NAME

OSSL_CMP_CTX_new, OSSL_CMP_CTX_free, OSSL_CMP_CTX_reinit, OSSL_CMP_CTX_set_option, OSSL_CMP_CTX_get_option, OSSL_CMP_CTX_set_log_cb, OSSL_CMP_CTX_set_log_verbosity, OSSL_CMP_CTX_print_errors, OSSL_CMP_CTX_set1_serverPath, OSSL_CMP_CTX_set1_server, OSSL_CMP_CTX_set_serverPort, OSSL_CMP_CTX_set1_proxy, OSSL_CMP_CTX_set1_no_proxy, OSSL_CMP_CTX_set_http_cb, OSSL_CMP_CTX_set_http_cb_arg, OSSL_CMP_CTX_get_http_cb_arg, OSSL_CMP_transfer_cb_t, OSSL_CMP_CTX_set_transfer_cb, OSSL_CMP_CTX_set_transfer_cb_arg, OSSL_CMP_CTX_get_transfer_cb_arg, OSSL_CMP_CTX_set1_srvCert, OSSL_CMP_CTX_set1_expected_sender, OSSL_CMP_CTX_set0_trustedStore, OSSL_CMP_CTX_get0_trustedStore, OSSL_CMP_CTX_set1_untrusted, OSSL_CMP_CTX_get0_untrusted, OSSL_CMP_CTX_set1_cert, OSSL_CMP_CTX_build_cert_chain, OSSL_CMP_CTX_set1_pkey, OSSL_CMP_CTX_set1_referenceValue, OSSL_CMP_CTX_set1_secretValue, OSSL_CMP_CTX_set1_recipient, OSSL_CMP_CTX_push0_geninfo_ITAV, OSSL_CMP_CTX_reset_geninfo_ITAVs, OSSL_CMP_CTX_set1_extraCertsOut, OSSL_CMP_CTX_set0_newPkey, OSSL_CMP_CTX_get0_newPkey, OSSL_CMP_CTX_set1_issuer, OSSL_CMP_CTX_set1_subjectName, OSSL_CMP_CTX_push1_subjectAltName, OSSL_CMP_CTX_set0_reqExtensions, OSSL_CMP_CTX_reqExtensions_have_SAN, OSSL_CMP_CTX_push0_policy, OSSL_CMP_CTX_set1_oldCert, OSSL_CMP_CTX_set1_p10CSR, OSSL_CMP_CTX_push0_genm_ITAV, OSSL_CMP_certConf_cb_t, OSSL_CMP_certConf_cb, OSSL_CMP_CTX_set_certConf_cb, OSSL_CMP_CTX_set_certConf_cb_arg, OSSL_CMP_CTX_get_certConf_cb_arg, OSSL_CMP_CTX_get_status, OSSL_CMP_CTX_get0_statusString, OSSL_CMP_CTX_get_failInfoCode, OSSL_CMP_CTX_get0_newCert, OSSL_CMP_CTX_get1_newChain, OSSL_CMP_CTX_get1_caPubs, OSSL_CMP_CTX_get1_extraCertsIn, OSSL_CMP_CTX_set1_transactionID, OSSL_CMP_CTX_set1_senderNonce - functions for managing the CMP client context data structure

