



Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'EVP_DigestSign.3oss1'

\$ man EVP_DigestSign.3oss1

EVP_DIGESTSIGNINIT(3oss1) OpenSSL EVP_DIGESTSIGNINIT(3oss1)

NAME

EVP_DigestSignInit_ex, EVP_DigestSignInit, EVP_DigestSignUpdate,
EVP_DigestSignFinal, EVP_DigestSign - EVP signing functions

SYNOPSIS

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

```
int EVP_DigestSignInit_ex(EVP_MD_CTX *ctx, EVP_PKEY_CTX **pctx,  
                        const char *mdname, OSSL_LIB_CTX *libctx,  
                        const char *props, EVP_PKEY *pkey,  
                        const OSSL_PARAM params[]);
```

```
int EVP_DigestSignInit(EVP_MD_CTX *ctx, EVP_PKEY_CTX **pctx,  
                      const EVP_MD *type, ENGINE *e, EVP_PKEY *pkey);
```

```
int EVP_DigestSignUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);
```

```
int EVP_DigestSignFinal(EVP_MD_CTX *ctx, unsigned char *sig, size_t *siglen);
```

```
int EVP_DigestSign(EVP_MD_CTX *ctx, unsigned char *sigret,  
                  size_t *siglen, const unsigned char *tbs,  
                  size_t tbslen);
```

DESCRIPTION

The EVP signature routines are a high-level interface to digital signatures. Input data is digested first before the signing takes place.

`EVP_DigestSignInit_ex()` sets up signing context `ctx` to use a digest with the name `mdname` and private key `pkey`. The name of the digest to be used is passed to the provider of the signature algorithm in use. How that provider interprets the digest name is provider specific. The provider may implement that digest directly itself or it may (optionally) choose to fetch it (which could result in a digest from a different provider being selected). If the provider supports fetching the digest then it may use the `props` argument for the properties to be used during the fetch. Finally, the passed parameters `params`, if not `NULL`, are set on the context before returning.

The `pkey` algorithm is used to fetch a `EVP_SIGNATURE` method implicitly, to be used for the actual signing. See "Implicit fetch" in `provider(7)` for more information about implicit fetches.

The OpenSSL default and legacy providers support fetching digests and can fetch those digests from any available provider. The OpenSSL FIPS provider also supports fetching digests but will only fetch digests that are themselves implemented inside the FIPS provider.

`ctx` must be created with `EVP_MD_CTX_new()` before calling this function.

If `pctx` is not `NULL`, the `EVP_PKEY_CTX` of the signing operation will be written to `*pctx`: this can be used to set alternative signing options.

Note that any existing value in `*pctx` is overwritten. The `EVP_PKEY_CTX`

value returned must not be freed directly by the application if ctx is not assigned an EVP_PKEY_CTX value before being passed to EVP_DigestSignInit_ex() (which means the EVP_PKEY_CTX is created inside EVP_DigestSignInit_ex() and it will be freed automatically when the EVP_MD_CTX is freed). If the EVP_PKEY_CTX to be used is created by EVP_DigestSignInit_ex then it will use the OSSL_LIB_CTX specified in libctx and the property query string specified in props.

The digest mdname may be NULL if the signing algorithm supports it. The props argument can always be NULL.

No EVP_PKEY_CTX will be created by EVP_DigestSignInit_ex() if the passed ctx has already been assigned one via EVP_MD_CTX_set_pkey_ctx(3). See also SM2(7).

Only EVP_PKEY types that support signing can be used with these functions. This includes MAC algorithms where the MAC generation is considered as a form of "signing". Built-in EVP_PKEY types supported by these functions are CMAC, Poly1305, DSA, ECDSA, HMAC, RSA, SipHash, Ed25519 and Ed448.

Not all digests can be used for all key types. The following combinations apply.

