genua Ein Unternehmen der Bundesdruckerei

~~Home~~

~~Professional~~

Ultimate

IO::Socket::SSL

genua Ein Unternehmen der Bundesdruckerei

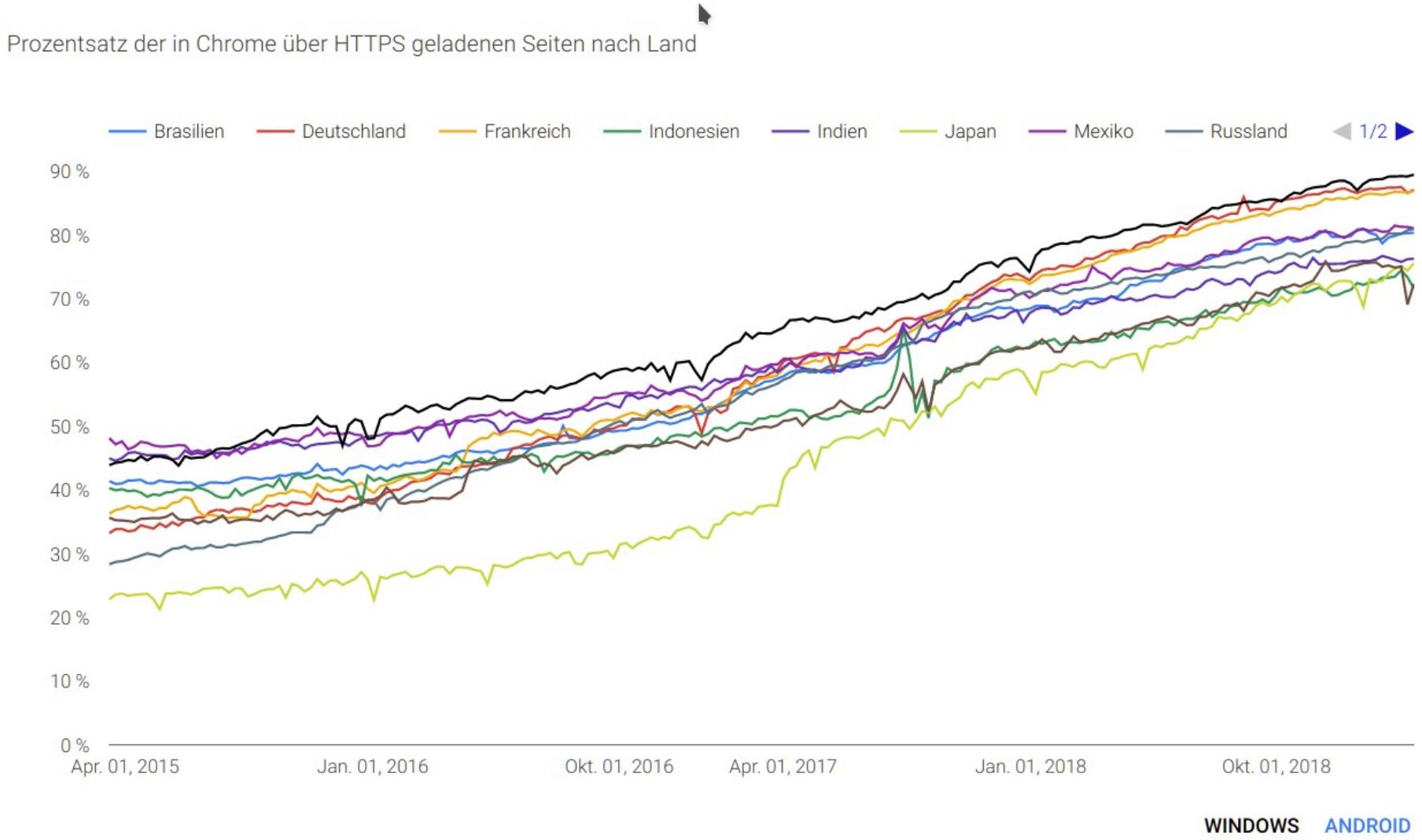
- Steffen Ullrich
- doing Perl seriously since 1997
  outside of web development
- genua GmbH since 2001
  Firewalls and ERP – both with heavy use of Perl
  IT-Security Research: Web 2.0, APT, AI/ML ...
- Maintainer IO::Socket::SSL since 2006
  also Net::SIP, Net::PcapWriter, Devel::TrackObjects,
  Mail::{SPF,DKIM,DMARC}::Iterator, Net::{INET6,SSL}Glue,
  Net::Inspect, Net::IMP*, App::DubiousHTTP, ...
  SSL+IPv6 support in Net::{SMTP,FTP,...}

genua Ein Unternehmen der Bundesdruckerei

- IO::Socket interface to SSL
  uses Net::SSLeay
  behaves **mostly** like other IO::Socket modules
  sane and secure defaults
  makes typical stuff simple and rare stuff possible
- Main differences to other IO::Socket mods
  - Encryption and authentication
    Resulting in security but also
    overhead impacting performance and
    additional (and complex) initial setup and teardown
  - Transport unit is frame not byte
    Resulting in trouble with non-blocking I/O

