CryptoEngine and Crypt API Technical Reference

Interniche Legacy Document

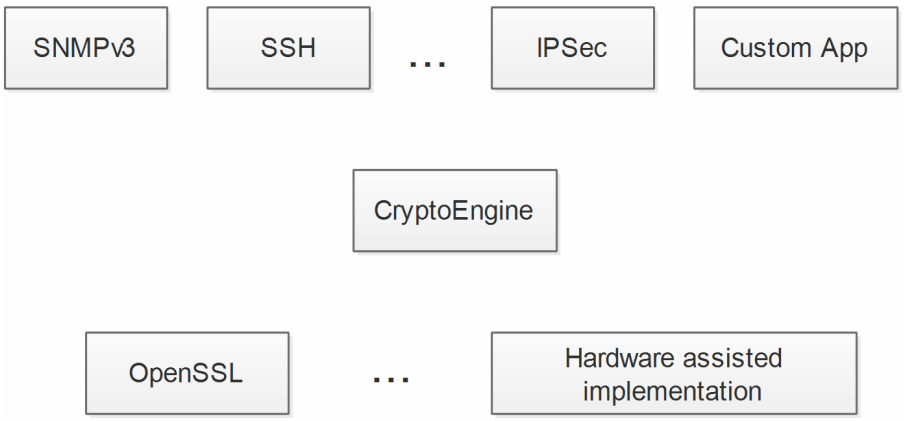
Version 1.00

Date: 05-May-2017 11:22

Table of Contents

1 Introduction

The CryptoEngine™ module provides an intermediate layer between security- enabled protocols (e.g., SNMPv3), security-related protocols (e.g., IPsec), proprietary security-enabled applications and a cryptography provider.



The complete list of supported operations is:

1. Encryption, decryption (via symmetric and asymmetric algorithms)
2. Digest (plain and HMAC) computation and validation
3. Generation of a shared secret via Diffie-Hellman
4. Signature generation and validation
5. X.509 certificate processing
6. Infrastructure (read (or write) keys and certificates from file or memory, base64 encoding and decoding, random number generation)

The complete list of security-enabled and security-related protocols is:

- ESMTP
- HTTP
- IKE
- IPSec
- PPP
- SNMPv3
- SSH
- SSL/TLS (NOTE: SSL/TLS implementations make direct crypto-library calls instead of using CE)

The IPSec, IKE, PPP, SNMPv3, and SSH modules all use functions in the CE API to perform cryptographic operations. This will make it easier to incorporate a new cryptographic provider into the build in the future.

The 4.1 InterNiche release supports the following three cryptography providers:

- OpenSSL
- Hardware-accelerated implementation (developed by InterNiche)

2 Terms and Conventions

CE

Throughout this document, the term "CE" is a shorthand notation for InterNiche's CryptoEngine.

Cryptography provider

A cryptography provider is a module that provides security-related functionality. This includes features such as encryption, decryption, digest authentication, digest validation, signature generation, and signature validation. An InterNiche build can include one or more cryptography providers. A provider may provide a software-only implementation or a hardware-only implementation or some combination of a software and hardware implementation.

3 Technical Overview

The CryptoEngine module exports a set of APIs to perform various security- related operations. Each function that is a part of this API has a name that starts with the "ce_" prefix.

3.1 Supported Algorithms

The CryptoEngine supports the following algorithms:

- DES, 3DES, AES
- MD4, MD5, SHA1, SHA2-256, SHA2-384, SHA2-512
- RSA
- DSA
- Diffie-Hellman
- X.509 certificates
- Base64 encoder/decoder

The CryptoEngine code exists as an InterNiche module; however, the module does not contain a task, and so a CE APIs execute in the context of the invoking task.

The CE code is designed to allow it to operate in a pre-emptive multitasking environment. The underlying cryptographic provider may have its own restrictions in this regard.

A call to a function in the CE API will not return until the requested operation has completed. The CE module does not support asynchronous callbacks.

The CE does not allow the dynamic (i.e., run-time) addition and deletion of providers.

Please note that some cryptographic providers will invoke a hardware- assisted implementation internally if the corresponding `#define` is enabled in their build.

4 Principles of Operation

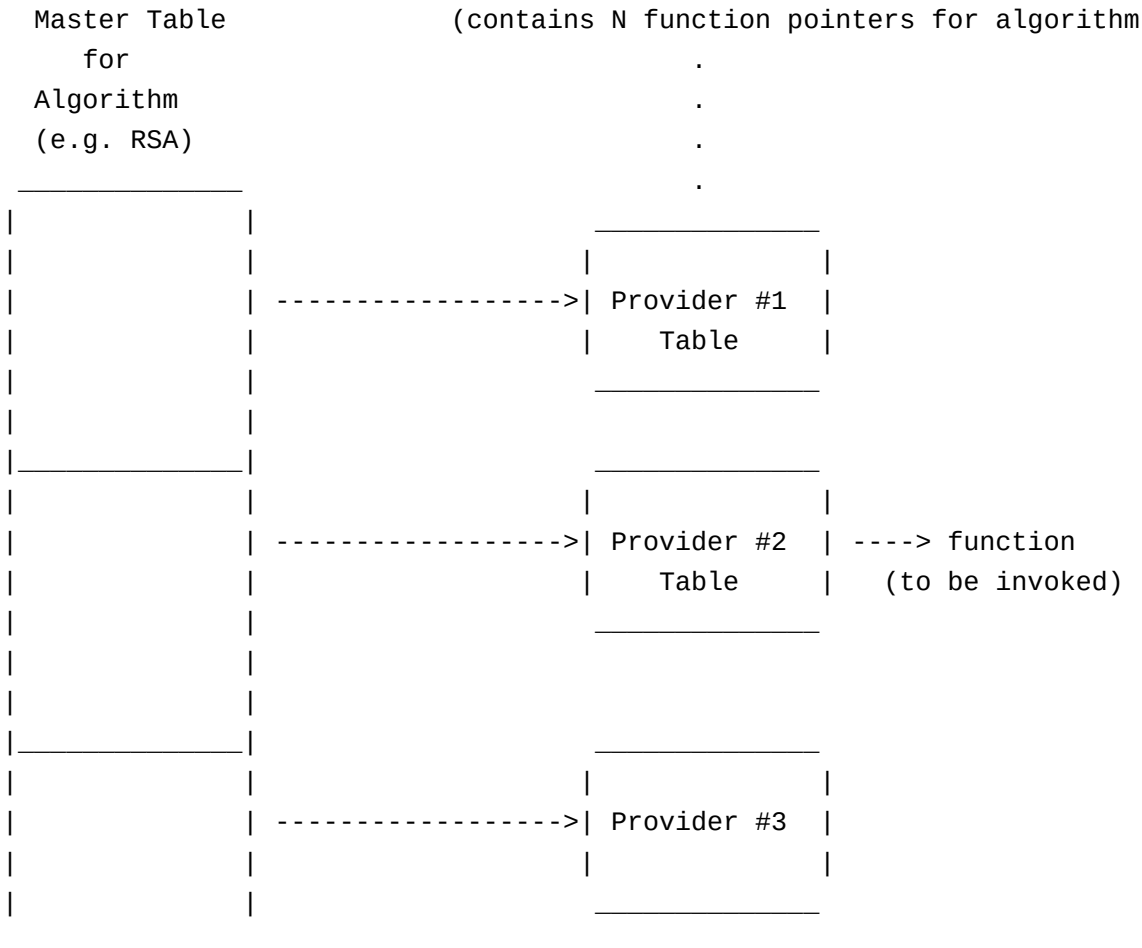
The user invokes a `ce_xxx()` function to perform the requested cryptographic operation. Here is an example API that performs symmetric encryption:

```
/* perform symmetric encryption (DES, 3DES, AES) */
int ce_sym_priv(CE_OP_PTR op, u_char *in, int inlen, u_char *iv, int ivlen,
                u_char *key, int keylen, u_char *out, int outlen);
```

The first parameter 'op' points to a `CE_OP` data structure that describes the operation to be performed (e.g., symmetric encryption via AES) and the cryptography provider to be used. The contents of the 'op' parameter are described in greater detail in a subsequent section.

Each provider registers one (or more) function pointer(s) in an algorithm-specific table to support the functionality provided by that algorithm. A master table for that algorithm contains pointers to one (or more) provider tables for the providers that support that algorithm.

The CryptoEngine module invokes the appropriate function based on the 'op' parameter.



4.1 Controlling #defines

Depending upon the set of cryptography providers included in the build, a user will include one (or more) of the following lines in their `ipport.h` file.

```
/* ipport.h */
#define CRYPT_PROVIDER1 0 /* OpenSSL */
#define CRYPT_PROVIDER2 1 /* hardware-assisted implementation (InterNiche) */
#define CRYPT_PROVIDER3 2 /* future XXX provider */
```

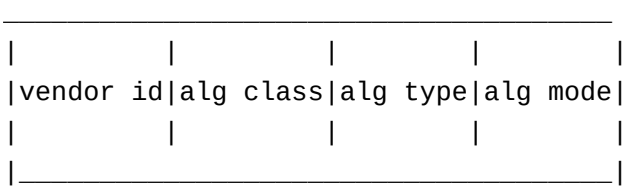
These `#defines` control the values of the `PROVIDER` enumeration.

```
 /* ce.h */
enum PROVIDER
{
#ifdef CRYPT_PROVIDER1
   OPENSSL_PROVIDER,
#endif

#ifdef CRYPT_PROVIDER2
   INICHE_HW_PROVIDER,
#endif

#ifdef CRYPT_PROVIDER3
   WCRYPT_PROVIDER,
#endif
};
```

4.2 Cipher Id

The CE uses the notion of a cipher id to uniquely identify an implementation of a security algorithm. The cipher id is an integer that has four byte-sized components:

1. Vendor id (e.g., OpenSSL)
2. Algorithm class (e.g., asymmetric privacy, authentication, key agreement, symmetric privacy, etc.)
3. Algorithm type (DES, 3DES, MD5, SHA1, etc.)
4. Algorithm mode (CBC, ECB, CFB, CTR, and OFB; only present for certain algorithm types)




```
struct ce_cipherid
{
   unsigned int vid:8;
   unsigned int algclass:8;
   unsigned int algid:8;
   unsigned int algmode:8;
};

typedef struct ce_cipherid CE_CID;

/* CryptoEngine operation specifier - used to specify the type of operation
 * requested (provided as the first parameter in every CE API call)
 */
struct ce_op
{
   CE_CID cid;
   void *specific;
   int *error;
};

typedef struct ce_op CE_OP;
typedef struct ce_op *CE_OP_PTR;

#define CE_DEFAULT_PROVIDER (OPENSSL_PROVIDER|CE_HW_PREFERRED)
```

Only 7 bits of the vendor id field are used to identify a cryptography provider. A value of 1 in the most significant bit is used to indicate that a hardware implementation (if one exists) of a particular algorithm is preferred to the software implementation. For example, a value of 128 in the vendor id field translates to the following statement: "I would like to perform an `AES-128-CBC` encryption via a function in the zero-th provider in the `PROVIDER` enumeration; however, if a hardware-assisted solution is available, please use that instead."

