

**NAME**

SSL\_CONF\_cmd\_value\_type, SSL\_CONF\_cmd - send configuration command

**SYNOPSIS**

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

```
int SSL_CONF_cmd(SSL_CONF_CTX *ctx, const char *option, const char *value);  
int SSL_CONF_cmd_value_type(SSL_CONF_CTX *ctx, const char *option);
```

**DESCRIPTION**

The function **SSL\_CONF\_cmd()** performs configuration operation **option** with optional parameter **value** on **ctx**. Its purpose is to simplify application configuration of **SSL\_CTX** or **SSL** structures by providing a common framework for command line options or configuration files.

**SSL\_CONF\_cmd\_value\_type()** returns the type of value that **option** refers to.

**SUPPORTED COMMAND LINE COMMANDS**

Currently supported **option** names for command lines (i.e. when the flag **SSL\_CONF\_FLAG\_CMDLINE** is set) are listed below. Note: all **option** names are case sensitive. Unless otherwise stated commands can be used by both clients and servers and the **value** parameter is not used. The default prefix for command line commands is - and that is reflected below.

**-bugs**

Various bug workarounds are set, same as setting **SSL\_OP\_ALL**.

**-no\_comp**

Disables support for SSL/TLS compression, same as setting **SSL\_OP\_NO\_COMPRESSION**. As of OpenSSL 1.1.0, compression is off by default.

**-comp**

Enables support for SSL/TLS compression, same as clearing **SSL\_OP\_NO\_COMPRESSION**. This command was introduced in OpenSSL 1.1.0. As of OpenSSL 1.1.0, compression is off by default.

**-no\_ticket**

Disables support for session tickets, same as setting **SSL\_OP\_NO\_TICKET**.

**-serverpref**

Use server and not client preference order when determining which cipher suite, signature algorithm or elliptic curve to use for an incoming connection. Equivalent to

**SSL\_OP\_CIPHER\_SERVER\_PREFERENCE.** Only used by servers.

**-client\_renegotiation**

Allows servers to accept client-initiated renegotiation. Equivalent to setting **SSL\_OP\_ALLOW\_CLIENT\_RENEGOTIATION.** Only used by servers.

**-legacy\_renegotiation**

Permits the use of unsafe legacy renegotiation. Equivalent to setting **SSL\_OP\_ALLOW\_UNSAFE\_LEGACY\_RENEGOTIATION.**

**-no\_renegotiation**

Disables all attempts at renegotiation in TLSv1.2 and earlier, same as setting **SSL\_OP\_NO\_RENEGOTIATION.**

**-no\_resumption\_on\_reneg**

Sets **SSL\_OP\_NO\_SESSION\_RESUMPTION\_ON\_RENEGOTIATION.** Only used by servers.

**-legacy\_server\_connect, -no\_legacy\_server\_connect**

Permits or prohibits the use of unsafe legacy renegotiation for OpenSSL clients only. Equivalent to setting or clearing **SSL\_OP\_LEGACY\_SERVER\_CONNECT.**

**-prioritize\_chacha**

Prioritize ChaCha ciphers when the client has a ChaCha20 cipher at the top of its preference list. This usually indicates a client without AES hardware acceleration (e.g. mobile) is in use. Equivalent to **SSL\_OP\_PRIORITIZE\_CHACHA.** Only used by servers. Requires **-serverpref.**

**-allow\_no\_dhe\_kex**

In TLSv1.3 allow a non-(ec)dhe based key exchange mode on resumption. This means that there will be no forward secrecy for the resumed session.

**-strict**

Enables strict mode protocol handling. Equivalent to setting **SSL\_CERT\_FLAG\_TLS\_STRICT.**

**-sigalgs *algs***

This sets the supported signature algorithms for TLSv1.2 and TLSv1.3. For clients this value is used directly for the supported signature algorithms extension. For servers it is used to determine which signature algorithms to support.

