Rocky Enterprise Linux 9.2 Manual Pages on command 'EVP_MAC_CTX_settable_params.3ossl'

$ man EVP_MAC_CTX_settable_params.3ossl

EVP_MAC(3ossl) OpenSSL EVP_MAC(3ossl)

NAME

EVP_MAC, EVP_MAC_fetch, EVP_MAC_up_ref, EVP_MAC_free, EVP_MAC_is_a,

EVP_MAC_get0_name, EVP_MAC_names_do_all, EVP_MAC_get0_description,

EVP_MAC_get0_provider, EVP_MAC_get_params, EVP_MAC_gettable_params,

EVP_MAC_CTX, EVP_MAC_CTX_new, EVP_MAC_CTX_free, EVP_MAC_CTX_dup,

EVP_MAC_CTX_get0_mac, EVP_MAC_CTX_get_params, EVP_MAC_CTX_set_params,

EVP_MAC_CTX_get_mac_size, EVP_MAC_CTX_get_block_size, EVP_Q_mac,

EVP_MAC_init, EVP_MAC_update, EVP_MAC_final, EVP_MAC_finalXOF,

EVP_MAC_gettable_ctx_params, EVP_MAC_settable_ctx_params,

EVP_MAC_CTX_gettable_params, EVP_MAC_CTX_settable_params,

EVP_MAC_do_all_provided - EVP MAC routines

SYNOPSIS

#include <openssl/evp.h>

typedef struct evp_mac_st EVP_MAC;

typedef struct evp_mac_ctx_st EVP_MAC_CTX;

EVP_MAC *EVP_MAC_fetch(OSSL_LIB_CTX *libctx, const char *algorithm,

const char *properties);

```
int EVP_MAC_up_ref(EVP_MAC *mac);
void EVP_MAC_free(EVP_MAC *mac);
int EVP_MAC_is_a(const EVP_MAC *mac, const char *name);
const char *EVP_MAC_get0_name(const EVP_MAC *mac);
int EVP_MAC_names_do_all(const EVP_MAC *mac,
                         void (*fn)(const char *name, void *data),
                         void *data);
const char *EVP_MAC_get0_description(const EVP_MAC *mac);
const OSSL_PROVIDER *EVP_MAC_get0_provider(const EVP_MAC *mac);
int EVP_MAC_get_params(EVP_MAC *mac, OSSL_PARAM params[]);
EVP_MAC_CTX *EVP_MAC_CTX_new(EVP_MAC *mac);
void EVP_MAC_CTX_free(EVP_MAC_CTX *ctx);
EVP_MAC_CTX *EVP_MAC_CTX_dup(const EVP_MAC_CTX *src);
EVP_MAC *EVP_MAC_CTX_get0_mac(EVP_MAC_CTX *ctx);
int EVP_MAC_CTX_get_params(EVP_MAC_CTX *ctx, OSSL_PARAM params[]);
int EVP_MAC_CTX_set_params(EVP_MAC_CTX *ctx, const OSSL_PARAM params[]);
size_t EVP_MAC_CTX_get_mac_size(EVP_MAC_CTX *ctx);
size_t EVP_MAC_CTX_get_block_size(EVP_MAC_CTX *ctx);
unsigned char *EVP_Q_mac(OSSL_LIB_CTX *libctx, const char *name, const char *propq,
                         const char *subalg, const OSSL_PARAM *params,
                         const void *key, size_t keylen,
                         const unsigned char *data, size_t datalen,
                         unsigned char *out, size_t outsize, size_t *outlen);
int EVP_MAC_init(EVP_MAC_CTX *ctx, const unsigned char *key, size_t keylen,
                 const OSSL_PARAM params[]);
int EVP_MAC_update(EVP_MAC_CTX *ctx, const unsigned char *data, size_t datalen);
int EVP_MAC_final(EVP_MAC_CTX *ctx,
                  unsigned char *out, size_t *outl, size_t outsize);
int EVP_MAC_finalXOF(EVP_MAC_CTX *ctx, unsigned char *out, size_t outsize);
const OSSL_PARAM *EVP_MAC_gettable_params(const EVP_MAC *mac);
const OSSL_PARAM *EVP_MAC_gettable_ctx_params(const EVP_MAC *mac);
const OSSL_PARAM *EVP_MAC_settable_ctx_params(const EVP_MAC *mac);
const OSSL_PARAM *EVP_MAC_CTX_gettable_params(EVP_MAC_CTX *ctx);
```

```
const OSSL_PARAM *EVP_MAC_CTX_settable_params(EVP_MAC_CTX *ctx);
void EVP_MAC_do_all_provided(OSSL_LIB_CTX *libctx,
                             void (*fn)(EVP_MAC *mac, void *arg),
                             void *arg);
```

DESCRIPTION

These types and functions help the application to calculate MACs of different types and with different underlying algorithms if there are any.

