

**NAME**

`EVP_PKEY_sign_init`, `EVP_PKEY_sign` - sign using a public key algorithm

**SYNOPSIS**

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

int EVP_PKEY_sign_init(EVP_PKEY_CTX *ctx);
int EVP_PKEY_sign(EVP_PKEY_CTX *ctx,
                  unsigned char *sig, size_t *siglen,
                  const unsigned char *tbs, size_t tbslen);
```

**DESCRIPTION**

The `EVP_PKEY_sign_init()` function initializes a public key algorithm context using key `pkey` for a signing operation.

The `EVP_PKEY_sign()` function performs a public key signing operation using `ctx`. The data to be signed is specified using the `tbs` and `tbslen` parameters. If `sig` is `NULL` then the maximum 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`.

**NOTES**

`EVP_PKEY_sign()` does not hash the data to be signed, and therefore is normally used to sign digests. For signing arbitrary messages, see the [`EVP\_DigestSignInit\(3\)`](#) and [`EVP\_SignInit\(3\)`](#) signing interfaces instead.

After the call to `EVP_PKEY_sign_init()` algorithm specific control operations can be performed to set any appropriate parameters for the operation (see [`EVP\_PKEY\_CTX\_ctrl\(3\)`](#)).

The function `EVP_PKEY_sign()` can be called more than once on the same context if several operations are performed using the same parameters.

**RETURN VALUES**

`EVP_PKEY_sign_init()` and `EVP_PKEY_sign()` return 1 for success and 0 or a negative value for failure. In particular a return value of -2 indicates the operation is not supported by the public key algorithm.

**EXAMPLE**

Sign data using RSA with PKCS#1 padding and SHA256 digest:

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

EVP_PKEY_CTX *ctx;
/* md is a SHA-256 digest in this example. */
unsigned char *md, *sig;
size_t mrlen = 32, siglen;
EVP_PKEY *signing_key;

/*
 * NB: assumes signing_key and md are set up before the next
 * step. signing_key must be an RSA private key and md must
 * point to the SHA-256 digest to be signed.
 */
ctx = EVP_PKEY_CTX_new(signing_key, NULL /* no engine */);
if (!ctx)
/* Error occurred */
if (EVP_PKEY_sign_init(ctx) <= 0)
/* Error */
if (EVP_PKEY_CTX_set_rsa_padding(ctx, RSA_PKCS1_PADDING) <= 0)
```

```
/* Error */
if (EVP_PKEY_CTX_set_signature_md(ctx, EVP_sha256()) <= 0)
/* Error */

/* Determine buffer length */
if (EVP_PKEY_sign(ctx, NULL, &siglen, md, mdlen) <= 0)
/* Error */

sig = OPENSSL_malloc(siglen);

if (!sig)
/* malloc failure */

if (EVP_PKEY_sign(ctx, sig, &siglen, md, mdlen) <= 0)
/* Error */

/* Signature is siglen bytes written to buffer sig */
```

**SEE ALSO**

[EVP\\_PKEY\\_CTX\\_new\(3\)](#), [EVP\\_PKEY\\_CTX\\_ctrl\(3\)](#), [EVP\\_PKEY\\_encrypt\(3\)](#),  
[EVP\\_PKEY\\_decrypt\(3\)](#), [EVP\\_PKEY\\_verify\(3\)](#), [EVP\\_PKEY\\_verify\\_recover\(3\)](#),  
[EVP\\_PKEY\\_derive\(3\)](#)

**HISTORY**

These functions were first added to OpenSSL 1.0.0.