NAME

Net::SSLeay – Perl extension for using OpenSSL

SYNOPSIS

```
  use Net::SSLeay qw(get_https post_https sslcat make_headers make_form);

  ($page) = get_https('www.bacus.pt', 443, '/');                 # Case 1

  ($page, $response, %reply_headers)
         = get_https('www.bacus.pt', 443, '/',                   # Case 2
                make_headers(User-Agent => 'Cryptozilla/5.0b1',
                             Referer    => 'https://www.bacus.pt'
                ));

  ($page, $result, %headers) =                                   # Case 2b
         = get_https('www.bacus.pt', 443, '/protected.html',
              make_headers(Authorization =>
                           'Basic ' . MIME::Base64::encode("$user:$pass",''))
              );

  ($page, $response, %reply_headers)
         = post_https('www.bacus.pt', 443, '/foo.cgi', '',       # Case 3
                make_form(OK   => '1',
                          name => 'Sampo'
                ));

  $reply = sslcat($host, $port, $request);                       # Case 4

  ($reply, $err, $server_cert) = sslcat($host, $port, $request); # Case 5

  $Net::SSLeay::trace = 2;  # 0=no debugging, 1=ciphers, 2=trace, 3=dump data

  Net::SSLeay::initialize(); # Initialize ssl library once
```

DESCRIPTION

Net::SSLeay module contains perl bindings to openssl (<http://www.openssl.org>) library.

COMPATIBILITY NOTE: Net::SSLeay cannot be built with pre–0.9.3 openssl. It is strongly recommended to use at least 0.9.7 (as older versions are not tested during development). Some low level API functions may be available with certain openssl versions.

It is compatible with OpenSSL 1.0 and 1.1. Some functions are not available under OpenSSL 1.1.

Net::SSLeay module basically comprise of:

- High level functions for accessing web servers (by using HTTP/HTTPS)
- Low level API (mostly mapped 1:1 to openssl's C functions)
- Convenience functions (related to low level API but with more perl friendly interface)

There is also a related module called Net::SSLeay::Handle included in this distribution that you might want to use instead. It has its own pod documentation.

High level functions for accessing web servers

This module offers some high level convenience functions for accessing web pages on SSL servers (for symmetry, the same API is offered for accessing http servers, too), an `sslcat()` function for writing your own clients, and finally access to the SSL api of the SSLeay/OpenSSL package so you can write servers or clients for more complicated applications.

For high level functions it is most convenient to import them into your main namespace as indicated in the synopsis.

Basic set of functions

- get_https
- post_https
- put_https
- head_https
- do_https
- sslcat
- https_cat
- make_form
- make_headers

Case 1 (in SYNOPSIS) demonstrates the typical invocation of **get_https()** to fetch an HTML page from secure server. The first argument provides the hostname or IP in dotted decimal notation of the remote server to contact. The second argument is the TCP port at the remote end (your own port is picked arbitrarily from high numbered ports as usual for TCP). The third argument is the URL of the page without the host name part. If in doubt consult the HTTP specifications at <http://www.w3c.org>.

Case 2 (in SYNOPSIS) demonstrates full fledged use of `get_https()`. As can be seen, `get_https()` parses the response and response headers and returns them as a list, which can be captured in a hash for later reference. Also a fourth argument to `get_https()` is used to insert some additional headers in the request. `make_headers()` is a function that will convert a list or hash to such headers. By default `get_https()` supplies `Host` (to make virtual hosting easy) and `Accept` (reportedly needed by IIS) headers.

Case 2b (in SYNOPSIS) demonstrates how to get a password protected page. Refer to the HTTP protocol specifications for further details (e.g. RFC−2617).

Case 3 (in SYNOPSIS) invokes `post_https()` to submit a HTML/CGI form to a secure server. The first four arguments are equal to `get_https()` (note that the empty string (`''`) is passed as header argument). The fifth argument is the contents of the form formatted according to CGI specification. Do not post UTF−8 data as content: use utf8::downgrade first. In this case the helper function `make_https()` is used to do the formatting, but you could pass any string. `post_https()` automatically adds `Content-Type` and `Content-Length` headers to the request.

Case 4 (in SYNOPSIS) shows the fundamental `sslcat()` function (inspired in spirit by the `netcat` utility :−). It's your swiss army knife that allows you to easily contact servers, send some data, and then get the response. You are responsible for formatting the data and parsing the response − `sslcat()` is just a transport.

Case 5 (in SYNOPSIS) is a full invocation of `sslcat()` which allows the return of errors as well as the server (peer) certificate.

The `$trace` global variable can be used to control the verbosity of the high level functions. Level 0 guarantees silence, level 1 (the default) only emits error messages.

Alternate versions of high-level API

- get_https3
- post_https3
- put_https3
- get_https4
- post_https4

- put_https4

The above mentioned functions actually return the response headers as a list, which only gets converted to hash upon assignment (this assignment looses information if the same header occurs twice, as may be the case with cookies). There are also other variants of the functions that return unprocessed headers and that return a reference to a hash.

```
($page, $response, @headers) = get_https('www.bacus.pt', 443, '/');
for ($i = 0; $i < $#headers; $i+=2) {
    print "$headers[$i] = " . $headers[$i+1] . "\n";
}

($page, $response, $headers, $server_cert)
  = get_https3('www.bacus.pt', 443, '/');
print "$headers\n";

($page, $response, $headers_ref)
  = get_https4('www.bacus.pt', 443, '/');
for $k (sort keys %{$headers_ref}) {
    for $v (@{$$headers_ref{$k}}) {
        print "$k = $v\n";
    }
}
```

All of the above code fragments accomplish the same thing: display all values of all headers. The API functions ending in “3” return the headers simply as a scalar string and it is up to the application to split them up. The functions ending in “4” return a reference to a hash of arrays (see perlref and perllol if you are not familiar with complex perl data structures). To access a single value of such a header hash you would do something like

```
print $$headers_ref{COOKIE}[0];
```

Variants 3 and 4 also allow you to discover the server certificate in case you would like to store or display it, e.g.

```
($p, $resp, $hdrs, $server_cert) = get_https3('www.bacus.pt', 443, '/');
if (!defined($server_cert) || ($server_cert == 0)) {
    warn "Subject Name: undefined, Issuer  Name: undefined";
} else {
    warn 'Subject Name: '
        . Net::SSLeay::X509_NAME_oneline(
               Net::SSLeay::X509_get_subject_name($server_cert))
            . 'Issuer  Name: '
                . Net::SSLeay::X509_NAME_oneline(
                      Net::SSLeay::X509_get_issuer_name($server_cert));
}
```

Beware that this method only allows after the fact verification of the certificate: by the time `get_https3()` has returned the https request has already been sent to the server, whether you decide to trust it or not. To do the verification correctly you must either employ the OpenSSL certificate verification framework or use the lower level API to first connect and verify the certificate and only then send the http data. See the implementation of `ds_https3()` for guidance on how to do this.

Using client certificates

Secure web communications are encrypted using symmetric crypto keys exchanged using encryption based on the certificate of the server. Therefore in all SSL connections the server must have a certificate. This serves both to authenticate the server to the clients and to perform the key exchange.

Sometimes it is necessary to authenticate the client as well. Two options are available: HTTP basic authentication and a client side certificate. The basic authentication over HTTPS is actually quite safe because HTTPS guarantees that the password will not travel in the clear. Never-the-less, problems like easily guessable passwords remain. The client certificate method involves authentication of the client at the SSL level using a certificate. For this to work, both the client and the server have certificates (which typically are different) and private keys.

The API functions outlined above accept additional arguments that allow one to supply the client side certificate and key files. The format of these files is the same as used for server certificates and the caveat about encrypting private keys applies.

```
 ($page, $result, %headers) =                                    # 2c
         = get_https('www.bacus.pt', 443, '/protected.html',
               make_headers(Authorization =>
                              'Basic ' . MIME::Base64::encode("$user:$pass",'')),
               '', $mime_type6, $path_to_crt7, $path_to_key8);

 ($page, $response, %reply_headers)
         = post_https('www.bacus.pt', 443, '/foo.cgi',           # 3b
               make_headers('Authorization' =>
                              'Basic ' . MIME::Base64::encode("$user:$pass",'')),
               make_form(OK   => '1', name => 'Sampo'),
               $mime_type6, $path_to_crt7, $path_to_key8);
```

Case 2c (in SYNOPSIS) demonstrates getting a password protected page that also requires a client certificate, i.e. it is possible to use both authentication methods simultaneously.

Case 3b (in SYNOPSIS) is a full blown POST to a secure server that requires both password authentication and a client certificate, just like in case 2c.

Note: The client will not send a certificate unless the server requests one. This is typically achieved by setting the verify mode to `VERIFY_PEER` on the server:

```
 Net::SSLeay::set_verify(ssl, Net::SSLeay::VERIFY_PEER, 0);
```

See `perldoc ~openssl/doc/ssl/SSL_CTX_set_verify.pod` for a full description.

Working through a web proxy

- set_proxy

`Net::SSLeay` can use a web proxy to make its connections. You need to first set the proxy host and port using `set_proxy()` and then just use the normal API functions, e.g:

```
 Net::SSLeay::set_proxy('gateway.myorg.com', 8080);
 ($page) = get_https('www.bacus.pt', 443, '/');
```

If your proxy requires authentication, you can supply a username and password as well

```
 Net::SSLeay::set_proxy('gateway.myorg.com', 8080, 'joe', 'salainen');
 ($page, $result, %headers) =
         = get_https('www.bacus.pt', 443, '/protected.html',
               make_headers(Authorization =>
                              'Basic ' . MIME::Base64::encode("susie:pass",''))
               );
```

This example demonstrates the case where we authenticate to the proxy as `"joe"` and to the final web server as `"susie"`. Proxy authentication requires the `MIME::Base64` module to work.

HTTP (without S) API

- get_http
- post_http

- tcpcat
- get_httpx
- post_httpx
- tcpxcat

Over the years it has become clear that it would be convenient to use the light-weight flavour API of `Net::SSLeay` for normal HTTP as well (see `LWP` for the heavy-weight object-oriented approach). In fact it would be nice to be able to flip https on and off on the fly. Thus regular HTTP support was evolved.

```
  use Net::SSLeay qw(get_http post_http tcpcat
                     get_httpx post_httpx tcpxcat
                     make_headers make_form);

  ($page, $result, %headers)
         = get_http('www.bacus.pt', 443, '/protected.html',
              make_headers(Authorization =>
                           'Basic ' . MIME::Base64::encode("$user:$pass",''))
              );

  ($page, $response, %reply_headers)
         = post_http('www.bacus.pt', 443, '/foo.cgi', '',
                make_form(OK   => '1',
                          name => 'Sampo'
                ));

  ($reply, $err) = tcpcat($host, $port, $request);

  ($page, $result, %headers)
         = get_httpx($usessl, 'www.bacus.pt', 443, '/protected.html',
              make_headers(Authorization =>
                           'Basic ' . MIME::Base64::encode("$user:$pass",''))
              );

  ($page, $response, %reply_headers)
         = post_httpx($usessl, 'www.bacus.pt', 443, '/foo.cgi', '',
                make_form(OK   => '1',  name => 'Sampo' ));

  ($reply, $err, $server_cert) = tcpxcat($usessl, $host, $port, $request);
```

As can be seen, the `"x"` family of APIs takes as the first argument a flag which indicates whether SSL is used or not.

Certificate verification and Certificate Revocation Lists (CRLs)

OpenSSL supports the ability to verify peer certificates. It can also optionally check the peer certificate against a Certificate Revocation List (CRL) from the certificates issuer. A CRL is a file, created by the certificate issuer that lists all the certificates that it previously signed, but which it now revokes. CRLs are in PEM format.

You can enable `Net::SSLeay CRL` checking like this:

```
            &Net::SSLeay::X509_STORE_set_flags
                (&Net::SSLeay::CTX_get_cert_store($ssl),
                 &Net::SSLeay::X509_V_FLAG_CRL_CHECK);
```

After setting this flag, if OpenSSL checks a peer's certificate, then it will attempt to find a CRL for the issuer. It does this by looking for a specially named file in the search directory specified by CTX_load_verify_locations. CRL files are named with the hash of the issuer's subject name, followed by

`.r0`, `.r1` etc. For example `ab1331b2.r0`, `ab1331b2.r1`. It will read all the .r files for the issuer, and then check for a revocation of the peer certificate in all of them. (You can also force it to look in a specific named CRL file., see below). You can find out the hash of the issuer subject name in a CRL with

```
        openssl crl -in crl.pem -hash -noout
```

If the peer certificate does not pass the revocation list, or if no CRL is found, then the handshaking fails with an error.

You can also force OpenSSL to look for CRLs in one or more arbitrarily named files.

```
    my $bio = Net::SSLeay::BIO_new_file($crlfilename, 'r');
    my $crl = Net::SSLeay::PEM_read_bio_X509_CRL($bio);
    if ($crl) {
        Net::SSLeay::X509_STORE_add_crl(
             Net::SSLeay::CTX_get_cert_store($ssl, $crl)
        );
    } else {
        error reading CRL....
    }
```

Usually the URLs where you can download the CRLs is contained in the certificate itself and you can extract them with

```
    my @url = Net::SSLeay::P_X509_get_crl_distribution_points($cert)
```

But there is no automatic downloading of the CRLs and often these CRLs are too huge to just download them to verify a single certificate. Also, these CRLs are often in DER format which you need to convert to PEM before you can use it:

```
    openssl crl -in crl.der -inform der -out crl.pem
```

So as an alternative for faster and timely revocation checks you better use the Online Status Revocation Protocol (OCSP).

Certificate verification and Online Status Revocation Protocol (OCSP)

While checking for revoked certificates is possible and fast with Certificate Revocation Lists, you need to download the complete and often huge list before you can verify a single certificate.

A faster way is to ask the CA to check the revocation of just a single or a few certificates using OCSP. Basically you generate for each certificate an OCSP_CERTID based on the certificate itself and its issuer, put the ids togetether into an OCSP_REQUEST and send the request to the URL given in the certificate.

As a result you get back an OCSP_RESPONSE and need to check the status of the response, check that it is valid (e.g. signed by the CA) and finally extract the information about each OCSP_CERTID to find out if the certificate is still valid or got revoked.

With Net::SSLeay this can be done like this:

```
    # get id(s) for given certs, like from get_peer_certificate
    # or get_peer_cert_chain. This will croak if
    # - one tries to make an OCSP_CERTID for a self-signed certificate
    # - the issuer of the certificate cannot be found in the SSL objects
    #   store, nor in the current certificate chain
    my $cert = Net::SSLeay::get_peer_certificate($ssl);
    my $id = eval { Net::SSLeay::OCSP_cert2ids($ssl,$cert) };
    die "failed to make OCSP_CERTID: $@" if $@;

    # create OCSP_REQUEST from id(s)
    # Multiple can be put into the same request, if the same OCSP responder
    # is responsible for them.
    my $req = Net::SSLeay::OCSP_ids2req($id);
```

```
# determine URI of OCSP responder
my $uri = Net::SSLeay::P_X509_get_ocsp_uri($cert);

# Send stringified OCSP_REQUEST with POST to $uri.
# We can ignore certificate verification for https, because the OCSP
# response itself is signed.
my $ua = HTTP::Tiny->new(verify_SSL => 0);
my $res = $ua->request( 'POST',$uri, {
    headers => { 'Content-type' => 'application/ocsp-request' },
    content => Net::SSLeay::i2d_OCSP_REQUEST($req)
});
my $content = $res && $res->{success} && $res->{content}
    or die "query failed";

# Extract OCSP_RESPONSE.
# this will croak if the string is not an OCSP_RESPONSE
my $resp = eval { Net::SSLeay::d2i_OCSP_RESPONSE($content) };

# Check status of response.
my $status = Net::SSLeay::OCSP_response_status($resp);
if ($status != Net::SSLeay::OCSP_RESPONSE_STATUS_SUCCESSFUL())
    die "OCSP response failed: ".
        Net::SSLeay::OCSP_response_status_str($status);
}

# Verify signature of response and if nonce matches request.
# This will croak if there is a nonce in the response, but it does not match
# the request. It will return false if the signature could not be verified,
# in which case details can be retrieved with Net::SSLeay::ERR_get_error.
# It will not complain if the response does not contain a nonce, which is
# usually the case with pre-signed responses.
if ( ! eval { Net::SSLeay::OCSP_response_verify($ssl,$resp,$req) }) {
    die "OCSP response verification failed";
}

# Extract information from OCSP_RESPONSE for each of the ids.

# If called in scalar context it will return the time (as time_t), when the
# next update is due (minimum of all successful responses inside $resp). It
# will croak on the following problems:
# - response is expired or not yet valid
# - no response for given OCSP_CERTID
# - certificate status is not good (e.g. revoked or unknown)
if ( my $nextupd = eval { Net::SSLeay::OCSP_response_results($resp,$id) }) {
    warn "certificate is valid, next update in ".
        ($nextupd-time())." seconds\n";
} else {
    die "certificate is not valid: $@";
}

# But in array context it will return detailed information about each given
# OCSP_CERTID instead croaking on errors:
# if no @ids are given it will return information about all single responses
# in the OCSP_RESPONSE
```

```
my @results = Net::SSLeay::OCSP_response_results($resp,@ids);
for my $r (@results) {
    print Dumper($r);
    # @results are in the same order as the @ids and contain:
    # $r->[0] - OCSP_CERTID
    # $r->[1] - undef if no error (certificate good) OR error message as string
    # $r->[2] - hash with details:
    #   thisUpdate - time_t of this single response
    #   nextUpdate - time_t when update is expected
    #   statusType - integer:
    #      V_OCSP_CERTSTATUS_GOOD(0)
    #      V_OCSP_CERTSTATUS_REVOKED(1)
    #      V_OCSP_CERTSTATUS_UNKNOWN(2)
    #   revocationTime - time_t (only if revoked)
    #   revocationReason - integer (only if revoked)
    #   revocationReason_str - reason as string (only if revoked)
}
```

To further speed up certificate revocation checking one can use a TLS extension to instruct the server to staple the OCSP response:

```
# set TLS extension before doing SSL_connect
Net::SSLeay::set_tlsext_status_type($ssl,
    Net::SSLeay::TLSEXT_STATUSTYPE_ocsp());

# setup callback to verify OCSP response
my $cert_valid = undef;
Net::SSLeay::CTX_set_tlsext_status_cb($context,sub {
    my ($ssl,$resp) = @_;
    if (!$resp) {
        # Lots of servers don't return an OCSP response.
        # In this case we must check the OCSP status outside the SSL
        # handshake.
        warn "server did not return stapled OCSP response\n";
        return 1;
    }
    # verify status
    my $status = Net::SSLeay::OCSP_response_status($resp);
    if ($status != Net::SSLeay::OCSP_RESPONSE_STATUS_SUCCESSFUL()) {
        warn "OCSP response failure: $status\n";
        return 1;
    }
    # verify signature - we have no OCSP_REQUEST here to check nonce
    if (!eval { Net::SSLeay::OCSP_response_verify($ssl,$resp) }) {
        warn "OCSP response verify failed\n";
        return 1;
    }
    # check if the certificate is valid
    # we should check here against the peer_certificate
    my $cert = Net::SSLeay::get_peer_certificate();
    my $certid = eval { Net::SSLeay::OCSP_cert2ids($ssl,$cert) } or do {
        warn "cannot get certid from cert: $@";
        $cert_valid = -1;
        return 1;
    };
```

```
            if ( $nextupd = eval {
                Net::SSLeay::OCSP_response_results($resp,$certid) }) {
                warn "certificate not revoked\n";
                $cert_valid = 1;
            } else {
                warn "certificate not valid: $@";
                $cert_valid = 0;
            }
        });

        # do SSL handshake here
        ....
        # check if certificate revocation was checked already
        if ( ! defined $cert_valid) {
            # check revocation outside of SSL handshake by asking OCSP responder
            ...
        } elsif ( ! $cert_valid ) {
            die "certificate not valid - closing SSL connection";
        } elsif ( $cert_valid<0 ) {
            die "cannot verify certificate revocation - self-signed ?";
        } else {
            # everything fine
            ...
        }
```

Using Net::SSLeay in multi-threaded applications

IMPORTANT: versions 1.42 or earlier are not thread-safe!

Net::SSLeay module implements all necessary stuff to be ready for multi-threaded environment – it requires openssl–0.9.7 or newer. The implementation fully follows thread safety related requirements of openssl library(see <http://www.openssl.org/docs/crypto/threads.html>).

If you are about to use Net::SSLeay (or any other module based on Net::SSLeay) in multi-threaded perl application it is recommended to follow this best-practice:

Initialization

Load and initialize Net::SSLeay module in the main thread:

```
    use threads;
    use Net::SSLeay;

    Net::SSLeay::load_error_strings();
    Net::SSLeay::SSLeay_add_ssl_algorithms();
    Net::SSLeay::randomize();

    sub do_master_job {
      #... call whatever from Net::SSLeay
    }

    sub do_worker_job {
      #... call whatever from Net::SSLeay
    }

    #start threads
    my $master  = threads->new(\&do_master_job, 'param1', 'param2');
    my @workers = threads->new(\&do_worker_job, 'arg1', 'arg2') for (1..10);
    #waiting for all threads to finish
    $_->join() for (threads->list);
```

NOTE: Openssl's `int SSL_library_init(void)` function (which is also aliased as `SSLeay_add_ssl_algorithms`, `OpenSSL_add_ssl_algorithms` and `add_ssl_algorithms`) is not re-entrant and multiple calls can cause a crash in threaded application. Net::SSLeay implements flags preventing repeated calls to this function, therefore even multiple initialization via **Net::SSLeay::SSLeay_add_ssl_algorithms()** should work without trouble.

Using callbacks

Do not use callbacks across threads (the module blocks cross-thread callback operations and throws a warning). Always do the callback setup, callback use and callback destruction within the same thread.

Using openssl elements

All openssl elements (X509, SSL_CTX, ...) can be directly passed between threads.

```
    use threads;
    use Net::SSLeay;

    Net::SSLeay::load_error_strings();
    Net::SSLeay::SSLeay_add_ssl_algorithms();
    Net::SSLeay::randomize();

    sub do_job {
      my $context = shift;
      Net::SSLeay::CTX_set_default_passwd_cb($context, sub { "secret" });
      #...
    }

    my $c = Net::SSLeay::CTX_new();
    threads->create(\&do_job, $c);
```

Or:

```
    use threads;
    use Net::SSLeay;

    my $context; #does not need to be 'shared'

    Net::SSLeay::load_error_strings();
    Net::SSLeay::SSLeay_add_ssl_algorithms();
    Net::SSLeay::randomize();

    sub do_job {
      Net::SSLeay::CTX_set_default_passwd_cb($context, sub { "secret" });
      #...
    }

    $context = Net::SSLeay::CTX_new();
    threads->create(\&do_job);
```

Using other perl modules based on Net::SSLeay

It should be fine to use any other module based on Net::SSLeay (like IO::Socket::SSL) in multi-threaded applications. It is generally recommended to do any global initialization of such a module in the main thread before calling `threads->new(..)` or `threads->create(..)` but it might differ module by module.

To be safe you can load and init Net::SSLeay explicitly in the main thread:

```
use Net::SSLeay;
use Other::SSLeay::Based::Module;

Net::SSLeay::load_error_strings();
Net::SSLeay::SSLeay_add_ssl_algorithms();
Net::SSLeay::randomize();
```

Or even safer:

```
use Net::SSLeay;
use Other::SSLeay::Based::Module;

BEGIN {
  Net::SSLeay::load_error_strings();
  Net::SSLeay::SSLeay_add_ssl_algorithms();
  Net::SSLeay::randomize();
}
```

Combining Net::SSLeay with other modules linked with openssl

BEWARE: This might be a big trouble! This is not guaranteed be thread-safe!

There are many other (XS) modules linked directly to openssl library (like Crypt::SSLeay).

As it is expected that also "another" module will call `SSLeay_add_ssl_algorithms` at some point we have again a trouble with multiple openssl initialization by Net::SSLeay and "another" module.

As you can expect Net::SSLeay is not able to avoid multiple initialization of openssl library called by "another" module, thus you have to handle this on your own (in some cases it might not be possible at all to avoid this).

Threading with get_https and friends

The convenience functions get_https, post_https etc all initialize the SSL library by calling Net::SSLeay::initialize which does the conventional library initialization:

```
Net::SSLeay::load_error_strings();
Net::SSLeay::SSLeay_add_ssl_algorithms();
Net::SSLeay::randomize();
```

Net::SSLeay::initialize initializes the SSL library at most once. You can override the Net::SSLeay::initialize function if you desire some other type of initialization behaviour by get_https and friends. You can call Net::SSLeay::initialize from your own code if you desire this conventional library initialization.

Convenience routines

To be used with Low level API

```
Net::SSLeay::randomize($rn_seed_file,$additional_seed);
Net::SSLeay::set_cert_and_key($ctx, $cert_path, $key_path);
$cert = Net::SSLeay::dump_peer_certificate($ssl);
Net::SSLeay::ssl_write_all($ssl, $message) or die "ssl write failure";
$got = Net::SSLeay::ssl_read_all($ssl) or die "ssl read failure";

$got = Net::SSLeay::ssl_read_CRLF($ssl [, $max_length]);
$got = Net::SSLeay::ssl_read_until($ssl [, $delimit [, $max_length]]);
Net::SSLeay::ssl_write_CRLF($ssl, $message);
```

- randomize

  seeds the openssl PRNG with `/dev/urandom` (see the top of `SSLeay.pm` for how to change or configure this) and optionally with user provided data. It is very important to properly seed your random numbers, so do not forget to call this. The high level API functions automatically call `randomize()` so it is not needed with them. See also caveats.

- set_cert_and_key

  takes two file names as arguments and sets the certificate and private key to those. This can be used to set either server certificates or client certificates.

- dump_peer_certificate

  allows you to get a plaintext description of the certificate the peer (usually the server) presented to us.

- ssl_read_all

  see ssl_write_all (below)

- ssl_write_all

  `ssl_read_all()` and `ssl_write_all()` provide true blocking semantics for these operations (see limitation, below, for explanation). These are much preferred to the low level API equivalents (which implement BSD blocking semantics). The message argument to `ssl_write_all()` can be a reference. This is helpful to avoid unnecessary copying when writing something big, e.g:

  ```
  $data = 'A' x 1000000000;
  Net::SSLeay::ssl_write_all($ssl, \$data) or die "ssl write failed";
  ```

- ssl_read_CRLF

  uses `ssl_read_all()` to read in a line terminated with a carriage return followed by a linefeed (CRLF). The CRLF is included in the returned scalar.

- ssl_read_until

  uses `ssl_read_all()` to read from the SSL input stream until it encounters a programmer specified delimiter. If the delimiter is undefined, `$/` is used. If `$/` is undefined, `\n` is used. One can optionally set a maximum length of bytes to read from the SSL input stream.

- ssl_write_CRLF

  writes `$message` and appends CRLF to the SSL output stream.

Initialization

In order to use the low level API you should start your programs with the following incantation:

```
use Net::SSLeay qw(die_now die_if_ssl_error);
Net::SSLeay::load_error_strings();
Net::SSLeay::SSLeay_add_ssl_algorithms();    # Important!
Net::SSLeay::ENGINE_load_builtin_engines();  # If you want built-in engines
Net::SSLeay::ENGINE_register_all_complete(); # If you want built-in engines
Net::SSLeay::randomize();
```

Error handling functions

I can not emphasize the need to check for error enough. Use these functions even in the most simple programs, they will reduce debugging time greatly. Do not ask questions on the mailing list without having first sprinkled these in your code.

- die_now
- die_if_ssl_error

  `die_now()` and `die_if_ssl_error()` are used to conveniently print the SSLeay error stack when something goes wrong:

  ```
  Net::SSLeay::connect($ssl) or die_now("Failed SSL connect ($!)");


  Net::SSLeay::write($ssl, "foo") or die_if_ssl_error("SSL write ($!)");
  ```

- print_errs

  You can also use `Net::SSLeay::print_errs()` to dump the error stack without exiting the program. As can be seen, your code becomes much more readable if you import the error reporting functions into your main name space.

Sockets

Perl uses file handles for all I/O. While SSLeay has a quite flexible BIO mechanism and perl has an evolved PerlIO mechanism, this module still sticks to using file descriptors. Thus to attach SSLeay to a socket you should use `fileno()` to extract the underlying file descriptor:

```
Net::SSLeay::set_fd($ssl, fileno(S));   # Must use fileno
```

You should also set `$|` to 1 to eliminate STDIO buffering so you do not get confused if you use perl I/O functions to manipulate your socket handle.

If you need to `select(2)` on the socket, go right ahead, but be warned that OpenSSL does some internal buffering so SSL_read does not always return data even if the socket selected for reading (just keep on selecting and trying to read). `Net::SSLeay` is no different from the C language OpenSSL in this respect.

Callbacks

You can establish a per-context verify callback function something like this:

```
        sub verify {
            my ($ok, $x509_store_ctx) = @_;
            print "Verifying certificate...\n";
                ...
            return $ok;
        }
```

It is used like this:

```
        Net::SSLeay::set_verify ($ssl, Net::SSLeay::VERIFY_PEER, \&verify);
```

Per-context callbacks for decrypting private keys are implemented.

```
        Net::SSLeay::CTX_set_default_passwd_cb($ctx, sub { "top-secret" });
        Net::SSLeay::CTX_use_PrivateKey_file($ctx, "key.pem",
                                             Net::SSLeay::FILETYPE_PEM)
            or die "Error reading private key";
        Net::SSLeay::CTX_set_default_passwd_cb($ctx, undef);
```

If Hello Extensions are supported by your OpenSSL, a session secret callback can be set up to be called when a session secret is set by openssl.

Establish it like this:
  Net::SSLeay::set_session_secret_cb($ssl, \&session_secret_cb, `$somedata`);

It will be called like this:

```
    sub session_secret_cb
    {
        my ($secret, \@cipherlist, \$preferredcipher, $somedata) = @_;
    }
```

No other callbacks are implemented. You do not need to use any callback for simple (i.e. normal) cases where the SSLeay built-in verify mechanism satisfies your needs.

It is required to reset these callbacks to undef immediately after use to prevent memory leaks, thread safety problems and crashes on exit that can occur if different threads set different callbacks.

If you want to use callback stuff, see examples/callback.pl! It's the only one I am able to make work reliably.

Low level API

In addition to the high level functions outlined above, this module contains straight-forward access to CRYPTO and SSL parts of OpenSSL C API.

See the `*.h` headers from OpenSSL C distribution for a list of low level SSLeay functions to call (check SSLeay.xs to see if some function has been implemented). The module strips the initial `"SSL_"` off of the SSLeay names. Generally you should use `Net::SSLeay::` in its place.

Note that some functions are prefixed with `"P_"` – these are very close to the original API however contain some kind of a wrapper making its interface more perl friendly.

For example:

In C:

```
        #include <ssl.h>

        err = SSL_set_verify (ssl, SSL_VERIFY_CLIENT_ONCE,
                                   &your_call_back_here);
```

In Perl:

```
        use Net::SSLeay;

        $err = Net::SSLeay::set_verify ($ssl,
                                        Net::SSLeay::VERIFY_CLIENT_ONCE,
                                        \&your_call_back_here);
```

If the function does not start with `SSL_` you should use the full function name, e.g.:

```
        $err = Net::SSLeay::ERR_get_error;
```

The following new functions behave in perlish way:

```
        $got = Net::SSLeay::read($ssl);
                                    # Performs SSL_read, but returns $got
                                    # resized according to data received.
                                    # Returns undef on failure.

        Net::SSLeay::write($ssl, $foo) || die;
                                    # Performs SSL_write, but automatically
                                    # figures out the size of $foo
```

Low level API: Version related functions

- SSLeay

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  Gives version number (numeric) of underlaying openssl library.

```
 my $ver_number = Net::SSLeay::SSLeay();
 # returns: the number identifying the openssl release
 #
 # 0x00903100 => openssl-0.9.3
 # 0x00904100 => openssl-0.9.4
 # 0x00905100 => openssl-0.9.5
 # 0x0090600f => openssl-0.9.6
 # 0x0090601f => openssl-0.9.6a
 # 0x0090602f => openssl-0.9.6b
 # ...
 # 0x009060df => openssl-0.9.6m
 # 0x0090700f => openssl-0.9.7
 # 0x0090701f => openssl-0.9.7a
```

```
# 0x0090702f => openssl-0.9.7b
# ...
# 0x009070df => openssl-0.9.7m
# 0x0090800f => openssl-0.9.8
# 0x0090801f => openssl-0.9.8a
# 0x0090802f => openssl-0.9.8b
# ...
# 0x0090814f => openssl-0.9.8t
# 0x1000000f => openssl-1.0.0
# 0x1000004f => openssl-1.0.0d
# 0x1000007f => openssl-1.0.0g
```

You can use it like this:

```
 if (Net::SSLeay::SSLeay() < 0x0090800f) {
   die "you need openssl-0.9.8 or higher";
 }
```

- SSLeay_version

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  Gives version number (string) of underlaying openssl library.

  ```
  my $ver_string = Net::SSLeay::SSLeay_version($type);
  # $type
  #   SSLEAY_VERSION  - e.g. 'OpenSSL 1.0.0d 8 Feb 2011'
  #   SSLEAY_CFLAGS   - e.g. 'compiler: gcc -D_WINDLL -DOPENSSL_USE_APPLINK .....'
  #   SSLEAY_BUILT_ON - e.g. 'built on: Fri May  6 00:00:46 GMT 2011'
  #   SSLEAY_PLATFORM - e.g. 'platform: mingw'
  #   SSLEAY_DIR      - e.g. 'OPENSSLDIR: "z:/...."'
  #
  # returns: string

  Net::SSLeay::SSLeay_version();
  #is equivalent to
  Net::SSLeay::SSLeay_version(SSLEAY_VERSION);
  ```

  Check openssl doc <https://www.openssl.org/docs/man1.0.2/crypto/SSLeay_version.html>

- OpenSSL_version_num

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0

  Gives version number (numeric) of underlaying openssl library. See “SSLeay” for interpreting the result.

  ```
  my $ver_number = Net::SSLeay::OpenSSL_version_num();
  # returns: the number identifying the openssl release
  ```

- OpenSSL_version

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0

  Gives version number (string) of underlaying openssl library.

```
my $ver_string = Net::SSLeay::OpenSSL_version($t);
# $t
#   OPENSSL_VERSION     - e.g. 'OpenSSL 1.1.0g  2 Nov 2017'
#   OPENSSL_CFLAGS      - e.g. 'compiler: cc -DDSO_DLFCN -DHAVE_DLFCN_H .....'
#   OPENSSL_BUILT_ON    - e.g. 'built on: reproducible build, date unspecified'
#   OPENSSL_PLATFORM    - e.g. 'platform: darwin64-x86_64-cc'
#   OPENSSL_DIR         - e.g. 'OPENSSLDIR: "/opt/openssl-1.1.0g"'
#   OPENSSL_ENGINES_DIR - e.g. 'ENGINESDIR: "/opt/openssl-1.1.0g/lib/engines-1.1
#
# returns: string

Net::SSLeay::OpenSSL_version();
#is equivalent to
Net::SSLeay::OpenSSL_version(OPENSSL_VERSION);
```

Check openssl doc <https://www.openssl.org/docs/crypto/OpenSSL_version.html>

Low level API: Initialization related functions

- library_init

  Initialize SSL library by registering algorithms.

  ```
  my $rv = Net::SSLeay::library_init();
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_library_init.html>

  While the original function from OpenSSL always returns 1, Net::SSLeay adds a wrapper around it to make sure that the OpenSSL function is only called once. Thus the function will return 1 if initialization was done and 0 if not, i.e. if initialization was done already before.

- add_ssl_algorithms

  The alias for "library_init"

  ```
  Net::SSLeay::add_ssl_algorithms();
  ```

- OpenSSL_add_ssl_algorithms

  The alias for "library_init"

  ```
  Net::SSLeay::OpenSSL_add_ssl_algorithms();
  ```

- SSLeay_add_ssl_algorithms

  The alias for "library_init"

  ```
  Net::SSLeay::SSLeay_add_ssl_algorithms();
  ```

- load_error_strings

  Registers the error strings for all libcrypto + libssl related functions.

  ```
  Net::SSLeay::load_error_strings();
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/ERR_load_crypto_strings.html>

- ERR_load_crypto_strings

  Registers the error strings for all libcrypto functions. No need to call this function if you have already called "load_error_strings".

```
Net::SSLeay::ERR_load_crypto_strings();
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/crypto/ERR_load_crypto_strings.html>

- ERR_load_RAND_strings

  Registers the error strings for RAND related functions. No need to call this function if you have already called "load_error_strings".

  ```
  Net::SSLeay::ERR_load_RAND_strings();
  #
  # returns: no return value
  ```

- ERR_load_SSL_strings

  Registers the error strings for SSL related functions. No need to call this function if you have already called "load_error_strings".

  ```
  Net::SSLeay::ERR_load_SSL_strings();
  #
  # returns: no return value
  ```

- OpenSSL_add_all_algorithms

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Add algorithms to internal table.

  ```
  Net::SSLeay::OpenSSL_add_all_algorithms();
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/OpenSSL_add_all_algorithms.html>

- OPENSSL_add_all_algorithms_conf

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Similar to "OpenSSL_add_all_algorithms" − will ALWAYS load the config file

  ```
  Net::SSLeay::OPENSSL_add_all_algorithms_conf();
  #
  # returns: no return value
  ```

- OPENSSL_add_all_algorithms_noconf

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Similar to "OpenSSL_add_all_algorithms" − will NEVER load the config file

  ```
  Net::SSLeay::OPENSSL_add_all_algorithms_noconf();
  #
  # returns: no return value
  ```

Low level API: ERR_* and SSL_alert_* related functions

NOTE: Please note that SSL_alert_* function have "SSL_" part stripped from their names.

- ERR_clear_error

  Clear the error queue.

  ```
  Net::SSLeay::ERR_clear_error();
  #
  # returns: no return value
  ```

Check openssl doc <http://www.openssl.org/docs/crypto/ERR_clear_error.html>

- ERR_error_string

  Generates a human-readable string representing the error code `$error`.

  ```
  my $rv = Net::SSLeay::ERR_error_string($error);
  # $error - (unsigned integer) error code
  #
  # returns: string
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/ERR_error_string.html>

- ERR_get_error

  Returns the earliest error code from the thread's error queue and removes the entry. This function can be called repeatedly until there are no more error codes to return.