The **algs** argument should be a colon separated list of signature algorithms in order of decreasing preference of the form **algorithm+hash** or **signature\_scheme**. **algorithm** is one of **RSA**, **DSA** or

**ECDSA** and **hash** is a supported algorithm OID short name such as **SHA1**, **SHA224**, **SHA256**, **SHA384** or **SHA512**. Note: algorithm and hash names are case sensitive. **signature\_scheme** is one of the signature schemes defined in TLSv1.3, specified using the IETF name, e.g., **ecdsa\_secp256r1\_sha256**, **ed25519**, or **rsa\_pss\_pss\_sha256**.

If this option is not set then all signature algorithms supported by the OpenSSL library are permissible.

Note: algorithms which specify a PKCS#1 v1.5 signature scheme (either by using **RSA** as the **algorithm** or by using one of the **rsa\_pkcs1\_\*** identifiers) are ignored in TLSv1.3 and will not be negotiated.

#### **-client\_sigalgs** *algs*

This sets the supported signature algorithms associated with client authentication for TLSv1.2 and TLSv1.3. For servers the **algs** is used in the **signature\_algorithms** field of a **CertificateRequest** message. For clients it is used to determine which signature algorithm to use with the client certificate. If a server does not request a certificate this option has no effect.

The syntax of **algs** is identical to **-sigalgs**. If not set, then the value set for **-sigalgs** will be used instead.

#### **-groups** *groups*

This sets the supported groups. For clients, the groups are sent using the supported groups extension. For servers, it is used to determine which group to use. This setting affects groups used for signatures (in TLSv1.2 and earlier) and key exchange. The first group listed will also be used for the **key\_share** sent by a client in a TLSv1.3 **ClientHello**.

The **groups** argument is a colon separated list of groups. The group can be either the **NIST** name (e.g. **P-256**), some other commonly used name where applicable (e.g. **X25519**, **ffdhe2048**) or an OpenSSL OID name (e.g. **prime256v1**). Group names are case sensitive. The list should be in order of preference with the most preferred group first.

Currently supported groups for TLSv1.3 are **P-256**, **P-384**, **P-521**, **X25519**, **X448**, **ffdhe2048**, **ffdhe3072**, **ffdhe4096**, **ffdhe6144**, **ffdhe8192**.

#### **-curves** *groups*

This is a synonym for the **-groups** command.

#### **-named\_curve** *curve*

This sets the temporary curve used for ephemeral ECDH modes. Only used by servers.

The **groups** argument is a curve name or the special value **auto** which picks an appropriate curve based on client and server preferences. The curve can be either the **NIST** name (e.g. **P-256**) or an OpenSSL OID name (e.g. **prime256v1**). Curve names are case sensitive.

**-cipher** *ciphers*

Sets the TLSv1.2 and below ciphersuite list to **ciphers**. This list will be combined with any configured TLSv1.3 ciphersuites. Note: syntax checking of **ciphers** is currently not performed unless a **SSL** or **SSL\_CTX** structure is associated with **ctx**.

**-ciphersuites** *1.3ciphers*

Sets the available ciphersuites for TLSv1.3 to value. This is a colon-separated list of TLSv1.3 ciphersuite names in order of preference. This list will be combined any configured TLSv1.2 and below ciphersuites. See **openssl-ciphers**(1) for more information.

**-min\_protocol** *minprot*, **-max\_protocol** *maxprot*

Sets the minimum and maximum supported protocol. Currently supported protocol values are **SSLv3**, **TLSv1**, **TLSv1.1**, **TLSv1.2**, **TLSv1.3** for TLS; **DTLSv1**, **DTLSv1.2** for DTLS, and **None** for no limit. If either the lower or upper bound is not specified then only the other bound applies, if specified. If your application supports both TLS and DTLS you can specify any of these options twice, once with a bound for TLS and again with an appropriate bound for DTLS. To restrict the supported protocol versions use these commands rather than the deprecated alternative commands below.

**-record\_padding** *padding*

Attempts to pad TLSv1.3 records so that they are a multiple of **padding** in length on send. A **padding** of 0 or 1 turns off padding. Otherwise, the **padding** must be >1 or <=16384.

**-debug\_broken\_protocol**

Ignored.

**-no\_middlebox**

Turn off "middlebox compatibility", as described below.

### Additional Options

The following options are accepted by **SSL\_CONF\_cmd()**, but are not processed by the OpenSSL commands.