Why is SSL relevant?

genua Ein Unternehmen der Bundesdruckerei



genua Ein Unternehmen der Bundesdruckerei

- Basic usage
  simple client and server
- More complex usage scenarios
  forking server
  start and stop SSL on existing connection
  multiple certificates on same server
  SSL and plain on same port
  non-blocking sockets
  design issues when subclassing from IO::Socket*
  workarounds for uncooperative modules

genua Ein Unternehmen der Bundesdruckerei

- Authentication
  secure use of self-signed certificates
  mutual authentication (client certificates)
  OCSP
- Encryption
  what are these ciphers
- Performance
  impact of key exchange
  session reuse
  reuse context
  ECC vs RSA certificates

genua Ein Unternehmen der Bundesdruckerei

- Easy creation of certificates
  certificate authority, leaf certificates
  RSA, ECC, key usage ...
  dynamic creation for man in the middle
- Typical problems
  missing chain certificates
  TLS 1.0 disabled
  3DES no longer compiled in
  how to debug problems
- Other stuff
  OpenSSL 1.1.1, TLS 1.3
  IPv6

Basic Usage

genua Ein Unternehmen der Bundesdruckerei

```
use strict;
use warnings;
use IO::Socket::SSL;

my $client = IO::Socket::SSL->new(
    'www.google.com:443')
    or die "$!,$SSL_ERROR!";
print $client "GET / HTTP/1.0\r\n".
    "Host: www.google.com\r\n\r\n";
print <$client>;
```

Looks simple – like other IO::Socket*

Behind the scenes

- secure cipher + protocol
- proper validation of certificate chain and hostname
- check revocation if OCSP stapling was used
- SNI extension

```
use strict;
use warnings;
use IO::Socket::SSL;

my $server = IO::Socket::SSL->new(
    Listen => 10,
    LocalAddr => '127.0.0.1:9999',
    SSL_cert_file => 'certs/server-cert.pem',
    SSL_key_file => 'certs/server-key.pem',
) or die "$!,$SSL_ERROR";

while(1) {
    my $cl = $server->accept or next;
    print $cl "Hello world!\n";
}
```

Behind the scenes

- secure cipher + protocol
- secure DH param
- ECDHE setup

Complex Use Cases

genua Ein Unternehmen der Bundesdruckerei

Bad

```
my $s = IO::Socket::SSL->new(...);
while (1) {
  my $c = accept($s) or next;
  last if fork() == 0; # child

  # parent
  close($c);
}

# child process
close($s);
.. do something with $c ..
```

Good

```
my $s = IO::Socket::IP->new(...);
while (1) {
  my $c = accept($s) or next;
  last if fork() == 0; # child

  # parent
  close($c);
}

# child process
close($s);
IO::Socket::SSL->start_SSL($cl,
  SSL_server => 1, ...);
.. do something with $c ..
```

`close($c)` in parent does `SSL_shutdown` which affects the state of the SSL session – no longer usable in child

`accept($s)` in parent blocks user-space until TLS handshake is complete (easy Denial of Service)

undef vs close

genua Ein Unternehmen der Bundesdruckerei

With TCP all of this results in closing the socket

```
$c = IO::Socket::IP->new();
undef $c;
```

```
$c = IO::Socket::IP->new();
close($c);
```

With SSL both close the underlying TCP socket, but only the explicit close will cause a SSL shutdown

```
$c = IO::Socket::SSL->new();
undef $c;
```

```
$c = IO::Socket::SSL->new();
close($c);
```

Beware of implicit undef

```
{
    my $c = IO::Socket::SSL->new();
    # undef $c – it's there, even if not shown...
}
```

genua Ein Unternehmen der Bundesdruckerei

```
my $s = IO::Socket::SSL->new(...);
while (1) {
   my $c = accept($s) or next;
   last if fork() == 0; # child

   # parent
   close($c)
   # implicit undef $c
}

# child process
close($s);
.. do something with $c ..
```

Kind of works since no SSL shutdown is done in parent
DoS problem in `accept($s)` remains – but maybe good enough for some use cases

