.2.1.24 EST_ERROR est_client_unforce_pop (EST_CTX * *ctx*)

est_client_unforce_pop() (p. 42) is used by an application to disable the proof-of-possession generation at the EST client. Please see the documentation for **est_client_force_pop()** (p. 33) for more information on the proof-of-possession check.

Parameters

ctx	Pointer to the EST context

This function may be called at any time.

Returns

EST_ERROR.

5.3 src/est/est_ossl_util.c File Reference

```
#include <openssl/pem.h>
#include <openssl/err.h>
#include <stdio.h>
#include <openssl/x509v3.h>
#include "est.h"
#include "est_locl.h"
#include "est_ossl_util.h"
#include "safe_mem_lib.h"
#include "safe_str_lib.h"
```

Functions

- EST_ERROR **est_get_subj_fld_from_cert** (void *cert_csr, EST_CERT_OR_CSR cert_or_csr, char *name, int len)
- int **ossl_verify_cb** (int ok, X509_STORE_CTX *ctx)
- EST_ERROR **ossl_init_cert_store** (X509_STORE *store, unsigned char *raw1, int size1)
- void **ossl_dump_ssl_errors** ()
- int **est_convert_p7b64_to_pem** (unsigned char *certs_p7, int certs_len, unsigned char **pem)

  est_convert_p7b64_to_pem() *(p. 42) converts the base64 encoded PKCS7 response from the EST server into PEM format.*

5.3.1 Function Documentation

5.3.1.1 int est_convert_p7b64_to_pem (unsigned char * *certs_p7,* int *certs_len,* unsigned char ** *pem*)

est_convert_p7b64_to_pem() (p. 42) converts the base64 encoded PKCS7 response from the EST server into PEM format.

Parameters

certs_p7	Points to a buffer containing the base64 encoded pkcs7 data.
certs_len	Indicates the size of the ∗certs_p7 buffer.
pem	Double pointer that will receive the PEM encoded data.

Several of the EST message return data that contains base64 encoded PKCS7 certificates. This function is used to convert the data to PEM format. This function will allocate memory pointed to by the ∗∗pem argument. The caller is responsible for releasing this memory. The return value is the length of the PEM buffer, or -1 on error.

Returns

int.