**-cert** *file*

Attempts to use **file** as the certificate for the appropriate context. It currently uses **SSL\_CTX\_use\_certificate\_chain\_file()** if an **SSL\_CTX** structure is set or

**SSL\_use\_certificate\_file()** with filetype PEM if an **SSL** structure is set. This option is only supported if certificate operations are permitted.

**-key *file***

Attempts to use **file** as the private key for the appropriate context. This option is only supported if certificate operations are permitted. Note: if no **-key** option is set then a private key is not loaded unless the flag **SSL\_CONF\_FLAG\_REQUIRE\_PRIVATE** is set.

**-dhparam *file***

Attempts to use **file** as the set of temporary DH parameters for the appropriate context. This option is only supported if certificate operations are permitted.

**-no\_ssl3, -no\_tls1, -no\_tls1\_1, -no\_tls1\_2, -no\_tls1\_3**

Disables protocol support for SSLv3, TLSv1.0, TLSv1.1, TLSv1.2 or TLSv1.3 by setting the corresponding options **SSL\_OP\_NO\_SSLv3**, **SSL\_OP\_NO\_TLSv1**, **SSL\_OP\_NO\_TLSv1\_1**, **SSL\_OP\_NO\_TLSv1\_2** and **SSL\_OP\_NO\_TLSv1\_3** respectively. These options are deprecated, use **-min\_protocol** and **-max\_protocol** instead.

**-anti\_replay, -no\_anti\_replay**

Switches replay protection, on or off respectively. With replay protection on, OpenSSL will automatically detect if a session ticket has been used more than once, TLSv1.3 has been negotiated, and early data is enabled on the server. A full handshake is forced if a session ticket is used a second or subsequent time. Anti-Replay is on by default unless overridden by a configuration file and is only used by servers. Anti-replay measures are required for compliance with the TLSv1.3 specification. Some applications may be able to mitigate the replay risks in other ways and in such cases the built-in OpenSSL functionality is not required. Switching off anti-replay is equivalent to **SSL\_OP\_NO\_ANTI\_REPLAY**.

## SUPPORTED CONFIGURATION FILE COMMANDS

Currently supported **option** names for configuration files (i.e., when the flag **SSL\_CONF\_FLAG\_FILE** is set) are listed below. All configuration file **option** names are case insensitive so **signaturealgorithms** is recognised as well as **SignatureAlgorithms**. Unless otherwise stated the **value** names are also case insensitive.

Note: the command prefix (if set) alters the recognised **option** values.

### CipherString

Sets the ciphersuite list for TLSv1.2 and below to **value**. This list will be combined with any configured TLSv1.3 ciphersuites. Note: syntax checking of **value** is currently not performed unless an **SSL** or **SSL\_CTX** structure is associated with **ctx**.

### Ciphersuites

Sets the available ciphersuites for TLSv1.3 to **value**. This is a colon-separated list of TLSv1.3 ciphersuite names in order of preference. This list will be combined any configured TLSv1.2 and below ciphersuites. See **openssl-ciphers(1)** for more information.

### Certificate

Attempts to use the file **value** as the certificate for the appropriate context. It currently uses **SSL\_CTX\_use\_certificate\_chain\_file()** if an **SSL\_CTX** structure is set or **SSL\_use\_certificate\_file()** with filetype PEM if an **SSL** structure is set. This option is only supported if certificate operations are permitted.

### PrivateKey

Attempts to use the file **value** as the private key for the appropriate context. This option is only supported if certificate operations are permitted. Note: if no **PrivateKey** option is set then a private key is not loaded unless the **SSL\_CONF\_FLAG\_REQUIRE\_PRIVATE** is set.

### ChainCAFile, ChainCAPath, VerifyCAFile, VerifyCAPath

These options indicate a file or directory used for building certificate chains or verifying certificate chains. These options are only supported if certificate operations are permitted.

### RequestCAFile

This option indicates a file containing a set of certificates in PEM form. The subject names of the certificates are sent to the peer in the **certificate\_authorities** extension for TLS 1.3 (in ClientHello or CertificateRequest) or in a certificate request for previous versions or TLS.

### ServerInfoFile

Attempts to use the file **value** in the "serverinfo" extension using the function **SSL\_CTX\_use\_serverinfo\_file**.

