Apache HTTP Server Version 2.4

Apache Module mod_ssl

Description:	Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
Status:	Extension
Module Identifier:	ssl_module
Source File:	mod_ssl.c

Summary

This module provides SSL v3 and TLS v1.x support for the Apache HTTP Server. SSL v2 is no longer supported.

This module relies on OpenSSL (↗ http://www.openssl.org/) to provide the cryptography engine.

Further details, discussion, and examples are provided in the SSL documentation (↗ ../ssl/) .

Environment Variables

This module can be configured to provide several items of SSL information as additional environment variables to the SSI and CGI namespace. Except for `HTTPS` and `SSL_TLS_SNI` which are always defined, this information is not provided by default for performance reasons. (See `SSLOptions StdEnvVars`, below) The generated variables are listed in the table below. For backward compatibility the information can be made available under different names, too. Look in the Compatibility (↗ ../ssl/ssl_compat.html) chapter for details on the compatibility variables.

Variable Name	Value Type	Description
`HTTPS`	flag	HTTPS is being used.
`SSL_PROTOCOL`	string	The SSL protocol version (SSLv3, TLSv1, TLSv1.1, TLSv1.2)
`SSL_SESSION_ID`	string	The hex-encoded SSL session id
`SSL_SESSION_RESUMED`	string	Initial or Resumed SSL Session. Note: multiple requests may be served over the same (Initial or Resumed) SSL session if HTTP KeepAlive is in use
`SSL_SECURE_RENEG`	string	`true` if secure renegotiation is supported, else `false`
`SSL_CIPHER`	string	The cipher specification name
`SSL_CIPHER_EXPORT`	string	`true` if cipher is an export cipher
`SSL_CIPHER_USEKEYSIZE`	number	Number of cipher bits (actually used)
`SSL_CIPHER_ALGKEYSIZE`	number	Number of cipher bits (possible)
`SSL_COMPRESS_METHOD`	string	SSL compression method negotiated
`SSL_VERSION_INTERFACE`	string	The mod_ssl program version
`SSL_VERSION_LIBRARY`	string	The OpenSSL program version
`SSL_CLIENT_M_VERSION`	string	The version of the client certificate
`SSL_CLIENT_M_SERIAL`	string	The serial of the client certificate
`SSL_CLIENT_S_DN`	string	Subject DN in client's certificate
`SSL_CLIENT_S_DN_`*x509*	string	Component of client's Subject DN
`SSL_CLIENT_SAN_Email_`*n*	string	Client certificate's subjectAltName extension entries of type rfc822Name
`SSL_CLIENT_SAN_DNS_`*n*	string	Client certificate's subjectAltName extension entries of type dNSName
`SSL_CLIENT_SAN_OTHER_msUPN_`*n*	string	Client certificate's subjectAltName extension entries of type otherName, Microsoft User Principal Name form (OID 1.3.6.1.4.1.311.20.2.3)
`SSL_CLIENT_I_DN`	string	Issuer DN of client's certificate
`SSL_CLIENT_I_DN_`*x509*	string	Component of client's Issuer DN
`SSL_CLIENT_V_START`	string	Validity of client's certificate (start time)
`SSL_CLIENT_V_END`	string	Validity of client's certificate (end time)
`SSL_CLIENT_V_REMAIN`	string	Number of days until client's certificate expires
`SSL_CLIENT_A_SIG`	string	Algorithm used for the signature of client's certificate
`SSL_CLIENT_A_KEY`	string	Algorithm used for the public key of client's certificate
`SSL_CLIENT_CERT`	string	PEM-encoded client certificate
`SSL_CLIENT_CERT_CHAIN_`*n*	string	PEM-encoded certificates in client certificate chain
`SSL_CLIENT_CERT_RFC4523_CEA`	string	Serial number and issuer of the certificate. The format matches that of the CertificateExactAssertion in RFC4523
`SSL_CLIENT_VERIFY`	string	`NONE`, `SUCCESS`, `GENEROUS` or `FAILED:`*reason*
`SSL_SERVER_M_VERSION`	string	The version of the server certificate
`SSL_SERVER_M_SERIAL`	string	The serial of the server certificate
`SSL_SERVER_S_DN`	string	Subject DN in server's certificate
`SSL_SERVER_SAN_Email_`*n*	string	Server certificate's subjectAltName extension entries of type rfc822Name
`SSL_SERVER_SAN_DNS_`*n*	string	Server certificate's subjectAltName extension entries of type dNSName
`SSL_SERVER_SAN_OTHER_dnsSRV_`*n*	string	Server certificate's subjectAltName extension entries of type otherName, SRVName form (OID 1.3.6.1.5.5.7.8.7, RFC 4985)
`SSL_SERVER_S_DN_`*x509*	string	Component of server's Subject DN
`SSL_SERVER_I_DN`	string	Issuer DN of server's certificate
`SSL_SERVER_I_DN_`*x509*	string	Component of server's Issuer DN
`SSL_SERVER_V_START`	string	Validity of server's certificate (start time)
`SSL_SERVER_V_END`	string	Validity of server's certificate (end time)
`SSL_SERVER_A_SIG`	string	Algorithm used for the signature of server's certificate
`SSL_SERVER_A_KEY`	string	Algorithm used for the public key of server's certificate
`SSL_SERVER_CERT`	string	PEM-encoded server certificate
`SSL_SRP_USER`	string	SRP username
`SSL_SRP_USERINFO`	string	SRP user info
`SSL_TLS_SNI`	string	Contents of the SNI TLS extension (if supplied with ClientHello)

x509 specifies a component of an X.509 DN; one of `C,ST,L,O,OU,CN,T,I,G,S,D,UID,Email`. In httpd 2.2.0 and later, *x509* may also include a numeric `_n` suffix. If the DN in question contains multiple attributes of the same name, this suffix is used as a zero-based index to select a particular attribute. For example, where the server certificate subject DN included two OU attributes, `SSL_SERVER_S_DN_OU_0` and `SSL_SERVER_S_DN_OU_1` could be used to reference each. A variable name without a `_n` suffix is equivalent to that name with a `_0` suffix; the first (or only) attribute. When the environment table is populated using the `StdEnvVars` option of the `SSLOptions` directive, the first (or only) attribute of any DN is added only under a non-suffixed name; i.e. no `_0` suffixed entries are added.

In httpd 2.4.32 and later, an optional *_RAW* suffix may be added to *x509* in a DN component, to suppress conversion of the attribute value to UTF-8. This must be placed after the index suffix (if any). For example, `SSL_SERVER_S_DN_OU_RAW` or `SSL_SERVER_S_DN_OU_0_RAW` could be used.

The format of the *_DN* variables has changed in Apache HTTPD 2.3.11. See the `LegacyDNStringFormat` option for `SSLOptions` for details.

`SSL_CLIENT_V_REMAIN` is only available in version 2.1 and later.

A number of additional environment variables can also be used in `SSLRequire` expressions, or in custom log formats:

```
HTTP_USER_AGENT         PATH_INFO             AUTH_TYPE
HTTP_REFERER            QUERY_STRING          SERVER_SOFTWARE
HTTP_COOKIE             REMOTE_HOST           API_VERSION
HTTP_FORWARDED          REMOTE_IDENT          TIME_YEAR
HTTP_HOST               IS_SUBREQ             TIME_MON
HTTP_PROXY_CONNECTION   DOCUMENT_ROOT         TIME_DAY
HTTP_ACCEPT             SERVER_ADMIN          TIME_HOUR
THE_REQUEST             SERVER_NAME           TIME_MIN
REQUEST_FILENAME        SERVER_PORT           TIME_SEC
REQUEST_METHOD          SERVER_PROTOCOL       TIME_WDAY
REQUEST_SCHEME          REMOTE_ADDR           TIME
REQUEST_URI             REMOTE_USER
```

In these contexts, two special formats can also be used:

`ENV:`*`variablename`*
: This will expand to the standard environment variable *variablename*.

`HTTP:`*`headername`*
: This will expand to the value of the request header with name *headername*.

Custom Log Formats

When `mod_ssl` is built into Apache or at least loaded (under DSO situation) additional functions exist for the Custom Log Format (↗ mod_log_config.html#formats) of `mod_log_config`. First there is an additional ``%{*varname*}x" eXtension format function which can be used to expand any variables provided by any module, especially those provided by mod_ssl which can you find in the above table.

For backward compatibility there is additionally a special ``%{*name*}c" cryptography format function provided. Information about this function is provided in the Compatibility (↗ ../ssl/ssl_compat.html) chapter.

Example

```
CustomLog "logs/ssl_request_log" "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
```

These formats even work without setting the `StdEnvVars` option of the `SSLOptions` directive.

Request Notes

`mod_ssl` sets "notes" for the request which can be used in logging with the `%{`*`name`*`}n` format string in `mod_log_config`.

The notes supported are as follows:

`ssl-access-forbidden`
: This note is set to the value `1` if access was denied due to an `SSLRequire` or `SSLRequireSSL` directive.

`ssl-secure-reneg`
: If `mod_ssl` is built against a version of OpenSSL which supports the secure renegotiation extension, this note is set to the value `1` if SSL is in used for the current connection, and the client also supports the secure renegotiation extension. If the client does not support the secure renegotiation extension, the note is set to the value `0`. If `mod_ssl` is not built against a version of OpenSSL which supports secure renegotiation, or if SSL is not in use for the current connection, the note is not set.

Expression Parser Extension

When `mod_ssl` is built into Apache or at least loaded (under DSO situation) any variables provided by `mod_ssl` can be used in expressions for the ap_expr Expression Parser (↗ ../expr.html) . The variables can be referenced using the syntax ``%{*varname*}". Starting with version 2.4.18 one can also use the `mod_rewrite` style syntax ``%{SSL:*varname*}" or the function style syntax ``ssl(*varname*)".

Example (using `mod_headers`)

```
Header set X-SSL-PROTOCOL "expr=%{SSL_PROTOCOL}"
Header set X-SSL-CIPHER "expr=%{SSL:SSL_CIPHER}"
```

This feature even works without setting the `StdEnvVars` option of the `SSLOptions` directive.

Authorization providers for use with Require

`mod_ssl` provides a few authentication providers for use with `mod_authz_core`'s `Require` directive.

Require ssl

The `ssl` provider denies access if a connection is not encrypted with SSL. This is similar to the `SSLRequireSSL` directive.

```
Require ssl
```

Require ssl-verify-client

The `ssl` provider allows access if the user is authenticated with a valid client certificate. This is only useful if `SSLVerifyClient optional` is in effect.