5.4 src/est/est_proxy.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include "safe_mem_lib.h"
#include "safe_str_lib.h"
#include "est.h"
#include "est_server_http.h"
#include "est_locl.h"
#include "est_server.h"
```

Functions

- void **proxy_cleanup** (EST_CTX ∗p_ctx)
- EST_ERROR **est_proxy_retrieve_cacerts** (EST_CTX ∗ctx, unsigned char ∗∗cacerts_rtn, int ∗cacerts_rtn_len)
- EST_ERROR **est_proxy_store_path_segment** (EST_CTX ∗ctx, char ∗path_seg)
- EST_ERROR **est_proxy_handle_simple_enroll** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, const char ∗ct, char ∗body, int body_len, char ∗path_seg, int reenroll, unsigned char ∗∗returned_cert, int ∗returned_cert_len)
- EST_ERROR **est_proxy_handle_server_keygen** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, const char ∗ct, char ∗body, int body_len, char ∗path_seg, unsigned char ∗∗returned_cert, int ∗returned_cert_len, unsigned char ∗∗returned_key, int ∗returned_key_len)
- EST_ERROR **est_proxy_handle_cacerts** (EST_CTX ∗ctx, void ∗http_ctx, char ∗path_seg)
- EST_ERROR **est_proxy_handle_csr_attrs** (EST_CTX ∗ctx, void ∗http_ctx, char ∗path_seg, unsigned char ∗∗returned_attrs, int ∗returned_attrs_len)
- EST_ERROR **est_proxy_http_request** (EST_CTX ∗ctx, void ∗http_ctx, char ∗method, char ∗uri, char ∗body, int body_len, const char ∗ct)
- EST_ERROR **est_proxy_start** (EST_CTX ∗ctx)

  est_proxy_start() *(p. 47) is used by an application to start the EST proxy after* ***est_proxy_init()*** *(p. 45) and* ***est_proxy_set_server()*** *(p. 47) have been called and all the required callback functions have been provided by the application.*
- EST_ERROR **est_proxy_stop** (EST_CTX ∗ctx)

est_proxy_stop() *(p. 48) is used by an application to stop the EST proxy. This should be called prior to* ***est_destroy()*** *(p. 20).*

- EST_CTX ∗ **est_proxy_init** (unsigned char ∗ca_chain, int ca_chain_len, unsigned char ∗cacerts_resp_chain, int cacerts_resp_chain_len, EST_CERT_FORMAT cert_format, char ∗http_realm, X509 ∗tls_id_cert, EVP_PKEY ∗tls_id_key, char ∗uid, char ∗pwd)

  est_proxy_init() *(p. 45) is used by an application to create a context in the EST library. This context is used when invoking other functions in the API while in Proxy mode.*
- EST_ERROR **est_proxy_coap_init_start** (EST_CTX ∗ctx, int port)

  est_proxy_coap_init_start() *(p. 44) is called by the application layer to specify the address/port of the EST server. It must be called after* ***est_proxy_init()*** *(p. 45) and prior to starting the EST proxy functionality.*
- EST_ERROR **est_proxy_set_auth_mode** (EST_CTX ∗ctx, EST_HTTP_AUTH_MODE amode)

  est_proxy_set_auth_mode() *(p. 46) is used by an application layer to configure the HTTP authentication method to use for validating the identity of an EST client.*
- EST_ERROR **est_proxy_set_auth_cred_cb** (EST_CTX ∗ctx, auth_credentials_cb callback)

  est_proxy_set_auth_cred_cb() *(p. 45) is used by an application to register its callback function.*
- EST_ERROR **est_proxy_set_read_timeout** (EST_CTX ∗ctx, int timeout)

  est_proxy_set_read_timeout() *(p. 47) is used by an application to set timeout value of read operations. After the EST proxy sends a request to the EST server it will attempt to read the response from the server. This timeout value limits the amount of time the proxy will wait for the response.*
- EST_ERROR **est_proxy_set_server** (EST_CTX ∗ctx, const char ∗server, int port)

  est_proxy_set_server() *(p. 47) is called by the application layer to specify the address/port of the EST server. It must be called after* ***est_proxy_init()*** *(p. 45) and prior to issuing any EST commands.*

5.4.1 Function Documentation

5.4.1.1 EST_ERROR est_proxy_coap_init_start (EST_CTX ∗ *ctx*, int *port*)

est_proxy_coap_init_start() (p. 44) is called by the application layer to specify the address/port of the EST server. It must be called after **est_proxy_init()** (p. 45) and prior to starting the EST proxy functionality.

Parameters

ctx	Pointer to EST context for a client session
port	UDP port that this EST proxy will listen on

Returns

EST_ERROR EST_ERR_NONE - Success. EST_ERR_NO_CTX - NULL value passed for EST context EST_↩ ERR_BAD_MODE - This can only be called when in server or proxy mode EST_ERR_INVALID_PORT_NUM - Invalid port number input, less than zero or greater than 65535 EST_ERR_CLIENT_COAP_MODE_NOT_SUP↩ PORTED - Returned if libest has not been built with CoAP support. EST_ERR_COAP_PROCESSING_ERROR - Procesing error occurred with the CoAP library.

est_proxy_coap_init_start error checks its input parameters and then stores the port number into the EST context. It then initializes the CoAP library and setting up the resources with which to respond.

5.4.1.2 EST_CTX* est_proxy_init (unsigned char * *ca_chain,* int *ca_chain_len,* unsigned char * *cacerts_resp_chain,* int *cacerts_resp_chain_len,* EST_CERT_FORMAT *cert_format,* char * *http_realm,* X509 * *tls_id_cert,* EVP_PKEY * *tls_id_key,* char * *uid,* char * *pwd*)

est_proxy_init() (p. 45) is used by an application to create a context in the EST library. This context is used when invoking other functions in the API while in Proxy mode.

Parameters

ca_chain	Char array containing PEM encoded CA certs & CRL entries. This chain of certificates is used as the trust anchor when establishing a TLS connection.
ca_chain_len	Length of ca_chain char array.
cacerts_resp_chain	Char array containing PEM encoded CA certs to include in the /cacerts response. This is an optional parameter. If it set, it contains the chain of certificates used by the proxy to respond to GET CA Certs requests from EST Clients. If this parameter is not included, then the proxy will obtain the CA certificate chain from the configured upstream EST server. If this parameter is not NULL, then the correct length of this buffer must be specified in cacerts_resp_chain_len.
cacerts_resp_chain_len	Length of cacerts_resp_chain char array
cert_format	Specifies the encoding of the local and external certificate chains (PEM/DER).
http_realm	Char array containing HTTP realm name for HTTP auth
tls_id_cert	Pointer to X509 that contains the proxy's certificate for the TLS layer.
tls_id_key	Pointer to EVP_PKEY that contains the private key associated with the proxy's certificate.
uid	User ID to use for authenticating with server
pwd	Password to use for authenticating with server

This function allows an application to initialize an EST server context for proxy mode operation, which is used when operating as an RA. The application must provide the trusted CA certificates to use for server operation using the ca_chain parameter. This certificate set should include the explicit trust anchor certificate, any number of implicit trust anchor certificates, and any intermediate sub-CA certificates required to complete the chain of trust between the identity certificate passed into the tls_id_cert parameter and the root certificate for that identity certificate. The CA certificates should be encoded using the format specified in the cert_format parameter (e.g. PEM) and may contain CRL entries that will be used when authenticating EST clients connecting to the server. The applications must also provide the HTTP realm to use for HTTP authentication and the server certificate/private key to use for TLS.

Warning: Including additional intermediate sub-CA certificates that are not needed to complete the chain of trust may result in a potential MITM attack.

Returns

EST_CTX.

5.4.1.3 EST_ERROR est_proxy_set_auth_cred_cb (EST_CTX * *ctx,* auth_credentials_cb *callback*)

est_proxy_set_auth_cred_cb() (p. 45) is used by an application to register its callback function.

Parameters

ctx	EST context obtained from the **est_proxy_init()** (p. 45) call.
auth_credentials_cb	Function pointer to the application layer callback

The registered callback function is used by the EST client library to obtain authentication credentials. The application can provide authentication credentials during initialization if they are available, such as the userid and password used with HTTP basic authentication. During the processing of a request, the EST client library will call this application callback in the event that it does not have the authentication credentials that are being requested by the EST server.

The callback function definition must match the following function prototype,

int (∗auth_credentials_cb)(EST_HTTP_AUTH_HDR ∗auth_credentials);

auth_credentials - A pointer to a EST_HTTP_AUTH_HDR structure. The structure is provided by the EST library and the callback function fills in the specific credentials being requested. These credential values must be passed in the format in which they will be sent to the server, that is, the EST client library will perform no reformatting of these credentials. Ownership of the memory holding these credential values is transferred from the application layer to the EST library when the application layer returns these values to the EST library. This allows the EST library to free up this memory as soon as it is done using these values.

The return value from the callback must be one of the following values:

EST_HTTP_AUTH_CRED_SUCCESS - If the callback was able to provide the requested credentials. EST_HTTP_A↩ UTH_CRED_NOT_AVAILABLE - If the callback could not provide the requested credentials.

The auth_credentials_cb parameter can be set to NULL to reset the callback function.

All string parameters are NULL terminated strings.

Returns

EST_ERROR. EST_ERR_NONE - Success. EST_ERR_NO_CTX

5.4.1.4 EST_ERROR est_proxy_set_auth_mode (EST_CTX ∗ *ctx,* EST_HTTP_AUTH_MODE *amode*)

est_proxy_set_auth_mode() (p. 46) is used by an application layer to configure the HTTP authentication method to use for validating the identity of an EST client.

Parameters

ctx	Pointer to the EST proxy context. This was returned from **est_proxy_init()** (p. 45).
amode	Should be either AUTH_BASIC or AUTH_DIGEST

This function can optionally be invoked by the application layer to change the default HTTP authentication mode. The default mode is HTTP Basic authentication. An application may desire to use Digest authentication instead, in which case this function can be used to set that mode. This function should be invoked prior to starting the EST proxy.

Returns

EST_ERROR.

5.4.1.5 EST_ERROR est_proxy_set_read_timeout (EST_CTX * *ctx*, int *timeout*)

est_proxy_set_read_timeout() (p. 47) is used by an application to set timeout value of read operations. After the EST proxy sends a request to the EST server it will attempt to read the response from the server. This timeout value limits the amount of time the proxy will wait for the response.

Parameters

ctx	Pointer to the EST context
timeout	Integer value representing the read timeout in seconds. The minimum value is EST_SSL_READ_TIMEOUT_MIN and the maximum value is EST_SSL_READ_TIMEOUT_MAX.

Returns

EST_ERROR.

5.4.1.6 EST_ERROR est_proxy_set_server (EST_CTX * *ctx*, const char * *server*, int *port*)

est_proxy_set_server() (p. 47) is called by the application layer to specify the address/port of the EST server. It must be called after **est_proxy_init()** (p. 45) and prior to issuing any EST commands.

Parameters

ctx	Pointer to EST context for a client session
server	Name of the EST server to connect to. The ASCII string representing the name of the server is limited to 254 characters
port	TCP port on the EST server to connect

Returns

EST_ERROR EST_ERR_NONE - Success. EST_ERR_NO_CTX - NULL value passed for EST context EST_ERR_INVALID_SERVER_NAME - NULL value passed for EST server name, or server name string too long EST_ERR_INVALID_PORT_NUM - Invalid port number input, less than zero or greater than 65535

est_proxy_set_server error checks its input parameters and then stores both the hostname and port number into the EST context.

5.4.1.7 EST_ERROR est_proxy_start (EST_CTX * *ctx*)

est_proxy_start() (p. 47) is used by an application to start the EST proxy after **est_proxy_init()** (p. 45) and **est_proxy_set_server()** (p. 47) have been called and all the required callback functions have been provided by the application.

Parameters

ctx	Pointer to the EST context

libEST uses HTTP code from the Mongoose HTTP server. This function allows the application to start the HTTP services layer, which is required by EST.

Returns

EST_ERROR.

5.4.1.8 EST_ERROR est_proxy_stop (EST_CTX * *ctx*)

est_proxy_stop() (p. 48) is used by an application to stop the EST proxy. This should be called prior to **est_destroy()** (p. 20).

Parameters

ctx	Pointer to the EST context

libEST uses HTTP code from the Mongoose HTTP server. This function allows the application to stop the HTTP services layer.

Returns

EST_ERROR.

5.5 src/est/est_server.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include "est.h"
#include "est_server.h"
#include "est_server_http.h"
#include "est_locl.h"
#include "est_ossl_util.h"
#include "safe_mem_lib.h"
#include "safe_str_lib.h"
#include <openssl/x509.h>
#include <openssl/x509v3.h>
#include <openssl/cms.h>
#include <openssl/bio.h>
```

Macros

- #define **EST_SERVER_RSA_KEYSIZE_4096** 4096
- #define **EST_SERVER_DEFAULT_ENHCD_CERT_PWD** "cisco"