### DHParameters

Attempts to use the file **value** as the set of temporary DH parameters for the appropriate context. This option is only supported if certificate operations are permitted.

### RecordPadding

Attempts to pad TLSv1.3 records so that they are a multiple of **value** in length on send. A **value** of 0 or 1 turns off padding. Otherwise, the **value** must be >1 or <=16384.

### SignatureAlgorithms

This sets the supported signature algorithms for TLSv1.2 and TLSv1.3. For clients this value is used directly for the supported signature algorithms extension. For servers it is used to determine

which signature algorithms to support.

The **value** argument should be a colon separated list of signature algorithms in order of decreasing preference of the form **algorithm+hash** or **signature\_scheme**. **algorithm** is one of **RSA**, **DSA** or **ECDSA** and **hash** is a supported algorithm OID short name such as **SHA1**, **SHA224**, **SHA256**, **SHA384** or **SHA512**. Note: algorithm and hash names are case sensitive. **signature\_scheme** is one of the signature schemes defined in TLSv1.3, specified using the IETF name, e.g., **ecdsa\_secp256r1\_sha256**, **ed25519**, or **rsa\_pss\_sha256**.

If this option is not set then all signature algorithms supported by the OpenSSL library are permissible.

Note: algorithms which specify a PKCS#1 v1.5 signature scheme (either by using **RSA** as the **algorithm** or by using one of the **rsa\_pkcs1\_\*** identifiers) are ignored in TLSv1.3 and will not be negotiated.

### ClientSignatureAlgorithms

This sets the supported signature algorithms associated with client authentication for TLSv1.2 and TLSv1.3. For servers the value is used in the **signature\_algorithms** field of a **CertificateRequest** message. For clients it is used to determine which signature algorithm to use with the client certificate. If a server does not request a certificate this option has no effect.

The syntax of **value** is identical to **SignatureAlgorithms**. If not set then the value set for **SignatureAlgorithms** will be used instead.

### Groups

This sets the supported groups. For clients, the groups are sent using the supported groups extension. For servers, it is used to determine which group to use. This setting affects groups used for signatures (in TLSv1.2 and earlier) and key exchange. The first group listed will also be used for the **key\_share** sent by a client in a TLSv1.3 **ClientHello**.

The **value** argument is a colon separated list of groups. The group can be either the **NIST** name (e.g. **P-256**), some other commonly used name where applicable (e.g. **X25519**, **ffdhe2048**) or an OpenSSL OID name (e.g. **prime256v1**). Group names are case sensitive. The list should be in order of preference with the most preferred group first.

Currently supported groups for TLSv1.3 are **P-256**, **P-384**, **P-521**, **X25519**, **X448**, **ffdhe2048**, **ffdhe3072**, **ffdhe4096**, **ffdhe6144**, **ffdhe8192**.

### Curves

This is a synonym for the "Groups" command.

### **MinProtocol**

This sets the minimum supported SSL, TLS or DTLS version.

Currently supported protocol values are **SSLv3**, **TLSv1**, **TLSv1.1**, **TLSv1.2**, **TLSv1.3**, **DTLSv1** and **DTLSv1.2**. The SSL and TLS bounds apply only to TLS-based contexts, while the DTLS bounds apply only to DTLS-based contexts. The command can be repeated with one instance setting a TLS bound, and the other setting a DTLS bound. The value **None** applies to both types of contexts and disables the limits.

### **MaxProtocol**

This sets the maximum supported SSL, TLS or DTLS version.

Currently supported protocol values are **SSLv3**, **TLSv1**, **TLSv1.1**, **TLSv1.2**, **TLSv1.3**, **DTLSv1** and **DTLSv1.2**. The SSL and TLS bounds apply only to TLS-based contexts, while the DTLS bounds apply only to DTLS-based contexts. The command can be repeated with one instance setting a TLS bound, and the other setting a DTLS bound. The value **None** applies to both types of contexts and disables the limits.

### **Protocol**

This can be used to enable or disable certain versions of the SSL, TLS or DTLS protocol.

The **value** argument is a comma separated list of supported protocols to enable or disable. If a protocol is preceded by - that version is disabled.