The following example grants access if the user is authenticated either with a client certificate or by username and password.

```
Require ssl-verify-client
Require valid-user
```

SSLCACertificateFile Directive

Description:	File of concatenated PEM-encoded CA Certificates for Client Auth
Syntax:	`SSLCACertificateFile` *`file-path`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to `SSLCACertificatePath`.

Example

```
SSLCACertificateFile "/usr/local/apache2/conf/ssl.crt/ca-bundle-client.crt"
```

SSLCACertificatePath Directive

Description:	Directory of PEM-encoded CA Certificates for Client Auth
Syntax:	`SSLCACertificatePath` *`directory-path`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose clients you deal with. These are used to verify the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*`.N`. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLCACertificatePath "/usr/local/apache2/conf/ssl.crt/"
```

SSLCADNRequestFile Directive

Description:	File of concatenated PEM-encoded CA Certificates for defining acceptable CA names
Syntax:	`SSLCADNRequestFile` *`file-path`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

When a client certificate is requested by mod_ssl, a list of *acceptable Certificate Authority names* is sent to the client in the SSL handshake. These CA names can be used by the client to select an appropriate client certificate out of those it has available.

If neither of the directives `SSLCADNRequestPath` or `SSLCADNRequestFile` are given, then the set of acceptable CA names sent to the client is the names of all the CA certificates given by the `SSLCACertificateFile` and `SSLCACertificatePath` directives; in other words, the names of the CAs which will actually be used to verify the client certificate.

In some circumstances, it is useful to be able to send a set of acceptable CA names which differs from the actual CAs used to verify the client certificate - for example, if the client certificates are signed by intermediate CAs. In such cases, `SSLCADNRequestPath` and/or `SSLCADNRequestFile` can be used; the acceptable CA names are then taken from the complete set of certificates in the directory and/or file specified by this pair of directives.

`SSLCADNRequestFile` must specify an *all-in-one* file containing a concatenation of PEM-encoded CA certificates.

Example

```
SSLCADNRequestFile "/usr/local/apache2/conf/ca-names.crt"
```

SSLCADNRequestPath Directive

Description:	Directory of PEM-encoded CA Certificates for defining acceptable CA names
Syntax:	`SSLCADNRequestPath` *`directory-path`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This optional directive can be used to specify the set of *acceptable CA names* which will be sent to the client when a client certificate is requested. See the `SSLCADNRequestFile` directive for more details.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*`.N`. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLCADNRequestPath "/usr/local/apache2/conf/ca-names.crt/"
```

SSLCARevocationCheck Directive

Description:	Enable CRL-based revocation checking
Syntax:	`SSLCARevocationCheck chain|leaf|none [flags ...]`
Default:	`SSLCARevocationCheck none`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Optional *flag*s available in httpd 2.4.21 or later

Enables certificate revocation list (CRL) checking. At least one of `SSLCARevocationFile` or `SSLCARevocationPath` must be configured. When set to `chain` (recommended setting), CRL checks are applied to all certificates in the chain, while setting it to `leaf` limits the checks to the end-entity cert.

The available *flag*s are:

- `no_crl_for_cert_ok`
  Prior to version 2.3.15, CRL checking in mod_ssl also succeeded when no CRL(s) for the checked certificate(s) were found in any of the locations configured with `SSLCARevocationFile` or `SSLCARevocationPath`.

  With the introduction of `SSLCARevocationFile`, the behavior has been changed: by default with `chain` or `leaf`, CRLs **must** be present for the validation to succeed - otherwise it will fail with an `"unable to get certificate CRL"` error.

  The *flag* `no_crl_for_cert_ok` allows to restore previous behaviour.

Example

```
SSLCARevocationCheck chain
```

Compatibility with versions 2.2

```
SSLCARevocationCheck chain no_crl_for_cert_ok
```

SSLCARevocationFile Directive

Description:	File of concatenated PEM-encoded CA CRLs for Client Auth
Syntax:	`SSLCARevocationFile file-path`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to `SSLCARevocationPath`.

Example

```
SSLCARevocationFile "/usr/local/apache2/conf/ssl.crl/ca-bundle-client.crl"
```

SSLCARevocationPath Directive

Description:	Directory of PEM-encoded CA CRLs for Client Auth
Syntax:	`SSLCARevocationPath directory-path`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose clients you deal with. These are used to revoke the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named *hash-value*`.rN`. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLCARevocationPath "/usr/local/apache2/conf/ssl.crl/"
```

SSLCertificateChainFile Directive

Description:	File of PEM-encoded Server CA Certificates
Syntax:	`SSLCertificateChainFile file-path`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

SSLCertificateChainFile is deprecated

`SSLCertificateChainFile` became obsolete with version 2.4.8, when `SSLCertificateFile` was extended to also load intermediate CA certificates from the server certificate file.

This directive sets the optional *all-in-one* file where you can assemble the certificates of Certification Authorities (CA) which form the certificate chain of the server certificate. This starts with the issuing CA certificate of the server certificate and can range up to the root CA certificate. Such a file is simply the concatenation of the various PEM-encoded CA Certificate files, usually in certificate chain order.

This should be used alternatively and/or additionally to `SSLCACertificatePath` for explicitly constructing the server certificate chain which is sent to the browser in addition to the server certificate. It is especially useful to avoid conflicts with CA certificates when using client authentication. Because although placing a CA certificate of the server certificate chain into `SSLCACertificatePath` has the same effect for the certificate chain construction, it has the side-effect that client certificates issued by this same CA certificate are also accepted on client authentication.

But be careful: Providing the certificate chain works only if you are using a *single* RSA *or* DSA based server certificate. If you are using a coupled RSA+DSA certificate pair, this will work only if actually both certificates use the *same* certificate chain. Else the browsers will be confused in this situation.

Example

```
SSLCertificateChainFile "/usr/local/apache2/conf/ssl.crt/ca.crt"
```

SSLCertificateFile Directive

Description:	Server PEM-encoded X.509 certificate data file or token identifier
Syntax:	`SSLCertificateFile` *`file-path|certid`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	*certid* available in 2.4.42 and later.

This directive points to a file with certificate data in PEM format, or the certificate identifier through a configured cryptographic token. If using a PEM file, at minimum, the file must include an end-entity (leaf) certificate. The directive can be used multiple times (referencing different filenames) to support multiple algorithms for server authentication - typically RSA, DSA, and ECC. The number of supported algorithms depends on the OpenSSL version being used for mod_ssl: with version 1.0.0 or later, `openssl list-public-key-algorithms` will output a list of supported algorithms, see also the note below about limitations of OpenSSL versions prior to 1.0.2 and the ways to work around them.

The files may also include intermediate CA certificates, sorted from leaf to root. This is supported with version 2.4.8 and later, and obsoletes `SSLCertificateChainFile`. When running with OpenSSL 1.0.2 or later, this allows to configure the intermediate CA chain on a per-certificate basis.

Custom DH parameters and an EC curve name for ephemeral keys, can also be added to end of the first file configured using `SSLCertificateFile`. This is supported in version 2.4.7 or later. Such parameters can be generated using the commands `openssl dhparam` and `openssl ecparam`. The parameters can be added as-is to the end of the first certificate file. Only the first file can be used for custom parameters, as they are applied independently of the authentication algorithm type.

Finally the end-entity certificate's private key can also be added to the certificate file instead of using a separate `SSLCertificateKeyFile` directive. This practice is highly discouraged. If it is used, the certificate files using such an embedded key must be configured after the certificates using a separate key file. If the private key is encrypted, the pass phrase dialog is forced at startup time.

As an alternative to storing certificates and private keys in files, a certificate identifier can be used to identify a certificate stored in a token. Currently, only PKCS#11 URIs (↗ https://tools.ietf.org/html/rfc7512) are recognized as certificate identifiers, and can be used in conjunction with the OpenSSL `pkcs11` engine. If `SSLCertificateKeyFile` is omitted, the certificate and private key can be loaded through the single identifier specified with `SSLCertificateFile`.

DH parameter interoperability with primes > 1024 bit

Beginning with version 2.4.7, mod_ssl makes use of standardized DH parameters with prime lengths of 2048, 3072 and 4096 bits and with additional prime lengths of 6144 and 8192 bits beginning with version 2.4.10 (from RFC 3526 (↗ http://www.ietf.org/rfc/rfc3526.txt)), and hands them out to clients based on the length of the certificate's RSA/DSA key. With Java-based clients in particular (Java 7 or earlier), this may lead to handshake failures - see this FAQ answer (↗ ../ssl/ssl_faq.html#javadh) for working around such issues.

Default DH parameters when using multiple certificates and OpenSSL versions prior to 1.0.2

When using multiple certificates to support different authentication algorithms (like RSA, DSA, but mainly ECC) and OpenSSL prior to 1.0.2, it is recommended to either use custom DH parameters (preferably) by adding them to the first certificate file (as described above), or to order the `SSLCertificateFile` directives such that RSA/DSA certificates are placed **after** the ECC one.
This is due to a limitation in older versions of OpenSSL which don't let the Apache HTTP Server determine the currently selected certificate at handshake time (when the DH parameters must be sent to the peer) but instead always provide the last configured certificate. Consequently, the server may select default DH parameters based on the length of the wrong certificate's key (ECC keys are much smaller than RSA/DSA ones and their length is not relevant for selecting DH primes).
Since custom DH parameters always take precedence over the default ones, this issue can be avoided by creating and configuring them (as described above), thus using a custom/suitable length.

Example

```
# Example using a PEM-encoded file.
SSLCertificateFile "/usr/local/apache2/conf/ssl.crt/server.crt"
# Example use of a certificate and private key from a PKCS#11 token:
SSLCertificateFile "pkcs11:token=My%20Token%20Name;id=45"
```

SSLCertificateKeyFile Directive

Description:	Server PEM-encoded private key file
Syntax:	`SSLCertificateKeyFile` *`file-path|keyid`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	*keyid* available in 2.4.42 and later.

This directive points to the PEM-encoded private key file for the server, or the key ID through a configured cryptographic token. If the contained private key is encrypted, the pass phrase dialog is forced at startup time.

The directive can be used multiple times (referencing different filenames) to support multiple algorithms for server authentication. For each `SSLCertificateKeyFile` directive, there must be a matching `SSLCertificateFile` directive.

The private key may also be combined with the certificate in the file given by `SSLCertificateFile`, but this practice is highly discouraged. If it is used, the certificate files using such an embedded key must be configured after the certificates using a separate key file.

As an alternative to storing private keys in files, a key identifier can be used to identify a private key stored in a token. Currently, only PKCS#11 URIs (↗ https://tools.ietf.org/html/rfc7512) are recognized as private key identifiers, and can be used in conjunction with the OpenSSL `pkcs11` engine.

Example