The algorithm class field is set to any one of the following values:

- CE_ALGCLASS_ASYM (asymmetric algorithms, e.g, RSA, DSA)
- CE_ALGCLASS_AUTH (authentication algorithms, e.g, MD5, SHA1, etc.)
- CE_ALGCLASS_BASE64 (base 64 encoding and decoding)
- CE_ALGCLASS_BN (big number arithmetic)
- CE_ALGCLASS_KAG (key agreement (e.g., Diffie-Hellman))
- CE_ALGCLASS_NULL
- CE_ALGCLASS_RAND (random number generation)
- CE_ALGCLASS_SYMPRIV (symmetric privacy algorithms, e.g., 3DES, AES)
- CE_ALGCLASS_X509CERT (X.509 certificates)

The algorithm type field is set to any one of the following values:

- CE_ALG_DES
- CE_ALG_3DES
- CE_ALG_AES
- CE_ALG_MD4
- CE_ALG_MD5
- CE_ALG_SHA1
- CE_ALG_SHA2_256
- CE_ALG_SHA2_384
- CE_ALG_SHA2_512
- CE_ALG_HMAC_MD4
- CE_ALG_HMAC_MD5
- CE_ALG_HMAC_SHA1
- CE_ALG_HMAC_SHA2_256
- CE_ALG_HMAC_SHA2_384
- CE_ALG_HMAC_SHA2_512
- CE_ALG_RSA
- CE_ALG_RSA
- CE_ALG_DH

The algorithm mode field is only relevant for the symmetric privacy algorithms, and must be set to one of the following values:

- CE_BLK_MODE_ECB
- CE_BLK_MODE_CBC
- CE_BLK_MODE_CFB
- CE_BLK_MODE_CTR
- CE_BLK_MODE_OFB

4.3 CryptoEngine Table Example

The following sample code provides support for various authentication algorithms in the CryptoEngine.

```
#ifdef USE_AUTH
#ifdef CRYPT_PROVIDER1
AUTHFP ce_auth_tbl_o[] =
{
   /* OpenSSL provider */
#ifdef USE_MD4
   ce_openssl_md4,
#endif
#ifdef USE_MD5
   ce_openssl_md5,
#endif
#ifdef USE_SHA1
   ce_openssl_sha1,
#endif
#ifdef USE_SHA2
#ifdef USE_SHA2_256
   ce_openssl_sha2_256,
#endif
#ifdef USE_SHA2_384
   ce_openssl_sha2_384,
#endif
#ifdef USE_SHA2_512
   ce_openssl_sha2_512,
#endif
#endif
};
#endif

...

AUTHFP *ce_auth_mtbl[] =
{
#ifdef CRYPT_PROVIDER1
   ce_auth_tbl_o,
#endif
#ifdef CRYPT_PROVIDER2
   ce_auth_tbl_h,
#endif
#ifdef CRYPT_PROVIDER3
   ce_auth_tbl_w,
#endif
};
```

4.4 Adding a new algorithm or provider

To add a new authentication algorithm to the table of authentication algorithms, it is necessary to create a new function pointer entry for all of the supported vendors. The function pointer entry may be NULL for one or more vendor-specific tables.

To include a NEW library for the implementation of the authentication algorithms in the build, one must create a new array (`ce_auth_tbl_NEW[]`), and populate it with the relevant function pointers. He must also add the table into the authentication master table (`ce_auth_mtbl[]`). The user will also need to add a provider id to the master list of provider ids (in `ipport.h`) and `PROVIDER` enumeration (in `ce.h`).

5 X.509 Certificate Authentication

The following paragraph provides information on the various steps required to configure X.509 certificate-based authentication for an IKE negotiation.

Note: Proper validation of an X.509 certificate requires the presence of a mechanism (such as InterNiche's SNTPv4 client) to obtain the current time.

5.1 Example

The configuration example uses two nodes: `Node_1` and `Node_2`. Each end must be configured to use X.509 certificate-based authentication. This is done by setting the `'-m'` option of the '`remote`' CLI command to '`RSASIG`'. (An alternate method of authentication uses a pre-shared key ('PresharedKey').)

```
[Node_1]<-----------secure connection via IKE/IPsec----------->[Node_2]
```

Each end identifies itself to the other by presenting it with a chain of one (or more) X.509 certificates that identify the sender. The chain must contain the end-user certificate, and may contain other intermediate-level or root-level certificates. Certificates other than the end-user certificate only need to be sent if the peer does not already have those certificates. (However, please note that a peer will typically only trust its own copy of a root-level certificate.)

In our example, `Node_1` identifies itself with the following two certificates.

1. supamcert.pem
2. cacert.pem

`Node_2` identifies itself with the following two certificates:

1. supascert.pem
2. cacert.pem

The '`supamcert.pem`' certificate belongs to the "Support Engineer (Americas)" entity. The '`supascert.pem`' certificate belongs to the "Support Engineer (Asia)" entity.

The '`supamcert.pem`' and '`supascert.pem`' certificates are signed by the "InterNiche Technologies, Inc." entity. The certificate for this top-level entity (Certificate Authority (CA)) is available in '`cacert.pem`', and is self-signed. The chain of two certificates (listed above for each of the two ends) is complete, i.e., no other certificates are required to validate the authenticity of '`supamcert.pem`' or '`supascert.pem`'.

All of these certificates were created via the OpenSSL certificate authority tool on a FreeBSD system.

The CryptoEngine module supports a X.509 certificate database. The database contains multiple application-specific domains. Each domain is identified by a numeric identifier. The complete list of domain identifiers is available in h/ce.h. (IKE uses the CE_X509CERT_DOMAIN_IKE domain identifier (with a value of 1) to store its certificates.) All certificates used in the system (regardless of whether they can be used to sign other certificates or not) must be added to a specific domain in this database. A certificate can be added to the database via the '-a' option to the 'x509' CLI command. The following command adds the supamcert.pem.pem certificate to the database. It also specifies the application-specific database domain identifier to which the certificate is being added via the '-d' option, location of the encrypted private key via the '-k' option, and the passphrase used for encryption via the '-h' option.

```
ce x509 -a supamcert.pem -d 1 -k supamkey.pem -h secret
```

If the private key is in the "clear" (i.e., it is not encrypted), the -h option is not required.

```
ce x509 -a supamcert.pem -d 1 -k supamkeyc.pem
```

The top-level CA certificate must also be added to the database via the following command. (Additional intermediate-level certificates can also be added as required.)

```
ce x509 -a cacert.pem -d 1
```

When authenticating via X.509 certificates, the user must also configure the use of ASN.1 Distinguished Names (DN) as the local ID (identifier). This is the only form of identifier that is allowed when using X.509 certificate-based authentication. The ASN.1 DNs for the two nodes are specified below.

```
[Node_1]

"CN=Support Engineer (Americas)/ST=California/C=US/Email=support-americas@iniche

[Node_2]

"CN=Support Engineer (Asia)/ST=California/C=US/Email=support-asia@iniche.com/O=I
```

The following section is pertinent only to users of InterNiche's IKE.

Here is an example of the 'ike remote' CLI command that specifies the use of ASN.1 DNs as identifiers for the two ends. Although the -f (peer identifier) and -g (local identifier) options require a <type>: <value> format, the value of the DN does not need to be specified at the command line (since it is available in the X.509 certificate specified via the '-c' option.

```
[Node_1]

ike remote -n dexter -z 2 -w -u -a HMAC-SHA-2-512 -e AES192-CBC -p HMAC-SHA-2-2
```

The set of X.509 certificates to be sent to the remote end is specified via the '`-c`' option to the 'ike remote' CLI command. In the example command above, `Node_1` specifies the two certificates (`supamcert.pem` and `cacert.pem`) to be sent to `Node_2`. Certificates that are available at the peer end do not have to be configured for transmission via the 'ike remote' CLI command. The primary X.509 certificate that contains the identification for the peer must be configured via the '`-b`' option.

The certificates received from the peer are considered untrusted, and are validated locally. Certificates in the local certificate database are considered trusted; however, only those certificates that are capable of signing other certificates are used in the validation process. The X.509 certificate database supports commands for the following operations:

- adding a certificate to the database (via the '`-a`' option)
- deleting a certificate from the database (via the '`-x`' option)
- printing the Subject, Issuer, start time, and end time for a certificate (via the '`-p`' option)
- listing all of the certificates in the database (via the '`-l`' option)
- verifying a set of trusted and untrusted certificates (via the '`-v`' option)

The following section is pertinent only to users of InterNiche's IKE.

The following command lines were used at two InterNiche IKE implementations to establish a secure connection between them that was authenticated via X.509 certificates.

```
[Node_1]

ce x509 -a cacert.pem -d 1
ce x509 -a supamcert.pem -d 1 -k supamkeyc.pem
ike remote -n dexter -z 2 -w -u -a HMAC-SHA-2-512 -e AES192-CBC -p HMAC-SHA-2-2

ipsec policy -n policy0 -g dexter -t ESP -e 3DES-CBC -a HMAC-SHA-2-512 -m tunne

[Node_2]

ce x509 -a cacert.pem -d 1
ce x509 -a supascert.pem -d 1 -k supaskeyc.pem
ike remote -n dexter -z 2 -w -u -a HMAC-SHA-2-512 -e AES192-CBC -p HMAC-SHA-2-2

ipsec policy -n policy0 -g dexter -t ESP -e 3DES-CBC -a HMAC-SHA-2-512 -m tunne
```

6 CE Command man page

ce x509 - add/delete/dump/verify X.509 certificates

Command Name

```
ce x509 - add/delete/dump/verify X.509 certificates
```

Syntax

```
ce x509 {-a <X.509 certificate file> -d <domain identifier> [-k <private
key file> [-h <passphrase>]]} | {-l -d <domain identifier>} | {-p <X.509
certificate file>} | {-v -t <trusted certificate file(s)> -u <untrusted
certificate file(s)>} | {-x <X.509 certificate file> -d <domain identifier>}
```

Parameters

`-a`	add X.509 certificate file to domain in database
`-d`	numeric identifier for domain (e.g., CE_X509CERT_DOMAIN_IKE = 1)
`-h`	passphrase for private key file (only required if file is encrypted)
`-k`	private key file
`-l`	list X.509 certificates present in domain
`-p`	print contents of one X.509 certificate
`-t`	comma-separated list of trusted X.509 certificate file(s)
`-u`	comma-separated list of untrusted X.509 certificate file(s)
`-v`	verify a chain of X.509 certificates
`-x`	delete an X.509 certificate file from domain in database

Description

This command is used to add, delete, dump, and verify X.509 certificates.

Notes/Status

- Here are examples that illustrate the various uses of this CLI command.
    1. The following example adds a Certificate Authority's X.509 certificate to the first domain in the certificate database.

       ```
       ce x509 -a cacert.pem -d 1
       ```

    2. The following example adds an end-user entity's X.509 certificate to the first domain in the certificate database. The corresponding private key file (supamkeyc.pem) is not encrypted.

       ```
       ce x509 -a supamcert.pem -d 1 -k supamkeyc.pem
       ```

    3. The following example adds an end-user entity's X.509 certificate to the first domain in the certificate database. The corresponding private key file (supamkey.pem) is encrypted.

       ```
       ce x509 -a supamcert.pem -d 1 -k supamkey.pem -h secret
       ```

    4. The following command dumps all of the X.509 certificates present in the first domain in the certificate database.

       ```
       ce x509 -l -d 1
       ```

    5. The following command dumps the contents of cacert.pem.

       ```
       ce x509 -p cacert.pem
       ```

    6. The following command verifies a chain of X.509 certificates that contains one trusted certificate (cacert.pem) and one untrusted certificate (supamcert.pem).

       ```
       ce x509 -v -t cacert.pem -u supamcert.pem
       ```

    7. The following example deletes cacert.pem from the first domain in the certificate database.

       ```
       ce x509 -x cacert.pem -d 1
       ```

Location

This command is provided by the `CryptoEngine` module when `USE_CRYPTOENG` and `CE_MENUS` are defined.

7 CE API man pages

7.1 ce_auth

API Name

`ce_auth()`

Syntax

```
int ce_auth(CE_OP_PTR op, u_char *in[], int inlen[], int count, u_char
*out, int outlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Array of input buffers
inlen	Array of input buffer lengths
count	Number of elements in array
out	Output buffer
outlen	Length of output buffer

Description

This function computes a digest of the specified input buffer(s) using the digest algorithm specified in the cipher identifier.

Returns

This function returns ESUCCESS if the digest is computed successfully; otherwise, it returns EFAILURE.

7.2 ce_auth_final

API Name

`ce_auth_final()`

Syntax

```
int ce_auth_final(CE_OP_PTR op, void *ctx, u_char *out, int outlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
ctx	Pointer to plain digest authentication context structure
out	Pointer to output buffer
outlen	Length of output buffer (bytes)

Description

This function finalizes the plain digest computation process, and copies the computed digest into the buffer provided by the caller.

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise it returns EFAILURE.

7.3 ce_auth_init

API Name

`ce_auth_init()`

Syntax

```
void *ce_auth_init(CE_OP_PTR op)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function creates a plain digest authentication context for the digest algorithm specified in the cipher identifier. The digest context is required for all subsequent (incremental) digest computations.

Returns

This function returns a pointer to the digest context, or NULL in the event of an error.

7.4 ce_auth_update

API Name

`ce_auth_update()`

Syntax

```
int ce_auth_update(CE_OP_PTR op, void *ctx, u_char *in, int inlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
ctx	Pointer to plain digest authentication context structure
in	Pointer to input buffer containing data to be digested
inlen	Length of input buffer (bytes)

Description

This function feeds the input data provided by the caller into the digest computation algorithm. The intermediate results are stored in the digest context.

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise it returns EFAILURE.