Upgrade from plain to TLS common (SMTP, IMAP,...)
Downgrade optionally in FTPS to achieve visibility for
FTP helpers in firewalls but protect authentication

```
my $c = IO::Socket::IP->new(...);
...
# upgrade to TLS to protect auth credentials
print $cl “AUTH TLS\r\n“;  # FTP TLS upgrade
<$cl> =~m{^2} or die;    # expect 2xx for success
IO::Socket::SSL->start_SSL($cl,   # upgrade to SSL
   %typical_sslargs
);
... do authentication ...
# downgrade after authentication
print $cl “CCC\r\n“;
<$cl> =~m{^2} or die;
$cl->stop_SSL();
```

multiple certificates on same server

genua Ein Unternehmen der Bundesdruckerei

```
my $cl = IO::Socket::SSL->new(
  LocalAddr => ..., Listen => ...,
  SSL_cert_file => {
     'foo.example.com' => 'cert-foo.pem',
     'bar.example.com' => 'cert-bar.pem',
     '' => 'cert-default.pem',
  },
  SSL_key_file => {
     'foo.example.com' => 'cert-foo-key.pem',
     'bar.example.com' => 'cert-bar-key.pem',
     '' => 'cert-default-key.pem',
  },
);
```

Client must support and use SNI (all modern clients do)
to get non-default certificate

In IO::Socket::SSL this is done with option
`SSL_hostname` which is set by default if possible

SSL and plain text on same port

genua Ein Unternehmen der Bundesdruckerei

Goal: accept plain text and SSL traffic on same socket
(i.e. both HTTP and HTTPS)
Works only fast for protocols where client sends first (i.e.
HTTP or SIP, but not FTP or SMTP)

```
my $s = IO::Socket::IP->new(...,Listen...);
while (1) {
  my $c = $s->accept or next;
  IO::Select->new->($c)->can_read or next;
  recv($c, my $buf, 10, MSG_PEEK);
  if ($buf =~m{\A\x16\0x03[\x00-\x03]}) {
    # start of SSLv3..TLSv1_2 ClientHello
    IO::Socket::SSL->start_SSL($c, ...) or die;
    ...
  } else {
    # assume plain text
    ...
  }
}
```

plain TCP: use select to detect if socket is readable

```
$c->blocking(0);
if (IO::Select->new($c)->can_read(timeout)) {
  sysread($c, my $buf, 8192); # will never block
  ...
}
```

SSL: select will only show if unread data in TCP connection. But there might be yet unread data in the last SSL frame read from the TCP connection

Will cause random errors if not properly handled

Alternative: always read maximum frame size (16384)

```
$c->blocking(0);
if ($c->pending or
  IO::Select->new($c)->can_read(timeout)) {
  sysread($c, my $buf, 8192); # will never block
  ...
}
```

plain TCP: wait until socket is writeable

```
use strict;
use warnings;
use IO::Socket::INET;
use IO::Select;

my $c = IO::Socket::INET->new(Type => SOCK_STREAM);
$c->blocking(0);
my $ok = connect($c,
    pack_sockaddr_in(80, inet_aton('1.1.1.1')));
{
    last if $ok;
    $!{EINPROGRESS} or die $!;
    IO::Select->new($c)->can_write(5) or die "timed out";
    my $err = getsockopt($c, SOL_SOCKET, SO_ERROR) or die;
    $err = unpack("i",$err) or last;
    $! = $err;
    die $!;
}
```

genua Ein Unternehmen der Bundesdruckerei

SSL: first connect with TCP, then upgrade to SSL
Handshake needs multiple read+writes
Retry `connect_SSL` until success or permanent failure

```
my $c = async_connect(...);

IO::Socket::SSL->start_SSL($c,
    SSL_startHandshake => 0,
    SSL_hostname => 'one.one.one.one',
) or die $SSL_ERROR;
my $sel = IO::Select->new($c);
while (1) {
    $c->connect_SSL and last;
    if ($SSL_ERROR == SSL_WANT_READ) {
        $sel->can_read(5) or die "timed out";
    } elsif ($SSL_ERROR == SSL_WANT_WRITE) {
        $sel->can_write(5) or die "timed out";
    } else {
        die "SSL connect failed: $SSL_ERROR";
    }
}
warn "SSL connect success - ". $c->get_cipher();
```

plain TCP: wait until socket is readable

```
my $s = IO::Socket::INET->new(
    LocalAddr => '127.0.0.1:9999',
    Listen => 10,
    Reuse => 1
) or die $!;
$s->blocking(0);

IO::Select->new($s)->can_read or next;
my $c = $s->accept or die;
$c->blocking(0);
```