```
# To use a private key from a PEM-encoded file:
SSLCertificateKeyFile "/usr/local/apache2/conf/ssl.key/server.key"
# To use a private key from a PKCS#11 token:
SSLCertificateKeyFile "pkcs11:token=My%20Token%20Name;id=45"
```

SSLCipherSuite Directive

Description:	Cipher Suite available for negotiation in SSL handshake
Syntax:	`SSLCipherSuite [protocol] cipher-spec`
Default:	`SSLCipherSuite DEFAULT (depends on OpenSSL version)`
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This complex directive uses a colon-separated *cipher-spec* string consisting of OpenSSL cipher specifications to configure the Cipher Suite the client is permitted to negotiate in the SSL handshake phase. The optional protocol specifier can configure the Cipher Suite for a specific SSL version. Possible values include "SSL" for all SSL Protocols up to and including TLSv1.2.

Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured Cipher Suite after the HTTP request was read but before the HTTP response is sent.

If the SSL library supports TLSv1.3 (OpenSSL 1.1.1 and later), the protocol specifier "TLSv1.3" can be used to configure the cipher suites for that protocol. Since TLSv1.3 does not offer renegotiations, specifying ciphers for it in a directory context is not allowed.

For a list of TLSv1.3 cipher names, see the OpenSSL documentation (↗ https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_ciphersuites.html) .

An SSL cipher specification in *cipher-spec* is composed of 4 major attributes plus a few extra minor ones:

- *Key Exchange Algorithm*:
  RSA, Diffie-Hellman, Elliptic Curve Diffie-Hellman, Secure Remote Password
- *Authentication Algorithm*:
  RSA, Diffie-Hellman, DSS, ECDSA, or none.
- *Cipher/Encryption Algorithm*:
  AES, DES, Triple-DES, RC4, RC2, IDEA, etc.
- *MAC Digest Algorithm*:
  MD5, SHA or SHA1, SHA256, SHA384.

An SSL cipher can also be an export cipher. SSLv2 ciphers are no longer supported. To specify which ciphers to use, one can either specify all the Ciphers, one at a time, or use aliases to specify the preference and order for the ciphers (see Table 1 (↗ #table1)). The actually available ciphers and aliases depends on the used openssl version. Newer openssl versions may include additional ciphers.

Tag	Description
Key Exchange Algorithm:	
`kRSA`	RSA key exchange
`kDHr`	Diffie-Hellman key exchange with RSA key
`kDHd`	Diffie-Hellman key exchange with DSA key
`kEDH`	Ephemeral (temp.key) Diffie-Hellman key exchange (no cert)
`kSRP`	Secure Remote Password (SRP) key exchange
Authentication Algorithm:	
`aNULL`	No authentication
`aRSA`	RSA authentication
`aDSS`	DSS authentication
`aDH`	Diffie-Hellman authentication
Cipher Encoding Algorithm:	
`eNULL`	No encryption
`NULL`	alias for eNULL
`AES`	AES encryption
`DES`	DES encryption
`3DES`	Triple-DES encryption
`RC4`	RC4 encryption
`RC2`	RC2 encryption
`IDEA`	IDEA encryption
MAC Digest Algorithm:	
`MD5`	MD5 hash function
`SHA1`	SHA1 hash function
`SHA`	alias for SHA1
`SHA256`	SHA256 hash function
`SHA384`	SHA384 hash function
Aliases:	
`SSLv3`	all SSL version 3.0 ciphers
`TLSv1`	all TLS version 1.0 ciphers
`EXP`	all export ciphers
`EXPORT40`	all 40-bit export ciphers only
`EXPORT56`	all 56-bit export ciphers only
`LOW`	all low strength ciphers (no export, single DES)
`MEDIUM`	all ciphers with 128 bit encryption
`HIGH`	all ciphers using Triple-DES
`RSA`	all ciphers using RSA key exchange
`DH`	all ciphers using Diffie-Hellman key exchange
`EDH`	all ciphers using Ephemeral Diffie-Hellman key exchange

ECDH	Elliptic Curve Diffie-Hellman key exchange
ADH	all ciphers using Anonymous Diffie-Hellman key exchange
AECDH	all ciphers using Anonymous Elliptic Curve Diffie-Hellman key exchange
SRP	all ciphers using Secure Remote Password (SRP) key exchange
DSS	all ciphers using DSS authentication
ECDSA	all ciphers using ECDSA authentication
aNULL	all ciphers using no authentication

Now where this becomes interesting is that these can be put together to specify the order and ciphers you wish to use. To speed this up there are also aliases (`SSLv3, TLSv1, EXP, LOW, MEDIUM, HIGH`) for certain groups of ciphers. These tags can be joined together with prefixes to form the *cipher-spec*. Available prefixes are:

- none: add cipher to list
- `+`: move matching ciphers to the current location in list
- `-`: remove cipher from list (can be added later again)
- `!`: kill cipher from list completely (can **not** be added later again)

> **`aNULL`, `eNULL` and `EXP` ciphers are always disabled**
>
> Beginning with version 2.4.7, null and export-grade ciphers are always disabled, as mod_ssl unconditionally adds `!aNULL:!eNULL:!EXP` to any cipher string at initialization.

A simpler way to look at all of this is to use the "`openssl ciphers -v`" command which provides a nice way to successively create the correct *cipher-spec* string. The default *cipher-spec* string depends on the version of the OpenSSL libraries used. Let's suppose it is "`RC4-SHA:AES128-SHA:HIGH:MEDIUM:!aNULL:!MD5`" which means the following: Put `RC4-SHA` and `AES128-SHA` at the beginning. We do this, because these ciphers offer a good compromise between speed and security. Next, include high and medium security ciphers. Finally, remove all ciphers which do not authenticate, i.e. for SSL the Anonymous Diffie-Hellman ciphers, as well as all ciphers which use `MD5` as hash algorithm, because it has been proven insufficient.

```
$ openssl ciphers -v 'RC4-SHA:AES128-SHA:HIGH:MEDIUM:!aNULL:!MD5'
RC4-SHA                 SSLv3 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=SHA1
AES128-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
...                     ...               ...     ...           ...
SEED-SHA                SSLv3 Kx=RSA      Au=RSA  Enc=SEED(128) Mac=SHA1
PSK-RC4-SHA             SSLv3 Kx=PSK      Au=PSK  Enc=RC4(128)  Mac=SHA1
KRB5-RC4-SHA            SSLv3 Kx=KRB5     Au=KRB5 Enc=RC4(128)  Mac=SHA1
```

The complete list of particular RSA & DH ciphers for SSL is given in Table 2 (↗ #table2) .

Example

```
SSLCipherSuite RSA:!EXP:!NULL:+HIGH:+MEDIUM:-LOW
```

Cipher-Tag	Protocol	Key Ex.	Auth.	Enc.	MAC	Type
RSA Ciphers:						
DES-CBC3-SHA	SSLv3	RSA	RSA	3DES(168)	SHA1	
IDEA-CBC-SHA	SSLv3	RSA	RSA	IDEA(128)	SHA1	
RC4-SHA	SSLv3	RSA	RSA	RC4(128)	SHA1	
RC4-MD5	SSLv3	RSA	RSA	RC4(128)	MD5	
DES-CBC-SHA	SSLv3	RSA	RSA	DES(56)	SHA1	
EXP-DES-CBC-SHA	SSLv3	RSA(512)	RSA	DES(40)	SHA1	export
EXP-RC2-CBC-MD5	SSLv3	RSA(512)	RSA	RC2(40)	MD5	export
EXP-RC4-MD5	SSLv3	RSA(512)	RSA	RC4(40)	MD5	export
NULL-SHA	SSLv3	RSA	RSA	None	SHA1	
NULL-MD5	SSLv3	RSA	RSA	None	MD5	
Diffie-Hellman Ciphers:						
ADH-DES-CBC3-SHA	SSLv3	DH	None	3DES(168)	SHA1	
ADH-DES-CBC-SHA	SSLv3	DH	None	DES(56)	SHA1	
ADH-RC4-MD5	SSLv3	DH	None	RC4(128)	MD5	
EDH-RSA-DES-CBC3-SHA	SSLv3	DH	RSA	3DES(168)	SHA1	
EDH-DSS-DES-CBC3-SHA	SSLv3	DH	DSS	3DES(168)	SHA1	
EDH-RSA-DES-CBC-SHA	SSLv3	DH	RSA	DES(56)	SHA1	
EDH-DSS-DES-CBC-SHA	SSLv3	DH	DSS	DES(56)	SHA1	
EXP-EDH-RSA-DES-CBC-SHA	SSLv3	DH(512)	RSA	DES(40)	SHA1	export
EXP-EDH-DSS-DES-CBC-SHA	SSLv3	DH(512)	DSS	DES(40)	SHA1	export
EXP-ADH-DES-CBC-SHA	SSLv3	DH(512)	None	DES(40)	SHA1	export
EXP-ADH-RC4-MD5	SSLv3	DH(512)	None	RC4(40)	MD5	export

SSLCompression Directive

Description:	Enable compression on the SSL level
Syntax:	`SSLCompression on|off`
Default:	`SSLCompression off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.3 and later, if using OpenSSL 0.9.8 or later; virtual host scope available if using OpenSSL 1.0.0 or later. The default used to be `on` in version 2.4.3.

This directive allows to enable compression on the SSL level.

Enabling compression causes security issues in most setups (the so called CRIME attack).

SSLCryptoDevice Directive

Description:	Enable use of a cryptographic hardware accelerator
Syntax:	`SSLCryptoDevice` *engine*
Default:	`SSLCryptoDevice builtin`
Context:	server config
Status:	Extension
Module:	mod_ssl

This directive enables use of a cryptographic hardware accelerator board to offload some of the SSL processing overhead. This directive can only be used if the SSL toolkit is built with "engine" support; OpenSSL 0.9.7 and later releases have "engine" support by default, the separate "-engine" releases of OpenSSL 0.9.6 must be used.

To discover which engine names are supported, run the command "`openssl engine`".

Example

```
# For a Broadcom accelerator:
SSLCryptoDevice ubsec
```

SSLEngine Directive

Description:	SSL Engine Operation Switch
Syntax:	`SSLEngine on|off|optional`
Default:	`SSLEngine off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive toggles the usage of the SSL/TLS Protocol Engine. This is should be used inside a `<VirtualHost>` section to enable SSL/TLS for a that virtual host. By default the SSL/TLS Protocol Engine is disabled for both the main server and all configured virtual hosts.

Example