  ```
  my $rv = Net::SSLeay::ERR_get_error();
  #
  # returns: (unsigned integer) error code
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/ERR_get_error.html>

- ERR_peek_error

  Returns the earliest error code from the thread's error queue without modifying it.

  ```
  my $rv = Net::SSLeay::ERR_peek_error();
  #
  # returns: (unsigned integer) error code
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/ERR_get_error.html>

- ERR_put_error

  Adds an error code to the thread's error queue. It signals that the error of `$reason` code reason occurred in function `$func` of library `$lib`, in line number `$line` of `$file`.

  ```
  Net::SSLeay::ERR_put_error($lib, $func, $reason, $file, $line);
  # $lib - (integer) library id (check openssl/err.h for constants e.g. ERR_LIB_SS
  # $func - (integer) function id (check openssl/ssl.h for constants e.g. SSL_F_SS
  # $reason - (integer) reason id (check openssl/ssl.h for constants e.g. SSL_R_SS
  # $file - (string) file name
  # $line - (integer) line number in $file
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/ERR_put_error.html> and <http://www.openssl.org/docs/crypto/err.html>

- alert_desc_string

  Returns a two letter string as a short form describing the reason of the alert specified by value.

  ```
  my $rv = Net::SSLeay::alert_desc_string($value);
  # $value - (integer) allert id (check openssl/ssl.h for SSL3_AD_* and TLS1_AD_*
  #
  # returns: description string (2 letters)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_alert_type_string.html>

- alert_desc_string_long

  Returns a string describing the reason of the alert specified by value.

```
my $rv = Net::SSLeay::alert_desc_string_long($value);
# $value - (integer) allert id (check openssl/ssl.h for SSL3_AD_* and TLS1_AD_*
#
# returns: description string
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_alert_type_string.html>

- alert_type_string

  Returns a one letter string indicating the type of the alert specified by value.

```
my $rv = Net::SSLeay::alert_type_string($value);
# $value - (integer) allert id (check openssl/ssl.h for SSL3_AD_* and TLS1_AD_*
#
# returns: string (1 letter)
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_alert_type_string.html>

- alert_type_string_long

  Returns a string indicating the type of the alert specified by value.

```
my $rv = Net::SSLeay::alert_type_string_long($value);
# $value - (integer) allert id (check openssl/ssl.h for SSL3_AD_* and TLS1_AD_*
#
# returns: string
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_alert_type_string.html>

Low level API: SSL_METHOD_* related functions

- SSLv23_method, SSLv23_server_method and SSLv23_client_method

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before.

  Returns SSL_METHOD structure corresponding to general-purpose version-flexible TLS method, the return value can be later used as a param of "CTX_new_with_method".

  NOTE: Consider using TLS_method, TLS_server_method or TLS_client_method with new code.

```
my $rv = Net::SSLeay::SSLv2_method();
#
# returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
```

- SSLv2_method

  Returns SSL_METHOD structure corresponding to SSLv2 method, the return value can be later used as a param of "CTX_new_with_method". Only available where supported by the underlying openssl.

```
my $rv = Net::SSLeay::SSLv2_method();
#
# returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
```

- SSLv3_method

  Returns SSL_METHOD structure corresponding to SSLv3 method, the return value can be later used as a param of "CTX_new_with_method".

```
my $rv = Net::SSLeay::SSLv3_method();
#
# returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

- TLSv1_method, TLSv1_server_method and TLSv1_client_method

  COMPATIBILITY: Server and client methods not available in Net−SSLeay−1.82 and before.

Returns SSL_METHOD structure corresponding to TLSv1 method, the return value can be later used as a param of "CTX_new_with_method".

```
my $rv = Net::SSLeay::TLSv1_method();
#
# returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

- TLSv1_1_method, TLSv1_1_server_method and TLSv1_1_client_method

  COMPATIBILITY: Server and client methods not available in Net−SSLeay−1.82 and before.

  Returns SSL_METHOD structure corresponding to TLSv1_1 method, the return value can be later used as a param of "CTX_new_with_method". Only available where supported by the underlying openssl.

  ```
  my $rv = Net::SSLeay::TLSv1_1_method();
  #
  # returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

- TLSv1_2_method, TLSv1_2_server_method and TLSv1_2_client_method

  COMPATIBILITY: Server and client methods not available in Net−SSLeay−1.82 and before.

  Returns SSL_METHOD structure corresponding to TLSv1_2 method, the return value can be later used as a param of "CTX_new_with_method". Only available where supported by the underlying openssl.

  ```
  my $rv = Net::SSLeay::TLSv1_2_method();
  #
  # returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

- TLS_method, TLS_server_method and TLS_client_method

  COMPATIBILITY: Not available in Net−SSLeay−1.82 and before.

  Returns SSL_METHOD structure corresponding to general-purpose version-flexible TLS method, the return value can be later used as a param of "CTX_new_with_method". Only available where supported by the underlying openssl.

  ```
  my $rv = Net::SSLeay::TLS_method();
  #
  # returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

Low level API: ENGINE_* related functions

- ENGINE_load_builtin_engines

  COMPATIBILITY: Requires an OpenSSL build with dynamic engine loading support.

  Load all bundled ENGINEs into memory and make them visible.

  ```
  Net::SSLeay::ENGINE_load_builtin_engines();
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/engine.html>

- ENGINE_register_all_complete

  COMPATIBILITY: Requires an OpenSSL build with dynamic engine loading support.

  Register all loaded ENGINEs for every algorithm they collectively implement.

```
Net::SSLeay::ENGINE_register_all_complete();
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/crypto/engine.html>

- ENGINE_set_default

  COMPATIBILITY: Requires an OpenSSL build with dynamic engine loading support.

  Set default engine to `$e` + set its flags to `$flags`.

  ```
  my $rv = Net::SSLeay::ENGINE_set_default($e, $flags);
  # $e - value corresponding to openssl's ENGINE structure
  # $flags - (integer) engine flags
  #          flags value can be made by bitwise "OR"ing:
  #          0x0001 - ENGINE_METHOD_RSA
  #          0x0002 - ENGINE_METHOD_DSA
  #          0x0004 - ENGINE_METHOD_DH
  #          0x0008 - ENGINE_METHOD_RAND
  #          0x0010 - ENGINE_METHOD_ECDH
  #          0x0020 - ENGINE_METHOD_ECDSA
  #          0x0040 - ENGINE_METHOD_CIPHERS
  #          0x0080 - ENGINE_METHOD_DIGESTS
  #          0x0100 - ENGINE_METHOD_STORE
  #          0x0200 - ENGINE_METHOD_PKEY_METHS
  #          0x0400 - ENGINE_METHOD_PKEY_ASN1_METHS
  #          Obvious all-or-nothing cases:
  #          0xFFFF - ENGINE_METHOD_ALL
  #          0x0000 - ENGINE_METHOD_NONE
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/engine.html>

- ENGINE_by_id

  Get ENGINE by its identification `$id`.

  COMPATIBILITY: Requires an OpenSSL build with dynamic engine loading support.

  ```
  my $rv = Net::SSLeay::ENGINE_by_id($id);
  # $id - (string) engine identification e.g. "dynamic"
  #
  # returns: value corresponding to openssl's ENGINE structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/engine.html>

Low level API: EVP_PKEY_* related functions

- EVP_PKEY_copy_parameters

  Copies the parameters from key `$from` to key `$to`.

  ```
  my $rv = Net::SSLeay::EVP_PKEY_copy_parameters($to, $from);
  # $to - value corresponding to openssl's EVP_PKEY structure
  # $from - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_PKEY_cmp.html>

- EVP_PKEY_new

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Creates a new EVP_PKEY structure.

  ```
  my $rv = Net::SSLeay::EVP_PKEY_new();
  #
  # returns: value corresponding to openssl's EVP_PKEY structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_PKEY_new.html>

- EVP_PKEY_free

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Free an allocated EVP_PKEY structure.

  ```
  Net::SSLeay::EVP_PKEY_free($pkey);
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_PKEY_new.html>

- EVP_PKEY_assign_RSA

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Set the key referenced by `$pkey` to `$key`

  NOTE: No reference counter will be increased, i.e. `$key` will be freed if `$pkey` is freed.

  ```
  my $rv = Net::SSLeay::EVP_PKEY_assign_RSA($pkey, $key);
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  # $key - value corresponding to openssl's RSA structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_PKEY_assign_RSA.html>

- EVP_PKEY_assign_EC_KEY

  COMPATIBILITY: not available in Net−SSLeay−1.74 and before

  Set the key referenced by `$pkey` to `$key`

  NOTE: No reference counter will be increased, i.e. `$key` will be freed if `$pkey` is freed.

  ```
  my $rv = Net::SSLeay::EVP_PKEY_assign_EC_KEY($pkey, $key);
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  # $key - value corresponding to openssl's EC_KEY structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_PKEY_assign_EC_KEY.html>

- EVP_PKEY_bits

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns the size of the key `$pkey` in bits.

```
my $rv = Net::SSLeay::EVP_PKEY_bits($pkey);
# $pkey - value corresponding to openssl's EVP_PKEY structure
#
# returns: size in bits
```

- EVP_PKEY_size

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns the maximum size of a signature in bytes. The actual signature may be smaller.

  ```
  my $rv = Net::SSLeay::EVP_PKEY_size($pkey);
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: the maximum size in bytes
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_SignInit.html>

- EVP_PKEY_id

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−1.0.0

  Returns `$pkey` type (integer value of corresponding NID).

  ```
  my $rv = Net::SSLeay::EVP_PKEY_id($pkey);
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: (integer) key type
  ```

  Example:

  ```
  my $pubkey = Net::SSLeay::X509_get_pubkey($x509);
  my $type = Net::SSLeay::EVP_PKEY_id($pubkey);
  print Net::SSLeay::OBJ_nid2sn($type);             #prints e.g. 'rsaEncryption'
  ```

Low level API: PEM_* related functions

Check openssl doc <http://www.openssl.org/docs/crypto/pem.html>

- PEM_read_bio_X509

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads PEM formatted X509 certificate via given BIO structure.

  ```
  my $rv = Net::SSLeay::PEM_read_bio_X509($bio);
  # $bio - value corresponding to openssl's BIO structure
  #
  # returns: value corresponding to openssl's X509 structure (0 on failure)
  ```

  Example:

  ```
  my $bio = Net::SSLeay::BIO_new_file($filename, 'r');
  my $x509 = Net::SSLeay::PEM_read_bio_X509($bio);
  Net::SSLeay::BIO_free($bio);
  ```

- PEM_read_bio_X509_REQ

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads PEM formatted X509_REQ object via given BIO structure.

  ```
  my $rv = Net::SSLeay::PEM_read_bio_X509_REQ($bio, $x=NULL, $cb=NULL, $u=NULL);
  # $bio - value corresponding to openssl's BIO structure
  #
  # returns: value corresponding to openssl's X509_REQ structure (0 on failure)
  ```

Example:

```
my $bio = Net::SSLeay::BIO_new_file($filename, 'r');
my $x509_req = Net::SSLeay::PEM_read_bio_X509_REQ($bio);
Net::SSLeay::BIO_free($bio);
```

- PEM_read_bio_DHparams

  Reads DH structure from BIO.

```
my $rv = Net::SSLeay::PEM_read_bio_DHparams($bio);
# $bio - value corresponding to openssl's BIO structure
#
# returns: value corresponding to openssl's DH structure (0 on failure)
```

- PEM_read_bio_X509_CRL

  Reads X509_CRL structure from BIO.

```
my $rv = Net::SSLeay::PEM_read_bio_X509_CRL($bio);
# $bio - value corresponding to openssl's BIO structure
#
# returns: value corresponding to openssl's X509_CRL structure (0 on failure)
```

- PEM_read_bio_PrivateKey

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads PEM formatted private key via given BIO structure.

```
my $rv = Net::SSLeay::PEM_read_bio_PrivateKey($bio, $cb, $data);
# $bio - value corresponding to openssl's BIO structure
# $cb - reference to perl callback function
# $data - data that will be passed to callback function (see examples below)
#
# returns: value corresponding to openssl's EVP_PKEY structure (0 on failure)
```

  Example:

```
my $bio = Net::SSLeay::BIO_new_file($filename, 'r');
my $privkey = Net::SSLeay::PEM_read_bio_PrivateKey($bio); #ask for password if n
Net::SSLeay::BIO_free($bio);
```

  To use password you have the following options:

```
$privkey = Net::SSLeay::PEM_read_bio_PrivateKey($bio, \&callback_func); # use ca
$privkey = Net::SSLeay::PEM_read_bio_PrivateKey($bio, \&callback_func, $data); #
$privkey = Net::SSLeay::PEM_read_bio_PrivateKey($bio, undef, "secret"); # use pa
$privkey = Net::SSLeay::PEM_read_bio_PrivateKey($bio, undef, "");       # use em
```

  Callback function signature:

```
sub callback_func {
  my ($max_passwd_size, $rwflag, $data) = @_;
  # $max_passwd_size - maximum size of returned password (longer values will be
  # $rwflag - indicates whether we are loading (0) or storing (1) - for PEM_read
  # $data - the data passed to PEM_read_bio_PrivateKey as 3rd parameter

  return "secret";
}
```

- PEM_X509_INFO_read_bio

  Reads a BIO containing a PEM formatted file into a STACK_OF(X509_INFO) structure.

```
my $rv = Net::SSLeay::PEM_X509_INFO_read_bio($bio);
# $bio - value corresponding to openssl's BIO structure
#
# returns: value corresponding to openssl's STACK_OF(X509_INFO) structure.
```

Example:

```
my $bio = Net::SSLeay::BIO_new_file($filename, 'r');
my $sk_x509_info = Net::SSLeay::PEM_X509_INFO_read_bio($bio);
Net::SSLeay::BIO_free($bio);
```

- PEM_get_string_X509

  NOTE: Does not exactly correspond to any low level API function

  Converts/exports X509 certificate to string (PEM format).

  ```
  Net::SSLeay::PEM_get_string_X509($x509);
  # $x509 - value corresponding to openssl's X509 structure
  #
  # returns: string with $x509 in PEM format
  ```

- PEM_get_string_PrivateKey

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Converts public key `$pk` into PEM formatted string (optionally protected with password).

  ```
  my $rv = Net::SSLeay::PEM_get_string_PrivateKey($pk, $passwd, $enc_alg);
  # $pk - value corresponding to openssl's EVP_PKEY structure
  # $passwd - [optional] (string) password to use for key encryption
  # $enc_alg - [optional] algorithm to use for key encryption (default: DES_CBC) -
  #
  # returns: PEM formatted string
  ```

  Examples:

  ```
  $pem_privkey = Net::SSLeay::PEM_get_string_PrivateKey($pk);
  $pem_privkey = Net::SSLeay::PEM_get_string_PrivateKey($pk, "secret");
  $pem_privkey = Net::SSLeay::PEM_get_string_PrivateKey($pk, "secret", Net::SSLeay
  ```

- PEM_get_string_X509_CRL

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Converts X509_CRL object `$x509_crl` into PEM formatted string.

  ```
  Net::SSLeay::PEM_get_string_X509_CRL($x509_crl);
  # $x509_crl - value corresponding to openssl's X509_CRL structure
  #
  # returns: no return value
  ```

- PEM_get_string_X509_REQ

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Converts X509_REQ object `$x509_crl` into PEM formatted string.

  ```
  Net::SSLeay::PEM_get_string_X509_REQ($x509_req);
  # $x509_req - value corresponding to openssl's X509_REQ structure
  #
  # returns: no return value
  ```

Low level API: d2i_* (DER format) related functions

- d2i_X509_bio

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads DER formatted X509 certificate via given BIO structure.

  ```
  my $rv = Net::SSLeay::d2i_X509_bio($bp);
  # $bp - value corresponding to openssl's BIO structure
  #
  # returns: value corresponding to openssl's X509 structure (0 on failure)
  ```

  Example:

  ```
  my $bio = Net::SSLeay::BIO_new_file($filename, 'rb');
  my $x509 = Net::SSLeay::d2i_X509_bio($bio);
  Net::SSLeay::BIO_free($bio);
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/d2i_X509.html>

- d2i_X509_CRL_bio

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads DER formatted X509_CRL object via given BIO structure.

  ```
  my $rv = Net::SSLeay::d2i_X509_CRL_bio($bp);
  # $bp - value corresponding to openssl's BIO structure
  #
  # returns: value corresponding to openssl's X509_CRL structure (0 on failure)
  ```

  Example:

  ```
  my $bio = Net::SSLeay::BIO_new_file($filename, 'rb');
  my $x509_crl = Net::SSLeay::d2i_X509_CRL_bio($bio);
  Net::SSLeay::BIO_free($bio);
  ```

- d2i_X509_REQ_bio

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads DER formatted X509_REQ object via given BIO structure.

  ```
  my $rv = Net::SSLeay::d2i_X509_REQ_bio($bp);
  # $bp - value corresponding to openssl's BIO structure
  #
  # returns: value corresponding to openssl's X509_REQ structure (0 on failure)
  ```

  Example:

  ```
  my $bio = Net::SSLeay::BIO_new_file($filename, 'rb');
  my $x509_req = Net::SSLeay::d2i_X509_REQ_bio($bio);
  Net::SSLeay::BIO_free($bio);
  ```

Low level API: PKCS12 related functions

- P_PKCS12_load_file

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Loads X509 certificate + private key + certificates of CA chain (if present in PKCS12 file).

```
my ($privkey, $cert, @cachain) = Net::SSLeay::P_PKCS12_load_file($filename, $loa
# $filename - name of PKCS12 file
# $load_chain - [optional] whether load (1) or not(0) CA chain (default: 0)
# $password - [optional] password for private key
#
# returns: triplet ($privkey, $cert, @cachain)
#          $privkey - value corresponding to openssl's EVP_PKEY structure
#          $cert - value corresponding to openssl's X509 structure
#          @cachain - array of values corresponding to openssl's X509 structure
```

IMPORTANT NOTE: after you do the job you need to call **X509_free()** on `$privkey` + all members of `@cachain` and **EVP_PKEY_free()** on `$privkey`.

Examples:

```
my ($privkey, $cert) = Net::SSLeay::P_PKCS12_load_file($filename);
#or
my ($privkey, $cert) = Net::SSLeay::P_PKCS12_load_file($filename, 0, $password);
#or
my ($privkey, $cert, @cachain) = Net::SSLeay::P_PKCS12_load_file($filename, 1);
#or
my ($privkey, $cert, @cachain) = Net::SSLeay::P_PKCS12_load_file($filename, 1, $

#BEWARE: THIS IS WRONG - MEMORY LEAKS! (you cannot free @cachain items)
my ($privkey, $cert) = Net::SSLeay::P_PKCS12_load_file($filename, 1, $password);
```

NOTE With some combinations of Windows, perl, compiler and compiler options, you may see a runtime error "no OPENSSL_Applink", when calling Net::SSLeay::P_PKCS12_load_file. See README.Win32 for more details.

Low level API: SESSION_* related functions

- d2i_SSL_SESSION

  COMPATIBILITY: does not work in Net−SSLeay−1.85 and before

  Transforms the binary ASN1 representation string of an SSL/TLS session into an SSL_SESSION object.

  ```
  my $ses = Net::SSLeay::d2i_SSL_SESSION($data);
  # $data - the session as ASN1 representation string
  #
  # returns: $ses - the new SSL_SESSION
  ```

  Check openssl doc <https://www.openssl.org/docs/ssl/i2d_SSL_SESSION.html>

- i2d_SSL_SESSION

  COMPATIBILITY: does not work in Net−SSLeay−1.85 and before

  Transforms the SSL_SESSION object in into the ASN1 representation and returns it as string.

  ```
  my $data = Net::SSLeay::i2d_SSL_SESSION($ses);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: $data - session as string
  ```

  Check openssl doc <https://www.openssl.org/docs/ssl/d2i_SSL_SESSION.html>

- SESSION_new

  Creates a new SSL_SESSION structure.

```
my $rv = Net::SSLeay::SESSION_new();
#
# returns: value corresponding to openssl's SSL_SESSION structure (0 on failure)
```

- SESSION_free

  Free an allocated SSL_SESSION structure.

  ```
  Net::SSLeay::SESSION_free($ses);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_free.html>

- SESSION_up_ref

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.0 or LibreSSL 2.7.0

  Increases the reference counter on a SSL_SESSION structure.

  ```
  Net::SSLeay::SESSION_up_ref($ses);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: 1 on success else 0
  ```

  Check openssl doc <https://www.openssl.org/docs/ssl/SSL_SESSION_up_ref.html>

- SESSION_dup

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Duplicates a SSL_SESSION structure.

  ```
  Net::SSLeay::SESSION_dup($ses);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: the duplicated session
  ```

  Check openssl doc <https://www.openssl.org/docs/ssl/SSL_SESSION_dup.html>

- SESSION_is_resumable

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Determine whether an SSL_SESSION object can be used for resumption.

  ```
  Net::SSLeay::SESSION_is_resumable($ses);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: (integer) 1 if it can or 0 if not
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_SESSION_is_resumable.html>

- SESSION_cmp

  Compare two SSL_SESSION structures.

```
my $rv = Net::SSLeay::SESSION_cmp($sesa, $sesb);
# $sesa - value corresponding to openssl's SSL_SESSION structure
# $sesb - value corresponding to openssl's SSL_SESSION structure
#
# returns: 0 if the two structures are the same
```

NOTE: Not available in openssl 1.0 or later

- SESSION_get_app_data

  Can be used to get application defined value/data.

  ```
  my $rv = Net::SSLeay::SESSION_get_app_data($ses);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: string/buffer/pointer ???
  ```

- SESSION_set_app_data

  Can be used to set some application defined value/data.

  ```
  my $rv = Net::SSLeay::SESSION_set_app_data($s, $a);
  # $s - value corresponding to openssl's SSL_SESSION structure
  # $a - (string/buffer/pointer ???) data
  #
  # returns: ???
  ```

- SESSION_get_ex_data

  Is used to retrieve the information for `$idx` from session `$ses`.

  ```
  my $rv = Net::SSLeay::SESSION_get_ex_data($ses, $idx);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  # $idx - (integer) index for application specific data
  #
  # returns: pointer to ???
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_ex_new_index.html>

- SESSION_set_ex_data

  Is used to store application data at arg for idx into the session object.

  ```
  my $rv = Net::SSLeay::SESSION_set_ex_data($ss, $idx, $data);
  # $ss - value corresponding to openssl's SSL_SESSION structure
  # $idx - (integer) ???
  # $data - (pointer) ???
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_ex_new_index.html>

- SESSION_get_ex_new_index

  Is used to register a new index for application specific data.

```
my $rv = Net::SSLeay::SESSION_get_ex_new_index($argl, $argp, $new_func, $dup_fur
# $argl - (long) ???
# $argp - (pointer) ???
# $new_func - function pointer ??? (CRYPTO_EX_new *)
# $dup_func - function pointer ??? (CRYPTO_EX_dup *)
# $free_func - function pointer ??? (CRYPTO_EX_free *)
#
# returns: (integer) ???
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_ex_new_index.html>

- SESSION_get_master_key

  NOTE: Does not exactly correspond to any low level API function

  Returns 'master_key' value from SSL_SESSION structure `$s`

  ```
  Net::SSLeay::SESSION_get_master_key($s);
  # $s - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: master key (binary data)
  ```

- SESSION_set_master_key

  Sets 'master_key' value for SSL_SESSION structure `$s`

  ```
  Net::SSLeay::SESSION_set_master_key($s, $key);
  # $s - value corresponding to openssl's SSL_SESSION structure
  # $key - master key (binary data)
  #
  # returns: no return value
  ```

  Not available with OpenSSL 1.1 and later. Code that previously used
  SESSION_set_master_key must now set `$secret` in the session_secret
  callback set with SSL_set_session_secret_cb.

- SESSION_get_time

  Returns the time at which the session s was established. The time is given in seconds since 1.1.1970.

  ```
  my $rv = Net::SSLeay::SESSION_get_time($s);
  # $s - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: timestamp (seconds since 1.1.1970)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_time.html>

- get_time

  Technically the same functionality as "SESSION_get_time".

  ```
  my $rv = Net::SSLeay::get_time($s);
  ```

- SESSION_get_timeout

  Returns the timeout value set for session `$s` in seconds.

  ```
  my $rv = Net::SSLeay::SESSION_get_timeout($s);
  # $s - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: timeout (in seconds)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_time.html>

- get_timeout

Technically the same functionality as "SESSION_get_timeout".

```
my $rv = Net::SSLeay::get_timeout($s);
```

- SESSION_print

  NOTE: Does not exactly correspond to any low level API function

  Prints session details (e.g. protocol version, cipher, session-id ...) to BIO.

  ```
  my $rv = Net::SSLeay::SESSION_print($fp, $ses);
  # $fp - value corresponding to openssl's BIO structure
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: 1 on success, 0 on failure
  ```

  You have to use necessary BIO functions like this:

  ```
  # let us have $ssl corresponding to openssl's SSL structure
  my $ses = Net::SSLeay::get_session($ssl);
  my $bio = Net::SSLeay::BIO_new(&Net::SSLeay::BIO_s_mem);
  Net::SSLeay::SESSION_print($bio, $ses);
  print Net::SSLeay::BIO_read($bio);
  ```

- SESSION_print_fp

  Prints session details (e.g. protocol version, cipher, session-id ...) to file handle.

  ```
  my $rv = Net::SSLeay::SESSION_print_fp($fp, $ses);
  # $fp - perl file handle
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Example:

  ```
  # let us have $ssl corresponding to openssl's SSL structure
  my $ses = Net::SSLeay::get_session($ssl);
  open my $fh, ">", "output.txt";
  Net::SSLeay::SESSION_print_fp($fh,$ses);
  ```

- SESSION_set_time

  Replaces the creation time of the session s with the chosen value `$t` (seconds since 1.1.1970).

  ```
  my $rv = Net::SSLeay::SESSION_set_time($ses, $t);
  # $ses - value corresponding to openssl's SSL_SESSION structure
  # $t - time value
  #
  # returns: 1 on success
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_time.html>

- set_time

  Technically the same functionality as "SESSION_set_time".

  ```
  my $rv = Net::SSLeay::set_time($ses, $t);
  ```

- SESSION_set_timeout

  Sets the timeout value for session s in seconds to `$t`.

```
my $rv = Net::SSLeay::SESSION_set_timeout($s, $t);
# $s - value corresponding to openssl's SSL_SESSION structure
# $t - timeout (in seconds)
#
# returns: 1 on success
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_SESSION_get_time.html>

- set_timeout

  Technically the same functionality as "SESSION_set_timeout".

  ```
  my $rv = Net::SSLeay::set_timeout($ses, $t);
  ```

Low level API: SSL_CTX_* related functions

NOTE: Please note that the function described in this chapter have "SSL_" part stripped from their original openssl names.

- CTX_add_client_CA

  Adds the CA name extracted from `$cacert` to the list of CAs sent to the client when requesting a client certificate for `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_add_client_CA($ctx, $cacert);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $cacert - value corresponding to openssl's X509 structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_client_CA_list.html>

- CTX_add_extra_chain_cert

  Adds the certificate `$x509` to the certificate chain presented together with the certificate. Several certificates can be added one after the other.

  ```
  my $rv = Net::SSLeay::CTX_add_extra_chain_cert($ctx, $x509);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $x509 - value corresponding to openssl's X509 structure
  #
  # returns: 1 on success, check out the error stack to find out the reason for fa
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_add_extra_chain_cert.html>

- CTX_add_session

  Adds the session `$ses` to the context `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_add_session($ctx, $ses);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_add_session.html>

- CTX_callback_ctrl

  ??? (more info needed)

```
my $rv = Net::SSLeay::CTX_callback_ctrl($ctx, $cmd, $fp);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $cmd - (integer) command id
# $fp - (function pointer) ???
#
# returns: ???
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_ctrl.html>

- CTX_check_private_key

  Checks the consistency of a private key with the corresponding certificate loaded into `$ctx`.

```
my $rv = Net::SSLeay::CTX_check_private_key($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_ctrl

  Internal handling function for SSL_CTX objects.

  BEWARE: openssl doc says: This function should never be called directly!

```
my $rv = Net::SSLeay::CTX_ctrl($ctx, $cmd, $larg, $parg);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $cmd - (integer) command id
# $larg - (integer) long ???
# $parg - (string/pointer) ???
#
# returns: (long) result of given command ???

#valid $cmd values
 1 - SSL_CTRL_NEED_TMP_RSA
 2 - SSL_CTRL_SET_TMP_RSA
 3 - SSL_CTRL_SET_TMP_DH
 4 - SSL_CTRL_SET_TMP_ECDH
 5 - SSL_CTRL_SET_TMP_RSA_CB
 6 - SSL_CTRL_SET_TMP_DH_CB
 7 - SSL_CTRL_SET_TMP_ECDH_CB
 8 - SSL_CTRL_GET_SESSION_REUSED
 9 - SSL_CTRL_GET_CLIENT_CERT_REQUEST
10 - SSL_CTRL_GET_NUM_RENEGOTIATIONS
11 - SSL_CTRL_CLEAR_NUM_RENEGOTIATIONS
12 - SSL_CTRL_GET_TOTAL_RENEGOTIATIONS
13 - SSL_CTRL_GET_FLAGS
14 - SSL_CTRL_EXTRA_CHAIN_CERT
15 - SSL_CTRL_SET_MSG_CALLBACK
16 - SSL_CTRL_SET_MSG_CALLBACK_ARG
17 - SSL_CTRL_SET_MTU
20 - SSL_CTRL_SESS_NUMBER
21 - SSL_CTRL_SESS_CONNECT
22 - SSL_CTRL_SESS_CONNECT_GOOD
23 - SSL_CTRL_SESS_CONNECT_RENEGOTIATE
24 - SSL_CTRL_SESS_ACCEPT
25 - SSL_CTRL_SESS_ACCEPT_GOOD
26 - SSL_CTRL_SESS_ACCEPT_RENEGOTIATE
```

```
27 - SSL_CTRL_SESS_HIT
28 - SSL_CTRL_SESS_CB_HIT
29 - SSL_CTRL_SESS_MISSES
30 - SSL_CTRL_SESS_TIMEOUTS
31 - SSL_CTRL_SESS_CACHE_FULL
32 - SSL_CTRL_OPTIONS
33 - SSL_CTRL_MODE
40 - SSL_CTRL_GET_READ_AHEAD
41 - SSL_CTRL_SET_READ_AHEAD
42 - SSL_CTRL_SET_SESS_CACHE_SIZE
43 - SSL_CTRL_GET_SESS_CACHE_SIZE
44 - SSL_CTRL_SET_SESS_CACHE_MODE
45 - SSL_CTRL_GET_SESS_CACHE_MODE
50 - SSL_CTRL_GET_MAX_CERT_LIST
51 - SSL_CTRL_SET_MAX_CERT_LIST
52 - SSL_CTRL_SET_MAX_SEND_FRAGMENT
53 - SSL_CTRL_SET_TLSEXT_SERVERNAME_CB
54 - SSL_CTRL_SET_TLSEXT_SERVERNAME_ARG
55 - SSL_CTRL_SET_TLSEXT_HOSTNAME
56 - SSL_CTRL_SET_TLSEXT_DEBUG_CB
57 - SSL_CTRL_SET_TLSEXT_DEBUG_ARG
58 - SSL_CTRL_GET_TLSEXT_TICKET_KEYS
59 - SSL_CTRL_SET_TLSEXT_TICKET_KEYS
60 - SSL_CTRL_SET_TLSEXT_OPAQUE_PRF_INPUT
61 - SSL_CTRL_SET_TLSEXT_OPAQUE_PRF_INPUT_CB
62 - SSL_CTRL_SET_TLSEXT_OPAQUE_PRF_INPUT_CB_ARG
63 - SSL_CTRL_SET_TLSEXT_STATUS_REQ_CB
64 - SSL_CTRL_SET_TLSEXT_STATUS_REQ_CB_ARG
65 - SSL_CTRL_SET_TLSEXT_STATUS_REQ_TYPE
66 - SSL_CTRL_GET_TLSEXT_STATUS_REQ_EXTS
67 - SSL_CTRL_SET_TLSEXT_STATUS_REQ_EXTS
68 - SSL_CTRL_GET_TLSEXT_STATUS_REQ_IDS
69 - SSL_CTRL_SET_TLSEXT_STATUS_REQ_IDS
70 - SSL_CTRL_GET_TLSEXT_STATUS_REQ_OCSP_RESP
71 - SSL_CTRL_SET_TLSEXT_STATUS_REQ_OCSP_RESP
72 - SSL_CTRL_SET_TLSEXT_TICKET_KEY_CB
73 - DTLS_CTRL_GET_TIMEOUT
74 - DTLS_CTRL_HANDLE_TIMEOUT
75 - DTLS_CTRL_LISTEN
76 - SSL_CTRL_GET_RI_SUPPORT
77 - SSL_CTRL_CLEAR_OPTIONS
78 - SSL_CTRL_CLEAR_MODE

82 - SSL_CTRL_GET_EXTRA_CHAIN_CERTS
83 - SSL_CTRL_CLEAR_EXTRA_CHAIN_CERTS

88 - SSL_CTRL_CHAIN
89 - SSL_CTRL_CHAIN_CERT

90 - SSL_CTRL_GET_CURVES
91 - SSL_CTRL_SET_CURVES
92 - SSL_CTRL_SET_CURVES_LIST
93 - SSL_CTRL_GET_SHARED_CURVE
94 - SSL_CTRL_SET_ECDH_AUTO
```

```
97 - SSL_CTRL_SET_SIGALGS
98 - SSL_CTRL_SET_SIGALGS_LIST
99 - SSL_CTRL_CERT_FLAGS
100 - SSL_CTRL_CLEAR_CERT_FLAGS
101 - SSL_CTRL_SET_CLIENT_SIGALGS
102 - SSL_CTRL_SET_CLIENT_SIGALGS_LIST
103 - SSL_CTRL_GET_CLIENT_CERT_TYPES
104 - SSL_CTRL_SET_CLIENT_CERT_TYPES
105 - SSL_CTRL_BUILD_CERT_CHAIN
106 - SSL_CTRL_SET_VERIFY_CERT_STORE
107 - SSL_CTRL_SET_CHAIN_CERT_STORE
108 - SSL_CTRL_GET_PEER_SIGNATURE_NID
109 - SSL_CTRL_GET_SERVER_TMP_KEY
110 - SSL_CTRL_GET_RAW_CIPHERLIST
111 - SSL_CTRL_GET_EC_POINT_FORMATS
112 - SSL_CTRL_GET_TLSA_RECORD
113 - SSL_CTRL_SET_TLSA_RECORD
114 - SSL_CTRL_PULL_TLSA_RECORD
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_ctrl.html>

- CTX_flush_sessions

  Causes a run through the session cache of `$ctx` to remove sessions expired at time `$tm`.

  ```
  Net::SSLeay::CTX_flush_sessions($ctx, $tm);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $tm - specifies the time which should be used for the expiration test (seconds
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_flush_sessions.html>

- CTX_free

  Free an allocated SSL_CTX object.

  ```
  Net::SSLeay::CTX_free($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_free.html>

- CTX_get_app_data

  Can be used to get application defined value/data.

  ```
  my $rv = Net::SSLeay::CTX_get_app_data($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: string/buffer/pointer ???
  ```

- CTX_set_app_data

  Can be used to set some application defined value/data.

```
my $rv = Net::SSLeay::CTX_set_app_data($ctx, $arg);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $arg - (string/buffer/pointer ???) data
#
# returns: ???
```

- CTX_get0_param

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Returns the current verification parameters.

  ```
  my $vpm = Net::SSLeay::CTX_get0_param($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: value corresponding to openssl's X509_VERIFY_PARAM structure
  ```

  Check openssl doc <https://www.openssl.org/docs/ssl/SSL_CTX_get0_param.html>

- CTX_get_cert_store

  Returns the current certificate verification storage.

  ```
  my $rv = Net::SSLeay::CTX_get_cert_store($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: value corresponding to openssl's X509_STORE structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_cert_store.html>

- CTX_get_client_CA_list

  Returns the list of client CAs explicitly set for `$ctx` using "CTX_set_client_CA_list".

  ```
  my $rv = Net::SSLeay::CTX_get_client_CA_list($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: value corresponding to openssl's X509_NAME_STACK structure (0 on fail
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_client_CA_list.html>

- CTX_get_ex_data

  Is used to retrieve the information for index `$idx` from `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_get_ex_data($ssl, $idx);
  # $ssl - value corresponding to openssl's SSL_CTX structure
  # $idx - (integer) index for application specific data
  #
  # returns: pointer to ???
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_get_ex_new_index.html>

- CTX_get_ex_new_index

  Is used to register a new index for application specific data.

```
my $rv = Net::SSLeay::CTX_get_ex_new_index($argl, $argp, $new_func, $dup_func, $
# $argl - (long) ???
# $argp - (pointer) ???
# $new_func - function pointer ??? (CRYPTO_EX_new *)
# $dup_func - function pointer ??? (CRYPTO_EX_dup *)
# $free_func - function pointer ??? (CRYPTO_EX_free *)
#
# returns: (integer) ???
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_get_ex_new_index.html>

- CTX_get_mode

  Returns the mode set for ctx.

```
my $rv = Net::SSLeay::CTX_get_mode($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: mode (bitmask)

#to decode the return value (bitmask) use:
0x00000001 corresponds to SSL_MODE_ENABLE_PARTIAL_WRITE
0x00000002 corresponds to SSL_MODE_ACCEPT_MOVING_WRITE_BUFFER
0x00000004 corresponds to SSL_MODE_AUTO_RETRY
0x00000008 corresponds to SSL_MODE_NO_AUTO_CHAIN
0x00000010 corresponds to SSL_MODE_RELEASE_BUFFERS
(note: some of the bits might not be supported by older openssl versions)
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_mode.html>

- CTX_set_mode

  Adds the mode set via bitmask in `$mode` to `$ctx`. Options already set before are not cleared.

```
my $rv = Net::SSLeay::CTX_set_mode($ctx, $mode);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $mode - mode bitmask
#
# returns: the new mode bitmask after adding $mode
```

  For bitmask details see "CTX_get_mode" (above).

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_mode.html>

- CTX_get_options

  Returns the options (bitmask) set for `$ctx`.

```
my $rv = Net::SSLeay::CTX_get_options($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: options (bitmask)
```

  BEWARE: The available constants and their values in bitmask depend on the TLS library. For example, SSL_OP_NO_TLSv1_3 became available much later than SSL_OP_NO_COMPRESS which is already deprecated by some libraries. Also, some previously used option values have been recycled and are now used for newer options. See the list of constants in this document for options Net::SSLeay currently supports.

