NAME

EVP_DigestVerifyInit_ex, EVP_DigestVerifyInit, EVP_DigestVerifyUpdate, EVP_DigestVerifyFinal, EVP_DigestVerify - EVP signature verification functions

SYNOPSIS

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

int EVP_DigestVerifyInit_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_DigestVerifyInit(EVP_MD_CTX *ctx, EVP_PKEY_CTX **pctx,
                         const EVP_MD *type, ENGINE *e, EVP_PKEY *pkey);
int EVP_DigestVerifyUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);
int EVP_DigestVerifyFinal(EVP_MD_CTX *ctx, const unsigned char *sig,
                          size_t siglen);
int EVP_DigestVerify(EVP_MD_CTX *ctx, const 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 signature verification takes place.

EVP_DigestVerifyInit_ex() sets up verification context **ctx** to use a digest with the name **mdname** and public 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 verification operation will be written to ***pctx**: this can be used to set

alternative verification 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_DigestVerifyInit_ex()** (which means the EVP_PKEY_CTX is created inside **EVP_DigestVerifyInit_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_DigestVerifyInit_ex then it will use the **OSSL_LIB_CTX** specified in *libctx* and the property query string specified in *props*.

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

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_DigestVerifyInit() works in the same way as **EVP_DigestVerifyInit_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 signature verification and digest algorithm implementations. **e** may be NULL.

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

EVP_DigestVerifyFinal() verifies the data in **ctx** against the signature in **sig** of length **siglen**.

EVP_DigestVerify() verifies **tbslen** bytes at **tbs** against the signature in **sig** of length **siglen**.

RETURN VALUES

EVP_DigestVerifyInit() and **EVP_DigestVerifyUpdate()** return 1 for success and 0 for failure.

EVP_DigestVerifyFinal() and **EVP_DigestVerify()** return 1 for success; any other value indicates failure. A return value of zero indicates that the signature did not verify successfully (that is, **tbs** did not match the original data or the signature had an invalid form), while other values indicate a more serious error (and sometimes also indicate an invalid signature form).

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_DigestVerify() is a one shot operation which verifies a single block of data in one function. For algorithms that support streaming it is equivalent to calling **EVP_DigestVerifyUpdate()** and **EVP_DigestVerifyFinal()**. For algorithms which do not support streaming (e.g. PureEdDSA) it is the only way to verify 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_DigestVerifyFinal()** internally finalizes a copy of the digest context. This means that **EVP_VerifyUpdate()** and **EVP_VerifyFinal()** can be called later to digest and verify additional data.

EVP_DigestVerifyInit() and **EVP_DigestVerifyInit_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*.

Ignoring failure returns of **EVP_DigestVerifyInit()** and **EVP_DigestVerifyInit_ex()** functions can lead to subsequent undefined behavior when calling **EVP_DigestVerifyUpdate()**, **EVP_DigestVerifyFinal()**, or **EVP_DigestVerify()**.

SEE ALSO

EVP_DigestSignInit(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_DigestVerifyInit(), **EVP_DigestVerifyUpdate()** and **EVP_DigestVerifyFinal()** were added in OpenSSL 1.0.0.

EVP_DigestVerifyInit_ex() was added in OpenSSL 3.0.

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

COPYRIGHT