NAME

X509_build_chain, X509_verify_cert, X509_STORE_CTX_verify - build and verify X509 certificate chain

SYNOPSIS

```
#include <openssl/x509_vfy.h>

STACK_OF(X509) *X509_build_chain(X509 *target, STACK_OF(X509) *certs,
                                 X509_STORE *store, int with_self_signed,
                                 OSSL_LIB_CTX *libctx, const char *propq);
int X509_verify_cert(X509_STORE_CTX *ctx);
int X509_STORE_CTX_verify(X509_STORE_CTX *ctx);
```

DESCRIPTION

X509_build_chain() builds a certificate chain starting from *target* using the optional list of intermediate CA certificates *certs*. If *store* is NULL it builds the chain as far down as possible, ignoring errors. Else the chain must reach a trust anchor contained in *store*. It internally uses a **X509_STORE_CTX** structure associated with the library context *libctx* and property query string *propq*, both of which may be NULL. In case there is more than one possibility for the chain, only one is taken.

On success it returns a pointer to a new stack of (up_ref'ed) certificates starting with *target* and followed by all available intermediate certificates. A self-signed trust anchor is included only if *target* is the trust anchor of *with_self_signed* is 1. If a non-NULL stack is returned the caller is responsible for freeing it.

The **X509_verify_cert()** function attempts to discover and validate a certificate chain based on parameters in *ctx*. The verification context, of type **X509_STORE_CTX**, can be constructed using **X509_STORE_CTX_new**(3) and **X509_STORE_CTX_init**(3). It usually includes a target certificate to be verified, a set of certificates serving as trust anchors, a list of non-trusted certificates that may be helpful for chain construction, flags such as X509_V_FLAG_X509_STRICT, and various other optional components such as a callback function that allows customizing the verification outcome. A complete description of the certificate verification process is contained in the **openssl-verification-options**(1) manual page.

Applications rarely call this function directly but it is used by OpenSSL internally for certificate validation, in both the S/MIME and SSL/TLS code.

A negative return value from **X509_verify_cert()** can occur if it is invoked incorrectly, such as with no certificate set in *ctx*, or when it is called twice in succession without reinitialising *ctx* for the second call. A negative return value can also happen due to internal resource problems or because an internal inconsistency has been detected. Applications must interpret any return value <= 0 as an error.

The **X509_STORE_CTX_verify()** behaves like **X509_verify_cert()** except that its target certificate is the first element of the list of untrusted certificates in *ctx* unless a target certificate is set explicitly.

RETURN VALUES

X509_build_chain() returns NULL on error, else a stack of certificates.

Both **X509_verify_cert()** and **X509_STORE_CTX_verify()** return 1 if a complete chain can be built and validated, otherwise they return 0, and in exceptional circumstances (such as malloc failure and internal errors) they can also return a negative code.

If a complete chain can be built and validated both functions return 1. If the certificate must be rejected on the basis of the data available or any required certificate status data is not available they return 0. If no definite answer possible they usually return a negative code.

On error or failure additional error information can be obtained by examining *ctx* using, for example, **X509_STORE_CTX_get_error**(3). Even if verification indicated success, the stored error code may be different from X509_V_OK, likely because a verification callback function has waived the error.

SEE ALSO

X509_STORE_CTX_new(3), **X509_STORE_CTX_init**(3), **X509_STORE_CTX_get_error**(3)

HISTORY

X509_build_chain() and **X509_STORE_CTX_verify()** were added in OpenSSL 3.0.

COPYRIGHT