  You are strongly encouraged to **check your TLS library** if you need to use numeric values directly. The following is a sample of historic values. It may not be correct anymore.

```
#to decode the return value (bitmask) use:
0x00000004 corresponds to SSL_OP_LEGACY_SERVER_CONNECT
0x00000800 corresponds to SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS
0x00004000 corresponds to SSL_OP_NO_TICKET
0x00010000 corresponds to SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION
0x00400000 corresponds to SSL_OP_CIPHER_SERVER_PREFERENCE
0x04000000 corresponds to SSL_OP_NO_TLSv1
```

Check openssl doc <https://www.openssl.org/docs/man1.0.2/ssl/SSL_CTX_set_options.html>

- CTX_set_options

  Adds the options set via bitmask in `$options` to ctx. Options already set before are not cleared.

  ```
  Net::SSLeay::CTX_set_options($ctx, $options);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $options - options bitmask
  #
  # returns: the new options bitmask after adding $options
  ```

  For bitmask details see "CTX_get_options" (above).

  Check openssl doc <https://www.openssl.org/docs/man1.0.2/ssl/SSL_CTX_set_options.html>

- CTX_get_quiet_shutdown

  Returns the 'quiet shutdown' setting of `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_get_quiet_shutdown($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: (integer) the current setting
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_quiet_shutdown.html>

- CTX_get_read_ahead

  ```
  my $rv = Net::SSLeay::CTX_get_read_ahead($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: (integer) read_ahead value
  ```

- CTX_get_session_cache_mode

  Returns the currently used cache mode (bitmask).

  ```
  my $rv = Net::SSLeay::CTX_get_session_cache_mode($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: mode (bitmask)
  ```

  BEWARE: SESS_CACHE_OFF and other constants are not available in Net−SSLeay−1.82 and before. If the constants are not available, the following values have historically been correct. You are strongly encouraged to **check your TLS library** for the current values.

```
#to decode the return value (bitmask) use:
0x0000 corresponds to SSL_SESS_CACHE_OFF
0x0001 corresponds to SSL_SESS_CACHE_CLIENT
0x0002 corresponds to SSL_SESS_CACHE_SERVER
0x0080 corresponds to SSL_SESS_CACHE_NO_AUTO_CLEAR
0x0100 corresponds to SSL_SESS_CACHE_NO_INTERNAL_LOOKUP
0x0200 corresponds to SSL_SESS_CACHE_NO_INTERNAL_STORE
(note: some of the bits might not be supported by older openssl versions)
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_session_cache_mode.html>

- CTX_set_session_cache_mode

  Enables/disables session caching by setting the operational mode for `$ctx` to `$mode`.

  ```
  my $rv = Net::SSLeay::CTX_set_session_cache_mode($ctx, $mode);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $mode - mode (bitmask)
  #
  # returns: previously set cache mode
  ```

  For bitmask details see "CTX_get_session_cache_mode" (above).

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_session_cache_mode.html>

- CTX_get_timeout

  Returns the currently set timeout value for `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_get_timeout($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: timeout in seconds
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_timeout.html>

- CTX_get_verify_depth

  Returns the verification depth limit currently set in `$ctx`. If no limit has been explicitly set, −1 is returned and the default value will be used.",

  ```
  my $rv = Net::SSLeay::CTX_get_verify_depth($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: depth limit currently set in $ctx, -1 if no limit has been explicitly
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_get_verify_mode.html>

- CTX_get_verify_mode

  Returns the verification mode (bitmask) currently set in `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_get_verify_mode($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: mode (bitmask)
  ```

  For bitmask details see "CTX_set_verify".

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_get_verify_mode.html>

- CTX_set_verify

  Sets the verification flags for `$ctx` to be `$mode` and specifies the verify_callback function to be used.

```
Net::SSLeay::CTX_set_verify($ctx, $mode, $callback);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $mode - mode (bitmask), see OpenSSL manual
# $callback - [optional] reference to perl callback function
#
# returns: no return value
```

Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_verify.html>

- CTX_set_post_handshake_auth

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Enable the Post-Handshake Authentication extension to be added to the ClientHello such that post-handshake authentication can be requested by the server.

```
Net::SSLeay::CTX_set_posthandshake_auth($ctx, $val);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $val - 0 then the extension is not sent, otherwise it is
#
# returns: no return value
```

  Check openssl doc https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_post_handshake_auth <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_post_handshake_auth.html>

- CTX_load_verify_locations

  Specifies the locations for `$ctx`, at which CA certificates for verification purposes are located. The certificates available via `$CAfile` and `$CApath` are trusted.

```
my $rv = Net::SSLeay::CTX_load_verify_locations($ctx, $CAfile, $CApath);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $CAfile - (string) file of CA certificates in PEM format, the file can contain
# $CApath - (string) directory containing CA certificates in PEM format (or '')
#
# returns: 1 on success, 0 on failure (check the error stack to find out the rea
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_load_verify_locations.html>

- CTX_need_tmp_RSA

  Return the result of `SSL_CTX_ctrl(ctx,SSL_CTRL_NEED_TMP_RSA,0,NULL)`

```
my $rv = Net::SSLeay::CTX_need_tmp_RSA($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: result of SSL_CTRL_NEED_TMP_RSA command
```

  Not available with OpenSSL 1.1 and later.

- CTX_new

  The same as "CTX_v23_new"

```
my $rv = Net::SSLeay::CTX_new();
#
# returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

  Not available with OpenSSL 1.1 and later.

- CTX_v2_new

  Creates a new SSL_CTX object − based on **SSLv2_method()** − as framework to establish TLS/SSL enabled connections.

  ```
  my $rv = Net::SSLeay::CTX_v2_new();
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

- CTX_v23_new

  Creates a new SSL_CTX object − based on **SSLv23_method()** − as framework to establish TLS/SSL enabled connections.

  ```
  my $rv = Net::SSLeay::CTX_v23_new();
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

- CTX_v3_new

  Creates a new SSL_CTX object − based on **SSLv3_method()** − as framework to establish TLS/SSL enabled connections.

  ```
  my $rv = Net::SSLeay::CTX_v3_new();
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

- CTX_tlsv1_new

  Creates a new SSL_CTX object − based on **TLSv1_method()** − as framework to establish TLS/SSL enabled connections.

  ```
  my $rv = Net::SSLeay::CTX_tlsv1_new();
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

- CTX_tlsv1_1_new

  Creates a new SSL_CTX object − based on **TLSv1_1_method()** − as framework to establish TLS/SSL enabled connections. Only available where supported by the underlying openssl.

  ```
  my $rv = Net::SSLeay::CTX_tlsv1_1_new();
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

- CTX_tlsv1_2_new

  Creates a new SSL_CTX object − based on **TLSv1_2_method()** − as framework to establish TLS/SSL enabled connections. Only available where supported by the underlying openssl.

  ```
  my $rv = Net::SSLeay::CTX_tlsv1_2_new();
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

- CTX_new_with_method

  Creates a new SSL_CTX object based on `$meth` method

  ```
  my $rv = Net::SSLeay::CTX_new_with_method($meth);
  # $meth - value corresponding to openssl's SSL_METHOD structure
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)

  #example
  my $ctx = Net::SSLeay::CTX_new_with_method(&Net::SSLeay::TLSv1_method);
  ```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_new.html>

- CTX_set_min_proto_version, CTX_set_max_proto_version, set_min_proto_version and set_max_proto_version,

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0 or LibreSSL 2.6.0

  Set the minimum and maximum supported protocol for `$ctx` or `$ssl`.

  ```
  my $rv = Net::SSLeay::CTX_set_min_proto_version($ctx, $version)
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $version - (integer) constat version value or 0 for automatic lowest or highes
  #
  # returns: 1 on success, 0 on failure

  #example: allow only TLS 1.2 for a SSL_CTX
  my $rv_min = Net::SSLeay::CTX_set_min_proto_version($ctx, Net::SSLeay::TLS1_2_VE
  my $rv_max = Net::SSLeay::CTX_set_max_proto_version($ctx, Net::SSLeay::TLS1_2_VE

  #example: allow only TLS 1.1 for a SSL
  my $rv_min = Net::SSLeay::set_min_proto_version($ssl, Net::SSLeay::TLS1_1_VERSIC
  my $rv_max = Net::SSLeay::set_max_proto_version($ssl, Net::SSLeay::TLS1_1_VERSIC
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_min_proto_version.html>

- CTX_get_min_proto_version, CTX_get_max_proto_version, get_min_proto_version and get_max_proto_version,

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0g

  Get the minimum and maximum supported protocol for `$ctx` or `$ssl`.

  ```
  my $version = Net::SSLeay::CTX_get_min_proto_version($ctx)
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: 0 automatic lowest or highest value, configured value otherwise
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_min_proto_version.html>

- CTX_remove_session

  Removes the session `$ses` from the context `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_remove_session($ctx, $ses);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $ses - value corresponding to openssl's SSL_SESSION structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_add_session.html>

- CTX_sess_accept

  ```
  my $rv = Net::SSLeay::CTX_sess_accept($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: number of started SSL/TLS handshakes in server mode
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_accept_good

```
my $rv = Net::SSLeay::CTX_sess_accept_good($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of successfully established SSL/TLS sessions in server mode
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_accept_renegotiate

```
my $rv = Net::SSLeay::CTX_sess_accept_renegotiate($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of start renegotiations in server mode
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_cache_full

```
my $rv = Net::SSLeay::CTX_sess_cache_full($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of sessions that were removed because the maximum session cach
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_cb_hits

```
my $rv = Net::SSLeay::CTX_sess_cb_hits($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of successfully retrieved sessions from the external session c
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_connect

```
my $rv = Net::SSLeay::CTX_sess_connect($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of started SSL/TLS handshakes in client mode
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_connect_good

```
my $rv = Net::SSLeay::CTX_sess_connect_good($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of successfully established SSL/TLS sessions in client mode
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_connect_renegotiate

```
my $rv = Net::SSLeay::CTX_sess_connect_renegotiate($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of start renegotiations in client mode
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_get_cache_size

  Returns the currently valid session cache size.

```
my $rv = Net::SSLeay::CTX_sess_get_cache_size($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: current size
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_set_cache_size.html>

- CTX_sess_hits

```
my $rv = Net::SSLeay::CTX_sess_hits($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of successfully reused sessions
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_misses

```
my $rv = Net::SSLeay::CTX_sess_misses($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of sessions proposed by clients that were not found in the int
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_number

```
my $rv = Net::SSLeay::CTX_sess_number($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: current number of sessions in the internal session cache
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_set_cache_size

Sets the size of the internal session cache of context `$ctx` to `$size`.

```
Net::SSLeay::CTX_sess_set_cache_size($ctx, $size);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $size - cache size (0 = unlimited)
#
# returns: previously valid size
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_set_cache_size.html>

- CTX_sess_timeouts

Returns the number of sessions proposed by clients and either found in the internal or external session cache in server mode, but that were invalid due to timeout. These sessions are not included in the SSL_CTX_sess_hits count.

```
my $rv = Net::SSLeay::CTX_sess_timeouts($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: number of sessions
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sess_number.html>

- CTX_sess_set_new_cb

COMPATIBILITY: not available in Net−SSLeay−1.85 and before

Sets the callback function, which is automatically called whenever a new session was negotiated.

```
Net::SSLeay::CTX_sess_set_new_cb($ctx, $func);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $func - perl reference to callback function
#
# returns: no return value
```

Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_sess_set_new_cb.html>

- CTX_sess_set_remove_cb

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before

  Sets the callback function, which is automatically called whenever a session is removed by the SSL engine.

```
Net::SSLeay::CTX_sess_set_remove_cb($ctx, $func);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $func - perl reference to callback function
#
# returns: no return value
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_sess_set_remove_cb.html>

- CTX_sessions

  Returns a pointer to the lhash databases containing the internal session cache for ctx.

```
my $rv = Net::SSLeay::CTX_sessions($ctx);
# $ctx - value corresponding to openssl's SSL_CTX structure
#
# returns: value corresponding to openssl's LHASH structure (0 on failure)
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_sessions.html>

- CTX_set1_param

  Applies X509 verification parameters `$vpm` on `$ctx`

```
my $rv = Net::SSLeay::CTX_set1_param($ctx, $vpm);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $vpm - value corresponding to openssl's X509_VERIFY_PARAM structure
#
# returns: 1 on success, 0 on failure
```

  Check openssl doc <https://www.openssl.org/docs/ssl/SSL_CTX_get0_param.html>

- CTX_set_cert_store

  Sets/replaces the certificate verification storage of `$ctx` to/with `$store`.

```
Net::SSLeay::CTX_set_cert_store($ctx, $store);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $store - value corresponding to openssl's X509_STORE structure
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_cert_store.html>

- CTX_set_cert_verify_callback

  Sets the verification callback function for `$ctx`. SSL objects that are created from `$ctx` inherit the setting valid at the time when `Net::SSLeay::new($ctx)` is called.

```
Net::SSLeay::CTX_set_cert_verify_callback($ctx, $func, $data);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $func - perl reference to callback function
# $data - [optional] data that will be passed to callback function when invoked
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_cert_verify_callback.html>

- CTX_set_cipher_list

  Sets the list of available ciphers for `$ctx` using the control string `$str`. The list of ciphers is inherited by all ssl objects created from `$ctx`.

```
my $rv = Net::SSLeay::CTX_set_cipher_list($s, $str);
# $s - value corresponding to openssl's SSL_CTX structure
# $str - (string) cipher list e.g. '3DES:+RSA'
#
# returns: 1 if any cipher could be selected and 0 on complete failure
```

  The format of `$str` is described in <http://www.openssl.org/docs/apps/ciphers.html>

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_cipher_list.html>

- CTX_set_ciphersuites

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Configure the available TLSv1.3 ciphersuites.

```
my $rv = Net::SSLeay::CTX_set_ciphersuites($ctx, $str);
# $ctx  - value corresponding to openssl's SSL_CTX structure
# $str  - colon (":") separated list of TLSv1.3 ciphersuite names in order of pr
#
# returns: (integer) 1 if the requested ciphersuite list was configured, and 0 c
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_ciphersuites.html>

- CTX_set_client_CA_list

  Sets the list of CAs sent to the client when requesting a client certificate for `$ctx`.

```
Net::SSLeay::CTX_set_client_CA_list($ctx, $list);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $list - value corresponding to openssl's X509_NAME_STACK structure
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_client_CA_list.html>

- CTX_set_default_passwd_cb

  Sets the default password callback called when loading/storing a PEM certificate with encryption.

```
Net::SSLeay::CTX_set_default_passwd_cb($ctx, $func);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $func - perl reference to callback function
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_default_passwd_cb.html>

- CTX_set_default_passwd_cb_userdata

  Sets a pointer to userdata which will be provided to the password callback on invocation.

  ```
  Net::SSLeay::CTX_set_default_passwd_cb_userdata($ctx, $userdata);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $userdata - data that will be passed to callback function when invoked
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_default_passwd_cb.html>

- CTX_set_default_verify_paths

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::CTX_set_default_verify_paths($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: 1 on success, 0 on failure
  ```

- CTX_set_ex_data

  Is used to store application data at `$data` for `$idx` into the `$ctx` object.

  ```
  my $rv = Net::SSLeay::CTX_set_ex_data($ssl, $idx, $data);
  # $ssl - value corresponding to openssl's SSL_CTX structure
  # $idx - (integer) ???
  # $data - (pointer) ???
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_get_ex_new_index.html>

- CTX_set_purpose

  ```
  my $rv = Net::SSLeay::CTX_set_purpose($s, $purpose);
  # $s - value corresponding to openssl's SSL_CTX structure
  # $purpose - (integer) purpose identifier
  #
  # returns: 1 on success, 0 on failure

  #avainable purpose identifier
  1 - X509_PURPOSE_SSL_CLIENT
  2 - X509_PURPOSE_SSL_SERVER
  3 - X509_PURPOSE_NS_SSL_SERVER
  4 - X509_PURPOSE_SMIME_SIGN
  5 - X509_PURPOSE_SMIME_ENCRYPT
  6 - X509_PURPOSE_CRL_SIGN
  7 - X509_PURPOSE_ANY
  8 - X509_PURPOSE_OCSP_HELPER
  9 - X509_PURPOSE_TIMESTAMP_SIGN

  #or use corresponding constants
  $purpose = &Net::SSLeay::X509_PURPOSE_SSL_CLIENT;
  ...
  $purpose = &Net::SSLeay::X509_PURPOSE_TIMESTAMP_SIGN;
  ```

- CTX_set_quiet_shutdown

  Sets the 'quiet shutdown' flag for `$ctx` to be mode. SSL objects created from `$ctx` inherit the mode valid at the time `Net::SSLeay::new($ctx)` is called.

```
Net::SSLeay::CTX_set_quiet_shutdown($ctx, $mode);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $mode - 0 or 1
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_quiet_shutdown.html>

- CTX_set_read_ahead

```
my $rv = Net::SSLeay::CTX_set_read_ahead($ctx, $val);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $val - read_ahead value to be set
#
# returns: the original read_ahead value
```

- CTX_set_session_id_context

  Sets the context `$sid_ctx` of length `$sid_ctx_len` within which a session can be reused for the `$ctx` object.

```
my $rv = Net::SSLeay::CTX_set_session_id_context($ctx, $sid_ctx, $sid_ctx_len);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $sid_ctx - data buffer
# $sid_ctx_len - length of data in $sid_ctx
#
# returns: 1 on success, 0 on failure (the error is logged to the error stack)
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_session_id_context.html>

- CTX_set_ssl_version

  Sets a new default TLS/SSL method for SSL objects newly created from this `$ctx`. SSL objects already created with `Net::SSLeay::new($ctx)` are not affected, except when `Net::SSLeay:clear($ssl)` is being called.

```
my $rv = Net::SSLeay::CTX_set_ssl_version($ctx, $meth);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $meth - value corresponding to openssl's SSL_METHOD structure
#
# returns: 1 on success, 0 on failure
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_ssl_version.html>

- CTX_set_timeout

  Sets the timeout for newly created sessions for `$ctx` to `$t`. The timeout value `$t` must be given in seconds.

```
my $rv = Net::SSLeay::CTX_set_timeout($ctx, $t);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $t - timeout in seconds
#
# returns: previously set timeout value
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_timeout.html>

- CTX_set_tmp_dh

  Sets DH parameters to be used to be `$dh`. The key is inherited by all ssl objects created from `$ctx`.

```
my $rv = Net::SSLeay::CTX_set_tmp_dh($ctx, $dh);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $dh - value corresponding to openssl's DH structure
#
# returns: 1 on success, 0 on failure
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_dh_callback.html>

- CTX_set_tmp_dh_callback

  Sets the callback function for `$ctx` to be used when a DH parameters are required to `$tmp_dh_callback`.

```
Net::SSLeay::CTX_set_tmp_dh_callback($ctx, $tmp_dh_callback);
# $ctx - value corresponding to openssl's SSL_CTX structure
# tmp_dh_callback - (function pointer) ???
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_dh_callback.html>

- CTX_set_tmp_rsa

  Sets the temporary/ephemeral RSA key to be used to be `$rsa`.

```
my $rv = Net::SSLeay::CTX_set_tmp_rsa($ctx, $rsa);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $rsa - value corresponding to openssl's RSA structure
#
# returns: 1 on success, 0 on failure
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_rsa_callback.html>

  Not available with OpenSSL 1.1 and later.

- CTX_set_tmp_rsa_callback

  Sets the callback function for ctx to be used when a temporary/ephemeral RSA key is required to `$tmp_rsa_callback`.

  ??? (does this function really work?)

```
Net::SSLeay::CTX_set_tmp_rsa_callback($ctx, $tmp_rsa_callback);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $tmp_rsa_callback - (function pointer) ???
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_rsa_callback.html>

  Not available with OpenSSL 1.1 and later.

- CTX_set_trust

```
my $rv = Net::SSLeay::CTX_set_trust($s, $trust);
# $s - value corresponding to openssl's SSL_CTX structure
# $trust - (integer) trust identifier
#
# returns: the original value

#available trust identifiers
1 - X509_TRUST_COMPAT
2 - X509_TRUST_SSL_CLIENT
3 - X509_TRUST_SSL_SERVER
```

```
4 - X509_TRUST_EMAIL
5 - X509_TRUST_OBJECT_SIGN
6 - X509_TRUST_OCSP_SIGN
7 - X509_TRUST_OCSP_REQUEST
8 - X509_TRUST_TSA

#or use corresponding constants
$trust = &Net::SSLeay::X509_TRUST_COMPAT;
...
$trust = &Net::SSLeay::X509_TRUST_TSA;
```

- CTX_set_verify_depth

  Sets the maximum depth for the certificate chain verification that shall be allowed for ctx.

  ```
  Net::SSLeay::CTX_set_verify_depth($ctx, $depth);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $depth - max. depth
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_verify.html>

- CTX_use_PKCS12_file

  Adds the certificate and private key from PKCS12 file `$p12filename` to `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_use_PKCS12_file($ctx, $p12filename, $password);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $p12filename - (string) filename
  # $password - (string) password to decrypt private key
  #
  # returns: 1 on success, 0 on failure
  ```

- CTX_use_PrivateKey

  Adds the private key `$pkey` to `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_use_PrivateKey($ctx, $pkey);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_use_PrivateKey_file

  Adds the first private key found in `$file` to `$ctx`.

  ```
  my $rv = Net::SSLeay::CTX_use_PrivateKey_file($ctx, $file, $type);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $file - (string) file name
  # $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSL
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_use_RSAPrivateKey

  Adds the RSA private key `$rsa` to `$ctx`.

```
my $rv = Net::SSLeay::CTX_use_RSAPrivateKey($ctx, $rsa);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $rsa - value corresponding to openssl's RSA structure
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_use_RSAPrivateKey_file

  Adds the first RSA private key found in `$file` to `$ctx`.

```
my $rv = Net::SSLeay::CTX_use_RSAPrivateKey_file($ctx, $file, $type);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $file - (string) file name
# $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSL
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

- CTX_use_certificate

  Loads the certificate `$x` into `$ctx`

```
my $rv = Net::SSLeay::CTX_use_certificate($ctx, $x);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $x - value corresponding to openssl's X509 structure
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_use_certificate_chain_file

  Loads a certificate chain from `$file` into `$ctx`. The certificates must be in PEM format and must be sorted starting with the subject's certificate (actual client or server certificate), followed by intermediate CA certificates if applicable, and ending at the highest level (root) CA.

```
my $rv = Net::SSLeay::CTX_use_certificate_chain_file($ctx, $file);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $file - (string) file name
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_use_certificate_file

  Loads the first certificate stored in `$file` into `$ctx`.

```
my $rv = Net::SSLeay::CTX_use_certificate_file($ctx, $file, $type);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $file - (string) file name
# $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSL
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- CTX_get_security_level

  COMPATIBILITY: not available in Net-SSLeay-1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

  Returns the security level associated with `$ctx`.

```
my $level = Net::SSLeay::CTX_get_security_level($ctx);
# $ctx   - value corresponding to openssl's SSL_CTX structure
#
# returns: (integer) current security level
```

Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_get_security_level.html>

- CTX_set_security_level

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

  Sets the security level associated with `$ctx` to `$level`.

  ```
  Net::SSLeay::CTX_set_security_level($ctx, $level);
  # $ssl   - value corresponding to openssl's SSL_CTX structure
  # $level - new security level
  #
  # returns: no return value
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_security_level.html>

- CTX_set_num_tickets

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Set number of TLSv1.3 session tickets that will be sent to a client.

  ```
  my $rv = Net::SSLeay::CTX_set_num_tickets($ctx, $number_of_tickets);
  # $ctx  - value corresponding to openssl's SSL_CTX structure
  # $number_of_tickets - number of tickets to send
  #
  # returns: 1 on success, 0 on failure
  ```

  Set to zero if you do not no want to support a session resumption.

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set_num_tickets.html>

- CTX_get_num_tickets

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Get number of TLSv1.3 session tickets that will be sent to a client.

  ```
  my $number_of_tickets = Net::SSLeay::CTX_get_num_tickets($ctx);
  # $ctx  - value corresponding to openssl's SSL_CTX structure
  #
  # returns: (integer) number of tickets to send
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_CTX_get_num_tickets.html>

Low level API: SSL_* related functions

NOTE: Please note that the function described in this chapter have "SSL_" part stripped from their original openssl names.

- new

  Creates a new SSL structure which is needed to hold the data for a TLS/SSL connection. The new structure inherits the settings of the underlying context `$ctx:` connection method

(SSLv2/v3/TLSv1), options, verification settings, timeout settings.

```
 my $rv = Net::SSLeay::new($ctx);
 # $ctx - value corresponding to openssl's SSL_CTX structure
 #
 # returns: value corresponding to openssl's SSL structure (0 on failure)
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_new.html>

- accept

  Waits for a TLS/SSL client to initiate the TLS/SSL handshake. The communication channel must already have been set and assigned to the ssl by setting an underlying BIO.

```
 my $rv = Net::SSLeay::accept($ssl);
 # $ssl - value corresponding to openssl's SSL structure
 #
 # returns: 1 = success, 0 = handshake not successful, <0 = fatal error during ha
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_accept.html>

- add_client_CA

  Adds the CA name extracted from cacert to the list of CAs sent to the client when requesting a client certificate for the chosen ssl, overriding the setting valid for ssl's SSL_CTX object.

```
 my $rv = Net::SSLeay::add_client_CA($ssl, $x);
 # $ssl - value corresponding to openssl's SSL structure
 # $x - value corresponding to openssl's X509 structure
 #
 # returns: 1 on success, 0 on failure
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_client_CA_list.html>

- callback_ctrl

  ??? (more info needed)

```
 my $rv = Net::SSLeay::callback_ctrl($ssl, $cmd, $fp);
 # $ssl - value corresponding to openssl's SSL structure
 # $cmd - (integer) command id
 # $fp - (function pointer) ???
 #
 # returns: ???
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_ctrl.html>

- check_private_key

  Checks the consistency of a private key with the corresponding certificate loaded into `$ssl`

```
 my $rv = Net::SSLeay::check_private_key($ssl);
 # $ssl - value corresponding to openssl's SSL structure
 #
 # returns: 1 on success, otherwise check out the error stack to find out the rea
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- clear

  Reset SSL object to allow another connection.

```
Net::SSLeay::clear($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_clear.html>

- connect

  Initiate the TLS/SSL handshake with an TLS/SSL server.

  ```
  my $rv = Net::SSLeay::connect($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 1 = success, 0 = handshake not successful, <0 = fatal error during ha
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_connect.html>

- copy_session_id

  Copies the session structure fro `$from` to `$to` (+ also the private key and certificate associated with `$from`).

  ```
  Net::SSLeay::copy_session_id($to, $from);
  # $to - value corresponding to openssl's SSL structure
  # $from - value corresponding to openssl's SSL structure
  #
  # returns: no return value
  ```

- ctrl

  Internal handling function for SSL objects.

  BEWARE: openssl doc says: This function should never be called directly!

  ```
  my $rv = Net::SSLeay::ctrl($ssl, $cmd, $larg, $parg);
  # $ssl - value corresponding to openssl's SSL structure
  # $cmd - (integer) command id
  # $larg - (integer) long ???
  # $parg - (string/pointer) ???
  #
  # returns: (long) result of given command ???
  ```

  For more details about valid `$cmd` values check "CTX_ctrl".

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_ctrl.html>

- do_handshake

  Will wait for a SSL/TLS handshake to take place. If the connection is in client mode, the handshake will be started. The handshake routines may have to be explicitly set in advance using either SSL_set_connect_state or **SSL_set_accept_state** (3).

  ```
  my $rv = Net::SSLeay::do_handshake($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 1 = success, 0 = handshake not successful, <0 = fatal error during ha
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_do_handshake.html>

- dup

  Returns a duplicate of `$ssl`.

```
my $rv = Net::SSLeay::dup($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: value corresponding to openssl's SSL structure (0 on failure)
```

- free

  Free an allocated SSL structure.

  ```
  Net::SSLeay::free($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_free.html>

- get0_param

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Returns the current verification parameters.

  ```
  my $vpm = Net::SSLeay::get0_param($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's X509_VERIFY_PARAM structure
  ```

  Check openssl doc <https://www.openssl.org/docs/ssl/SSL_CTX_get0_param.html>

- get_SSL_CTX

  Returns a pointer to the SSL_CTX object, from which `$ssl` was created with Net::SSLeay::new.

  ```
  my $rv = Net::SSLeay::get_SSL_CTX($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's SSL_CTX structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_SSL_CTX.html>

- set_SSL_CTX

  Sets the SSL_CTX the corresponds to an SSL session.

  ```
  my $the_ssl_ctx = Net::SSLeay::set_SSL_CTX($ssl, $ssl_ctx);
  # $ssl - value corresponding to openssl's SSL structure
  # $ssl_ctx - Change the ssl object to the given ssl_ctx
  #
  # returns - the ssl_ctx
  ```

- get_app_data

  Can be used to get application defined value/data.

  ```
  my $rv = Net::SSLeay::get_app_data($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: string/buffer/pointer ???
  ```

- set_app_data

  Can be used to set some application defined value/data.

```
my $rv = Net::SSLeay::set_app_data($ssl, $arg);
# $ssl - value corresponding to openssl's SSL structure
# $arg - (string/buffer/pointer ???) data
#
# returns: ???
```

- get_certificate

  Gets X509 certificate from an established SSL connection.

```
my $rv = Net::SSLeay::get_certificate($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: value corresponding to openssl's X509 structure (0 on failure)
```

- get_cipher

  Obtains the name of the currently used cipher.

```
my $rv = Net::SSLeay::get_cipher($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (string) cipher name e.g. 'DHE-RSA-AES256-SHA' or '', when no sessior
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_current_cipher.html>

- get_cipher_bits

  Obtain the number of secret/algorithm bits used.

```
my $rv = Net::SSLeay::get_cipher_bits($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: number of secret bits used by current cipher
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_current_cipher.html> and <http://www.openssl.org/docs/ssl/SSL_CIPHER_get_name.html>

- get_cipher_list

  Returns the name (string) of the SSL_CIPHER listed for `$ssl` with priority `$n`.

```
my $rv = Net::SSLeay::get_cipher_list($ssl, $n);
# $ssl - value corresponding to openssl's SSL structure
# $n - (integer) priority
#
# returns: (string) cipher name e.g. 'EDH-DSS-DES-CBC3-SHA' or '' in case of err
```

  Call Net::SSLeay::get_cipher_list with priority starting from 0 to obtain the sorted list of available ciphers, until '' is returned:

```
my $priority = 0;
while (my $c = Net::SSLeay::get_cipher_list($ssl, $priority)) {
  print "cipher[$priority] = $c\n";
  $priority++;
}
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_ciphers.html>

- get_client_CA_list

  Returns the list of client CAs explicitly set for `$ssl` using `Net::SSleay::set_client_CA_list` or `$ssl`'s SSL_CTX object with `Net::SSLeay::CTX_set_client_CA_list`, when in server mode.

In client mode, returns the list of client CAs sent from the server, if any.

```
my $rv = Net::SSLeay::get_client_CA_list($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: value corresponding to openssl's STACK_OF(X509_NAME) structure (0 on
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_client_CA_list.html>

- get_current_cipher

  Returns the cipher actually used.

  ```
  my $rv = Net::SSLeay::get_current_cipher($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's SSL_CIPHER structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_current_cipher.html>

- get_default_timeout

  Returns the default timeout value assigned to SSL_SESSION objects negotiated for the protocol valid for `$ssl`.

  ```
  my $rv = Net::SSLeay::get_default_timeout($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: (long) timeout in seconds
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_default_timeout.html>

- get_error

  Returns a result code for a preceding call to `connect`, `accept`, `do_handshake`, `read`, `peek` or `write` on `$ssl`.

  ```
  my $rv = Net::SSLeay::get_error($ssl, $ret);
  # $ssl - value corresponding to openssl's SSL structure
  # $ret - return value of preceding TLS/SSL I/O operation
  #
  # returns: result code, which is one of the following values:
  #  0 - SSL_ERROR_NONE
  #  1 - SSL_ERROR_SSL
  #  2 - SSL_ERROR_WANT_READ
  #  3 - SSL_ERROR_WANT_WRITE
  #  4 - SSL_ERROR_WANT_X509_LOOKUP
  #  5 - SSL_ERROR_SYSCALL
  #  6 - SSL_ERROR_ZERO_RETURN
  #  7 - SSL_ERROR_WANT_CONNECT
  #  8 - SSL_ERROR_WANT_ACCEPT
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_error.html>

- get_ex_data

  Is used to retrieve the information for `$idx` from `$ssl`.

```
my $rv = Net::SSLeay::get_ex_data($ssl, $idx);
# $ssl - value corresponding to openssl's SSL structure
# $idx - (integer) index for application specific data
#
# returns: pointer to ???
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_ex_new_index.html>

- set_ex_data

  Is used to store application data at `$data` for `$idx` into the `$ssl` object.

  ```
  my $rv = Net::SSLeay::set_ex_data($ssl, $idx, $data);
  # $ssl - value corresponding to openssl's SSL structure
  # $idx - (integer) ???
  # $data - (pointer) ???
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_ex_new_index.html>

- get_ex_new_index

  Is used to register a new index for application specific data.

  ```
  my $rv = Net::SSLeay::get_ex_new_index($argl, $argp, $new_func, $dup_func, $free
  # $argl - (long) ???
  # $argp - (pointer) ???
  # $new_func - function pointer ??? (CRYPTO_EX_new *)
  # $dup_func - function pointer ??? (CRYPTO_EX_dup *)
  # $free_func - function pointer ??? (CRYPTO_EX_free *)
  #
  # returns: (integer) ???
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_ex_new_index.html>

- get_fd

  Returns the file descriptor which is linked to `$ssl`.

  ```
  my $rv = Net::SSLeay::get_fd($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: file descriptor (>=0) or -1 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_fd.html>

- get_finished

  Obtains the latest 'Finished' message sent to the peer. Return value is zero if there's been no Finished message yet. Default count is 2*EVP_MAX_MD_SIZE that is long enough for all possible Finish messages. If you supply a non-default count, the resulting return value may be longer than returned buf's length.

  ```
  my $rv = Net::SSLeay::get_finished($ssl, $buf, $count);
  # $ssl - value corresponding to openssl's SSL structure
  # $buf - buffer where the returned data will be stored
  # $count - [optional] max size of return data - default is 2*EVP_MAX_MD_SIZE
  #
  # returns: length of latest Finished message
  ```

- get_peer_finished

  Obtains the latest 'Finished' message expected from the peer. Parameters and return value are similar to **get_finished**().

```
my $rv = Net::SSLeay::get_peer_finished($ssl, $buf, $count);
# $ssl - value corresponding to openssl's SSL structure
# $buf - buffer where the returned data will be stored
# $count - [optional] max size of return data - default is 2*EVP_MAX_MD_SIZE
#
# returns: length of latest Finished message
```

- get_keyblock_size

  Gets the length of the TLS keyblock.

  NOTE: Does not exactly correspond to any low level API function.

```
my $rv = Net::SSLeay::get_keyblock_size($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: keyblock size, -1 on error
```

- get_mode

  Returns the mode (bitmask) set for `$ssl`.

```
my $rv = Net::SSLeay::get_mode($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: mode (bitmask)
```

  To decode the return value (bitmask) see documentation for "CTX_get_mode".

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_mode.html>

- set_mode

  Adds the mode set via bitmask in `$mode` to `$ssl`. Options already set before are not cleared.

```
my $rv = Net::SSLeay::set_mode($ssl, $mode);
# $ssl - value corresponding to openssl's SSL structure
# $mode - mode (bitmask)
#
# returns: the new mode bitmask after adding $mode
```

  For `$mode` bitmask details see "CTX_get_mode".

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_mode.html>

- get_options

  Returns the options (bitmask) set for `$ssl`.

```
my $rv = Net::SSLeay::get_options($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: options (bitmask)
```

  To decode the return value (bitmask) see documentation for "CTX_get_options".

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_options.html>

- set_options

  Adds the options set via bitmask in `$options` to `$ssl`. Options already set before are not cleared!

```
Net::SSLeay::set_options($ssl, $options);
# $ssl - value corresponding to openssl's SSL structure
# $options - options (bitmask)
#
# returns: the new options bitmask after adding $options
```

For `$options` bitmask details see "CTX_get_options".

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_options.html>

- get_peer_certificate

  Get the X509 certificate of the peer.

  ```
  my $rv = Net::SSLeay::get_peer_certificate($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's X509 structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_peer_certificate.html>

- get_peer_cert_chain

  Get the certificate chain of the peer as an array of X509 structures.

  ```
  my @rv = Net::SSLeay::get_peer_cert_chain($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: list of X509 structures
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_peer_certificate.html>

- get_quiet_shutdown

  Returns the 'quiet shutdown' setting of ssl.

  ```
  my $rv = Net::SSLeay::get_quiet_shutdown($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: (integer) current 'quiet shutdown' value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_quiet_shutdown.html>

- get_rbio

  Get 'read' BIO linked to an SSL object `$ssl`.

  ```
  my $rv = Net::SSLeay::get_rbio($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's BIO structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_rbio.html>

- get_read_ahead

  ```
  my $rv = Net::SSLeay::get_read_ahead($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: (integer) read_ahead value
  ```

- set_read_ahead

```
Net::SSLeay::set_read_ahead($ssl, $val);
# $ssl - value corresponding to openssl's SSL structure
# $val - read_ahead value to be set
#
# returns: the original read_ahead value
```

- get_security_level

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

  Returns the security level associated with `$ssl`.

  ```
  my $level = Net::SSLeay::get_security_level($ssl);
  # $ssl   - value corresponding to openssl's SSL structure
  #
  # returns: (integer) current security level
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_get_security_level.html>

- set_security_level

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

  Sets the security level associated with `$ssl` to `$level`.

  ```
  Net::SSLeay::set_security_level($ssl, $level);
  # $ssl   - value corresponding to openssl's SSL structure
  # $level - new security level
  #
  # returns: no return value
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_set_security_level.html>

- set_num_tickets

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Set number of TLSv1.3 session tickets that will be sent to a client.

  ```
  my $rv = Net::SSLeay::set_num_tickets($ssl, $number_of_tickets);
  # $ssl  - value corresponding to openssl's SSL structure
  # $number_of_tickets - number of tickets to send
  #
  # returns: 1 on success, 0 on failure
  ```

  Set to zero if you do not no want to support a session resumption.

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_set_num_tickets.html>

- get_num_tickets

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Get number of TLSv1.3 session tickets that will be sent to a client.

  ```
  my $number_of_tickets = Net::SSLeay::get_num_tickets($ctx);
  # $ctx  - value corresponding to openssl's SSL structure
  #
  # returns: number of tickets to send
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_get_num_tickets.html>

- get_server_random

  Returns internal SSLv3 server_random value.