```
<VirtualHost _default_:443>
SSLEngine on
#...
</VirtualHost>
```

In Apache 2.1 and later, `SSLEngine` can be set to `optional`. This enables support for RFC 2817 (↗ http://www.ietf.org/rfc/rfc2817.txt) , Upgrading to TLS Within HTTP/1.1. At this time no web browsers support RFC 2817.

SSLFIPS Directive

Description:	SSL FIPS mode Switch
Syntax:	`SSLFIPS on|off`
Default:	`SSLFIPS off`
Context:	server config
Status:	Extension
Module:	mod_ssl

This directive toggles the usage of the SSL library FIPS_mode flag. It must be set in the global server context and cannot be configured with conflicting settings (SSLFIPS on followed by SSLFIPS off or similar). The mode applies to all SSL library operations.

If httpd was compiled against an SSL library which did not support the FIPS_mode flag, `SSLFIPS on` will fail. Refer to the FIPS 140-2 Security Policy document of the SSL provider library for specific requirements to use mod_ssl in a FIPS 140-2 approved mode of operation; note that mod_ssl itself is not validated, but may be described as using FIPS 140-2 validated cryptographic module, when all components are assembled and operated under the guidelines imposed by the applicable Security Policy.

SSLHonorCipherOrder Directive

Description:	Option to prefer the server's cipher preference order
Syntax:	`SSLHonorCipherOrder on|off`
Default:	`SSLHonorCipherOrder off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

When choosing a cipher during an SSLv3 or TLSv1 handshake, normally the client's preference is used. If this directive is enabled, the server's preference will be used instead.

Example

```
SSLHonorCipherOrder on
```

SSLInsecureRenegotiation Directive

Description:	Option to enable support for insecure renegotiation
Syntax:	`SSLInsecureRenegotiation on|off`
Default:	`SSLInsecureRenegotiation off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.2.15 and later, if using OpenSSL 0.9.8m or later

As originally specified, all versions of the SSL and TLS protocols (up to and including TLS/1.2) were vulnerable to a Man-in-the-Middle attack (CVE-2009-3555 (↗ http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2009-3555)) during a renegotiation. This vulnerability allowed an attacker to "prefix" a chosen plaintext to the HTTP request as seen by the web server. A protocol extension was developed which fixed this vulnerability if supported by both client and server.

If `mod_ssl` is linked against OpenSSL version 0.9.8m or later, by default renegotiation is only supported with clients supporting the new protocol extension. If this directive is enabled, renegotiation will be allowed with old (unpatched) clients, albeit insecurely.

Security warning

If this directive is enabled, SSL connections will be vulnerable to the Man-in-the-Middle prefix attack as described in CVE-2009-3555 (↗ http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2009-3555) .

Example

```
SSLInsecureRenegotiation on
```

The `SSL_SECURE_RENEG` environment variable can be used from an SSI or CGI script to determine whether secure renegotiation is supported for a given SSL connection.

SSLOCSPDefaultResponder Directive

Description:	Set the default responder URI for OCSP validation
Syntax:	`SSLOCSPDefaultResponder` *uri*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This option sets the default OCSP responder to use. If `SSLOCSPOverrideResponder` is not enabled, the URI given will be used only if no responder URI is specified in the certificate being verified.

SSLOCSPEnable Directive

Description:	Enable OCSP validation of the client certificate chain
Syntax:	`SSLOCSPEnable on|leaf|off`
Default:	`SSLOCSPEnable off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Mode *leaf* available in httpd 2.4.34 and later

This option enables OCSP validation of the client certificate chain. If this option is enabled, certificates in the client's certificate chain will be validated against an OCSP responder after normal verification (including CRL checks) have taken place. In mode 'leaf', only the client certificate itself will be validated.

The OCSP responder used is either extracted from the certificate itself, or derived by configuration; see the `SSLOCSPDefaultResponder` and `SSLOCSPOverrideResponder` directives.

Example

```
SSLVerifyClient on
SSLOCSPEnable on
SSLOCSPDefaultResponder "http://responder.example.com:8888/responder"
SSLOCSPOverrideResponder on
```

SSLOCSPNoverify Directive

Description:	skip the OCSP responder certificates verification
Syntax:	`SSLOCSPNoverify on|off`
Default:	`SSLOCSPNoverify off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.26 and later, if using OpenSSL 0.9.7 or later

Skip the OCSP responder certificates verification, mostly useful when testing an OCSP server.

SSLOCSPOverrideResponder Directive

Description:	Force use of the default responder URI for OCSP validation
Syntax:	`SSLOCSPOverrideResponder on|off`
Default:	`SSLOCSPOverrideResponder off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This option forces the configured default OCSP responder to be used during OCSP certificate validation, regardless of whether the certificate being validated references an OCSP responder.

SSLOCSPProxyURL Directive

Description:	Proxy URL to use for OCSP requests
Syntax:	`SSLOCSPProxyURL` *url*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.19 and later

This option allows to set the URL of a HTTP proxy that should be used for all queries to OCSP responders.

SSLOCSPResponderCertificateFile Directive

Description:	Set of trusted PEM encoded OCSP responder certificates
Syntax:	`SSLOCSPResponderCertificateFile` *`file`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.26 and later, if using OpenSSL 0.9.7 or later

This supplies a list of trusted OCSP responder certificates to be used during OCSP responder certificate validation. The supplied certificates are implicitly trusted without any further validation. This is typically used where the OCSP responder certificate is self signed or omitted from the OCSP response.

SSLOCSPResponderTimeout Directive

Description:	Timeout for OCSP queries
Syntax:	`SSLOCSPResponderTimeout` *`seconds`*
Default:	`SSLOCSPResponderTimeout 10`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This option sets the timeout for queries to OCSP responders, when `SSLOCSPEnable` is turned on.

SSLOCSPResponseMaxAge Directive

Description:	Maximum allowable age for OCSP responses
Syntax:	`SSLOCSPResponseMaxAge` *`seconds`*
Default:	`SSLOCSPResponseMaxAge -1`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This option sets the maximum allowable age ("freshness") for OCSP responses. The default value (`-1`) does not enforce a maximum age, which means that OCSP responses are considered valid as long as their `nextUpdate` field is in the future.

SSLOCSPResponseTimeSkew Directive

Description:	Maximum allowable time skew for OCSP response validation
Syntax:	`SSLOCSPResponseTimeSkew` *`seconds`*
Default:	`SSLOCSPResponseTimeSkew 300`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This option sets the maximum allowable time skew for OCSP responses (when checking their `thisUpdate` and `nextUpdate` fields).

SSLOCSPUseRequestNonce Directive

Description:	Use a nonce within OCSP queries
Syntax:	`SSLOCSPUseRequestNonce on|off`
Default:	`SSLOCSPUseRequestNonce on`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.10 and later

This option determines whether queries to OCSP responders should contain a nonce or not. By default, a query nonce is always used and checked against the response's one. When the responder does not use nonces (e.g. Microsoft OCSP Responder), this option should be turned `off`.

SSLOpenSSLConfCmd Directive

Description:	Configure OpenSSL parameters through its *SSL_CONF* API
Syntax:	`SSLOpenSSLConfCmd` *`command-name command-value`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.8 and later, if using OpenSSL 1.0.2 or later

This directive exposes OpenSSL's *SSL_CONF* API to mod_ssl, allowing a flexible configuration of OpenSSL parameters without the need of implementing additional `mod_ssl` directives when new features are added to OpenSSL.

The set of available `SSLOpenSSLConfCmd` commands depends on the OpenSSL version being used for `mod_ssl` (at least version 1.0.2 is required). For a list of supported command names, see the section *Supported configuration file commands* in the SSL_CONF_cmd(3) (↗ http://www.openssl.org/docs/man1.0.2/ssl/SSL_CONF_cmd.html#SUPPORTED-CONFIGURATION-FILE-COMMANDS) manual page for OpenSSL.

Some of the `SSLOpenSSLConfCmd` commands can be used as an alternative to existing directives (such as `SSLCipherSuite` or `SSLProtocol`), though it should be noted that the syntax / allowable values for the parameters may sometimes differ.

Examples

```
SSLOpenSSLConfCmd Options -SessionTicket,ServerPreference
SSLOpenSSLConfCmd ECDHParameters brainpoolP256r1
SSLOpenSSLConfCmd ServerInfoFile "/usr/local/apache2/conf/server-info.pem"
SSLOpenSSLConfCmd Protocol "-ALL, TLSv1.2"
SSLOpenSSLConfCmd SignatureAlgorithms RSA+SHA384:ECDSA+SHA256
```

SSLOptions Directive

Description:	Configure various SSL engine run-time options
Syntax:	`SSLOptions [+|-]`*`option`* `...`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Extension
Module:	mod_ssl

This directive can be used to control various run-time options on a per-directory basis. Normally, if multiple `SSLOptions` could apply to a directory, then the most specific one is taken completely; the options are not merged. However if *all* the options on the `SSLOptions` directive are preceded by a plus (`+`) or minus (`-`) symbol, the options are merged. Any options preceded by a `+` are added to the options currently in force, and any options preceded by a `-` are removed from the options currently in force.

The available *option*s are:

- `StdEnvVars`
  When this option is enabled, the standard set of SSL related CGI/SSI environment variables are created. This per default is disabled for performance reasons, because the information extraction step is a rather expensive operation. So one usually enables this option for CGI and SSI requests only.

- `ExportCertData`
  When this option is enabled, additional CGI/SSI environment variables are created: `SSL_SERVER_CERT`, `SSL_CLIENT_CERT` and `SSL_CLIENT_CERT_CHAIN_`*n* (with *n* = 0,1,2,..). These contain the PEM-encoded X.509 Certificates of server and client for the current HTTPS connection and can be used by CGI scripts for deeper Certificate checking. Additionally all other certificates of the client certificate chain are provided, too. This bloats up the environment a little bit which is why you have to use this option to enable it on demand.

- `FakeBasicAuth`
  When this option is enabled, the Subject Distinguished Name (DN) of the Client X509 Certificate is translated into a HTTP Basic Authorization username. This means that the standard Apache authentication methods can be used for access control. The user name is just the Subject of the Client's X509 Certificate (can be determined by running OpenSSL's `openssl x509` command: `openssl x509 -noout -subject -in` *certificate*`.crt`). Note that no password is obtained from the user. Every entry in the user file needs this password: "`xxj31ZMTZzkVA`", which is the DES-encrypted version of the word "`password`". Those who live under MD5-based encryption (for instance under FreeBSD or BSD/OS, etc.) should use the following MD5 hash of the same word: "`$1$OXLyS...$Owx8s2/m9/gfkcRVXzgoE/`".

  Note that the `AuthBasicFake` directive within `mod_auth_basic` can be used as a more general mechanism for faking basic authentication, giving control over the structure of both the username and password.

- `StrictRequire`
  This *forces* forbidden access when `SSLRequireSSL` or `SSLRequire` successfully decided that access should be forbidden. Usually the default is that in the case where a "`Satisfy any`" directive is used, and other access restrictions are passed, denial of access due to `SSLRequireSSL` or `SSLRequire` is overridden (because that's how the Apache `Satisfy` mechanism should work.) But for strict access restriction you can use `SSLRequireSSL` and/or `SSLRequire` in combination with an "`SSLOptions +StrictRequire`". Then an additional "`Satisfy Any`" has no chance once mod_ssl has decided to deny access.

- `OptRenegotiate`
  This enables optimized SSL connection renegotiation handling when SSL directives are used in per-directory context. By default a strict scheme is enabled where *every* per-directory reconfiguration of SSL parameters causes a *full* SSL renegotiation handshake. When this option is used mod_ssl tries to avoid unnecessary handshakes by doing more granular (but still safe) parameter checks. Nevertheless these granular checks sometimes may not be what the user expects, so enable this on a per-directory basis only, please.

- `LegacyDNStringFormat`
  This option influences how values of the `SSL_{CLIENT,SERVER}_{I,S}_DN` variables are formatted. Since version 2.3.11, Apache HTTPD uses a RFC 2253 compatible format by default. This uses commas as delimiters between the attributes, allows the use of non-ASCII characters (which are converted to UTF8), escapes various special characters with backslashes, and sorts the attributes with the "C" attribute last.

  If `LegacyDNStringFormat` is set, the old format will be used which sorts the "C" attribute first, uses slashes as separators, and does not handle non-ASCII and special characters in any consistent way.

Example