MACs are a bit complex insofar that some of them use other algorithms for actual computation. HMAC uses a digest, and CMAC uses a cipher. Therefore, there are sometimes two contexts to keep track of, one for the MAC algorithm itself and one for the underlying computation algorithm if there is one.

To make things less ambiguous, this manual talks about a "context" or "MAC context", which is to denote the MAC level context, and about a "underlying context", or "computation context", which is to denote the context for the underlying computation algorithm if there is one.

Types

EVP_MAC is a type that holds the implementation of a MAC.

EVP_MAC_CTX is a context type that holds internal MAC information as well as a reference to a computation context, for those MACs that rely on an underlying computation algorithm.

Algorithm implementation fetching

EVP_MAC_fetch() fetches an implementation of a MAC algorithm, given a library context libctx and a set of properties. See "ALGORITHM FETCHING" in crypto(7) for further information.

See "Message Authentication Code (MAC)" in OSSL_PROVIDER-default(7) for the list of algorithms supported by the default provider.

The returned value must eventually be freed with EVP_MAC_free(3).

EVP_MAC_up_ref() increments the reference count of an already fetched MAC.

EVP_MAC_free() frees a fetched algorithm. NULL is a valid parameter, for which this function is a no-op.

Context manipulation functions

EVP_MAC_CTX_new() creates a new context for the MAC type mac. The created context can then be used with most other functions described here.

EVP_MAC_CTX_free() frees the contents of the context, including an underlying context if there is one, as well as the context itself. NULL is a valid parameter, for which this function is a no-op.

EVP_MAC_CTX_dup() duplicates the src context and returns a newly allocated context.

EVP_MAC_CTX_get0_mac() returns the EVP_MAC associated with the context ctx.

Computing functions

EVP_Q_mac() computes the message authentication code of data with length datalen using the MAC algorithm name and the key key with length keylen. The MAC algorithm is fetched using any given libctx and property query string propq. It takes parameters subalg and further params, both of which may be NULL if not needed. If out is not NULL, it places the result in the memory pointed at by out, but only if outsize is sufficient (otherwise no computation is made). If out is NULL, it allocates and uses a buffer of suitable length, which will be returned on success and must be freed by the caller. In either case, also on error, it assigns the number of bytes written to *outlen unless outlen is NULL.

EVP_MAC_init() sets up the underlying context ctx with information given via the key and params arguments. The MAC key has a length of keylen and the parameters in params are processed before setting the key. If key is NULL, the key must be set via params either as part of this call or separately using EVP_MAC_CTX_set_params(). Providing non-NULL params to this function is equivalent to calling EVP_MAC_CTX_set_params() with those params for the same ctx beforehand. EVP_MAC_init() should be called before EVP_MAC_update() and EVP_MAC_final().

EVP_MAC_update() adds datalen bytes from data to the MAC input.

EVP_MAC_final() does the final computation and stores the result in the memory pointed at by out of size outsize, and sets the number of bytes written in *outl at. If out is NULL or outsize is too small, then no computation is made. To figure out what the output length will be and allocate space for it dynamically, simply call with out being NULL and outl pointing at a valid location, then allocate space and make a second call with out pointing at the allocated space.

EVP_MAC_finalXOF() does the final computation for an XOF based MAC and stores the result in the memory pointed at by out of size outsize.

EVP_MAC_get_params() retrieves details about the implementation mac. The set of parameters given with params determine exactly what parameters should be retrieved. Note that a parameter that is unknown in the underlying context is simply ignored.

EVP_MAC_CTX_get_params() retrieves chosen parameters, given the context ctx and its underlying context. The set of parameters given with params determine exactly what parameters should be retrieved. Note that a parameter that is unknown in the underlying context is simply ignored.

EVP_MAC_CTX_set_params() passes chosen parameters to the underlying context, given a context ctx. The set of parameters given with params determine exactly what parameters are passed down. If params are NULL, the unterlying context should do nothing and return 1. Note that a parameter that is unknown in the underlying context is simply ignored. Also, what happens when a needed parameter isn't passed down is defined by the implementation.

EVP_MAC_gettable_params() returns an OSSL_PARAM array that describes the retrievable and settable parameters. EVP_MAC_gettable_params() returns parameters that can be used with EVP_MAC_get_params(). See OSSL_PARAM(3) for the use of OSSL_PARAM as a parameter descriptor.

EVP_MAC_gettable_ctx_params() and EVP_MAC_CTX_gettable_params() return constant OSSL_PARAM arrays that describe the retrievable parameters that can be used with EVP_MAC_CTX_get_params().