SYNOPSIS

```
#include <openssl/cmp.h>

OSSL_CMP_CTX *OSSL_CMP_CTX_new(OSSL_LIB_CTX *libctx, const char *propq);
void OSSL_CMP_CTX_free(OSSL_CMP_CTX *ctx);
int OSSL_CMP_CTX_reinit(OSSL_CMP_CTX *ctx);
int OSSL_CMP_CTX_set_option(OSSL_CMP_CTX *ctx, int opt, int val);
int OSSL_CMP_CTX_get_option(const OSSL_CMP_CTX *ctx, int opt);

/* logging and error reporting: */
int OSSL_CMP_CTX_set_log_cb(OSSL_CMP_CTX *ctx, OSSL_CMP_log_cb_t cb);
```

```
#define OSSL_CMP_CTX_set_log_verbosity(ctx, level)
void OSSL_CMP_CTX_print_errors(const OSSL_CMP_CTX *ctx);

/* message transfer: */
int OSSL_CMP_CTX_set1_serverPath(OSSL_CMP_CTX *ctx, const char *path);
int OSSL_CMP_CTX_set1_server(OSSL_CMP_CTX *ctx, const char *address);
int OSSL_CMP_CTX_set_serverPort(OSSL_CMP_CTX *ctx, int port);
int OSSL_CMP_CTX_set1_proxy(OSSL_CMP_CTX *ctx, const char *name);
int OSSL_CMP_CTX_set1_no_proxy(OSSL_CMP_CTX *ctx, const char *names);
int OSSL_CMP_CTX_set_http_cb(OSSL_CMP_CTX *ctx, HTTP_bio_cb_t cb);
int OSSL_CMP_CTX_set_http_cb_arg(OSSL_CMP_CTX *ctx, void *arg);
void *OSSL_CMP_CTX_get_http_cb_arg(const OSSL_CMP_CTX *ctx);
typedef OSSL_CMP_MSG *(*OSSL_CMP_transfer_cb_t)(OSSL_CMP_CTX *ctx,
                                                const OSSL_CMP_MSG *req);
int OSSL_CMP_CTX_set_transfer_cb(OSSL_CMP_CTX *ctx,
                                 OSSL_CMP_transfer_cb_t cb);
int OSSL_CMP_CTX_set_transfer_cb_arg(OSSL_CMP_CTX *ctx, void *arg);
void *OSSL_CMP_CTX_get_transfer_cb_arg(const OSSL_CMP_CTX *ctx);

/* server authentication: */
int OSSL_CMP_CTX_set1_srvCert(OSSL_CMP_CTX *ctx, X509 *cert);
int OSSL_CMP_CTX_set1_expected_sender(OSSL_CMP_CTX *ctx,
                                      const X509_NAME *name);
int OSSL_CMP_CTX_set0_trustedStore(OSSL_CMP_CTX *ctx, X509_STORE *store);
X509_STORE *OSSL_CMP_CTX_get0_trustedStore(const OSSL_CMP_CTX *ctx);
int OSSL_CMP_CTX_set1_untrusted(OSSL_CMP_CTX *ctx, STACK_OF(X509) *certs);
STACK_OF(X509) *OSSL_CMP_CTX_get0_untrusted(const OSSL_CMP_CTX *ctx);

/* client authentication: */
int OSSL_CMP_CTX_set1_cert(OSSL_CMP_CTX *ctx, X509 *cert);
int OSSL_CMP_CTX_build_cert_chain(OSSL_CMP_CTX *ctx, X509_STORE *own_trusted,
                                  STACK_OF(X509) *candidates);
int OSSL_CMP_CTX_set1_pkey(OSSL_CMP_CTX *ctx, EVP_PKEY *pkey);
int OSSL_CMP_CTX_set1_referenceValue(OSSL_CMP_CTX *ctx,
                                     const unsigned char *ref, int len);
int OSSL_CMP_CTX_set1_secretValue(OSSL_CMP_CTX *ctx,
                                  const unsigned char *sec, int len);

/* CMP message header and extra certificates: */
int OSSL_CMP_CTX_set1_recipient(OSSL_CMP_CTX *ctx, const X509_NAME *name);
```

```
int OSSL_CMP_CTX_push0_geninfo_ITAV(OSSL_CMP_CTX *ctx, OSSL_CMP_ITAV *itav);
int OSSL_CMP_CTX_reset_geninfo_ITAVs(OSSL_CMP_CTX *ctx);
int OSSL_CMP_CTX_set1_extraCertsOut(OSSL_CMP_CTX *ctx,
                                    STACK_OF(X509) *extraCertsOut);

/* certificate template: */
int OSSL_CMP_CTX_set0_newPkey(OSSL_CMP_CTX *ctx, int priv, EVP_PKEY *pkey);
EVP_PKEY *OSSL_CMP_CTX_get0_newPkey(const OSSL_CMP_CTX *ctx, int priv);
int OSSL_CMP_CTX_set1_issuer(OSSL_CMP_CTX *ctx, const X509_NAME *name);
int OSSL_CMP_CTX_set1_subjectName(OSSL_CMP_CTX *ctx, const X509_NAME *name);
int OSSL_CMP_CTX_push1_subjectAltName(OSSL_CMP_CTX *ctx,
                                      const GENERAL_NAME *name);
int OSSL_CMP_CTX_set0_reqExtensions(OSSL_CMP_CTX *ctx, X509_EXTENSIONS *exts);
int OSSL_CMP_CTX_reqExtensions_have_SAN(OSSL_CMP_CTX *ctx);
int OSSL_CMP_CTX_push0_policy(OSSL_CMP_CTX *ctx, POLICYINFO *pinfo);
int OSSL_CMP_CTX_set1_oldCert(OSSL_CMP_CTX *ctx, X509 *cert);
int OSSL_CMP_CTX_set1_p10CSR(OSSL_CMP_CTX *ctx, const X509_REQ *csr);

/* misc body contents: */
int OSSL_CMP_CTX_push0_genm_ITAV(OSSL_CMP_CTX *ctx, OSSL_CMP_ITAV *itav);

/* certificate confirmation: */
typedef int (*OSSL_CMP_certConf_cb_t)(OSSL_CMP_CTX *ctx, X509 *cert,
                                      int fail_info, const char **txt);
int OSSL_CMP_certConf_cb(OSSL_CMP_CTX *ctx, X509 *cert, int fail_info,
                         const char **text);
int OSSL_CMP_CTX_set_certConf_cb(OSSL_CMP_CTX *ctx, OSSL_CMP_certConf_cb_t cb);
int OSSL_CMP_CTX_set_certConf_cb_arg(OSSL_CMP_CTX *ctx, void *arg);
void *OSSL_CMP_CTX_get_certConf_cb_arg(const OSSL_CMP_CTX *ctx);

/* result fetching: */
int OSSL_CMP_CTX_get_status(const OSSL_CMP_CTX *ctx);
OSSL_CMP_PKIFREETEXT *OSSL_CMP_CTX_get0_statusString(const OSSL_CMP_CTX *ctx);
int OSSL_CMP_CTX_get_failInfoCode(const OSSL_CMP_CTX *ctx);

X509 *OSSL_CMP_CTX_get0_newCert(const OSSL_CMP_CTX *ctx);
STACK_OF(X509) *OSSL_CMP_CTX_get1_newChain(const OSSL_CMP_CTX *ctx);
STACK_OF(X509) *OSSL_CMP_CTX_get1_caPubs(const OSSL_CMP_CTX *ctx);
STACK_OF(X509) *OSSL_CMP_CTX_get1_extraCertsIn(const OSSL_CMP_CTX *ctx);
```

```
/* for testing and debugging purposes: */
int OSSL_CMP_CTX_set1_transactionID(OSSL_CMP_CTX *ctx,
                                    const ASN1_OCTET_STRING *id);
int OSSL_CMP_CTX_set1_senderNonce(OSSL_CMP_CTX *ctx,
                                  const ASN1_OCTET_STRING *nonce);
```