```
SSLOptions +FakeBasicAuth -StrictRequire
<Files ~ "\.(cgi|shtml)$">
    SSLOptions +StdEnvVars -ExportCertData
</Files>
```

SSLPassPhraseDialog Directive

Description:	Type of pass phrase dialog for encrypted private keys
Syntax:	`SSLPassPhraseDialog` *`type`*
Default:	`SSLPassPhraseDialog builtin`
Context:	server config
Status:	Extension
Module:	mod_ssl

When Apache starts up it has to read the various Certificate (see `SSLCertificateFile`) and Private Key (see `SSLCertificateKeyFile`) files of the SSL-enabled virtual servers. Because for security reasons the Private Key files are usually encrypted, mod_ssl needs to query the administrator for a Pass Phrase in order to decrypt those files. This query can be done in two ways which can be configured by *type*:

- `builtin`
  This is the default where an interactive terminal dialog occurs at startup time just before Apache detaches from the terminal. Here the administrator has to manually enter the Pass Phrase for each encrypted Private Key file. Because a lot of SSL-enabled virtual hosts can be configured, the following reuse-scheme is used to minimize the dialog: When a Private Key file is encrypted, all known Pass Phrases (at the beginning there are none, of course) are tried. If one of those known Pass Phrases succeeds no dialog pops up for this particular Private Key file. If none succeeded, another Pass Phrase is queried on the terminal and remembered for the next round (where it perhaps can be reused).

  This scheme allows mod_ssl to be maximally flexible (because for N encrypted Private Key files you *can* use N different Pass Phrases - but then you have to enter all of them, of course) while minimizing the terminal dialog (i.e. when you use a single Pass Phrase for all N Private Key files this Pass Phrase is queried only once).

- `|/path/to/program [args...]`

This mode allows an external program to be used which acts as a pipe to a particular input device; the program is sent the standard prompt text used for the `builtin` mode on `stdin`, and is expected to write password strings on `stdout`. If several passwords are needed (or an incorrect password is entered), additional prompt text will be written subsequent to the first password being returned, and more passwords must then be written back.

- `exec:/path/to/program`
  Here an external program is configured which is called at startup for each encrypted Private Key file. It is called with two arguments (the first is of the form ``servername:portnumber`", the second is either ``RSA", ``DSA", ``ECC" or an integer index starting at 3 if more than three keys are configured), which indicate for which server and algorithm it has to print the corresponding Pass Phrase to `stdout`. In versions 2.4.8 (unreleased) and 2.4.9, it is called with one argument, a string of the form ``servername:portnumber:index" (with `index` being a zero-based integer number), which indicate the server, TCP port and certificate number. The intent is that this external program first runs security checks to make sure that the system is not compromised by an attacker, and only when these checks were passed successfully it provides the Pass Phrase.

  Both these security checks, and the way the Pass Phrase is determined, can be as complex as you like. Mod_ssl just defines the interface: an executable program which provides the Pass Phrase on `stdout`. Nothing more or less! So, if you're really paranoid about security, here is your interface. Anything else has to be left as an exercise to the administrator, because local security requirements are so different.

  The reuse-algorithm above is used here, too. In other words: The external program is called only once per unique Pass Phrase.

Example

```
SSLPassPhraseDialog "exec:/usr/local/apache/sbin/pp-filter"
```

SSLProtocol Directive

Description:	Configure usable SSL/TLS protocol versions
Syntax:	`SSLProtocol [+|-]`*`protocol`* `...`
Default:	`SSLProtocol all -SSLv3 (up to 2.4.16: all)`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive can be used to control which versions of the SSL/TLS protocol will be accepted in new connections.

The available (case-insensitive) *protocol*s are:

- `SSLv3`
  This is the Secure Sockets Layer (SSL) protocol, version 3.0, from the Netscape Corporation. It is the successor to SSLv2 and the predecessor to TLSv1, but is deprecated in RFC 7568 (↗ http://www.ietf.org/rfc/rfc7568.txt) .

- `TLSv1`
  This is the Transport Layer Security (TLS) protocol, version 1.0. It is the successor to SSLv3 and is defined in RFC 2246 (↗ http://www.ietf.org/rfc/rfc2246.txt) . It is supported by nearly every client.

- `TLSv1.1` (when using OpenSSL 1.0.1 and later)
  A revision of the TLS 1.0 protocol, as defined in RFC 4346 (↗ http://www.ietf.org/rfc/rfc4346.txt) .

- `TLSv1.2` (when using OpenSSL 1.0.1 and later)
  A revision of the TLS 1.1 protocol, as defined in RFC 5246 (↗ http://www.ietf.org/rfc/rfc5246.txt) .

- `TLSv1.3` (when using OpenSSL 1.1.1 and later)
  A new version of the TLS protocol, as defined in RFC 8446 (↗ http://www.ietf.org/rfc/rfc8446.txt) .

- `all`
  This is a shortcut for ``+SSLv3 +TLSv1" or - when using OpenSSL 1.0.1 and later - ``+SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2", respectively (except for OpenSSL versions compiled with the ``no-ssl3" configuration option, where `all` does not include `+SSLv3`).

Example

```
SSLProtocol TLSv1
```

`SSLProtocol` for name-based virtual hosts

Before OpenSSL 1.1.1, even though the Server Name Indication (SNI) allowed to determine the targeted virtual host early in the TLS handshake, it was not possible to switch the TLS protocol version of the connection at this point, and thus the `SSLProtocol` negotiated was always based off the one of the *base virtual host* (first virtual host declared on the listening `IP:port` of the connection).
Beginning with Apache HTTP server version 2.4.42, when built/linked against OpenSSL 1.1.1 or later, and when the SNI is provided by the client in the TLS handshake, the `SSLProtocol` of each (name-based) virtual host can and will be honored.
For compatibility with previous versions, if no `SSLProtocol` is configured in a name-based virtual host, the one from the base virtual host still applies, **unless** `SSLProtocol` is configured globally in which case the global value applies (this latter exception is more sensible than compatible, though).

SSLProxyCACertificateFile Directive

Description:	File of concatenated PEM-encoded CA Certificates for Remote Server Auth
Syntax:	`SSLProxyCACertificateFile` *`file-path`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose *remote servers* you deal with. These are used for Remote Server Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to `SSLProxyCACertificatePath`.

Example

```
SSLProxyCACertificateFile "/usr/local/apache2/conf/ssl.crt/ca-bundle-remote-server.crt"
```

SSLProxyCACertificatePath Directive

Description:	Directory of PEM-encoded CA Certificates for Remote Server Auth
Syntax:	`SSLProxyCACertificatePath` *`directory-path`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose remote servers you deal with. These are used to verify the remote server certificate on Remote Server Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*`.N`. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLProxyCACertificatePath "/usr/local/apache2/conf/ssl.crt/"
```

SSLProxyCARevocationCheck Directive

Description:	Enable CRL-based revocation checking for Remote Server Auth
Syntax:	`SSLProxyCARevocationCheck chain|leaf|none`
Default:	`SSLProxyCARevocationCheck none`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

Enables certificate revocation list (CRL) checking for the *remote servers* you deal with. At least one of `SSLProxyCARevocationFile` or `SSLProxyCARevocationPath` must be configured. When set to `chain` (recommended setting), CRL checks are applied to all certificates in the chain, while setting it to `leaf` limits the checks to the end-entity cert.

When set to `chain` or `leaf`, CRLs *must* be available for successful validation

Prior to version 2.3.15, CRL checking in mod_ssl also succeeded when no CRL(s) were found in any of the locations configured with `SSLProxyCARevocationFile` or `SSLProxyCARevocationPath`. With the introduction of this directive, the behavior has been changed: when checking is enabled, CRLs *must* be present for the validation to succeed - otherwise it will fail with an `"unable to get certificate CRL"` error.

Example

```
SSLProxyCARevocationCheck chain
```

SSLProxyCARevocationFile Directive

Description:	File of concatenated PEM-encoded CA CRLs for Remote Server Auth
Syntax:	`SSLProxyCARevocationFile` *`file-path`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose *remote servers* you deal with. These are used for Remote Server Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to `SSLProxyCARevocationPath`.