EVP_MAC_gettable_ctx_params() returns the parameters that can be

retrieved from the algorithm, whereas EVP_MAC_CTX_gettable_params() returns the parameters that can be retrieved in the context's current state. See OSSL_PARAM(3) for the use of OSSL_PARAM as a parameter descriptor.

EVP_MAC_settable_ctx_params() and EVP_MAC_CTX_settable_params() return constant OSSL_PARAM arrays that describe the settable parameters that can be used with EVP_MAC_CTX_set_params().

EVP_MAC_settable_ctx_params() returns the parameters that can be retrieved from the algorithm, whereas EVP_MAC_CTX_settable_params() returns the parameters that can be retrieved in the context's current state. See OSSL_PARAM(3) for the use of OSSL_PARAM as a parameter descriptor.

Information functions

EVP_MAC_CTX_get_mac_size() returns the MAC output size for the given context.

EVP_MAC_CTX_get_block_size() returns the MAC block size for the given context. Not all MAC algorithms support this.

EVP_MAC_is_a() checks if the given mac is an implementation of an algorithm that's identifiable with name.

EVP_MAC_get0_provider() returns the provider that holds the implementation of the given mac.

EVP_MAC_do_all_provided() traverses all MAC implemented by all activated providers in the given library context libctx, and for each of the implementations, calls the given function fn with the implementation method and the given arg as argument.

EVP_MAC_get0_name() return the name of the given MAC. For fetched MACs with multiple names, only one of them is returned; it's recommended to use EVP_MAC_names_do_all() instead.

EVP_MAC_names_do_all() traverses all names for mac, and calls fn with each name and data.

EVP_MAC_get0_description() returns a description of the mac, meant for display and human consumption. The description is at the discretion of the mac implementation.

PARAMETERS

Parameters are identified by name as strings, and have an expected data type and maximum size. OpenSSL has a set of macros for parameter names it expects to see in its own MAC implementations. Here, we show all three, the OpenSSL macro for the parameter name, the name in string form, and a type description.

The standard parameter names are:

"key" (OSSL_MAC_PARAM_KEY) <octet string>

Its value is the MAC key as an array of bytes.

For MACs that use an underlying computation algorithm, the algorithm must be set first, see parameter names "algorithm" below.

"iv" (OSSL_MAC_PARAM_IV) <octet string>

Some MAC implementations (GMAC) require an IV, this parameter sets the IV.

"custom" (OSSL_MAC_PARAM_CUSTOM) <octet string>

Some MAC implementations (KMAC, BLAKE2) accept a Customization String, this parameter sets the Customization String. The default value is the empty string.

"salt" (OSSL_MAC_PARAM_SALT) <octet string>

This option is used by BLAKE2 MAC.

"xof" (OSSL_MAC_PARAM_XOF) <integer>

It's a simple flag, the value 0 or 1 are expected.

This option is used by KMAC.

"digest-noinit" (OSSL_MAC_PARAM_DIGEST_NOINIT) <integer>

A simple flag to set the MAC digest to not initialise the implementation specific data. The value 0 or 1 is expected.

This option is used by HMAC.

"digest-oneshot" (OSSL_MAC_PARAM_DIGEST_ONESHOT) <integer>

A simple flag to set the MAC digest to be a oneshot operation. The value 0 or 1 is expected.

This option is used by HMAC.

"properties" (OSSL_MAC_PARAM_PROPERTIES) <UTF8 string>

"digest" (OSSL_MAC_PARAM_DIGEST) <UTF8 string>

"cipher" (OSSL_MAC_PARAM_CIPHER) <UTF8 string>

For MAC implementations that use an underlying computation cipher or digest, these parameters set what the algorithm should be.
The value is always the name of the intended algorithm, or the properties.
Note that not all algorithms may support all digests. HMAC does not support variable output length digests such as SHAKE128 or SHAKE256.

"size" (OSSL_MAC_PARAM_SIZE) <unsigned integer>

For MAC implementations that support it, set the output size that EVP_MAC_final() should produce. The allowed sizes vary between MAC implementations, but must never exceed what can be given with a size_t.

"tls-data-size" (OSSL_MAC_PARAM_TLS_DATA_SIZE) <unsigned integer>

This parameter is only supported by HMAC. If set then special handling is activated for calculating the MAC of a received mac-then-encrypt TLS record where variable length record padding has been used (as in the case of CBC mode ciphersuites). The value represents the total length of the record that is having the MAC calculated including the received MAC and the record padding.
When used EVP_MAC_update must be called precisely twice. The first time with the 13 bytes of TLS "header" data, and the second time with the entire record including the MAC itself and any padding.
The entire record length must equal the value passed in the "tls-data-size" parameter. The length passed in the datalen parameter to EVP_MAC_update() should be equal to the length of the record after the MAC and any padding has been removed.

