KAJIAN PENERAPAN *PUBLIC KEY INFRASTRUCTURE* PADA *SECURE SOCKETS LAYER*

Sulistyo Unggul Wicaksono– NIM: 13503058

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail: if13058@students.if.itb.ac.id

Abstrak

Secure Sockets Layer (SSL) adalah protokol yang menggunakan *program layer* yang terletak diantara *layer Hpertext Transfer Protocol* (HTTP) dan layer *Transport Control Protocol* (TCP) pada Internet. Sebagai protokol yang memfasilitasi transmisi dokumen via Internet, peran utama SSL adalah untuk menjamin keamanan lalu lintas data dalam web. Keamanan dalam konteks ini meliputi kerahasiaan, integritas pesan, dan otentikasi yang dipenuhi dengan penerapan kriptografi, tanda tangan digital, dan sertifikat.

Dalam menjamin keamanan lalu lintas data, digunakan kriptografi simetris dan kriptografi asimetris, yang lazim disebut *Public Key Infrastructure* (PKI). Makalah ini akan menitikberatkan pembahasan pada penerapan PKI pada tahapan-tahapan pengamanan dalam SSL, yang esensial dalam memenuhi persyaratan elemen-elemen keamanan yang harus dijamin oleh SSL. Selain itu, makalah juga akan membahas tingkat keamanan yang diperoleh dengan penerapan PKI, termasuk kekuatan, kelemahan, dan prospek penerapan PKI dalam SSL maupun prospek SSL itu sendiri.

Kata Kunci : *Kriptografi,* Public Key Infrastructure, Secure Sockets Layer, Web, *Kunci*, *Algoritma.*

1. Pendahuluan

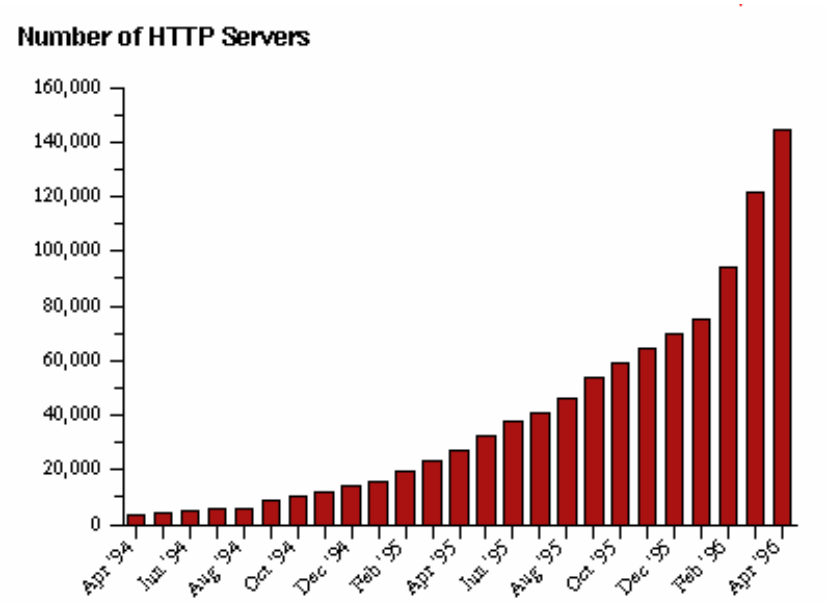
1.1. Latar Belakang

Internet sebagai jaringan data publik sangat diragukan keamanannya untuk transaksi dagang / bisnis. Bagaimana usaha yang dilakukan untuk menjamin keamanan data yang dikirimkan melalui jaringan publik Internet akan dicoba diterangkan pada kesempatan ini. Bentuk tulisan ini akan mendasarkan dirinya untuk menjawab beberapa pertanyaan yang sering muncul.

