

**NAME**

openssl-verify, verify - Utility to verify certificates

**SYNOPSIS**

```
openssl verify [-help] [-CAfile file] [-CApath directory] [-no-CAfile] [-no-CApath]
[-allow_proxy_certs] [-atime timestamp] [-check_ss_sig] [-CRLfile file] [-crl_download] [-crl_check]
[-crl_check_all] [-engine id] [-explicit_policy] [-extended_crl] [-ignore_critical] [-inhibit_any]
[-inhibit_map] [-no_check_time] [-partial_chain] [-policy_arg] [-policy_check] [-policy_print]
[-purpose purpose] [-suiteB_128] [-suiteB_128_only] [-suiteB_192] [-trusted_first] [-no_alt_chains]
[-untrusted file] [-trusted file] [-use_deltas] [-verbose] [-auth_level level] [-verify_depth num]
[-verify_email email] [-verify_hostname hostname] [-verify_ip ip] [-verify_name name] [-x509_strict]
[-show_chain] [-] [certificates]
```

**DESCRIPTION**

The **verify** command verifies certificate chains.

**OPTIONS****-help**

Print out a usage message.

**-CAfile file**

A **file** of trusted certificates. The file should contain one or more certificates in PEM format.

**-CApath directory**

A directory of trusted certificates. The certificates should have names of the form: hash.0 or have symbolic links to them of this form (“hash” is the hashed certificate subject name: see the **-hash** option of the **x509** utility). Under Unix the **c\_rehash** script will automatically create symbolic links to a directory of certificates.

**-no-CAfile**

Do not load the trusted CA certificates from the default file location

**-no-CApath**

Do not load the trusted CA certificates from the default directory location

**-allow\_proxy\_certs**

Allow the verification of proxy certificates

**-atime timestamp**

Perform validation checks using time specified by **timestamp** and not current system time. **timestamp** is the number of seconds since 01.01.1970 (UNIX time).

**-check\_ss\_sig**

Verify the signature on the self-signed root CA. This is disabled by default because it doesn't add any security.

**-CRLfile file**

The **file** should contain one or more CRLs in PEM format. This option can be specified more than once to include CRLs from multiple **files**.

**-crl\_download**

Attempt to download CRL information for this certificate.

**-crl\_check**

Checks end entity certificate validity by attempting to look up a valid CRL. If a valid CRL cannot be found an error occurs.

**-crl\_check\_all**

Checks the validity of **all** certificates in the chain by attempting to look up valid CRLs.

**-engine id**

Specifying an engine **id** will cause *verify(1)* to attempt to load the specified engine. The engine will then be set as the default for all its supported algorithms. If you want to load certificates or CRLs that

require engine support via any of the **-trusted**, **-untrusted** or **-CRLfile** options, the **-engine** option must be specified before those options.

**-explicit\_policy**

Set policy variable require-explicit-policy (see RFC5280).

**-extended\_crl**

Enable extended CRL features such as indirect CRLs and alternate CRL signing keys.

**-ignore\_critical**

Normally if an unhandled critical extension is present which is not supported by OpenSSL the certificate is rejected (as required by RFC5280). If this option is set critical extensions are ignored.

**-inhibit\_any**

Set policy variable inhibit-any-policy (see RFC5280).

**-inhibit\_map**

Set policy variable inhibit-policy-mapping (see RFC5280).

**-no\_check\_time**

This option suppresses checking the validity period of certificates and CRLs against the current time. If option **-atime timestamp** is used to specify a verification time, the check is not suppressed.

**-partial\_chain**

Allow verification to succeed even if a *complete* chain cannot be built to a self-signed trust-anchor, provided it is possible to construct a chain to a trusted certificate that might not be self-signed.

**-policy\_arg**

Enable policy processing and add **arg** to the user-initial-policy-set (see RFC5280). The policy **arg** can be an object name an OID in numeric form. This argument can appear more than once.

**-policy\_check**

Enables certificate policy processing.

**-policy\_print**

Print out diagnostics related to policy processing.

**-purpose purpose**

The intended use for the certificate. If this option is not specified, **verify** will not consider certificate purpose during chain verification. Currently accepted uses **aresslclient**, **sslser ver**, **nssslserver**, **smimesign**, **smimeencrypt**. See the **VERIFY OPERATION** section for more information.