7.5 ce_b64_dec

API Name

`ce_b64_dec()`

Syntax

```
int ce_b64_dec(CE_OP_PTR op, uint8_t *in, int inlen, uint8_t *out, int
*outlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Pointer to input buffer containing base64-encoded data
inlen	Length of input buffer
out	Pointer to output buffer
outlen	Pointer to integer containing length of output buffer (input), and pointer to integer to store length of decoded output (output)

Description

This function performs base-64 decoding on the data in the input buffer, and stores the decoded data into the caller-provided output buffer. The length of the decoded data is copied into '*outlen'.

Returns

This function returns ESUCCESS if the requested operation was completed successfully; otherwise, it returns EFAILURE.

7.6 ce_b64_enc

API Name

`ce_b64_enc()`

Syntax

```
int ce_b64_enc(CE_OP_PTR op, uint8_t *in, int inlen, uint8_t *out, int outlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Pointer to input buffer
inlen	Length of input buffer
out	Pointer to output buffer
outlen	Length of output buffer

Description

This function performs base-64 encoding on the data in the input buffer, and stores the encoded data into the caller-provided output buffer.

Returns

This function returns ESUCCESS if the requested operation was completed successfully; otherwise, it returns EFAILURE.

7.7 ce_bn_bin2bn

API Name

`ce_bn_bin2bn()`

Syntax

```
void *ce_bn_bin2bn(CE_OP_PTR op, u_char *in, int inlen, void *bn);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Pointer to array of bytes comprising the input bignum (in big-endian order)
inlen	Length of the array of bytes comprising the input bignum
bn	Pointer to output bignum structure that will be filled with the big number stored at 'in'

Description

This function "fills" the bignum structure at 'bn' with the value of the big number at 'in'.

Returns

Returns a pointer to the filled-in bignum structure ('bn'), or NULL in the event of an error.

7.8 ce_bn_bn2bin

API Name

`ce_bn_bn2bin()`

Syntax

```
int ce_bn_bn2bin(CE_OP_PTR op, void *bn, u_char *out);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
bn	Pointer to input bignum structure
out	Pointer to array of bytes where the bignum will be stored (in big-endian order)

Description

This function copies out the bignum stored at 'bn' into 'out'. The caller must ensure that 'out' points to storage of adequate size.

Returns

Returns ESUCCESS if the bignum was written out successfully, and EFAILURE in the event of an error.

7.9 ce_bn_dup

API Name

`ce_bn_dup()`

Syntax

```
void *ce_bn_dup(CE_OP_PTR op, void *bn);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
bn	Pointer to structure containing bignum that is to be copied

Description

This function creates a copy of the bignum stored at 'bn'.

Returns

Returns a pointer to the duplicate bignum structure, or NULL in the event that a duplicate could not be created.

7.10 ce_bn_free

API Name

`ce_bn_free()`

Syntax

```
void ce_bn_free(CE_OP_PTR op, void *bn);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
bn	Pointer to structure containing bignum

Description

This function frees the bignum structure located at 'bn'.

Returns

None.

7.11 ce_bn_hex2bn

API Name

`ce_bn_hex2bn()`

Syntax

```
void *ce_bn_hex2bn(CE_OP_PTR op, void *bn, char *in);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
bn	Pointer to pointer to output bignum structure that will be filled with the big number stored at 'in'
in	Pointer to (NULL-terminated) hexadecimal string containing the input bignum (in big-endian order)

Description

This function "fills" the bignum structure at '*bn' with the value of the big number at 'in'. If '*bn' is NULL, a new bignum structure is allocated prior to use.

Returns

This function returns ESUCCESS if the input data was copied into the bignum structure, and EFAILURE in the event of an error.

7.12 ce_bn_new

API Name

`ce_bn_new()`

Syntax

```
void *ce_bn_new(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function allocates a new bignum structure.

Returns

Pointer to bignum structure, or NULL if the structure could not be allocated.

7.13 ce_bn_num_bytes

API Name

`ce_bn_num_bytes()`

Syntax

```
int ce_bn_num_bytes(CE_OP_PTR op, void *bn);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
bn	Pointer to structure containing bignum

Description

This function returns the size (in bytes) of the bignum stored at 'bn'.

Returns

Size of big number stored at 'bn'.

7.14 ce_dh_free

API Name

`ce_dh_free()`

Syntax

```
void ce_dh_free(CE_OP_PTR op, void *dh);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure

Description

This function frees a DH structure.

Returns

None.

7.15 ce_dh_gen_local_pubpriv

API Name

`ce_dh_gen_local_pubpriv()`

Syntax

```
int ce_dh_gen_local_pubpriv(CE_OP_PTR op, void *dh, u_char *lcl_pub, int
*pub_len, u_char *lcl_prv, int *prv_len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure
lcl_pub	Output buffer to store public DH value
pub_len	Length of 'lcl_pub' buffer
lcl_prv	Output buffer to store private DH value
prv_len	Length of 'lcl_prv' buffer

Description

This function generates the public ('g^x_mod(p)') and private ('x') DH values.

Returns

This function returns ESUCCESS if the generation of public and private DH values is successful; otherwise, it returns EFAILURE. If an output buffer is specified as NULL, or if the buffer is insufficiently long, this function will write the length of the corresponding parameter to '*pub_len' (for the public DH value) or '*prv_len' (for the private DH value).

7.16 ce_dh_gen_shared_secret

API Name

`ce_dh_gen_shared_secret()`

Syntax

```
int ce_dh_gen_shared_secret(CE_OP_PTR op, void *dh, u_char *secret, int
*secret_len, const u_char *rem_pub, int rem_len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure
secret	Output buffer (for storing DH shared secret)
secret_len	Length of output buffer (input) and required length (output)
rem_pub	Public DH value of remote end (peer)
rem_len	Length of peer's public DH value

Description

This function computes the DH shared secret, and copies it into the specified output buffer. The length of the computed secret is copied into '*secret_len'.

Returns

This function returns ESUCCESS if the shared secret computation is successful; otherwise, it returns EFAILURE.

7.17 ce_dh_gen_shared_secret0

API Name

`ce_dh_gen_shared_secret0()`

Syntax

```
int ce_dh_gen_shared_secret0(CE_OP_PTR op, void *dh, u_char *secret, void
*rem_pub);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure
secret	Output buffer (for storing DH shared secret)
rem_pub	Pointer to big number containing public DH value of remote end (peer)

Description

This function computes the DH shared secret, and copies it into the specified output buffer.

Returns

This function returns ESUCCESS if the shared secret computation is successful; otherwise, it returns EFAILURE.

7.18 ce_dh_gen_shared_secret2

API Name

`ce_dh_gen_shared_secret2()`

Syntax

```
int ce_dh_gen_shared_secret2(CE_OP_PTR op, u_char *rem_pub, int rem_len,
u_char *lcl_priv, int lcl_len, u_char *secret, int *secret_len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rem_pub	Pointer to input buffer containing public DH value of remote end (peer)
rem_len	Length of peer's public DH value
lcl_priv	Pointer to input buffer containing private DH value of local end
lcl_len	Length of local end's private DH value
secret	Output buffer (for storing DH shared secret)
secret_len	Pointer to integer containing length of output buffer (input) and required length (output)

Description

This function computes the DH shared secret, and copies it into the specified output buffer. The length of the computed secret is copied into '*secret_len'.

Returns

This function returns ESUCCESS if the shared secret computation is successful; otherwise, it returns EFAILURE.

7.19 ce_dh_get_pub

API Name

`ce_dh_get_pub()`

Syntax

```
void *ce_dh_get_pub(CE_OP_PTR op, void *dh);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to Diffie-Hellman structure

Description

This function returns a pointer to the big number that comprises the public value for the specified Diffie-Hellman key structure.

Returns

Pointer to the big number that comprises the public value for the specified Diffie-Hellman key structure.

7.20 ce_dh_new

API Name

`ce_dh_new()`

Syntax

```
void *ce_dh_new(CE_OP_PTR op, bool_t init, int len, int generator);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
init	Boolean that determines whether an "empty" (0) or filled-in (1) Diffie-Hellman (DH) structure is returned
len	Length in bits of safe prime ('p') to be generated
generator	Generator ('g', 2 or 5)

Description

This function allocates a new (empty or initialized) Diffie-Hellman (DH) structure.

Returns

Pointer to DH structure, or NULL in the event of an error.

7.21 ce_dh_setparms_pg

API Name

`ce_dh_setparms_pg()`

Syntax

```
int ce_dh_setparms_pg(CE_OP_PTR op, void *dh, const u_char *p, int plen,
const u_char *g, int glen);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure
p	Pointer to input buffer containing prime ('p' in g^x_mod(p))
plen	Length of prime (in bytes)
g	Pointer to input buffer containing generator ('g' in g^x_mod(p))
glen	Length of generator (in bytes)

Description

This function configures the prime 'p' and generator 'g' in the Diffie-Hellman structure. The 'p' ang 'g' parameters must point to an array of bytes in network byte order that comprise the prime and generator respectively.

Returns

This function returns ESUCCESS if the Diffie-Hellman structure was successfully configured; otherwise, it returns EFAILURE.

7.22 ce_dh_setparms_pg2

API Name

`ce_dh_setparms_pg2()`

Syntax

```
int ce_dh_setparms_pg2(CE_OP_PTR op, void *dh, void *p, void *g);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure
p	Pointer to big number containing prime ('p' in g^x_mod(p))
g	Pointer to big number containing generator ('g' in g^x_mod(p))

Description

This function configures the prime 'p' and generator 'g' in the Diffie-Hellman structure.

Returns

This function returns ESUCCESS if the Diffie-Hellman structure was successfully configured; otherwise, it returns EFAILURE.

7.23 ce_dh_size

API Name

`ce_dh_size()`

Syntax

```
int ce_dh_size(CE_OP_PTR op, void *dh);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dh	Pointer to Diffie-Hellman (DH) structure

Description

This function returns the size (in bytes) of the Diffie-Hellman prime ('p').

Returns

Size (in bytes) of the Diffie-Hellman prime ('p').

7.24 ce_dsa_d2i_prvkey

API Name

`ce_dsa_d2i_prvkey()`

Syntax

```
void *ce_dsa_d2i_prvkey(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Pointer to input buffer containing DER-encoded DSA private key
len	Length of input buffer

Description

This function returns a pointer to a DSA key structure that is created from the DER-encoded DSA private key.

Returns

Pointer to DSA key structure, or NULL in the event of an error.

7.25 ce_dsa_d2i_pubkey

API Name

`ce_dsa_d2i_pubkey()`

Syntax

```
void *ce_dsa_d2i_pubkey(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Pointer to input buffer containing DER-encoded DSA public key
len	Length of input buffer

Description

This function returns a pointer to a DSA key structure that is created from the DER-encoded DSA public key.

Returns

Pointer to DSA key structure, or NULL in the event of an error.

7.26 ce_dsa_d2i_pubkey2

API Name

`ce_dsa_d2i_pubkey2()`

Syntax

```
void *ce_dsa_d2i_pubkey2(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Pointer to input buffer containing DER-encoded DSA public key (in the SubjectPublicKeyInfo format)
len	Length of input buffer

Description

This function returns a pointer to a DSA key structure that is created from the DER-encoded DSA public key (in the SubjectPublicKeyInfo format). This format is described in RFC 2459 ("Internet X.509 Public Key Infrastructure").

Returns

Pointer to DSA key structure, or NULL in the event of an error.

7.27 ce_dsa_d2i_sig

API Name

`ce_dsa_d2i_sig()`

Syntax

```
void *ce_dsa_d2i_sig(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Pointer to input buffer containing DER-encoded DSA signature
len	Length of input buffer

Description

This function returns a pointer to a DSA signature structure that is created from the DER-encoded DSA signature.

Returns

Pointer to DSA signature structure, or NULL in the event of an error.

7.28 ce_dsa_free

API Name

`ce_dsa_free()`

Syntax

```
void ce_dsa_free(CE_OP_PTR op, void *dsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA structure

Description

This function frees a DSA structure.

Returns

None.

7.29 ce_dsa_get_g

API Name

`ce_dsa_get_g()`

Syntax

```
void *ce_dsa_get_g(CE_OP_PTR op, void *dsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA key structure

Description

This function returns a pointer to the big number that comprises 'g' for the specified DSA key structure. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

Pointer to the big number that comprises 'g' for the specified DSA key structure.

7.30 ce_dsa_get_p

API Name

`ce_dsa_get_p()`

Syntax

```
void *ce_dsa_get_p(CE_OP_PTR op, void *dsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA key structure

Description

This function returns a pointer to the big number that comprises 'p' for the specified DSA key structure. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

Pointer to the big number that comprises 'p' for the specified DSA key structure.