DESCRIPTION

This is the context API for using CMP (Certificate Management Protocol) with OpenSSL.

OSSL_CMP_CTX_new() allocates an **OSSL_CMP_CTX** structure associated with the library context *libctx* and property query string *propq*, both of which may be NULL to select the defaults. It initializes the remaining fields to their default values - for instance, the logging verbosity is set to OSSL_CMP_LOG_INFO, the message timeout is set to 120 seconds, and the proof-of-possession method is set to OSSL_CRMF_POPO_SIGNATURE.

OSSL_CMP_CTX_free() deallocates an OSSL_CMP_CTX structure.

OSSL_CMP_CTX_reinit() prepares the given *ctx* for a further transaction by clearing the internal CMP transaction (aka session) status, PKIStatusInfo, and any previous results (newCert, newChain, caPubs, and extraCertsIn) from the last executed transaction. It also clears any ITAVs that were added by **OSSL_CMP_CTX_push0_genm_ITAV()**. All other field values (i.e., CMP options) are retained for potential reuse.

OSSL_CMP_CTX_set_option() sets the given value for the given option (e.g., OSSL_CMP_OPT_IMPLICIT_CONFIRM) in the given OSSL_CMP_CTX structure.

The following options can be set:

OSSL_CMP_OPT_LOG_VERBOSITY

The level of severity needed for actually outputting log messages
due to errors, warnings, general info, debugging, etc.
Default is OSSL_CMP_LOG_INFO. See also L<OSSL_CMP_log_open(3)>.

OSSL_CMP_OPT_KEEP_ALIVE

If the given value is 0 then HTTP connections are not kept open
after receiving a response, which is the default behavior for HTTP 1.0.
If the value is 1 or 2 then persistent connections are requested.
If the value is 2 then persistent connections are required,
i.e., in case the server does not grant them an error occurs.
The default value is 1: prefer to keep the connection open.

OSSL_CMP_OPT_MSG_TIMEOUT

Number of seconds a CMP request-response message round trip
is allowed to take before a timeout error is returned.
A value <= 0 means no limitation (waiting indefinitely).
Default is to use the B<OSSL_CMP_OPT_TOTAL_TIMEOUT> setting.

OSSL_CMP_OPT_TOTAL_TIMEOUT

Maximum total number of seconds a transaction may take,
including polling etc.
A value <= 0 means no limitation (waiting indefinitely).
Default is 0.

OSSL_CMP_OPT_VALIDITY_DAYS

Number of days new certificates are asked to be valid for.

OSSL_CMP_OPT_SUBJECTALTNAME_NODEFAULT

Do not take default Subject Alternative Names
from the reference certificate.

OSSL_CMP_OPT_SUBJECTALTNAME_CRITICAL

Demand that the given Subject Alternative Names are flagged as critical.

OSSL_CMP_OPT_POLICIES_CRITICAL

Demand that the given policies are flagged as critical.

OSSL_CMP_OPT_POPO_METHOD

Select the proof of possession method to use. Possible values are:

OSSL_CRMF_POPO_NONE - ProofOfPossession field omitted
OSSL_CRMF_POPO_RAVERIFIED - assert that the RA has already
verified the PoPo
OSSL_CRMF_POPO_SIGNATURE - sign a value with private key,
which is the default.
OSSL_CRMF_POPO_KEYENC - decrypt the encrypted certificate
("indirect method")

Note that a signature-based POPO can only be produced if a private key
is provided as the newPkey or client’s pkey component of the CMP context.

OSSL_CMP_OPT_DIGEST_ALGNID

The NID of the digest algorithm to be used in RFC 4210's MSG_SIG_ALG for signature-based message protection and Proof-of-Possession (POPO). Default is SHA256.

OSSL_CMP_OPT_OWF_ALGNID The NID of the digest algorithm to be used as one-way function (OWF) for MAC-based message protection with password-based MAC (PBM). See RFC 4210 section 5.1.3.1 for details. Default is SHA256.

OSSL_CMP_OPT_MAC_ALGNID The NID of the MAC algorithm to be used for message protection with PBM. Default is HMAC-SHA1 as per RFC 4210.

OSSL_CMP_OPT_REVOCATION_REASON

The reason code to be included in a Revocation Request (RR); values: 0..10 (RFC 5210, 5.3.1) or -1 for none, which is the default.