All protocol versions are enabled by default. You need to disable at least one protocol version for this setting have any effect. Only enabling some protocol versions does not disable the other protocol versions.

Currently supported protocol values are **SSLv3**, **TLSv1**, **TLSv1.1**, **TLSv1.2**, **TLSv1.3**, **DTLSv1** and **DTLSv1.2**. The special value **ALL** refers to all supported versions.

This can't enable protocols that are disabled using **MinProtocol** or **MaxProtocol**, but can disable protocols that are still allowed by them.

The **Protocol** command is fragile and deprecated; do not use it. Use **MinProtocol** and **MaxProtocol** instead. If you do use **Protocol**, make sure that the resulting range of enabled protocols has no "holes", e.g. if TLS 1.0 and TLS 1.2 are both enabled, make sure to also leave TLS 1.1 enabled.



## Options

The **value** argument is a comma separated list of various flags to set. If a flag string is preceded - it is disabled. See the **SSL\_CTX\_set\_options(3)** function for more details of individual options.

Each option is listed below. Where an operation is enabled by default the **-flag** syntax is needed to disable it.

**SessionTicket**: session ticket support, enabled by default. Inverse of **SSL\_OP\_NO\_TICKET**: that is **-SessionTicket** is the same as setting **SSL\_OP\_NO\_TICKET**.

**Compression**: SSL/TLS compression support, disabled by default. Inverse of **SSL\_OP\_NO\_COMPRESSION**.

**EmptyFragments**: use empty fragments as a countermeasure against a SSL 3.0/TLS 1.0 protocol vulnerability affecting CBC ciphers. It is set by default. Inverse of **SSL\_OP\_DONT\_INSERT\_EMPTY\_FRAGMENTS**.

**Bugs**: enable various bug workarounds. Same as **SSL\_OP\_ALL**.

**DHSingle**: enable single use DH keys, set by default. Inverse of **SSL\_OP\_DH\_SINGLE**. Only used by servers.

**ECDHSingle**: enable single use ECDH keys, set by default. Inverse of **SSL\_OP\_ECDH\_SINGLE**. Only used by servers.

**ServerPreference**: use server and not client preference order when determining which cipher suite, signature algorithm or elliptic curve to use for an incoming connection. Equivalent to **SSL\_OP\_CIPHER\_SERVER\_PREFERENCE**. Only used by servers.

**PrioritizeChaCha**: prioritizes ChaCha ciphers when the client has a ChaCha20 cipher at the top of its preference list. This usually indicates a mobile client is in use. Equivalent to **SSL\_OP\_PRIORITIZE\_CHACHA**. Only used by servers.

**NoResumptionOnRenegotiation**: set **SSL\_OP\_NO\_SESSION\_RESUMPTION\_ON\_RENEGOTIATION** flag. Only used by servers.

**NoRenegotiation**: disables all attempts at renegotiation in TLSv1.2 and earlier, same as setting **SSL\_OP\_NO\_RENEGOTIATION**.

**UnsafeLegacyRenegotiation**: permits the use of unsafe legacy renegotiation. Equivalent to

**SSL\_OP\_ALLOW\_UNSAFE\_LEGACY\_RENEGOTIATION.**

**UnsafeLegacyServerConnect:** permits the use of unsafe legacy renegotiation for OpenSSL clients only. Equivalent to **SSL\_OP\_LEGACY\_SERVER\_CONNECT**.

**EncryptThenMac:** use encrypt-then-mac extension, enabled by default. Inverse of **SSL\_OP\_NO\_ENCRYPT\_THEN\_MAC**: that is, **-EncryptThenMac** is the same as setting **SSL\_OP\_NO\_ENCRYPT\_THEN\_MAC**.

**AllowNoDHEKEX:** In TLSv1.3 allow a non-(ec)dhe based key exchange mode on resumption. This means that there will be no forward secrecy for the resumed session. Equivalent to **SSL\_OP\_ALLOW\_NO\_DHE\_KEX**.