Functions

- void **est_send_http_error** (EST_CTX ∗ctx, void ∗http_ctx, int fail_code)
- EST_ERROR **est_server_send_http_retry_after** (EST_CTX ∗ctx, void ∗http_ctx, int delay)
- int **est_handle_cacerts** (EST_CTX ∗ctx, unsigned char ∗ca_certs, int ca_certs_len, void ∗http_ctx, char ∗path↩ _seg)
- int **est_server_handle_cacerts** (EST_CTX ∗ctx, void ∗http_ctx, char ∗path_seg)
- EST_ERROR **est_server_set_key_generation_cb** (EST_CTX ∗ctx, int(∗cb)(EVP_PKEY ∗∗priv_key))

  est_server_set_key_generation_cb() *(*p. *60) is used by an application to set a callback to handle the generation of the private key.*
- char ∗ **est_server_generate_auth_digest** (EST_HTTP_AUTH_HDR ∗ah, char ∗HA1)

  est_server_generate_auth_digest() *(*p. *56) is used by an application to calculate the HTTP Digest value based on the header values provided by an EST client.*
- EST_AUTH_STATE **est_enroll_auth** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, char ∗path_seg, EST_ENRO↩ LL_REQ_TYPE enroll_req, char ∗csr_buf, int csr_buf_len, X509_REQ ∗∗csr_p)
- EST_AUTH_STATE **est_brski_auth** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, char ∗path_seg)
- X509_REQ ∗ **est_server_parse_csr** (unsigned char ∗pkcs10, int pkcs10_len, EST_CSR_BASE64_DECODE base64_decode)
- int **est_tls_uid_auth** (EST_CTX ∗ctx, SSL ∗ssl, X509_REQ ∗req)
- int **est_server_check_csr** (X509_REQ ∗req)
- EST_ERROR **est_handle_simple_enroll** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, X509 ∗peer_cert, const char ∗ct, char ∗body, int body_len, char ∗path_seg, EST_ENROLL_REQ_TYPE enroll_req, unsigned char ∗∗returned_cert, int ∗returned_cert_len)
- EST_ERROR **est_handle_server_keygen** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, X509 ∗peer_cert, const char ∗ct, char ∗body, int body_len, char ∗path_seg, unsigned char ∗∗returned_cert, int ∗returned_cert_len, unsigned char ∗∗returned_key, int ∗returned_key_len)
- int **est_handle_csr_attrs** (EST_CTX ∗ctx, void ∗http_ctx, SSL ∗ssl, X509 ∗peer_cert, char ∗path_seg, unsigned char ∗∗returned_attrs, int ∗returned_attrs_len)
- int **est_http_request** (EST_CTX ∗ctx, void ∗http_ctx, char ∗method, char ∗uri, char ∗body, int body_len, const char ∗ct)
- EST_ERROR **est_server_start** (EST_CTX ∗ctx)

  est_server_start() *(*p. *61) is used by an application to start the EST server after* ***est_server_init()*** *(*p. *57) has been called and all the required callback functions have been provided by the application.*
- EST_ERROR **est_server_stop** (EST_CTX ∗ctx)

  est_server_stop() *(*p. *61) is used by an application to stop the EST server. This should be called prior to* ***est_destroy()*** *(*p. *20).*
- EST_CTX ∗ **est_server_init** (unsigned char ∗ca_chain, int ca_chain_len, unsigned char ∗cacerts_resp_chain, int cacerts_resp_chain_len, EST_CERT_FORMAT cert_format, char ∗http_realm, X509 ∗tls_id_cert, EVP_PKEY ∗tls_id_key)

  est_server_init() *(*p. *57) is used by an application to create a context in the EST library when operating as an EST server that fronts a CA. This context is used when invoking other functions in the API.*
- EST_ERROR **est_server_set_auth_mode** (EST_CTX ∗ctx, EST_HTTP_AUTH_MODE amode)

  est_server_set_auth_mode() *(*p. *58) is used by an application to configure the HTTP authentication method to use for validating the identity of an EST client.*
- EST_ERROR **est_set_ca_enroll_cb** (EST_CTX ∗ctx, int(∗cb)(unsigned char ∗pkcs10, int p10_len, unsigned char ∗∗pkcs7, int ∗pkcs7_len, char ∗user_id, X509 ∗peer_cert, char ∗path_seg, void ∗ex_data))

est_set_ca_enroll_cb() *(p. 63) is used by an application to install a handler for signing incoming PKCS10 requests.*

- EST_ERROR **est_set_ca_reenroll_cb** (EST_CTX *ctx, int(*cb)(unsigned char *pkcs10, int p10_len, unsigned char **pkcs7, int *pkcs7_len, char *user_id, X509 *peer_cert, char *path_seg, void *ex_data))

  est_set_ca_reenroll_cb() *(p. 63) is used by an application to install a handler for re-enrolling certificates.*

- EST_ERROR **est_set_server_side_keygen_enroll_cb** (EST_CTX *ctx, int(*cb)(unsigned char *pkcs10, int p10_len, unsigned char **pkcs7, int *pkcs7_len, unsigned char **pkcs8, int *pkcs8_len, char *user_id, X509 *peer_cert, char *path_seg, void *ex_data))

  est_set_srvr_side_keygen_enroll_cb() is used by an application to install a handler for signing incoming PKCS10 requests.

- EST_ERROR **est_set_csr_cb** (EST_CTX *ctx, unsigned char *(*cb)(int *csr_len, char *path_seg, X509 *peer_cert, void *ex_data))

  est_set_csr_cb() *(p. 64) is used by an application to install a handler for retrieving the CSR attributes from the CA server.*

- EST_ERROR **est_set_cacerts_cb** (EST_CTX *ctx, unsigned char *(*cb)(int *csr_len, char *path_seg, void *ex_data))

  est_set_cacerts_cb() *(p. 64) is used by an application to install a handler for retrieving the CA certs from the CA server.*

- EST_ERROR **est_set_http_auth_cb** (EST_CTX *ctx, int(*cb)(EST_CTX *ctx, EST_HTTP_AUTH_HDR *ah, X509 *peer_cert, char *path_seg, void *ex_data))

  est_set_http_auth_cb() *(p. 67) is used by an application to install a handler for authenticating EST clients.*

- EST_ERROR **est_set_http_auth_required** (EST_CTX *ctx, EST_HTTP_AUTH_REQUIRED required)

  est_set_http_auth_required() *(p. 67) is used by an application to define whether HTTP authentication should be required in addition to using client certificates.*

- LIBEST_API EST_ERROR **est_server_enable_enhanced_cert_auth** (EST_CTX *ctx, int local_pki_subj_↩ field_nid, const char *ah_pwd, EST_ECA_CSR_CHECK_FLAG csr_check_enabled)

  est_server_enable_enhanced_cert_auth() *(p. 53) is used by an application to enable the use of the Enhanced Certificate Authentication mode on the server. When this mode is enabled for an HTTP context, http authentication is required, and a valid certificate is received during authentication the authentication header will be automatically created. When the context being used is in CoAP mode and this feature is enabled, this feature will always be used during auth. The UserID field will be set to the value of a subject field parsed out of the cert received. The selection of this field is determined by the following decisions which are made in real-time by the server.*

- LIBEST_API EST_ERROR **est_server_enhanced_cert_auth_add_mfg_info** (EST_CTX *ctx, char *mfg_↩ name, int mfg_subj_field_nid, unsigned char *truststore_buf, int truststore_buf_len)

  est_server_enhanced_cert_auth_add_mfg_info() *(p. 56) is used by an application to add a new registered device manufacturer to be used with the enhanced cert auth mode. This new manufacturer information and truststore will then be used during the auth to determine which subject line field will be used for the user.*

- LIBEST_API EST_ERROR **est_server_disable_enhanced_cert_auth** (EST_CTX *ctx)

  est_server_disable_enhanced_cert_auth() *(p. 52) is used by an application to disable the use of Enhanced Certificate Authentication Mode on the server. When this mode is disabled the EST server will revert back to normal operation of HTTP Authentication. This means that instead of getting the user from the subject line of the received certificate and using a server set password, the HTTP Authentication headers received from the client's request will be used. In CoAP mode when Enhanced Cert Auth is off all successful DTLS connections will skip the auth step and continue on to perform the rest of the steps for the EST request.*

- EST_ERROR **est_server_enable_srp** (EST_CTX *ctx, int(*cb)(SSL *s, int *ad, void *arg))

  est_server_enable_srp() *(p. 54) is used by an application to enable the TLS-SRP authentication. This allows EST clients that provide SRP credentials at the TLS layer to be authenticated by the EST server. This function must be invoked to enable server-side SRP support.*