Example

```
SSLProxyCARevocationFile "/usr/local/apache2/conf/ssl.crl/ca-bundle-remote-server.crl"
```

SSLProxyCARevocationPath Directive

Description:	Directory of PEM-encoded CA CRLs for Remote Server Auth
Syntax:	`SSLProxyCARevocationPath` *`directory-path`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose remote servers you deal with. These are used to revoke the remote server certificate on Remote Server Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named *hash-value*`.rN`. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLProxyCARevocationPath "/usr/local/apache2/conf/ssl.crl/"
```

SSLProxyCheckPeerCN Directive

Description:	Whether to check the remote server certificate's CN field
Syntax:	`SSLProxyCheckPeerCN on|off`
Default:	`SSLProxyCheckPeerCN on`
Context:	server config, virtual host, proxy section
Status:	Extension

Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets whether the remote server certificate's CN field is compared against the hostname of the request URL. If both are not equal a 502 status code (Bad Gateway) is sent. `SSLProxyCheckPeerCN` is superseded by `SSLProxyCheckPeerName` in release 2.4.5 and later.

In all releases 2.4.5 through 2.4.20, setting `SSLProxyCheckPeerName off` was sufficient to enable this behavior (as the `SSLProxyCheckPeerCN` default was `on`.) In these releases, both directives must be set to `off` to completely avoid remote server certificate name validation. Many users reported this to be very confusing.

As of release 2.4.21, all configurations which enable either one of the `SSLProxyCheckPeerName` or `SSLProxyCheckPeerCN` options will use the new `SSLProxyCheckPeerName` behavior, and all configurations which disable either one of the `SSLProxyCheckPeerName` or `SSLProxyCheckPeerCN` options will suppress all remote server certificate name validation. Only the following configuration will trigger the legacy certificate CN comparison in 2.4.21 and later releases;

Example

```
SSLProxyCheckPeerCN on
SSLProxyCheckPeerName off
```

SSLProxyCheckPeerExpire Directive

Description:	Whether to check if remote server certificate is expired
Syntax:	`SSLProxyCheckPeerExpire on|off`
Default:	`SSLProxyCheckPeerExpire on`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets whether it is checked if the remote server certificate is expired or not. If the check fails a 502 status code (Bad Gateway) is sent.

Example

```
SSLProxyCheckPeerExpire on
```

SSLProxyCheckPeerName Directive

Description:	Configure host name checking for remote server certificates
Syntax:	`SSLProxyCheckPeerName on|off`
Default:	`SSLProxyCheckPeerName on`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	Apache HTTP Server 2.4.5 and later The proxy section context is allowed in httpd 2.4.30 and later

This directive configures host name checking for server certificates when mod_ssl is acting as an SSL client. The check will succeed if the host name from the request URI matches one of the CN attribute(s) of the certificate's subject, or matches the subjectAltName extension. If the check fails, the SSL request is aborted and a 502 status code (Bad Gateway) is returned.

Wildcard matching is supported for specific cases: an subjectAltName entry of type dNSName, or CN attributes starting with `*.` will match with any host name of the same number of name elements and the same suffix. E.g. `*.example.org` will match `foo.example.org`, but will not match `foo.bar.example.org`, because the number of elements in the respective host names differs.

This feature was introduced in 2.4.5 and superseded the behavior of the `SSLProxyCheckPeerCN` directive, which only tested the exact value in the first CN attribute against the host name. However, many users were confused by the behavior of using these directives individually, so the mutual behavior of `SSLProxyCheckPeerName` and `SSLProxyCheckPeerCN` directives were improved in release 2.4.21. See the `SSLProxyCheckPeerCN` directive description for the original behavior and details of these improvements.

SSLProxyCipherSuite Directive

Description:	Cipher Suite available for negotiation in SSL proxy handshake
Syntax:	`SSLProxyCipherSuite [`*`protocol`*`] `*`cipher-spec`*
Default:	`SSLProxyCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+EXP`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

Equivalent to `SSLCipherSuite`, but for the proxy connection. Please refer to `SSLCipherSuite` for additional information.

SSLProxyEngine Directive

Description:	SSL Proxy Engine Operation Switch
Syntax:	`SSLProxyEngine on|off`
Default:	`SSLProxyEngine off`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive toggles the usage of the SSL/TLS Protocol Engine for proxy. This is usually used inside a `<VirtualHost>` section to enable SSL/TLS for proxy usage in a particular virtual host. By default the SSL/TLS Protocol Engine is disabled for proxy both for the main server and all configured virtual hosts.

Note that the `SSLProxyEngine` directive should not, in general, be included in a virtual host that will be acting as a forward proxy (using `<Proxy>` or `ProxyRequests` directives). `SSLProxyEngine` is not required to enable a forward proxy server to proxy SSL/TLS requests.

Example

```
<VirtualHost _default_:443>
    SSLProxyEngine on
    #...
</VirtualHost>
```

SSLProxyMachineCertificateChainFile Directive

Description:	File of concatenated PEM-encoded CA certificates to be used by the proxy for choosing a certificate
Syntax:	`SSLProxyMachineCertificateChainFile` *`filename`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the all-in-one file where you keep the certificate chain for all of the client certs in use. This directive will be needed if the remote server presents a list of CA certificates that are not direct signers of one of the configured client certificates.

This referenced file is simply the concatenation of the various PEM-encoded certificate files. Upon startup, each client certificate configured will be examined and a chain of trust will be constructed.

Security warning

If this directive is enabled, all of the certificates in the file will be trusted as if they were also in `SSLProxyCACertificateFile`.

Example

```
SSLProxyMachineCertificateChainFile "/usr/local/apache2/conf/ssl.crt/proxyCA.pem"
```

SSLProxyMachineCertificateFile Directive

Description:	File of concatenated PEM-encoded client certificates and keys to be used by the proxy
Syntax:	`SSLProxyMachineCertificateFile` *`filename`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the all-in-one file where you keep the certificates and keys used for authentication of the proxy server to remote servers.

This referenced file is simply the concatenation of the various PEM-encoded certificate files. Use this directive alternatively or additionally to `SSLProxyMachineCertificatePath`. The referenced file can contain any number of pairs of client certificate and associated private key. Each pair can be specified in either (certificate, key) or (key, certificate) order. If the file includes any non-leaf certificate, or any unmatched key and certificate pair, a configuration error will be issued at startup.

When challenged to provide a client certificate by a remote server, the server should provide a list of *acceptable certificate authority names* in the challenge. If such a list is *not* provided, `mod_ssl` will use the first configured client cert/key. If a list of CA names *is* provided, `mod_ssl` will iterate through that list, and attempt to find a configured client cert which was issued either directly by that CA, or indirectly via any number of intermediary CA certificates. The chain of intermediate CA certificates can be built from those configured with `SSLProxyMachineCertificateChainFile`. The first configured matching certificate will then be supplied in response to the challenge.

If the list of CA names *is* provided by the remote server, and *no* matching client certificate can be found, no client certificate will be provided by `mod_ssl`, which will likely fail the SSL/TLS handshake (depending on the remote server configuration).

Currently there is no support for encrypted private keys

Only keys encoded in PKCS1 RSA, DSA or EC format are supported. Keys encoded in PKCS8 format, ie. starting with "`-----BEGIN PRIVATE KEY-----`", must be converted, eg. using "`openssl rsa -in private-pkcs8.pem -outform pem`".

Example

```
SSLProxyMachineCertificateFile "/usr/local/apache2/conf/ssl.crt/proxy.pem"
```

SSLProxyMachineCertificatePath Directive

Description:	Directory of PEM-encoded client certificates and keys to be used by the proxy
Syntax:	`SSLProxyMachineCertificatePath` *`directory`*
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets the directory where you keep the client certificates and keys used for authentication of the proxy server to remote servers.

mod_ssl will attempt to load every file inside the specified directory as if it was configured individually with `SSLProxyMachineCertificateFile`.

Currently there is no support for encrypted private keys

Only keys encoded in PKCS1 RSA, DSA or EC format are supported. Keys encoded in PKCS8 format, ie. starting with "`-----BEGIN PRIVATE KEY-----`", must be converted, eg. using "`openssl rsa -in private-pkcs8.pem -outform pem`".

Example

```
SSLProxyMachineCertificatePath "/usr/local/apache2/conf/proxy.crt/"
```

SSLProxyProtocol Directive

Description:	Configure usable SSL protocol flavors for proxy usage
Syntax:	`SSLProxyProtocol [+|-]`*`protocol`* `...`
Default:	`SSLProxyProtocol all -SSLv3 (up to 2.4.16: all)`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive can be used to control the SSL protocol flavors mod_ssl should use when establishing its server environment for proxy . It will only connect to servers using one of the provided protocols.

Please refer to `SSLProtocol` for additional information.

SSLProxyVerify Directive

Description:	Type of remote server Certificate verification
Syntax:	`SSLProxyVerify` *`level`*
Default:	`SSLProxyVerify none`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

When a proxy is configured to forward requests to a remote SSL server, this directive can be used to configure certificate verification of the remote server.

The following levels are available for *level*:

- **none**: no remote server Certificate is required at all
- **optional**: the remote server *may* present a valid Certificate
- **require**: the remote server *has to* present a valid Certificate
- **optional_no_ca**: the remote server may present a valid Certificate but it need not to be (successfully) verifiable.

In practice only levels **none** and **require** are really interesting, because level **optional** doesn't work with all servers and level **optional_no_ca** is actually against the idea of authentication (but can be used to establish SSL test pages, etc.)

Example

```
SSLProxyVerify require
```

SSLProxyVerifyDepth Directive

Description:	Maximum depth of CA Certificates in Remote Server Certificate verification
Syntax:	`SSLProxyVerifyDepth` *`number`*
Default:	`SSLProxyVerifyDepth 1`
Context:	server config, virtual host, proxy section
Status:	Extension
Module:	mod_ssl
Compatibility:	The proxy section context is allowed in httpd 2.4.30 and later

This directive sets how deeply mod_ssl should verify before deciding that the remote server does not have a valid certificate.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the remote server certificate. A depth of 0 means that self-signed remote server certificates are accepted only, the default depth of 1 means the remote server certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under `SSLProxyCACertificatePath`), etc.

Example