  ```
  Net::SSLeay::get_server_random($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: server_random value (binary data)
  ```

- get_client_random

  NOTE: Does not exactly correspond to any low level API function

  Returns internal SSLv3 client_random value.

  ```
  Net::SSLeay::get_client_random($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: client_random value (binary data)
  ```

- export_keying_material

  Returns keying material based on the string `$label` and optional `$context`. Note that with TLSv1.2 and lower, empty context (empty string) and undefined context (no value or 'undef') will return different values.

  ```
   my $out = Net::SSLeay::export_keying_material($ssl, $olen, $label, $context);
   # $ssl - value corresponding to openssl's SSL structure
   # $olen - number of bytes to return
   # $label - application specific label
   # $context - [optional] context - default is undef for no context
   #
   # returns: keying material (binary data) or undef on error
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_export_keying_material.html>

- get_session

  Retrieve TLS/SSL session data used in `$ssl`. The reference count of the SSL_SESSION is NOT incremented.

  ```
  my $rv = Net::SSLeay::get_session($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's SSL_SESSION structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_session.html>

- SSL_get0_session

  The alias for "get_session" (note that the name is `SSL_get0_session` NOT `get0_session`).

  ```
  my $rv = Net::SSLeay::SSL_get0_session();
  ```

- get1_session

  Returns a pointer to the SSL_SESSION actually used in `$ssl`. The reference count of the SSL_SESSION is incremented by 1.

  ```
  my $rv = Net::SSLeay::get1_session($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's SSL_SESSION structure (0 on failure)
  ```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_session.html>

- get_shared_ciphers

  Returns string with a list (colon ':' separated) of ciphers shared between client and server within SSL session `$ssl`.

  ```
  my $rv = Net::SSLeay::get_shared_ciphers()
  #
  # returns: string like 'ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES2
  ```

- get_shutdown

  Returns the shutdown mode of `$ssl`.

  ```
  my $rv = Net::SSLeay::get_shutdown($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: shutdown mode (bitmask) of ssl

  #to decode the return value (bitmask) use:
  0 - No shutdown setting, yet
  1 - SSL_SENT_SHUTDOWN
  2 - SSL_RECEIVED_SHUTDOWN
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_shutdown.html>

- get_ssl_method

  Returns a function pointer to the TLS/SSL method set in `$ssl`.

  ```
  my $rv = Net::SSLeay::get_ssl_method($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: value corresponding to openssl's SSL_METHOD structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_ssl_version.html>

- in_init, in_before, is_init_finished, in_connect_init, in_accept_init

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before.

  Retrieve information about the handshake state machine. All functions take `$ssl` as the only argument and return 0 or 1. These functions are recommended over **get_state()** and **state()**.

  ```
  my $rv = Net::SSLeay::is_init_finished($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: All functions return 1 or 0
  ```

  Check openssl doc https://www.openssl.org/docs/ssl/SSL_in_init.html <http://www.openssl.org/docs/ssl/SSL_in_init.html>

- get_state

  COMPATIBILITY: OpenSSL 1.1.0 and later use different constants which are not made available. Use **is_init_finished()** and related functions instead.

  Returns the SSL connection state.

```
my $rv = Net::SSLeay::get_state($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (integer) state value
#          to decode the returned state check:
#          SSL_ST_* constants in openssl/ssl.h
#          SSL2_ST_* constants in openssl/ssl2.h
#          SSL23_ST_* constants in openssl/ssl23.h
#          SSL3_ST_* + DTLS1_ST_* constants in openssl/ssl3.h
```

- state

  Exactly the same as "get_state".

  ```
  my $rv = Net::SSLeay::state($ssl);
  ```

- set_state

  Sets the SSL connection state.

  ```
  Net::SSLeay::set_state($ssl,Net::SSLeay::SSL_ST_ACCEPT());
  ```

  Not available with OpenSSL 1.1 and later.

- get_verify_depth

  Returns the verification depth limit currently set in `$ssl`.

  ```
  my $rv = Net::SSLeay::get_verify_depth($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: current depth or -1 if no limit has been explicitly set
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_get_verify_mode.html>

- set_verify_depth

  Sets the maximum depth for the certificate chain verification that shall be allowed for `$ssl`.

  ```
  Net::SSLeay::set_verify_depth($ssl, $depth);
  # $ssl - value corresponding to openssl's SSL structure
  # $depth - (integer) depth
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_verify.html>

- get_verify_mode

  Returns the verification mode (bitmask) currently set in `$ssl`.

  ```
  my $rv = Net::SSLeay::get_verify_mode($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: mode (bitmask)
  ```

  To decode the return value (bitmask) see documentation for "CTX_get_verify_mode".

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_get_verify_mode.html>

- set_verify

  Sets the verification flags for `$ssl` to be `$mode` and specifies the `$verify_callback` function to be used.

```
Net::SSLeay::set_verify($ssl, $mode, $callback);
# $ssl - value corresponding to openssl's SSL structure
# $mode - mode (bitmask)
# $callback - [optional] reference to perl callback function
#
# returns: no return value
```

For `$mode` bitmask details see "CTX_get_verify_mode".

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_verify.html>

- set_post_handshake_auth

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Enable the Post-Handshake Authentication extension to be added to the ClientHello such that post-handshake authentication can be requested by the server.

```
Net::SSLeay::set_posthandshake_auth($ssl, $val);
# $ssl - value corresponding to openssl's SSL structure
# $val - 0 then the extension is not sent, otherwise it is
#
# returns: no return value
```

  Check openssl doc https://www.openssl.org/docs/manmaster/man3/SSL_set_post_handshake_auth <https://www.openssl.org/docs/manmaster/man3/SSL_set_post_handshake_auth.html>

- verify_client_post_handshake

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  verify_client_post_handshake causes a CertificateRequest message to be sent by a server on the given ssl connection.

```
my $rv = Net::SSLeay::verify_client_post_handshake($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: 1 if the request succeeded, and 0 if the request failed. The error st
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_verify_client_post_handshake.html>

- get_verify_result

  Returns the result of the verification of the X509 certificate presented by the peer, if any.

```
my $rv = Net::SSLeay::get_verify_result($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (integer)
#      0 - X509_V_OK: ok
#      2 - X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT: unable to get issuer certificat
#      3 - X509_V_ERR_UNABLE_TO_GET_CRL: unable to get certificate CRL
#      4 - X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE: unable to decrypt certif
#      5 - X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE: unable to decrypt CRL's s
#      6 - X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY: unable to decode issue
#      7 - X509_V_ERR_CERT_SIGNATURE_FAILURE: certificate signature failure
#      8 - X509_V_ERR_CRL_SIGNATURE_FAILURE: CRL signature failure
#      9 - X509_V_ERR_CERT_NOT_YET_VALID: certificate is not yet valid
#     10 - X509_V_ERR_CERT_HAS_EXPIRED: certificate has expired
```

```
 #     11 - X509_V_ERR_CRL_NOT_YET_VALID: CRL is not yet valid
 #     12 - X509_V_ERR_CRL_HAS_EXPIRED: CRL has expired
 #     13 - X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD: format error in certificat
 #     14 - X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD: format error in certificate
 #     15 - X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD: format error in CRL's last
 #     16 - X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD: format error in CRL's next
 #     17 - X509_V_ERR_OUT_OF_MEM: out of memory
 #     18 - X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT: self signed certificate
 #     19 - X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN: self signed certificate in cert
 #     20 - X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY: unable to get local iss
 #     21 - X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE: unable to verify the firs
 #     22 - X509_V_ERR_CERT_CHAIN_TOO_LONG: certificate chain too long
 #     23 - X509_V_ERR_CERT_REVOKED: certificate revoked
 #     24 - X509_V_ERR_INVALID_CA: invalid CA certificate
 #     25 - X509_V_ERR_PATH_LENGTH_EXCEEDED: path length constraint exceeded
 #     26 - X509_V_ERR_INVALID_PURPOSE: unsupported certificate purpose
 #     27 - X509_V_ERR_CERT_UNTRUSTED: certificate not trusted
 #     28 - X509_V_ERR_CERT_REJECTED: certificate rejected
 #     29 - X509_V_ERR_SUBJECT_ISSUER_MISMATCH: subject issuer mismatch
 #     30 - X509_V_ERR_AKID_SKID_MISMATCH: authority and subject key identifier m
 #     31 - X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH: authority and issuer serial n
 #     32 - X509_V_ERR_KEYUSAGE_NO_CERTSIGN:key usage does not include certificat
 #     50 - X509_V_ERR_APPLICATION_VERIFICATION: application verification failure
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_verify_result.html>

- set_verify_result

  Override result of peer certificate verification.

```
 Net::SSLeay::set_verify_result($ssl, $v);
 # $ssl - value corresponding to openssl's SSL structure
 # $v - (integer) result value
 #
 # returns: no return value
```

  For more info about valid return values see "get_verify_result"

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_verify_result.html>

- get_wbio

  Get 'write' BIO linked to an SSL object `$ssl`.

```
 my $rv = Net::SSLeay::get_wbio($ssl);
 # $ssl - value corresponding to openssl's SSL structure
 #
 # returns: value corresponding to openssl's BIO structure (0 on failure)
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_get_rbio.html>

- load_client_CA_file

  Load X509 certificates from file (PEM formatted).

```
 my $rv = Net::SSLeay::load_client_CA_file($file);
 # $file - (string) file name
 #
 # returns: value corresponding to openssl's STACK_OF(X509_NAME) structure (0 on
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_load_client_CA_file.html>

- clear_num_renegotiations

  Executes SSL_CTRL_CLEAR_NUM_RENEGOTIATIONS command on `$ssl`.

  ```
  my $rv = Net::SSLeay::clear_num_renegotiations($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: command result
  ```

- need_tmp_RSA

  Executes SSL_CTRL_NEED_TMP_RSA command on `$ssl`.

  ```
  my $rv = Net::SSLeay::need_tmp_RSA($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: command result
  ```

  Not available with OpenSSL 1.1 and later.

- num_renegotiations

  Executes SSL_CTRL_GET_NUM_RENEGOTIATIONS command on `$ssl`.

  ```
  my $rv = Net::SSLeay::num_renegotiations($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: command result
  ```

- total_renegotiations

  Executes SSL_CTRL_GET_TOTAL_RENEGOTIATIONS command on `$ssl`.

  ```
  my $rv = Net::SSLeay::total_renegotiations($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: command result
  ```

- peek

  Copies `$max` bytes from the specified `$ssl` into the returned value. In contrast to the `Net::SSLeay::read()` function, the data in the SSL buffer is unmodified after the **SSL_peek()** operation.

  ```
  Net::SSLeay::peek($ssl, $max);
  # $ssl - value corresponding to openssl's SSL structure
  # $max - [optional] max bytes to peek (integer) - default is 32768
  #
  # in scalar context: data read from the TLS/SSL connection, undef on error
  # in list context:   two-item array consisting of data read (undef on error),
  #                      and return code from SSL_peek().
  ```

- peek_ex

  COMPATIBILITY: not available in Net-SSLeay-1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Copies `$max` bytes from the specified `$ssl` into the returned value. In contrast to the `Net::SSLeay::read_ex()` function, the data in the SSL buffer is unmodified after the **SSL_peek_ex()** operation.

```
my($got, $rv) = Net::SSLeay::peek_ex($ssl, $max);
# $ssl - value corresponding to openssl's SSL structure
# $max - [optional] max bytes to peek (integer) - default is 32768
#
# returns a list: two-item list consisting of data read (undef on error),
#                 and return code from SSL_peek_ex().
```

Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_peek_ex.html>

- pending

  Obtain number of readable bytes buffered in `$ssl` object.

  ```
  my $rv = Net::SSLeay::pending($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: the number of bytes pending
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_pending.html>

- has_pending

  COMPATIBILITY: not available in Net–SSLeay–1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

  Returns 1 if `$ssl` has buffered data (whether processed or unprocessed) and 0 otherwise.

  ```
  my $rv = Net::SSLeay::has_pending($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: (integer) 1 or 0
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_has_pending.html>

- read

  Tries to read `$max` bytes from the specified `$ssl`.

  ```
  my $got = Net::SSLeay::read($ssl, $max);
  my($got, $rv) = Net::SSLeay::read($ssl, $max);
  # $ssl - value corresponding to openssl's SSL structure
  # $max - [optional] max bytes to read (integer) - default is 32768
  #
  # returns:
  # in scalar context: data read from the TLS/SSL connection, undef on error
  # in list context:   two-item array consisting of data read (undef on error),
  #                      and return code from SSL_read().
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_read.html>

- read_ex

  COMPATIBILITY: not available in Net–SSLeay–1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Tries to read `$max` bytes from the specified `$ssl`.

  ```
  my($got, $rv) = Net::SSLeay::read_ex($ssl, $max);
  # $ssl - value corresponding to openssl's SSL structure
  # $max - [optional] max bytes to read (integer) - default is 32768
  #
  # returns a list: two-item list consisting of data read (undef on error),
  #                 and return code from SSL_read_ex().
  ```

Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_read_ex.html>

- renegotiate

  Turn on flags for renegotiation so that renegotiation will happen

  ```
  my $rv = Net::SSLeay::renegotiate($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 1 on success, 0 on failure
  ```

- rstate_string

  Returns a 2 letter string indicating the current read state of the SSL object `$ssl`.

  ```
  my $rv = Net::SSLeay::rstate_string($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 2-letter string
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_rstate_string.html>

- rstate_string_long

  Returns a string indicating the current read state of the SSL object ssl.

  ```
  my $rv = Net::SSLeay::rstate_string_long($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: string with current state
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_rstate_string.html>

- session_reused

  Query whether a reused session was negotiated during handshake.

  ```
  my $rv = Net::SSLeay::session_reused($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 0 - new session was negotiated; 1 - session was reused.
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_session_reused.html>

- set1_param

  Applies X509 verification parameters `$vpm` on `$ssl`

  ```
  my $rv = Net::SSLeay::set1_param($ssl, $vpm);
  # $ssl - value corresponding to openssl's SSL structure
  # $vpm - value corresponding to openssl's X509_VERIFY_PARAM structure
  #
  # returns: 1 on success, 0 on failure
  ```

- set_accept_state

  Sets `$ssl` to work in server mode.

  ```
  Net::SSLeay::set_accept_state($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_connect_state.html>

- set_bio

Connects the BIOs `$rbio` and `$wbio` for the read and write operations of the TLS/SSL (encrypted) side of `$ssl`.

```
Net::SSLeay::set_bio($ssl, $rbio, $wbio);
# $ssl - value corresponding to openssl's SSL structure
# $rbio - value corresponding to openssl's BIO structure
# $wbio - value corresponding to openssl's BIO structure
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_bio.html>

- set_cipher_list

  Sets the list of ciphers only for ssl.

  ```
  my $rv = Net::SSLeay::set_cipher_list($ssl, $str);
  # $ssl - value corresponding to openssl's SSL structure
  # $str - (string) cipher list e.g. '3DES:+RSA'
  #
  # returns: 1 if any cipher could be selected and 0 on complete failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_cipher_list.html>

- set_ciphersuites

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Configure the available TLSv1.3 ciphersuites.

  ```
  my $rv = Net::SSLeay::set_ciphersuites($ssl, $str);
  # $ssl  - value corresponding to openssl's SSL structure
  # $str  - colon (":") separated list of TLSv1.3 ciphersuite names in order of pr
  #
  # returns: (integer) 1 if the requested ciphersuite list was configured, and 0 c
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_set_ciphersuites.html>

- set_client_CA_list

  Sets the list of CAs sent to the client when requesting a client certificate for the chosen `$ssl`, overriding the setting valid for `$ssl`'s SSL_CTX object.

  ```
  my $rv = Net::SSLeay::set_client_CA_list($ssl, $list);
  # $ssl - value corresponding to openssl's SSL structure
  # $list - value corresponding to openssl's STACK_OF(X509_NAME) structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_client_CA_list.html>

- set_connect_state

  Sets `$ssl` to work in client mode.

  ```
  Net::SSLeay::set_connect_state($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_connect_state.html>

- set_fd

Sets the file descriptor `$fd` as the input/output facility for the TLS/SSL (encrypted) side of `$ssl`, `$fd` will typically be the socket file descriptor of a network connection.

```
my $rv = Net::SSLeay::set_fd($ssl, $fd);
# $ssl - value corresponding to openssl's SSL structure
# $fd - (integer) file handle (got via perl's fileno)
#
# returns: 1 on success, 0 on failure
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_fd.html>

- set_psk_client_callback

  Sets the psk client callback.

```
Net::SSLeay::set_psk_client_callback($ssl, sub { my $hint = shift; return ($iden
# $ssl - value corresponding to openssl's SSL structure
# $hint - PSK identity hint send by the server
# $identity - PSK identity
# $key - PSK key, hex string without the leading '0x', e.g. 'deadbeef'
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_psk_client_callback.html>

- set_rfd

  Sets the file descriptor `$fd` as the input (read) facility for the TLS/SSL (encrypted) side of `$ssl`.

```
my $rv = Net::SSLeay::set_rfd($ssl, $fd);
# $ssl - value corresponding to openssl's SSL structure
# $fd - (integer) file handle (got via perl's fileno)
#
# returns: 1 on success, 0 on failure
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_fd.html>

- set_wfd

```
my $rv = Net::SSLeay::set_wfd($ssl, $fd);
# $ssl - value corresponding to openssl's SSL structure
# $fd - (integer) file handle (got via perl's fileno)
#
# returns: 1 on success, 0 on failure
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_fd.html>

- set_info_callback

  Sets the callback function, that can be used to obtain state information for `$ssl` during connection setup and use. When callback is undef, the callback setting currently valid for ctx is used.

```
Net::SSLeay::set_info_callback($ssl, $cb, [$data]);
# $ssl - value corresponding to openssl's SSL structure
# $cb - sub { my ($ssl,$where,$ret,$data) = @_; ... }
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_info_callback.html>

- CTX_set_info_callback

  Sets the callback function on ctx, that can be used to obtain state information during ssl connection setup and use. When callback is undef, an existing callback will be disabled.

```
Net::SSLeay::CTX_set_info_callback($ssl, $cb, [$data]);
# $ssl - value corresponding to openssl's SSL structure
# $cb - sub { my ($ssl,$where,$ret,$data) = @_; ... }
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_info_callback.html>

- set_pref_cipher

  Sets the list of available ciphers for `$ssl` using the control string `$str`.

  ```
  my $rv = Net::SSLeay::set_pref_cipher($ssl, $str);
  # $ssl - value corresponding to openssl's SSL structure
  # $str - (string) cipher list e.g. '3DES:+RSA'
  #
  # returns: 1 if any cipher could be selected and 0 on complete failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_cipher_list.html>

- CTX_set_psk_client_callback

  Sets the psk client callback.

  ```
  Net::SSLeay::CTX_set_psk_client_callback($ssl, sub { my $hint = shift; return ($
  # $ssl - value corresponding to openssl's SSL structure
  # $hint - PSK identity hint send by the server
  # $identity - PSK identity
  # $key - PSK key, hex string without the leading '0x', e.g. 'deadbeef'
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_psk_client_callback.html>

- set_purpose

  ```
  my $rv = Net::SSLeay::set_purpose($ssl, $purpose);
  # $ssl - value corresponding to openssl's SSL structure
  # $purpose - (integer) purpose identifier
  #
  # returns: 1 on success, 0 on failure
  ```

  For more info about available `$purpose` identifiers see "CTX_set_purpose".

- set_quiet_shutdown

  Sets the 'quiet shutdown' flag for `$ssl` to be `$mode`.

  ```
  Net::SSLeay::set_quiet_shutdown($ssl, $mode);
  # $ssl - value corresponding to openssl's SSL structure
  # $mode - 0 or 1
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_quiet_shutdown.html>

- set_session

  Set a TLS/SSL session to be used during TLS/SSL connect.

```
my $rv = Net::SSLeay::set_session($to, $ses);
# $to - value corresponding to openssl's SSL structure
# $ses - value corresponding to openssl's SSL_SESSION structure
#
# returns: 1 on success, 0 on failure
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_session.html>

- set_session_id_context

  Sets the context `$sid_ctx` of length `$sid_ctx_len` within which a session can be reused for the `$ssl` object.

  ```
  my $rv = Net::SSLeay::set_session_id_context($ssl, $sid_ctx, $sid_ctx_len);
  # $ssl - value corresponding to openssl's SSL structure
  # $sid_ctx - data buffer
  # $sid_ctx_len - length of data in $sid_ctx
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_session_id_context.html>

- set_session_secret_cb

  Setup pre-shared secret session resumption function.

  ```
  Net::SSLeay::set_session_secret_cb($ssl, $func, $data);
  # $ssl - value corresponding to openssl's SSL structure
  # $func - perl reference to callback function
  # $data - [optional] data that will be passed to callback function when invoked
  #
  # returns: no return value
  ```

  The callback function will be called like: callback_function($secret, `$ciphers`, `$pref_cipher`, `$data`);

  # `$secret` is the current master session key, usually all 0s at the beginning of a session # `$ciphers` is ref to an array of peer cipher names # `$pref_cipher` is a ref to an index into the list of cipher names of # the preferred cipher. Set it if you want to specify a preferred cipher # `$data` is the data passed to set_session_secret_cb

  The callback function should return 1 if it likes the suggested cipher (or has selected an alternative by setting pref_cipher), else it should return 0 (in which case OpenSSL will select its own preferred cipher).

  With OpenSSL 1.1 and later, callback_function can change the master key for the session by altering `$secret` and returning 1.

- CTX_set_tlsext_ticket_getkey_cb

  Setup encryption for TLS session tickets (stateless session reuse).

  ```
  Net::SSLeay::CTX_set_tlsext_ticket_getkey_cb($ctx, $func, $data);
  # $ctx  - value corresponding to openssl's SSL_CTX structure
  # $func - perl reference to callback function
  # $data - [optional] data that will be passed to callback function when invoked
  #
  # returns: no return value
  ```

  The callback function will be called like: getkey($data,[$key_name]) −> ($key,$current_key_name)

  # `$data` is the data passed to set_session_secret_cb # `$key_name` is the name of the key OpenSSL has extracted from the session ticket # `$key` is the requested key for ticket encryption + HMAC #

`$current_key_name` is the name for the currently valid key

OpenSSL will call the function without a key name if it generates a new ticket. It then needs the callback to return the encryption+HMAC key and an identifier (key name) for this key.

When OpenSSL gets a session ticket from the client it extracts the key name and calls the callback with this name as argument. It then expects the callback to return the encryption+HMAC key matching the requested key name and and also the key name which should be used at the moment. If the requested key name and the returned key name differ it means that this session ticket was created with an expired key and need to be renewed. In this case OpenSSL will call the callback again with no key name to create a new session ticket based on the old one.

The key must be at least 32 byte of random data which can be created with RAND_bytes. Internally the first 16 byte are used as key in AES−128 encryption while the next 16 byte are used for the SHA−256 HMAC. The key name are binary data and must be exactly 16 byte long.

Example:

```
Net::SSLeay::RAND_bytes(my $oldkey,32);
Net::SSLeay::RAND_bytes(my $newkey,32);
my $oldkey_name = pack("a16",'oldsecret');
my $newkey_name = pack("a16",'newsecret');

my @keys = (
    [ $newkey_name, $newkey ], # current active key
    [ $oldkey_name, $oldkey ], # already expired
);

Net::SSLeay::CTX_set_tlsext_ticket_getkey_cb($server2->_ctx, sub {
    my ($mykeys,$name) = @_;

    # return (current_key, current_key_name) if no name given
    return ($mykeys->[0][1],$mykeys->[0][0]) if ! $name;

    # return (matching_key, current_key_name) if we find a key matching
    # the given name
    for(my $i = 0; $i<@$mykeys; $i++) {
        next if $name ne $mykeys->[$i][0];
        return ($mykeys->[$i][1],$mykeys->[0][0]);
    }

    # no matching key found
    return;
},\@keys);
```

This function is based on the OpenSSL function SSL_CTX_set_tlsext_ticket_key_cb but provides a simpler to use interface. For more information see <http://www.openssl.org/docs/ssl/SSL_CTX_set_tlsext_ticket_key_cb.html>

- set_session_ticket_ext_cb

  Setup callback for TLS session tickets (stateless session reuse).

```
Net::SSLeay::set_session_ticket_ext_cb($ssl, $func, $data);
# $ssl  - value corresponding to openssl's SSL structure
# $func - perl reference to callback function
# $data - [optional] data that will be passed to callback function when invoked
#
# returns: no return value
```

The callback function will be called like: getticket($ssl,$ticket,$data) −> `$return_value`

`$ssl` is a value corresponding to openssl's SSL structure # `$ticket` is a value of received TLS session ticket (can also be empty) # `$data` is the data passed to set_session_ticket_ext_cb # `$return_value` is either 0 (failure) or 1 (success)

This function is based on the OpenSSL function SSL_set_session_ticket_ext_cb.

- set_session_ticket_ext

  Set TLS session ticket (stateless session reuse).

  ```
  Net::SSLeay::set_session_ticket_ext($ssl, $ticket);
  # $ssl    - value corresponding to openssl's SSL structure
  # $ticket - is a value of TLS session ticket which client will send (can also be
  #
  # returns: no return value
  ```

  The callback function will be called like: getticket($ssl,$ticket,$data) −> `$return_value`

  # `$ssl` is a value corresponding to openssl's SSL structure # `$ticket` is a value of received TLS session ticket (can also be empty) # `$data` is the data passed to set_session_ticket_ext_cb # `$return_value` is either 0 (failure) or 1 (success)

  This function is based on the OpenSSL function SSL_set_session_ticket_ext_cb.

- set_shutdown

  Sets the shutdown state of `$ssl` to `$mode`.

  ```
  Net::SSLeay::set_shutdown($ssl, $mode);
  # $ssl - value corresponding to openssl's SSL structure
  # $mode - (integer) shutdown mode:
  #         0 - No shutdown
  #         1 - SSL_SENT_SHUTDOWN
  #         2 - SSL_RECEIVED_SHUTDOWN
  #         3 - SSL_RECEIVED_SHUTDOWN+SSL_SENT_SHUTDOWN
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_set_shutdown.html>

- set_ssl_method

  Sets a new TLS/SSL method for a particular `$ssl` object.

  ```
  my $rv = Net::SSLeay::set_ssl_method($ssl, $method);
  # $ssl - value corresponding to openssl's SSL structure
  # $method - value corresponding to openssl's SSL_METHOD structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_ssl_version.html>

- set_tmp_dh

  Sets DH parameters to be used to be `$dh`.

```
my $rv = Net::SSLeay::set_tmp_dh($ssl, $dh);
# $ssl - value corresponding to openssl's SSL structure
# $dh - value corresponding to openssl's DH structure
#
# returns: 1 on success, 0 on failure
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_dh_callback.html>

- set_tmp_dh_callback

  Sets the callback function for `$ssl` to be used when a DH parameters are required to `$dh_cb`.

  ??? (does this function really work?)

```
Net::SSLeay::set_tmp_dh_callback($ssl, $dh);
# $ssl - value corresponding to openssl's SSL structure
# $dh_cb - pointer to function ???
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_dh_callback.html>

- set_tmp_rsa

  Sets the temporary/ephemeral RSA key to be used in `$ssl` to be `$rsa`.

```
my $rv = Net::SSLeay::set_tmp_rsa($ssl, $rsa);
# $ssl - value corresponding to openssl's SSL structure
# $rsa - value corresponding to openssl's RSA structure
#
# returns: 1 on success, 0 on failure
```

  Example:

```
$rsakey = Net::SSLeay::RSA_generate_key();
Net::SSLeay::set_tmp_rsa($ssl, $rsakey);
Net::SSLeay::RSA_free($rsakey);
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_rsa_callback.html>

- set_tmp_rsa_callback

  Sets the callback function for `$ssl` to be used when a temporary/ephemeral RSA key is required to `$tmp_rsa_callback`.

  ??? (does this function really work?)

```
Net::SSLeay::set_tmp_rsa_callback($ssl, $tmp_rsa_callback);
# $ssl - value corresponding to openssl's SSL structure
# $tmp_rsa_callback - (function pointer) ???
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_tmp_rsa_callback.html>

- set_trust

```
my $rv = Net::SSLeay::set_trust($ssl, $trust);
# $ssl - value corresponding to openssl's SSL structure
# $trust - (integer) trust identifier
#
# returns: the original value
```

  For more details about `$trust` values see “CTX_set_trust”.

- shutdown

  Shuts down an active TLS/SSL connection. It sends the 'close notify' shutdown alert to the peer.

  ```
  my $rv = Net::SSLeay::shutdown($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 1 - shutdown was successfully completed
  #          0 - shutdown is not yet finished,
  #         -1 - shutdown was not successful
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_shutdown.html>

- state_string

  Returns a 6 letter string indicating the current state of the SSL object `$ssl`.

  ```
  my $rv = Net::SSLeay::state_string($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: 6-letter string
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_state_string.html>

- state_string_long

  Returns a string indicating the current state of the SSL object `$ssl`.

  ```
  my $rv = Net::SSLeay::state_string_long($ssl);
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: state strings
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_state_string.html>

- set_default_passwd_cb

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0f. Not needed with LibreSSL.

  Sets the default password callback called when loading/storing a PEM certificate with encryption for `$ssl`.

  ```
  Net::SSLeay::set_default_passwd_cb($ssl, $func);
  # $ssl - value corresponding to openssl's SSL structure
  # $func - perl reference to callback function
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_default_passwd_cb.html>

- set_default_passwd_cb_userdata

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0f. Not needed with LibreSSL.

  Sets a pointer to userdata which will be provided to the password callback of `$ssl` on invocation.

  ```
  Net::SSLeay::set_default_passwd_cb_userdata($ssl, $userdata);
  # $ssl - value corresponding to openssl's SSL structure
  # $userdata - data that will be passed to callback function when invoked
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_set_default_passwd_cb.html>

- use_PrivateKey

  Adds `$pkey` as private key to `$ssl`.

  ```
  my $rv = Net::SSLeay::use_PrivateKey($ssl, $pkey);
  # $ssl - value corresponding to openssl's SSL structure
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_PrivateKey_ASN1

  Adds the private key of type `$pk` stored in `$data` to `$ssl`.

  ```
  my $rv = Net::SSLeay::use_PrivateKey_ASN1($pk, $ssl, $d, $len);
  # $pk - (integer) key type, NID of corresponding algorithm
  # $ssl - value corresponding to openssl's SSL structure
  # $data - key data (binary)
  # $len - length of $data
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_PrivateKey_file

  Adds the first private key found in `$file` to `$ssl`.

  ```
  my $rv = Net::SSLeay::use_PrivateKey_file($ssl, $file, $type);
  # $ssl - value corresponding to openssl's SSL structure
  # $file - (string) file name
  # $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSI
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_RSAPrivateKey

  Adds `$rsa` as RSA private key to `$ssl`.

  ```
  my $rv = Net::SSLeay::use_RSAPrivateKey($ssl, $rsa);
  # $ssl - value corresponding to openssl's SSL structure
  # $rsa - value corresponding to openssl's RSA structure
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_RSAPrivateKey_ASN1

  Adds RSA private key stored in `$data` to `$ssl`.

  ```
  my $rv = Net::SSLeay::use_RSAPrivateKey_ASN1($ssl, $data, $len);
  # $ssl - value corresponding to openssl's SSL structure
  # $data - key data (binary)
  # $len - length of $data
  #
  # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_RSAPrivateKey_file

  Adds the first RSA private key found in `$file` to `$ssl`.

  ```
   my $rv = Net::SSLeay::use_RSAPrivateKey_file($ssl, $file, $type);
   # $ssl - value corresponding to openssl's SSL structure
   # $file - (string) file name
   # $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSl
   #
   # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_certificate

  Loads the certificate `$x` into `$ssl`.

  ```
   my $rv = Net::SSLeay::use_certificate($ssl, $x);
   # $ssl - value corresponding to openssl's SSL structure
   # $x - value corresponding to openssl's X509 structure
   #
   # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_certificate_ASN1

  Loads the ASN1 encoded certificate from `$data` to `$ssl`.

  ```
   my $rv = Net::SSLeay::use_certificate_ASN1($ssl, $data, $len);
   # $ssl - value corresponding to openssl's SSL structure
   # $data - certificate data (binary)
   # $len - length of $data
   #
   # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_certificate_chain_file

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.1.0

  Loads a certificate chain from `$file` into `$ssl`. The certificates must be in PEM format and must be sorted starting with the subject's certificate (actual client or server certificate), followed by intermediate CA certificates if applicable, and ending at the highest level (root) CA.

  ```
   my $rv = Net::SSLeay::use_certificate_chain_file($ssl, $file);
   # $ssl - value corresponding to openssl's SSL structure
   # $file - (string) file name
   #
   # returns: 1 on success, otherwise check out the error stack to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- use_certificate_file

  Loads the first certificate stored in `$file` into `$ssl`.

```
my $rv = Net::SSLeay::use_certificate_file($ssl, $file, $type);
# $ssl - value corresponding to openssl's SSL structure
# $file - (string) file name
# $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSI
#
# returns: 1 on success, otherwise check out the error stack to find out the rea
```

Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html>

- get_version

  Returns SSL/TLS protocol name

```
my $rv = Net::SSLeay::get_version($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (string) protocol name, see OpenSSL manual for the full list
#          TLSv1
#          TLSv1.3
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_get_version.html>

- version

  Returns SSL/TLS protocol version

```
my $rv = Net::SSLeay::version($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (integer) protocol version, see OpenSSL manual for the full list
#          0x0301 - TLS1_VERSION  (TLSv1)
#          0xFEFF - DTLS1_VERSION (DTLSv1)
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_version.html>

- client_version

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

  Returns TLS protocol version used by the client when initiating the connection

```
my $rv = Net::SSLeay::client_version($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (integer) protocol version, see OpenSSL manual for the full list
#          0x0301 - TLS1_VERSION  (TLSv1)
#          0xFEFF - DTLS1_VERSION (DTLSv1)
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_client_version.html>

- is_dtls

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.0, not in LibreSSL

```
my $rv = Net::SSLeay::is_dtls($ssl);
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (integer) zero or one
#          0 - connection is not using DTLS
#          1 - connection is using DTLS
```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_is_dtls.html>

- want

  Returns state information for the SSL object `$ssl`.

  ```
   my $rv = Net::SSLeay::want($ssl);
   # $ssl - value corresponding to openssl's SSL structure
   #
   # returns: state
   #          1 - SSL_NOTHING
   #          2 - SSL_WRITING
   #          3 - SSL_READING
   #          4 - SSL_X509_LOOKUP
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_want.html>

- write

  Writes data from the buffer `$data` into the specified `$ssl` connection.

  ```
   my $rv = Net::SSLeay::write($ssl, $data);
   # $ssl - value corresponding to openssl's SSL structure
   # $data - data to be written
   #
   # returns: >0 - (success) number of bytes actually written to the TLS/SSL connec
   #           0 - write not successful, probably the underlying connection was clo
   #          <0 - error
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_write.html>

- write_ex

  COMPATIBILITY: not available in Net–SSLeay–1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

  Writes data from the buffer `$data` into the specified `$ssl` connection.

  ```
   my ($len, $rv) = Net::SSLeay::write_ex($ssl, $data);
   # $ssl - value corresponding to openssl's SSL structure
   # $data - data to be written
   #
   # returns a list: two-item list consisting of number of bytes written,
   #                 and return code from SSL_write_ex()
  ```

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/SSL_write_ex.html>

- write_partial

  NOTE: Does not exactly correspond to any low level API function

  Writes a fragment of data in `$data` from the buffer `$data` into the specified `$ssl` connection. This is a non-blocking function like **Net::SSLeay::write()**.

  ```
   my $rv = Net::SSLeay::write_partial($ssl, $from, $count, $data);
   # $ssl - value corresponding to openssl's SSL structure
   # $from - (integer) offset from the beginning of $data
   # $count - (integer) length of data to be written
   # $data - data buffer
   #
   # returns: >0 - (success) number of bytes actually written to the TLS/SSL connec
   #           0 - write not successful, probably the underlying connection was clo
   #          <0 - error
  ```

- set_tlsext_host_name

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.8f

  Sets TLS servername extension on SLL object `$ssl` to value `$name`.

  ```
  my $rv = set_tlsext_host_name($ssl, $name);
  # $ssl - value corresponding to openssl's SSL structure
  # $name - (string) name to be set
  #
  # returns: 1 on success, 0 on failure
  ```

Low level API: RAND_* related functions

Check openssl doc related to RAND stuff <http://www.openssl.org/docs/crypto/rand.html>

- RAND_add

  Mixes the `$num` bytes at `$buf` into the PRNG state.

  ```
  Net::SSLeay::RAND_add($buf, $num, $entropy);
  # $buf - buffer with data to be mixed into the PRNG state
  # $num - number of bytes in $buf
  # $entropy - estimate of how much randomness is contained in $buf (in bytes)
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_add.html>

- RAND_seed

  Equivalent to "RAND_add" when `$num` == `$entropy`.

  ```
  Net::SSLeay::RAND_seed($buf);   # Perlishly figures out buf size
  # $buf - buffer with data to be mixed into the PRNG state
  # $num - number of bytes in $buf
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_add.html>

- RAND_status

  Gives PRNG status (seeded enough or not).

  ```
  my $rv = Net::SSLeay::RAND_status();
  #returns: 1 if the PRNG has been seeded with enough data, 0 otherwise
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_add.html>

- RAND_bytes

  Puts `$num` cryptographically strong pseudo-random bytes into `$buf`.

  ```
  my $rv = Net::SSLeay::RAND_bytes($buf, $num);
  # $buf - buffer where the random data will be stored
  # $num - the size (in bytes) of requested random data
  #
  # returns: 1 on success, -1 if not supported by the current RAND method, or 0 on
  ```

  Check openssl doc <http://www.openssl.org/docs/manmaster/man3/RAND_bytes.html>

- RAND_priv_bytes

  COMPATIBILITY: not available in Net−SSLeay−1.85 and before; requires at least OpenSSL 1.1.1, not in LibreSSL

Puts `$num` cryptographically strong pseudo-random bytes into `$buf`.

```
my $rv = Net::SSLeay::RAND_priv_bytes($buf, $num);
# $buf - buffer where the random data will be stored
# $num - the size (in bytes) of requested random data
#
# returns: 1 on success, -1 if not supported by the current RAND method, or 0 on
```

RAND_priv_bytes has the same semantics as RAND_bytes, but see see the documentation for more information.