- EST_ERROR **est_server_enable_pop** (EST_CTX *ctx)

  est_server_enable_pop() *(p. 54) is used by an application to enable the proof-of-possession check on the EST server. This proves the EST client that sent the CSR to the server is in possession of the private key that was used to sign the CSR. This binds the TLS session ID to the CSR.*

- EST_ERROR **est_server_disable_pop** (EST_CTX *ctx)

  est_server_disable_pop() *(p. 52) is used by an application to disable the proof-of-possession check on the EST server. Please see the documentation for* ***est_server_enable_pop()*** *(p. 54) for more information on the proof-of-possession check.*

- EST_ERROR **est_server_set_retry_period** (EST_CTX ∗ctx, int seconds)

  est_server_set_retry_period() *(*p. *60) is used by an application to change the default retry-after period sent to the EST client when the CA server is not configured for auto-enroll. This retry-after value notifies the client about how long it should wait before attempting the enroll operation again to see if the CA has approved the original CSR.*
- EST_ERROR **est_server_set_ecdhe_curve** (EST_CTX ∗ctx, int nid)

  est_server_set_ecdhe_curve() *(*p. *59) is used by an application to specify the ECC curve that should be used for ephemeral diffie-hellman keys during the TLS handshake. Ephemeral diffie-hellman is enabled by libest and provides better forward secrecy. If the curve is not specified by the application using this function, then the prime256v1 curve is used as the default curve.*
- EST_ERROR **est_server_set_dh_parms** (EST_CTX ∗ctx, DH ∗parms)

  est_server_set_dh_parms() *(*p. *59) is used by an application to specify the Diffie-Hellman parameters to be used for single use DH key generation during the TLS handshake. If these parameters are not used, then single-use DH key generation is not enabled. This should be enabled to improve the forward secrecy of the TLS handshake operation.*
- EST_ERROR **est_server_init_csrattrs** (EST_CTX ∗ctx, char ∗csrattrs, int csrattrs_len)

  est_server_init_csrattrs() *(*p. *57) is used by an application to initialize a fixed set of CSR attributes. These attributes will be used by libest in response to a client CSR attributes request. The attributes must be an ASN.1 base64 encoded character string.*
- EST_ERROR **est_server_enable_tls10** (EST_CTX ∗ctx)

  est_server_enable_tls10() *(*p. *55) is a deprecated function. TLS 1.0 is a violation of RFC7030 and it is no longer supported by the EST library. This function will log an error message and return EST_ERR_BAD_MODE.*
- EST_ERROR **est_server_enforce_csrattr** (EST_CTX ∗ctx)

  est_server_enforce_csrattrs() is used by an application to enable checking of the CSR attributes on the EST server. When enabled, the EST client must provide all the CSR attributes that were in the /csrattrs response sent by the server. The enrollment will fail if the client fails to provide all the CSR attributes. This setting applies to simple enroll and reenroll operations. This setting applies only to server mode and has no bearing on proxy mode operation.
- EST_ERROR **est_server_set_read_timeout** (EST_CTX ∗ctx, int timeout)

  est_server_set_read_timeout() *(*p. *60) is used by an application to set timeout value of server read operations. Once a socket is opened the EST server begins attempting to read from this socket. This timeout value limits the amount of time the client will wait for the response. The default value is set to EST_SSL_READ_TIMEOUT_DEF.*
- EST_ERROR **est_set_brski_voucher_req_cb** (EST_CTX ∗ctx, brski_voucher_req_cb cb)

  est_set_brski_voucher_req_cb() *(*p. *62) is used by an application to install a handler for processing incoming BRSKI client voucher requests.*
- EST_ERROR **est_set_brski_voucher_status_cb** (EST_CTX ∗ctx, brski_voucher_status_cb cb)

  est_set_brski_voucher_status_cb() *(*p. *62) is used by an application to install a handler for processing incoming BRSKI client voucher status indications.*
- EST_ERROR **est_set_brski_enroll_status_cb** (EST_CTX ∗ctx, brski_enroll_status_cb cb)

  est_set_brski_enroll_status_cb() *(*p. *62) is used by an application to install a handler for processing incoming BRSKI client certificate status indications.*
- EST_ERROR **est_server_set_brski_retry_period** (EST_CTX ∗ctx, int seconds)

  est_server_set_brski_retry_period() *(*p. *58) is used by an application to change the default retry-after period sent to the BRSKI pledge when the registrar is not able to immediately provide the voucher. This retry-after value notifies the client how long to wait before attempting the voucher request operation again to see if the registrar is ready to respond with a voucher.*
- void **est_set_est_err_event_cb** (est_est_err_event_cb_t new_est_err_cb)

  est_set_est_err_event_cb() *(*p. *66) is used by an application to install a handler for receiving notification when EST_L↩ OG_ERR() is called with EST_LOG_LVL_ERR.*
- void **est_set_ssl_proto_err_event_cb** (est_ssl_proto_err_event_cb_t new_ssl_proto_err_cb)

  est_set_ssl_proto_err_event_cb() *(*p. *68) is used by an application to install a handler for receiving notification when an SSL protocol-related error occurs. (e.g. ossl_dump_ssl_errors() for OpenSSL errors)*
- void **est_set_enroll_req_event_cb** (EST_CTX ∗ctx, est_enroll_req_event_cb_t new_est_enroll_event_cb)

  est_set_enroll_req_event_cb() *(*p. *66) is used by an application to install a handler for receiving notification when an EST enroll request occurs.*

- void **est_set_enroll_rsp_event_cb** (EST_CTX *ctx, est_enroll_rsp_event_cb_t new_est_enroll_event_cb)

  est_set_enroll_rsp_event_cb() *(p. 66) is used by an application to install a handler for receiving notification when an EST enroll response occurs.*
- void **est_set_enroll_auth_result_event_cb** (EST_CTX *ctx, est_enroll_auth_result_event_cb_t new_est_↩ auth_result_cb)

  est_set_enroll_auth_result_event_cb() *(p. 65) is used by an application to install a handler for receiving notification when an EST enroll authentication result is received from the configured authentication service.*
- void **est_set_endpoint_req_event_cb** (EST_CTX *ctx, est_endpoint_req_event_cb_t new_endpoint_req_cb)

  est_set_endpoint_req_event_cb() *(p. 65) is used by an application to install a handler for receiving notification when an EST request is received from an endpoint.*
- void **est_invoke_est_err_event_cb** (char *format, va_list arg_list)
- void **est_invoke_ssl_proto_err_event_cb** (char *err_msg)
- EST_ERROR **est_invoke_enroll_get_ip_port** (EST_CTX *ctx, SSL *ssl, void *addr_info, char *src_ipstr, int src_ipstr_len, int *src_port)
- EST_ERROR **est_invoke_enroll_req_event_cb** (EST_CTX *ctx, SSL *ssl, X509 *peer_cert, unsigned char *csr_buf, int csr_len, void *addr_info, char *path_seg, EST_ENROLL_REQ_TYPE enroll_req)
- EST_ERROR **est_invoke_enroll_rsp_event_cb** (EST_CTX *ctx, SSL *ssl, X509 *peer_cert, unsigned char *csr_buf, int csr_len, void *addr_info, char *path_seg, EST_ENROLL_REQ_TYPE enroll_req, unsigned char *returned_cert, int returned_cert_len, EST_ERROR rc)
- void **est_invoke_endpoint_req_event_cb** (EST_CTX *ctx, X509 *peer_cert, SSL *ssl, void *addr_info, const char *uri, EST_ENDPOINT_EVENT_TYPE event_type)

5.5.1 Function Documentation

5.5.1.1 LIBEST_API EST_ERROR est_server_disable_enhanced_cert_auth (EST_CTX * *ctx*)

est_server_disable_enhanced_cert_auth() (p. 52) is used by an application to disable the use of Enhanced Certificate Authentication Mode on the server. When this mode is disabled the EST server will revert back to normal operation of HTTP Authentication. This means that instead of getting the user from the subject line of the received certificate and using a server set password, the HTTP Authentication headers received from the client's request will be used. In CoAP mode when Enhanced Cert Auth is off all successful DTLS connections will skip the auth step and continue on to perform the rest of the steps for the EST request.