7.31 ce_dsa_get_pub_key

API Name

`ce_dsa_get_pub_key()`

Syntax

```
void *ce_dsa_get_pub_key(CE_OP_PTR op, void *dsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA key structure

Description

This function returns a pointer to the big number that comprises 'y' (public key) for the specified DSA key structure. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

Pointer to the big number that comprises 'y' (public key) for the specified DSA key structure.

7.32 ce_dsa_get_q

API Name

`ce_dsa_get_q()`

Syntax

```
void *ce_dsa_get_q(CE_OP_PTR op, void *dsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA key structure

Description

This function returns a pointer to the big number that comprises 'q' for the specified DSA key structure. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

Pointer to the big number that comprises 'q' for the specified DSA key structure.

7.33 ce_dsa_i2d_sig

API Name

`ce_dsa_i2d_sig()`

Syntax

```
int ce_dsa_i2d_sig(CE_OP_PTR op, void *sig, u_char *buf, int *len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA signature structure
buf	Pointer to output buffer
len	Pointer to integer containing length of output buffer (input), and actual number of bytes copied into output buffer (output)

Description

This function writes out a DSA signature into the specified output buffer. If the 'buf' argument is NULL or if the buffer is insufficiently long, this function will write the amount of space required into '*len'.

Returns

This function returns ESUCCESS if the requested operation was successfully completed; otherwise, it returns EFAILURE.

7.34 ce_dsa_new

API Name

`ce_dsa_new()`

Syntax

```
void *ce_dsa_new(CE_OP_PTR op, int group_size, int modulus_size);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
group_size	Group size
modulus_size	Modulus size

Description

This function allocates a new DSA structure. The capabilities provided by this function vary by cryptography provider.

Returns

Pointer to DSA structure, or NULL in the event of a failure.

7.35 ce_dsa_new0

API Name

`ce_dsa_new0()`

Syntax

```
void *ce_dsa_new0(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function allocates an "empty" (uninitialized) DSA key structure.

Returns

Pointer to DSA key structure, or NULL in the event of an error.

7.36 ce_dsa_parm_size

API Name

`ce_dsa_parm_size()`

Syntax

```
int ce_dsa_parm_size(CE_OP_PTR op, void *dsa, DSA_PARM parm_id);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA structure
parm_id	Identifier for DSA parameter

Description

This function returns the size (in bytes) of the specified parameter (CE_DSA_PARM_G, CE_DSA_PARM_P, CE_DSA_PARM_Q, CE_DSA_PARM_X (private key), CE_DSA_PARM_Y (public key)) for the specified key.

Returns

Size (in bytes) of requested parameter.

7.37 ce_dsa_rd_prv_key_file

API Name

`ce_dsa_rd_prv_key_file()`

Syntax

```
void *ce_dsa_rd_prv_key_file(CE_OP_PTR op, char *file_name, int file_type,
void *passphrase);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
file_name	Name of file containing DSA private key
file_type	Type of file (unused)
passphrase	Passphrase to decrypt file (required only if private key is encrypted)

Description

This function creates a DSA private key from the contents of the specified file. The private key file must be encoded in the PEM format. If the private key is stored in the clear, the 'passphrase' parameter must be specified as NULL.

Returns

Pointer to DSA key structure.

7.38 ce_dsa_set_pqgy

API Name

`ce_dsa_set_pqgy()`

Syntax

```
int ce_dsa_set_pqgy(CE_OP_PTR op, void *dsa, void *p, void *q, void *g,
void *pub_key);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA key structure
p	Pointer to big number containing 'p'
q	Pointer to big number containing 'q'
g	Pointer to big number containing 'g'
pub_key	Pointer to big number containing 'y' (public key)

Description

This function sets the 'p', 'q', 'g', and 'y' (public key) parameters for the specified DSA key structure to the specified values. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise, it returns EFAILURE.

7.39 ce_dsa_sig_free

API Name

`ce_dsa_sig_free()`

Syntax

```
void ce_dsa_sig_free(CE_OP_PTR op, void *sig);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
sig	Pointer to DSA signature structure

Description

This function frees a DSA signature structure.

Returns

None.

7.40 ce_dsa_sig_get_r

API Name

`ce_dsa_sig_get_r()`

Syntax

```
void *ce_dsa_sig_get_r(CE_OP_PTR op, void *sig);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA signature structure

Description

This function returns a pointer to the big number that comprises 'r' for the specified DSA signature structure. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

Pointer to the big number that comprises 'r' for the specified DSA signature structure.

7.41 ce_dsa_sig_get_s

API Name

`ce_dsa_sig_get_s()`

Syntax

```
void *ce_dsa_sig_get_s(CE_OP_PTR op, void *sig);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA signature structure

Description

This function returns a pointer to the big number that comprises 's' for the specified DSA signature structure. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

Pointer to the big number that comprises 's' for the specified DSA signature structure.

7.42 ce_dsa_sig_new

API Name

`ce_dsa_sig_new()`

Syntax

```
void *ce_dsa_sig_new(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function allocates an "empty" (uninitialized) DSA signature structure.

Returns

Pointer to DSA signature structure, or NULL in the event of an error.

7.43 ce_dsa_sig_set_rs

API Name

`ce_dsa_sig_set_rs()`

Syntax

```
int ce_dsa_sig_set_rs(CE_OP_PTR op, void *sig, void *r, void *s);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
sig	Pointer to DSA signature structure
r	Pointer to big number containing 'r'
s	Pointer to big number containing 's'

Description

This function sets the 'r' and 's' parameters for the specified DSA signature structure to the specified values. DSA parameters are described in FIPS PUB 186-4 ("Digital Signature Standard").

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise, it returns EFAILURE.

7.44 ce_dsa_sign

API Name

`ce_dsa_sign()`

Syntax

```
int ce_dsa_sign(CE_OP_PTR op, int alg, const u_char *in, int inlen, u_char
*out, unsigned int *outlen, void *dsa)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
alg	(unused)
in	Input buffer
inlen	Length of input buffer
out	Output buffer
outlen	Pointer to integer containing length of input buffer (input), and pointer to integer to store length of signature (output)
dsa	Pointer to DSA key structure

Description

This function computes the DSA signature on an input buffer, and copies it into the caller-provided output buffer. The length of the signature is copied into '*outlen'.

Returns

This function returns ESUCCESS if the signature was computed successfully; otherwise, it returns EFAILURE.

7.45 ce_dsa_size

API Name

`ce_dsa_size()`

Syntax

```
int ce_dsa_size(CE_OP_PTR op, void *dsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
dsa	Pointer to DSA structure

Description

This function returns the length of an ASN.1-encoded DSA signature (in bytes).

Returns

Modulus size

7.46 ce_dsa_vrfy

API Name

`ce_dsa_vrfy()`

Syntax

```
int ce_dsa_vrfy(CE_OP_PTR op, int alg, const u_char *in, int inlen, u_char
*sig, int siglen, void *dsa)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
alg	(unused)
in	Pointer to input
inlen	Length of input
sig	Pointer to signature
siglen	Length of signature
dsa	Pointer to DSA key structure

Description

This function validates the DSA signature on an input buffer.

Returns

This function returns ESUCCESS if the signature was verified successfully; otherwise, it returns EFAILURE.

7.47 ce_hmac_auth

API Name

`ce_hmac_auth()`

Syntax

```
int ce_hmac_auth(CE_OP_PTR op, u_char *in[], int inlen[], int count, u_char
*key, int keylen, u_char *out, int outlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Array of input buffers
inlen	Array of input buffer lengths
count	Number of elements in array
key	Key buffer
keylen	Length of key buffer
out	Output buffer
outlen	Length of output buffer

Description

This function computes the HMAC digest of the specified input buffer(s) using the digest algorithm specified in the cipher identifier.

Returns

This function returns ESUCCESS if the digest is computed successfully; otherwise, it returns EFAILURE.

7.48 ce_hmac_auth_final

API Name

`ce_hmac_auth_final()`

Syntax

```
int ce_hmac_auth_final(CE_OP_PTR op, void *ctx, u_char *out, int outlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
ctx	Pointer to HMAC digest authentication context structure
out	Pointer to output buffer
outlen	Length of output buffer (bytes)

Description

This function finalizes the HMAC digest computation process, and copies the computed digest into the buffer provided by the caller.

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise it returns EFAILURE.

7.49 ce_hmac_auth_init

API Name

`ce_hmac_auth_init()`

Syntax

```
void *ce_hmac_auth_init(CE_OP_PTR op, u_char *key, int keylen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
key	Pointer to input buffer containing key
keylen	Length of key (bytes)

Description

This function creates a HMAC digest authentication context for the digest algorithm specified in the cipher identifier. The digest context is required for all subsequent (incremental) digest computations.

Returns

This function returns a pointer to the digest context, or NULL in the event of an error.

7.50 ce_hmac_auth_update

API Name

`ce_hmac_auth_update()`

Syntax

```
int ce_hmac_auth_update(CE_OP_PTR op, void *ctx, u_char *in, int inlen)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
ctx	Pointer to HMAC digest authentication context structure
in	Pointer to input buffer containing data to be digested
inlen	Length of input buffer (bytes)

Description

This function feeds the input data provided by the caller into the digest computation algorithm. The intermediate results are stored in the digest context.

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise it returns EFAILURE.

7.51 ce_rand_bytes

API Name

`ce_rand_bytes()`

Syntax

```
int ce_rand_bytes(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Output buffer
len	Length of output buffer

Description

This function generates the requested number of random bytes, and copies them into the caller-provided buffer.

Returns

This function returns ESUCCESS if it can provide the requested data; otherwise, it returns EFAILURE.

7.52 ce_rand_cleanup

API Name

`ce_rand_cleanup()`

Syntax

```
int ce_rand_cleanup(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function cleans up the internal state of the specified random number generator.

Returns

This function returns ESUCCESS if the requested operation was successfully performed; otherwise, it returns EFAILURE.

7.53 ce_rand_init

API Name

`ce_rand_init()`

Syntax

```
int ce_rand_init(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function initializes the specified random number generator.

Returns

This function returns ESUCCESS if the requested operation was completed successfully; otherwise, it returns EFAILURE.

7.54 ce_rand_seed

API Name

`ce_rand_seed()`

Syntax

```
int ce_rand_seed(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Input buffer
len	Length of input buffer

Description

This function seeds the random number generator with the caller-provided random data.

Returns

This function always returns ESUCCESS.

7.55 ce_rand_status

API Name

`ce_rand_status()`

Syntax

```
int ce_rand_status(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function returns the status of the specified random number generator.

Returns

This function returns ESUCCESS if the specified random number generator is up and running; otherwise, it returns EFAILURE.

7.56 ce_rsa_d2i_prvkey

API Name

`ce_rsa_d2i_prvkey()`

Syntax

```
void *ce_rsa_d2i_prvkey(CE_OP_PTR op, u_char *key, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
key	Pointer to input buffer containing RSA private key
len	Length of input buffer

Description

This function returns a pointer to a RSA key structure that is created from the DER-encoded RSA private key. The latter is stored in the PKCS#1 RSAPrivateKey format (RFC 2437, "PKCS #1: RSA Cryptography Specifications").

Returns

Pointer to RSA key structure.

7.57 ce_rsa_d2i_pubkey

API Name

`ce_rsa_d2i_pubkey()`

Syntax

```
void *ce_rsa_d2i_pubkey(CE_OP_PTR op, u_char *key, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
key	Pointer to input buffer containing RSA public key
len	Length of input buffer

Description

This function returns a pointer to a RSA key structure that is created from the DER-encoded RSA public key. The latter is stored in the PKCS#1 RSAPublicKey format (RFC 2437, "PKCS #1: RSA Cryptography Specifications").

Returns

Pointer to RSA key structure.

7.58 ce_rsa_d2i_pubkey2

API Name

`ce_rsa_d2i_pubkey2()`

Syntax

```
void *ce_rsa_d2i_pubkey2(CE_OP_PTR op, u_char *key, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
key	Pointer to input buffer containing RSA public key
len	Length of input buffer

Description

This function returns a pointer to a RSA key structure that is created from the DER-encoded RSA public key. The latter is stored in the SubjectPublicKeyInfo format (RFC 2459, "Internet X.509 Public Key Infrastructure").

Returns

Pointer to RSA key structure.

7.59 ce_rsa_free

API Name

`ce_rsa_free()`

Syntax

```
void ce_rsa_free(CE_OP_PTR op, void *rsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA key structure

Description

This function frees a RSA key structure.

Returns

None.

7.60 ce_rsa_get_modulus

API Name

`ce_rsa_get_modulus()`

Syntax

```
void *ce_rsa_get_modulus(CE_OP_PTR op, void *rsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA key structure

Description

This function returns a pointer to the big number that comprises the modulus for the specified RSA key structure.

Returns

Pointer to the big number that comprises the modulus for the specified RSA key structure.