**MiddleboxCompat:** If set then dummy Change Cipher Spec (CCS) messages are sent in TLSv1.3. This has the effect of making TLSv1.3 look more like TLSv1.2 so that middleboxes that do not understand TLSv1.3 will not drop the connection. This option is set by default. A future version of OpenSSL may not set this by default. Equivalent to **SSL\_OP\_ENABLE\_MIDDLEBOX\_COMPAT**.

**AntiReplay:** If set then OpenSSL will automatically detect if a session ticket has been used more than once, TLSv1.3 has been negotiated, and early data is enabled on the server. A full handshake is forced if a session ticket is used a second or subsequent time. This option is set by default and is only used by servers. Anti-replay measures are required to comply with the TLSv1.3 specification. Some applications may be able to mitigate the replay risks in other ways and in such cases the built-in OpenSSL functionality is not required. Disabling anti-replay is equivalent to setting **SSL\_OP\_NO\_ANTI\_REPLAY**.

**ExtendedMasterSecret:** use extended master secret extension, enabled by default. Inverse of **SSL\_OP\_NO\_EXTENDED\_MASTER\_SECRET**: that is, **-ExtendedMasterSecret** is the same as setting **SSL\_OP\_NO\_EXTENDED\_MASTER\_SECRET**.

**CANames:** use CA names extension, enabled by default. Inverse of **SSL\_OP\_DISABLE\_TLSEXT\_CA\_NAMES**: that is, **-CANames** is the same as setting **SSL\_OP\_DISABLE\_TLSEXT\_CA\_NAMES**.

**KTLS:** Enables kernel TLS if support has been compiled in, and it is supported by the negotiated ciphersuites and extensions. Equivalent to **SSL\_OP\_ENABLE\_KTLS**.

**VerifyMode**

The **value** argument is a comma separated list of flags to set.

**Peer** enables peer verification: for clients only.

**Request** requests but does not require a certificate from the client. Servers only.

**Require** requests and requires a certificate from the client: an error occurs if the client does not present a certificate. Servers only.

**Once** requests a certificate from a client only on the initial connection: not when renegotiating. Servers only.

**RequestPostHandshake** configures the connection to support requests but does not require a certificate from the client post-handshake. A certificate will not be requested during the initial handshake. The server application must provide a mechanism to request a certificate post-handshake. Servers only. TLSv1.3 only.

**RequiresPostHandshake** configures the connection to support requests and requires a certificate from the client post-handshake: an error occurs if the client does not present a certificate. A certificate will not be requested during the initial handshake. The server application must provide a mechanism to request a certificate post-handshake. Servers only. TLSv1.3 only.

#### **ClientCAFile, ClientCAPath**

A file or directory of certificates in PEM format whose names are used as the set of acceptable names for client CAs. Servers only. This option is only supported if certificate operations are permitted.

### **SUPPORTED COMMAND TYPES**

The function `SSL_CONF_cmd_value_type()` currently returns one of the following types:

#### **SSL\_CONF\_TYPE\_UNKNOWN**

The **option** string is unrecognised, this return value can be use to flag syntax errors.

#### **SSL\_CONF\_TYPE\_STRING**

The value is a string without any specific structure.

#### **SSL\_CONF\_TYPE\_FILE**

The value is a filename.

#### **SSL\_CONF\_TYPE\_DIR**

The value is a directory name.

## SSL\_CONF\_TYPE\_NONE

The value string is not used e.g. a command line option which doesn't take an argument.

## NOTES

The order of operations is significant. This can be used to set either defaults or values which cannot be overridden. For example if an application calls:

```
SSL_CONF_cmd(ctx, "Protocol", "-SSLv3");  
SSL_CONF_cmd(ctx, userparam, uservalue);
```

it will disable SSLv3 support by default but the user can override it. If however the call sequence is:

```
SSL_CONF_cmd(ctx, userparam, uservalue);  
SSL_CONF_cmd(ctx, "Protocol", "-SSLv3");
```

SSLv3 is **always** disabled and attempt to override this by the user are ignored.

By checking the return code of **SSL\_CONF\_cmd()** it is possible to query if a given **option** is recognised, this is useful if **SSL\_CONF\_cmd()** values are mixed with additional application specific operations.

For example an application might call **SSL\_CONF\_cmd()** and if it returns -2 (unrecognised command) continue with processing of application specific commands.