```
SSLProxyVerifyDepth 10
```

SSLRandomSeed Directive

Description:	Pseudo Random Number Generator (PRNG) seeding source
Syntax:	`SSLRandomSeed` *`context source`* `[`*`bytes`*`]`
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures one or more sources for seeding the Pseudo Random Number Generator (PRNG) in OpenSSL at startup time (*context* is `startup`) and/or just before a new SSL connection is established (*context* is `connect`). This directive can only be used in the global server context because the PRNG is a global facility.

The following *source* variants are available:

- `builtin`
  This is the always available builtin seeding source. Its usage consumes minimum CPU cycles under runtime and hence can be always used without drawbacks. The source used for seeding the PRNG contains of the current time, the current process id and a randomly chosen 128 bytes extract of the stack. The drawback is that this is not really a strong source and at startup time (where the scoreboard is still not available) this source just produces a few bytes of entropy. So you should always, at least for the startup, use an additional seeding source.

- `file:/path/to/source`
  This variant uses an external file `/path/to/source` as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of the file form the entropy (and *bytes* is given to `/path/to/source` as the first argument). When *bytes* is not specified the whole file forms the entropy (and `0` is given to

`/path/to/source` as the first argument). Use this especially at startup time, for instance with an available `/dev/random` and/or `/dev/urandom` devices (which usually exist on modern Unix derivatives like FreeBSD and Linux).

But be careful: Usually `/dev/random` provides only as much entropy data as it actually has, i.e. when you request 512 bytes of entropy, but the device currently has only 100 bytes available two things can happen: On some platforms you receive only the 100 bytes while on other platforms the read blocks until enough bytes are available (which can take a long time). Here using an existing `/dev/urandom` is better, because it never blocks and actually gives the amount of requested data. The drawback is just that the quality of the received data may not be the best.

- `exec:/path/to/program`
  This variant uses an external executable `/path/to/program` as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of its `stdout` contents form the entropy. When *bytes* is not specified, the entirety of the data produced on `stdout` form the entropy. Use this only at startup time when you need a very strong seeding with the help of an external program (for instance as in the example above with the `truerand` utility you can find in the mod_ssl distribution which is based on the AT&T *truerand* library). Using this in the connection context slows down the server too dramatically, of course. So usually you should avoid using external programs in that context.

- `egd:/path/to/egd-socket` (Unix only)
  This variant uses the Unix domain socket of the external Entropy Gathering Daemon (EGD) (see http://www.lothar.com/tech /crypto/ (↗ http://www.lothar.com/tech/crypto/)) to seed the PRNG. Use this if no random device exists on your platform.

Example

```
SSLRandomSeed startup builtin
SSLRandomSeed startup "file:/dev/random"
SSLRandomSeed startup "file:/dev/urandom" 1024
SSLRandomSeed startup "exec:/usr/local/bin/truerand" 16
SSLRandomSeed connect builtin
SSLRandomSeed connect "file:/dev/random"
SSLRandomSeed connect "file:/dev/urandom" 1024
```

SSLRenegBufferSize Directive

Description:	Set the size for the SSL renegotiation buffer
Syntax:	`SSLRenegBufferSize` *bytes*
Default:	`SSLRenegBufferSize 131072`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

If an SSL renegotiation is required in per-location context, for example, any use of `SSLVerifyClient` in a Directory or Location block, then `mod_ssl` must buffer any HTTP request body into memory until the new SSL handshake can be performed. This directive can be used to set the amount of memory that will be used for this buffer.

Note that in many configurations, the client sending the request body will be untrusted so a denial of service attack by consumption of memory must be considered when changing this configuration setting.

Example

```
SSLRenegBufferSize 262144
```

SSLRequire Directive

Description:	Allow access only when an arbitrarily complex boolean expression is true
Syntax:	`SSLRequire` *expression*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

SSLRequire is deprecated

`SSLRequire` is deprecated and should in general be replaced by Require expr (↗ mod_authz_core.html#reqexpr) . The so called ap_expr (↗ ../expr.html) syntax of `Require expr` is a superset of the syntax of `SSLRequire`, with the following exception:
In `SSLRequire`, the comparison operators <, <=, ... are completely equivalent to the operators `lt`, `le`, ... and work in a somewhat peculiar way that first compares the length of two strings and then the lexical order. On the other hand, ap_expr (↗ ../expr.html) has two sets of comparison operators: The operators <, <=, ... do lexical string comparison, while the operators `-lt`, `-le`, ... do integer comparison. For the latter, there are also aliases without the leading dashes: `lt`, `le`, ...

This directive specifies a general access requirement which has to be fulfilled in order to allow access. It is a very powerful directive because the requirement specification is an arbitrarily complex boolean expression containing any number of access checks.

The *expression* must match the following syntax (given as a BNF grammar notation):

```
expr     ::= "true" | "false"
           | "!" expr
           | expr "&&" expr
           | expr "||" expr
           | "(" expr ")"
           | comp

comp     ::= word "==" word | word "eq" word
           | word "!=" word | word "ne" word
           | word "<"  word | word "lt" word
           | word "<=" word | word "le" word
           | word ">"  word | word "gt" word
           | word ">=" word | word "ge" word
           | word "in" "{" wordlist "}"
           | word "in" "PeerExtList(" word ")"
           | word "=~" regex
           | word "!~" regex
```

```
wordlist ::= word
           | wordlist "," word

word     ::= digit
           | cstring
           | variable
           | function

digit    ::= [0-9]+
cstring  ::= "..."
variable ::= "%{" varname "}"
function ::= funcname "(" funcargs ")"
```

For `varname` any of the variables described in Environment Variables (↗ #envvars) can be used. For `funcname` the available functions are listed in the ap_expr documentation (↗ ../expr.html#functions) .

The *expression* is parsed into an internal machine representation when the configuration is loaded, and then evaluated during request processing. In .htaccess context, the *expression* is both parsed and executed each time the .htaccess file is encountered during request processing.

Example

```
SSLRequire (    %{SSL_CIPHER} !~ m/^(EXP|NULL)-/                  \
            and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd."         \
            and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"}   \
            and %{TIME_WDAY} -ge 1 and %{TIME_WDAY} -le 5         \
            and %{TIME_HOUR} -ge 8 and %{TIME_HOUR} -le 20       ) \
           or %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+$/
```

The `PeerExtList(`*`object-ID`*`)` function expects to find zero or more instances of the X.509 certificate extension identified by the given *object ID* (OID) in the client certificate. The expression evaluates to true if the left-hand side string matches exactly against the value of an extension identified with this OID. (If multiple extensions with the same OID are present, at least one extension must match).

Example

```
SSLRequire "foobar" in PeerExtList("1.2.3.4.5.6")
```

Notes on the PeerExtList function

- The object ID can be specified either as a descriptive name recognized by the SSL library, such as `"nsComment"`, or as a numeric OID, such as `"1.2.3.4.5.6"`.
- Expressions with types known to the SSL library are rendered to a string before comparison. For an extension with a type not recognized by the SSL library, mod_ssl will parse the value if it is one of the primitive ASN.1 types UTF8String, IA5String, VisibleString, or BMPString. For an extension of one of these types, the string value will be converted to UTF-8 if necessary, then compared against the left-hand-side expression.

See also

- Environment Variables in Apache HTTP Server, for additional examples.
- Require expr
- Generic expression syntax in Apache HTTP Server

SSLRequireSSL Directive

Description:	Deny access when SSL is not used for the HTTP request
Syntax:	`SSLRequireSSL`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive forbids access unless HTTP over SSL (i.e. HTTPS) is enabled for the current connection. This is very handy inside the SSL-enabled virtual host or directories for defending against configuration errors that expose stuff that should be protected. When this directive is present all requests are denied which are not using SSL.

Example

```
SSLRequireSSL
```

SSLSessionCache Directive

Description:	Type of the global/inter-process SSL Session Cache
Syntax:	`SSLSessionCache` *`type`*
Default:	`SSLSessionCache none`
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures the storage type of the global/inter-process SSL Session Cache. This cache is an optional facility which speeds up parallel request processing. For requests to the same server process (via HTTP keep-alive), OpenSSL already caches the SSL session information locally. But because modern clients request inlined images and other data via parallel requests (usually up to four parallel requests are common) those requests are served by *different* pre-forked server processes. Here an inter-process cache helps to avoid unnecessary session handshakes.

The following five storage *type*s are currently supported:

- `none`
  This disables the global/inter-process Session Cache. This will incur a noticeable speed penalty and may cause problems if using certain browsers, particularly if client certificates are enabled. This setting is not recommended.
- `nonenotnull`
  This disables any global/inter-process Session Cache. However it does force OpenSSL to send a non-null session ID to accommodate buggy clients that require one.
- `dbm:/path/to/datafile`

This makes use of a DBM hashfile on the local disk to synchronize the local OpenSSL memory caches of the server processes. This session cache may suffer reliability issues under high load. To use this, ensure that `mod_socache_dbm` is loaded.

- `shmcb:/path/to/datafile[(size)]`
  This makes use of a high-performance cyclic buffer (approx. *size* bytes in size) inside a shared memory segment in RAM (established via `/path/to/datafile`) to synchronize the local OpenSSL memory caches of the server processes. This is the recommended session cache. To use this, ensure that `mod_socache_shmcb` is loaded.

- `dc:UNIX:/path/to/socket`
  This makes use of the distcache (↗ http://distcache.sourceforge.net/) distributed session caching libraries. The argument should specify the location of the server or proxy to be used using the distcache address syntax; for example, `UNIX:/path/to/socket` specifies a UNIX domain socket (typically a local dc_client proxy); `IP:server.example.com:9001` specifies an IP address. To use this, ensure that `mod_socache_dc` is loaded.

Examples

```
SSLSessionCache "dbm:/usr/local/apache/logs/ssl_gcache_data"
SSLSessionCache "shmcb:/usr/local/apache/logs/ssl_gcache_data(512000)"
```

The `ssl-cache` mutex is used to serialize access to the session cache to prevent corruption. This mutex can be configured using the `Mutex` directive.

SSLSessionCacheTimeout Directive

Description:	Number of seconds before an SSL session expires in the Session Cache
Syntax:	`SSLSessionCacheTimeout` *`seconds`*
Default:	`SSLSessionCacheTimeout 300`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Applies also to RFC 5077 TLS session resumption in Apache 2.4.10 and later

This directive sets the timeout in seconds for the information stored in the global/inter-process SSL Session Cache, the OpenSSL internal memory cache and for sessions resumed by TLS session resumption (RFC 5077). It can be set as low as 15 for testing, but should be set to higher values like 300 in real life.

Example

```
SSLSessionCacheTimeout 600
```

SSLSessionTicketKeyFile Directive

Description:	Persistent encryption/decryption key for TLS session tickets
Syntax:	`SSLSessionTicketKeyFile` *`file-path`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.0 and later, if using OpenSSL 0.9.8h or later

Optionally configures a secret key for encrypting and decrypting TLS session tickets, as defined in RFC 5077 (↗ http://www.ietf.org/rfc/rfc5077.txt) . Primarily suitable for clustered environments where TLS sessions information should be shared between multiple nodes. For single-instance httpd setups, it is recommended to *not* configure a ticket key file, but to rely on (random) keys generated by mod_ssl at startup, instead.

The ticket key file must contain 48 bytes of random data, preferably created from a high-entropy source. On a Unix-based system, a ticket key file can be created as follows:

```
dd if=/dev/random of=/path/to/file.tkey bs=1 count=48
```

Ticket keys should be rotated (replaced) on a frequent basis, as this is the only way to invalidate an existing session ticket - OpenSSL currently doesn't allow to specify a limit for ticket lifetimes. A new ticket key only gets used after restarting the web server. All existing session tickets become invalid after a restart.

> The ticket key file contains sensitive keying material and should be protected with file permissions similar to those used for `SSLCertificateKeyFile`.

SSLSessionTickets Directive

Description:	Enable or disable use of TLS session tickets
Syntax:	`SSLSessionTickets on|off`
Default:	`SSLSessionTickets on`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.11 and later, if using OpenSSL 0.9.8f or later.

This directive allows to enable or disable the use of TLS session tickets (RFC 5077).

> TLS session tickets are enabled by default. Using them without restarting the web server with an appropriate frequency (e.g. daily) compromises perfect forward secrecy.

SSLSRPUnknownUserSeed Directive

Description:	SRP unknown user seed
Syntax:	`SSLSRPUnknownUserSeed` *`secret-string`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.4 and later, if using OpenSSL 1.0.1 or later

This directive sets the seed used to fake SRP user parameters for unknown users, to avoid leaking whether a given user exists. Specify a secret string. If this directive is not used, then Apache will return the UNKNOWN_PSK_IDENTITY alert to clients who specify an unknown username.

Example

```
SSLSRPUnknownUserSeed "secret"
```

SSLSRPVerifierFile Directive

Description:	Path to SRP verifier file
Syntax:	`SSLSRPVerifierFile` *`file-path`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.4.4 and later, if using OpenSSL 1.0.1 or later

This directive enables TLS-SRP and sets the path to the OpenSSL SRP (Secure Remote Password) verifier file containing TLS-SRP usernames, verifiers, salts, and group parameters.

Example

```
SSLSRPVerifierFile "/path/to/file.srpv"
```

The verifier file can be created with the `openssl` command line utility:

Creating the SRP verifier file

```
openssl srp -srpvfile passwd.srpv -userinfo "some info" -add username
```

The value given with the optional `-userinfo` parameter is available in the `SSL_SRP_USERINFO` request environment variable.

SSLStaplingCache Directive

Description:	Configures the OCSP stapling cache
Syntax:	`SSLStaplingCache` *`type`*
Context:	server config
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

Configures the cache used to store OCSP responses which get included in the TLS handshake if `SSLUseStapling` is enabled. Configuration of a cache is mandatory for OCSP stapling. With the exception of `none` and `nonenotnull`, the same storage types are supported as with `SSLSessionCache`.

SSLStaplingErrorCacheTimeout Directive

Description:	Number of seconds before expiring invalid responses in the OCSP stapling cache
Syntax:	`SSLStaplingErrorCacheTimeout` *`seconds`*
Default:	`SSLStaplingErrorCacheTimeout 600`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

Sets the timeout in seconds before *invalid* responses in the OCSP stapling cache (configured through `SSLStaplingCache`) will expire. To set the cache timeout for valid responses, see `SSLStaplingStandardCacheTimeout`.

SSLStaplingFakeTryLater Directive

Description:	Synthesize "tryLater" responses for failed OCSP stapling queries
Syntax:	`SSLStaplingFakeTryLater on|off`
Default:	`SSLStaplingFakeTryLater on`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

When enabled and a query to an OCSP responder for stapling purposes fails, mod_ssl will synthesize a "tryLater" response for the client. Only effective if `SSLStaplingReturnResponderErrors` is also enabled.

SSLStaplingForceURL Directive

Description:	Override the OCSP responder URI specified in the certificate's AIA extension
Syntax:	`SSLStaplingForceURL` *`uri`*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

This directive overrides the URI of an OCSP responder as obtained from the authorityInfoAccess (AIA) extension of the certificate. One potential use is when a proxy is used for retrieving OCSP queries.

SSLStaplingResponderTimeout Directive

Description:	Timeout for OCSP stapling queries
Syntax:	`SSLStaplingResponderTimeout` *`seconds`*
Default:	`SSLStaplingResponderTimeout 10`

Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

This option sets the timeout for queries to OCSP responders when `SSLUseStapling` is enabled and mod_ssl is querying a responder for OCSP stapling purposes.

SSLStaplingResponseMaxAge Directive

Description:	Maximum allowable age for OCSP stapling responses
Syntax:	`SSLStaplingResponseMaxAge` *seconds*
Default:	`SSLStaplingResponseMaxAge -1`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

This option sets the maximum allowable age ("freshness") when considering OCSP responses for stapling purposes, i.e. when `SSLUseStapling` is turned on. The default value (`-1`) does not enforce a maximum age, which means that OCSP responses are considered valid as long as their `nextUpdate` field is in the future.

SSLStaplingResponseTimeSkew Directive

Description:	Maximum allowable time skew for OCSP stapling response validation
Syntax:	`SSLStaplingResponseTimeSkew` *seconds*
Default:	`SSLStaplingResponseTimeSkew 300`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

This option sets the maximum allowable time skew when mod_ssl checks the `thisUpdate` and `nextUpdate` fields of OCSP responses which get included in the TLS handshake (OCSP stapling). Only applicable if `SSLUseStapling` is turned on.

SSLStaplingReturnResponderErrors Directive

Description:	Pass stapling related OCSP errors on to client
Syntax:	`SSLStaplingReturnResponderErrors on|off`
Default:	`SSLStaplingReturnResponderErrors on`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

When enabled, mod_ssl will pass responses from unsuccessful stapling related OCSP queries (such as responses with an overall status other than "successful", responses with a certificate status other than "good", expired responses etc.) on to the client. If set to `off`, only responses indicating a certificate status of "good" will be included in the TLS handshake.

SSLStaplingStandardCacheTimeout Directive

Description:	Number of seconds before expiring responses in the OCSP stapling cache
Syntax:	`SSLStaplingStandardCacheTimeout` *seconds*
Default:	`SSLStaplingStandardCacheTimeout 3600`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

Sets the timeout in seconds before responses in the OCSP stapling cache (configured through `SSLStaplingCache`) will expire. This directive applies to *valid* responses, while `SSLStaplingErrorCacheTimeout` is used for controlling the timeout for invalid/unavailable responses.

SSLStrictSNIVHostCheck Directive

Description:	Whether to allow non-SNI clients to access a name-based virtual host.
Syntax:	`SSLStrictSNIVHostCheck on|off`
Default:	`SSLStrictSNIVHostCheck off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in Apache 2.2.12 and later

This directive sets whether a non-SNI client is allowed to access a name-based virtual host. If set to `on` in the default name-based virtual host, clients that are SNI unaware will not be allowed to access *any* virtual host, belonging to this particular IP / port combination. If set to `on` in any other virtual host, SNI unaware clients are not allowed to access this particular virtual host.

This option is only available if httpd was compiled against an SNI capable version of OpenSSL.

Example

```
SSLStrictSNIVHostCheck on
```

SSLUserName Directive

Description:	Variable name to determine user name
Syntax:	`SSLUserName` *`varname`*
Context:	server config, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets the "user" field in the Apache request object. This is used by lower modules to identify the user with a character string. In particular, this may cause the environment variable `REMOTE_USER` to be set. The *varname* can be any of the SSL environment variables (↗ #envvars) .

Note that this directive has no effect if the `FakeBasicAuth` option is used (see SSLOptions (↗ #ssloptions)).

Example

```
SSLUserName SSL_CLIENT_S_DN_CN
```

SSLUseStapling Directive

Description:	Enable stapling of OCSP responses in the TLS handshake
Syntax:	`SSLUseStapling on|off`
Default:	`SSLUseStapling off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available if using OpenSSL 0.9.8h or later

This option enables OCSP stapling, as defined by the "Certificate Status Request" TLS extension specified in RFC 6066. If enabled (and requested by the client), mod_ssl will include an OCSP response for its own certificate in the TLS handshake. Configuring an `SSLStaplingCache` is a prerequisite for enabling OCSP stapling.

OCSP stapling relieves the client of querying the OCSP responder on its own, but it should be noted that with the RFC 6066 specification, the server's `CertificateStatus` reply may only include an OCSP response for a single cert. For server certificates with intermediate CA certificates in their chain (the typical case nowadays), stapling in its current implementation therefore only partially achieves the stated goal of "saving roundtrips and resources" - see also RFC 6961 (↗ http://www.ietf.org/rfc/rfc6961.txt) (TLS Multiple Certificate Status Extension).

When OCSP stapling is enabled, the `ssl-stapling` mutex is used to control access to the OCSP stapling cache in order to prevent corruption, and the `sss-stapling-refresh` mutex is used to control refreshes of OCSP responses. These mutexes can be configured using the `Mutex` directive.

SSLVerifyClient Directive

Description:	Type of Client Certificate verification
Syntax:	`SSLVerifyClient` *`level`*
Default:	`SSLVerifyClient none`
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets the Certificate verification level for the Client Authentication. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured client verification level after the HTTP request was read but before the HTTP response is sent.

The following levels are available for *level*:

- **none**: no client Certificate is required at all
- **optional**: the client *may* present a valid Certificate
- **require**: the client *has to* present a valid Certificate
- **optional_no_ca**: the client may present a valid Certificate
  but it need not to be (successfully) verifiable. This option cannot be relied upon for client authentication.

Example

```
SSLVerifyClient require
```

SSLVerifyDepth Directive

Description:	Maximum depth of CA Certificates in Client Certificate verification
Syntax:	`SSLVerifyDepth` *`number`*
Default:	`SSLVerifyDepth 1`
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets how deeply mod_ssl should verify before deciding that the clients don't have a valid certificate. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured client verification depth after the HTTP request was read but before the HTTP response is sent.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the client certificate. A depth of 0 means that self-signed client certificates are accepted only, the default depth of 1 means the client certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under `SSLCACertificatePath`), etc.

Example

```
SSLVerifyDepth 10
```

Comments

Notice:
This is not a Q&A section. Comments placed here should be pointed towards suggestions on improving the documentation or server, and may be removed by our moderators if they are either implemented or considered invalid/off-topic. Questions on how to manage the Apache HTTP Server should be directed at either our IRC channel, #httpd, on Libera.chat, or sent to our mailing lists.