7.61 ce_rsa_get_pub_exp

API Name

`ce_rsa_get_pub_exp()`

Syntax

```
void *ce_rsa_get_pub_exp(CE_OP_PTR op, void *rsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA key structure

Description

This function returns a pointer to the big number that comprises the public exponent for the specified RSA key structure.

Returns

Pointer to the big number that comprises the public exponent for the specified RSA key structure.

7.62 ce_rsa_i2d_pubkey2

API Name

`ce_rsa_i2d_pubkey2()`

Syntax

```
int ce_rsa_i2d_pubkey2(CE_OP_PTR op, void *rsa, u_char *buf, int *len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA key structure
buf	Pointer to output buffer
len	Pointer to integer containing length of output buffer (input), and actual number of bytes copied into output buffer (output)

Description

This function writes out a RSA public key into the specified output buffer. The data is written out in the SubjectPublicKeyInfo format described in RFC 2459 ("Internet X.509 Public Key Infrastructure"). If the 'buf' argument is NULL or if the buffer is insufficiently long, this function will write the amount of space required into '*len'.

Returns

This function returns ESUCCESS if the requested operation was successfully completed; otherwise, it returns EFAILURE.

7.63 ce_rsa_new

API Name

`ce_rsa_new()`

Syntax

```
void *ce_rsa_new(CE_OP_PTR op, int size, int e);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
size	Length of modulus (bits)
e	Public exponent

Description

This function allocates a new RSA key structure.

Returns

Pointer to RSA key structure, or NULL in the event of an error.

7.64 ce_rsa_new0

API Name

`ce_rsa_new0()`

Syntax

```
void *ce_rsa_new0(CE_OP_PTR op);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)

Description

This function allocates an "empty" (uninitialized) RSA key structure.

Returns

Pointer to RSA key structure, or NULL in the event of an error.

7.65 ce_rsa_parm_size

API Name

`ce_rsa_parm_size()`

Syntax

```
int ce_rsa_parm_size(CE_OP_PTR op, void *rsa, RSA_PARM parm_id);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA structure
parm_id	Identifier for RSA parameter

Description

This function returns the size (in bytes) of the specified parameter (CE_RSA_PARM_D (private exponent), CE_RSA_PARM_E (public exponent), or CE_RSA_PARM_N (modulus)) for the specified key.

Returns

Size (in bytes) of requested parameter.

7.66 ce_rsa_prv_decr

API Name

`ce_rsa_prv_decr()`

Syntax

```
int ce_rsa_prv_decr(CE_OP_PTR op, u_char *in, int inlen, u_char *out, int
outlen, void *rsa, int padding);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Input buffer
inlen	Length of input buffer
out	Output buffer
outlen	Length of output buffer
rsa	Pointer to RSA key data structure.
padding	Type of padding (CE_RSA_PAD_NONE, CE_RSA_PAD_PKCS1, CE_RSA_PAD_PKCS1_OAEP, or CE_RSA_PAD_SSLV23)

Description

This function decrypts the input buffer using a RSA private key, and stores the output in the caller-provided buffer.

Returns

This function returns ESUCCESS if the decryption was successful; otherwise, it returns EFAILURE.

7.67 ce_rsa_prv_encr

API Name

`ce_rsa_prv_encr()`

Syntax

```
int ce_rsa_prv_encr(u_char *in, int inlen, u_char *out, int outlen, void
*rsa, int padding);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Input buffer
inlen	Length of input buffer
out	Output buffer
outlen	Length of output buffer
rsa	Pointer to RSA key data structure.
padding	Type of padding (CE_RSA_PAD_NONE, CE_RSA_PAD_PKCS1, CE_RSA_PAD_PKCS1_OAEP, or CE_RSA_PAD_SSLV23)

Description

This function encrypts the input buffer using a RSA private key, and stores the output in the caller-provided buffer.

Returns

This function returns ESUCCESS if the encryption was successful; otherwise, it returns EFAILURE.

7.68 ce_rsa_pub_decr

API Name

`ce_rsa_pub_decr()`

Syntax

```
int ce_rsa_pub_decr(u_char *in, int inlen, u_char *out, int outlen, void
*rsa, int padding);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Input buffer
inlen	Length of input buffer
out	Output buffer
outlen	Length of output buffer
rsa	Pointer to RSA key data structure.
padding	Type of padding (CE_RSA_PAD_NONE, CE_RSA_PAD_PKCS1, CE_RSA_PAD_PKCS1_OAEP, or CE_RSA_PAD_SSLV23)

Description

This function decrypts the input buffer using a RSA public key, and stores the output in the caller-provided buffer.

Returns

This function returns ESUCCESS if the decryption was successful; otherwise, it returns EFAILURE.

7.69 ce_rsa_pub_encr

API Name

`ce_rsa_pub_encr()`

Syntax

```
int ce_rsa_pub_encr(u_char *in, int inlen, u_char *out, int outlen, void
*rsa, int padding);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Input buffer
inlen	Length of input buffer
out	Output buffer
outlen	Length of output buffer
rsa	Pointer to RSA key data structure
padding	Type of padding (CE_RSA_PAD_NONE, CE_RSA_PAD_PKCS1, CE_RSA_PAD_PKCS1_OAEP, or CE_RSA_PAD_SSLV23)

Description

This function encrypts the input buffer using a RSA public key, and stores the output in the caller-provided buffer.

Returns

This function returns ESUCCESS if the encryption was successful; otherwise, it returns EFAILURE.

7.70 ce_rsa_rd_prv_key_file

API Name

`ce_rsa_rd_prv_key_file()`

Syntax

```
void *ce_rsa_rd_prv_key_file(CE_OP_PTR op, char *file_name, int file_type,
void *passphrase);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
file_name	Name of file containing RSA private key
file_type	Type of file (unused)
passphrase	Passphrase required to decrypt file (only if private key is encrypted)

Description

This function creates a RSA private key from the contents of the specified file. The private key in the file must be in the PKCS#1 RSAPrivateKey format (RFC 2437, "PKCS #1: RSA Cryptography Specifications"), and the file must be encoded in the PEM format. If the private key is stored in the clear, the 'passphrase' parameter must be specified as NULL.

Returns

Pointer to RSA key structure.

7.71 ce_rsa_set_modulus_and_pub_exp

API Name

`ce_rsa_set_modulus_and_pub_exp()`

Syntax

```
int ce_rsa_set_modulus_and_pub_exp(CE_OP_PTR op, void *rsa, void *modulus,
void *pub_exp);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA key structure
modulus	Pointer to big number containing modulus
pub_exp	Pointer to big number containing public exponent

Description

This function sets the modulus and public exponent for the specified RSA key structure to the specified values.

Returns

This function returns ESUCCESS if the requested operation was successful; otherwise, it returns EFAILURE.

7.72 ce_rsa_sign

API Name

`ce_rsa_sign()`

Syntax

```
int ce_rsa_sign(int alg, u_char *in, unsigned int inlen, u_char *out,
unsigned int *outlen, void *rsa)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
alg	Algorithm used to generate message digest (stored in the input buffer)
in	Input buffer
inlen	Length of input buffer
out	Output buffer
outlen	(output) Pointer to location to store length of signature
rsa	Pointer to RSA key structure

Description

This function computes the RSA signature on an input buffer, and copies it into the caller-provided output buffer. The length of the signature is copied into '*outlen'.

Returns

This function returns ESUCCESS if the signature was computed successfully; otherwise, it returns EFAILURE.

7.73 ce_rsa_size

API Name

`ce_rsa_size()`

Syntax

```
int ce_rsa_size(CE_OP_PTR op, void *rsa);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
rsa	Pointer to RSA structure

Description

This function returns the modulus size in bytes.

Returns

Modulus size

7.74 ce_rsa_vrfy

API Name

`ce_rsa_vrfy()`

Syntax

```
int ce_rsa_vrfy(int alg, u_char *in, unsigned int inlen, u_char *sig,
unsigned int siglen, void *rsa)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
alg	Algorithm used to generate message digest
in	Pointer to input
inlen	Length of input
sig	Pointer to signature
siglen	Length of signature
rsa	Pointer to RSA key structure

Description

This function validates the RSA signature on an input buffer.

Returns

This function returns ESUCCESS if the signature was verified successfully; otherwise, it returns EFAILURE.

7.75 ce_sym_priv

API Name

`ce_sym_priv()`

Syntax

```
int ce_sym_priv(CE_OP_PTR op, u_char *in, int inlen, u_char *iv, int ivlen,
u_char *key, int keylen, u_char *out, int outlen, void *vptr,
CE_PRIV_BLK_DESC mode)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
in	Input buffer
inlen	Length of input buffer
iv	Initialization Vector (IV) buffer
ivlen	Length of IV
key	Key buffer
keylen	Length of key buffer
out	Output buffer
outlen	Length of output buffer
vptr	Pointer to structure used in CFB, CTR, and OFB modes of operation (used for both input and output)
mode	Specifies mode of requested operation (encryption or decryption, update IV or don't update IV (CBC mode), and number of bits fed back (CFB and OFB modes))

Description

This function encrypts (or decrypts) the input buffer using the privacy algorithm specified in the cipher identifier.

Returns

This function returns ESUCCESS if the encryption (or decryption) is performed successfully; otherwise, it returns EFAILURE.

7.76 ce_utils_des_set_odd_parity

API Name

`ce_utils_des_set_odd_parity()`

Syntax

```
void ce_utils_des_set_odd_parity(DES_CBLOCK *key);
```

Parameters

key	Pointer to buffer containing DES key

Description

This function updates the DES key provided by the caller to ensure that it has odd parity.

Returns

None.

7.77 ce_utils_get_auth_params

API Name

`ce_utils_get_auth_params()`

Syntax

```
int ce_utils_get_auth_params(const char *name, int *len);
```

Parameters

name	Name of digest algorithm
len	Pointer to integer containing length of digest algorithm (output)

Description

This function returns the CryptoEngine numeric identifier (e.g., CE_ALG_MD5) associated with the specified digest algorithm (e.g., "MD5"). It also copies the digest length into '*len'.

Returns

This function returns the CryptoEngine numeric identifier (e.g., CE_ALG_MD5) associated with the specified digest algorithm (e.g., "MD5"). If the digest algorithm is not supported, it returns EFAILURE.

7.78 ce_x509cert_db_add

API Name

`ce_x509cert_db_add()`

Syntax

```
int ce_x509cert_db_add(CE_OP_PTR op, int domain, char *cert, char *prvkey,
char *passphrase)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier
cert	Name of file containing X.509 certificate
prvkey	Name of file containing private key (corresponding to the public key stored in the certificate)
passphrase	Passphrase to decrypt (encrypted) private key file (NULL, if the private key file is in the clear)

Description

This function adds a X.509 certificate to the specified domain. It can also be used to update an existing entry.

Returns

This function returns ESUCCESS if the certificate was successfully added; otherwise, it returns EFAILURE.

7.79 ce_x509cert_db_create

API Name

`ce_x509cert_db_create()`

Syntax

```
int ce_x509cert_db_create(CE_OP_PTR op, int domain)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier

Description

This function creates a new application-specific domain for the storage and processing of X.509 certificates. All certificates stored in a domain are considered trusted.

Returns

This function returns ESUCCESS if the domain was successfully created; otherwise, it returns EFAILURE.

7.80 ce_x509cert_db_del

API Name

`ce_x509cert_db_del()`

Syntax

```
int ce_x509cert_db_del(CE_OP_PTR op, int domain, char *cert)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier
cert	Name of file containing X.509 certificate

Description

This function deletes a X.509 certificate from the specified domain.

Returns

This function returns ESUCCESS if the certificate was successfully deleted; otherwise, it returns EFAILURE.

7.81 ce_x509cert_db_destroy

API Name

`ce_x509cert_db_destroy()`

Syntax

```
int ce_x509cert_db_destroy(CE_OP_PTR op, int domain)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier

Description

This function deletes the specified domain for the storage and processing of X.509 certificates.

Returns

This function returns ESUCCESS if the domain was successfully deleted; otherwise, it returns EFAILURE.

7.82 ce_x509cert_db_find

API Name

`ce_x509cert_db_find()`

Syntax

```
void *ce_x509cert_db_find(CE_OP_PTR op, int domain, char *cert)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier
cert	Name of file containing X.509 certificate

Description

This function is used to locate a X.509 certificate in the specified domain.

Returns

This function returns a pointer to the matching certificate table entry. If a match cannot be located, it returns NULL.

