# ***SSL (Secure Socket Layer)***

SSL (Secure Socket Layer) adalah arguably internet yang paling banyak digunakan untuk enkripsi. Ditambah lagi, SSL digunakan tidak hanya keamanan koneksi web, tetapi untuk berbagai aplikasi yang memerlukan enkripsi jaringan end-to-end.

Secure Sockets Layer (SSL) merupakan sistem yang digunakan untuk mengenkripsi  
pengiriman informasi pada internet, sehingga data dapat dikirim dengan aman. Protokol SSL mengatur keamanan dan integritas menggunakan enkripsi, autentikasi, dan kode autentikasi pesan. SSL protocol menyedian privasi komunikasi di internet. SSL tidak mendukung file encryption, access-control, atau proteksi virus, jadi SSL tidak dapat membantu mengatur data sensitif setelah dan sebelum pengiriman yang aman.

SSL menawarkan tiga tingkat keamanan, yaitu:

1. ***Authentication*** : Memastikan bahwa message yang diterima berasal dari seseorang yang tersurat.
2. ***Confidentiality*** : Melindungi pesan dari suatu usaha pembacaan oleh penerima yang tidak berhak disepanjang perjalanannya
3. ***Integrity*** : Memastikan bahwa pesan asli, tidak mengalami perubahan dalam perjalanannya.

## *RFC yang bersangkutan dengan SSL protokol*

TLS adalah Internet Engineering Task Force (IETF) standar track protokol, pertama kali didefinisikan pada tahun 1999 dan diperbarui dalam RFC 5246 (Agustus 2008) dan RFC 6176 (Maret 2011). Hal ini didasarkan pada spesifikasi SSL sebelumnya (1994, 1995, 1996) yang dikembangkan oleh Netscape Communications untuk menambahkan protokol HTTPS ke browser web Navigator mereka.

**Berikut merupakan versi dari SSL dengan perubahan yang dilakukan oleh RFC:**

* ***SSL 1.0, 2.0 dan 3.0***

Netscape mengembangkan protokol SSL **Versi 1.0** dan tidak pernah diperkenalakan ke publik karena kelemahan keamanan dalam protokol

* **SSL versi 2.0**, dikembangkan pada bulan Februari 1995, "berisi sejumlah kelemahan keamanan yang akhirnya menyebabkan desain SSL versi 3.0"

* **SSL versi 3.0**, dikembangkan pada tahun 1996, merupakan perubahan yang lengkap oleh protokol, yang diproduksi oleh Paul Kocher bekerja dengan Netscape insinyur Phil Karlton dan Alan Freier. Versi yang lebih baru dari SSL / TLS didasarkan pada SSL 3.0. 1996 draft SSL 3.0 diterbitkan oleh IETF sebagai dokumen sejarah dalam **RFC 6101**. Dr Taher ElGamal, kepala ilmuwan di Netscape Communications 1995-1998.  
  Pada 2014 versi 3.0 dari SSL dianggap tidak aman karena rentan terhadap serangan POODLE yang mempengaruhi semua cipher blok SSL ; dan RC4, satu-satunya non-blok cipher didukung oleh SSL 3.0, juga feasibly rusak seperti yang digunakan dalam SSL 3.0.
* ***TLS 1.0***

TLS 1.0 pertama kali didefinisikan dalam **RFC 2246** pada bulan Januari 1999 sebagai upgrade dari SSL Versi 3.0. Seperti yang tercantum dalam RFC, "perbedaan antara protokol ini dan SSL 3.0 tidak beda jauh, tetapi mereka cukup signifikan untuk mencegah interoperabilitas antara TLS 1.0 dan SSL 3.0".

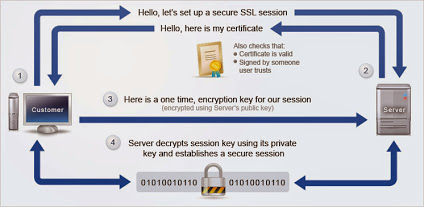
* ***TLS1.1***TLS 1.1 yang didefinisikan dalam **RFC 4346** pada bulan April 2006. Ini merupakan update dari TLS versi 1.0.
* ***TLS1.2***  
  TLS 1.2 yang didefinisikan dalam **RFC 5246** pada bulan Agustus 2008. Semua versi TLS telah disempurnakan dalam **RFC 6176** Maret 2011 yang menghapus kompatibilitas dengan SSL .