OSSL_CMP_OPT_IMPLICIT_CONFIRM

Request server to enable implicit confirm mode, where the client does not need to send confirmation upon receiving the certificate. If the server does not enable implicit confirmation in the return message, then confirmation is sent anyway.

OSSL_CMP_OPT_DISABLE_CONFIRM

Do not confirm enrolled certificates, to cope with broken servers not supporting implicit confirmation correctly.

B<WARNING:> This setting leads to unspecified behavior and it is meant exclusively to allow interoperability with server implementations violating RFC 4210.

OSSL_CMP_OPT_UNPROTECTED_SEND

Send request or response messages without CMP-level protection.

OSSL_CMP_OPT_UNPROTECTED_ERRORS

Accept unprotected error responses which are either explicitly unprotected or where protection verification failed. Applies to regular error messages as well as certificate responses (IP/CP/KUP) and revocation responses (RP) with rejection.

B<WARNING:> This setting leads to unspecified behavior and it is meant exclusively to allow interoperability with server implementations violating RFC 4210.

OSSL_CMP_OPT_IGNORE_KEYUSAGE

Ignore key usage restrictions in the signer's certificate when

validating signature-based protection in received CMP messages.
Else, 'digitalSignature' must be allowed by CMP signer certificates.

OSSL_CMP_OPT_PERMIT_TA_IN_EXTRACERTS_FOR_IR
Allow retrieving a trust anchor from extraCerts and using that to validate the certificate chain of an IP message.

OSSL_CMP_CTX_get_option() reads the current value of the given option (e.g., OSSL_CMP_OPT_IMPLICIT_CONFIRM) from the given OSSL_CMP_CTX structure.

OSSL_CMP_CTX_set_log_cb() sets in *ctx* the callback function *cb* for handling error queue entries and logging messages. When *cb* is NULL errors are printed to STDERR (if available, else ignored) any log messages are ignored. Alternatively, **OSSL_CMP_log_open**(3) may be used to direct logging to STDOUT.

OSSL_CMP_CTX_set_log_verbosity() is a macro setting the OSSL_CMP_OPT_LOG_VERBOSITY context option to the given level.

OSSL_CMP_CTX_print_errors() outputs any entries in the OpenSSL error queue. It is similar to **ERR_print_errors_cb**(3) but uses the CMP log callback function if set in the *ctx* for uniformity with CMP logging if given. Otherwise it uses **ERR_print_errors**(3) to print to STDERR (unless OPENSSL_NO_STDIO is defined).

OSSL_CMP_CTX_set1_serverPath() sets the HTTP path of the CMP server on the host, also known as "CMP alias". The default is "/".

OSSL_CMP_CTX_set1_server() sets the given server *address* (which may be a hostname or IP address or NULL) in the given *ctx*.

OSSL_CMP_CTX_set_serverPort() sets the port of the CMP server to connect to. If not used or the *port* argument is 0 the default port applies, which is 80 for HTTP and 443 for HTTPS.

OSSL_CMP_CTX_set1_proxy() sets the HTTP proxy to be used for connecting to the given CMP server unless overruled by any "no_proxy" settings (see below). If TLS is not used this defaults to the value of the environment variable "http_proxy" if set, else "HTTP_PROXY". Otherwise defaults to the value of "https_proxy" if set, else "HTTPS_PROXY". An empty proxy string specifies not to use a proxy. Else the format is "[http[s]://]address[:port][/path]", where any path given is ignored. The default port number is 80, or 443 in case "https:" is given.

OSSL_CMP_CTX_set1_no_proxy() sets the list of server hostnames not to use an HTTP proxy for.

The names may be separated by commas and/or whitespace. Defaults to the environment variable "no_proxy" if set, else "NO_PROXY".

OSSL_CMP_CTX_set_http_cb() sets the optional BIO connect/disconnect callback function, which has the prototype

```
typedef BIO *(*HTTP_bio_cb_t) (BIO *bio, void *ctx, int connect, int detail);
```

The callback may modify the *bio* provided by **OSSL_CMP_MSG_http_perform**(3), whereby it may make use of a custom defined argument *ctx* stored in the OSSL_CMP_CTX by means of **OSSL_CMP_CTX_set_http_cb_arg()**. During connection establishment, just after calling **BIO_do_connect_retry()**, the function is invoked with the *connect* argument being 1 and the *detail* argument being 1 if HTTPS is requested, i.e., SSL/TLS should be enabled. On disconnect *connect* is 0 and *detail* is 1 in case no error occurred, else 0. For instance, on connect the function may prepend a TLS BIO to implement HTTPS; after disconnect it may do some diagnostic output and/or specific cleanup. The function should return NULL to indicate failure. After disconnect the modified BIO will be deallocated using **BIO_free_all()**.

OSSL_CMP_CTX_set_http_cb_arg() sets an argument, respectively a pointer to a structure containing arguments, optionally to be used by the http connect/disconnect callback function. *arg* is not consumed, and it must therefore explicitly be freed when not needed any more. *arg* may be NULL to clear the entry.

OSSL_CMP_CTX_get_http_cb_arg() gets the argument, respectively the pointer to a structure containing arguments, previously set by **OSSL_CMP_CTX_set_http_cb_arg()** or NULL if unset.