7.83 ce_x509cert_db_list

API Name

`ce_x509cert_db_list()`

Syntax

```
int ce_x509cert_db_list(CE_OP_PTR op, int domain)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier

Description

This function displays the Subject, Issuer, start date, and end date for each of the X.509 certificates in the specified domain.

Returns

This function returns ESUCCESS if the domain identifier is correct; otherwise, it returns EFAILURE.

7.84 ce_x509cert_free

API Name

`ce_x509cert_free()`

Syntax

```
void ce_x509cert_free(CE_OP_PTR op, void *cert)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure

Description

This function frees the specified X.509 certificate data structure.

Returns

None.

7.85 ce_x509cert_get_pubkey

API Name

`ce_x509cert_get_pubkey()`

Syntax

```
void *ce_x509cert_get_pubkey(CE_OP_PTR op, void *cert);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure

Description

This function returns a pointer to the public key structure associated with the Subject in the specified X.509 certificate.

Returns

Pointer to the public key associated with the Subject in the X.509 certificate.

7.86 ce_x509cert_get_subject_altname

API Name

`ce_x509cert_get_subject_altname()`

Syntax

```
int ce_x509cert_get_subject_altname(CE_OP_PTR op, void *cert, int pos, char
*altname, int *length, int *type)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure
pos	Index of Subject Alternative Name (SAN) that is to be retrieved
altname	Pointer to output buffer for storage of retrieved SAN
length	Pointer to integer containing the length of the retrieved SAN (output)
type	Type of SAN that was retrieved (GEN_DNS, etc.)

Description

This function retrieves the n-th Subject Alternative Name specified in the X.509 certificate structure. (The desired index is specified via the 'pos' parameter.)

Returns

This function returns ESUCCESS if the requested parameter was successfully retrieved; otherwise, it returns EFAILURE.

7.87 ce_x509cert_getparm

API Name

`ce_x509cert_getparm()`

Syntax

```
int ce_x509cert_getparm(CE_OP_PTR op, void *cert, int param, u_char *buf,
int *len)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure
param	Parameter being requested (Subject (CE_X509CERT_FIELD_SUBJECT or CE_X509CERT_FIELD_SUBJECT_DER), Issuer (CE_X509CERT_FIELD_ISSUER), start date (CE_X509CERT_FIELD_NOTBEFORE), end date (CE_X509CERT_FIELD_NOTAFTER))
buf	Output buffer
len	Pointer to integer containing the length of output buffer (input), and actual length required (output)

Description

This function writes the contents of the specified parameter in the X.509 certificate into the output buffer in text (or DER, for CE_X509CERT_FIELD_SUBJECT_DER) format. The textual data is not NULL-terminated.

Returns

This function returns ESUCCESS if the buffer was written to successfully; otherwise, it returns EFAILURE. The 'len' parameter is updated with the amount of data written (if the buffer was adequately long), or the required length of the buffer (if it wasn't).

7.88 ce_x509cert_print

API Name

`ce_x509cert_print()`

Syntax

```
int ce_x509cert_print(CE_OP_PTR op, void *cert, u_char *buf, int *len)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure
buf	Output buffer
len	Pointer to the length of buffer (input), and actual length (output)

Description

This function writes the contents of the specified X.509 certificate into the output buffer in text format. The data is not NULL-terminated.

Returns

This function returns ESUCCESS if the buffer was written to successfully; otherwise, it returns EFAILURE. The 'len' parameter is updated with the amount of data written (if the buffer was adequately long), or the required length of the buffer (if it wasn't).

7.89 ce_x509cert_rd_buf

API Name

`ce_x509cert_rd_buf()`

Syntax

```
void *ce_x509cert_rd_buf(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Buffer containing X.509 certificate in PEM format
len	Length of buffer

Description

This function reads a buffer containing a X.509 certificate, and creates the corresponding X.509 data structure.

Returns

This function returns a pointer to a X.509 data structure.

7.90 ce_x509cert_rd_buf2

API Name

`ce_x509cert_rd_buf2()`

Syntax

```
void *ce_x509cert_rd_buf2(CE_OP_PTR op, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
buf	Buffer containing X.509 certificate in DER format
len	Length of certificate (bytes)

Description

This function reads a buffer containing a X.509 certificate, and creates the corresponding X.509 structure.

Returns

This function returns a pointer to a X.509 structure.

7.91 ce_x509cert_rd_file

API Name

`ce_x509cert_rd_file()`

Syntax

```
void *ce_x509cert_rd_file(CE_OP_PTR op, char *name);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
name	File containing X.509 certificate in PEM format

Description

This function reads a file containing a X.509 certificate, and creates the corresponding X.509 data structure.

Returns

This function returns a pointer to a X.509 data structure.

7.92 ce_x509cert_rd_prvkey2

API Name

`ce_x509cert_rd_prvkey2()`

Syntax

```
int ce_x509cert_rd_prvkey2(CE_OP_PTR op, int domain, char *name, char
*passphrase, u_char *obuf, int *olen);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier (e.g., CE_X509CERT_DOMAIN_IKE)
name	Name of file containing private key in PEM format
passphrase	Passphrase required to decrypt file (only if private key is encrypted)
obuf	Pointer to output buffer
olen	Pointer to integer containing length of DER-formatted private key (output)

Description

This function reads a file containing a private key, and copies the DER-formatted version into the output buffer provided by the caller. It also updates '*olen' with the amount of data written to 'obuf'.

Returns

This function returns ESUCCESS if the requested operation was performed successfully; otherwise, it returns EFAILURE.

7.93 ce_x509cert_store_add_trusted_cert

API Name

`ce_x509cert_store_add_trusted_cert()`

Syntax

```
int ce_x509cert_store_add_trusted_cert(CE_OP_PTR op, void *store, void
*cert)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
store	Pointer to certificate store
cert	Pointer to X.509 certificate

Description

This function adds a trusted certificate to the specified certificate store.

Returns

This function returns ESUCCESS if the certificate was successfully added; otherwise, it returns EFAILURE.

7.94 ce_x509cert_store_add_untrusted_cert

API Name

`ce_x509cert_store_add_untrusted_cert()`

Syntax

```
int ce_x509cert_store_add_untrusted_cert(CE_OP_PTR op, void *store, void
*cert)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
store	Pointer to certificate store
cert	Pointer to X.509 certificate

Description

This function adds an untrusted certificate to the specified certificate store. When adding multiple untrusted certificates (one at a time) via this function, the last certificate must belong to the entity whose identity is being verified.

Returns

This function returns ESUCCESS if the certificate was successfully added; otherwise, it returns EFAILURE.

7.95 ce_x509cert_store_create

API Name

`ce_x509cert_store_create()`

Syntax

```
void *ce_x509cert_store_create(CE_OP_PTR op, int domain)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
domain	Domain identifier

Description

This function creates a new certificate store for the validation of an entity's X.509 certificate. The validation process uses trusted certificates from the configured domain (specified at creation time) and those that are provided via ce_x509cert_store_add_trusted_cert().

Returns

This function returns a pointer to the newly created certificate store, or NULL if the store could not be created.

7.96 ce_x509cert_store_destroy

API Name

`ce_x509cert_store_destroy()`

Syntax

```
int ce_x509cert_store_destroy(CE_OP_PTR op, void *store)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
store	Pointer to certificate store

Description

This function deletes the specified certificate store.

Returns

This function returns ESUCCESS if the certificate store was successfully deleted; otherwise, it returns EFAILURE.

7.97 ce_x509cert_store_verify

API Name

`ce_x509cert_store_verify()`

Syntax

```
int ce_x509cert_store_verify(CE_OP_PTR op, void *store)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
store	Pointer to certificate store

Description

This function validates an (untrusted) entity's X.509 certificate by using the various trusted and untrusted certificates in the specified certificate store.

Returns

This function returns ESUCCESS if the entity's certificate is validated successfully; otherwise, it returns EFAILURE.

7.98 ce_x509cert_verify

API Name

`ce_x509cert_verify()`

Syntax

```
int ce_x509cert_verify(void *trusted_certs[], int tcount, void
*untrusted_certs[], int utcount)
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
trusted_certs	Array of pointers to trusted X.509 certificate data structures
tcount	Number of elements in the 'trusted_certs' array
untrusted_certs	Array of pointers to untrusted X.509 certificate data structures
utcount	Number of elements in the 'untrusted_certs' array

Description

This function verifies an entity's X.509 certificate using the various trusted and untrusted certificates. The last certificate in the set of untrusted certificates must belong to the entity whose identity is being verified.

Returns

This function returns ESUCCESS if the certificate was successfully deleted; otherwise, it returns EFAILURE.

7.99 ce_x509cert_wr_buf

API Name

`ce_x509cert_wr_buf()`

Syntax

```
void *ce_x509cert_wr_buf(CE_OP_PTR op, void *cert, u_char *buf, int len);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure
buf	Buffer (to be written to)
len	Length of buffer

Description

This function writes the contents of the specified X.509 certificate data structure into the output buffer in DER format.

Returns

This function returns ESUCCESS if the buffer was written to successfully; otherwise, it returns EFAILURE.

7.100 ce_x509cert_wr_file

API Name

`ce_x509cert_wr_file()`

Syntax

```
void *ce_x509cert_wr_file(CE_OP_PTR op, void *cert, char *name, int format);
```

Parameters

op	Pointer to structure containing information about cryptographic operation requested (input) and status (output)
cert	Pointer to X.509 certificate structure
name	Name of file (to be written to)
format	Format of data written to file (DER (CE_FILETYPE_DER) or PEM (CE_FILETYPE_PEM))

Description

This function writes the contents of the X.509 data structure into the specified file in the requested (DER or PEM) format.

Returns

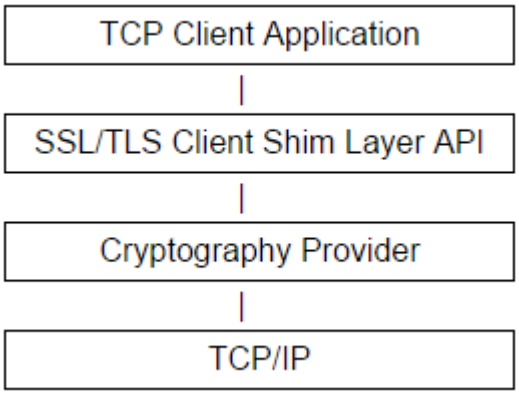
This function returns ESUCCESS if the file was written to successfully; otherwise, it returns EFAILURE.

8 Using the SSL Client Shim

NOTE: This section of the manual pertains only to application developers integrating InterNiche's SSL into their application.

The SSL/TLS client shim layer API provides an abstraction for various aspects of the SSL client operation (especially those related to the initialization and session establishment). This results in a simplification of the process required for the development of an SSL/TLS-enabled TCP client application.

The user can also create a SSL/TLS-enabled TCP client application that only uses the native APIs provided by the underlying cryptography provider.



8.1 SSL/TLS client configuration data structure

All of the SSL/TLS client-related data structures are defined in `h/sslclnt.h`. The various fields in the main SSL/TLS client data structure (struct sslclnt_cfg) are explained below.

name

Name of SSL/TLS client application (e.g., "ESMTP client").

ca_certs

Pointer to an array of one (or more) Certificate Authority X.509 certificate descriptor(s). These certificates are used to validate the X.509 certificate(s) received from the server.

num_ca_certs

Number of elements in the array of Certificate Authority certificate descriptors.

local_certs

Pointer to an array of one (or more) local endpoint X.509 certificate descriptor(s).

num_local_certs

Number of elements in the array of local endpoint certificate descriptors. The first certificate in this array must be the local endpoint's own certificate. The remaining certificates can be intermediate or top-level CA certificates. All of the certificates in this list will be sent to the server during the SSL/TLS handshake process.

local_prvkey

Pointer to a descriptor for local endpoint's private key.

cipher_list

This describes the set of cipher suites (encryption algorithms, authentication algorithms, etc.) that are advertised to the server. The server picks one suite from this list, and that selection determines the various security algorithms used for the secure connection. If set to NULL, the underlying cryptography provider selects a default set of cipher suites based on the enabled algorithms.

verify_peer

This boolean field determines if the peer's identity (based on information extracted from the X.509 certificates) will be verified. The X.509 certificates received from the peer are considered untrusted.

verify_fail_if_no_peer_cert

This boolean field determines if the peer will fail the validation process if 'verify_peer' is TRUE and the peer does not present any X.509 certificate.

grp_handshake_msgs

This boolean field determines if multiple SSL/TLS handshake messages can be sent in the same record.

partial_wr

This boolean field allows a secure write (or send) call to return after sending less than the entire length requested by the caller.

cert_cbk

This field is not used.

ssl_versions

This variable determines the SSL/TLS protocol version used by the client. The client can choose any one of SSLCLNT_SSLV3, SSLCLNT_TLSV1, SSLCLNT_TLSV11, and SSLCLNT_TLSV12 flags to select the corresponding version. Alternatively, he can set this variable to SSLCLNT_ALL_VERSIONS to allow for maximum flexibility. The availability of SSL/TLS protocol versions varies by cryptography provider.

8.2 SSL Shim - Example Configurations

Here is an example configuration of the SSL client functionality in the InterNiche ESMTP module.