genua Ein Unternehmen der Bundesdruckerei

SSL: first accept with TCP, then upgrade to SSL
Handshake needs multiple read+writes
Retry `accept_SSL` until success or permanent failure

```
my $c = async_accept(...);

IO::Socket::SSL->start_SSL($c,
    SSL_server => 1,
    SSL_startHandshake => 0,
    SSL_cert_file => ..., SSL_key_file => ...,
) or die $SSL_ERROR;
my $sel = IO::Select->new($c);
while (1) {
    $c->accept_SSL and last;
    if ($SSL_ERROR == SSL_WANT_READ) {
        $sel->can_read(5) or die "timed out";
    } elsif ($SSL_ERROR == SSL_WANT_WRITE) {
        $sel->can_write(5) or die "timed out";
    } else {
        die "SSL connect failed: $SSL_ERROR";
    }
}
warn "SSL accept success - ". $c->get_cipher();
```

Design issues when using IO::Socket*

genua Ein Unternehmen der Bundesdruckerei

- Some modules subclassed in the past from IO::Socket::INET, which made it painful to add IPv6 and SSL support:
  Net::SMTP, Net::FTP, ...
- Ugly workaround:
  Monkey patch module to replace IO::Socket::INET in @ISA with a module which can also do IPv6 and SSL, done by Net::INET6Glue and Net::SSLGlue for Net::SMTP, ...
- Still terrible workaround:
  Redesign the module so that it subclasses from whatever IO::Socket* module is available and maybe switch dynamically if needed (starttls) – done by Net::SMTP, ...
- Better: don't subclass from IO::Socket*
  It is hard to redesign later.

Workarounds for uncooperative modules

genua Ein Unternehmen der Bundesdruckerei

Some modules use IO::Socket::SSL but fail to make necessary options available or overwrite these with own (and sometimes insecure) values

Email::Sender::Transport::SMTPS , Mojo::UserAgent, Net::LDAP, ...

```
IO::Socket::SSL::set_args_filter_hack(sub {
    my ($is_server,$args) = @_;
    $args->{SSL_ca_file} = .... # use own trust store
    $args->{SSL_version} = 'TLSv1_2'; # enforce TLS 1.2
    ...
});

# or simply override anything with sane defaults
IO::Socket::SSL::set_args_filter_hack('use_defaults');
```

Authentication & Encryption

Self-signed or otherwise special certificates

genua Ein Unternehmen der Bundesdruckerei

Common recommendation for certificate problems is to disable validation. This is obviously a bad idea.

Other recommendations are to put the certificate in question into the trust store: does not work with leaf certificates in OpenSSL.

Recommended: match against a known fingerprint

```
IO::Socket::SSL->new(
    PeerAddr => 'example.com',
    SSL_fingerprint   => 'sha256$99111c0cc0...',  # cert
    # SSL_fingerprint => 'sha1$pub$39d64bba...' , # pubkey

    # multiple fingerprints are possible too
    # SSL_fingerprint => [
    #    'sha256$99111c0cc0...',
    #    'sha1$pub$39d64bba...',
    # ],

    # since 2.064 – hash method is detected from length of fp
    # SSL_fingerprint => '99:11:1C:0C:C0:...',
);
```

Self-signed or otherwise special certificates

genua Ein Unternehmen der Bundesdruckerei

- Common recommendation for certificate problems is to disable validation. This is obviously a bad idea.
- Other recommendations are to put the certificate in question into the trust store: ~~does not work with leaf certificates in OpenSSL.~~
  - Works when using OpenSSL 1.1.0+, IO::Socket::SSL 2.062+ and Net::SSLeay 1.83+
    - also LibreSSL 2.7.0+, Net::SSLeay 1.86
- Still checks hostname, while SSL_fingerprint only cares about the fingerprint

---

```
IO::Socket::SSL->new(
    PeerAddr => 'example.com',
   SSL_ca_file => 'cert.pem',
);
```

genua Ein Unternehmen der Bundesdruckerei

Usually only server authenticates to client
With mutual authentication client presents certificate too
Authentication can be optional or required

```
my $cl = IO::Socket::SSL->new(
    PeerAddr => 'example.com:443',
    SSL_cert_file => 'client-cert.pem',
    SSL_key_file => 'client-key.pem',
);

my $srv = IO::Socket::SSL->new(
    LocalAddr => ..., Listen => 10,
    SSL_cert_file => ..., SSL_key_file => ...,
    SSL_verify_mode => VERIFY_PEER
       | SSL_VERIFY_FAIL_IF_NO_PEER_CERT,   # required
    SSL_ca_file => 'ca-for-clientcert.pem',
);
```

Server will send list of accepted CA to client and will check that the certificate is issued by such CA