OSSL_CMP_CTX_set_transfer_cb() sets the message transfer callback function, which has the type

```
typedef OSSL_CMP_MSG *(*OSSL_CMP_transfer_cb_t) (OSSL_CMP_CTX *ctx,
                                                 const OSSL_CMP_MSG *req);
```

Returns 1 on success, 0 on error.

Default is NULL, which implies the use of **OSSL_CMP_MSG_http_perform**(3). The callback should send the CMP request message it obtains via the *req* parameter and on success return the response, else it must return NULL. The transfer callback may make use of a custom defined argument stored in the ctx by means of **OSSL_CMP_CTX_set_transfer_cb_arg()**, which may be retrieved again through **OSSL_CMP_CTX_get_transfer_cb_arg()**.

OSSL_CMP_CTX_set_transfer_cb_arg() sets an argument, respectively a pointer to a structure

containing arguments, optionally to be used by the transfer callback. *arg* is not consumed, and it must therefore explicitly be freed when not needed any more. *arg* may be NULL to clear the entry.

OSSL_CMP_CTX_get_transfer_cb_arg() gets the argument, respectively the pointer to a structure containing arguments, previously set by **OSSL_CMP_CTX_set_transfer_cb_arg()** or NULL if unset.

OSSL_CMP_CTX_set1_srvCert() sets the expected server cert in *ctx* and trusts it directly (even if it is expired) when verifying signed response messages. This pins the accepted CMP server and results in ignoring whatever may be set using **OSSL_CMP_CTX_set0_trustedStore()**. Any previously set value is freed. The *cert* argument may be NULL to clear the entry. If set, the subject of the certificate is also used as default value for the recipient of CMP requests and as default value for the expected sender of CMP responses.

OSSL_CMP_CTX_set1_expected_sender() sets the Distinguished Name (DN) expected in the sender field of incoming CMP messages. Defaults to the subject of the pinned server certificate, if any. This can be used to make sure that only a particular entity is accepted as CMP message signer, and attackers are not able to use arbitrary certificates of a trusted PKI hierarchy to fraudulently pose as CMP server. Note that this gives slightly more freedom than **OSSL_CMP_CTX_set1_srvCert()**, which pins the server to the holder of a particular certificate, while the expected sender name will continue to match after updates of the server cert.

OSSL_CMP_CTX_set0_trustedStore() sets in the CMP context *ctx* the certificate store of type X509_STORE containing trusted certificates, typically of root CAs. This is ignored when a certificate is pinned using **OSSL_CMP_CTX_set1_srvCert()**. The store may also hold CRLs and a certificate verification callback function used for signature-based peer authentication. Any store entry already set before is freed. When given a NULL parameter the entry is cleared.

OSSL_CMP_CTX_get0_trustedStore() extracts from the CMP context *ctx* the pointer to the currently set certificate store containing trust anchors etc., or an empty store if unset.

OSSL_CMP_CTX_set1_untrusted() sets up a list of non-trusted certificates of intermediate CAs that may be useful for path construction for the own CMP signer certificate, for the own TLS certificate (if any), when verifying peer CMP protection certificates, and when verifying newly enrolled certificates. The reference counts of those certificates handled successfully are increased.

OSSL_CMP_CTX_get0_untrusted(OSSL_CMP_CTX *ctx) returns a pointer to the list of untrusted certs, which may be empty if unset.

OSSL_CMP_CTX_set1_cert() sets the CMP signer certificate, also called protection certificate, related to the private key for signature-based message protection. Therefore the public key of this *cert* must

correspond to the private key set before or thereafter via **OSSL_CMP_CTX_set1_pkey()**. When using signature-based protection of CMP request messages this CMP signer certificate will be included first in the extraCerts field. It serves as fallback reference certificate, see **OSSL_CMP_CTX_set1_oldCert()**. The subject of this *cert* will be used as the sender field of outgoing messages, while the subject of any cert set via **OSSL_CMP_CTX_set1_oldCert()** and any value set via **OSSL_CMP_CTX_set1_subjectName()** are used as fallback.

The *cert* argument may be NULL to clear the entry.

OSSL_CMP_CTX_build_cert_chain() builds a certificate chain for the CMP signer certificate previously set in the *ctx*. It adds the optional *candidates*, a list of intermediate CA certs that may already constitute the targeted chain, to the untrusted certs that may already exist in the *ctx*. Then the function uses this augmented set of certs for chain construction. If *own_trusted* is NULL it builds the chain as far down as possible and ignores any verification errors. Else the CMP signer certificate must be verifiable where the chain reaches a trust anchor contained in *own_trusted*. On success the function stores the resulting chain in *ctx* for inclusion in the extraCerts field of signature-protected messages. Calling this function is optional; by default a chain construction is performed on demand that is equivalent to calling this function with the *candidates* and *own_trusted* arguments being NULL.

OSSL_CMP_CTX_set1_pkey() sets the client's private key corresponding to the CMP signer certificate set via **OSSL_CMP_CTX_set1_cert()**. This key is used create signature-based protection (protectionAlg = MSG_SIG_ALG) of outgoing messages unless a symmetric secret has been set via **OSSL_CMP_CTX_set1_secretValue()**. The *pkey* argument may be NULL to clear the entry.