## *Cara Kerja SSL protokol*

Dalam berkomunikasi Secure Socket Layer (SSL) didefinisikan secara berbeda, pada client atau pada server. Client dan server mempunyai perbedaan utama pada aksi yang dilakukan ketika negosiasi tentang parameter keamanan. Client bertugas untuk mengajukan opsi SSL yang akan digunakan pada saat pertukaran pesan, dan server menentukan opsi mana yang akan digunakan. Pada implementasinya SSL berjalan pada transport layer dengan aplikasi SSLeay dan OpenSSL.

SSL juga memiliki penerus yaitu TLS ( Transport Layer Security) . Kedua protocol bersifat identik yaitu menjaga data online anda aman.

Secara umum cara kerja SSL dapat dijelaskan dalam beberapa tahapan yaitu sebagai berikut :

1. Tahapan Pembangunan Chanel.
2. Client membentuk koneksi awal ke server dan meminta koneksi SSL.
3. Jika server yang dihubungi telah dikonfigurasi dengan benar, maka server ini mengirimkan public key miliknya kepada client.
4. Client membandingkan sertifikat dari server ke basis data trusted authorities. Ji-ka sertifikat terdaftar di dalamnya, artinya client mempercayai (trust) server itudan akan maju ke tahap 4. Sehingga pemakai harus menambahkan sertifikat ter-sebut ke trusted database sebelum maju ke langkah ke 4.

* Client menggunakan Public Key yang didapatnya untuk mengenkripsi dan mengirimkan session key ke server. Jika server meminta sertifikat client di tahap 2, maka client harus mengirimnya sekarang.
* Jika server di setup untuk menerima sertifikat, maka server akan membandingkan sertifikat yang diterimanya dengan basis data trusted authorities dan akan menerima atau menolak koneksi yang diminta.
* Jika kondisi ditolak, suatu pesan kegagalan akan dikirimkan ke client. Apabila koneksi diterima, atau bila server tidak di setup untuk menerima sertifikat, maka server akan mendekripsi session key yang didapat dari client dengan private key milik server danmengirimkan pesan berhasil ke client yang dengan demikian membuka suatu securedata chanell.

## *Informasi Request dan Response SSL Protokol*

***The Hello Exchange***

Ketika klien SSL dan server mulai berkomunikasi, mereka melalukan proses Hand-Shaking. Dengan maksud, klien mengirimkan pesan “Hello Klien” ke server, kemudian Server harus membalas pesan “Halo” atau jika tidak maka kesalahan terjadi dari koneksi gagal. The Client Hello dan Server Hello digunakan untuk membangun kemampuan tambahan keamanan antara klien dan server.

***Client Exchange***

**Client Sertificate [Optional]**

Ini adalah pesan pertama bahwa klien mengirimkan setelah client menerima pesan ”Server Hello Done”. Pesan ini hanya dikirim jika server meminta sertifikat. Jika tidak ada sertifikat yang cocok dan tersedia, klien akan mengirimkan sebuah peringatan no\_certificate . Pada server mungkin menanggapi peringatan tersebut dengan peringatan kegagalan Hand-Shaking.

**Client Key Exchange**

Isi pesan ini tergantung pada algoritma kunci publik yang dipilih antara pesan Client Hello dan Server Hello. Klien menggunakan salah kunci premaster dienkripsi oleh Rivest-Shamir-Addleman (RSA) algoritma untuk perjanjian kunci dan otentikasi. Ketika RSA digunakan untuk server otentikasi dan pertukaran kunci, pre\_master\_secret 48-byte yang dihasilkan oleh klien, dienkripsi bawah kunci publik server akan dikirim ke server. Server menggunakan kunci pribadi untuk mendekripsi pre\_master\_secret tersebut. Kedua belah pihak kemudian dikonversi pre\_master\_secret ke master\_secret tersebut.

**Sertificate Verify**

Jika klien mengirimkan sertifikat dengan kemampuan penandatanganan, Sertifikat digital ditandatangani dengan Verifikasi pesan yang dikirim secara eksplisit untuk memverifikasi sertifikat.