**-suiteB\_128\_only, -suiteB\_128, -suiteB\_192**

enable the Suite B mode operation at 128 bit Level of Security, 128 bit or 192 bit, or only 192 bit Level of Security respectively. See RFC6460 for details. In particular the supported signature algorithms are reduced to support only ECDSA and SHA256 or SHA384 and only the elliptic curves P-256 and P-384.

**-trusted\_first**

When constructing the certificate chain, use the trusted certificates specified via **-CAfile**, **-CApath** or **-trusted** before any certificates specified via **-untrusted**. This can be useful in environments with Bridge or Cross-Certified CAs. As of OpenSSL 1.1.0 this option is on by default and cannot be disabled.

**-no\_alt\_chains**

By default, unless **-trusted\_first** is specified, when building a certificate chain, if the first certificate chain found is not trusted, then OpenSSL will attempt to replace untrusted issuer certificates with certificates from the trust store to see if an alternative chain can be found that is trusted. As of OpenSSL 1.1.0, with **-trusted\_first** always on, this option has no effect.

**-untrusted file**

A **file** of additional untrusted certificates (intermediate issuer CAs) used to construct a certificate chain from the subject certificate to a trust-anchor. The **file** should contain one or more certificates in PEM

format. This option can be specified more than once to include untrusted certificates from multiple files.

**-trusted file**

A **file** of trusted certificates, which must be self-signed, unless the **-partial\_chain** option is specified. The **file** contains one or more certificates in PEM format. With this option, no additional (e.g., default) certificate lists are consulted. That is, the only trust-anchors are those listed in **file**. This option can be specified more than once to include trusted certificates from multiple **files**. This option implies the **-no-CAfile** and **-no-CApath** options. This option cannot be used in combination with either of the **-CAfile** or **-CApath** options.

**-use\_deltas**

Enable support for delta CRLs.

**-verbose**

Print extra information about the operations being performed.

**-auth\_level level**

Set the certificate chain authentication security level to **level**. The authentication security level determines the acceptable signature and public key strength when verifying certificate chains. For a certificate chain to validate, the public keys of all the certificates must meet the specified security **level**. The signature algorithm security level is enforced for all the certificates in the chain except for the chain's *trust anchor*, which is either directly trusted or validated by means other than its signature. See [SSL\\_CTX\\_set\\_security\\_level\(3\)](#) for the definitions of the available levels. The default security level is -1, or "not set". At security level 0 or lower all algorithms are acceptable. Security level 1 requires at least 80-bit-equivalent security and is broadly interoperable, though it will, for example, reject MD5 signatures or RSA keys shorter than 1024 bits.

**-verify\_depth num**

Limit the certificate chain to **num** intermediate CA certificates. A maximal depth chain can have up to **num+2** certificates, since neither the end-entity certificate nor the trust-anchor certificate count against the **-verify\_depth** limit.

**-verify\_email email**

Verify if the **email** matches the email address in Subject Alternative Name or the email in the subject Distinguished Name.

**-verify\_hostname hostname**

Verify if the **hostname** matches DNS name in Subject Alternative Name or Common Name in the subject certificate.

**-verify\_ip ip**

Verify if the **ip** matches the IP address in Subject Alternative Name of the subject certificate.

**-verify\_name name**

Use default verification policies like trust model and required certificate policies identified by **name**. The trust model determines which auxiliary trust or reject OIDs are applicable to verifying the given certificate chain. See the **-addtrust** and **-addreject** options of the [x509\(1\)](#) command-line utility. Supported policy names include: **default**, **pkcs7**, **smime\_sign**, **ssl\_client**, **ssl\_server**. These mimics the combinations of purpose and trust settings used in SSL, CMS and S/MIME. As of OpenSSL 1.1.0, the trust model is inferred from the purpose when not specified, so the **-verify\_name** options are functionally equivalent to the corresponding **-purpose** settings.

**-x509\_strict**

For strict X.509 compliance, disable non-compliant workarounds for broken certificates.

**-show\_chain**

Display information about the certificate chain that has been built (if successful). Certificates in the chain that came from the untrusted list will be flagged as "untrusted".

- Indicates the last option. All arguments following this are assumed to be certificate files. This is useful if the first certificate filename begins with a -.

**certificates**

One or more certificates to verify. If no certificates are given, **verify** will attempt to read a certificate from standard input. Certificates must be in PEM format.

**VERIFY OPERATION**

The **verify** program uses the same functions as the internal SSL and S/MIME verification, therefore this description applies to these verify operations too.