Subject will not be checked by server unless such check is explicitly implemented

```
use IO::Socket::SSL::Utils;
my $srv = IO::Socket::SSL->new(
    LocalAddr => ..., Listen => 10,
    SSL_cert_file => ..., SSL_key_file => ...,
    SSL_verify_mode => VERIFY_PEER
       | SSL_VERIFY_FAIL_IF_NO_PEER_CERT,   # required
    SSL_ca_file => 'ca-for-clientcert.pem',
    SSL_verify_callback => sub {
        my ($ok,$store,$certstring,$err,$cert,$depth) = @_;
        return $ok if $depth>0; # chain
        return 0 if CERT_asHash($cert)
            ->{subject}{commonName} !~m{goodguy};
        return $ok;
    },
);
```

Revocation check

genua Ein Unternehmen der Bundesdruckerei

The revocation of a certificate can be validated against a local CRL file or online against an OCSP server. None of this is done by default.

If the server sends a stapled OCSP response it is checked automatically.

```
my $cl = IO::Socket::SSL->new(
    PeerAddr => 'example.com:443',

    # CRL check:
    # somehow a current crl.pem must exist locally
    SSL_check_crl => 1,
    SSL_crl_file => 'crl.pem',
);
```

Revocation check

genua Ein Unternehmen der Bundesdruckerei

Any OCSP checks apart from checking a stapled response must be explicitly invoked since they involve additional (blocking) HTTP requests

Implementation of non-blocking OCSP handling is possible by iteratively feeding responses into the `ocsp_resolver`

```
my $cl = IO::Socket::SSL->new(
    PeerAddr => 'example.com:443',
    SSL_ocsp_mode =>
      SSL_OCSP_FULL_CHAIN | # check full chain (default off)
      SSL_OCSP_FAIL_HARD,   # fail hard if no response (off)
) or die;
if (my $errors =
    $cl->ocsp_resolver()->resolve_blocking()) {
    die “OCSP verification failed: $errors“;
}
```

Ciphers

genua Ein Unternehmen der Bundesdruckerei

Ciphers define a combination of algorithms to be used for
- Authentication (RSA/ECC certificates, PSK...)
- Key exchange (RSA, DH, ECDH...)
- Payload encryption (AES, 3DES, ChaCha20...)
- Payload authentication (SHA1, SHA256, Poly1305...)

IO::Socket::SSL uses a sane set of ciphers based on what browsers do and recommendations of Mozilla

Key `SSL_cipher_list`, Syntax see: `man ciphers`

TLS 1.3 Ciphers define ... algorithms to be used for
- ~~Authentication (RSA/ECC certificates, PSK...)~~
- ~~Key exchange (RSA, DH, ECDH...)~~
- Payload encryption (AES, ~~3DES,~~ ChaCha20...)
- Payload authentication (~~SHA1~~, SHA256, Poly1305...)

IO::Socket::SSL disables no TLS 1.3 ciphers

No configuration key yet for TLS 1.3 ciphers

Performance

genua Ein Unternehmen der Bundesdruckerei

- Latency
  2 Roundtrips for full Handshake
  1 Roundtrip for session Reuse or TLS 1.3
  (0 Roundtrip TLS 1.3)
- Bandwidth
  Large certificates and chains in TLS handshake
  Overhead of each SSL frame compared to payload
- CPU: Key exchange is costly
  RSA cheapest but no Forward Secrecy
  ECDH a bit worse, DH terrible
- CPU: Symmetric encryption is cheap
  As long as hardware supported: AES
  Or explicitly designed to be fast in software: ChaCha20

Reduce latency with session reuse - client

genua Ein Unternehmen der Bundesdruckerei

Expensive key exchange, certificate transfer and certificate validation are done only once. New TCP session can use estabished SSL session

Client needs to store session information for reuse

```
my $ctx = IO::Socket::SSL::SSL_Context->new(
   SSL_session_cache_size => 100,
);
my $cl0 = IO::Socket::SSL->new(
   PeerAddr => 'example.com:443',
   SSL_reuse_ctx => $ctx,
);
my $cl1 = IO::Socket::SSL->new(
   PeerAddr => 'example.com:443',
   SSL_reuse_ctx => $ctx,
);
```

genua Ein Unternehmen der Bundesdruckerei

Check if a session was reused

```
my $ctx = IO::Socket::SSL::SSL_Context->new(
    SSL_session_cache_size => 100
);
my $cl0 = IO::Socket::SSL->new(
    PeerAddr => 'example.com:443',
    SSL_reuse_ctx => $ctx,
) or die;
my $cl1 = IO::Socket::SSL->new(
    PeerAddr => 'example.com:443',
    SSL_reuse_ctx => $ctx,
) or die;
printf "reused cl0=%d cl1=%d\n",
    $cl0->get_session_reused || 0,
    $cl1->get_session_reused || 0;
```