Beberapa cerita tambahan tentang ketahanan *sistem security* RSA juga akan dicoba dikembangkan & mungkin dapat menjadi masukan sejauh mana ketahanan sistem ini. Informasi tentang security sebetulnya cukup mudah di peroleh di Internet seperti di http://www.netscape.com, http://www.rsa.com atau melalui berbagai *search engine* di Internet menggunakan *keyword* yang berkaitan dengan *network security*.

Web sebagai media aplikasi untuk dunia bisnis di Internet merupakan *tool* yang paling menarik karena kemampuan multimedia & interaktifnya. Perkembangan yang pesat dari Web server ternyata cukup pesat, hal ini dapat kita lihat dengan jelas dari grafik pertumbuhan Web server di Internet yang di publikasikan oleh WebCrawler salah satu *search engine* yang besar di Internet.

Security merupakan prasyarat dasar karena pada dasarnya Internet adalah sebuah jaringan yang tidak aman. Jutaan komputer membentuk *public network* Internet dimana komunikasi dapat di dengarkan di tengah jalan. Pada saat data bergerak dari pengirim ke penerima, pasti melalui beberapa koneksi (*router* dll) sebelum mencapai tujuan akhirnya. Artinya komputer lain di luar dua komputer yang saling berkomunikasi tersebut sangat mungkin untuk akses data tersebut. Oleh karenanya, *security* merupakan prasyarat mutlak jika kita ingin data kita selamat melalui jaringan publik tersebut.



Gambar 1. Pertumbuhan web server di Internet

Pada dasarnya pada saat data dikirim melalui Internet akan terbuka pada tiga (3) macam resiko keamanan, yaitu:

1. *Eavesdropping* - komputer antara mendengarkan pembicaraan dari dua komputer yang sedang berbicara secara *private*.

2. Manipulasi - komputer antara akan mengganti informasi dalam data yang dikirimkan.

3. Impersonasi - sebuah komputer akan berpura-pura menjadi komputer lain (misalnya menjadi komputer tempat melakukan transaksi dagang).

Sebetulnya situasi ini sama persis dengan kondisi kita pada saat melakukan pembelian mail order melalui telepon. Ketiga resiko di atas akan terjadi pada saat melakukan transaksi *mail order*.

Internet pada dasarnya adalah jaringan yang tidak aman. Sebetulnya tidak ada teknik security di dunia ini yang berani mengklaim sebagai teknik yang tidak bisa di tembus / paling aman. Untuk itu perlu digiatkan para pengguna Internet untuk menjamin keamanannya untuk melakukan beberapa hal:

1. Selalu menggunakan versi terakhir dari *software* yang digunakan. Umumnya vendor *software* akan mengeluarkan versi terbaru untuk memperbaiki berbagai lubang *security*/*bug* yang ada.

2. Sedapat mungkin gunakan versi security tertinggi. Saat ini pengguna Netscape di US & Canada dapat menggunakan *software* Netscape yang mempunyai kunci 128-bit yang akan memberikan keamanan jauh lebih tinggi daripada kunci 40-bit yang saat ini boleh di-*export* keluar US. Regulasi di US sedang diperjuangkan untuk di ubah untuk memungkinkan penggunaan kunci dengan jumlah bit yang tinggi.

Pada saat ini ada banyak sekali pengembangan perangkat keamanan komputer maupun yang dapat dijalankan di jaringan komputer. Salah satu yang cukup pesat berkembang antara lain adalah *Secure Socket Layer* (SSL) yang dikembangkan oleh Netscape.

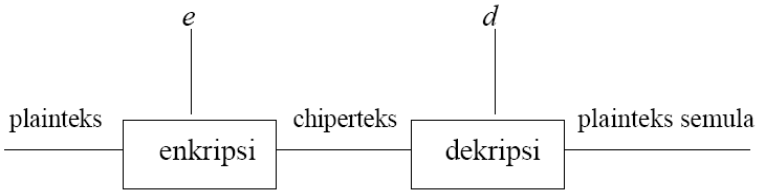
2. *Public Key Infrastructure*

Sampai akhir tahun 1970, hanya ada sistem kriptografi simetri. Karena sistem kriptografi simetri menggunakan kunci yang sama untuk enkripsi dan dekripsi, maka hal ini mengimplikasikan dua pihak yang berkomunikasi saling mempercayai. Kedua pihak harus menjaga kerahasiaan kunci (sehingga, junci enkripsi/dekripsi disebut juga *secret key*)