OSSL_CMP_CTX_set1_secretValue() sets in *ctx* the byte string *sec* of length *len* to use as pre-shared secret, or clears it if the *sec* argument is NULL. If present, this secret is used to create MAC-based authentication and integrity protection (rather than applying signature-based protection) of outgoing messages and to verify authenticity and integrity of incoming messages that have MAC-based protection (protectionAlg = "MSG_MAC_ALG").

OSSL_CMP_CTX_set1_referenceValue() sets the given referenceValue *ref* with length *len* in the given *ctx* or clears it if the *ref* argument is NULL. According to RFC 4210 section 5.1.1, if no value for the sender field in CMP message headers can be determined (i.e., no CMP signer certificate and no subject DN is set via **OSSL_CMP_CTX_set1_subjectName()** then the sender field will contain the NULL-DN and the senderKID field of the CMP message header must be set. When signature-based protection is used the senderKID will be set to the subjectKeyIdentifier of the CMP signer certificate as far as present. If not present or when MAC-based protection is used the *ref* value is taken as the fallback value for the senderKID.

OSSL_CMP_CTX_set1_recipient() sets the recipient name that will be used in the PKIHeader of CMP

request messages, i.e. the X509 name of the (CA) server.

The recipient field in the header of a CMP message is mandatory. If not given explicitly the recipient is determined in the following order: the subject of the CMP server certificate set using **OSSL_CMP_CTX_set1_srvCert()**, the value set using **OSSL_CMP_CTX_set1_issuer()**, the issuer of the certificate set using **OSSL_CMP_CTX_set1_oldCert()**, the issuer of the CMP signer certificate, as far as any of those is present, else the NULL-DN as last resort.

OSSL_CMP_CTX_push0_geninfo_ITAV() adds *itav* to the stack in the *ctx* to be added to the GeneralInfo field of the CMP PKIMessage header of a request message sent with this context.

OSSL_CMP_CTX_reset_geninfo_ITAVs() clears any ITAVs that were added by **OSSL_CMP_CTX_push0_geninfo_ITAV()**.

OSSL_CMP_CTX_set1_extraCertsOut() sets the stack of extraCerts that will be sent to remote.

OSSL_CMP_CTX_set0_newPkey() can be used to explicitly set the given EVP_PKEY structure as the private or public key to be certified in the CMP context. The *priv* parameter must be 0 if and only if the given key is a public key.

OSSL_CMP_CTX_get0_newPkey() gives the key to use for certificate enrollment dependent on fields of the CMP context structure: the newPkey (which may be a private or public key) if present, else the public key in the p10CSR if present, else the client's private key. If the *priv* parameter is not 0 and the selected key does not have a private component then NULL is returned.

OSSL_CMP_CTX_set1_issuer() sets the name of the intended issuer that will be set in the CertTemplate, i.e., the X509 name of the CA server.

OSSL_CMP_CTX_set1_subjectName() sets the subject DN that will be used in the CertTemplate structure when requesting a new cert. For Key Update Requests (KUR), it defaults to the subject DN of the reference certificate, see **OSSL_CMP_CTX_set1_oldCert()**. This default is used for Initialization Requests (IR) and Certification Requests (CR) only if no SANs are set. The *subjectName* is also used as fallback for the sender field of outgoing CMP messages if no reference certificate is available.

OSSL_CMP_CTX_push1_subjectAltName() adds the given X509 name to the list of alternate names on the certificate template request. This cannot be used if any Subject Alternative Name extension is set via **OSSL_CMP_CTX_set0_reqExtensions()**. By default, unless **OSSL_CMP_OPT_SUBJECTALTNAME_NODEFAULT** has been set, the Subject Alternative Names are copied from the reference certificate, see **OSSL_CMP_CTX_set1_oldCert()**. If set and the subject DN is not set with **OSSL_CMP_CTX_set1_subjectName()** then the certificate template of an IR and

CR will not be filled with the default subject DN from the reference certificate. If a subject DN is desired it needs to be set explicitly with **OSSL_CMP_CTX_set1_subjectName()**.

OSSL_CMP_CTX_set0_reqExtensions() sets the X.509v3 extensions to be used in IR/CR/KUR.

OSSL_CMP_CTX_reqExtensions_have_SAN() returns 1 if the context contains a Subject Alternative Name extension, else 0 or -1 on error.

OSSL_CMP_CTX_push0_policy() adds the certificate policy info object to the X509_EXTENSIONS of the requested certificate template.

OSSL_CMP_CTX_set1_oldCert() sets the old certificate to be updated in Key Update Requests (KUR) or to be revoked in Revocation Requests (RR). It must be given for RR, else it defaults to the CMP signer certificate. The *reference certificate* determined in this way, if any, is also used for deriving default subject DN, public key, Subject Alternative Names, and the default issuer entry in the requested certificate template of IR/CR/KUR. The subject of the reference certificate is used as the sender field value in CMP message headers. Its issuer is used as default recipient in CMP message headers.