Check openssl doc <http://www.openssl.org/docs/manmaster/man3/RAND_priv_bytes.html>

- RAND_pseudo_bytes

  Puts `$num` pseudo-random (not necessarily unpredictable) bytes into `$buf`.

```
my $rv = Net::SSLeay::RAND_pseudo_bytes($buf, $num);
# $buf - buffer where the random data will be stored
# $num - the size (in bytes) of requested random data
#
# returns: 1 if the bytes generated are cryptographically strong, 0 otherwise
```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_bytes.html>

- RAND_cleanup

  Erase the PRNG state.

```
Net::SSLeay::RAND_cleanup();
# no args, no return value
```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_cleanup.html>

- RAND_egd_bytes

  Queries the entropy gathering daemon EGD on socket `$path` for `$bytes` bytes.

```
my $rv = Net::SSLeay::RAND_egd_bytes($path, $bytes);
# $path - path to a socket of entropy gathering daemon EGD
# $bytes - number of bytes we want from EGD
#
# returns: the number of bytes read from the daemon on success, and -1 on failur
```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_egd.html>

- RAND_file_name

  Generates a default path for the random seed file.

```
my $file = Net::SSLeay::RAND_file_name($num);
# $num - maximum size of returned file name
#
# returns: string with file name on success, '' (empty string) on failure
```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_load_file.html>

- RAND_load_file

  COMPATIBILITY: Is no longer functional on LibreSSL

  Reads `$max_bytes` of bytes from `$file_name` and adds them to the PRNG.

```
my $rv = Net::SSLeay::RAND_load_file($file_name, $max_bytes);
# $file_name - the name of file
# $max_bytes - bytes to read from $file_name; -1 => the complete file is read
#
# returns: the number of bytes read
```

Check openssl doc <http://www.openssl.org/docs/crypto/RAND_load_file.html>

- RAND_write_file

  Writes 1024 random bytes to `$file_name` which can be used to initialize the PRNG by calling "RAND_load_file" in a later session.

  ```
  my $rv = Net::SSLeay::RAND_write_file($file_name);
  # $file_name - the name of file
  #
  # returns: the number of bytes written, and -1 if the bytes written were generat
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/RAND_load_file.html>

- RAND_poll

  Collects some entropy from operating system and adds it to the PRNG.

  ```
  my $rv = Net::SSLeay::RAND_poll();
  # returns: 1 on success, 0 on failure (unable to gather reasonable entropy)
  ```

Low level API: OBJ_* related functions

- OBJ_cmp

  Compares ASN1_OBJECT `$a` to ASN1_OBJECT `$b`.

  ```
  my $rv = Net::SSLeay::OBJ_cmp($a, $b);
  # $a - value corresponding to openssl's ASN1_OBJECT structure
  # $b - value corresponding to openssl's ASN1_OBJECT structure
  #
  # returns: if the two are identical 0 is returned
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_dup

  Returns a copy/duplicate of `$o`.

  ```
  my $rv = Net::SSLeay::OBJ_dup($o);
  # $o - value corresponding to openssl's ASN1_OBJECT structure
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_nid2ln

  Returns long name for given NID `$n`.

  ```
  my $rv = Net::SSLeay::OBJ_nid2ln($n);
  # $n - (integer) NID
  #
  # returns: (string) long name e.g. 'commonName'
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_ln2nid

  Returns NID corresponding to given long name `$n`.

```
my $rv = Net::SSLeay::OBJ_ln2nid($s);
# $s - (string) long name e.g. 'commonName'
#
# returns: (integer) NID
```

- OBJ_nid2sn

  Returns short name for given NID `$n`.

  ```
  my $rv = Net::SSLeay::OBJ_nid2sn($n);
  # $n - (integer) NID
  #
  # returns: (string) short name e.g. 'CN'
  ```

  Example:

  ```
  print Net::SSLeay::OBJ_nid2sn(&Net::SSLeay::NID_commonName);
  ```

- OBJ_sn2nid

  Returns NID corresponding to given short name `$s`.

  ```
  my $rv = Net::SSLeay::OBJ_sn2nid($s);
  # $s - (string) short name e.g. 'CN'
  #
  # returns: (integer) NID
  ```

  Example:

  ```
  print "NID_commonName constant=", &Net::SSLeay::NID_commonName;
  print "OBJ_sn2nid('CN')=", Net::SSLeay::OBJ_sn2nid('CN');
  ```

- OBJ_nid2obj

  Returns ASN1_OBJECT for given NID `$n`.

  ```
  my $rv = Net::SSLeay::OBJ_nid2obj($n);
  # $n - (integer) NID
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_obj2nid

  Returns NID corresponding to given ASN1_OBJECT `$o`.

  ```
  my $rv = Net::SSLeay::OBJ_obj2nid($o);
  # $o - value corresponding to openssl's ASN1_OBJECT structure
  #
  # returns: (integer) NID
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_txt2obj

  Converts the text string s into an ASN1_OBJECT structure. If `$no_name` is 0 then long names (e.g. 'commonName') and short names (e.g. 'CN') will be interpreted as well as numerical forms (e.g. '2.5.4.3'). If `$no_name` is 1 only the numerical form is acceptable.

  ```
  my $rv = Net::SSLeay::OBJ_txt2obj($s, $no_name);
  # $s - text string to be converted
  # $no_name - (integer) 0 or 1
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_obj2txt

  Converts the ASN1_OBJECT a into a textual representation.

```
 Net::SSLeay::OBJ_obj2txt($a, $no_name);
 # $a - value corresponding to openssl's ASN1_OBJECT structure
 # $no_name - (integer) 0 or 1
 #
 # returns: textual representation e.g. 'commonName' ($no_name=0), '2.5.4.3' ($nc
```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

- OBJ_txt2nid

  Returns NID corresponding to text string `$s` which can be a long name, a short name or the numerical representation of an object.

```
 my $rv = Net::SSLeay::OBJ_txt2nid($s);
 # $s - (string) e.g. 'commonName' or 'CN' or '2.5.4.3'
 #
 # returns: (integer) NID
```

  Example:

```
 my $nid = Net::SSLeay::OBJ_txt2nid('2.5.4.3');
 Net::SSLeay::OBJ_nid2sn($n);
```

  Check openssl doc <http://www.openssl.org/docs/crypto/OBJ_nid2obj.html>

Low level API: ASN1_INTEGER_* related functions

- ASN1_INTEGER_new

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Creates a new ASN1_INTEGER structure.

```
 my $rv = Net::SSLeay::ASN1_INTEGER_new();
 #
 # returns: value corresponding to openssl's ASN1_INTEGER structure (0 on failure
```

- ASN1_INTEGER_free

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Free an allocated ASN1_INTEGER structure.

```
 Net::SSLeay::ASN1_INTEGER_free($i);
 # $i - value corresponding to openssl's ASN1_INTEGER structure
 #
 # returns: no return value
```

- ASN1_INTEGER_get

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns integer value of given ASN1_INTEGER object.

  BEWARE: If the value stored in ASN1_INTEGER is greater than max. integer that can be stored in 'long' type (usually 32bit but may vary according to platform) then this function will return −1. For getting large ASN1_INTEGER values consider using "P_ASN1_INTEGER_get_dec" or "P_ASN1_INTEGER_get_hex".

```
my $rv = Net::SSLeay::ASN1_INTEGER_get($a);
# $a - value corresponding to openssl's ASN1_INTEGER structure
#
# returns: integer value of ASN1_INTEGER object in $a
```

- ASN1_INTEGER_set

  COMPATIBILITY: not available in Net‐SSLeay‐1.45 and before

  Sets value of given ASN1_INTEGER object to value `$val`

  BEWARE: `$val` has max. limit (= max. integer that can be stored in 'long' type). For setting large ASN1_INTEGER values consider using "P_ASN1_INTEGER_set_dec" or "P_ASN1_INTEGER_set_hex".

```
my $rv = Net::SSLeay::ASN1_INTEGER_set($i, $val);
# $i - value corresponding to openssl's ASN1_INTEGER structure
# $val - integer value
#
# returns: 1 on success, 0 on failure
```

- P_ASN1_INTEGER_get_dec

  COMPATIBILITY: not available in Net‐SSLeay‐1.45 and before

  Returns string with decimal representation of integer value of given ASN1_INTEGER object.

```
Net::SSLeay::P_ASN1_INTEGER_get_dec($i);
# $i - value corresponding to openssl's ASN1_INTEGER structure
#
# returns: string with decimal representation
```

- P_ASN1_INTEGER_get_hex

  COMPATIBILITY: not available in Net‐SSLeay‐1.45 and before

  Returns string with hexadecimal representation of integer value of given ASN1_INTEGER object.

```
Net::SSLeay::P_ASN1_INTEGER_get_hex($i);
# $i - value corresponding to openssl's ASN1_INTEGER structure
#
# returns: string with hexadecimal representation
```

- P_ASN1_INTEGER_set_dec

  COMPATIBILITY: not available in Net‐SSLeay‐1.45 and before

  Sets value of given ASN1_INTEGER object to value `$val` (decimal string, suitable for large integers)

```
Net::SSLeay::P_ASN1_INTEGER_set_dec($i, $str);
# $i - value corresponding to openssl's ASN1_INTEGER structure
# $str - string with decimal representation
#
# returns: 1 on success, 0 on failure
```

- P_ASN1_INTEGER_set_hex

  COMPATIBILITY: not available in Net‐SSLeay‐1.45 and before

  Sets value of given ASN1_INTEGER object to value `$val` (hexadecimal string, suitable for large integers)

```
Net::SSLeay::P_ASN1_INTEGER_set_hex($i, $str);
# $i - value corresponding to openssl's ASN1_INTEGER structure
# $str - string with hexadecimal representation
#
# returns: 1 on success, 0 on failure
```

Low level API: ASN1_STRING_* related functions

- P_ASN1_STRING_get

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns string value of given ASN1_STRING object.

```
Net::SSLeay::P_ASN1_STRING_get($s, $utf8_decode);
# $s - value corresponding to openssl's ASN1_STRING structure
# $utf8_decode - [optional] 0 or 1 whether the returned value should be utf8 dec
#
# returns: string

$string = Net::SSLeay::P_ASN1_STRING_get($s);
#is the same as:
$string = Net::SSLeay::P_ASN1_STRING_get($s, 0);
```

Low level API: ASN1_TIME_* related functions

- ASN1_TIME_new

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

```
my $time = ASN1_TIME_new();
# returns: value corresponding to openssl's ASN1_TIME structure
```

- ASN1_TIME_free

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

```
ASN1_TIME_free($time);
# $time - value corresponding to openssl's ASN1_TIME structure
```

- ASN1_TIME_set

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

```
ASN1_TIME_set($time, $t);
# $time - value corresponding to openssl's ASN1_TIME structure
# $t - time value in seconds since 1.1.1970
```

  BEWARE: It is platform dependent how this function will handle dates after 2038. Although perl's integer is large enough the internal implementation of this function is dependent on the size of time_t structure (32bit time_t has problem with 2038).

  If you want to safely set date and time after 2038 use function "P_ASN1_TIME_set_isotime".

- P_ASN1_TIME_get_isotime

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7e

  NOTE: Does not exactly correspond to any low level API function

  Gives ISO−8601 string representation of ASN1_TIME structure.

```
my $datetime_string = P_ASN1_TIME_get_isotime($time);
# $time - value corresponding to openssl's ASN1_TIME structure
#
# returns: datetime string like '2033-05-16T20:39:37Z' or '' on failure
```

The output format is compatible with module DateTime::Format::RFC3339

- P_ASN1_TIME_set_isotime

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7e

  NOTE: Does not exactly correspond to any low level API function

  Sets time and date value of ANS1_time structure.

  ```
  my $rv = P_ASN1_TIME_set_isotime($time, $string);
  # $time - value corresponding to openssl's ASN1_TIME structure
  # $string - ISO-8601 timedate string like '2033-05-16T20:39:37Z'
  #
  # returns: 1 on success, 0 on failure
  ```

  The `$string` parameter has to be in full form like `"2012-03-22T23:55:33"` or `"2012-03-22T23:55:33Z"` or `"2012-03-22T23:55:33CET"`. Short forms like `"2012-03-22T23:55"` or `"2012-03-22"` are not supported.

- P_ASN1_TIME_put2string

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before, has bugs with openssl−0.9.8i

  NOTE: Does not exactly correspond to any low level API function

  Gives string representation of ASN1_TIME structure.

  ```
  my $str = P_ASN1_TIME_put2string($time);
  # $time - value corresponding to openssl's ASN1_TIME structure
  #
  # returns: datetime string like 'May 16 20:39:37 2033 GMT'
  ```

- P_ASN1_UTCTIME_put2string

  NOTE: deprecated function, only for backward compatibility, just an alias for "P_ASN1_TIME_put2string"

Low level API: X509_* related functions

- X509_new

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Allocates and initializes a X509 structure.

  ```
  my $rv = Net::SSLeay::X509_new();
  #
  # returns: value corresponding to openssl's X509 structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_new.html>

- X509_free

  Frees up the X509 structure.

  ```
  Net::SSLeay::X509_free($a);
  # $a - value corresponding to openssl's X509 structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_new.html>

- X509_check_host

  COMPATIBILITY: not available in Net−SSLeay−1.68 and before; requires at least OpenSSL 1.0.2. X509_CHECK_FLAG_NEVER_CHECK_SUBJECT requires OpenSSL 1.1.0.

  Checks f the certificate Subject Alternative Name (SAN) or Subject CommonName (CN) matches the specified host name.

  ```
  my $rv = Net::SSLeay::X509_check_host($cert, $name, $flags, $peername);
  # $cert - value corresponding to openssl's X509 structure
  # $name - host name to check
  # $flags (optional, default: 0) - can be the bitwise OR of:
  #   &Net::SSLeay::X509_CHECK_FLAG_ALWAYS_CHECK_SUBJECT
  #   &Net::SSLeay::X509_CHECK_FLAG_NO_WILDCARDS
  #   &Net::SSLeay::X509_CHECK_FLAG_NO_PARTIAL_WILDCARDS
  #   &Net::SSLeay::X509_CHECK_FLAG_MULTI_LABEL_WILDCARDS
  #   &Net::SSLeay::X509_CHECK_FLAG_SINGLE_LABEL_SUBDOMAINS
  #   &Net::SSLeay::X509_CHECK_FLAG_NEVER_CHECK_SUBJECT
  # $peername (optional) - If not omitted and $host matches $cert,
  #                        a copy of the matching SAN or CN from
  #                        the peer certificate is stored in $peername.
  #
  # returns:
  #   1 for a successful match
  #   0 for a failed match
  #  -1 for an internal error
  #  -2 if the input is malformed
  ```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_check_host.html>.

- X509_check_email

  COMPATIBILITY: not available in Net−SSLeay−1.68 and before; requires at least OpenSSL 1.0.2.

  Checks if the certificate matches the specified email address.

  ```
  my $rv = Net::SSLeay::X509_check_email($cert, $address, $flags);
  # $cert - value corresponding to openssl's X509 structure
  # $address - email address to check
  # $flags (optional, default: 0) - see X509_check_host()
  #
  # returns: see X509_check_host()
  ```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_check_email.html>.

- X509_check_ip

  COMPATIBILITY: not available in Net−SSLeay−1.68 and before; requires at least OpenSSL 1.0.2.

  Checks if the certificate matches the specified IPv4 or IPv6 address.

  ```
  my $rv = Net::SSLeay::X509_check_email($cert, $address, $flags);
  # $cert - value corresponding to openssl's X509 structure
  # $address - IP address to check in binary format, in network byte order
  # $flags (optional, default: 0) - see X509_check_host()
  #
  # returns: see X509_check_host()
  ```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_check_ip.html>.

- X509_check_ip_asc

  COMPATIBILITY: not available in Net−SSLeay−1.68 and before; requires at least OpenSSL 1.0.2.

Checks if the certificate matches the specified IPv4 or IPv6 address.

```
my $rv = Net::SSLeay::X509_check_email($cert, $address, $flags);
# $cert - value corresponding to openssl's X509 structure
# $address - IP address to check in text representation
# $flags (optional, default: 0) - see X509_check_host()
#
# returns: see X509_check_host()
```

Check openssl doc <https://www.openssl.org/docs/crypto/X509_check_ip_asc.html>.

- X509_certificate_type

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns bitmask with type of certificate `$x`.

```
my $rv = Net::SSLeay::X509_certificate_type($x);
# $x - value corresponding to openssl's X509 structure
#
# returns: (integer) bitmask with certificate type

#to decode bitmask returned by this function use these constants:
&Net::SSLeay::EVP_PKS_DSA
&Net::SSLeay::EVP_PKS_EC
&Net::SSLeay::EVP_PKS_RSA
&Net::SSLeay::EVP_PKT_ENC
&Net::SSLeay::EVP_PKT_EXCH
&Net::SSLeay::EVP_PKT_EXP
&Net::SSLeay::EVP_PKT_SIGN
&Net::SSLeay::EVP_PK_DH
&Net::SSLeay::EVP_PK_DSA
&Net::SSLeay::EVP_PK_EC
&Net::SSLeay::EVP_PK_RSA
```

- X509_digest

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Computes digest/fingerprint of X509 `$data` using `$type` hash function.

```
my $digest_value = Net::SSLeay::X509_digest($data, $type);
# $data - value corresponding to openssl's X509 structure
# $type - value corresponding to openssl's EVP_MD structure - e.g. got via EVP_g
#
# returns: hash value (binary)

#to get printable (hex) value of digest use:
print unpack('H*', $digest_value);
```

- X509_issuer_and_serial_hash

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sort of a checksum of issuer name and serial number of X509 certificate `$x`. The result is not a full hash (e.g. sha−1), it is kind-of-a-hash truncated to the size of 'unsigned long' (32 bits). The resulting value might differ across different openssl versions for the same X509 certificate.

```
my $rv = Net::SSLeay::X509_issuer_and_serial_hash($x);
# $x - value corresponding to openssl's X509 structure
#
# returns: number representing checksum
```

- X509_issuer_name_hash

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sort of a checksum of issuer name of X509 certificate `$x`. The result is not a full hash (e.g. sha−1), it is kind-of-a-hash truncated to the size of 'unsigned long' (32 bits). The resulting value might differ across different openssl versions for the same X509 certificate.

```
my $rv = Net::SSLeay::X509_issuer_name_hash($x);
# $x - value corresponding to openssl's X509 structure
#
# returns: number representing checksum
```

- X509_subject_name_hash

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sort of a checksum of subject name of X509 certificate `$x`. The result is not a full hash (e.g. sha−1), it is kind-of-a-hash truncated to the size of 'unsigned long' (32 bits). The resulting value might differ across different openssl versions for the same X509 certificate.

```
my $rv = Net::SSLeay::X509_subject_name_hash($x);
# $x - value corresponding to openssl's X509 structure
#
# returns: number representing checksum
```

- X509_pubkey_digest

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Computes digest/fingerprint of public key from X509 certificate `$data` using `$type` hash function.

```
my $digest_value = Net::SSLeay::X509_pubkey_digest($data, $type);
# $data - value corresponding to openssl's X509 structure
# $type - value corresponding to openssl's EVP_MD structure - e.g. got via EVP_g
#
# returns: hash value (binary)

#to get printable (hex) value of digest use:
print unpack('H*', $digest_value);
```

- X509_set_issuer_name

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets issuer of X509 certificate `$x` to `$name`.

```
my $rv = Net::SSLeay::X509_set_issuer_name($x, $name);
# $x - value corresponding to openssl's X509 structure
# $name - value corresponding to openssl's X509_NAME structure
#
# returns: 1 on success, 0 on failure
```

- X509_set_pubkey

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets public key of X509 certificate `$x` to `$pkey`.

```
my $rv = Net::SSLeay::X509_set_pubkey($x, $pkey);
# $x - value corresponding to openssl's X509 structure
# $pkey - value corresponding to openssl's EVP_PKEY structure
#
# returns: 1 on success, 0 on failure
```

- X509_set_serialNumber

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets serial number of X509 certificate `$x` to `$serial`.

  ```
  my $rv = Net::SSLeay::X509_set_serialNumber($x, $serial);
  # $x - value corresponding to openssl's X509 structure
  # $serial - value corresponding to openssl's ASN1_INTEGER structure
  #
  # returns: 1 on success, 0 on failure

  #to create $serial value use one of these:
  $serial = Net::SSLeay::P_ASN1_INTEGER_set_hex('45ad6f');
  $serial = Net::SSLeay::P_ASN1_INTEGER_set_dec('7896541238529631478');
  $serial = Net::SSLeay::ASN1_INTEGER_set(45896);
  ```

- X509_set_subject_name

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets subject of X509 certificate `$x` to `$name`.

  ```
  my $rv = Net::SSLeay::X509_set_subject_name($x, $name);
  # $x - value corresponding to openssl's X509 structure
  # $name - value corresponding to openssl's X509_NAME structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_set_version

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Set 'version' value for X509 certificate $ to `$version`.

  ```
  my $rv = Net::SSLeay::X509_set_version($x, $version);
  # $x - value corresponding to openssl's X509 structure
  # $version - (integer) version number
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_sign

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sign X509 certificate `$x` with private key `$pkey` (using digest algorithm `$md`).

  ```
  my $rv = Net::SSLeay::X509_sign($x, $pkey, $md);
  # $x - value corresponding to openssl's X509 structure
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  # $md - value corresponding to openssl's EVP_MD structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_verify

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

Verifies X509 object `$a` using public key `$r` (pubkey of issuing CA).

```
 my $rv = Net::SSLeay::X509_verify($x, $r);
 # $x - value corresponding to openssl's X509 structure
 # $r - value corresponding to openssl's EVP_PKEY structure
 #
 # returns: 0 - verify failure, 1 - verify OK, <0 - error
```

- X509_get_ext_count

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns the total number of extensions in X509 object `$x`.

```
 my $rv = Net::SSLeay::X509_get_ext_count($x);
 # $x - value corresponding to openssl's X509 structure
 #
 # returns: count of extensions
```

- X509_get_pubkey

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns public key corresponding to given X509 object `$x`.

```
 my $rv = Net::SSLeay::X509_get_pubkey($x);
 # $x - value corresponding to openssl's X509 structure
 #
 # returns: value corresponding to openssl's EVP_PKEY structure (0 on failure)
```

  NOTE: This method returns only the public key's key bits, without the algorithm or parameters. Use `X509_get_X509_PUBKEY()` to return the full public key (SPKI) instead.

- X509_get_X509_PUBKEY

  COMPATIBILITY: not available in Net−SSLeay−1.72 and before

  Returns the full public key (SPKI) of given X509 certificate `$x`.

```
 Net::SSLeay::X509_get_X509_PUBKEY($x);
 # $x - value corresponding to openssl's X509 structure
 #
 # returns: public key data in DER format (binary)
```

- X509_get_serialNumber

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns serial number of X509 certificate `$x`.

```
 my $rv = Net::SSLeay::X509_get_serialNumber($x);
 # $x - value corresponding to openssl's X509 structure
 #
 # returns: value corresponding to openssl's ASN1_INTEGER structure (0 on failure
```

  See "P_ASN1_INTEGER_get_dec", "P_ASN1_INTEGER_get_hex" or "ASN1_INTEGER_get" to decode ASN1_INTEGER object.

- X509_get0_serialNumber

  COMPATIBILITY: available in Net−SSLeay−1.86 onwards

  X509_get0_serialNumber() is the same as **X509_get_serialNumber()** except it accepts a const parameter and returns a const result.

- X509_get_version

COMPATIBILITY: not available in Net−SSLeay−1.45 and before

Returns 'version' value of given X509 certificate `$x`.

```
my $rv = Net::SSLeay::X509_get_version($x);
# $x - value corresponding to openssl's X509 structure
#
# returns: (integer) version
```

- X509_get_ext

Returns X509_EXTENSION from `$x509` based on given position/index.

```
my $rv = Net::SSLeay::X509_get_ext($x509, $index);
# $x509 - value corresponding to openssl's X509 structure
# $index - (integer) position/index of extension within $x509
#
# returns: value corresponding to openssl's X509_EXTENSION structure (0 on failu
```

- X509_get_ext_by_NID

Returns X509_EXTENSION from `$x509` based on given NID.

```
my $rv = Net::SSLeay::X509_get_ext_by_NID($x509, $nid, $loc);
# $x509 - value corresponding to openssl's X509 structure
# $nid - (integer) NID value
# $loc - (integer) position to start lookup at
#
# returns: position/index of extension, negative value on error
#          call Net::SSLeay::X509_get_ext($x509, $rv) to get the actual extensic
```

- X509_get_fingerprint

Returns fingerprint of certificate `$cert`.

NOTE: Does not exactly correspond to any low level API function. The implementation is basen on openssl's `X509_digest()`.

```
Net::SSLeay::X509_get_fingerprint($x509, $type);
# $x509 - value corresponding to openssl's X509 structure
# $type - (string) digest type, currently supported values:
#         "md5"
#         "sha1"
#         "sha256"
#         "ripemd160"
#
# returns: certificate digest - hexadecimal string (NOT binary data!)
```

- X509_get_issuer_name

Return an X509_NAME object representing the issuer of the certificate `$cert`.

```
my $rv = Net::SSLeay::X509_get_issuer_name($cert);
# $cert - value corresponding to openssl's X509 structure
#
# returns: value corresponding to openssl's X509_NAME structure (0 on failure)
```

- X509_get_notAfter

Return an object giving the time after which the certificate `$cert` is not valid.

```
my $rv = Net::SSLeay::X509_get_notAfter($cert);
# $cert - value corresponding to openssl's X509 structure
#
# returns: value corresponding to openssl's ASN1_TIME structure (0 on failure)
```

To get human readable/printable form the return value you can use:

```
my $time = Net::SSLeay::X509_get_notAfter($cert);
print "notAfter=", Net::SSLeay::P_ASN1_TIME_get_isotime($time), "\n";
```

- X509_get_notBefore

  Return an object giving the time before which the certificate `$cert` is not valid

  ```
  my $rv = Net::SSLeay::X509_get_notBefore($cert);
  # $cert - value corresponding to openssl's X509 structure
  #
  # returns: value corresponding to openssl's ASN1_TIME structure (0 on failure)
  ```

  To get human readable/printable form the return value you can use:

  ```
  my $time = Net::SSLeay::X509_get_notBefore($cert);
  print "notBefore=", Net::SSLeay::P_ASN1_TIME_get_isotime($time), "\n";
  ```

- X509_get_subjectAltNames

  NOTE: Does not exactly correspond to any low level API function.

  Returns the list of alternative subject names from X509 certificate `$cert`.

  ```
  my @rv = Net::SSLeay::X509_get_subjectAltNames($cert);
  # $cert - value corresponding to openssl's X509 structure
  #
  # returns: list containing pairs - name_type (integer), name_value (string)
  #          where name_type can be:
  #          0 - GEN_OTHERNAME
  #          1 - GEN_EMAIL
  #          2 - GEN_DNS
  #          3 - GEN_X400
  #          4 - GEN_DIRNAME
  #          5 - GEN_EDIPARTY
  #          6 - GEN_URI
  #          7 - GEN_IPADD
  #          8 - GEN_RID
  ```

  Note: type 7 – GEN_IPADD contains the IP address as a packed binary address.

- X509_get_subject_name

  Returns the subject of the certificate `$cert`.

  ```
  my $rv = Net::SSLeay::X509_get_subject_name($cert);
  # $cert - value corresponding to openssl's X509 structure
  #
  # returns: value corresponding to openssl's X509_NAME structure (0 on failure)
  ```

- X509_gmtime_adj

  Adjust th ASN1_TIME object to the timestamp (in GMT).

```
my $rv = Net::SSLeay::X509_gmtime_adj($s, $adj);
# $s - value corresponding to openssl's ASN1_TIME structure
# $adj - timestamp (seconds since 1.1.1970)
#
# returns: value corresponding to openssl's ASN1_TIME structure (0 on failure)
```

BEWARE: this function may fail for dates after 2038 as it is dependent on time_t size on your system (32bit time_t does not work after 2038). Consider using "P_ASN1_TIME_set_isotime" instead).

- X509_load_cert_crl_file

  Takes PEM file and loads all X509 certificates and X509 CRLs from that file into X509_LOOKUP structure.

```
my $rv = Net::SSLeay::X509_load_cert_crl_file($ctx, $file, $type);
# $ctx - value corresponding to openssl's X509_LOOKUP structure
# $file - (string) file name
# $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSL
#                          if not FILETYPE_PEM then behaves as Net::SSLeay::X509
#
# returns: 1 on success, 0 on failure
```

- X509_load_cert_file

  Loads/adds X509 certificate from `$file` to X509_LOOKUP structure

```
my $rv = Net::SSLeay::X509_load_cert_file($ctx, $file, $type);
# $ctx - value corresponding to openssl's X509_LOOKUP structure
# $file - (string) file name
# $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSL
#
# returns: 1 on success, 0 on failure
```

- X509_load_crl_file

  Loads/adds X509 CRL from `$file` to X509_LOOKUP structure

```
my $rv = Net::SSLeay::X509_load_crl_file($ctx, $file, $type);
# $ctx - value corresponding to openssl's X509_LOOKUP structure
# $file - (string) file name
# $type - (integer) type - use constants &Net::SSLeay::FILETYPE_PEM or &Net::SSL
#
# returns: 1 on success, 0 on failure
```

- X509_policy_level_get0_node

  ??? (more info needed)

```
my $rv = Net::SSLeay::X509_policy_level_get0_node($level, $i);
# $level - value corresponding to openssl's X509_POLICY_LEVEL structure
# $i - (integer) index/position
#
# returns: value corresponding to openssl's X509_POLICY_NODE structure (0 on fai
```

- X509_policy_level_node_count

  ??? (more info needed)

```
my $rv = Net::SSLeay::X509_policy_level_node_count($level);
# $level - value corresponding to openssl's X509_POLICY_LEVEL structure
#
# returns: (integer) node count
```

- X509_policy_node_get0_parent

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_policy_node_get0_parent($node);
  # $node - value corresponding to openssl's X509_POLICY_NODE structure
  #
  # returns: value corresponding to openssl's X509_POLICY_NODE structure (0 on fai
  ```

- X509_policy_node_get0_policy

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_policy_node_get0_policy($node);
  # $node - value corresponding to openssl's X509_POLICY_NODE structure
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

- X509_policy_node_get0_qualifiers

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_policy_node_get0_qualifiers($node);
  # $node - value corresponding to openssl's X509_POLICY_NODE structure
  #
  # returns: value corresponding to openssl's STACK_OF(POLICYQUALINFO) structure (
  ```

- X509_policy_tree_free

  ??? (more info needed)

  ```
  Net::SSLeay::X509_policy_tree_free($tree);
  # $tree - value corresponding to openssl's X509_POLICY_TREE structure
  #
  # returns: no return value
  ```

- X509_policy_tree_get0_level

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_policy_tree_get0_level($tree, $i);
  # $tree - value corresponding to openssl's X509_POLICY_TREE structure
  # $i - (integer) level index
  #
  # returns: value corresponding to openssl's X509_POLICY_LEVEL structure (0 on fa
  ```

- X509_policy_tree_get0_policies

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_policy_tree_get0_policies($tree);
  # $tree - value corresponding to openssl's X509_POLICY_TREE structure
  #
  # returns: value corresponding to openssl's X509_POLICY_NODE structure (0 on fai
  ```

- X509_policy_tree_get0_user_policies

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_policy_tree_get0_user_policies($tree);
  # $tree - value corresponding to openssl's X509_POLICY_TREE structure
  #
  # returns: value corresponding to openssl's X509_POLICY_NODE structure (0 on fai
  ```

- X509_policy_tree_level_count

??? (more info needed)

```
 my $rv = Net::SSLeay::X509_policy_tree_level_count($tree);
 # $tree - value corresponding to openssl's X509_POLICY_TREE structure
 #
 # returns: (integer) count
```

- X509_verify_cert_error_string

  Returns a human readable error string for verification error `$n`.

  ```
   my $rv = Net::SSLeay::X509_verify_cert_error_string($n);
   # $n - (long) numeric error code
   #
   # returns: error string
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_STORE_CTX_get_error.html>

- P_X509_add_extensions

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Adds one or more X509 extensions to X509 object `$x`.

  ```
   my $rv = Net::SSLeay::P_X509_add_extensions($x, $ca_cert, $nid, $value);
   # $x - value corresponding to openssl's X509 structure
   # $ca_cert - value corresponding to openssl's X509 structure (issuer's cert - ne
   # $nid - NID identifying extension to be set
   # $value - extension value
   #
   # returns: 1 on success, 0 on failure
  ```

  You can set more extensions at once:

  ```
   my $rv = Net::SSLeay::P_X509_add_extensions($x509, $ca_cert,
                 &Net::SSLeay::NID_key_usage => 'digitalSignature,keyEncipherment'
                 &Net::SSLeay::NID_subject_key_identifier => 'hash',
                 &Net::SSLeay::NID_authority_key_identifier => 'keyid',
                 &Net::SSLeay::NID_authority_key_identifier => 'issuer',
                 &Net::SSLeay::NID_basic_constraints => 'CA:FALSE',
                 &Net::SSLeay::NID_ext_key_usage => 'serverAuth,clientAuth',
                 &Net::SSLeay::NID_netscape_cert_type => 'server',
                 &Net::SSLeay::NID_subject_alt_name => 'DNS:s1.dom.com,DNS:s2.dom.
           );
  ```

- P_X509_copy_extensions

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Copies X509 extensions from X509_REQ object to X509 object − handy when you need to turn X509_REQ into X509 certificate.

  ```
   Net::SSLeay::P_X509_copy_extensions($x509_req, $x509, $override);
   # $x509_req - value corresponding to openssl's X509_REQ structure
   # $x509 - value corresponding to openssl's X509 structure
   # $override - (integer) flag indication whether to override already existing ite
   #
   # returns: 1 on success, 0 on failure
  ```

- P_X509_get_crl_distribution_points

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Get the list of CRL distribution points from X509 certificate.

```
my @cdp = Net::SSLeay::P_X509_get_crl_distribution_points($x509);
# $x509 - value corresponding to openssl's X509 structure
#
# returns: list of distribution points (usually URLs)
```

- P_X509_get_ext_key_usage

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Gets the list of extended key usage of given X509 certificate `$cert`.

  ```
  my @ext_usage = Net::SSLeay::P_X509_get_ext_key_usage($cert, $format);
  # $cert - value corresponding to openssl's X509 structure
  # $format - choose type of return values: 0=OIDs, 1=NIDs, 2=shortnames, 3=longna
  #
  # returns: list of values
  ```

  Examples:

  ```
  my @extkeyusage_oid = Net::SSLeay::P_X509_get_ext_key_usage($x509,0);
  # returns for example: ("1.3.6.1.5.5.7.3.1", "1.3.6.1.5.5.7.3.2")

  my @extkeyusage_nid = Net::SSLeay::P_X509_get_ext_key_usage($x509,1);
  # returns for example: (129, 130)

  my @extkeyusage_sn  = Net::SSLeay::P_X509_get_ext_key_usage($x509,2);
  # returns for example: ("serverAuth", "clientAuth")

  my @extkeyusage_ln  = Net::SSLeay::P_X509_get_ext_key_usage($x509,3);
  # returns for example: ("TLS Web Server Authentication",  "TLS Web Client Authen
  ```

- P_X509_get_key_usage

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Gets the list of key usage of given X509 certificate `$cert`.

  ```
  my @keyusage = Net::SSLeay::P_X509_get_key_usage($cert);
  # $cert - value corresponding to openssl's X509 structure
  #
  # returns: list of key usage values which can be none, one or more from the foll
  #          "digitalSignature"
  #          "nonRepudiation"
  #          "keyEncipherment"
  #          "dataEncipherment"
  #          "keyAgreement"
  #          "keyCertSign"
  #          "cRLSign"
  #          "encipherOnly"
  #          "decipherOnly"
  ```

- P_X509_get_netscape_cert_type

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Gets the list of Netscape cert types of given X509 certificate `$cert`.

```
Net::SSLeay::P_X509_get_netscape_cert_type($cert);
# $cert - value corresponding to openssl's X509 structure
#
# returns: list of Netscape type values which can be none, one or more from the
#          "client"
#          "server"
#          "email"
#          "objsign"
#          "reserved"
#          "sslCA"
#          "emailCA"
#          "objCA"
```

- P_X509_get_pubkey_alg

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns ASN1_OBJECT corresponding to X509 certificate public key algorithm.

  ```
  my $rv = Net::SSLeay::P_X509_get_pubkey_alg($x);
  # $x - value corresponding to openssl's X509 structure
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

  To get textual representation use:

  ```
  my $alg = Net::SSLeay::OBJ_obj2txt(Net::SSLeay::P_X509_get_pubkey_alg($x509));
  # returns for example: "rsaEncryption"
  ```

- P_X509_get_signature_alg

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns ASN1_OBJECT corresponding to X509 signarite key algorithm.