Parameters

ctx	Pointer to the EST context

Returns

EST_ERROR.

The default mode is ENHANCED_CERT_AUTH_DISABLED. To use Enhanced Certificate Authentication mode the user will call the est_server_enable_enhanced_cert_auth API. Once enabled the user can then disable this mode using the this API. Disabling Enhancecd Cert Auth will destroy all manufacturer info and truststores currently registered.

5.5.1.2 EST_ERROR est_server_disable_pop (EST_CTX * *ctx*)

est_server_disable_pop() (p. 52) is used by an application to disable the proof-of-possession check on the EST server. Please see the documentation for **est_server_enable_pop()** (p. 54) for more information on the proof-of-possession check.

Parameters

ctx	Pointer to the EST context

This function may be called at any time.

Returns

EST_ERROR.

5.5.1.3 LIBEST_API EST_ERROR est_server_enable_enhanced_cert_auth (EST_CTX * *ctx,* int *local_pki_subj_field_nid,* const char * *ah_pwd,* EST_ECA_CSR_CHECK_FLAG *csr_check_enabled*)

est_server_enable_enhanced_cert_auth() (p. 53) is used by an application to enable the use of the Enhanced Certificate Authentication mode on the server. When this mode is enabled for an HTTP context, http authentication is required, and a valid certificate is received during authentication the authentication header will be automatically created. When the context being used is in CoAP mode and this feature is enabled, this feature will always be used during auth. The UserID field will be set to the value of a subject field parsed out of the cert received. The selection of this field is determined by the following decisions which are made in real-time by the server.

First the server attempts to verify its cert against the manufacturer truststores that have been registered via the enhanced cert auth APIs. If the client cert verifies against a manufacturer it grabs the field registered for that manufacturer from the certificate.

If the cert was not verified by any of the manufacturers, it is assumed that the cert is a part of the local pki domain and should use the subject field for the local pki domain.

If the CSR Check is enabled, the subject field used as the username will be compared with with the local pki domain subject field in the CSR to ensure that the identifying information will be copied into the newly enrolled local PKI domain cert.

SECURITY ISSUE: The Enhanced Cert Auth CSR Check can be bypassed if there are two or more different manufacturer NIDs being used. In this scenario, it is possible to masquerade as another device during a local PKI domain enrollment. If there is only one unique manufacturer NID then it can be ensured that identifying information from the manufacturer cert was copied into the local PKI domain and will continue to be copied into all local PKI domain CSRs during enrollment.

Use the est_server_enhanced_cert_auth_add_mfg_info API to add a new registered manufacturer to an EST context. This new manufacturer will then be used in the auth of a request as described before.

The password will be set to a user defined value or the default Cisco specific password.

Parameters

ctx	Pointer to the EST context
local_pki_subj_field_nid	integer name identifier (NID) for the subject field that should be obtained from local pki domain certificates
ah_pwd	String containing the desired auth header password. If set to NULL the default password "cisco" will be used.
csr_check_enabled	Flag specifying whether the Enhanced Cert Auth CSR copy check should be performed during authentication. The purpose of this check is to ensure that the identifying
Generated by Doxygen	information in the client's peer cert gets propagated into the newly enrolled local pki domain cert. If this check is on and the information wasn't copied into the csr, the enrollment will fail.

Returns

EST_ERROR.

The default mode is ENHANCED_CERT_AUTH_DISABLED. To use Enhanced Certificate Authentication mode the user will call this API. The user can then also call the est_server_enhanced_cert_auth_add_mfg_info API to add a new registered device manufacturer for use with this auth feature. If no registered manufacturer is added it will be assumed that all devices connecting are part of the local pki domain and use the subject field associated with the local PKI domain NID during auth. Finally, the user can disable this mode using the est_server_disable_enhanced_cert_auth API. Disabling this feature will remove all registered manufacturers.

5.5.1.4 EST_ERROR est_server_enable_pop (EST_CTX * *ctx*)

est_server_enable_pop() (p. 54) is used by an application to enable the proof-of-possession check on the EST server. This proves the EST client that sent the CSR to the server is in possession of the private key that was used to sign the CSR. This binds the TLS session ID to the CSR.

Note, if the CSR attributes configured on the server require PoP checking, then there is no need to call this function to enable PoP. The PoP will be enabled automatically under this scenario.

Note, PoP checking is not possible when an EST proxy is used to between the EST client and EST server. Since the proxy will not be in possession of the private key, an EST server would fail the PoP check. However, an EST proxy can enable this feature to ensure the EST client has the signing key.

Parameters

ctx	Pointer to the EST context

This function may be called at any time.

Returns

EST_ERROR.

5.5.1.5 EST_ERROR est_server_enable_srp (EST_CTX * *ctx,* int(*)(SSL *s, int *ad, void *arg) *cb*)

est_server_enable_srp() (p. 54) is used by an application to enable the TLS-SRP authentication. This allows EST clients that provide SRP credentials at the TLS layer to be authenticated by the EST server. This function must be invoked to enable server-side SRP support.

Parameters

ctx	Pointer to the EST context
cb	Function address of the application specific SRP verifier handler

This function should be invoked prior to starting the EST server. This is used to specify the handler for SRP authentication at the TLS layer. When a TLS-SRP cipher suite is negotiated at the TLS layer, the handler will be invoked by libest

to retrieve the SRP parameters for user authentication. Your application must provide the SRP parameters for the user.

The handler should use the following logic:

1. Invoke SSL_get_srp_username() to get the SRP user name from the TLS layer.
2. Lookup the user's SRP parameters in the application specific user database. These parameters include the N, g, s, and v parameters.
3. Invoke SSL_set_srp_server_param() to forward the SRP parameters to the TLS layer, allowing the TLS handshake to proceed.

libest includes an example server application that uses this handler for SRP support. This example uses the OpenSSL SRP verifier file capability to manage SRP parameters for individual users. Your application could use this approach, or it may utilize another facility for managing user specific SRP parameters. Please refer to RFC 2945 and RFC 5054 for a full understanding of SRP.

Returns

EST_ERROR.

5.5.1.6 EST_ERROR est_server_enable_tls10 (EST_CTX * *ctx*)

est_server_enable_tls10() (p. 55) is a deprecated function. TLS 1.0 is a violation of RFC7030 and it is no longer supported by the EST library. This function will log an error message and return EST_ERR_BAD_MODE.

Parameters

ctx	Pointer to the EST context

This function must be called prior to starting the EST server.

Returns

EST_ERROR.

5.5.1.7 EST_ERROR est_server_enforce_csrattr (EST_CTX * *ctx*)

est_server_enforce_csrattrs() is used by an application to enable checking of the CSR attributes on the EST server. When enabled, the EST client must provide all the CSR attributes that were in the /csrattrs response sent by the server. The enrollment will fail if the client fails to provide all the CSR attributes. This setting applies to simple enroll and reenroll operations. This setting applies only to server mode and has no bearing on proxy mode operation.

Parameters

ctx	Pointer to the EST context

This function must be called prior to starting the EST server.

Returns

EST_ERROR.

5.5.1.8 LIBEST_API EST_ERROR est_server_enhanced_cert_auth_add_mfg_info (EST_CTX * *ctx,* char * *mfg_name,* int *mfg_subj_field_nid,* unsigned char * *truststore_buf,* int *truststore_buf_len*)

est_server_enhanced_cert_auth_add_mfg_info() (p. 56) is used by an application to add a new registered device manufacturer to be used with the enhanced cert auth mode. This new manufacturer information and truststore will then be used during the auth to determine which subject line field will be used for the user.

Parameters

ctx	Pointer to the EST context
mfg_name	String containing the name of the device manufacturer
mfg_subj_field_nid	Integer name identifier (NID) for the subject field that should be obtained from certificates of devices that came from this manufacturer
truststore_buf	Buffer that contains the truststore to be used to identify a device as being from this manufacturer
truststore_buf_len	Integer length of the truststore buffer

Returns

EST_ERROR

This API must only be used after the EST context already has enhanced cert auth enabled. If no manufacturer is registered using this API it is assumed that all requests should use the local pki domain subject field during auth.