OSSL_CMP_CTX_set1_p10CSR() sets the PKCS#10 CSR to use in P10CR messages. If such a CSR is provided, its subject, public key, and extension fields are also used as fallback values for the certificate template of IR/CR/KUR messages.

OSSL_CMP_CTX_push0_genm_ITAV() adds *itav* to the stack in the *ctx* which will be the body of a General Message sent with this context.

OSSL_CMP_certConf_cb() is the default certificate confirmation callback function. If the callback argument is not NULL it must point to a trust store. In this case the function checks that the newly enrolled certificate can be verified using this trust store and untrusted certificates from the *ctx*, which have been augmented by the list of extraCerts received. During this verification, any certificate status checking is disabled. If the callback argument is NULL the function tries building an approximate chain as far as possible using the same untrusted certificates from the *ctx*, and if this fails it takes the received extraCerts as fallback. The resulting cert chain can be retrieved using **OSSL_CMP_CTX_get1_newChain()**.

OSSL_CMP_CTX_set_certConf_cb() sets the callback used for evaluating the newly enrolled certificate before the library sends, depending on its result, a positive or negative certConf message to the server. The callback has type

```
typedef int (*OSSL_CMP_certConf_cb_t) (OSSL_CMP_CTX *ctx, X509 *cert,
                                       int fail_info, const char **txt);
```

and should inspect the certificate it obtains via the *cert* parameter and may overrule the pre-decision given in the *fail_info* and **txt* parameters. If it accepts the certificate it must return 0, indicating success. Else it must return a bit field reflecting PKIFailureInfo with at least one failure bit and may set the **txt* output parameter to point to a string constant with more detail. The transfer callback may make use of a custom defined argument stored in the *ctx* by means of **OSSL_CMP_CTX_set_certConf_cb_arg()**, which may be retrieved again through **OSSL_CMP_CTX_get_certConf_cb_arg()**. Typically, the callback will check at least that the certificate can be verified using a set of trusted certificates. It also could compare the subject DN and other fields of the newly enrolled certificate with the certificate template of the request.

OSSL_CMP_CTX_set_certConf_cb_arg() sets an argument, respectively a pointer to a structure containing arguments, optionally to be used by the certConf callback. *arg* is not consumed, and it must therefore explicitly be freed when not needed any more. *arg* may be NULL to clear the entry.

OSSL_CMP_CTX_get_certConf_cb_arg() gets the argument, respectively the pointer to a structure containing arguments, previously set by **OSSL_CMP_CTX_set_certConf_cb_arg()**, or NULL if unset.

OSSL_CMP_CTX_get_status() returns for client contexts the PKIstatus from the last received CertRepMessage or Revocation Response or error message: =item **OSSL_CMP_PKISTATUS_accepted** on successful receipt of a GENP message:

OSSL_CMP_PKISTATUS_request
: if an IR/CR/KUR/RR/GENM request message could not be produced,

OSSL_CMP_PKISTATUS_trans
: on a transmission error or transaction error for this type of request, and

OSSL_CMP_PKISTATUS_unspecified
: if no such request was attempted or **OSSL_CMP_CTX_reinit()** has been called.

For server contexts it returns **OSSL_CMP_PKISTATUS_trans** if a transaction is open, otherwise **OSSL_CMP_PKISTATUS_unspecified**.

OSSL_CMP_CTX_get0_statusString() returns the statusString from the last received CertRepMessage or Revocation Response or error message, or NULL if unset.

OSSL_CMP_CTX_get_failInfoCode() returns the error code from the failInfo field of the last received CertRepMessage or Revocation Response or error message, or -1 if no such response was received or **OSSL_CMP_CTX_reinit()** has been called. This is a bit field and the flags for it are specified in the header file <*openssl/cmp.h*>. The flags start with OSSL_CMP_CTX_FAILINFO, for example:

OSSL_CMP_CTX_FAILINFO_badAlg. Returns -1 if the failInfoCode field is unset.

OSSL_CMP_CTX_get0_newCert() returns the pointer to the newly obtained certificate in case it is available, else NULL.

OSSL_CMP_CTX_get1_newChain() returns a pointer to a duplicate of the stack of X.509 certificates computed by **OSSL_CMP_certConf_cb()** (if this function has been called) on the last received certificate response message IP/CP/KUP.

OSSL_CMP_CTX_get1_caPubs() returns a pointer to a duplicate of the list of X.509 certificates in the caPubs field of the last received certificate response message (of type IP, CP, or KUP), or an empty stack if no caPubs have been received in the current transaction.

OSSL_CMP_CTX_get1_extraCertsIn() returns a pointer to a duplicate of the list of X.509 certificates contained in the extraCerts field of the last received response message (except for pollRep and PKIConf), or an empty stack if no extraCerts have been received in the current transaction.

OSSL_CMP_CTX_set1_transactionID() sets the given transaction ID in the given OSSL_CMP_CTX structure.

OSSL_CMP_CTX_set1_senderNonce() stores the last sent sender *nonce* in the *ctx*. This will be used to validate the recipNonce in incoming messages.

NOTES

CMP is defined in RFC 4210 (and CRMF in RFC 4211).

RETURN VALUES

OSSL_CMP_CTX_free() and **OSSL_CMP_CTX_print_errors()** do not return anything.