genua Ein Unternehmen der Bundesdruckerei

Incomplete shutdown can invalidate a session

```
my $cl;
$cl = IO::Socket::SSL->new(
  PeerAddr => $dst,
  SSL_reuse_ctx => $ctx,
) or die;
undef $cl;

$cl = IO::Socket::SSL->new(
  PeerAddr => $dst,
  SSL_reuse_ctx => $ctx,
) or die;
printf "reused=%d\n",
  $cl->get_session_reused;
```

```
reused=0
```

```
my $cl;
$cl = IO::Socket::SSL->new(
  PeerAddr => $dst,
  SSL_reuse_ctx => $ctx,
) or die;
close($cl);

$cl = IO::Socket::SSL->new(
  PeerAddr => $dst,
  SSL_reuse_ctx => $ctx,
) or die;
printf "reused=%d\n",
  $cl->get_session_reused;
```

```
reused=1
```

Reduce latency with session reuse - client

genua Ein Unternehmen der Bundesdruckerei

In some cases it is possible or even necessary to use the same session for different targets
example: FTPS control and data connections

```
my $ctx = IO::Socket::SSL::SSL_Context->new(
   SSL_session_cache_size => 100,
);
my $control = IO::Socket::SSL->new(
   PeerAddr => 'example.com:990',
   SSL_reuse_ctx => $ctx,
   SSL_session_key => 'example.com',
);
my $data = IO::Socket::SSL->new(
   PeerAddr => 'example.com:12345',
   SSL_reuse_ctx => $ctx,
   SSL_session_key => 'example.com',
);
```

Reduce latency with session reuse - server

genua Ein Unternehmen der Bundesdruckerei

Server needs to store session information for reuse
Old: SessionId – server stores all information locally
New: Session tickets – all information are stored in encrypted ticket, server only has the key locally.
Single server: just reuse context

```
my $ctx = IO::Socket::SSL::SSL_Context->new(
   SSL_server => 1,
   SSL_cert_file => ..., SSL_key_file => ...
);
my $srv = IO::Socket::INET->new(Listen => 10,
   LocalAddr => '0.0.0.0:443');
my $cl = $srv->accept;
IO::Socket::SSL->start_SSL($cl,
   SSL_server => 1,
   SSL_reuse_ctx => $ctx,
);
```

Reduce latency with session reuse - server

genua Ein Unternehmen der Bundesdruckerei

Session tickets – all information are stored in encrypted ticket, server only has the key locally.
Multiple server can share key and thus SSL sessions.

```
Net::SSLeay::RAND_bytes(my $key, 32);
my $key_name = pack("a16", 'secret');
my $ctx = IO::Socket::SSL::SSL_Context->new(
   SSL_server => 1,
   SSL_cert_file => ..., SSL_key_file => ...
   SSL_ticket_keycb => sub {
      # see documentation for how to rotate keys
      return ($key,$key_name);
   }
);
my $srv = IO::Socket::SSL->new(Listen => 10,
   LocalAddr => '0.0.0.0:443',
   SSL_reuse_ctx => $ctx,
);
```

Reduce latency with session reuse - server

genua Ein Unternehmen der Bundesdruckerei

With mutual authentication a session id context must be given to make session reuse possible

```
my $ctx = IO::Socket::SSL::SSL_Context->new(
   SSL_server => 1,
   SSL_cert_file => ..., SSL_key_file => ...,
   SSL_session_id_context => 'foobar',
);
my $srv = IO::Socket::SSL->new(
   ...
   SSL_server => 1,
   SSL_reuse_ctx => $ctx,
);
```

Reduce latency with session reuse – TLS 1.3

genua Ein Unternehmen der Bundesdruckerei

Session Reuse got completly reworked in TLS 1.3:

- No support for session id
- Session tickets are send outside of TLS handshake
- Multiple session tickets can be provided by server
- Session tickets should better be only used once

Support with IO::Socket::SSL 2.061+, Net::SSLeay 1.86+

```
$c = IO::Socket::SSL->new(
   PeerAddr => ...
   SSL_reuse_ctx => ...
);
# session known with TLS 1.2, not with TLS 1.3
$s = Net::SSLeay::get1_session($c->_get_ssl_object);

<$c>;
# session known with both TLS 1.2 and TLS 1.3
$s = Net::SSLeay::get1_session($c->_get_ssl_object);
```