```
 {
   struct sslclnt_cfg esmtp_cfg;

   esmtp_cfg.name = "ESMTP client";
   esmtp_cfg.ca_certs = sslclnt_ca_certs;
   esmtp_cfg.num_ca_certs = sizeof(sslclnt_ca_certs)/
                            sizeof(struct sslclnt_x509cert_src);
#ifdef SSLCLNT_CLIENT_AUTH
   esmtp_cfg.local_certs = sslclnt_local_certs;
   esmtp_cfg.num_local_certs = sizeof(sslclnt_local_certs)/
                               sizeof(struct sslclnt_x509cert_src);
   esmtp_cfg.local_prvkey = &sslclnt_prvkey;
#endif
   esmtp_cfg.cipher_list = NULL;
   esmtp_cfg.verify_peer = TRUE;
   esmtp_cfg.verify_fail_if_no_peer_cert = TRUE;
   esmtp_cfg.grp_handshake_msgs = TRUE;
   esmtp_cfg.partial_wr = TRUE;
   esmtp_cfg.blocking_mode = FALSE;
   esmtp_cfg.ssl_versions = SSLCLNT_ALL_VERSIONS;
   esmtp_cfg.cert_cbk = NULL;

   esmtp_ctx = sslclnt_app_init(&esmtp_cfg);

   if (esmtp_ctx == NULL)
   {
      dprintf("esmtp_ssl_cfg: sslclnt_app_init() failed!\n");
      return (EFAILURE);
   }

   return (ESUCCESS);
}
```

Here is an example configuration of the array of Certificate Authority X.509 certificate descriptors.

```
/* DER version */
u_char cacert_der1[] =
{
   0x30, 0x82, 0x03, 0x20, 0x30, 0x82, 0x02, 0x89,
   ...
};

u_char cacert_file3[] = "verisign.pem";

/* list of trusted Certificate Authority certificates (used when validating
 * the X.509 certificate chain received from the server
 */
struct sslclnt_x509cert_src sslclnt_ca_certs[] =
{
   /* buffer(s) */
   {
      cacert_der1,
      sizeof(cacert_der1),
      SSLCLNT_DATA_LOCFMT_BUF_DER,
      SSLCLNT_X509CERT_TYPE_CA
   },
   /* file(s) */
   {
      cacert_file3,
      0,
      SSLCLNT_DATA_LOCFMT_FILE_PEM,
      SSLCLNT_X509CERT_TYPE_CA
   },
};
```

8.3 Client Shim API Usage

The following sequence of steps describes the use of the SSL/TLS client- and socket-related APIs to create a SSL/TLS-enabled TCP client. The man pages provide additional details about the various functions.

1. Create a stream (TCP) socket that will be used as the I/O mechanism by the client application via `t_socket()`.
2. Bind the listening socket to a local address/port combination via `t_bind()`. This is the address that the client will use as its end point identifier.
3. Create and initialize all of the fields in the SSL/TLS client configuration data structure (`struct sslclnt_cfg`), and invoke `sslclnt_app_init()` to create the client application's top-level security context. The client configuration data structure specifies various items required for securing the connection: Certificate Authority X.509 certificate(s), local entity's X.509 certificate(s), local entity's private key file, local entity's passphrase callback, desired cipher suite descriptor, X.509 certificate verification requirements, list of supported SSL/TLS versions, etc. The top-level security context is an "umbrella" data structure, and an application only needs to create one instance of this data structure during its lifetime (regardless of the number of secure connections that it creates to accomplish its goals).
4. Connect to a TCP server by invoking `t_connect()`.
5. When the TCP connection handshake is complete, `t_connect()` will return ESUCCESS. The socket is now ready for the establishment of a secure connection via a SSL/TLS handshake.
6. Invoke `sslsrv_create_conn()` to initiate the SSL/TLS connection handshake. The handshake occurs on the same socket that was used for establishing the TCP connection. Upon the completion of this process, the two endpoints have keys required for the selected encryption and authentication algorithms. The `sslsrv_create_conn()` function returns an identifier for the newly created secure connection. This identifier is used in subsequent calls for sending, receiving, and terminating the connection.
7. The client can invoke sslclnt_send() to send data to the server.
8. The client can invoke sslclnt_recv() to receive data from the server.
9. The client can invoke sslclnt_term_conn() to initiate the termination process on the connection.
10. The client can invoke sslclnt_del_conn() to release all resources held by the connection, and t_socketclose() to close the socket that was used for I/O with the server.
11. When the client decides to shut down, it can invoke sslclnt_app_term() to release the top-level security context.

8.4 SSL Client Shim API

sslclnt_app_init

API Name

`sslclnt_app_init()`

Syntax

```
SSLCLNT_APP_CTX *sslclnt_app_init(struct sslclnt_cfg *cfg);
```

Parameters

cfg	Pointer to structure containing information about client configuration parameters

Description

This function creates a new SSL/TLS-based client application context.

Returns

This function returns a pointer to the newly created context.

sslclnt_create_conn

API Name

`sslclnt_create_conn()`

Syntax

```
SSLCLNT_CONN *sslclnt_create_conn(SSLCLNT_APP_CTX *actx, SSLCLNT_CONN
*conn, SOCKTYPE sock, int *status);
```

Parameters

actx	Pointer to SSL/TLS-based client application's context
conn	Pointer to SSL/TLS-based client application's secure connection
sock	Identifier for TCP socket used for secure connection
status	Pointer to integer for storing the status of the process of establishment of the secure connection (output)

Description

This function creates a new SSL/TLS-based secure connection that can be used for I/O by the client. When using a non-blocking socket, 'conn' is NULL in the first invocation, and non-NULL (i.e., equal to the value returned from the first invocation) in all subsequent invocations. When using a blocking socket, 'conn' is set to NULL. The 'status' variable can be set (by this function) to any one of the following three values: SSLCLNT_CONNECT_INCOMPLETE, SSLCLNT_CONNECT_ERROR, or SSLCLNT_CONNECT_COMPLETE. When using non-blocking sockets, this function will set 'status' to SSLCLNT_CONNECT_INCOMPLETE until the SSL/TLS connection establishment process is complete. A status of SSLCLNT_CONNECT_ERROR indicates that the connection establishment process has failed. An application client can perform data I/O only after the SSL/TLS connection establishment process has successfully completed.

Returns

This function returns a pointer to the newly created connection.

sslclnt_send

API Name

`sslclnt_send()`

Syntax

```
int sslclnt_send(SSLCLNT_CONN *conn, char *buf, int length);
```

Parameters

conn	Pointer to SSL/TLS-based client application's secure connection
buf	Pointer to buffer containing data to be sent
length	Amount of data in 'buf'

Description

This function is used to transmit data on the secure connection.

Returns

This function returns the number of bytes written on the connection or -1 (EFAILURE).

sslclnt_recv

API Name

`sslclnt_recv()`

Syntax

```
int sslclnt_recv(SSLCLNT_CONN *conn, char *buf, int length);
```

Parameters

conn	Pointer to SSL/TLS-based client application's secure connection
buf	Pointer to buffer for receiving data
length	Length of 'buf'

Description

This function is used to receive data on the secure connection.

Returns

This function returns the number of bytes received on the connection, 0 (e.g., if the connection has been closed), or -1 (EFAILURE).

sslclnt_term_conn

API Name

`sslclnt_term_conn()`

Syntax

```
int sslclnt_term_conn(SSLCLNT_CONN *conn, bool_t *bidirectional_shutdown);
```

Parameters

conn	Pointer to SSL/TLS-based client application's secure connection
bidirectional_shutdown	Pointer to boolean parameter (TRUE or FALSE) indicating if the shutdown was bidirectional(output)

Description

This function initiates the termination process for a SSL/TLS-based secure connection by sending a Close Notify alert to the peer.

Returns

This function returns ESUCCESS if the termination was successful; otherwise, it returns EFAILURE.

sslclnt_del_conn

API Name

`sslclnt_del_conn()`

Syntax

```
int sslclnt_del_conn(SSLCLNT_CONN *conn);
```

Parameters

conn	Pointer to SSL/TLS-based client application's secure connection structure

Description

This function deletes all data structures associated with the specified SSL/TLS-based secure connection.

Returns

This function returns ESUCCESS if the deletion was successful; otherwise, it returns EFAILURE.

sslclnt_app_term

API Name

`sslclnt_app_term()`

Syntax

```
int sslclnt_app_term(SSLCLNT_APP_CTX *actx);
```

Parameters

actx	Pointer to SSL/TLS-based client application's context structure (returned from a prior call to sslclnt_app_init())

Description

This function destroys a SSL/TLS-based client application context.

Returns

This function returns ESUCCESS if the cleanup was completed successfully; otherwise, it returns EFAILURE.

sslclnt_get_stats

API Name

`sslclnt_get_stats()`

Syntax

```
void sslclnt_get_stats(SSLCLNT_STATS *stats);
```

Parameters

stats	Pointer to SSL client statistics data structure

Description

This function copies the SSL client statistics into the caller-provided data structure. The statistics are collected across all SSL client-based applications.

Returns

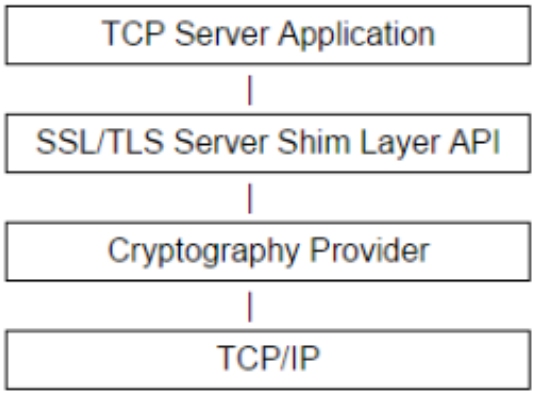
None.

9 Using the SSL Server Shim

NOTE: This section of the manual pertains only to application developers integrating InterNiche's SSL into their application.

The SSL/TLS server shim layer API is a wrapper around the SSL/TLS server functionality provided by a cryptography provider. It provides functions that allow the server to create and use a secure connection for data I/O with a TCP client.

The SSL/TLS server shim layer API provides an abstraction for various aspects of the SSL server operation (especially those related to the initialization and session establishment). This results in a simplification of the process required for the development of an SSL/TLS-enabled TCP server application.



9.1 SSL-TLS server configuration data structure

All of the SSL/TLS server-related data structures are defined in `h/sslsrv.h`. The various fields in the main SSL/TLS server data structure (`struct sslsrv_cfg`) are explained below.

name	Name of SSL/TLS server application (e.g., "HTTP server").
local_certs	Local endpoint X.509 certificate descriptor. The first certificate in this set must be the local endpoint's own certificate. The remaining certificates can be intermediate or top-level CA certificates. All of the certificates in this list will be sent to the client during the SSL/TLS handshake process.
ca_certs	Certificate Authority X.509 certificate descriptor. These certificates are used to validate the X.509 certificate(s) received from the client.
local_prvkey	Local endpoint's private key descriptor.
ctx_pp_cbk	Function pointer for callback function that provides passphrase for decrypting private key. If the private key is not encrypted, this field must be set to NULL.
cipher_list	This describes the set of cipher suites (encryption algorithms, authentication algorithms, etc.) that are advertised to the server. The server picks one suite from this list, and that selection determines the various security algorithms used for the secure connection. If set to NULL, the underlying cryptography provider selects a default set of cipher suites based on the enabled algorithms.
verify_peer	This boolean field determines if the peer's identity (based on information extracted from the X.509 certificates) will be verified. The X.509 certificates received from the peer are considered untrusted.
verify_client_once	This field determines if the server application requests a certificate from the client during the initial handshake only (TRUE), or also during subsequent renegotiations (FALSE).
verify_fail_if_no_peer_cert	This boolean field determines if the peer will fail the validation process if 'verify_peer' is TRUE and the peer does not present any X.509 certificate.
grp_handshake_msgs	This boolean field determines if multiple SSL/TLS handshake messages can be sent in the same record.
partial_wr	This boolean field allows a secure write (or send) call to return after sending less than the entire length requested by the caller.

blocking_mode	This field determines if the listening sockets are to be configured for blocking mode (TRUE) or not (FALSE). An accepted socket (i.e., one returned from the t_accept()) call inherits the blocking capability of its parent listening socket.addrsPointer to an array of IPADDRU_X data structures. Each element in the structure contains an IP address that the server application intends to listen on for incoming connections.
addr_count	This field contains the number of elements in the 'addrs' array.
listen_q_len	This field is passed in as the second parameter to the t_listen() call, and determines the length of the queue of pending connections for the server (in the TCP/IP stack).
port	This field contains the port number at which the server listens for incoming connections. (The same port number if used for TCP/IPv4 and TCP/IPv6 connections.)
ip_versions	This field determines the types of listening sockets (SSLSRV_IPV4 and/or SSLSRV_IPV6) created by the SSL/TLS shim layer on behalf of the user.
ssl_versions	This field determines the SSL/TLS protocol version used by the server. The server can choose any one of SSLSRV_SSLV3, SSLSRV_TLSV1, SSLSRV_TLSV11, and SSLSRV_TLSV12 flags to select the corresponding version. Alternatively, he can set this variable to SSLSRV_ALL_VERSIONS to allow for maximum flexibility. The availability of SSL/TLS protocol versions varies by cryptography provider.
cert_cbk	This field is not used.

The `blocking_mode`, `addrs`, `addr_count`, `listen_q_len`, `port`, and `ip_versions` fields are not relevant if the server application intends to create its own sockets.

9.2 Example Code

The following code fragment provides an example of how to initialize the SSL server configuration data structure.