5.5.1.9 char* est_server_generate_auth_digest (EST_HTTP_AUTH_HDR * *ah,* char * *HA1*)

est_server_generate_auth_digest() (p. 56) is used by an application to calculate the HTTP Digest value based on the header values provided by an EST client.

Parameters

ah	Authentication header values from client, provided by libEST
HA1	The precalculated HA1 value for the user. HA1 is defined in RFC 2617. It's the MD5 calculation of the user's ID, HTTP realm, and the user's password.

This is a helper function that an application can use to calculate the HTTP Digest value when performing HTTP Digest Authentication of an EST client. libEST does not maintain a user database. This is left up to the application, with the intent of integrating an external user database (e.g. Radius/AAA).

The HA1 value should be calculated by the application as defined in RFC 2617. HA1 is the MD5 hash of the user ID, HTTP realm, and user password. This MD5 value is then converted to a hex string. HA1 is expected to be 32 bytes long.

Returns

char* containing the digest, or NULL if an error occurred.

5.5.1.10 EST_CTX* est_server_init (unsigned char * *ca_chain,* int *ca_chain_len,* unsigned char * *cacerts_resp_chain,* int *cacerts_resp_chain_len,* EST_CERT_FORMAT *cert_format,* char * *http_realm,* X509 * *tls_id_cert,* EVP_PKEY * *tls_id_key*)

est_server_init() (p. 57) is used by an application to create a context in the EST library when operating as an EST server that fronts a CA. This context is used when invoking other functions in the API.

Parameters

ca_chain	Char array containing PEM encoded CA certs & CRL entries
ca_chain_len	Length of ca_chain char array
cacerts_resp_chain	Char array containing PEM encoded CA certs to include in the /cacerts response
cacerts_resp_chain_len	Length of cacerts_resp_chain char array
cert_format	Specifies the encoding of the local and external certificate chains (PEM/DER).
http_realm	Char array containing HTTP realm name for HTTP auth
tls_id_cert	Pointer to X509 that contains the server's certificate for the TLS layer.
tls_id_key	Pointer to EVP_PKEY that contains the private key associated with the server's certificate.

This function allows an application to initialize an EST server context that is used with a CA (not an RA). The application must provide the trusted CA certificates to use for server operation using the ca_chain parameter. This certificate set should include the explicit trust anchor certificate, any number of implicit trust anchor certificates, and any intermediate sub-CA certificates required to complete the chain of trust between the identity certificate passed into the tls_id_cert parameter and the root certificate for that identity certificate. The CA certificates should be encoded using the format specified in the cert_format parameter (e.g. PEM) and may contain CRL entries that will be used when authenticating EST clients connecting to the server. The applications must also provide the HTTP realm to use for HTTP authentication and the server certificate/private key to use for the TLS stack to identify the server.

Warning: Including additional intermediate sub-CA certificates that are not needed to complete the chain of trust may result in a potential MITM attack.

Returns

EST_CTX.

5.5.1.11 EST_ERROR est_server_init_csrattrs (EST_CTX * *ctx,* char * *csrattrs,* int *csrattrs_len*)

est_server_init_csrattrs() (p. 57) is used by an application to initialize a fixed set of CSR attributes. These attributes will be used by libest in response to a client CSR attributes request. The attributes must be an ASN.1 base64 encoded character string.

Parameters

ctx	Pointer to the EST context
csrattrs	Pointer CSR attributes in ASN.1 base64 encoded format, a NULL pointer clears the attributes and length.
csrattrs_len	Length of the CSR attributes character string

The est_get_csr_cb callback function maintains precedence over this method for CSR attributes. If est_get_csr_cb is initialized by the application it will be used. If not, then libest will use the attributes initialized here.

This function should be called prior to starting the EST server. PoP configuration(est_server_enable_pop or est_server_disable_pop) should be called prior to this function.

Returns

EST_ERROR.

5.5.1.12 EST_ERROR est_server_set_auth_mode (EST_CTX * *ctx*, EST_HTTP_AUTH_MODE *amode*)

est_server_set_auth_mode() (p. 58) is used by an application to configure the HTTP authentication method to use for validating the identity of an EST client.

Parameters

ctx	Pointer to the EST context
amode	Must be one of the following: AUTH_BASIC, AUTH_DIGEST, AUTH_TOKEN

This function can optionally be invoked by the application to change the default HTTP authentication mode. The default mode is HTTP Basic authentication. An application may desire to use Digest or Token authentication instead, in which case this function can be used to set that mode. This function must be invoked prior to starting the EST server.

Returns

EST_ERROR.

5.5.1.13 EST_ERROR est_server_set_brski_retry_period (EST_CTX * *ctx*, int *seconds*)

est_server_set_brski_retry_period() (p. 58) is used by an application to change the default retry-after period sent to the BRSKI pledge when the registrar is not able to immediately provide the voucher. This retry-after value notifies the client how long to wait before attempting the voucher request operation again to see if the registrar is ready to respond with a voucher.

Parameters

ctx	Pointer to the EST context
seconds	Number of seconds the server will use in the retry-after response.

This function may be called at any time after a context has been created.

Returns

EST_ERROR.

5.5.1.14 EST_ERROR est_server_set_dh_parms (EST_CTX * *ctx*, DH * *parms*)

est_server_set_dh_parms() (p. 59) is used by an application to specify the Diffie-Hellman parameters to be used for single use DH key generation during the TLS handshake. If these parameters are not used, then single-use DH key generation is not enabled. This should be enabled to improve the forward secrecy of the TLS handshake operation.

The DH parameters provided through this API should not be hard-coded in the application. The parameters should be generated at the time of product installation. Reusing the parameters across multiple installations of the product results in a vulnerable product.

Parameters

ctx	Pointer to the EST context
parms	Pointer to OpenSSL DH parameters

This function must be called prior to starting the EST server.

Returns

EST_ERROR.

5.5.1.15 EST_ERROR est_server_set_ecdhe_curve (EST_CTX * *ctx*, int *nid*)

est_server_set_ecdhe_curve() (p. 59) is used by an application to specify the ECC curve that should be used for ephemeral diffie-hellman keys during the TLS handshake. Ephemeral diffie-hellman is enabled by libest and provides better forward secrecy. If the curve is not specified by the application using this function, then the prime256v1 curve is used as the default curve.

Parameters

ctx	Pointer to the EST context
nid	OpenSSL NID value for the desired curve

This function must be called prior to starting the EST server. The NID values are defined in <openssl/obj_mac.h>. Typical NID values provided to this function would include:

NID_X9_62_prime192v1 NID_X9_62_prime256v1 NID_secp384r1 NID_secp521r1

Returns

EST_ERROR.

5.5.1.16 EST_ERROR est_server_set_key_generation_cb (EST_CTX * *ctx,* int(*)(EVP_PKEY **priv_key) *cb*)

est_server_set_key_generation_cb() (p. 60) is used by an application to set a callback to handle the generation of the private key.

Parameters

ctx	EST_CTX
cb	The callback that will be used to generate the key. It should match unsigned char * (EVP_PKEY **, int *)

This function is used by an application to set a callback to handle the generation of the private key.

Returns

EST_ERROR

5.5.1.17 EST_ERROR est_server_set_read_timeout (EST_CTX * *ctx,* int *timeout*)

est_server_set_read_timeout() (p. 60) is used by an application to set timeout value of server read operations. Once a socket is opened the EST server begins attempting to read from this socket. This timeout value limits the amount of time the client will wait for the response. The default value is set to EST_SSL_READ_TIMEOUT_DEF.

Parameters

ctx	Pointer to the EST context
timeout	Integer value representing the read timeout in seconds. The minimum value is EST_SSL_READ_TIMEOUT_MIN and the maximum value is EST_SSL_READ_TIMEOUT_MAX.

Returns

EST_ERROR.

