

NAME

CMS_verify, CMS_get0_signers - verify a CMS SignedData structure

SYNOPSIS

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

```
int CMS_verify(CMS_ContentInfo *cms, STACK_OF(X509) *certs, X509_STORE *store,  
               BIO *indata, BIO *out, unsigned int flags);
```

```
STACK_OF(X509) *CMS_get0_signers(CMS_ContentInfo *cms);
```

DESCRIPTION

CMS_verify() is very similar to **PKCS7_verify(3)**. It verifies a **CMS SignedData** structure contained in a structure of type **CMS_ContentInfo**. *cms* points to the **CMS_ContentInfo** structure to verify. The optional *certs* parameter refers to a set of certificates in which to search for signing certificates. *cms* may contain extra untrusted CA certificates that may be used for chain building as well as CRLs that may be used for certificate validation. *store* may be NULL or point to the trusted certificate store to use for chain verification. *indata* refers to the signed data if the content is detached from *cms*. Otherwise *indata* should be NULL and the signed data must be in *cms*. The content is written to the BIO *out* unless it is NULL. *flags* is an optional set of flags, which can be used to modify the operation.

CMS_get0_signers() retrieves the signing certificate(s) from *cms*, it may only be called after a successful **CMS_verify()** operation.

VERIFY PROCESS

Normally the verify process proceeds as follows.

Initially some sanity checks are performed on *cms*. The type of *cms* must be SignedData. There must be at least one signature on the data and if the content is detached *indata* cannot be NULL.

An attempt is made to locate all the signing certificate(s), first looking in the *certs* parameter (if it is not NULL) and then looking in any certificates contained in the *cms* structure unless **CMS_NOINTERN** is set. If any signing certificate cannot be located the operation fails.

Each signing certificate is chain verified using the *smimesign* purpose and using the trusted certificate store *store* if supplied. Any internal certificates in the message, which may have been added using **CMS_add1_cert(3)**, are used as untrusted CAs. If CRL checking is enabled in *store* and **CMS_NOCRL** is not set, any internal CRLs, which may have been added using **CMS_add1_crl(3)**, are used in addition to attempting to look them up in *store*. If *store* is not NULL and any chain verify fails an error code is returned.

Finally the signed content is read (and written to *out* unless it is NULL) and the signature is checked.

If all signatures verify correctly then the function is successful.

Any of the following flags (ored together) can be passed in the *flags* parameter to change the default verify behaviour.

If **CMS_NOINTERN** is set the certificates in the message itself are not searched when locating the signing certificate(s). This means that all the signing certificates must be in the *certs* parameter.

If **CMS_NOCRL** is set and CRL checking is enabled in *store* then any CRLs in the message itself are ignored.

If the **CMS_TEXT** flag is set MIME headers for type **text/plain** are deleted from the content. If the content is not of type **text/plain** then an error is returned.

If **CMS_NO_SIGNER_CERT_VERIFY** is set the signing certificates are not chain verified, unless **CMS_CADES** flag is also set.

If **CMS_NO_ATTR_VERIFY** is set the signed attributes signature is not verified, unless **CMS_CADES** flag is also set.

If **CMS_CADES** is set, each signer certificate is checked against the ESS signingCertificate or ESS signingCertificateV2 extension that is required in the signed attributes of the signature.

If **CMS_NO_CONTENT_VERIFY** is set then the content digest is not checked.

NOTES

One application of **CMS_NOINTERN** is to only accept messages signed by a small number of certificates. The acceptable certificates would be passed in the *certs* parameter. In this case if the signer certificate is not one of the certificates supplied in *certs* then the verify will fail because the signer cannot be found.

In some cases the standard techniques for looking up and validating certificates are not appropriate: for example an application may wish to lookup certificates in a database or perform customised verification. This can be achieved by setting and verifying the signer certificates manually using the signed data utility functions.

Care should be taken when modifying the default verify behaviour, for example setting **CMS_NO_CONTENT_VERIFY** will totally disable all content verification and any modified content

will be considered valid. This combination is however useful if one merely wishes to write the content to *out* and its validity is not considered important.

Chain verification should arguably be performed using the signing time rather than the current time. However, since the signing time is supplied by the signer it cannot be trusted without additional evidence (such as a trusted timestamp).

RETURN VALUES

CMS_verify() returns 1 for a successful verification and 0 if an error occurred.

CMS_get0_signers() returns all signers or NULL if an error occurred.

The error can be obtained from **ERR_get_error(3)**

BUGS

The trusted certificate store is not searched for the signing certificate. This is primarily due to the inadequacies of the current **X509_STORE** functionality.

The lack of single pass processing means that the signed content must all be held in memory if it is not detached.

SEE ALSO

PKCS7_verify(3), **CMS_add1_cert(3)**, **CMS_add1_crl(3)**, **OSSL_ESS_check_signing_certs(3)**, **ERR_get_error(3)**, **CMS_sign(3)**

COPYRIGHT

Copyright 2008-2022 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>.