There is one crucial difference between the verify operations performed by the **verify** program: wherever possible an attempt is made to continue after an error whereas normally the verify operation would halt on the first error. This allows all the problems with a certificate chain to be determined.

The verify operation consists of a number of separate steps.

Firstly a certificate chain is built up starting from the supplied certificate and ending in the root CA. It is an error if the whole chain cannot be built up. The chain is built up by looking up the issuers certificate of the current certificate. If a certificate is found which is its own issuer it is assumed to be the root CA.

The process of 'looking up the issuers certificate' itself involves a number of steps. After all certificates whose subject name matches the issuer name of the current certificate are subject to further tests. The relevant authority key identifier components of the current certificate (if present) must match the subject key identifier (if present) and issuer and serial number of the candidate issuer, in addition the keyUsage extension of the candidate issuer (if present) must permit certificate signing.

The lookup first looks in the list of untrusted certificates and if no match is found the remaining lookups are from the trusted certificates. The root CA is always looked up in the trusted certificate list: if the certificate to verify is a root certificate then an exact match must be found in the trusted list.

The second operation is to check every untrusted certificate's extensions for consistency with the supplied purpose. If the **-purpose** option is not included then no checks are done. The supplied or "leaf" certificate must have extensions compatible with the supplied purpose and all other certificates must also be valid CA certificates. The precise extensions required are described in more detail in the **CERTIFICATE EXTENSIONS** section of the **x509** utility.

The third operation is to check the trust settings on the root CA. The root CA should be trusted for the supplied purpose. For compatibility with previous versions of OpenSSL, a certificate with no trust settings is considered to be valid for all purposes.

The final operation is to check the validity of the certificate chain. The validity period is checked against the current system time and the notBefore and notAfter dates in the certificate. The certificate signatures are also checked at this point.

If all operations complete successfully then certificate is considered valid. If any operation fails then the certificate is not valid.

**DIAGNOSTICS**

When a verify operation fails the output messages can be somewhat cryptic. The general form of the error message is:

```
server.pem: /C=AU/ST=Queensland/O=CryptSoft Pty Ltd/CN=Test CA (1024 bit)
error 24 at 1 depth lookup:invalid CA certificate
```

The first line contains the name of the certificate being verified followed by the subject name of the certificate. The second line contains the error number and the depth. The depth is number of the certificate being verified when a problem was detected starting with zero for the certificate being verified itself then 1 for the CA that signed the certificate and so on. Finally a text version of the error number is presented.

A partial list of the error codes and messages is shown below, this also includes the name of the error code as defined in the header file x509\_vfy.h Some of the error codes are defined but never returned: these are described as "unused".

**X509\_V\_OK**

The operation was successful.

**X509\_V\_ERR\_UNSPECIFIED**

Unspecified error; should not happen.

**X509\_V\_ERR\_UNABLE\_TO\_GET\_ISSUER\_CERT**

The issuer certificate of a looked up certificate could not be found. This normally means the list of trusted certificates is not complete.

**X509\_V\_ERR\_UNABLE\_TO\_GET\_CRL**

The CRL of a certificate could not be found.

**X509\_V\_ERR\_UNABLE\_TO\_DECRYPT\_CERT\_SIGNATURE**

The certificate signature could not be decrypted. This means that the actual signature value could not be determined rather than it not matching the expected value, this is only meaningful for RSA keys.

**X509\_V\_ERR\_UNABLE\_TO\_DECRYPT\_CRL\_SIGNATURE**

The CRL signature could not be decrypted: this means that the actual signature value could not be determined rather than it not matching the expected value. Unused.

**X509\_V\_ERR\_UNABLE\_TO\_DECODE\_ISSUER\_PUBLIC\_KEY**

The public key in the certificate SubjectPublicKeyInfo could not be read.

**X509\_V\_ERR\_CERT\_SIGNATURE\_FAILURE**

The signature of the certificate is invalid.

**X509\_V\_ERR\_CRL\_SIGNATURE\_FAILURE**

The signature of the certificate is invalid.

**X509\_V\_ERR\_CERT\_NOT\_YET\_VALID**

The certificate is not yet valid: the notBefore date is after the current time.

**X509\_V\_ERR\_CERT\_HAS\_EXPIRED**

The certificate has expired: that is the notAfter date is before the current time.

**X509\_V\_ERR\_CRL\_NOT\_YET\_VALID**

The CRL is not yet valid.