5.5.1.18 EST_ERROR est_server_set_retry_period (EST_CTX * *ctx,* int *seconds*)

est_server_set_retry_period() (p. 60) is used by an application to change the default retry-after period sent to the EST client when the CA server is not configured for auto-enroll. This retry-after value notifies the client about how long it should wait before attempting the enroll operation again to see if the CA has approved the original CSR.

Parameters

ctx	Pointer to the EST context
seconds	Number of seconds the server will use in the retry-after response.

This function may be called at any time after a context has been created.

Returns

EST_ERROR.

5.5.1.19 EST_ERROR est_server_start (EST_CTX * *ctx*)

est_server_start() (p. 61) is used by an application to start the EST server after **est_server_init()** (p. 57) has been called and all the required callback functions have been provided by the application.

Parameters

ctx	Pointer to the EST context

libest uses HTTP code from the Mongoose HTTP server. This function allows the application to start the HTTP services layer, which is required by EST.

Returns

EST_ERROR.

5.5.1.20 EST_ERROR est_server_stop (EST_CTX * *ctx*)

est_server_stop() (p. 61) is used by an application to stop the EST server. This should be called prior to **est_destroy()** (p. 20).

Parameters

ctx	Pointer to the EST context

libest uses HTTP code from the Mongoose HTTP server. This function allows the application to stop the HTTP services layer.

Returns

EST_ERROR.

5.5.1.21 EST_ERROR est_set_brski_enroll_status_cb (EST_CTX * *ctx*, brski_enroll_status_cb *cb*)

est_set_brski_enroll_status_cb() (p. 62) is used by an application to install a handler for processing incoming BRSKI client certificate status indications.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must be defined to be of the brski↩ _enroll_status_cb function prototype.

This function is called by libest when in server mode and receives a BRSKI /enrollstatus primitive. The callback function will be passed the JSON based status from the BRSKI client

Returns

EST_ERROR_NONE on success, or EST based error

5.5.1.22 EST_ERROR est_set_brski_voucher_req_cb (EST_CTX * *ctx*, brski_voucher_req_cb *cb*)

est_set_brski_voucher_req_cb() (p. 62) is used by an application to install a handler for processing incoming BRSKI client voucher requests.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must be defined to be of the brski↩ _voucher_req_cb function prototype.

This function is called by libest when in server mode and receives a BRSKI /requestvoucher request. The callback function will be passed the JSON based request from the BRSKI client

Returns

EST_ERROR_NONE on success, or EST based error

5.5.1.23 EST_ERROR est_set_brski_voucher_status_cb (EST_CTX * *ctx*, brski_voucher_status_cb *cb*)

est_set_brski_voucher_status_cb() (p. 62) is used by an application to install a handler for processing incoming B↩ RSKI client voucher status indications.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must be defined to be of the brski↩ _voucher_status_cb function prototype.

This function is called by libest when in server mode and receives a BRSKI /voucher_status request. The callback function will be passed the JSON based response from the BRSKI client

Returns

EST_ERROR_NONE on success, or EST based error

5.5.1.24 EST_ERROR est_set_ca_enroll_cb (EST_CTX * *ctx,* int(*)(unsigned char *pkcs10, int p10_len, unsigned char **pkcs7, int *pkcs7_len, char *user_id, X509 *peer_cert, char *path_seg, void *ex_data) *cb*)

est_set_ca_enroll_cb() (p. 63) is used by an application to install a handler for signing incoming PKCS10 requests.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must match the following prototype:

```
int func(unsigned char*, int, unsigned char**, int*, char*, X509*, char *, void *);
```

This function is called by libest when a certificate request needs to be signed by the CA server. The application will need to forward the request to the signing authority and return the response. The response should be a PKCS7 signed certificate.

Returns

EST_ERROR.

5.5.1.25 EST_ERROR est_set_ca_reenroll_cb (EST_CTX * *ctx,* int(*)(unsigned char *pkcs10, int p10_len, unsigned char **pkcs7, int *pkcs7_len, char *user_id, X509 *peer_cert, char *path_seg, void *ex_data) *cb*)

est_set_ca_reenroll_cb() (p. 63) is used by an application to install a handler for re-enrolling certificates.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must match the following prototype:

```
int func(unsigned char*, int, unsigned char**, int*, char*, X509*)
```

This function is called by libest when a certificate needs to be renewed by the CA server. The application will need to forward the request to the signing authority and return the response. The response should be a PKCS7 signed certificate.

Returns

EST_ERROR.

5.5.1.26 EST_ERROR est_set_cacerts_cb (EST_CTX * *ctx,* unsigned char *(*)(int *csr_len, char *path_seg, void *ex_data) *cb*)

est_set_cacerts_cb() (p. 64) is used by an application to install a handler for retrieving the CA certs from the CA server.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must match the following prototype:

```
unsigned char *(*cb)(int *csr_len, char *path_seg, void *ex_data)
```

This function is called by libest when a CAcerts request is received. The CA certs chain is provided by the application layer and returned as a char array.

Returns

EST_ERROR.

5.5.1.27 EST_ERROR est_set_csr_cb (EST_CTX * *ctx,* unsigned char *(*)(int *csr_len, char *path_seg, X509 *peer_cert, void *ex_data) *cb*)

est_set_csr_cb() (p. 64) is used by an application to install a handler for retrieving the CSR attributes from the CA server.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must match the following prototype:

```
unsigned char *(*cb)(int*csr_len, char *path_seg, void *ex_data)
```

This function is called by libest when a CSR attributes request is received. The attributes are provided by the CA server and returned as a char array.

Returns

EST_ERROR.

5.5.1.28 void est_set_endpoint_req_event_cb (EST_CTX * *ctx,* est_endpoint_req_event_cb_t *new_endpoint_req_cb*)

est_set_endpoint_req_event_cb() (p. 65) is used by an application to install a handler for receiving notification when an EST request is received from an endpoint.

Parameters

ctx	EST context with which to register.
new_est_endpoint_req_cb	Callback function to call when an EST endpoint request is received. If NULL, then the callback function reverts to being disabled.

The callback function must be of type est_endpoint_req_event_cb_t.

Returns

None.

5.5.1.29 void est_set_enroll_auth_result_event_cb (EST_CTX * *ctx,* est_enroll_auth_result_event_cb_t *new_est_auth_result_cb*)

est_set_enroll_auth_result_event_cb() (p. 65) is used by an application to install a handler for receiving notification when an EST enroll authentication result is received from the configured authentication service.

Parameters

ctx	EST context with which to register.
new_est_enroll_auth_result_cb	Callback function to call when an EST enroll authentication result is received. If NULL, then the callback function reverts to being disabled.

The callback function must be of type est_enroll_auth_result_event_cb_t.

Returns

None.

5.5.1.30 void est_set_enroll_req_event_cb (EST_CTX * *ctx,* est_enroll_req_event_cb_t *new_est_enroll_event_cb*)

est_set_enroll_req_event_cb() (p. 66) is used by an application to install a handler for receiving notification when an EST enroll request occurs.

Parameters

ctx	EST context with which to register.
new_est_enroll_cb	Callback function to call when an EST enroll request occurs. If NULL, then the callback function reverts to being disabled.

The callback function must be of type est_enroll_req_event_cb_t.

Returns

None.

5.5.1.31 void est_set_enroll_rsp_event_cb (EST_CTX * *ctx,* est_enroll_rsp_event_cb_t *new_est_enroll_event_cb*)

est_set_enroll_rsp_event_cb() (p. 66) is used by an application to install a handler for receiving notification when an EST enroll response occurs.

Parameters

ctx	EST context with which to register.
new_est_enroll_cb	Callback function to call when an EST enroll response occurs. If NULL, then the callback function reverts to being disabled.

The callback function must be of type est_enroll_rsp_event_cb_t.

Returns

None.

5.5.1.32 void est_set_est_err_event_cb (est_est_err_event_cb_t *new_est_err_cb*)

est_set_est_err_event_cb() (p. 66) is used by an application to install a handler for receiving notification when EST↩ _LOG_ERR() is called with EST_LOG_LVL_ERR.