Reduce bandwidth with optimized certs

genua Ein Unternehmen der Bundesdruckerei

Full TLS handshake includes leaf certificate and chain.
Certificates get bigger, chains not smaller. Increased size can badly interact with TCP flow control.
ECC certificates are much smaller than RSA but not universally supported by clients.
Why not use both and pick the best on connect? Support available since OpenSSL 1.0.2, IO::Socket::SSL 2.063.

```
my $srv = IO::Socket::SSL->new(
   LocalAddr => ..., Listen => ...,
   SSL_cert_file => {
      'foo.com' => 'foo-cert.pem',
      'foo.com%ecc' => 'foo-cert-ecc.pem',
      '' => 'default-cert.pem',
      '%ecc' => 'default-cert-ecc.pem',
   },
   SSL_key_file => ...
);
```

Reduce bandwidth with optimized writes

genua Ein Unternehmen der Bundesdruckerei

Each SSL write will result in a new SSL frame, with overhead of encryption taking time and HMAC taking space.

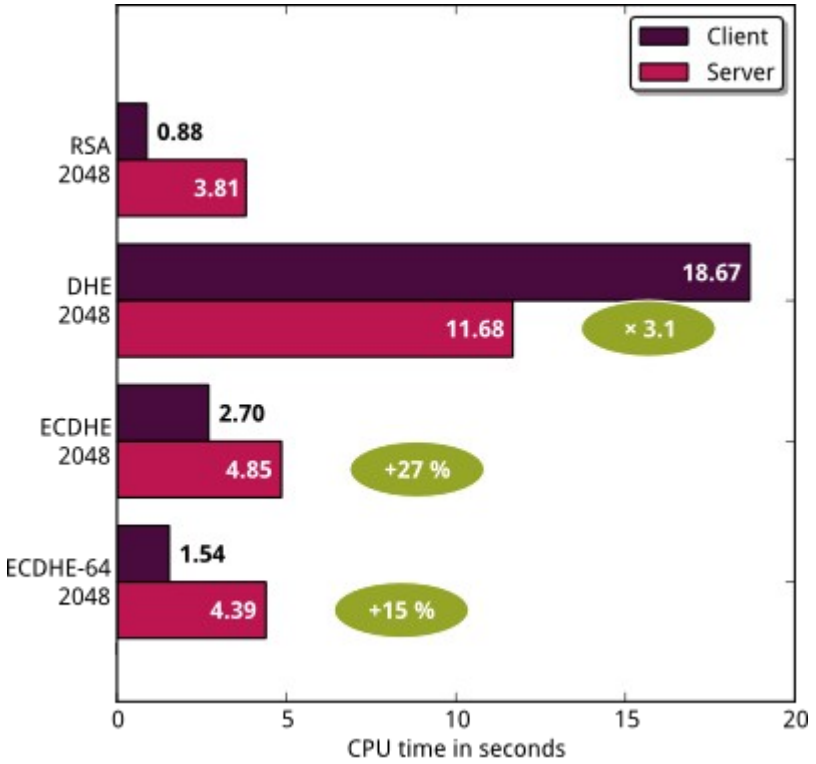
Choice of HMAC in cipher will affect overhead too. Default ciphers will prefer SHA256 instead SHA384.

```
# will likely result in one TCP package 'foobar'
print $tcpfh 'foo';
print $tcpfh 'bar';

# will result in two SSL frames for 'foo' and 'bar', each
# having their own encryption and HMAC (32 byte for SHA256).
print $sslfh 'foo';
print $sslfh 'bar';

# Better
print $sslfh 'foo'.'bar';
```

RSA key exchange is cheapest but does not provide forward secrecy – sniffed data can be decrypted later if private key of certificate is known

DH is really expensive

ECDH is more expensive than RSA, but cheap enough. Default ciphers will prefer ECDH.



TLS & Perfect Forward Secrecy
Vincent Bernat 28.11.2011
https://vincent.bernat.ch/en/blog/2011-ssl-perfect-forward-secrecy

Performance of symmetric encryption

genua Ein Unternehmen der Bundesdruckerei

AES is usually hardware-accelerated except for cheaper CPU platforms like lower end ARM

ChaCha20 is optimized for software, but has more CPU load than hardware-assisted AES

Browsers commonly set cipher order based on what is best for their platform – but this might affect server performance. https://calomel.org/aesni_ssl_performance.html

IO::Socket::SSL 2.065 prefers AES and enforces server order by default. Can be changed.

```
IO::Socket::SSL->new(
   Listen => 10, LocalAddr => ...,
   SSL_cert_file => ..., SSL_key_file => ...,
   SSL_cipher_list => ...,
   SSL_honor_cipher_order => 0, # prefer clients order
);
```