Applications can also use **SSL\_CONF\_cmd()** to process command lines though the utility function **SSL\_CONF\_cmd\_argv()** is normally used instead. One way to do this is to set the prefix to an appropriate value using **SSL\_CONF\_CTX\_set1\_prefix()**, pass the current argument to **option** and the following argument to **value** (which may be NULL).

In this case if the return value is positive then it is used to skip that number of arguments as they have been processed by **SSL\_CONF\_cmd()**. If -2 is returned then **option** is not recognised and application specific arguments can be checked instead. If -3 is returned a required argument is missing and an error is indicated. If 0 is returned some other error occurred and this can be reported back to the user.

The function **SSL\_CONF\_cmd\_value\_type()** can be used by applications to check for the existence of a command or to perform additional syntax checking or translation of the command value. For example if the return value is **SSL\_CONF\_TYPE\_FILE** an application could translate a relative pathname to an absolute pathname.

## RETURN VALUES

**SSL\_CONF\_cmd()** returns 1 if the value of **option** is recognised and **value** is **NOT** used and 2 if both **option** and **value** are used. In other words it returns the number of arguments processed. This is useful when processing command lines.

A return value of -2 means **option** is not recognised.

A return value of -3 means **option** is recognised and the command requires a value but **value** is NULL.

A return code of 0 indicates that both **option** and **value** are valid but an error occurred attempting to perform the operation: for example due to an error in the syntax of **value** in this case the error queue may provide additional information.

## EXAMPLES

Set supported signature algorithms:

```
SSL_CONF_cmd(ctx, "SignatureAlgorithms", "ECDSA+SHA256:RSA+SHA256:DSA+SHA256");
```

There are various ways to select the supported protocols.

This set the minimum protocol version to TLSv1, and so disables SSLv3. This is the recommended way to disable protocols.

```
SSL_CONF_cmd(ctx, "MinProtocol", "TLSv1");
```

The following also disables SSLv3:

```
SSL_CONF_cmd(ctx, "Protocol", "-SSLv3");
```

The following will first enable all protocols, and then disable SSLv3. If no protocol versions were disabled before this has the same effect as "-SSLv3", but if some versions were disabled this will re-enable them before disabling SSLv3.

```
SSL_CONF_cmd(ctx, "Protocol", "ALL,-SSLv3");
```

Only enable TLSv1.2:

```
SSL_CONF_cmd(ctx, "MinProtocol", "TLSv1.2");
```

```
SSL_CONF_cmd(ctx, "MaxProtocol", "TLSv1.2");
```

This also only enables TLSv1.2:

```
SSL_CONF_cmd(ctx, "Protocol", "-ALL,TLSv1.2");
```

Disable TLS session tickets:

```
SSL_CONF_cmd(ctx, "Options", "-SessionTicket");
```

Enable compression:

```
SSL_CONF_cmd(ctx, "Options", "Compression");
```

Set supported curves to P-256, P-384:

```
SSL_CONF_cmd(ctx, "Curves", "P-256:P-384");
```

## SEE ALSO

`ssl(7)`, `SSL_CONF_CTX_new(3)`, `SSL_CONF_CTX_set_flags(3)`, `SSL_CONF_CTX_set1_prefix(3)`, `SSL_CONF_CTX_set_ssl_ctx(3)`, `SSL_CONF_cmd_argv(3)`, `SSL_CTX_set_options(3)`

## HISTORY

The `SSL_CONF_cmd()` function was added in OpenSSL 1.0.2.

The `SSL_OP_NO_SSL2` option doesn't have effect since 1.1.0, but the macro is retained for backwards compatibility.

The `SSL_CONF_TYPE_NONE` was added in OpenSSL 1.1.0. In earlier versions of OpenSSL passing a command which didn't take an argument would return `SSL_CONF_TYPE_UNKNOWN`.

`MinProtocol` and `MaxProtocol` were added in OpenSSL 1.1.0.

`AllowNoDHEKEX` and `PrioritizeChaCha` were added in OpenSSL 1.1.1.

The `UnsafeLegacyServerConnect` option is no longer set by default from OpenSSL 3.0.

## COPYRIGHT

Copyright 2012-2022 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>.