```
#define TESTAPP_SRV_NAME "TESTAPP server"
#define TESTAPP_CACERT_FILE "cacert.pem"
#define TESTAPP_LOCAL_CERT_FILE "supamcert.pem"
#define TESTAPP_LOCAL_CERTS_FILE "srv_certs.pem"
#define TESTAPP_PRVKEY_FILE "supamkey.pem" /* encrypted */

/* passphrase callback for encrypted private key */
int
testapp_get_passphrase(char *dest, int size, int rw, void *hint)
{
   int length;
   char *passphrase = "secret";

   length = strlen(passphrase);
   if (length > size)
      return (EFAILURE);

   MEMCPY(dest, passphrase, length);

   return (length);
}

{
   struct sslsrv_cfg testapp_cfg;

   char *ca_cert_file = TESTAPP_CACERT_FILE;
   char *local_certs_file = TESTAPP_LOCAL_CERTS_FILE;
   char *local_prvkey_file = TESTAPP_PRVKEY_FILE;
   bool_t multiple_certs;

   /* name of server application */
   testapp_cfg.name = TESTAPP_SRV_NAME;

   /* format and location of Certificate Authority certificates */
   testapp_cfg.ca_certs.format = SSLSRV_FORMAT_PEM;
   testapp_cfg.ca_certs.location = SSLSRV_LOCATION_FILE;
   testapp_cfg.ca_certs.src.files.file = &ca_cert_file;
   multiple_certs = FALSE;
   testapp_cfg.ca_certs.src.files.multiple_certs = &multiple_certs;
   testapp_cfg.ca_certs.src.files.count = 1;

   testapp_cfg.local_certs.format = SSLSRV_FORMAT_PEM;
   testapp_cfg.local_certs.location = SSLSRV_LOCATION_FILE;
   testapp_cfg.local_certs.src.files.file = &local_certs_file;
   multiple_certs = TRUE; /* two certificates */
   testapp_cfg.local_certs.src.files.multiple_certs = &multiple_certs;
   testapp_cfg.local_certs.src.files.count = 1;

   testapp_cfg.local_prvkey.format = SSLSRV_FORMAT_PEM;
```

```
    testapp_cfg.local_prvkey.location = SSLSRV_LOCATION_FILE;
    testapp_cfg.local_prvkey.src.file = local_prvkey_file;

    testapp_cfg.ctx_pp_cbk.hint = NULL;
    testapp_cfg.ctx_pp_cbk.func = testapp_get_passphrase;

    testapp_cfg.cipher_list = NULL;
    testapp_cfg.verify_peer = FALSE;
    testapp_cfg.verify_client_once = FALSE;
    testapp_cfg.verify_fail_if_no_peer_cert = FALSE;
    testapp_cfg.grp_handshake_msgs = FALSE;
    testapp_cfg.partial_wr = FALSE;
    testapp_cfg.addrs = NULL;
    testapp_cfg.blocking_mode = 0;
    testapp_cfg.addr_count = 0;
    testapp_cfg.listen_q_len = 0;
    testapp_cfg.port = 0;
    testapp_cfg.ssl_versions = SSLSRV_ALL_VERSIONS;
    testapp_cfg.cert_cbk = NULL;
}
```

9.3 API Usage

The following sequence of steps describes the use of the SSL/TLS server- and socket-related APIs to create a SSL/TLS-enabled TCP server. The man pages provide additional details about the various functions.

1. Create a stream (TCP) socket that will be used as the listening socket by the server application via t_socket().
2. Bind the listening socket to an address/port combination via t_bind(). This is the address at which the server will listen for incoming client connection requests.
3. Invoke t_listen() to mark the socket as a "listening" socket.
4. Create and initialize all of the fields in the SSL server configuration data structure (struct sslsrv_cfg), and invoke sslsrv_app_init() to create the server application's top-level security context. The server configuration data structure specifies various items required for securing the connection: Certificate Authority X.509 certificate(s), local entity's X.509 certificate(s), local entity's private key file, local entity's passphrase callback, desired cipher suite descriptor, X.509 certificate verification requirements, list of supported SSL/TLS versions, etc. The top-level security context is an "umbrella" data structure, and an application only needs to create one instance of this data structure during its lifetime (regardless of the number of secure connections that it creates to accomplish its goals).
5. Wait for an incoming connection (from a TCP client) by invoking t_accept().
6. When the TCP connection handshake is complete, t_accept() will return to its caller with an identifier for the newly created socket. This socket will be used for all secure communication between the client and the server.
7. Invoke sslsrv_create_conn() to initiate the SSL connection handshake. The handshake occurs on the socket returned from t_accept(). Upon the completion of this process, the two ends have keys required for the selected encryption and authentication algorithms. The sslsrv_create_conn() function returns an identifier for the newly created secure connection. This identifier is used in subsequent calls for sending, receiving, and terminating the connection.
8. The server can obtain the X.509 certificates presented by the client during the secure negotiation process by invoking sslsrv_get_client_certs(). The top-level certificate contains information that identifies the client entity.
9. The server can invoke sslsrv_send() to send data to the client.
10. The server can invoke sslsrv_recv() to receive data from the client.
11. The server can invoke sslsrv_term_conn() to initiate the termination process on the connection.
12. The server can invoke sslsrv_del_conn() to release all resources held by the connection, and t_socketclose() to close the socket that was used for I/O with the client (i.e., the one returned from t_accept()).
13. When the server decides to shut down, it can invoke sslsrv_app_term() to release the top-level security context, and invoke t_socketclose() on the listening socket.

9.4 SSL Server Shim API

sslsrv_app_init

API Name

`sslsrv_app_init()`

Syntax

```
SSLSRV_APP_CTX *sslsrv_app_init(struct sslsrv_cfg *cfg, bool_t
create_socks);
```

Parameters

cfg	Pointer to structure containing information about server configuration parameters
create_socks	Boolean parameter indicating whether this function should create listening socket(s)

Description

This function creates a new SSL/TLS-based server application context (and listening sockets, if requested by caller).

Returns

This function returns a pointer to the newly created context.

sslsrv_create_conn

API Name

`sslsrv_create_conn()`

Syntax

```
SSLSRV_CONN *sslsrv_create_conn(SSLSRV_APP_CTX *actx, SSLSRV_CONN *conn,
SOCKTYPE sock, int *status);
```

Parameters

actx	Pointer to SSL/TLS-based server application's context
conn	Pointer to SSL/TLS-based server application's secure connection
sock	Identifier for TCP socket used for secure connection
status	Pointer to integer for storing the status of the process of establishment of the secure connection (output)

Description

This function creates a new SSL/TLS-based secure connection that can be used for I/O by the server. When using a non-blocking socket, 'conn' is NULL in the first invocation, and non-NULL (i.e., equal to the value returned from the first invocation) in all subsequent invocations. When using a blocking socket, 'conn' is set to NULL. The 'status' variable can be set (by this function) to any one of the following three values: SSL_ACCEPT_INCOMPLETE, SSL_ACCEPT_ERROR, or SSL_ACCEPT_COMPLETE. When using non-blocking sockets, this function will set 'status' to SSL_ACCEPT_INCOMPLETE until the SSL/TLS connection establishment process is complete. A status of SSL_ACCEPT_ERROR indicates that the connection establishment process has failed. An application server can perform data I/O only after the SSL/TLS connection establishment process has successfully completed.

Returns

This function returns a pointer to the newly created connection.

sslsrv_get_client_certs

API Name

`sslsrv_get_client_certs()`

Syntax

```
int sslsrv_get_client_certs(SSLSRV_CONN *conn, SSLSRV_X509CERT_CHAIN
**client_certs);
```

Parameters

conn	Pointer to SSL/TLS-based server application's secure connection
client_certs	Pointer to location to store a pointer to an array of SSLSRV_X509CERT_CHAIN structures (output)

Description

This function retrieves the X.509 certificates provided by the client. Each SSLSRV_X509CERT_CHAIN element contains the DER-formatted data and length of one X.509 certificate from the client's certificate chain.

Returns

This function returns the number of certificates in the client's certificate chain.

sslsrv_send

API Name

`sslsrv_send()`

Syntax

```
int sslsrv_send(SSLSRV_CONN *conn, char *buf, int length);
```

Parameters

conn	Pointer to SSL/TLS-based server application's secure connection
buf	Pointer to buffer containing data to be sent
length	Amount of data in 'buf'

Description

This function is used to transmit data on the secure connection.

Returns

This function returns the number of bytes written on the connection or -1 (EFAILURE).

sslsrv_recv

API Name

`sslsrv_recv()`

Syntax

```
int sslsrv_recv(SSLSRV_CONN *conn, char *buf, int length);
```

Parameters

conn	Pointer to SSL/TLS-based server application's secure connection
buf	Pointer to buffer for receiving data
length	Length of 'buf'

Description

This function is used to receive data on the secure connection.

Returns

This function returns the number of bytes received on the connection, 0 (e.g., if the connection has been closed), or -1 (EFAILURE).

sslsrv_term_conn

API Name

`sslsrv_term_conn()`

Syntax

```
int sslsrv_term_conn(SSLSRV_CONN *conn, bool_t *bidirectional_shutdown);
```

Parameters

conn	Pointer to SSL/TLS-based server application's secure connection
bidirectional_shutdown	Pointer to boolean parameter (TRUE or FALSE) indicating if the shutdown was bidirectional(output)

Description

This function initiates the termination process for a SSL/TLS-based secure connection by sending a Close Notify alert to the peer.

Returns

This function returns ESUCCESS if the termination was successful; otherwise, it returns EFAILURE.

sslsrv_del_conn

API Name

`sslsrv_del_conn()`

Syntax

```
int sslsrv_del_conn(SSLSRV_CONN *conn);
```

Parameters

conn	Pointer to SSL/TLS-based server application's secure connection

Description

This function deletes all data structures associated with the specified SSL/TLS-based secure connection.

Returns

This function returns ESUCCESS if the deletion was successful; otherwise, it returns EFAILURE.

sslsrv_app_term

API Name

`sslsrv_app_term()`

Syntax

```
int sslsrv_app_term(SSLSRV_APP_CTX *actx);
```

Parameters

actx	Pointer to SSL/TLS-based server application's context structure (returned from a prior call to sslsrv_app_init())

Description

This function destroys a SSL/TLS-based server application context.

Returns

This function returns ESUCCESS if the cleanup was completed successfully; otherwise, it returns EFAILURE.

sslsrv_get_stats

API Name

`sslsrv_get_stats()`

Syntax

```
void sslsrv_get_stats(SSLSRV_STATS *stats);
```

Parameters

stats	Pointer to SSL server statistics data structure

Description

This function copies the SSL server statistics into the caller-provided data structure. The statistics are collected across all SSL server-based applications.

Returns

None.

sslsrv_get_conn_err

API Name

`sslsrv_get_conn_err()`

Syntax

```
int sslsrv_get_conn_err(SSLSRV_CONN *conn);
```

Parameters

conn	Pointer to SSL/TLS-based server application's secure connection

Description

This function retrieves the error number (errno) associated with the TCP socket used for I/O by the secure connection.

Returns

This function returns the error number for the specified connection; if the connection is NULL, it returns EINVAL.