DSA Supports SHA1, SHA224, SHA256, SHA384 and SHA512

ECDSA

Supports SHA1, SHA224, SHA256, SHA384, SHA512 and SM3

RSA with no padding

Supports no digests (the digest type must be NULL)

RSA with X931 padding

Supports SHA1, SHA256, SHA384 and SHA512

All other RSA padding types

Support SHA1, SHA224, SHA256, SHA384, SHA512, MD5, MD5_SHA1, MD2, MD4, MDC2, SHA3-224, SHA3-256, SHA3-384, SHA3-512

Ed25519 and Ed448

Support no digests (the digest type must be NULL)

HMAC

Supports any digest

CMAC, Poly1305 and SipHash

Will ignore any digest provided.

If RSA-PSS is used and restrictions apply then the digest must match.

`EVP_DigestSignInit()` works in the same way as `EVP_DigestSignInit_ex()` except that the `mdname` parameter will be inferred from the supplied digest type, and `props` will be NULL. Where supplied the ENGINE `e` will be used for the signing and digest algorithm implementations. `e` may be NULL.

`EVP_DigestSignUpdate()` hashes `cnt` bytes of data at `d` into the signature context `ctx`. This function can be called several times on the same `ctx` to include additional data.

Unless `sig` is NULL `EVP_DigestSignFinal()` signs the data in `ctx` and places the signature in `sig`. Otherwise the maximum necessary size of the output buffer is written to the `siglen` parameter. If `sig` is not NULL then before the call the `siglen` parameter should contain the length of the sig buffer. If the call is successful the signature is written to `sig` and the amount of data written to `siglen`.

EVP_DigestSign() signs tbslen bytes of data at tbs and places the signature in sig and its length in siglen in a similar way to EVP_DigestSignFinal(). In the event of a failure EVP_DigestSign() cannot be called again without reinitialising the EVP_MD_CTX. If sig is NULL before the call then siglen will be populated with the required size for the sig buffer. If sig is non-NULL before the call then siglen should contain the length of the sig buffer.

RETURN VALUES

EVP_DigestSignInit(), EVP_DigestSignUpdate(), EVP_DigestSignFinal() and EVP_DigestSign() return 1 for success and 0 for failure.

The error codes can be obtained from ERR_get_error(3).

NOTES

The EVP interface to digital signatures should almost always be used in preference to the low-level interfaces. This is because the code then becomes transparent to the algorithm used and much more flexible.

EVP_DigestSign() is a one shot operation which signs a single block of data in one function. For algorithms that support streaming it is equivalent to calling EVP_DigestSignUpdate() and EVP_DigestSignFinal(). For algorithms which do not support streaming (e.g. PureEdDSA) it is the only way to sign data.

In previous versions of OpenSSL there was a link between message digest types and public key algorithms. This meant that "clone" digests such as EVP_dss1() needed to be used to sign using SHA1 and DSA. This is no longer necessary and the use of clone digest is now discouraged.

For some key types and parameters the random number generator must be seeded. If the automatic seeding or reseeding of the OpenSSL CSPRNG

fails due to external circumstances (see RAND(7)), the operation will fail.

The call to EVP_DigestSignFinal() internally finalizes a copy of the digest context. This means that calls to EVP_DigestSignUpdate() and EVP_DigestSignFinal() can be called later to digest and sign additional data.

EVP_DigestSignInit() and EVP_DigestSignInit_ex() functions can be called multiple times on a context and the parameters set by previous calls should be preserved if the pkey parameter is NULL. The call then just resets the state of the ctx.

The use of EVP_PKEY_get_size() with these functions is discouraged because some signature operations may have a signature length which depends on the parameters set. As a result EVP_PKEY_get_size() would have to return a value which indicates the maximum possible signature for any set of parameters.

SEE ALSO

EVP_DigestVerifyInit(3), EVP_DigestInit(3), evp(7), HMAC(3), MD2(3), MD5(3), MDC2(3), RIPEMD160(3), SHA1(3), openssl-dgst(1), RAND(7)

HISTORY

EVP_DigestSignInit(), EVP_DigestSignUpdate() and EVP_DigestSignFinal() were added in OpenSSL 1.0.0.

EVP_DigestSignInit_ex() was added in OpenSSL 3.0.

EVP_DigestSignUpdate() was converted from a macro to a function in OpenSSL 3.0.

Copyright 2006-2022 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.

3.0.7 2023-07-13 EVP_DIGESTSIGNINIT(3ossl)