Easy creation of certificates

genua Ein Unternehmen der Bundesdruckerei

Typical process to create own certificates:
- create key, CA with openssl cmdline
- create key, CSR with openssl cmdline
- sign CSR with CA

```
$ openssl genrsa -aes256 -out ca-key.pem 2048
$ openssl req -x509 -new -nodes -extensions v3_ca \
   -key ca-key.pem -days 1024 -out ca-root.pem -sha512
$ openssl genrsa -out cert-key.pem 4096
$ openssl req -new -key cert-key.pem -out cert.csr -sha512
$ openssl x509 -req -in cert.csr -CA ca-root.pem \
   -CAkey ca-key.pem -CAcreateserial -out cert.pem \
   -days 365 -sha512
```

Terrible complex and not even sufficient:
- no subject alternative names
- leaf certificate is version 1 not 3
- no key usage information
- how about ECC certificates?

genua Ein Unternehmen der Bundesdruckerei

Can be both easier and better done with the power of Perl and sane defaults

```
use IO::Socket::SSL::Utils;
my @ca = CERT_create(CA => 1,
    subject => { CN => 'root CA' }
);
PEM_cert2file($ca[0],'ca.pem');
PEM_key2file($ca[1],'ca-key.pem');

my ($cert,$key) = CERT_create(
    issuer => \@ca,
    subject => { CN => 'server' },
    subjectAltNames => [
        ['IP','192.168.178.4'],
        ['DNS','example.com']
    ],
    key => KEY_create_ec(),      # optional, default RSA 2048
    purpose => 'client,server', # optional, sane default
    not_after => 10*365*86400,  # optional, default 1 year
    digest => 'sha512',         # optional, default sha256
);
```

genua Ein Unternehmen der Bundesdruckerei

IO::Socket::SSL::Utils

create certificates: `CERT_create`
extract information: `CERT_asHash`
load + save: `PEM_cert2file, PEM_file2cert, PEM_string2cert, PEM_key2file, ...`
create keys: `KEY_create_rsa, KEY_create_ec`

```
# dynamic certificate generation for MITM attacks

my @ca = CERT_create(CA => 1);
my $cl = IO::Socket::SSL->new('google.com:443');
my $old = CERT_asHash($cl->peer_certificate);
delete $old->{ext};
my @new = CERT_create(%$old, issuer => \@ca);
```

IO::Socket::SSL::Intercept

Dynamic creation of certificates from ProxyCA with persistent caching

Various stuff

Typical problems

genua Ein Unternehmen der Bundesdruckerei

- 3DES not compiled into newer openssl but peer needs it: build your own openssl lib with 3DES enabled and link against it
- Server requires TLS 1.2+: use a current version of openssl (specifically Mac OS)
- Missing chain certificates: fix server setup, import missing CA into trust store or use SSL_fingerprint
- Problem with unknown cause: debug with
  `perl -MIO::Socket::SSL=debug9 program.pl`
  Ask at stackoverflow, but provide versions of Net::SSLeay, openssl and IO::Socket::SSL
- Older problems and more information see talk from 2015 Debugging SSL/TLS
  https://noxxi.de/pws/2015/pws2015-ssl-debugged.pdf

genua Ein Unternehmen der Bundesdruckerei

- TLS 1.3 offers
  1-RTT handshake by default (instead of 2-RTT)
  optional 0-RTT handshake on reuse (unsupported)
  Improved security (ciphers, key exchange, early encryption includes certificates, ... encrypted SNI comes later)
- Requires Net::SSLeay 1.86 - not yet released
  Net::SSLeay 1.85 will not work properly
- Many behavior changes in both TLS 1.3 and OpenSSL 1.1.1, like ...
  Session reuse is totally different with TLS 1.3
  Selection of protocol version is different with TLS 1.3
  Cipher selection is totally different
  SSL_read behaves (again) slightly different
  https://wiki.openssl.org/index.php/TLS1.3

genua Ein Unternehmen der Bundesdruckerei

IO::Socket::SSL tries to subclass from IO::Socket::IP or IO::Socket::INET6, fallback is IO::Socket::INET
This means IPv6 is the default (this is 2019!!)
Some sites show different behavior with IPv6 and IPv4 due to differently broken setups: maybe enforce IPv4

```
# enforce IPv4 per socket
IO::Socket::SSL->new(
    PeerAddr => 'example.com:443',
    Family => AF_INET
);

# enforce globally – need to be on first use in code
use IO::Socket::SSL 'inet4'; # only IPv4
use IO::Socket::SSL 'inet6'; # only IPv6
use IO::Socket::SSL;         # both, prefer IPv6

# enforce temporarily
perl -MIO::Socket::SSL=inet4 program.pl
```

The End