**X509\_V\_ERR\_CRL\_HAS\_EXPIRED**

The CRL has expired.

**X509\_V\_ERR\_ERROR\_IN\_CERT\_NOT\_BEFORE\_FIELD**

The certificate notBefore field contains an invalid time.

**X509\_V\_ERR\_ERROR\_IN\_CERT\_NOT\_AFTER\_FIELD**

The certificate notAfter field contains an invalid time.

**X509\_V\_ERR\_ERROR\_IN\_CRL\_LAST\_UPDATE\_FIELD**

The CRL lastUpdate field contains an invalid time.

**X509\_V\_ERR\_ERROR\_IN\_CRL\_NEXT\_UPDATE\_FIELD**

The CRL nextUpdate field contains an invalid time.

**X509\_V\_ERR\_OUT\_OF\_MEM**

An error occurred trying to allocate memory. This should never happen.

**X509\_V\_ERR\_DEPTH\_ZERO\_SELF\_SIGNED\_CERT**

The passed certificate is self-signed and the same certificate cannot be found in the list of trusted certificates.

**X509\_V\_ERR\_SELF\_SIGNED\_CERT\_IN\_CHAIN**

The certificate chain could be built up using the untrusted certificates but the root could not be found locally.

**X509\_V\_ERR\_UNABLE\_TO\_GET\_ISSUER\_CERT\_LOCALLY**

The issuer certificate could not be found: this occurs if the issuer certificate of an untrusted certificate cannot be found.

**X509\_V\_ERR\_UNABLE\_TO\_VERIFY\_LEAF\_SIGNATURE**

No signatures could be verified because the chain contains only one certificate and it is not self signed.

**X509\_V\_ERR\_CERT\_CHAIN\_TOO\_LONG**

The certificate chain length is greater than the supplied maximum depth. Unused.

**X509\_V\_ERR\_CERT\_REVOKED**

The certificate has been revoked.

**X509\_V\_ERR\_INVALID\_CA**

A CA certificate is invalid. Either it is not a CA or its extensions are not consistent with the supplied purpose.

**X509\_V\_ERR\_PATH\_LENGTH\_EXCEEDED**

The basicConstraints pathlength parameter has been exceeded.

**X509\_V\_ERR\_INVALID\_PURPOSE**

The supplied certificate cannot be used for the specified purpose.

**X509\_V\_ERR\_CERT\_UNTRUSTED**

the root CA is not marked as trusted for the specified purpose.

**X509\_V\_ERR\_CERT\_REJECTED**

The root CA is marked to reject the specified purpose.

**X509\_V\_ERR\_SUBJECT\_ISSUER\_MISMATCH**

not used as of OpenSSL 1.1.0 as a result of the deprecation of the **-issuer\_checks** option.

**X509\_V\_ERR\_AKID\_SKID\_MISMATCH**

Not used as of OpenSSL 1.1.0 as a result of the deprecation of the **-issuer\_checks** option.

**X509\_V\_ERR\_AKID\_ISSUER\_SERIAL\_MISMATCH**

Not used as of OpenSSL 1.1.0 as a result of the deprecation of the **-issuer\_checks** option.

**X509\_V\_ERR\_KEYUSAGE\_NO\_CERTSIGN**

Not used as of OpenSSL 1.1.0 as a result of the deprecation of the **-issuer\_checks** option.

**X509\_V\_ERR\_UNABLE\_TO\_GET\_CRL\_ISSUER**

Unable to get CRL issuer certificate.

**X509\_V\_ERR\_UNHANDLED\_CRITICAL\_EXTENSION**

Unhandled critical extension.

**X509\_V\_ERR\_KEYUSAGE\_NO\_CRL\_SIGN**

Key usage does not include CRL signing.

**X509\_V\_ERR\_UNHANDLED\_CRITICAL\_CRL\_EXTENSION**

Unhandled critical CRL extension.

**X509\_V\_ERR\_INVALID\_NON\_CA**

Invalid non-CA certificate has CA markings.

**X509\_V\_ERR\_PROXY\_PATH\_LENGTH\_EXCEEDED**

Proxy path length constraint exceeded.

**X509\_V\_ERR\_PROXY\_SUBJECT\_INVALID**

Proxy certificate subject is invalid. It MUST be the same as the issuer with a single CN component added.

**X509\_V\_ERR\_KEYUSAGE\_NO\_DIGITAL\_SIGNATURE**

Key usage does not include digital signature.