OSSL_CMP_CTX_new(), **OSSL_CMP_CTX_get_http_cb_arg()**, **OSSL_CMP_CTX_get_transfer_cb_arg()**, **OSSL_CMP_CTX_get0_trustedStore()**, **OSSL_CMP_CTX_get0_untrusted()**, **OSSL_CMP_CTX_get0_newPkey()**, **OSSL_CMP_CTX_get_certConf_cb_arg()**, **OSSL_CMP_CTX_get0_statusString()**, **OSSL_CMP_CTX_get0_newCert()**, **OSSL_CMP_CTX_get0_newChain()**, **OSSL_CMP_CTX_get1_caPubs()**, and **OSSL_CMP_CTX_get1_extraCertsIn()** return the intended pointer value as described above or NULL on error.

OSSL_CMP_CTX_get_option(), **OSSL_CMP_CTX_reqExtensions_have_SAN()**, **OSSL_CMP_CTX_get_status()**, and **OSSL_CMP_CTX_get_failInfoCode()** return the intended value as described above or -1 on error.

OSSL_CMP_certConf_cb() returns *fail_info* if it is not equal to 0, else 0 on successful validation, or else a bit field with the **OSSL_CMP_PKIFAILUREINFO_incorrectData** bit set.

All other functions, including **OSSL_CMP_CTX_reinit()** and **OSSL_CMP_CTX_reset_geninfo_ITAVs()**, return 1 on success, 0 on error.

EXAMPLES

The following code omits error handling.

Set up a CMP client context for sending requests and verifying responses:

```
cmp_ctx = OSSL_CMP_CTX_new();
OSSL_CMP_CTX_set1_server(cmp_ctx, name_or_address);
OSSL_CMP_CTX_set1_serverPort(cmp_ctx, port_string);
OSSL_CMP_CTX_set1_serverPath(cmp_ctx, path_or_alias);
OSSL_CMP_CTX_set0_trustedStore(cmp_ctx, ts);
```

Set up symmetric credentials for MAC-based message protection such as PBM:

```
OSSL_CMP_CTX_set1_referenceValue(cmp_ctx, ref, ref_len);
OSSL_CMP_CTX_set1_secretValue(cmp_ctx, sec, sec_len);
```

Set up the details for certificate requests:

```
OSSL_CMP_CTX_set1_subjectName(cmp_ctx, name);
OSSL_CMP_CTX_set0_newPkey(cmp_ctx, 1, initialKey);
```

Perform an Initialization Request transaction:

```
initialCert = OSSL_CMP_exec_IR_ses(cmp_ctx);
```

Reset the transaction state of the CMP context and the credentials:

```
OSSL_CMP_CTX_reinit(cmp_ctx);
OSSL_CMP_CTX_set1_referenceValue(cmp_ctx, NULL, 0);
OSSL_CMP_CTX_set1_secretValue(cmp_ctx, NULL, 0);
```

Perform a Certification Request transaction, making use of the new credentials:

```
OSSL_CMP_CTX_set1_cert(cmp_ctx, initialCert);
```

```
OSSL_CMP_CTX_set1_pkey(cmp_ctx, initialKey);
OSSL_CMP_CTX_set0_newPkey(cmp_ctx, 1, curentKey);
currentCert = OSSL_CMP_exec_CR_ses(cmp_ctx);
```

Perform a Key Update Request, signed using the cert (and key) to be updated:

```
OSSL_CMP_CTX_reinit(cmp_ctx);
OSSL_CMP_CTX_set1_cert(cmp_ctx, currentCert);
OSSL_CMP_CTX_set1_pkey(cmp_ctx, currentKey);
OSSL_CMP_CTX_set0_newPkey(cmp_ctx, 1, updatedKey);
currentCert = OSSL_CMP_exec_KUR_ses(cmp_ctx);
currentKey = updatedKey;
```

Perform a General Message transaction including, as an example, the id-it-signKeyPairTypes OID and prints info on the General Response contents:

```
OSSL_CMP_CTX_reinit(cmp_ctx);

ASN1_OBJECT *type = OBJ_txt2obj("1.3.6.1.5.5.7.4.2", 1);
OSSL_CMP_ITAV *itav = OSSL_CMP_ITAV_create(type, NULL);
OSSL_CMP_CTX_push0_genm_ITAV(cmp_ctx, itav);

STACK_OF(OSSL_CMP_ITAV) *itavs;
itavs = OSSL_CMP_exec_GENM_ses(cmp_ctx);
print_itavs(itavs);
sk_OSSL_CMP_ITAV_pop_free(itavs, OSSL_CMP_ITAV_free);
```

SEE ALSO

OSSL_CMP_exec_IR_ses(3), **OSSL_CMP_exec_CR_ses**(3), **OSSL_CMP_exec_KUR_ses**(3), **OSSL_CMP_exec_GENM_ses**(3), **OSSL_CMP_exec_certreq**(3), **OSSL_CMP_MSG_http_perform**(3), **ERR_print_errors_cb**(3)

HISTORY

The OpenSSL CMP support was added in OpenSSL 3.0.

OSSL_CMP_CTX_reset_geninfo_ITAVs() was added in OpenSSL 3.0.8.

COPYRIGHT