Parameters

new_est_err_cb	Callback function to call when EST_LOG_ERR() called. If NULL, then the callback function reverts to being disabled.

Because est_log() does not have an EST_CTX argument, only one callback may be registered at a time.

The callback function must be of type est_err_cb_t.

Returns

None.

5.5.1.33 EST_ERROR est_set_http_auth_cb (EST_CTX * *ctx,* int(*)(EST_CTX *ctx, EST_HTTP_AUTH_HDR *ah, X509 *peer_cert, char *path_seg, void *ex_data) *cb*)

est_set_http_auth_cb() (p. 67) is used by an application to install a handler for authenticating EST clients.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must match the following prototype:

int (*cb)(EST_CTX *ctx, EST_HTTP_AUTH_HDR *ah, X509 *peer_cert, char *path_seg, void *ex_data)

This function is called by libest when performing HTTP authentication. libest will pass the EST_HTTP_AUTH_HDR struct to the application, allowing the application to hook into a Radius, AAA, or some user authentication database. The X509 certificate from the TLS peer (EST client) is also provided through this callback facility, allowing the application layer to check for specific attributes in the X509 certificate such as an 802.1AR device ID. In addition, the path segment string is passed up if there was one in the request URI.

Returns

EST_ERROR.

5.5.1.34 EST_ERROR est_set_http_auth_required (EST_CTX * *ctx,* EST_HTTP_AUTH_REQUIRED *required*)

est_set_http_auth_required() (p. 67) is used by an application to define whether HTTP authentication should be required in addition to using client certificates.

Parameters

ctx	Pointer to the EST context
required	Flag indicating that HTTP authentication is required. Set to HTTP_AUTH_REQUIRED value to require HTTP auth. Set to HTTP_AUTH_NOT_REQUIRED if HTTP auth should occur only when TLS client authentication fails.

Returns

EST_ERROR.

The default mode is HTTP_AUTH_REQUIRED. This means that HTTP authentication will be attempted even when TLS client authentication succeeds. If HTTP authentication is only needed when TLS client auth fails, then set this to HTTP_AUTH_NOT_REQUIRED.

5.5.1.35 EST_ERROR est_set_server_side_keygen_enroll_cb (EST_CTX * *ctx*, int(*)(unsigned char *pkcs10, int p10_len, unsigned char **pkcs7, int *pkcs7_len, unsigned char **pkcs8, int *pkcs8_len, char *user_id, X509 *peer_cert, char *path_seg, void *ex_data) *cb*)

est_set_srvr_side_keygen_enroll_cb() is used by an application to install a handler for signing incoming PKCS10 requests.

Parameters

ctx	Pointer to the EST context
cb	Function address of the handler

This function must be called prior to starting the EST server. The callback function must match the following prototype:

```
int func (unsigned char *, int, unsigned char **, int *, unsigned char **,
          int *, char *, X509 *, char *, void *)
```

This function is called by libEST when a server keygen request had been made to the server server. The application will generate a key via this callback

Returns

EST_ERROR.

5.5.1.36 void est_set_ssl_proto_err_event_cb (est_ssl_proto_err_event_cb_t *new_ssl_proto_err_cb*)

est_set_ssl_proto_err_event_cb() (p. 68) is used by an application to install a handler for receiving notification when an SSL protocol-related error occurs. (e.g. ossl_dump_ssl_errors() for OpenSSL errors)

Parameters

new_ssl_proto_err_cb	Callback function to call when an SSL protocol-related error occurs. If NULL, then the callback function reverts to being disabled.

Because est_log() does not have an EST_CTX argument, only one callback may be registered at a time.

The callback function must be of type ssl_proto_err_cb_t.

Returns

None.

5.6 src/est/est_server_http.c File Reference

```
#include <openssl/err.h>
#include <openssl/ssl.h>
#include <openssl/x509v3.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/poll.h>
#include <errno.h>
#include <signal.h>
#include <fcntl.h>
#include <time.h>
#include <stdlib.h>
#include <stdarg.h>
#include <assert.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <stddef.h>
#include <stdio.h>
#include "safe_lib.h"
#include "safe_str_lib.h"
#include "safe_mem_lib.h"
#include "est.h"
#include "est_locl.h"
#include "est_ossl_util.h"
#include "est_server_http.h"
#include "est_server.h"
```

Data Structures

- struct **vec**
- struct **file**

Macros

- #define **_LARGEFILE_SOURCE**
- #define **__STDC_FORMAT_MACROS**
- #define **__STDC_LIMIT_MACROS**
- #define **MONGOOSE_VERSION** "3.5"
- #define **PASSWORDS_FILE_NAME** ".htpasswd"
- #define **MG_BUF_LEN** 8192
- #define **MAX_REQUEST_SIZE** 16384
- #define **ARRAY_SIZE**(array) (sizeof(array) / sizeof(array[0]))

- #define **_DARWIN_UNLIMITED_SELECT**
- #define **MSG_NOSIGNAL** 0
- #define **SOMAXCONN** 100
- #define **PATH_MAX** 4096
- #define **STRUCT_FILE_INITIALIZER** { 0, 0, 0, NULL, NULL }
- #define **MAX_SRC_ADDR** 20
- #define **send_http_error** mg_send_http_error
- #define **MSEC_POLL_WAIT_TIME** 250 /∗ .25 second wait on the the poll ∗/
- #define **HEXTOI**(x) (isdigit(x) ? x - '0' : x - 'W')
- #define **MAX_AUTH_HDR_LEN** (256)+(EST_MAX_PATH_SEGMENT_LEN)

Functions

- const void ∗ **mg_get_conn_ssl** (struct mg_connection ∗conn)
- const char ∗ **mg_version** (void)
- struct mg_request_info ∗ **mg_get_request_info** (struct mg_connection ∗conn)
- const char ∗ **mg_get_header** (const struct mg_connection ∗conn, const char ∗name)
- void **mg_send_http_error** (struct mg_connection ∗conn, int status, const char ∗reason, const char ∗fmt,...)
- int **mg_read** (struct mg_connection ∗conn, void ∗buf, size_t len)
- int **mg_write** (struct mg_connection ∗conn, const void ∗buf, size_t len)
- EST_HTTP_AUTH_HDR_RESULT **mg_parse_auth_header** (struct mg_connection ∗conn, EST_HTTP_AUT↩ H_HDR ∗ah)
- void **mg_send_authorization_request** (struct mg_connection ∗conn)
- EST_ERROR **est_server_handle_request** (EST_CTX ∗ctx, int fd)

  est_server_handle_request() *(p. 70) is used by an application to process an EST request. The application is responsible for opening a listener socket. When an EST request comes in on the socket, the application uses this function to hand-off the request to libEST.*
- void **mg_stop** (struct mg_context ∗ctx)
- struct mg_context ∗ **mg_start** (void ∗user_data)
- EST_ERROR **est_send_csrattr_data** (EST_CTX ∗ctx, char ∗csr_data, int csr_len, void ∗http_ctx)

5.6.1 Function Documentation

5.6.1.1 EST_ERROR est_server_handle_request (EST_CTX ∗ *ctx,* int *fd*)

est_server_handle_request() (p. 70) is used by an application to process an EST request. The application is responsible for opening a listener socket. When an EST request comes in on the socket, the application uses this function to hand-off the request to libEST.

Parameters

ctx	Pointer to the EST_CTX, which was provided when **est_server_init()** (p. 57) or **est_proxy_init()** (p. 45) was invoked.
fd	File descriptor that will be read to retrieve the HTTP request from the client. This is typically a TCP socket file descriptor.

est_server_handle_request() (p. 70) is used by an application when an incoming EST request needs to be processed.

This request would be a cacerts, simpleenroll, reenroll, or csrattrs request. This is used when implementing an EST server. The application is responsible for opening and listening to a TCP socket for incoming EST requests. When data is ready to be read from the socket, this API entry point should be used to allow libEST to read the request from the socket and respond to the request.

Returns

EST_ERROR.

Index