- X509\_V\_ERR\_PROXY\_CERTIFICATES\_NOT\_ALLOWED**  
Proxy certificates not allowed, please use **-allow\_proxy\_certs**.
- X509\_V\_ERR\_INVALID\_EXTENSION**  
Invalid or inconsistent certificate extension.
- X509\_V\_ERR\_INVALID\_POLICY\_EXTENSION**  
Invalid or inconsistent certificate policy extension.
- X509\_V\_ERR\_NO\_EXPLICIT\_POLICY**  
No explicit policy.
- X509\_V\_ERR\_DIFFERENT\_CRL\_SCOPE**  
Different CRL scope.
- X509\_V\_ERR\_UNSUPPORTED\_EXTENSION\_FEATURE**  
Unsupported extension feature.
- X509\_V\_ERR\_UNNESTED\_RESOURCE**  
RFC 3779 resource not subset of parent's resources.
- X509\_V\_ERR\_PERMITTED\_VIOLATION**  
Permitted subtree violation.
- X509\_V\_ERR\_EXCLUDED\_VIOLATION**  
Excluded subtree violation.
- X509\_V\_ERR\_SUBTREE\_MINMAX**  
Name constraints minimum and maximum not supported.
- X509\_V\_ERR\_APPLICATION\_VERIFICATION**  
Application verification failure. Unused.
- X509\_V\_ERR\_UNSUPPORTED\_CONSTRAINT\_TYPE**  
Unsupported name constraint type.
- X509\_V\_ERR\_UNSUPPORTED\_CONSTRAINT\_SYNTAX**  
Unsupported or invalid name constraint syntax.
- X509\_V\_ERR\_UNSUPPORTED\_NAME\_SYNTAX**  
Unsupported or invalid name syntax.
- X509\_V\_ERR\_CRL\_PATH\_VALIDATION\_ERROR**  
CRL path validation error.
- X509\_V\_ERR\_PATH\_LOOP**  
Path loop.
- X509\_V\_ERR\_SUITE\_B\_INVALID\_VERSION**  
Suite B: certificate version invalid.
- X509\_V\_ERR\_SUITE\_B\_INVALID\_ALGORITHM**  
Suite B: invalid public key algorithm.
- X509\_V\_ERR\_SUITE\_B\_INVALID\_CURVE**  
Suite B: invalid ECC curve.
- X509\_V\_ERR\_SUITE\_B\_INVALID\_SIGNATURE\_ALGORITHM**  
Suite B: invalid signature algorithm.
- X509\_V\_ERR\_SUITE\_B\_LOS\_NOT\_ALLOWED**  
Suite B: curve not allowed for this LOS.
- X509\_V\_ERR\_SUITE\_B\_CANNOT\_SIGN\_P\_384\_WITH\_P\_256**  
Suite B: cannot sign P-384 with P-256.

**X509\_V\_ERR\_HOSTNAME\_MISMATCH**

Hostname mismatch.

**X509\_V\_ERR\_EMAIL\_MISMATCH**

Email address mismatch.

**X509\_V\_ERR\_IP\_ADDRESS\_MISMATCH**

IP address mismatch.

**X509\_V\_ERR\_DANE\_NO\_MATCH**

DANE TLSA authentication is enabled, but no TLSA records matched the certificate chain. This error is only possible in [s\\_client\(1\)](#).

**BUGS**

Although the issuer checks are a considerable improvement over the old technique they still suffer from limitations in the underlying X509\_LOOKUP API. One consequence of this is that trusted certificates with matching subject name must either appear in a file (as specified by the **-CAfile** option) or a directory (as specified by **-CApath**). If they occur in both then only the certificates in the file will be recognised.

Previous versions of OpenSSL assume certificates with matching subject name are identical and mishandled them.

Previous versions of this documentation swapped the meaning of the

**X509\_V\_ERR\_UNABLE\_TO\_GET\_ISSUER\_CERT**

and **X509\_V\_ERR\_UNABLE\_TO\_GET\_ISSUER\_CERT\_LOCALLY** error codes.

**SEE ALSO**

[x509\(1\)](#)

**HISTORY**

The **-show\_chain** option was first added to OpenSSL 1.1.0.

The **-issuer\_checks** option is deprecated as of OpenSSL 1.1.0 and is silently ignored.

**COPYRIGHT**

Copyright 2000-2016 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the OpenSSL license (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>.