***Cipher Change***

**Change Cipher Spec Messages**

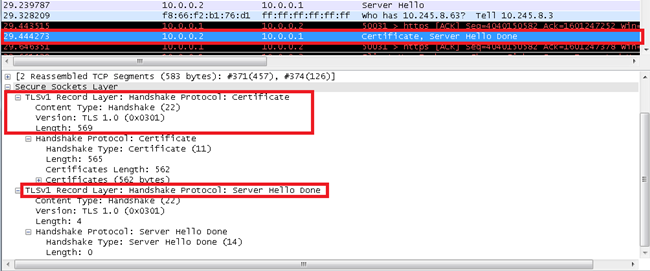
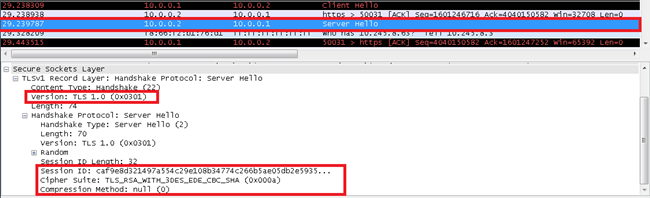
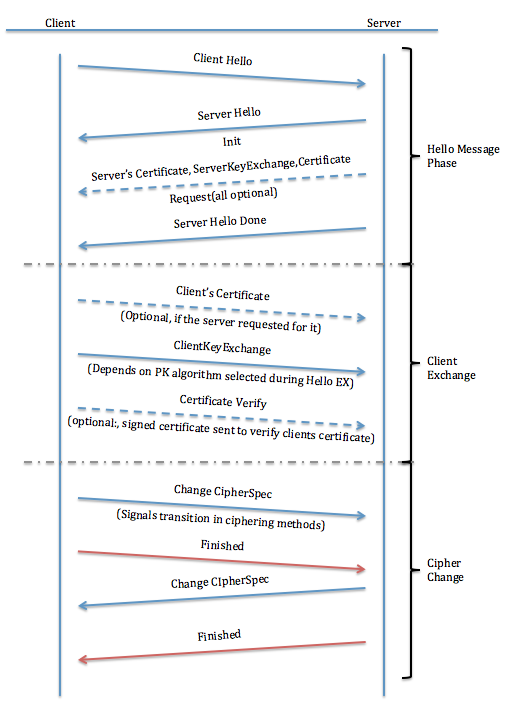
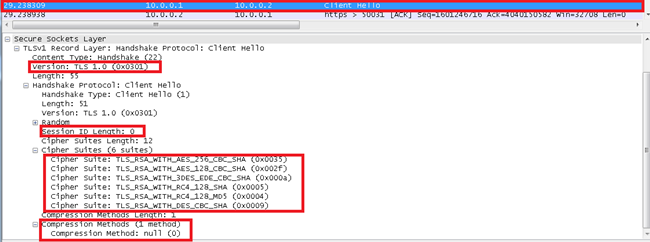
Perubahan Cipher Spec pesan dikirim oleh klien. Protokol perubahan Cipher Spec digunakan untuk sinyal transisi dalam strategi pengkodean. Protokol ini terdiri dari satu pesan yang dienkripsi dan dikompresi di bawah Cipher Spec. Pesan itu dikirim oleh kedua klien dan server untuk memberitahu pihak yang menerima bahwa catatan berikutnya dilindungi. Klien mengirimkan Hand Shaking dengan arti pertukaran kunci pesan Perubahan Cipher Spec berikut dan Sertifikat Verifikasi pesan (jika ada), dan server akan mengirimkan satu setelah berhasil memproses pesan pertukaran kunci itu diterima dari klien. Ketika sesi sebelumnya dilanjutkan, pesan Ganti Cipher Spec dikirim setelah pesan Hello.

**Finished Messages**

Pesan selesai selalu dikirim segera setelah pesan Perubahan Cipher Spec untuk memverifikasi bahwa pertukaran tersebut serta proses otentikasi kunci tersebut sukses. Pesan Selesai dilindungi paket pertama dengan yang terakhir yaitu negosiasi algoritma, kunci, dan rahasia. Pihak dapat mulai mengirim data terenkripsi segera setelah mereka mengirim pesan Finished.

**Berikut adalah gambaran yang ditunjukkan dalam struktur kasar dan struktur di wireshark:**

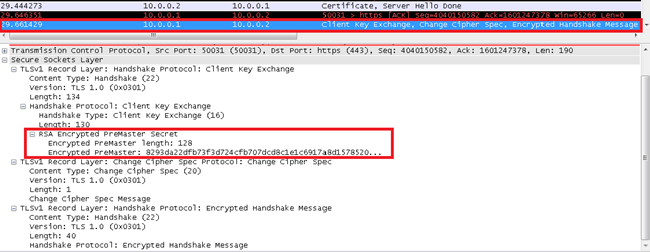
*Struktur Kasar dan Struktur di Wireshark*