  ```
  my $rv = Net::SSLeay::P_X509_get_signature_alg($x);
  # $x - value corresponding to openssl's X509 structure
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

  To get textual representation use:

  ```
  my $alg = Net::SSLeay::OBJ_obj2txt(Net::SSLeay::P_X509_get_signature_alg($x509))
  # returns for example: "sha1WithRSAEncryption"
  ```

- sk_X509_new_null

  Returns a new, empty, STACK_OF(X509) structure.

  ```
  my $rv = Net::SSLeay::sk_X509_new_null();
  #
  # returns: value corresponding to openssl's STACK_OF(X509) structure
  ```

- sk_X509_push

  Pushes an X509 structure onto a STACK_OF(X509) structure.

  ```
  my $rv = Net::SSLeay::sk_X509_push($sk_x509, $x509);
  # $sk_x509 - value corresponding to openssl's STACK_OF(X509) structure
  # $x509 - value corresponding to openssl's X509 structure
  #
  # returns: 1 if successful, 0 if unsuccessful
  ```

Low level API: X509_REQ_* related functions

- X509_REQ_new

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Creates a new X509_REQ structure.

  ```
  my $rv = Net::SSLeay::X509_REQ_new();
  #
  # returns: value corresponding to openssl's X509_REQ structure (0 on failure)
  ```

- X509_REQ_free

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Free an allocated X509_REQ structure.

  ```
  Net::SSLeay::X509_REQ_free($x);
  # $x - value corresponding to openssl's X509_REQ structure
  #
  # returns: no return value
  ```

- X509_REQ_add1_attr_by_NID

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Adds an attribute whose name is defined by a NID `$nid`. The field value to be added is in `$bytes`.

  ```
  my $rv = Net::SSLeay::X509_REQ_add1_attr_by_NID($req, $nid, $type, $bytes);
  # $req - value corresponding to openssl's X509_REQ structure
  # $nid - (integer) NID value
  # $type - (integer) type of data in $bytes (see below)
  # $bytes - data to be set
  #
  # returns: 1 on success, 0 on failure

  # values for $type - use constants:
  &Net::SSLeay::MBSTRING_UTF8     - $bytes contains utf8 encoded data
  &Net::SSLeay::MBSTRING_ASC      - $bytes contains ASCII data
  ```

- X509_REQ_digest

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Computes digest/fingerprint of X509_REQ `$data` using `$type` hash function.

  ```
  my $digest_value = Net::SSLeay::X509_REQ_digest($data, $type);
  # $data - value corresponding to openssl's X509_REQ structure
  # $type - value corresponding to openssl's EVP_MD structure - e.g. got via EVP_g
  #
  # returns: hash value (binary)

  #to get printable (hex) value of digest use:
  print unpack('H*', $digest_value);
  ```

- X509_REQ_get_attr_by_NID

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Retrieve the next index matching `$nid` after `$lastpos` ($lastpos should initially be set to −1).

```
my $rv = Net::SSLeay::X509_REQ_get_attr_by_NID($req, $nid, $lastpos=-1);
# $req - value corresponding to openssl's X509_REQ structure
# $nid - (integer) NID value
# $lastpos - [optional] (integer) index where to start search (default -1)
#
# returns: index (-1 if there are no more entries)
```

Note: use "P_X509_REQ_get_attr" to get the actual attribute value – e.g.

```
my $index = Net::SSLeay::X509_REQ_get_attr_by_NID($req, $nid);
my @attr_values = Net::SSLeay::P_X509_REQ_get_attr($req, $index);
```

- X509_REQ_get_attr_by_OBJ

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Retrieve the next index matching `$obj` after `$lastpos` ($lastpos should initially be set to −1).

  ```
  my $rv = Net::SSLeay::X509_REQ_get_attr_by_OBJ($req, $obj, $lastpos=-1);
  # $req - value corresponding to openssl's X509_REQ structure
  # $obj - value corresponding to openssl's ASN1_OBJECT structure
  # $lastpos - [optional] (integer) index where to start search (default -1)
  #
  # returns: index (-1 if there are no more entries)
  ```

  Note: use "P_X509_REQ_get_attr" to get the actual attribute value – e.g.

  ```
  my $index = Net::SSLeay::X509_REQ_get_attr_by_NID($req, $nid);
  my @attr_values = Net::SSLeay::P_X509_REQ_get_attr($req, $index);
  ```

- X509_REQ_get_attr_count

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns the total number of attributes in `$req`.

  ```
  my $rv = Net::SSLeay::X509_REQ_get_attr_count($req);
  # $req - value corresponding to openssl's X509_REQ structure
  #
  # returns: (integer) items count
  ```

- X509_REQ_get_pubkey

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns public key corresponding to given X509_REQ object `$x`.

  ```
  my $rv = Net::SSLeay::X509_REQ_get_pubkey($x);
  # $x - value corresponding to openssl's X509_REQ structure
  #
  # returns: value corresponding to openssl's EVP_PKEY structure (0 on failure)
  ```

- X509_REQ_get_subject_name

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns X509_NAME object corresponding to subject name of given X509_REQ object `$x`.

  ```
  my $rv = Net::SSLeay::X509_REQ_get_subject_name($x);
  # $x - value corresponding to openssl's X509_REQ structure
  #
  # returns: value corresponding to openssl's X509_NAME structure (0 on failure)
  ```

- X509_REQ_get_version

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

Returns 'version' value for given X509_REQ object `$x`.

```
my $rv = Net::SSLeay::X509_REQ_get_version($x);
# $x - value corresponding to openssl's X509_REQ structure
#
# returns: (integer) version e.g. 0 = "version 1"
```

- X509_REQ_set_pubkey

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets public key of given X509_REQ object `$x` to `$pkey`.

  ```
  my $rv = Net::SSLeay::X509_REQ_set_pubkey($x, $pkey);
  # $x - value corresponding to openssl's X509_REQ structure
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_REQ_set_subject_name

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets subject name of given X509_REQ object `$x` to X509_NAME object `$name`.

  ```
  my $rv = Net::SSLeay::X509_REQ_set_subject_name($x, $name);
  # $x - value corresponding to openssl's X509_REQ structure
  # $name - value corresponding to openssl's X509_NAME structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_REQ_set_version

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sets 'version' of given X509_REQ object `$x` to `$version`.

  ```
  my $rv = Net::SSLeay::X509_REQ_set_version($x, $version);
  # $x - value corresponding to openssl's X509_REQ structure
  # $version - (integer) e.g. 0 = "version 1"
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_REQ_sign

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sign X509_REQ object `$x` with private key `$pk` (using digest algorithm `$md`).

  ```
  my $rv = Net::SSLeay::X509_REQ_sign($x, $pk, $md);
  # $x - value corresponding to openssl's X509_REQ structure
  # $pk - value corresponding to openssl's EVP_PKEY structure (requestor's private
  # $md - value corresponding to openssl's EVP_MD structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_REQ_verify

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Verifies X509_REQ object `$x` using public key `$r` (pubkey of requesting party).

```
my $rv = Net::SSLeay::X509_REQ_verify($x, $r);
# $x - value corresponding to openssl's X509_REQ structure
# $r - value corresponding to openssl's EVP_PKEY structure
#
# returns: 0 - verify failure, 1 - verify OK, <0 - error
```

- P_X509_REQ_add_extensions

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Adds one or more X509 extensions to X509_REQ object $x.

  ```
  my $rv = Net::SSLeay::P_X509_REQ_add_extensions($x, $nid, $value);
  # $x - value corresponding to openssl's X509_REQ structure
  # $nid - NID identifying extension to be set
  # $value - extension value
  #
  # returns: 1 on success, 0 on failure
  ```

  You can set more extensions at once:

  ```
  my $rv = Net::SSLeay::P_X509_REQ_add_extensions($x509_req,
             &Net::SSLeay::NID_key_usage => 'digitalSignature,keyEncipherment',
             &Net::SSLeay::NID_basic_constraints => 'CA:FALSE',
             &Net::SSLeay::NID_ext_key_usage => 'serverAuth,clientAuth',
             &Net::SSLeay::NID_netscape_cert_type => 'server',
             &Net::SSLeay::NID_subject_alt_name => 'DNS:s1.com,DNS:s2.com',
             &Net::SSLeay::NID_crl_distribution_points => 'URI:http://pki.com/crl1
           );
  ```

- P_X509_REQ_get_attr

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Returns attribute value for X509_REQ's attribute at index $n.

  ```
  Net::SSLeay::P_X509_REQ_get_attr($req, $n);
  # $req - value corresponding to openssl's X509_REQ structure
  # $n - (integer) attribute index
  #
  # returns: value corresponding to openssl's ASN1_STRING structure
  ```

Low level API: X509_CRL_* related functions

- X509_CRL_new

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Creates a new X509_CRL structure.

  ```
  my $rv = Net::SSLeay::X509_CRL_new();
  #
  # returns: value corresponding to openssl's X509_CRL structure (0 on failure)
  ```

- X509_CRL_free

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Free an allocated X509_CRL structure.

  ```
  Net::SSLeay::X509_CRL_free($x);
  # $x - value corresponding to openssl's X509_CRL structure
  #
  # returns: no return value
  ```

- X509_CRL_digest

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Computes digest/fingerprint of X509_CRL `$data` using `$type` hash function.

  ```
  my $digest_value = Net::SSLeay::X509_CRL_digest($data, $type);
  # $data - value corresponding to openssl's X509_CRL structure
  # $type - value corresponding to openssl's EVP_MD structure - e.g. got via EVP_g
  #
  # returns: hash value (binary)
  ```

  Example:

  ```
  my $x509_crl
  my $md = Net::SSLeay::EVP_get_digestbyname("sha1");
  my $digest_value = Net::SSLeay::X509_CRL_digest($x509_crl, $md);
  #to get printable (hex) value of digest use:
  print "digest=", unpack('H*', $digest_value), "\n";
  ```

- X509_CRL_get_ext

  COMPATIBILITY: not available in Net−SSLeay−1.54 and before

  Returns X509_EXTENSION from `$x509` based on given position/index.

  ```
  my $rv = Net::SSLeay::X509_CRL_get_ext($x509, $index);
  # $x509 - value corresponding to openssl's X509_CRL structure
  # $index - (integer) position/index of extension within $x509
  #
  # returns: value corresponding to openssl's X509_EXTENSION structure (0 on failu
  ```

- X509_CRL_get_ext_by_NID

  COMPATIBILITY: not available in Net−SSLeay−1.54 and before

  Returns X509_EXTENSION from `$x509` based on given NID.

  ```
  my $rv = Net::SSLeay::X509_CRL_get_ext_by_NID($x509, $nid, $loc);
  # $x509 - value corresponding to openssl's X509_CRL structure
  # $nid - (integer) NID value
  # $loc - (integer) position to start lookup at
  #
  # returns: position/index of extension, negative value on error
  #          call Net::SSLeay::X509_CRL_get_ext($x509, $rv) to get the actual exte
  ```

- X509_CRL_get_ext_count

  COMPATIBILITY: not available in Net−SSLeay−1.54 and before

  Returns the total number of extensions in X509_CRL object `$x`.

  ```
  my $rv = Net::SSLeay::X509_CRL_get_ext_count($x);
  # $x - value corresponding to openssl's X509_CRL structure
  #
  # returns: count of extensions
  ```

- X509_CRL_get_issuer

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns X509_NAME object corresponding to the issuer of X509_CRL `$x`.

```
my $rv = Net::SSLeay::X509_CRL_get_issuer($x);
# $x - value corresponding to openssl's X509_CRL structure
#
# returns: value corresponding to openssl's X509_NAME structure (0 on failure)
```

See other `X509_NAME_*` functions to get more info from X509_NAME structure.

- X509_CRL_get_lastUpdate

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns 'lastUpdate' date-time value of X509_CRL object `$x`.

  ```
  my $rv = Net::SSLeay::X509_CRL_get_lastUpdate($x);
  # $x - value corresponding to openssl's X509_CRL structure
  #
  # returns: value corresponding to openssl's ASN1_TIME structure (0 on failure)
  ```

- X509_CRL_get_nextUpdate

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns 'nextUpdate' date-time value of X509_CRL object `$x`.

  ```
  my $rv = Net::SSLeay::X509_CRL_get_nextUpdate($x);
  # $x - value corresponding to openssl's X509_CRL structure
  #
  # returns: value corresponding to openssl's ASN1_TIME structure (0 on failure)
  ```

- X509_CRL_get_version

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns 'version' value of given X509_CRL structure `$x`.

  ```
  my $rv = Net::SSLeay::X509_CRL_get_version($x);
  # $x - value corresponding to openssl's X509_CRL structure
  #
  # returns: (integer) version
  ```

- X509_CRL_set_issuer_name

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Sets the issuer of X509_CRL object `$x` to X509_NAME object `$name`.

  ```
  my $rv = Net::SSLeay::X509_CRL_set_issuer_name($x, $name);
  # $x - value corresponding to openssl's X509_CRL structure
  # $name - value corresponding to openssl's X509_NAME structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_CRL_set_lastUpdate

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Sets 'lastUpdate' value of X509_CRL object `$x` to `$tm`.

  ```
  my $rv = Net::SSLeay::X509_CRL_set_lastUpdate($x, $tm);
  # $x - value corresponding to openssl's X509_CRL structure
  # $tm - value corresponding to openssl's ASN1_TIME structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_CRL_set_nextUpdate

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

Sets 'nextUpdate' value of X509_CRL object `$x` to `$tm`.

```
my $rv = Net::SSLeay::X509_CRL_set_nextUpdate($x, $tm);
# $x - value corresponding to openssl's X509_CRL structure
# $tm - value corresponding to openssl's ASN1_TIME structure
#
# returns: 1 on success, 0 on failure
```

- X509_CRL_set_version

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Sets 'version' value of given X509_CRL structure `$x` to `$version`.

  ```
  my $rv = Net::SSLeay::X509_CRL_set_version($x, $version);
  # $x - value corresponding to openssl's X509_CRL structure
  # $version - (integer) version number (1 = version 2 CRL)
  #
  # returns: 1 on success, 0 on failure
  ```

  Note that if you want to use any X509_CRL extension you need to set "version 2 CRL" − `Net::SSLeay::X509_CRL_set_version($x, 1)`.

- X509_CRL_sign

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Sign X509_CRL object `$x` with private key `$pkey` (using digest algorithm `$md`).

  ```
  my $rv = Net::SSLeay::X509_CRL_sign($x, $pkey, $md);
  # $x - value corresponding to openssl's X509_CRL structure
  # $pkey - value corresponding to openssl's EVP_PKEY structure
  # $md - value corresponding to openssl's EVP_MD structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_CRL_sort

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Sorts the data of X509_CRL object so it will be written in serial number order.

  ```
  my $rv = Net::SSLeay::X509_CRL_sort($x);
  # $x - value corresponding to openssl's X509_CRL structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_CRL_verify

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Verifies X509_CRL object `$a` using public key `$r` (pubkey of issuing CA).

  ```
  my $rv = Net::SSLeay::X509_CRL_verify($a, $r);
  # $a - value corresponding to openssl's X509_CRL structure
  # $r - value corresponding to openssl's EVP_PKEY structure
  #
  # returns: 0 - verify failure, 1 - verify OK, <0 - error
  ```

- P_X509_CRL_add_revoked_serial_hex

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Adds given serial number `$serial_hex` to X509_CRL object `$crl`.

```
Net::SSLeay::P_X509_CRL_add_revoked_serial_hex($crl, $serial_hex, $rev_time, $re
# $crl - value corresponding to openssl's X509_CRL structure
# $serial_hex - string (hexadecimal) representation of serial number
# $rev_time - (revocation time) value corresponding to openssl's ASN1_TIME struc
# $reason_code - [optional] (integer) reason code (see below) - default 0
# $comp_time - [optional] (compromise time) value corresponding to openssl's ASN
#
# returns: no return value

reason codes:
0 - unspecified
1 - keyCompromise
2 - CACompromise
3 - affiliationChanged
4 - superseded
5 - cessationOfOperation
6 - certificateHold
7 - removeFromCRL
```

- P_X509_CRL_get_serial

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Returns serial number of X509_CRL object.

```
my $rv = Net::SSLeay::P_X509_CRL_get_serial($crl);
# $crl - value corresponding to openssl's X509_CRL structure
#
# returns: value corresponding to openssl's ASN1_INTEGER structure (0 on failure
```

- P_X509_CRL_set_serial

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.7

  Sets serial number of X509_CRL object to `$crl_number`.

```
my $rv = Net::SSLeay::P_X509_CRL_set_serial($crl, $crl_number);
# $crl - value corresponding to openssl's X509_CRL structure
# $crl_number - value corresponding to openssl's ASN1_INTEGER structure
#
# returns: 1 on success, 0 on failure
```

Low level API: X509_EXTENSION_* related functions

- X509_EXTENSION_get_critical

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns 'critical' flag of given X509_EXTENSION object `$ex`.

```
my $rv = Net::SSLeay::X509_EXTENSION_get_critical($ex);
# $ex - value corresponding to openssl's X509_EXTENSION structure
#
# returns: (integer) 1 - critical, 0 - noncritical
```

- X509_EXTENSION_get_data

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns value (raw data) of X509_EXTENSION object `$ne`.

```
my $rv = Net::SSLeay::X509_EXTENSION_get_data($ne);
# $ne - value corresponding to openssl's X509_EXTENSION structure
#
# returns: value corresponding to openssl's ASN1_OCTET_STRING structure (0 on fa
```

Note: you can use "P_ASN1_STRING_get" to convert ASN1_OCTET_STRING into perl scalar variable.

- X509_EXTENSION_get_object

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns OID (ASN1_OBJECT) of X509_EXTENSION object `$ne`.

  ```
  my $rv = Net::SSLeay::X509_EXTENSION_get_object($ex);
  # $ex - value corresponding to openssl's X509_EXTENSION structure
  #
  # returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
  ```

- X509V3_EXT_print

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns string representation of given X509_EXTENSION object `$ext`.

  ```
  Net::SSLeay::X509V3_EXT_print($ext, $flags, $utf8_decode);
  # $ext - value corresponding to openssl's X509_EXTENSION structure
  # $flags - [optional] (integer) Currently the flag argument is unused and shoulc
  # $utf8_decode - [optional] 0 or 1 whether the returned value should be utf8 dec
  #
  # returns: no return value
  ```

- X509V3_EXT_d2i

  Parses an extension and returns its internal structure.

  ```
  my $rv = Net::SSLeay::X509V3_EXT_d2i($ext);
  # $ext - value corresponding to openssl's X509_EXTENSION structure
  #
  # returns: pointer ???
  ```

Low level API: X509_NAME_* related functions

- X509_NAME_ENTRY_get_data

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Retrieves the field value of `$ne` in and ASN1_STRING structure.

  ```
  my $rv = Net::SSLeay::X509_NAME_ENTRY_get_data($ne);
  # $ne - value corresponding to openssl's X509_NAME_ENTRY structure
  #
  # returns: value corresponding to openssl's ASN1_STRING structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_ENTRY_get_object.html>

- X509_NAME_ENTRY_get_object

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Retrieves the field name of `$ne` in and ASN1_OBJECT structure.

```
my $rv = Net::SSLeay::X509_NAME_ENTRY_get_object($ne);
# $ne - value corresponding to openssl's X509_NAME_ENTRY structure
#
# returns: value corresponding to openssl's ASN1_OBJECT structure (0 on failure)
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_ENTRY_get_object.html>

- X509_NAME_new

  COMPATIBILITY: not available in Net−SSLeay−1.55 and before; requires at least openssl−0.9.5

  Creates a new X509_NAME structure. Adds a field whose name is defined by a string `$field`. The field value to be added is in `$bytes`.

  ```
  my $rv = Net::SSLeay::X509_NAME_new();
  #
  # returns: value corresponding to openssl's X509_NAME structure (0 on failure)
  ```

- X509_NAME_hash

  COMPATIBILITY: not available in Net−SSLeay−1.55 and before; requires at least openssl−0.9.5

  Sort of a checksum of issuer name `$name`. The result is not a full hash (e.g. sha−1), it is kind-of-a-hash truncated to the size of 'unsigned long' (32 bits). The resulting value might differ across different openssl versions for the same X509 certificate.

  ```
  my $rv = Net::SSLeay::X509_NAME_hash($name);
  # $name - value corresponding to openssl's X509_NAME structure
  #
  # returns: number representing checksum
  ```

- X509_NAME_add_entry_by_txt

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.5

  Adds a field whose name is defined by a string `$field`. The field value to be added is in `$bytes`.

  ```
  my $rv = Net::SSLeay::X509_NAME_add_entry_by_txt($name, $field, $type, $bytes, $
  # $name - value corresponding to openssl's X509_NAME structure
  # $field - (string) field definition (name) - e.g. "organizationName"
  # $type - (integer) type of data in $bytes (see below)
  # $bytes - data to be set
  # $loc - [optional] (integer) index where the new entry is inserted: if it is -1
  # $set - [optional] (integer) determines how the new type is added. If it is 0 (
  #
  # returns: 1 on success, 0 on failure

  # values for $type - use constants:
  &Net::SSLeay::MBSTRING_UTF8     - $bytes contains utf8 encoded data
  &Net::SSLeay::MBSTRING_ASC      - $bytes contains ASCII data
  ```

  Unicode note: when passing non-ascii (unicode) string in `$bytes` do not forget to set `$flags = &Net::SSLeay::MBSTRING_UTF8` and encode the perl `$string` via `$bytes = encode('utf-8', $string)`.

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_add_entry_by_txt.html>

- X509_NAME_add_entry_by_NID

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.5

  Adds a field whose name is defined by a NID `$nid`. The field value to be added is in `$bytes`.

```
my $rv = Net::SSLeay::X509_NAME_add_entry_by_NID($name, $nid, $type, $bytes, $le
# $name - value corresponding to openssl's X509_NAME structure
# $nid - (integer) field definition - NID value
# $type - (integer) type of data in $bytes (see below)
# $bytes - data to be set
# $loc - [optional] (integer) index where the new entry is inserted: if it is -1
# $set - [optional] (integer) determines how the new type is added. If it is 0 (
#
# returns: 1 on success, 0 on failure
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_add_entry_by_txt.html>

- X509_NAME_add_entry_by_OBJ

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−0.9.5

  Adds a field whose name is defined by a object (OID) `$obj` . The field value to be added is in `$bytes`.

```
my $rv = Net::SSLeay::X509_NAME_add_entry_by_OBJ($name, $obj, $type, $bytes, $le
# $name - value corresponding to openssl's X509_NAME structure
# $obj - field definition - value corresponding to openssl's ASN1_OBJECT structu
# $type - (integer) type of data in $bytes (see below)
# $bytes - data to be set
# $loc - [optional] (integer) index where the new entry is inserted: if it is -1
# $set - [optional] (integer) determines how the new type is added. If it is 0 (
#
# returns: 1 on success, 0 on failure
```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_add_entry_by_txt.html>

- X509_NAME_cmp

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Compares two X509_NAME obejcts.

```
my $rv = Net::SSLeay::X509_NAME_cmp($a, $b);
# $a - value corresponding to openssl's X509_NAME structure
# $b - value corresponding to openssl's X509_NAME structure
#
# returns: 0 if $a matches $b; non zero otherwise
```

- X509_NAME_digest

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Computes digest/fingerprint of X509_NAME `$data` using `$type` hash function.

```
my $digest_value = Net::SSLeay::X509_NAME_digest($data, $type);
# $data - value corresponding to openssl's X509_NAME structure
# $type - value corresponding to openssl's EVP_MD structure - e.g. got via EVP_g
#
# returns: hash value (binary)

#to get printable (hex) value of digest use:
print unpack('H*', $digest_value);
```

- X509_NAME_entry_count

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns the total number of entries in `$name`.

```
my $rv = Net::SSLeay::X509_NAME_entry_count($name);
# $name - value corresponding to openssl's X509_NAME structure
#
# returns: (integer) entries count
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_get_index_by_NID.html>

- X509_NAME_get_entry

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Retrieves the X509_NAME_ENTRY from `$name` corresponding to index `$loc`. Acceptable values for `$loc` run from 0 to `Net::SSLeay::X509_NAME_entry_count($name)- 1`. The value returned is an internal pointer which must not be freed.

```
my $rv = Net::SSLeay::X509_NAME_get_entry($name, $loc);
# $name - value corresponding to openssl's X509_NAME structure
# $loc - (integer) index of wanted entry
#
# returns: value corresponding to openssl's X509_NAME_ENTRY structure (0 on fail
```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_get_index_by_NID.html>

- X509_NAME_print_ex

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns a string with human readable version of `$name`.

```
Net::SSLeay::X509_NAME_print_ex($name, $flags, $utf8_decode);
# $name - value corresponding to openssl's X509_NAME structure
# $flags - [optional] conversion flags (default XN_FLAG_RFC2253) - see below
# $utf8_decode - [optional] 0 or 1 whether the returned value should be utf8 dec
#
# returns: string representation of $name

#available conversion flags - use constants:
&Net::SSLeay::XN_FLAG_COMPAT
&Net::SSLeay::XN_FLAG_DN_REV
&Net::SSLeay::XN_FLAG_DUMP_UNKNOWN_FIELDS
&Net::SSLeay::XN_FLAG_FN_ALIGN
&Net::SSLeay::XN_FLAG_FN_LN
&Net::SSLeay::XN_FLAG_FN_MASK
&Net::SSLeay::XN_FLAG_FN_NONE
&Net::SSLeay::XN_FLAG_FN_OID
&Net::SSLeay::XN_FLAG_FN_SN
&Net::SSLeay::XN_FLAG_MULTILINE
&Net::SSLeay::XN_FLAG_ONELINE
&Net::SSLeay::XN_FLAG_RFC2253
&Net::SSLeay::XN_FLAG_SEP_COMMA_PLUS
&Net::SSLeay::XN_FLAG_SEP_CPLUS_SPC
&Net::SSLeay::XN_FLAG_SEP_MASK
&Net::SSLeay::XN_FLAG_SEP_MULTILINE
&Net::SSLeay::XN_FLAG_SEP_SPLUS_SPC
&Net::SSLeay::XN_FLAG_SPC_EQ
```

  Most likely you will be fine with default:

```
Net::SSLeay::X509_NAME_print_ex($name, &Net::SSLeay::XN_FLAG_RFC2253);
```

  Or you might want RFC2253−like output without utf8 chars escaping:

```
use Net::SSLeay qw/XN_FLAG_RFC2253 ASN1_STRFLGS_ESC_MSB/;
my $flag_rfc22536_utf8 = (XN_FLAG_RFC2253) & (~ ASN1_STRFLGS_ESC_MSB);
my $result = Net::SSLeay::X509_NAME_print_ex($name, $flag_rfc22536_utf8, 1);
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_print_ex.html>

- X509_NAME_get_text_by_NID

  Retrieves the text from the first entry in name which matches `$nid`, if no such entry exists −1 is returned.

  openssl note: this is a legacy function which has various limitations which makes it of minimal use in practice. It can only find the first matching entry and will copy the contents of the field verbatim: this can be highly confusing if the target is a multicharacter string type like a BMPString or a UTF8String.

  ```
  Net::SSLeay::X509_NAME_get_text_by_NID($name, $nid);
  # $name - value corresponding to openssl's X509_NAME structure
  # $nid - NID value (integer)
  #
  # returns: text value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_get_index_by_NID.html>

- X509_NAME_oneline

  Return an ASCII version of `$name`.

  ```
  Net::SSLeay::X509_NAME_oneline($name);
  # $name - value corresponding to openssl's X509_NAME structure
  #
  # returns: (string) ASCII version of $name
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_NAME_print_ex.html>

- sk_X509_NAME_free

  Free an allocated STACK_OF(X509_NAME) structure.

  ```
  Net::SSLeay::sk_X509_NAME_free($sk);
  # $sk - value corresponding to openssl's STACK_OF(X509_NAME) structure
  #
  # returns: no return value
  ```

- sk_X509_NAME_num

  Return number of items in STACK_OF(X509_NAME)

  ```
  my $rv = Net::SSLeay::sk_X509_NAME_num($sk);
  # $sk - value corresponding to openssl's STACK_OF(X509_NAME) structure
  #
  # returns: number of items
  ```

- sk_X509_NAME_value

  Returns X509_NAME from position `$index` in STACK_OF(X509_NAME)

  ```
  my $rv = Net::SSLeay::sk_X509_NAME_value($sk, $i);
  # $sk - value corresponding to openssl's STACK_OF(X509_NAME) structure
  # $i - (integer) index/position
  #
  # returns: value corresponding to openssl's X509_NAME structure (0 on failure)
  ```

- add_file_cert_subjects_to_stack

  Add a file of certs to a stack. All certs in `$file` that are not already in the `$stackCAs` will be added.

```
my $rv = Net::SSLeay::add_file_cert_subjects_to_stack($stackCAs, $file);
# $stackCAs - value corresponding to openssl's STACK_OF(X509_NAME) structure
# $file - (string) filename
#
# returns: 1 on success, 0 on failure
```

- add_dir_cert_subjects_to_stack

  Add a directory of certs to a stack. All certs in `$dir` that are not already in the `$stackCAs` will be added.

```
my $rv = Net::SSLeay::add_dir_cert_subjects_to_stack($stackCAs, $dir);
# $stackCAs - value corresponding to openssl's STACK_OF(X509_NAME) structure
# $dir - (string) the directory to append from. All files in this directory will
#
# returns: 1 on success, 0 on failure
```

Low level API: X509_STORE_* related functions

- X509_STORE_CTX_new

  returns a newly initialised X509_STORE_CTX structure.

- X509_STORE_CTX_init

  X509_STORE_CTX_init() sets up an X509_STORE_CTX for a subsequent verification operation. It must be called before each call to **X509_verify_cert()**.

  Net::SSLeay::X509_STORE_CTX_init($x509_store_ctx, `$x509_store`, `$x509`, `$chain`);

  # `$x509_store_ctx` – value corresponding to openssl's X509_STORE_CTX structure (required) # `$x509_store` – value corresponding to openssl's X509_STORE structure (optional) # `$x509` – value corresponding to openssl's X509 structure (optional) # `$chain` – value corresponding to openssl's STACK_OF(X509) structure (optional)

  Check openssl doc <https://www.openssl.org/docs/man1.0.2/crypto/X509_STORE_CTX_init.html>

- X509_STORE_CTX_free

  Frees an X509_STORE_CTX structure.

  Net::SSLeay::X509_STORE_CTX_free($x509_store_ctx);

  # `$x509_store_ctx` – value corresponding to openssl's X509_STORE_CTX structure

- X509_verify_cert

  The **X509_verify_cert()** function attempts to discover and validate a certificate chain based on parameters in ctx. A complete description of the process is contained in the **verify** (1) manual page.

  If this function returns 0, use X509_STORE_CTX_get_error to get additional error information.

  my `$rv` = Net::SSLeay::X509_verify_cert($x509_store_ctx); # `$x509_store_ctx` – value corresponding to openssl's X509_STORE_CTX structure # # returns: 1 if a complete chain can be built and validated, otherwise 0

  Check openssl doc <https://www.openssl.org/docs/manmaster/man3/X509_verify_cert.html>

- X509_STORE_CTX_get_current_cert

  Returns the certificate in ctx which caused the error or 0 if no certificate is relevant.

```
my $rv = Net::SSLeay::X509_STORE_CTX_get_current_cert($x509_store_ctx);
# $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
#
# returns: value corresponding to openssl's X509 structure (0 on failure)
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_STORE_CTX_get_error.html>

- X509_STORE_CTX_get_error

  Returns the error code of `$ctx`.

  ```
  my $rv = Net::SSLeay::X509_STORE_CTX_get_error($x509_store_ctx);
  # $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
  #
  # returns: (integer) error code
  ```

  For more info about erro code values check function "get_verify_result".

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_STORE_CTX_get_error.html>

- X509_STORE_CTX_get_error_depth

  Returns the depth of the error. This is a non-negative integer representing where in the certificate chain the error occurred. If it is zero it occurred in the end entity certificate, one if it is the certificate which signed the end entity certificate and so on.

  ```
  my $rv = Net::SSLeay::X509_STORE_CTX_get_error_depth($x509_store_ctx);
  # $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
  #
  # returns: (integer) depth
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_STORE_CTX_get_error.html>

- X509_STORE_CTX_get_ex_data

  Is used to retrieve the information for `$idx` from `$x509_store_ctx`.

  ```
  my $rv = Net::SSLeay::X509_STORE_CTX_get_ex_data($x509_store_ctx, $idx);
  # $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
  # $idx - (integer) index for application specific data
  #
  # returns: pointer to ???
  ```

- X509_STORE_CTX_set_ex_data

  Is used to store application data at arg for idx into `$x509_store_ctx`.

  ```
  my $rv = Net::SSLeay::X509_STORE_CTX_set_ex_data($x509_store_ctx, $idx, $data);
  # $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
  # $idx - (integer) ???
  # $data - (pointer) ???
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_STORE_CTX_set_cert

  Sets the certificate to be verified in `$x509_store_ctx` to `$x`.

  ```
  Net::SSLeay::X509_STORE_CTX_set_cert($x509_store_ctx, $x);
  # $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
  # $x - value corresponding to openssl's X509 structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_STORE_CTX_new.html>

- X509_STORE_new

  Returns a newly initialized X509_STORE structure.

  my `$rv` = **Net::SSLeay::X509_STORE_new**(); # # returns: value corresponding to openssl's

X509_STORE structure (0 on failure)

- X509_STORE_free

  Frees an X509_STORE structure

  Net::SSLeay::X509_STORE_free($x509_store); # $x509_store − value corresponding to openssl's X509_STORE structure

- X509_STORE_add_lookup

  Adds a lookup to an X509_STORE for a given lookup method.

  my $method = &Net::SSLeay::X509_LOOKUP_hash_dir; my $rv = Net::SSLeay::X509_STORE_add_lookup($x509_store, $method); # $method − value corresponding to openssl's X509_LOOKUP_METHOD structure # $x509_store − value corresponding to openssl's X509_STORE structure # # returns: value corresponding to openssl's X509_LOOKUP structure

  Check openssl doc <https://www.openssl.org/docs/man1.1.1/man3/X509_load_crl_file.html>

- X509_STORE_CTX_set_error

  Sets the error code of $ctx to $s. For example it might be used in a verification callback to set an error based on additional checks.

```
 Net::SSLeay::X509_STORE_CTX_set_error($x509_store_ctx, $s);
 # $x509_store_ctx - value corresponding to openssl's X509_STORE_CTX structure
 # $s - (integer) error id
 #
 # returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_STORE_CTX_get_error.html>

- X509_STORE_add_cert

  Adds X509 certificate $x into the X509_STORE $store.

```
 my $rv = Net::SSLeay::X509_STORE_add_cert($store, $x);
 # $store - value corresponding to openssl's X509_STORE structure
 # $x - value corresponding to openssl's X509 structure
 #
 # returns: 1 on success, 0 on failure
```

- X509_STORE_add_crl

  Adds X509 CRL $x into the X509_STORE $store.

```
 my $rv = Net::SSLeay::X509_STORE_add_crl($store, $x);
 # $store - value corresponding to openssl's X509_STORE structure
 # $x - value corresponding to openssl's X509_CRL structure
 #
 # returns: 1 on success, 0 on failure
```

- X509_STORE_set1_param

  ??? (more info needed)

```
 my $rv = Net::SSLeay::X509_STORE_set1_param($store, $pm);
 # $store - value corresponding to openssl's X509_STORE structure
 # $pm - value corresponding to openssl's X509_VERIFY_PARAM structure
 #
 # returns: 1 on success, 0 on failure
```

- X509_LOOKUP_hash_dir

Returns an X509_LOOKUP structure that instructs an X509_STORE to load files from a directory containing certificates with filenames in the format *hash.N* or crls with filenames in the format *hash.***r***N*

my `$rv` = **Net::SSLeay::X509_LOOKUP_hash_dir()**; # # returns: value corresponding to openssl's X509_LOOKUP_METHOD structure, with the hashed directory method

Check openssl doc <https://www.openssl.org/docs/man1.1.1/man3/X509_load_crl_file.html>

- X509_LOOKUP_add_dir

  Add a directory to an X509_LOOKUP structure, usually obtained from X509_STORE_add_lookup.

  my `$method` = &Net::SSLeay::X509_LOOKUP_hash_dir; my `$lookup` = Net::SSLeay::X509_STORE_add_lookup($x509_store, `$method`); my `$type` = &Net::SSLeay::X509_FILETYPE_PEM; Net::SSLeay::X509_LOOKUP_add_dir($lookup, `$dir`, `$type`); # `$lookup` – value corresponding to openssl's X509_LOOKUP structure # `$dir` – string path to a directory s# `$type` – constant corresponding to the type of file in the directory – can be X509_FILETYPE_PEM, X509_FILETYPE_DEFAULT, or X509_FILETYPE_ASN1

- X509_STORE_set_flags

```
 Net::SSLeay::X509_STORE_set_flags($ctx, $flags);
 # $ctx - value corresponding to openssl's X509_STORE structure
 # $flags - (unsigned long) flags to be set (bitmask)
 #
 # returns: no return value

 #to create $flags value use corresponding constants like
 $flags = Net::SSLeay::X509_V_FLAG_CRL_CHECK();
```

  For more details about `$flags` bitmask see "X509_VERIFY_PARAM_set_flags".

- X509_STORE_set_purpose

```
 Net::SSLeay::X509_STORE_set_purpose($ctx, $purpose);
 # $ctx - value corresponding to openssl's X509_STORE structure
 # $purpose - (integer) purpose identifier
 #
 # returns: no return value
```

  For more details about `$purpose` identifier check "CTX_set_purpose".

- X509_STORE_set_trust

```
 Net::SSLeay::X509_STORE_set_trust($ctx, $trust);
 # $ctx - value corresponding to openssl's X509_STORE structure
 # $trust - (integer) trust identifier
 #
 # returns: no return value
```

  For more details about `$trust` identifier check "CTX_set_trust".

Low Level API: X509_INFO related functions

- sk_X509_INFO_num

  Returns the number of values in a STACK_OF(X509_INFO) structure.

```
 my $rv = Net::SSLeay::sk_X509_INFO_num($sk_x509_info);
 # $sk_x509_info - value corresponding to openssl's STACK_OF(X509_INFO) structure
 #
 # returns: number of values in $sk_X509_info
```

- sk_X509_INFO_value

Returns the value of a STACK_OF(X509_INFO) structure at a given index.

```
my $rv = Net::SSLeay::sk_X509_INFO_value($sk_x509_info, $index);
# $sk_x509_info - value corresponding to openssl's STACK_OF(X509_INFO) structure
# $index - index into the stack
#
# returns: value corresponding to openssl's X509_INFO structure at the given inc
```

- P_X509_INFO_get_x509

  Returns the X509 structure stored in an X509_INFO structure.

  ```
  my $rv = Net::SSLeay::P_X509_INFO_get_x509($x509_info);
  # $x509_info - value corresponding to openssl's X509_INFO structure
  #
  # returns: value corresponding to openssl's X509 structure
  ```

Low level API: X509_VERIFY_PARAM_* related functions

- X509_VERIFY_PARAM_add0_policy

  Enables policy checking (it is disabled by default) and adds `$policy` to the acceptable policy set.

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_add0_policy($param, $policy);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  # $policy - value corresponding to openssl's ASN1_OBJECT structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_add0_table

  ??? (more info needed)

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_add0_table($param);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_VERIFY_PARAM_add1_host

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Adds an additional reference identifier that can match the peer's certificate.

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_add1_host($param, $name);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  # $name - (string) name to be set
  #
  # returns: 1 on success, 0 on failure
  ```

  See also OpenSSL docs, "X509_VERIFY_PARAM_set1_host" and "X509_VERIFY_PARAM_set_hostflags" for more information, including wildcard matching.

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_clear_flags

  Clears the flags `$flags` in param.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_clear_flags($param, $flags);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $flags - (unsigned long) flags to be set (bitmask)
#
# returns: 1 on success, 0 on failure
```

For more details about `$flags` bitmask see "X509_VERIFY_PARAM_set_flags".

Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_free

  Frees up the X509_VERIFY_PARAM structure.

```
Net::SSLeay::X509_VERIFY_PARAM_free($param);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
#
# returns: no return value
```

- X509_VERIFY_PARAM_get0_peername

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Returns the DNS hostname or subject CommonName from the peer certificate that matched one of the reference identifiers.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_get0_peername($param);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
#
# returns: (string) name e.g. '*.example.com' or undef
```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_get_depth

  Returns the current verification depth.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_get_depth($param);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
#
# returns: (ineger) depth
```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_get_flags

  Returns the current verification flags.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_get_flags($param);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
#
# returns: (unsigned long) flags to be set (bitmask)
```

  For more details about returned flags bitmask see "X509_VERIFY_PARAM_set_flags".

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set_flags

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_set_flags($param, $flags);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $flags - (unsigned long) flags to be set (bitmask)
#
# returns: 1 on success, 0 on failure
```

```
 #to create $flags value use corresponding constants like
 $flags = Net::SSLeay::X509_V_FLAG_CRL_CHECK();
```

For more details about `$flags` bitmask, see the OpenSSL docs below.

Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_inherit

  ??? (more info needed)

  ```
   my $rv = Net::SSLeay::X509_VERIFY_PARAM_inherit($to, $from);
   # $to - value corresponding to openssl's X509_VERIFY_PARAM structure
   # $from - value corresponding to openssl's X509_VERIFY_PARAM structure
   #
   # returns: 1 on success, 0 on failure
  ```

- X509_VERIFY_PARAM_lookup

  Finds X509_VERIFY_PARAM by name.

  ```
   my $rv = Net::SSLeay::X509_VERIFY_PARAM_lookup($name);
   # $name - (string) name we want to find
   #
   # returns: value corresponding to openssl's X509_VERIFY_PARAM structure (0 on fa
  ```

- X509_VERIFY_PARAM_new

  Creates a new X509_VERIFY_PARAM structure.

  ```
   my $rv = Net::SSLeay::X509_VERIFY_PARAM_new();
   #
   # returns: value corresponding to openssl's X509_VERIFY_PARAM structure (0 on fa
  ```

- X509_VERIFY_PARAM_set1

  Sets the name of X509_VERIFY_PARAM structure `$to` to the same value as the name of X509_VERIFY_PARAM structure `$from`.

  ```
   my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1($to, $from);
   # $to - value corresponding to openssl's X509_VERIFY_PARAM structure
   # $from - value corresponding to openssl's X509_VERIFY_PARAM structure
   #
   # returns: 1 on success, 0 on failure
  ```

- X509_VERIFY_PARAM_set1_email

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Sets the expected RFC822 email address to email.

  ```
   my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1_email($param, $email);
   # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
   # $email - (string) email to be set
   #
   # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set1_host

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Sets the expected DNS hostname to name clearing any previously specified host name or names.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1_host($param, $name);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $name - (string) name to be set
#
# returns: 1 on success, 0 on failure
```

See also OpenSSL docs, "X509_VERIFY_PARAM_add1_host" and "X509_VERIFY_PARAM_set_hostflags" for more information, including wildcard matching.

Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set1_ip

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Sets the expected IP address to ip.

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1_ip($param, $ip);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  # $ip - (binary) 4 octet IPv4 or 16 octet IPv6 address
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set1_ip_asc

  COMPATIBILITY: not available in Net−SSLeay−1.82 and before; requires at least OpenSSL 1.0.2

  Sets the expected IP address to ipasc.

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1_asc($param, $ipasc);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  # $ip - (string) IPv4 or IPv6 address
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set1_name

  Sets the name of X509_VERIFY_PARAM structure `$param` to `$name`.

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1_name($param, $name);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  # $name - (string) name to be set
  #
  # returns: 1 on success, 0 on failure
  ```

- X509_VERIFY_PARAM_set1_policies

  Enables policy checking (it is disabled by default) and sets the acceptable policy set to policies. Any existing policy set is cleared. The policies parameter can be 0 to clear an existing policy set.

  ```
  my $rv = Net::SSLeay::X509_VERIFY_PARAM_set1_policies($param, $policies);
  # $param - value corresponding to openssl's X509_VERIFY_PARAM structure
  # $policies - value corresponding to openssl's STACK_OF(ASN1_OBJECT) structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set_depth

  Sets the maximum verification depth to depth. That is the maximum number of untrusted CA certificates that can appear in a chain.

```
Net::SSLeay::X509_VERIFY_PARAM_set_depth($param, $depth);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $depth - (integer) depth to be set
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set_hostflags

```
Net::SSLeay::X509_VERIFY_PARAM_set_hostflags($param, $flags);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $flags - (unsigned int) flags to be set (bitmask)
#
# returns: no return value
```

See also OpenSSL docs, "X509_VERIFY_PARAM_add1_host" and "X509_VERIFY_PARAM_set1_host" for more information. The flags for controlling wildcard checks and other features are defined in OpenSSL docs.

Check openssl doc <https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set_purpose

Sets the verification purpose in `$param` to `$purpose`. This determines the acceptable purpose of the certificate chain, for example SSL client or SSL server.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_set_purpose($param, $purpose);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $purpose - (integer) purpose identifier
#
# returns: 1 on success, 0 on failure
```

For more details about `$purpose` identifier check "CTX_set_purpose".

Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set_time

Sets the verification time in `$param` to `$t`. Normally the current time is used.

```
Net::SSLeay::X509_VERIFY_PARAM_set_time($param, $t);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $t - (time_t) time in seconds since 1.1.1970
#
# returns: no return value
```

Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_set_trust

Sets the trust setting in `$param` to `$trust`.

```
my $rv = Net::SSLeay::X509_VERIFY_PARAM_set_trust($param, $trust);
# $param - value corresponding to openssl's X509_VERIFY_PARAM structure
# $trust - (integer) trust identifier
#
# returns: 1 on success, 0 on failure
```

For more details about `$trust` identifier check "CTX_set_trust".

Check openssl doc <http://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html>

- X509_VERIFY_PARAM_table_cleanup

  ??? (more info needed)

  ```
  Net::SSLeay::X509_VERIFY_PARAM_table_cleanup();
  #
  # returns: no return value
  ```

Low level API: Cipher (EVP_CIPHER_*) related functions

- EVP_get_cipherbyname

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before

  Returns an EVP_CIPHER structure when passed a cipher name.

  ```
  my $rv = Net::SSLeay::EVP_get_cipherbyname($name);
  # $name - (string) cipher name e.g. 'aes-128-cbc', 'camellia-256-ecb', 'des-ede'
  #
  # returns: value corresponding to openssl's EVP_CIPHER structure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/EVP_EncryptInit.html>

Low level API: Digest (EVP_MD_*) related functions

- OpenSSL_add_all_digests

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  ```
  Net::SSLeay::OpenSSL_add_all_digests();
  # no args, no return value
  ```

  http://www.openssl.org/docs/crypto/OpenSSL_add_all_algorithms.html

- P_EVP_MD_list_all

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−1.0.0

  NOTE: Does not exactly correspond to any low level API function

  ```
  my $rv = Net::SSLeay::P_EVP_MD_list_all();
  #
  # returns: arrayref - list of available digest names
  ```

  The returned digest names correspond to values expected by “EVP_get_digestbyname”.

  Note that some of the digests are available by default and some only after calling “OpenSSL_add_all_digests”.

- EVP_get_digestbyname

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  ```
  my $rv = Net::SSLeay::EVP_get_digestbyname($name);
  # $name - string with digest name
  #
  # returns: value corresponding to openssl's EVP_MD structure
  ```

  The `$name` param can be:

```
md2
md4
md5
mdc2
ripemd160
sha
sha1
sha224
sha256
sha512
whirlpool
```

Or better check the supported digests by calling "P_EVP_MD_list_all".

- EVP_MD_type

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

```
my $rv = Net::SSLeay::EVP_MD_type($md);
# $md - value corresponding to openssl's EVP_MD structure
#
# returns: the NID (integer) of the OBJECT IDENTIFIER representing the given mes
```

- EVP_MD_size

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

```
my $rv = Net::SSLeay::EVP_MD_size($md);
# $md - value corresponding to openssl's EVP_MD structure
#
# returns: the size of the message digest in bytes (e.g. 20 for SHA1)
```

- EVP_MD_CTX_md

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

```
Net::SSLeay::EVP_MD_CTX_md($ctx);
# $ctx - value corresponding to openssl's EVP_MD_CTX structure
#
# returns: value corresponding to openssl's EVP_MD structure
```

- EVP_MD_CTX_create

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Allocates, initializes and returns a digest context.

```
my $rv = Net::SSLeay::EVP_MD_CTX_create();
#
# returns: value corresponding to openssl's EVP_MD_CTX structure
```

  The complete idea behind EVP_MD_CTX looks like this example:

```
 Net::SSLeay::OpenSSL_add_all_digests();

 my $md = Net::SSLeay::EVP_get_digestbyname("sha1");
 my $ctx = Net::SSLeay::EVP_MD_CTX_create();
 Net::SSLeay::EVP_DigestInit($ctx, $md);

 while(my $chunk = get_piece_of_data()) {
   Net::SSLeay::EVP_DigestUpdate($ctx,$chunk);
 }
```

```
  my $result = Net::SSLeay::EVP_DigestFinal($ctx);
  Net::SSLeay::EVP_MD_CTX_destroy($ctx);

  print "digest=", unpack('H*', $result), "\n"; #print hex value
```

- EVP_DigestInit_ex

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Sets up digest context `$ctx` to use a digest `$type` from ENGINE `$impl`, `$ctx` must be initialized before calling this function, type will typically be supplied by a function such as "EVP_get_digestbyname". If `$impl` is 0 then the default implementation of digest `$type` is used.

```
 my $rv = Net::SSLeay::EVP_DigestInit_ex($ctx, $type, $impl);
 # $ctx  - value corresponding to openssl's EVP_MD_CTX structure
 # $type - value corresponding to openssl's EVP_MD structure
 # $impl - value corresponding to openssl's ENGINE structure
 #
 # returns: 1 for success and 0 for failure
```

- EVP_DigestInit

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Behaves in the same way as "EVP_DigestInit_ex" except the passed context `$ctx` does not have to be initialized, and it always uses the default digest implementation.

```
 my $rv = Net::SSLeay::EVP_DigestInit($ctx, $type);
 # $ctx - value corresponding to openssl's EVP_MD_CTX structure
 # $type - value corresponding to openssl's EVP_MD structure
 #
 # returns: 1 for success and 0 for failure
```

- EVP_MD_CTX_destroy

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Cleans up digest context `$ctx` and frees up the space allocated to it, it should be called only on a context created using "EVP_MD_CTX_create".

```
 Net::SSLeay::EVP_MD_CTX_destroy($ctx);
 # $ctx - value corresponding to openssl's EVP_MD_CTX structure
 #
 # returns: no return value
```

- EVP_DigestUpdate

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

```
 my $rv = Net::SSLeay::EVP_DigestUpdate($ctx, $data);
 # $ctx  - value corresponding to openssl's EVP_MD_CTX structure
 # $data - data to be hashed
 #
 # returns: 1 for success and 0 for failure
```

- EVP_DigestFinal_ex

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Retrieves the digest value from `$ctx`. After calling "EVP_DigestFinal_ex" no additional calls to "EVP_DigestUpdate" can be made, but "EVP_DigestInit_ex" can be called to initialize a new digest operation.

```
my $digest_value = Net::SSLeay::EVP_DigestFinal_ex($ctx);
# $ctx - value corresponding to openssl's EVP_MD_CTX structure
#
# returns: hash value (binary)

#to get printable (hex) value of digest use:
print unpack('H*', $digest_value);
```

- EVP_DigestFinal

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Similar to "EVP_DigestFinal_ex" except the digest context ctx is automatically cleaned up.

  ```
  my $rv = Net::SSLeay::EVP_DigestFinal($ctx);
  # $ctx - value corresponding to openssl's EVP_MD_CTX structure
  #
  # returns: hash value (binary)

  #to get printable (hex) value of digest use:
  print unpack('H*', $digest_value);
  ```

- MD2

  COMPATIBILITY: no supported by default in openssl−1.0.0

  Computes MD2 from given `$data` (all data needs to be loaded into memory)

  ```
  my $digest = Net::SSLeay::MD2($data);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- MD4

  Computes MD4 from given `$data` (all data needs to be loaded into memory)

  ```
  my $digest = Net::SSLeay::MD4($data);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- MD5

  Computes MD5 from given `$data` (all data needs to be loaded into memory)

  ```
  my $digest = Net::SSLeay::MD5($data);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- RIPEMD160

  Computes RIPEMD160 from given `$data` (all data needs to be loaded into memory)

  ```
  my $digest = Net::SSLeay::RIPEMD160($data);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- SHA1

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  Computes SHA1 from given `$data` (all data needs to be loaded into memory)

  ```
  my $digest = Net::SSLeay::SHA1($data);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- SHA256

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.8

  Computes SHA256 from given `$data` (all data needs to be loaded into memory)

```
my $digest = Net::SSLeay::SHA256($data);
print "digest(hexadecimal)=", unpack('H*', $digest);
```

- SHA512

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.8

  Computes SHA512 from given `$data` (all data needs to be loaded into memory)

  ```
  my $digest = Net::SSLeay::SHA512($data);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- EVP_Digest

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.7

  Computes "any" digest from given `$data` (all data needs to be loaded into memory)

  ```
  my $md = Net::SSLeay::EVP_get_digestbyname("sha1"); #or any other algorithm
  my $digest = Net::SSLeay::EVP_Digest($data, $md);
  print "digest(hexadecimal)=", unpack('H*', $digest);
  ```

- EVP_sha1

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  ```
  my $md = Net::SSLeay::EVP_sha1();
  #
  # returns: value corresponding to openssl's EVP_MD structure
  ```

- EVP_sha256

  COMPATIBILITY: requires at least openssl−0.9.8

  ```
  my $md = Net::SSLeay::EVP_sha256();
  #
  # returns: value corresponding to openssl's EVP_MD structure
  ```

- EVP_sha512

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before; requires at least openssl−0.9.8

  ```
  my $md = Net::SSLeay::EVP_sha512();
  #
  # returns: value corresponding to openssl's EVP_MD structure
  ```

- EVP_add_digest

  ```
  my $rv = Net::SSLeay::EVP_add_digest($digest);
  # $digest - value corresponding to openssl's EVP_MD structure
  #
  # returns: 1 on success, 0 otherwise
  ```

Low level API: CIPHER_* related functions

- CIPHER_get_name

  COMPATIBILITY: not available in Net−SSLeay−1.42 and before

  Returns name of the cipher used.

  ```
  my $rv = Net::SSLeay::CIPHER_description($cipher);
  # $cipher - value corresponding to openssl's SSL_CIPHER structure
  #
  # returns: (string) cipher name e.g. 'DHE-RSA-AES256-SHA'
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CIPHER_get_name.html>

Example:

```
my $ssl_cipher = Net::SSLeay::get_current_cipher($ssl);
my $cipher_name = Net::SSLeay::CIPHER_get_name($ssl_cipher);
```

- CIPHER_description

  Returns a textual description of the cipher used.

  ??? (does this function really work?)

```
my $rv = Net::SSLeay::CIPHER_description($cipher, $buf, $size);
# $cipher - value corresponding to openssl's SSL_CIPHER structure
# $bufer - (string/buffer) ???
# $size - (integer) ???
#
# returns: (string) cipher description e.g. 'DHE-RSA-AES256-SHA SSLv3 Kx=DH Au=R
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CIPHER_get_name.html>

- CIPHER_get_bits

  Returns the number of secret bits used for cipher.

```
my $rv = Net::SSLeay::CIPHER_get_bits($c);
# $c - value corresponding to openssl's SSL_CIPHER structure
#
# returns: (integert) number of secret bits, 0 on error
```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_CIPHER_get_name.html>

Low level API: RSA_* related functions

- RSA_generate_key

  Generates a key pair and returns it in a newly allocated RSA structure. The pseudo-random number generator must be seeded prior to calling RSA_generate_key.

```
my $rv = Net::SSLeay::RSA_generate_key($bits, $e, $perl_cb, $perl_cb_arg);
# $bits - (integer) modulus size in bits e.g. 512, 1024, 2048
# $e - (integer) public exponent, an odd number, typically 3, 17 or 65537
# $perl_cb - [optional] reference to perl callback function
# $perl_cb_arg - [optional] data that will be passed to callback function when i
#
# returns: value corresponding to openssl's RSA structure (0 on failure)
```

  Check openssl doc <http://www.openssl.org/docs/crypto/RSA_generate_key.html>

- RSA_free

  Frees the RSA structure and its components. The key is erased before the memory is returned to the system.

```
Net::SSLeay::RSA_free($r);
# $r - value corresponding to openssl's RSA structure
#
# returns: no return value
```

  Check openssl doc <http://www.openssl.org/docs/crypto/RSA_new.html>

- RSA_get_key_parameters

  Returns a list of pointers to BIGNUMs representing the parameters of the key in this order: (n, e, d, p, q, dmp1, dmq1, iqmp) Caution: returned list consists of SV pointers to BIGNUMs, which would need to be blessed as Crypt::OpenSSL::Bignum for further use my (@params) = RSA_get_key_parameters($r);

Low level API: BIO_* related functions

- BIO_eof

  Returns 1 if the BIO has read EOF, the precise meaning of 'EOF' varies according to the BIO type.

  ```
  my $rv = Net::SSLeay::BIO_eof($s);
  # $s - value corresponding to openssl's BIO structure
  #
  # returns: 1 if EOF has been reached 0 otherwise
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_ctrl.html>

- BIO_f_ssl

  Returns the SSL BIO method. This is a filter BIO which is a wrapper round the OpenSSL SSL routines adding a BIO 'flavour' to SSL I/O.

  ```
  my $rv = Net::SSLeay::BIO_f_ssl();
  #
  # returns: value corresponding to openssl's BIO_METHOD structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_f_ssl.html>

- BIO_free

  Frees up a single BIO.

  ```
  my $rv = Net::SSLeay::BIO_free($bio;);
  # $bio; - value corresponding to openssl's BIO structure
  #
  # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_new.html>

- BIO_new

  Returns a new BIO using method `$type`

  ```
  my $rv = Net::SSLeay::BIO_new($type);
  # $type - value corresponding to openssl's BIO_METHOD structure
  #
  # returns: value corresponding to openssl's BIO structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_new.html>

- BIO_new_buffer_ssl_connect

  Creates a new BIO chain consisting of a buffering BIO, an SSL BIO (using ctx) and a connect BIO.

  ```
  my $rv = Net::SSLeay::BIO_new_buffer_ssl_connect($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: value corresponding to openssl's BIO structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_f_ssl.html>

- BIO_new_file

  Creates a new file BIO with mode `$mode` the meaning of mode is the same as the stdio function **fopen**(). The BIO_CLOSE flag is set on the returned BIO.

```
my $rv = Net::SSLeay::BIO_new_file($filename, $mode);
# $filename - (string) filename
# $mode - (string) opening mode (as mode by stdio function fopen)
#
# returns: value corresponding to openssl's BIO structure (0 on failure)
```

Check openssl doc <http://www.openssl.org/docs/crypto/BIO_s_file.html>

- BIO_new_ssl

  Allocates an SSL BIO using SSL_CTX ctx and using client mode if client is non zero.

  ```
  my $rv = Net::SSLeay::BIO_new_ssl($ctx, $client);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $client - (integer) 0 or 1 - indicates ssl client mode
  #
  # returns: value corresponding to openssl's BIO structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_f_ssl.html>

- BIO_new_ssl_connect

  Creates a new BIO chain consisting of an SSL BIO (using ctx) followed by a connect BIO.

  ```
  my $rv = Net::SSLeay::BIO_new_ssl_connect($ctx);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  #
  # returns: value corresponding to openssl's BIO structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_f_ssl.html>

- BIO_pending

  Return the number of pending characters in the BIOs read buffers.

  ```
  my $rv = Net::SSLeay::BIO_pending($s);
  # $s - value corresponding to openssl's BIO structure
  #
  # returns: the amount of pending data
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_ctrl.html>

- BIO_wpending

  Return the number of pending characters in the BIOs write buffers.

  ```
  my $rv = Net::SSLeay::BIO_wpending($s);
  # $s - value corresponding to openssl's BIO structure
  #
  # returns: the amount of pending data
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_ctrl.html>

- BIO_read

  Read the underlying descriptor.

  ```
  Net::SSLeay::BIO_read($s, $max);
  # $s - value corresponding to openssl's BIO structure
  # $max - [optional] max. bytes to read (if not specified, the value 32768 is use
  #
  # returns: data
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_read.html>

- BIO_write

  Attempts to write data from `$buffer` to BIO `$b`.

  ```
   my $rv = Net::SSLeay::BIO_write($b, $buffer);
   # $b - value corresponding to openssl's BIO structure
   # $buffer - data
   #
   # returns: amount of data successfully written
   #          or that no data was successfully read or written if the result is 0 c
   #          or -2 when the operation is not implemented in the specific BIO type
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_read.html>

- BIO_s_mem

  Return the memory BIO method function.

  ```
   my $rv = Net::SSLeay::BIO_s_mem();
   #
   # returns: value corresponding to openssl's BIO_METHOD structure (0 on failure)
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_s_mem.html>

- BIO_ssl_copy_session_id

  Copies an SSL session id between BIO chains from and to. It does this by locating the SSL BIOs in each chain and calling **SSL_copy_session_id()** on the internal SSL pointer.

  ```
   my $rv = Net::SSLeay::BIO_ssl_copy_session_id($to, $from);
   # $to - value corresponding to openssl's BIO structure
   # $from - value corresponding to openssl's BIO structure
   #
   # returns: 1 on success, 0 on failure
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_f_ssl.html>

- BIO_ssl_shutdown

  Closes down an SSL connection on BIO chain bio. It does this by locating the SSL BIO in the chain and calling **SSL_shutdown()** on its internal SSL pointer.

  ```
   Net::SSLeay::BIO_ssl_shutdown($ssl_bio);
   # $ssl_bio - value corresponding to openssl's BIO structure
   #
   # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/BIO_f_ssl.html>

Low level API: Server side Server Name Indication (SNI) support

- set_tlsext_host_name

  TBA

- get_servername

  TBA

- get_servername_type

  TBA

- CTX_set_tlsext_servername_callback

  COMPATIBILITY: requires at least OpenSSL 0.9.8f

  This function is used in a server to support Server side Server Name Indication (SNI).

```
 Net::SSLeay::CTX_set_tlsext_servername_callback($ctx, $code)
 # $ctx - SSL context
 # $code - reference to a subroutine that will be called when a new connection is
 #
 # returns: no return value
On the client side:
use set_tlsext_host_name($ssl, $servername) before initiating the SSL connection.
```

On the server side: Set up an additional **SSL_CTX()** for each different certificate;

Add a servername callback to each **SSL_CTX()** using **CTX_set_tlsext_servername_callback()**;

The callback function is required to retrieve the client-supplied servername with get_servername(ssl). Figure out the right SSL_CTX to go with that host name, then switch the SSL object to that SSL_CTX with **set_SSL_CTX()**.

Example:

```
 # set callback
 Net::SSLeay::CTX_set_tlsext_servername_callback($ctx,
    sub {
      my $ssl = shift;
      my $h = Net::SSLeay::get_servername($ssl);
      Net::SSLeay::set_SSL_CTX($ssl, $hostnames{$h}->{ctx}) if exists $hostnames{
    } );
```

More complete example:

```
 # ... initialize Net::SSLeay

 my %hostnames = (
   'sni1' => { cert=>'sni1.pem', key=>'sni1.key' },
   'sni2' => { cert=>'sni2.pem', key=>'sni2.key' },
 );

 # create a new context for each certificate/key pair
 for my $name (keys %hostnames) {
   $hostnames{$name}->{ctx} = Net::SSLeay::CTX_new or die;
   Net::SSLeay::CTX_set_cipher_list($hostnames{$name}->{ctx}, 'ALL');
   Net::SSLeay::set_cert_and_key($hostnames{$name}->{ctx},
   $hostnames{$name}->{cert}, $hostnames{$name}->{key}) or die;
 }

 # create default context
 my $ctx = Net::SSLeay::CTX_new or die;
 Net::SSLeay::CTX_set_cipher_list($ctx, 'ALL');
 Net::SSLeay::set_cert_and_key($ctx, 'cert.pem','key.pem') or die;

 # set callback
 Net::SSLeay::CTX_set_tlsext_servername_callback($ctx, sub {
   my $ssl = shift;
   my $h = Net::SSLeay::get_servername($ssl);
   Net::SSLeay::set_SSL_CTX($ssl, $hostnames{$h}->{ctx}) if exists $hostnames{$h}
   } );

 # ... later

 $s = Net::SSLeay::new($ctx);
```

```
    Net::SSLeay::set_fd($s, fileno($accepted_socket));
    Net::SSLeay::accept($s);
```

Low level API: NPN (next protocol negotiation) related functions

NPN is being replaced with ALPN, a more recent TLS extension for application protocol negotiation that's in process of being adopted by IETF. Please look below for APLN API description.

Simple approach for using NPN support looks like this:

```
 ### client side
 use Net::SSLeay;
 use IO::Socket::INET;

 Net::SSLeay::initialize();
 my $sock = IO::Socket::INET->new(PeerAddr=>'encrypted.google.com:443') or die;
 my $ctx = Net::SSLeay::CTX_tlsv1_new() or die;
 Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL);
 Net::SSLeay::CTX_set_next_proto_select_cb($ctx, ['http1.1','spdy/2']);
 my $ssl = Net::SSLeay::new($ctx) or die;
 Net::SSLeay::set_fd($ssl, fileno($sock)) or die;
 Net::SSLeay::connect($ssl);

 warn "client:negotiated=",Net::SSLeay::P_next_proto_negotiated($ssl), "\n";
 warn "client:last_status=", Net::SSLeay::P_next_proto_last_status($ssl), "\n";

 ### server side
 use Net::SSLeay;
 use IO::Socket::INET;

 Net::SSLeay::initialize();
 my $ctx = Net::SSLeay::CTX_tlsv1_new() or die;
 Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL);
 Net::SSLeay::set_cert_and_key($ctx, "cert.pem", "key.pem");
 Net::SSLeay::CTX_set_next_protos_advertised_cb($ctx, ['spdy/2','http1.1']);
 my $sock = IO::Socket::INET->new(LocalAddr=>'localhost', LocalPort=>5443, Proto=>'t

 while (1) {
   my $ssl = Net::SSLeay::new($ctx);
   warn("server:waiting for incoming connection...\n");
   my $fd = $sock->accept();
   Net::SSLeay::set_fd($ssl, $fd->fileno);
   Net::SSLeay::accept($ssl);
   warn "server:negotiated=",Net::SSLeay::P_next_proto_negotiated($ssl),"\n";
   my $got = Net::SSLeay::read($ssl);
   Net::SSLeay::ssl_write_all($ssl, "length=".length($got));
   Net::SSLeay::free($ssl);
   $fd->close();
 }
 # check with: openssl s_client -connect localhost:5443 -nextprotoneg http/1.1,spdy/
```

Please note that the selection (negotiation) is performed by client side, the server side simply advertise the list of supported protocols.

Advanced approach allows you to implement your own negotiation algorithm.

```
#see below documentation for:
Net::SSleay::CTX_set_next_proto_select_cb($ctx, $perl_callback_function, $callback_
Net::SSleay::CTX_set_next_protos_advertised_cb($ctx, $perl_callback_function, $call
```

Detection of NPN support (works even in older Net::SSLeay versions):

```
use Net::SSLeay;

if (exists &Net::SSLeay::P_next_proto_negotiated) {
  # do NPN stuff
}
```

- CTX_set_next_proto_select_cb

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−1.0.1

  NOTE: You need CTX_set_next_proto_select_cb on **client side** of SSL connection.

  Simple usage − in this case a ‘‘common’’ negotiation algorithm (as implemented by openssl’s function SSL_select_next_proto) is used.

  ```
  $rv = Net::SSleay::CTX_set_next_proto_select_cb($ctx, $arrayref);
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $arrayref - list of accepted protocols - e.g. ['http1.0', 'http1.1']
  #
  # returns: 0 on success, 1 on failure
  ```

  Advanced usage (you probably do not need this):

  ```
  $rv = Net::SSleay::CTX_set_next_proto_select_cb($ctx, $perl_callback_function, $
  # $ctx - value corresponding to openssl's SSL_CTX structure
  # $perl_callback_function - reference to perl function
  # $callback_data - [optional] data to passed to callback function when invoked
  #
  # returns: 0 on success, 1 on failure

  # where callback function looks like
  sub npn_advertised_cb_invoke {
    my ($ssl, $arrayref_proto_list_advertised_by_server, $callback_data) = @_;
    my $status;
    # ...
    $status = 1;   #status can be:
                   # 0 - OPENSSL_NPN_UNSUPPORTED
                   # 1 - OPENSSL_NPN_NEGOTIATED
                   # 2 - OPENSSL_NPN_NO_OVERLAP
    return $status, ['http1.1','spdy/2']; # the callback has to return 2 values
  }
  ```

  To undefine/clear this callback use:

  ```
  Net::SSleay::CTX_set_next_proto_select_cb($ctx, undef);
  ```

- CTX_set_next_protos_advertised_cb

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−1.0.1

  NOTE: You need CTX_set_next_proto_select_cb on **server side** of SSL connection.

  Simple usage:

```
$rv = Net::SSleay::CTX_set_next_protos_advertised_cb($ctx, $arrayref);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $arrayref - list of advertised protocols - e.g. ['http1.0', 'http1.1']
#
# returns: 0 on success, 1 on failure
```

Advanced usage (you probably do not need this):

```
$rv = Net::SSleay::CTX_set_next_protos_advertised_cb($ctx, $perl_callback_functi
# $ctx - value corresponding to openssl's SSL_CTX structure
# $perl_callback_function - reference to perl function
# $callback_data - [optional] data to passed to callback function when invoked
#
# returns: 0 on success, 1 on failure

# where callback function looks like
sub npn_advertised_cb_invoke {
  my ($ssl, $callback_data) = @_;
  # ...
  return ['http1.1','spdy/2']; # the callback has to return arrayref
}
```

To undefine/clear this callback use:

```
Net::SSleay::CTX_set_next_protos_advertised_cb($ctx, undef);
```

- P_next_proto_negotiated

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−1.0.1

  Returns the name of negotiated protocol for given SSL connection `$ssl`.

  ```
  $rv = Net::SSLeay::P_next_proto_negotiated($ssl)
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: (string) negotiated protocol name (or undef if no negotiation was dor
  ```

- P_next_proto_last_status

  COMPATIBILITY: not available in Net−SSLeay−1.45 and before; requires at least openssl−1.0.1

  Returns the result of the last negotiation for given SSL connection `$ssl`.

  ```
  $rv = Net::SSLeay::P_next_proto_last_status($ssl)
  # $ssl - value corresponding to openssl's SSL structure
  #
  # returns: (integer) negotiation status
  #          0 - OPENSSL_NPN_UNSUPPORTED
  #          1 - OPENSSL_NPN_NEGOTIATED
  #          2 - OPENSSL_NPN_NO_OVERLAP
  ```

Low level API: ALPN (application layer protocol negotiation) related functions

Application protocol can be negotiated via two different mechanisms employing two different TLS extensions: NPN (obsolete) and ALPN (recommended).

The API is rather similar, with slight differences reflecting protocol specifics. In particular, with ALPN the protocol negotiation takes place on server, while with NPN the client implements the protocol negotiation logic.

With ALPN, the most basic implementation looks like this:

```
### client side
use Net::SSLeay;
use IO::Socket::INET;

Net::SSLeay::initialize();
my $sock = IO::Socket::INET->new(PeerAddr=>'encrypted.google.com:443') or die;
my $ctx = Net::SSLeay::CTX_tlsv1_new() or die;
Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL);
Net::SSLeay::CTX_set_alpn_protos($ctx, ['http/1.1', 'http/2.0', 'spdy/3]);
my $ssl = Net::SSLeay::new($ctx) or die;
Net::SSLeay::set_fd($ssl, fileno($sock)) or die;
Net::SSLeay::connect($ssl);

warn "client:selected=",Net::SSLeay::P_alpn_selected($ssl), "\n";

### server side
use Net::SSLeay;
use IO::Socket::INET;

Net::SSLeay::initialize();
my $ctx = Net::SSLeay::CTX_tlsv1_new() or die;
Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL);
Net::SSLeay::set_cert_and_key($ctx, "cert.pem", "key.pem");
Net::SSLeay::CTX_set_alpn_select_cb($ctx, ['http/1.1', 'http/2.0', 'spdy/3]);
my $sock = IO::Socket::INET->new(LocalAddr=>'localhost', LocalPort=>5443, Proto=>'t

while (1) {
  my $ssl = Net::SSLeay::new($ctx);
  warn("server:waiting for incoming connection...\n");
  my $fd = $sock->accept();
  Net::SSLeay::set_fd($ssl, $fd->fileno);
  Net::SSLeay::accept($ssl);
  warn "server:selected=",Net::SSLeay::P_alpn_selected($ssl),"\n";
  my $got = Net::SSLeay::read($ssl);
  Net::SSLeay::ssl_write_all($ssl, "length=".length($got));
  Net::SSLeay::free($ssl);
  $fd->close();
}
# check with: openssl s_client -connect localhost:5443 -alpn spdy/3,http/1.1
```

Advanced approach allows you to implement your own negotiation algorithm.

```
#see below documentation for:
Net::SSleay::CTX_set_alpn_select_cb($ctx, $perl_callback_function, $callback_data);
```

Detection of ALPN support (works even in older Net::SSLeay versions):

```
use Net::SSLeay;

if (exists &Net::SSLeay::P_alpn_selected) {
  # do ALPN stuff
}
```

- CTX_set_alpn_select_cb

  COMPATIBILITY: not available in Net−SSLeay−1.55 and before; requires at least openssl−1.0.2

  NOTE: You need CTX_set_alpn_select_cb on **server side** of TLS connection.

Simple usage – in this case a "common" negotiation algorithm (as implemented by openssl's function SSL_select_next_proto) is used.

```
$rv = Net::SSleay::CTX_set_alpn_select_cb($ctx, $arrayref);
# $ctx - value corresponding to openssl's SSL_CTX structure
# $arrayref - list of accepted protocols - e.g. ['http/2.0', 'http/1.1', 'spdy/3
#
# returns: 0 on success, 1 on failure
```

Advanced usage (you probably do not need this):

```
$rv = Net::SSleay::CTX_set_alpn_select_cb($ctx, $perl_callback_function, $callba
# $ctx - value corresponding to openssl's SSL_CTX structure
# $perl_callback_function - reference to perl function
# $callback_data - [optional] data to passed to callback function when invoked
#
# returns: 0 on success, 1 on failure

# where callback function looks like
sub alpn_select_cb_invoke {
  my ($ssl, $arrayref_proto_list_advertised_by_client, $callback_data) = @_;
  # ...
  if ($negotiated) {
    return 'http/2.0';
  } else {
    return undef;
  }
}
```

To undefine/clear this callback use:

```
Net::SSleay::CTX_set_alpn_select_cb($ctx, undef);
```

- set_alpn_protos

  COMPATIBILITY: not available in Net–SSLeay–1.55 and before; requires at least openssl–1.0.2

  NOTE: You need set_alpn_protos on **client side** of TLS connection.

  This adds list of supported application layer protocols to ClientHello message sent by a client. It advertises the enumeration of supported protocols:

  ```
  Net::SSLeay::set_alpn_protos($ssl, ['http/1.1', 'http/2.0', 'spdy/3]);
  # returns 0 on success
  ```

- CTX_set_alpn_protos

  COMPATIBILITY: not available in Net–SSLeay–1.55 and before; requires at least openssl–1.0.2

  NOTE: You need CTX_set_alpn_protos on **client side** of TLS connection.

  This adds list of supported application layer protocols to ClientHello message sent by a client. It advertises the enumeration of supported protocols:

  ```
  Net::SSLeay::CTX_set_alpn_protos($ctx, ['http/1.1', 'http/2.0', 'spdy/3]);
  # returns 0 on success
  ```

- P_alpn_selected

  COMPATIBILITY: not available in Net–SSLeay–1.55 and before; requires at least openssl–1.0.2

  Returns the name of negotiated protocol for given TLS connection `$ssl`.

```
$rv = Net::SSLeay::P_alpn_selected($ssl)
# $ssl - value corresponding to openssl's SSL structure
#
# returns: (string) negotiated protocol name (or undef if no negotiation was dor
```

Low level API: DANE Support

OpenSSL version 1.0.2 adds preliminary support RFC6698 Domain Authentication of Named Entities (DANE) Transport Layer Association within OpenSSL

- SSL_get_tlsa_record_byname

  COMPATIBILITY: DELETED from net-ssleay, since it is not supported by OpenSSL

  In order to facilitate DANE there is additional interface, SSL_get_tlsa_record_byname, accepting hostname, port and socket type that returns packed TLSA record. In order to make it even easier there is additional SSL_ctrl function that calls SSL_get_tlsa_record_byname for you. Latter is recommended for programmers that wish to maintain broader binary compatibility, e.g. make application work with both 1.0.2 and prior version (in which case call to SSL_ctrl with new code returning error would have to be ignored when running with prior version).

  Net::SSLeay::get_tlsa_record_byname($name, `$port`, `$type`);

Low level API: Other functions

- COMP_add_compression_method

  Adds the compression method cm with the identifier id to the list of available compression methods. This list is globally maintained for all SSL operations within this application. It cannot be set for specific SSL_CTX or SSL objects.

  ```
  my $rv = Net::SSLeay::COMP_add_compression_method($id, $cm);
  # $id - (integer) compression method id
  #       0 to 63:    methods defined by the IETF
  #       64 to 192:  external party methods assigned by IANA
  #       193 to 255: reserved for private use
  #
  # $cm - value corresponding to openssl's COMP_METHOD structure
  #
  # returns: 0 on success, 1 on failure (check the error queue to find out the rea
  ```

  Check openssl doc <http://www.openssl.org/docs/ssl/SSL_COMP_add_compression_method.html>

- DH_free

  Frees the DH structure and its components. The values are erased before the memory is returned to the system.

  ```
  Net::SSLeay::DH_free($dh);
  # $dh - value corresponding to openssl's DH structure
  #
  # returns: no return value
  ```

  Check openssl doc <http://www.openssl.org/docs/crypto/DH_new.html>

- FIPS_mode_set

  Enable or disable FIPS mode in a FIPS capable OpenSSL.

  ```
  Net::SSLeay:: FIPS_mode_set($enable);
  # $enable - (integer) 1 to enable, 0 to disable
  ```

Low level API: EC related functions

- CTX_set_tmp_ecdh

  TBA

- EC_KEY_free

  TBA

- EC_KEY_new_by_curve_name

  TBA

- EC_KEY_generate_key

  Generates a EC key and returns it in a newly allocated EC_KEY structure. The EC key then can be used to create a PKEY which can be used in calls like X509_set_pubkey.

```
 my $key = Net::SSLeay::EVP_PKEY_new();
 my $ec  = Net::SSLeay::EC_KEY_generate_key($curve);
 Net::SSLeay::EVP_PKEY_assign_EC_KEY($key,$ec);

 # $curve - curve name like 'secp521r1' or the matching Id (integer) of the curve
 #
 # returns: value corresponding to openssl's EC_KEY structure (0 on failure)
```

  This function has no equivalent in OpenSSL but combines multiple OpenSSL functions for an easier interface.