All these parameters should be used before the calls to any of EVP_MAC_init(), EVP_MAC_update() and EVP_MAC_final() for a full computation. Anything else may give undefined results.

NOTES

The MAC life-cycle is described in life_cycle-mac(7). In the future, the transitions described there will be enforced. When this is done,

it will not be considered a breaking change to the API.

The usage of the parameter names "custom", "iv" and "salt" correspond to the names used in the standard where the algorithm was defined.

RETURN VALUES

EVP_MAC_fetch() returns a pointer to a newly fetched EVP_MAC, or NULL if allocation failed.

EVP_MAC_up_ref() returns 1 on success, 0 on error.

EVP_MAC_names_do_all() returns 1 if the callback was called for all names. A return value of 0 means that the callback was not called for any names.

EVP_MAC_free() returns nothing at all.

EVP_MAC_is_a() returns 1 if the given method can be identified with the given name, otherwise 0.

EVP_MAC_get0_name() returns a name of the MAC, or NULL on error.

EVP_MAC_get0_provider() returns a pointer to the provider for the MAC, or NULL on error.

EVP_MAC_CTX_new() and EVP_MAC_CTX_dup() return a pointer to a newly created EVP_MAC_CTX, or NULL if allocation failed.

EVP_MAC_CTX_free() returns nothing at all.

EVP_MAC_CTX_get_params() and EVP_MAC_CTX_set_params() return 1 on success, 0 on error.

EVP_Q_mac() returns a pointer to the computed MAC value, or NULL on error.

EVP_MAC_init(), EVP_MAC_update(), EVP_MAC_final(), and EVP_MAC_finalXOF() return 1 on success, 0 on error.

EVP_MAC_CTX_get_mac_size() returns the expected output size, or 0 if it isn't set. If it isn't set, a call to EVP_MAC_init() will set it.

EVP_MAC_CTX_get_block_size() returns the block size, or 0 if it isn't set. If it isn't set, a call to EVP_MAC_init() will set it.

EVP_MAC_do_all_provided() returns nothing at all.

EXAMPLES

```
#include <stdlib.h>
#include <stdio.h>
```

```
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/params.h>
int main() {
    EVP_MAC *mac = EVP_MAC_fetch(NULL, getenv("MY_MAC"), NULL);
    const char *cipher = getenv("MY_MAC_CIPHER");
    const char *digest = getenv("MY_MAC_DIGEST");
    const char *key = getenv("MY_KEY");
    EVP_MAC_CTX *ctx = NULL;
    unsigned char buf[4096];
    size_t read_l;
    size_t final_l;
    size_t i;
    OSSL_PARAM params[3];
    size_t params_n = 0;
    if (cipher != NULL)
        params[params_n++] =
            OSSL_PARAM_construct_utf8_string("cipher", (char*)cipher, 0);
    if (digest != NULL)
        params[params_n++] =
            OSSL_PARAM_construct_utf8_string("digest", (char*)digest, 0);
    params[params_n] = OSSL_PARAM_construct_end();
    if (mac == NULL
        || key == NULL
        || (ctx = EVP_MAC_CTX_new(mac)) == NULL
        || !EVP_MAC_init(ctx, (const unsigned char *)key, strlen(key),
                         params))
        goto err;
    while ( (read_l = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
        if (!EVP_MAC_update(ctx, buf, read_l))
```

```
            goto err;
        }
        if (!EVP_MAC_final(ctx, buf, &final_l, sizeof(buf)))
            goto err;
        printf("Result: ");
        for (i = 0; i < final_l; i++)
            printf("%02X", buf[i]);
        printf("\n");
        EVP_MAC_CTX_free(ctx);
        EVP_MAC_free(mac);
        exit(0);
     err:
        EVP_MAC_CTX_free(ctx);
        EVP_MAC_free(mac);
        fprintf(stderr, "Something went wrong\n");
        ERR_print_errors_fp(stderr);
        exit (1);
    }
```

A run of this program, called with correct environment variables, can look like this:

```
 $ MY_MAC=cmac MY_KEY=secret0123456789 MY_MAC_CIPHER=aes-128-cbc \
   LD_LIBRARY_PATH=. ./foo < foo.c
 Result: C5C06683CD9DDEF904D754505C560A4E
```

(in this example, that program was stored in foo.c and compiled to ./foo)

SEE ALSO

property(7) OSSL_PARAM(3), EVP_MAC-BLAKE2(7), EVP_MAC-CMAC(7), EVP_MAC-GMAC(7), EVP_MAC-HMAC(7), EVP_MAC-KMAC(7), EVP_MAC-Siphash(7), EVP_MAC-Poly1305(7), provider-mac(7), life_cycle-mac(7)

HISTORY

These functions were added in OpenSSL 3.0.

COPYRIGHT