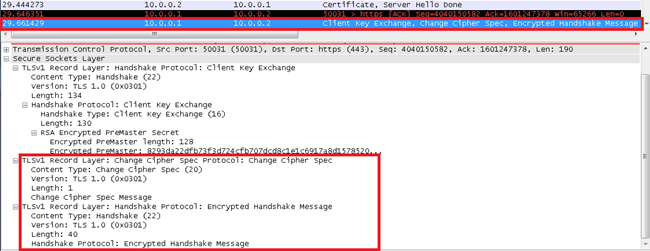
***Hello message phrase***

Client Hello

Server Hello

Server Hello Done

***Client Exchange***

***Chiper Change***

**Heartbleed Hello Ekstensi**

 Sebuah rekan  tidak bisa hanya menunjukkan bahwa pelaksanaannya mendukung Heartbeat, ia dapat memilih apakah mau menerima HeartbeatRequest  pesan dan menanggapi dengan pesan HeartbeatResponse atau hanya bersedia  untuk mengirim pesan HeartbeatRequest. Kemudian ditunjukkan dengan menggunakan peer\_allowed\_to\_send sebagai HeartbeatMode tersebut; yang terakhir ini ditunjukkan menggunakan peer\_not\_allowed\_to\_send sebagai modus Heartbeat. Keputusan ini  dapat diubah setiap renegosiasi. Pesan HeartbeatRequest   TIDAK HARUS dikirim ke rekan untuk menunjukkan peer\_not\_allowed\_to\_send. Jika  titik akhir yang telah menunjukkan peer\_not\_allowed\_to\_send menerima HeartbeatRequest pesan, titik akhir HARUS menjatuhkan pesan diam-diam dan MUNGKIN mengirim pesan Siaga unexpected\_message.

   Format Heartbeat Hello Ekstensi didefinisikan oleh:

   enum {

      peer\_allowed\_to\_send (1),

      peer\_not\_allowed\_to\_send (2),

      (255)

   } HeartbeatMode;

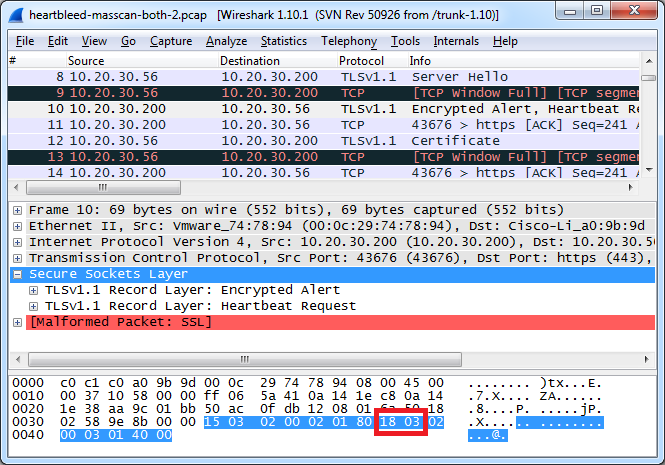
   struct {

      Modus HeartbeatMode;

   } HeartbeatExtension;

   Setelah penerimaan mode diketahui, pesan Siaga kesalahan menggunakan    illegal\_parameter sebagai AlertDescription yang harus dikirim dalam request

**Heartbeat Request and Response Messages**



Pesan protokol Heartbeat terdiri dari payload dan padding.

    struct {

HeartbeatMessageType type;

uint16 payload\_length;

opaque payload[HeartbeatMessage.payload\_length];

opaque padding[padding\_length];

} HeartbeatMessage;

   Panjang total dari HeartbeatMessage TIDAK HARUS melebihi 2 ^ 14 atau  max\_fragment\_length ketika dinegosiasikan sebagaimana didefinisikan dalam [RFC6066].

   Jenis: Jenis pesan, baik heartbeat\_request atau heartbeat\_response.

   payload\_length: Panjang payload.

   payload: payload terdiri dari konten yang asal-asalan.

   padding: Padding adalah konten acak yang harus diabaikan oleh penerima. Panjang HeartbeatMessage adalah TLSPlaintext.length  untuk TLS dan DTLSPlaintext.length untuk DTLS. Selain itu,  panjang jenis lapangan adalah 1 byte, dan panjang payload\_length adalah 2. Oleh karena itu, padding\_length adalah  TLSPlaintext.length - payload\_length - 3 untuk TLS dan  DTLSPlaintext.length - payload\_length - 3 untuk DTLS. Itu padding\_length HARUS setidaknya 16.

   Pengirim HeartbeatMessage HARUS menggunakan padding acak setidaknya 16 byte. Padding pesan HeartbeatMessage yang diterima  HARUS diabaikan.  Jika payload\_length dari HeartbeatMessage yang diterima terlalu besar,  yang HeartbeatMessage menerima HARUS dibuang diam-diam.

https://tools.ietf.org/html/rfc6520#page-5

http://www.cisco.com/c/en/us/support/docs/security-vpn/secure-socket-layer-ssl/116181-technote-product-00.html