- CTX_set_ecdh_auto, set_ecdh_auto

  These functions enable or disable the automatic curve selection on the server side by calling SSL_CTX_set_ecdh_auto or SSL_set_ecdh_auto respectively. If enabled the highest preference curve is automatically used for ECDH temporary keys used during key exchange. This function is no longer available for OpenSSL 1.1.0 or higher.

```
  Net::SSLeay::CTX_set_ecdh_auto($ctx,1);
  Net::SSLeay::set_ecdh_auto($ssl,1);
```

- CTX_set1_curves_list, set1_curves_list

  These functions set the supported curves (in order of preference) by calling SSL_CTX_set1_curves_list or SSL_set1_curves_list respectively. For a TLS client these curves are offered to the server in the supported curves extension while on the server side these are used to determine the shared curve. These functions are only available since OpenSSL 1.1.0.

```
  Net::SSLeay::CTX_set1_curves_list($ctx,"P-521:P-384:P-256");
  Net::SSLeay::set1_curves_list($ssl,"P-521:P-384:P-256");
```

- CTX_set1_groups_list, set1_groups_list

  These functions set the supported groups (in order of preference) by calling SSL_CTX_set1_groups_list or SSL_set1_groups_list respectively. This is practically the same as CTX_set1_curves_list and set1_curves_list except that all DH groups can be given as supported by TLS 1.3. These functions are only available since OpenSSL 1.1.1.

```
  Net::SSLeay::CTX_set1_groups_list($ctx,"P-521:P-384:P-256");
  Net::SSLeay::set1_groups_list($ssl,"P-521:P-384:P-256");
```

Constants

There are many openssl constants available in Net::SSLeay. You can use them like this:

```
use Net::SSLeay;
print &Net::SSLeay::NID_commonName;
#or
print Net::SSLeay::NID_commonName();
```

Or you can import them and use:

```
use Net::SSLeay qw/NID_commonName/;
print &NID_commonName;
#or
print NID_commonName();
#or
print NID_commonName;
```

The constants names are derived from openssl constants, however constants starting with `SSL_` prefix have name with `SSL_` part stripped − e.g. openssl's constant `SSL_OP_ALL` is available as `Net::SSleay::OP_ALL`

The list of all available constant names:

```
ASN1_STRFLGS_ESC_CTRL           NID_netscape                              R_UNKNOWN
ASN1_STRFLGS_ESC_MSB            NID_netscape_base_url                     R_UNKNOWN
ASN1_STRFLGS_ESC_QUOTE          NID_netscape_ca_policy_url                R_X509_LI
ASN1_STRFLGS_RFC2253            NID_netscape_ca_revocation_url            SENT_SHUT
CB_ACCEPT_EXIT                  NID_netscape_cert_extension               SESSION_A
CB_ACCEPT_LOOP                  NID_netscape_cert_sequence                SESS_CACH
CB_ALERT                        NID_netscape_cert_type                    SESS_CACH
CB_CONNECT_EXIT                 NID_netscape_comment                      SESS_CACH
CB_CONNECT_LOOP                 NID_netscape_data_type                    SESS_CACH
CB_EXIT                         NID_netscape_renewal_url                  SESS_CACH
CB_HANDSHAKE_DONE               NID_netscape_revocation_url               SESS_CACH
CB_HANDSHAKE_START              NID_netscape_ssl_server_name              SESS_CACH
CB_LOOP                         NID_ns_sgc                                SESS_CACH
CB_READ                         NID_organizationName                      SSL3_VERS
CB_READ_ALERT                   NID_organizationalUnitName                SSLEAY_BU
CB_WRITE                        NID_pbeWithMD2AndDES_CBC                  SSLEAY_CF
CB_WRITE_ALERT                  NID_pbeWithMD2AndRC2_CBC                  SSLEAY_DI
ERROR_NONE                      NID_pbeWithMD5AndCast5_CBC                SSLEAY_PL
ERROR_SSL                       NID_pbeWithMD5AndDES_CBC                  SSLEAY_VE
ERROR_SYSCALL                   NID_pbeWithMD5AndRC2_CBC                  ST_ACCEPT
ERROR_WANT_ACCEPT               NID_pbeWithSHA1AndDES_CBC                 ST_BEFORE
ERROR_WANT_CONNECT              NID_pbeWithSHA1AndRC2_CBC                 ST_CONNEC
ERROR_WANT_READ                 NID_pbe_WithSHA1And128BitRC2_CBC          ST_INIT
ERROR_WANT_WRITE                NID_pbe_WithSHA1And128BitRC4              ST_OK
ERROR_WANT_X509_LOOKUP          NID_pbe_WithSHA1And2_Key_TripleDES_CBC    ST_READ_B
ERROR_ZERO_RETURN               NID_pbe_WithSHA1And3_Key_TripleDES_CBC    ST_READ_H
EVP_PKS_DSA                     NID_pbe_WithSHA1And40BitRC2_CBC           TLS1_1_VE
EVP_PKS_EC                      NID_pbe_WithSHA1And40BitRC4               TLS1_2_VE
EVP_PKS_RSA                     NID_pbes2                                 TLS1_3_VE
EVP_PKT_ENC                     NID_pbmac1                                TLS1_VERS
EVP_PKT_EXCH                    NID_pkcs                                  TLSEXT_ST
EVP_PKT_EXP                     NID_pkcs3                                 VERIFY_CL
EVP_PKT_SIGN                    NID_pkcs7                                 VERIFY_FA
EVP_PK_DH                       NID_pkcs7_data                            VERIFY_NO
EVP_PK_DSA                      NID_pkcs7_digest                          VERIFY_PE
EVP_PK_EC                       NID_pkcs7_encrypted                       VERIFY_PO
EVP_PK_RSA                      NID_pkcs7_enveloped                       V_OCSP_CE
```

```
FILETYPE_ASN1                     NID_pkcs7_signed                  V_OCSP_CE
FILETYPE_PEM                      NID_pkcs7_signedAndEnveloped      V_OCSP_CE
F_CLIENT_CERTIFICATE              NID_pkcs8ShroudedKeyBag           WRITING
F_CLIENT_HELLO                    NID_pkcs9                         X509_CHEC
F_CLIENT_MASTER_KEY               NID_pkcs9_challengePassword       X509_CHEC
F_D2I_SSL_SESSION                 NID_pkcs9_contentType             X509_CHEC
F_GET_CLIENT_FINISHED             NID_pkcs9_countersignature        X509_CHEC
F_GET_CLIENT_HELLO                NID_pkcs9_emailAddress            X509_CHEC
F_GET_CLIENT_MASTER_KEY           NID_pkcs9_extCertAttributes       X509_CHEC
F_GET_SERVER_FINISHED             NID_pkcs9_messageDigest           X509_FILE
F_GET_SERVER_HELLO                NID_pkcs9_signingTime             X509_FILE
F_GET_SERVER_VERIFY               NID_pkcs9_unstructuredAddress     X509_FILE
F_I2D_SSL_SESSION                 NID_pkcs9_unstructuredName        X509_LOOK
F_READ_N                          NID_private_key_usage_period      X509_PURP
F_REQUEST_CERTIFICATE             NID_rc2_40_cbc                    X509_PURP
F_SERVER_HELLO                    NID_rc2_64_cbc                    X509_PURP
F_SSL_CERT_NEW                    NID_rc2_cbc                       X509_PURP
F_SSL_GET_NEW_SESSION             NID_rc2_cfb64                     X509_PURP
F_SSL_NEW                         NID_rc2_ecb                       X509_PURP
F_SSL_READ                        NID_rc2_ofb64                     X509_PURP
F_SSL_RSA_PRIVATE_DECRYPT         NID_rc4                           X509_PURP
F_SSL_RSA_PUBLIC_ENCRYPT          NID_rc4_40                        X509_PURP
F_SSL_SESSION_NEW                 NID_rc5_cbc                       X509_TRUS
F_SSL_SESSION_PRINT_FP            NID_rc5_cfb64                     X509_TRUS
F_SSL_SET_FD                      NID_rc5_ecb                       X509_TRUS
F_SSL_SET_RFD                     NID_rc5_ofb64                     X509_TRUS
F_SSL_SET_WFD                     NID_ripemd160                     X509_TRUS
F_SSL_USE_CERTIFICATE             NID_ripemd160WithRSA              X509_TRUS
F_SSL_USE_CERTIFICATE_ASN1        NID_rle_compression               X509_TRUS
F_SSL_USE_CERTIFICATE_FILE        NID_rsa                           X509_TRUS
F_SSL_USE_PRIVATEKEY              NID_rsaEncryption                 X509_V_ER
F_SSL_USE_PRIVATEKEY_ASN1         NID_rsadsi                        X509_V_ER
F_SSL_USE_PRIVATEKEY_FILE         NID_safeContentsBag               X509_V_ER
F_SSL_USE_RSAPRIVATEKEY           NID_sdsiCertificate               X509_V_ER
F_SSL_USE_RSAPRIVATEKEY_ASN1      NID_secretBag                     X509_V_ER
F_SSL_USE_RSAPRIVATEKEY_FILE      NID_serialNumber                  X509_V_ER
F_WRITE_PENDING                   NID_server_auth                   X509_V_ER
GEN_DIRNAME                       NID_sha                           X509_V_ER
GEN_DNS                           NID_sha1                          X509_V_ER
GEN_EDIPARTY                      NID_sha1WithRSA                   X509_V_ER
GEN_EMAIL                         NID_sha1WithRSAEncryption         X509_V_ER
GEN_IPADD                         NID_shaWithRSAEncryption          X509_V_ER
GEN_OTHERNAME                     NID_stateOrProvinceName           X509_V_ER
GEN_RID                           NID_subject_alt_name              X509_V_ER
GEN_URI                           NID_subject_key_identifier        X509_V_ER
GEN_X400                          NID_surname                       X509_V_ER
LIBRESSL_VERSION_NUMBER           NID_sxnet                         X509_V_ER
MBSTRING_ASC                      NID_time_stamp                    X509_V_ER
MBSTRING_BMP                      NID_title                         X509_V_ER
MBSTRING_FLAG                     NID_undef                         X509_V_ER
MBSTRING_UNIV                     NID_uniqueIdentifier              X509_V_ER
MBSTRING_UTF8                     NID_x509Certificate               X509_V_ER
MIN_RSA_MODULUS_LENGTH_IN_BYTES NID_x509Crl                         X509_V_ER
MODE_ACCEPT_MOVING_WRITE_BUFFER NID_zlib_compression                X509_V_ER
```

```
    MODE_AUTO_RETRY                NOTHING                                  X509_V_ER
    MODE_ENABLE_PARTIAL_WRITE      OCSP_RESPONSE_STATUS_INTERNALERROR       X509_V_ER
    MODE_RELEASE_BUFFERS           OCSP_RESPONSE_STATUS_MALFORMEDREQUEST    X509_V_ER
    NID_OCSP_sign                  OCSP_RESPONSE_STATUS_SIGREQUIRED         X509_V_ER
    NID_SMIMECapabilities          OCSP_RESPONSE_STATUS_SUCCESSFUL          X509_V_ER
    NID_X500                       OCSP_RESPONSE_STATUS_TRYLATER            X509_V_ER
    NID_X509                       OCSP_RESPONSE_STATUS_UNAUTHORIZED        X509_V_ER
    NID_ad_OCSP                    OPENSSL_BUILT_ON                         X509_V_ER
    NID_ad_ca_issuers              OPENSSL_CFLAGS                           X509_V_ER
    NID_algorithm                  OPENSSL_DIR                              X509_V_ER
    NID_authority_key_identifier   OPENSSL_ENGINES_DIR                      X509_V_ER
    NID_basic_constraints          OPENSSL_PLATFORM                         X509_V_ER
    NID_bf_cbc                     OPENSSL_VERSION                          X509_V_ER
    NID_bf_cfb64                   OPENSSL_VERSION_NUMBER                   X509_V_ER
    NID_bf_ecb                     OP_ALL                                   X509_V_ER
    NID_bf_ofb64                   OP_ALLOW_NO_DHE_KEX                      X509_V_ER
    NID_cast5_cbc                  OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION     X509_V_ER
    NID_cast5_cfb64                OP_CIPHER_SERVER_PREFERENCE              X509_V_ER
    NID_cast5_ecb                  OP_CISCO_ANYCONNECT                      X509_V_ER
    NID_cast5_ofb64                OP_COOKIE_EXCHANGE                       X509_V_ER
    NID_certBag                    OP_CRYPTOPRO_TLSEXT_BUG                  X509_V_ER
    NID_certificate_policies       OP_DONT_INSERT_EMPTY_FRAGMENTS           X509_V_ER
    NID_client_auth                OP_ENABLE_MIDDLEBOX_COMPAT               X509_V_ER
    NID_code_sign                  OP_EPHEMERAL_RSA                         X509_V_ER
    NID_commonName                 OP_LEGACY_SERVER_CONNECT                 X509_V_ER
    NID_countryName                OP_MICROSOFT_BIG_SSLV3_BUFFER            X509_V_ER
    NID_crlBag                     OP_MICROSOFT_SESS_ID_BUG                 X509_V_ER
    NID_crl_distribution_points    OP_MSIE_SSLV2_RSA_PADDING                X509_V_ER
    NID_crl_number                 OP_NETSCAPE_CA_DN_BUG                    X509_V_ER
    NID_crl_reason                 OP_NETSCAPE_CHALLENGE_BUG                X509_V_ER
    NID_delta_crl                  OP_NETSCAPE_DEMO_CIPHER_CHANGE_BUG       X509_V_ER
    NID_des_cbc                    OP_NETSCAPE_REUSE_CIPHER_CHANGE_BUG      X509_V_ER
    NID_des_cfb64                  OP_NON_EXPORT_FIRST                      X509_V_ER
    NID_des_ecb                    OP_NO_ANTI_REPLAY                        X509_V_ER
    NID_des_ede                    OP_NO_CLIENT_RENEGOTIATION               X509_V_ER
    NID_des_ede3                   OP_NO_COMPRESSION                        X509_V_ER
    NID_des_ede3_cbc               OP_NO_ENCRYPT_THEN_MAC                   X509_V_ER
    NID_des_ede3_cfb64             OP_NO_QUERY_MTU                          X509_V_ER
    NID_des_ede3_ofb64             OP_NO_RENEGOTIATION                      X509_V_ER
    NID_des_ede_cbc                OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION X509_V_ER
    NID_des_ede_cfb64              OP_NO_SSL_MASK                           X509_V_ER
    NID_des_ede_ofb64              OP_NO_SSLv2                              X509_V_ER
    NID_des_ofb64                  OP_NO_SSLv3                              X509_V_ER
    NID_description                OP_NO_TICKET                             X509_V_ER
    NID_desx_cbc                   OP_NO_TLSv1                              X509_V_ER
    NID_dhKeyAgreement             OP_NO_TLSv1_1                            X509_V_ER
    NID_dnQualifier                OP_NO_TLSv1_2                            X509_V_ER
    NID_dsa                        OP_NO_TLSv1_3                            X509_V_ER
    NID_dsaWithSHA                 OP_PKCS1_CHECK_1                         X509_V_ER
    NID_dsaWithSHA1                OP_PKCS1_CHECK_2                         X509_V_ER
    NID_dsaWithSHA1_2              OP_PRIORITIZE_CHACHA                     X509_V_ER
    NID_dsa_2                      OP_SAFARI_ECDHE_ECDSA_BUG                X509_V_FL
    NID_email_protect              OP_SINGLE_DH_USE                         X509_V_FL
    NID_ext_key_usage              OP_SINGLE_ECDH_USE                       X509_V_FL
```

```
NID_ext_req                     OP_SSLEAY_080_CLIENT_DH_BUG               X509_V_FL
NID_friendlyName                OP_SSLREF2_REUSE_CERT_TYPE_BUG            X509_V_FL
NID_givenName                   OP_TLSEXT_PADDING                         X509_V_FL
NID_hmacWithSHA1                OP_TLS_BLOCK_PADDING_BUG                  X509_V_FL
NID_id_ad                       OP_TLS_D5_BUG                             X509_V_FL
NID_id_ce                       OP_TLS_ROLLBACK_BUG                       X509_V_FL
NID_id_kp                       READING                                   X509_V_FL
NID_id_pbkdf2                   RECEIVED_SHUTDOWN                         X509_V_FL
NID_id_pe                       RSA_3                                     X509_V_FL
NID_id_pkix                     RSA_F4                                    X509_V_FL
NID_id_qt_cps                   R_BAD_AUTHENTICATION_TYPE                 X509_V_FL
NID_id_qt_unotice               R_BAD_CHECKSUM                            X509_V_FL
NID_idea_cbc                    R_BAD_MAC_DECODE                          X509_V_FL
NID_idea_cfb64                  R_BAD_RESPONSE_ARGUMENT                   X509_V_FL
NID_idea_ecb                    R_BAD_SSL_FILETYPE                        X509_V_FL
NID_idea_ofb64                  R_BAD_SSL_SESSION_ID_LENGTH               X509_V_FL
NID_info_access                 R_BAD_STATE                               X509_V_FL
NID_initials                    R_BAD_WRITE_RETRY                         X509_V_FL
NID_invalidity_date             R_CHALLENGE_IS_DIFFERENT                  X509_V_FL
NID_issuer_alt_name             R_CIPHER_TABLE_SRC_ERROR                  X509_V_FL
NID_keyBag                      R_INVALID_CHALLENGE_LENGTH                X509_V_OK
NID_key_usage                   R_NO_CERTIFICATE_SET                      XN_FLAG_C
NID_localKeyID                  R_NO_CERTIFICATE_SPECIFIED                XN_FLAG_D
NID_localityName                R_NO_CIPHER_LIST                          XN_FLAG_D
NID_md2                         R_NO_CIPHER_MATCH                         XN_FLAG_F
NID_md2WithRSAEncryption        R_NO_PRIVATEKEY                           XN_FLAG_F
NID_md5                         R_NO_PUBLICKEY                            XN_FLAG_F
NID_md5WithRSA                  R_NULL_SSL_CTX                            XN_FLAG_F
NID_md5WithRSAEncryption        R_PEER_DID_NOT_RETURN_A_CERTIFICATE       XN_FLAG_F
NID_md5_sha1                    R_PEER_ERROR                              XN_FLAG_F
NID_mdc2                        R_PEER_ERROR_CERTIFICATE                  XN_FLAG_M
NID_mdc2WithRSA                 R_PEER_ERROR_NO_CIPHER                    XN_FLAG_O
NID_ms_code_com                 R_PEER_ERROR_UNSUPPORTED_CERTIFICATE_TYPE XN_FLAG_R
NID_ms_code_ind                 R_PUBLIC_KEY_ENCRYPT_ERROR                XN_FLAG_S
NID_ms_ctl_sign                 R_PUBLIC_KEY_IS_NOT_RSA                   XN_FLAG_S
NID_ms_efs                      R_READ_WRONG_PACKET_TYPE                  XN_FLAG_S
NID_ms_ext_req                  R_SHORT_READ                              XN_FLAG_S
NID_ms_sgc                      R_SSL_SESSION_ID_IS_DIFFERENT             XN_FLAG_S
NID_name                        R_UNABLE_TO_EXTRACT_PUBLIC_KEY            XN_FLAG_S
```

INTERNAL ONLY functions (do not use these)

The following functions are not intended for use from outside of Net::SSLeay module. They might be removed, renamed or changed without prior notice in future version.

Simply **DO NOT USE THEM**!

- hello
- blength
- constant

EXAMPLES

One very good example to look at is the implementation of `sslcat()` in the `SSLeay.pm` file.

The following is a simple SSLeay client (with too little error checking :-(

```
#!/usr/bin/perl
use Socket;
use Net::SSLeay qw(die_now die_if_ssl_error) ;
Net::SSLeay::load_error_strings();
Net::SSLeay::SSLeay_add_ssl_algorithms();
Net::SSLeay::randomize();

($dest_serv, $port, $msg) = @ARGV;      # Read command line
$port = getservbyname ($port, 'tcp') unless $port =~ /^\d+$/;
$dest_ip = gethostbyname ($dest_serv);
$dest_serv_params  = sockaddr_in($port, $dest_ip);

socket  (S, &AF_INET, &SOCK_STREAM, 0)  or die "socket: $!";
connect (S, $dest_serv_params)          or die "connect: $!";
select  (S); $| = 1; select (STDOUT);   # Eliminate STDIO buffering

# The network connection is now open, lets fire up SSL

$ctx = Net::SSLeay::CTX_new() or die_now("Failed to create SSL_CTX $!");
Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL)
     or die_if_ssl_error("ssl ctx set options");
$ssl = Net::SSLeay::new($ctx) or die_now("Failed to create SSL $!");
Net::SSLeay::set_fd($ssl, fileno(S));   # Must use fileno
$res = Net::SSLeay::connect($ssl) and die_if_ssl_error("ssl connect");
print "Cipher `" . Net::SSLeay::get_cipher($ssl) . "'\n";

# Exchange data

$res = Net::SSLeay::write($ssl, $msg);  # Perl knows how long $msg is
die_if_ssl_error("ssl write");
CORE::shutdown S, 1;  # Half close --> No more output, sends EOF to server
$got = Net::SSLeay::read($ssl);         # Perl returns undef on failure
die_if_ssl_error("ssl read");
print $got;

Net::SSLeay::free ($ssl);               # Tear down connection
Net::SSLeay::CTX_free ($ctx);
close S;
```

The following is a simple SSLeay echo server (non forking):

```
#!/usr/bin/perl -w
use Socket;
use Net::SSLeay qw(die_now die_if_ssl_error);
Net::SSLeay::load_error_strings();
Net::SSLeay::SSLeay_add_ssl_algorithms();
Net::SSLeay::randomize();

$our_ip = "\0\0\0\0"; # Bind to all interfaces
$port = 1235;
$sockaddr_template = 'S n a4 x8';
$our_serv_params = pack ($sockaddr_template, &AF_INET, $port, $our_ip);

socket (S, &AF_INET, &SOCK_STREAM, 0)  or die "socket: $!";
bind (S, $our_serv_params)             or die "bind:   $!";
listen (S, 5)                          or die "listen: $!";
```

```
$ctx = Net::SSLeay::CTX_new ()         or die_now("CTX_new ($ctx): $!");
Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL)
     or die_if_ssl_error("ssl ctx set options");

# Following will ask password unless private key is not encrypted
Net::SSLeay::CTX_use_RSAPrivateKey_file ($ctx, 'plain-rsa.pem',
                                         &Net::SSLeay::FILETYPE_PEM);
die_if_ssl_error("private key");
Net::SSLeay::CTX_use_certificate_file ($ctx, 'plain-cert.pem',
                                       &Net::SSLeay::FILETYPE_PEM);
die_if_ssl_error("certificate");

while (1) {
    print "Accepting connections...\n";
    ($addr = accept (NS, S))           or die "accept: $!";
    select (NS); $| = 1; select (STDOUT);  # Piping hot!

    ($af,$client_port,$client_ip) = unpack($sockaddr_template,$addr);
    @inetaddr = unpack('C4',$client_ip);
    print "$af connection from " .
    join ('.', @inetaddr) . ":$client_port\n";

    # We now have a network connection, lets fire up SSLeay...

    $ssl = Net::SSLeay::new($ctx)      or die_now("SSL_new ($ssl): $!");
    Net::SSLeay::set_fd($ssl, fileno(NS));

    $err = Net::SSLeay::accept($ssl) and die_if_ssl_error('ssl accept');
    print "Cipher `" . Net::SSLeay::get_cipher($ssl) . "'\n";

    # Connected. Exchange some data.

    $got = Net::SSLeay::read($ssl);     # Returns undef on fail
    die_if_ssl_error("ssl read");
    print "Got `$got' (" . length ($got) . " chars)\n";

    Net::SSLeay::write ($ssl, uc ($got)) or die "write: $!";
    die_if_ssl_error("ssl write");

    Net::SSLeay::free ($ssl);           # Tear down connection
    close NS;
}
```

Yet another echo server. This one runs from `/etc/inetd.conf` so it avoids all the socket code overhead. Only caveat is opening an rsa key file – it had better be without any encryption or else it will not know where to ask for the password. Note how `STDIN` and `STDOUT` are wired to SSL.

```
#!/usr/bin/perl
# /etc/inetd.conf
#    ssltst stream tcp nowait root /path/to/server.pl server.pl
# /etc/services
#    ssltst         1234/tcp

use Net::SSLeay qw(die_now die_if_ssl_error);
Net::SSLeay::load_error_strings();
Net::SSLeay::SSLeay_add_ssl_algorithms();
```

```
    Net::SSLeay::randomize();

    chdir '/key/dir' or die "chdir: $!";
    $| = 1;  # Piping hot!
    open LOG, ">>/dev/console" or die "Can't open log file $!";
    select LOG; print "server.pl started\n";

    $ctx = Net::SSLeay::CTX_new()     or die_now "CTX_new ($ctx) ($!)";
    $ssl = Net::SSLeay::new($ctx)     or die_now "new ($ssl) ($!)";
    Net::SSLeay::set_options($ssl, &Net::SSLeay::OP_ALL)
         and die_if_ssl_error("ssl set options");

    # We get already open network connection from inetd, now we just
    # need to attach SSLeay to STDIN and STDOUT
    Net::SSLeay::set_rfd($ssl, fileno(STDIN));
    Net::SSLeay::set_wfd($ssl, fileno(STDOUT));

    Net::SSLeay::use_RSAPrivateKey_file ($ssl, 'plain-rsa.pem',
                                         Net::SSLeay::FILETYPE_PEM);
    die_if_ssl_error("private key");
    Net::SSLeay::use_certificate_file ($ssl, 'plain-cert.pem',
                                       Net::SSLeay::FILETYPE_PEM);
    die_if_ssl_error("certificate");

    Net::SSLeay::accept($ssl) and die_if_ssl_err("ssl accept: $!");
    print "Cipher `" . Net::SSLeay::get_cipher($ssl) . "'\n";

    $got = Net::SSLeay::read($ssl);
    die_if_ssl_error("ssl read");
    print "Got `$got' (" . length ($got) . " chars)\n";

    Net::SSLeay::write ($ssl, uc($got)) or die "write: $!";
    die_if_ssl_error("ssl write");

    Net::SSLeay::free ($ssl);         # Tear down the connection
    Net::SSLeay::CTX_free ($ctx);
    close LOG;
```

There are also a number of example/test programs in the examples directory:

```
    sslecho.pl   -  A simple server, not unlike the one above
    minicli.pl   -  Implements a client using low level SSLeay routines
    sslcat.pl    -  Demonstrates using high level sslcat utility function
    get_page.pl  -  Is a utility for getting html pages from secure servers
    callback.pl  -  Demonstrates certificate verification and callback usage
    stdio_bulk.pl       - Does SSL over Unix pipes
    ssl-inetd-serv.pl   - SSL server that can be invoked from inetd.conf
    httpd-proxy-snif.pl - Utility that allows you to see how a browser
                          sends https request to given server and what reply
                          it gets back (very educative :-)
    makecert.pl  -  Creates a self signed cert (does not use this module)
```

INSTALLATION

See README and README.* in the distribution directory for installation guidance on a variety of platforms.

LIMITATIONS

`Net::SSLeay::read()` uses an internal buffer of 32KB, thus no single read will return more. In practice one read returns much less, usually as much as fits in one network packet. To work around this, you should use a loop like this:

```
$reply = '';
while ($got = Net::SSLeay::read($ssl)) {
    last if print_errs('SSL_read');
    $reply .= $got;
}
```

Although there is no built-in limit in `Net::SSLeay::write()`, the network packet size limitation applies here as well, thus use:

```
$written = 0;

while ($written < length($message)) {
    $written += Net::SSLeay::write($ssl, substr($message, $written));
    last if print_errs('SSL_write');
}
```

Or alternatively you can just use the following convenience functions:

```
Net::SSLeay::ssl_write_all($ssl, $message) or die "ssl write failure";
$got = Net::SSLeay::ssl_read_all($ssl) or die "ssl read failure";
```

KNOWN BUGS AND CAVEATS

An OpenSSL bug CVE−2015−0290 "OpenSSL Multiblock Corrupted Pointer Issue" can cause POST requests of over 90kB to fail or crash. This bug is reported to be fixed in OpenSSL 1.0.2a.

Autoloader emits a

```
Argument "xxx" isn't numeric in entersub at blib/lib/Net/SSLeay.pm'
```

warning if die_if_ssl_error is made autoloadable. If you figure out why, drop me a line.

Callback set using `SSL_set_verify()` does not appear to work. This may well be an openssl problem (e.g. see `ssl/ssl_lib.c` line 1029). Try using `SSL_CTX_set_verify()` instead and do not be surprised if even this stops working in future versions.

Callback and certificate verification stuff is generally too little tested.

Random numbers are not initialized randomly enough, especially if you do not have `/dev/random` and/or `/dev/urandom` (such as in Solaris platforms − but it's been suggested that cryptorand daemon from the SUNski package solves this). In this case you should investigate third party software that can emulate these devices, e.g. by way of a named pipe to some program.

Another gotcha with random number initialization is randomness depletion. This phenomenon, which has been extensively discussed in OpenSSL, Apache-SSL, and Apache−mod_ssl forums, can cause your script to block if you use `/dev/random` or to operate insecurely if you use `/dev/urandom`. What happens is that when too much randomness is drawn from the operating system's randomness pool then randomness can temporarily be unavailable. `/dev/random` solves this problem by waiting until enough randomness can be gathered − and this can take a long time since blocking reduces activity in the machine and less activity provides less random events: a vicious circle. `/dev/urandom` solves this dilemma more pragmatically by simply returning predictable "random" numbers. Some `/dev/urandom` emulation software however actually seems to implement `/dev/random` semantics. Caveat emptor.

I've been pointed to two such daemons by Mik Firestone <mik@@speed.stdio._com> who has used them on Solaris 8:

1. Entropy Gathering Daemon (EGD) at <http://www.lothar.com/tech/crypto/>
2. Pseudo-random number generating daemon (PRNGD) at <http://www.aet.tu−cottbus.de/personen/jaenicke/postfix_tls/prngd.html>

If you are using the low level API functions to communicate with other SSL implementations, you would do well to call

```
    Net::SSLeay::CTX_set_options($ctx, &Net::SSLeay::OP_ALL)
          or die_if_ssl_error("ssl ctx set options");
```

to cope with some well know bugs in some other SSL implementations. The high level API functions always set all known compatibility options.

Sometimes `sslcat()` (and the high level HTTPS functions that build on it) is too fast in signaling the EOF to legacy HTTPS servers. This causes the server to return empty page. To work around this problem you can set the global variable

```
    $Net::SSLeay::slowly = 1;   # Add sleep so broken servers can keep up
```

HTTP/1.1 is not supported. Specifically this module does not know to issue or serve multiple http requests per connection. This is a serious shortcoming, but using the SSL session cache on your server helps to alleviate the CPU load somewhat.

As of version 1.09 many newer OpenSSL auxiliary functions were added (from `REM_AUTOMATICALLY_GENERATED_1_09` onwards in `SSLeay.xs`). Unfortunately I have not had any opportunity to test these. Some of them are trivial enough that I believe they "just work", but others have rather complex interfaces with function pointers and all. In these cases you should proceed wit great caution.

This module defaults to using OpenSSL automatic protocol negotiation code for automatically detecting the version of the SSL/TLS protocol that the other end talks. With most web servers this works just fine, but once in a while I get complaints from people that the module does not work with some web servers. Usually this can be solved by explicitly setting the protocol version, e.g.

```
   $Net::SSLeay::ssl_version = 2;  # Insist on SSLv2
   $Net::SSLeay::ssl_version = 3;  # Insist on SSLv3
   $Net::SSLeay::ssl_version = 10; # Insist on TLSv1
   $Net::SSLeay::ssl_version = 11; # Insist on TLSv1.1
   $Net::SSLeay::ssl_version = 12; # Insist on TLSv1.2
   $Net::SSLeay::ssl_version = 13; # Insist on TLSv1.3
```

Although the autonegotiation is nice to have, the SSL standards do not formally specify any such mechanism. Most of the world has accepted the SSLeay/OpenSSL way of doing it as the de facto standard. But for the few that think differently, you have to explicitly speak the correct version. This is not really a bug, but rather a deficiency in the standards. If a site refuses to respond or sends back some nonsensical error codes (at the SSL handshake level), try this option before mailing me.

On some systems, OpenSSL may be compiled without support for SSLv2. If this is the case, Net::SSLeay will warn if ssl_version has been set to 2.

The high level API returns the certificate of the peer, thus allowing one to check what certificate was supplied. However, you will only be able to check the certificate after the fact, i.e. you already sent your form data by the time you find out that you did not trust them, oops.

So, while being able to know the certificate after the fact is surely useful, the security minded would still choose to do the connection and certificate verification first and only then exchange data with the site. Currently none of the high level API functions do this, thus you would have to program it using the low level API. A good place to start is to see how the `Net::SSLeay::http_cat()` function is implemented.

The high level API functions use a global file handle `SSLCAT_S` internally. This really should not be a problem because there is no way to interleave the high level API functions, unless you use threads (but threads are not very well supported in perl anyway). However, you may run into problems if you call undocumented internal functions in an interleaved fashion. The best solution is to "require Net::SSLeay" in one thread after all the threads have been created.

DIAGNOSTICS

Random number generator not seeded!!!

(W) This warning indicates that `randomize()` was not able to read `/dev/random` or `/dev/urandom`, possibly because your system does not have them or they are differently named. You can still use SSL, but the encryption will not be as strong.

open_tcp_connection: destination host not found:'server' (port 123) ($!)

Name lookup for host named `server` failed.

open_tcp_connection: failed 'server', 123 ($!)

The name was resolved, but establishing the TCP connection failed.

msg 123: 1 − error:140770F8:SSL routines:SSL23_GET_SERVER_HELLO:unknown proto

SSLeay error string. The first number (123) is the PID, the second number (1) indicates the position of the error message in SSLeay error stack. You often see a pile of these messages as errors cascade.

msg 123: 1 − **error:02001002::lib** (2) :**func** (1) :**reason** (2)

The same as above, but you didn't call **load_error_strings()** so SSLeay couldn't verbosely explain the error. You can still find out what it means with this command:

```
    /usr/local/ssl/bin/ssleay errstr 02001002
```

Password is being asked for private key

This is normal behaviour if your private key is encrypted. Either you have to supply the password or you have to use an unencrypted private key. Scan OpenSSL.org for the FAQ that explains how to do this (or just study examples/makecert.pl which is used during `make test` to do just that).

SECURITY

You can mitigate some of the security vulnerabilities that might be present in your SSL/TLS application:

BEAST Attack

http://blogs.cisco.com/security/beat−the−beast−with−tls/
https://community.qualys.com/blogs/securitylabs/2011/10/17/mitigating−the−beast−attack−on−tls
http://blog.zoller.lu/2011/09/beast−summary−tls−cbc−countermeasures.html

The BEAST attack relies on a weakness in the way CBC mode is used in SSL/TLS. In OpenSSL versions 0.9.6d and later, the protocol-level mitigation is enabled by default, thus making it not vulnerable to the BEAST attack.

Solutions:

- Compile with OpenSSL versions 0.9.6d or later, which enables SSL_OP_ALL by default
- Ensure SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS is not enabled (its not enabled by default)
- Don't support SSLv2, SSLv3
- Actively control the ciphers your server supports with set_cipher_list:

Net::SSLeay::set_cipher_list($ssl, 'RC4−SHA:HIGH:!ADH');

Session Resumption

http://www.openssl.org/docs/ssl/SSL_CTX_set_options.html

The SSL Labs vulnerability test on your SSL server might report in red:

Session resumption No (IDs assigned but not accepted)

This report is not really bug or a vulnerability, since the server will not accept session resumption requests. However, you can prevent this noise in the report by disabling the session cache altogether: Net::SSLeay::CTX_set_session_cache_mode($ssl_ctx, **Net::SSLeay::SESS_CACHE_OFF()**); Use 0 if you don't have SESS_CACHE_OFF constant.

Secure Renegotiation and DoS Attack

https://community.qualys.com/blogs/securitylabs/2011/10/31/tls−renegotiation−and−denial−of−service−attacks

This is not a "security flaw," it is more of a DoS vulnerability.

Solutions:

- Do not support SSLv2
- Do not set the SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION option
- Compile with OpenSSL 0.9.8m or later

BUGS

If you encounter a problem with this module that you believe is a bug, please report it in one of the following ways:

- create a new issue <https://github.com/radiator-software/p5-net-ssleay/issues/new> under the Net-SSLeay GitHub project at <https://github.com/radiator−software/p5−net−ssleay>;
- open a ticket <https://rt.cpan.org/Ticket/Create.html?Queue=Net-SSLeay> using the CPAN RT bug tracker's web interface at <https://rt.cpan.org/Dist/Display.html?Queue=Net−SSLeay>;
- send an email to the CPAN RT bug tracker at bug−Net−SSLeay@rt.cpan.org <mailto:bug-Net-SSLeay@rt.cpan.org>.

Please make sure your bug report includes the following information:

- the code you are trying to run;
- your operating system name and version;
- the output of `perl −V`;
- the version of OpenSSL or LibreSSL you are using.

AUTHOR

Originally written by Sampo Kellomäki.

Maintained by Florian Ragwitz between November 2005 and January 2010.

Maintained by Mike McCauley between November 2005 and June 2018.

Maintained by Chris Novakovic, Tuure Vartiainen and Heikki Vatiainen since June 2018.

COPYRIGHT

LICENSE

This module is released under the terms of the Artistic License 2.0. For details, see the `LICENSE` file distributed with Net-SSLeay's source code.

SEE ALSO

```
Net::SSLeay::Handle                      - File handle interface
./examples                               - Example servers and a clients
<http://www.openssl.org/>                - OpenSSL source, documentation, etc
openssl-users-request@openssl.org        - General OpenSSL mailing list
<http://www.ietf.org/rfc/rfc2246.txt>    - TLS 1.0 specification
<http://www.w3c.org>                     - HTTP specifications
<http://www.ietf.org/rfc/rfc2617.txt>    - How to send password
<http://www.lothar.com/tech/crypto/>     - Entropy Gathering Daemon (EGD)
<http://www.aet.tu-cottbus.de/personen/jaenicke/postfix_tls/prngd.html>
                          - pseudo-random number generating daemon (PRNGD)
perl(1)
perlref(1)
perllol(1)
perldoc ~openssl/doc/ssl/SSL_CTX_set_verify.pod
```