Pada sistem kriptografi kunci-publik, kunci kriptografi dibuat sepasang, satu kunci untuk enkripsi dan satu kunci untuk dekripsi

- Kunci untuk enkripsi diumumkan kepada publik–oleh karena itu tidak rahasia–sehingga dinamakan kunci publik (*public-key*), disimbolkan dengan e.
- Kunci untuk dekripsi bersifat rahasia-sehingga dinamakan kunci privat (*private key*), disimbolkan dengan d.

Karena ada kunci enkripsi yang tidak sama dengan kunci dekripsi, maka sistem kriptografi kunci-publik kadang-kadang disebut juga sistem kriptografi asimetri.



Gambar 2. Sistem kriptografi kunci publik

Ket: *e = public key, d = private key*

Serangan yang umum terjadi pada kunci publik tanpa identitas adalah penyamaran (*impersonation attack*). Seseorang yang memiliki kunci publik orang lain dapat menyamar seolah-olah dialah pemilik kunci itu. Serangan semacam ini adalah masalah yang muncul dari penggunaan kriptografi kunci-publik.

Contohnya, dalam teknologi *e-commerce*, pembayaran transaksi dilakukan dengan menggunakan kartu kredit. Pelanggan mengirimkan informasi kartu kreditnya (yang bersifat rahasia) melalui *website* pedagang *online*. Selama pengiriman, informasi kartu kredit tersebut dilindungi dengan cara mengenkripsinya dengan kunci publik pedagang *online*. Masalahnya adalah bagaimana pelanggan itu memastikan bahwa *website* pedagang *online* tersebut memang benar milik pedagang *online* dan bukan *website* pihak lain yang menyamar sebagai *website* pedagang asli dengan tujuan untuk mencuri informasi kartu kredit.

Untuk menjawab masalah di atas, solusinya adalah dengan memberikan sertifikat digital pada kunci publik. Sertifikat digital dikeluarkan oleh pemegang otoritas sertifikasi (*Certification Authority* atau CA). CA biasanya adalah institusi keuangan (seperti bank) atau institusi yang terpercaya.

Sertifikat digital adalah dokumen digital yang berisi informasi sebagai berikut:

- nama subjek (perusahaan/individu yang disertifikasi)
- kunci publik si subjek
- waktu kadaluarsa sertifikat (*expired time*)
- informasi relevan lain seperti nomor seri sertifikat, dll

CA membangkitkan nilai hash dari sertifikat digital tersebut (misalnya dengan fungsi hash satu-arah MD5 atau SHA), lalu menandatangani nilai hash tersebut dengan menggunakan kunci privat CA. Contoh sebuah sertifikat digital: Bob membawa kunci publiknya dan mendatangi CA untuk meminta sertifikat digital. CA mengeluarkan sertifikat digital dan menandatangani sertifikat tersebut dengan cara mengenkripsi nilai hash dari kunci publik Bob (atau nilai hash dari sertifikat digital keseluruhan) dengan menggunakan kunci privat

Jadi, sertifikat digital mengikat kunci publik dengan identitas pemilik kunci publik. Sertifikat ini dapat dianggap sebagai ‘surat pengantar’ dari CA. Supaya sertifikat digital itu dapat diverifikasi (dicek kebenarannya), maka kunci publik CA harus diketahui secara luas. Seseorang yang memiliki kunci publik CA dapat memverifikasi bahwa tanda tangan di dalam

suatu sertifikat itu sah dan karena itu mendapat jaminan bahwa kunci publik di dalam sertifikat itu memang benar. Sertifikat digital sendiri tidak rahasia, tersedia secara publik, dan disimpan oleh CA di dalam *certificate repositories*. Salinan (*copy*) sertifikat tersebut juga dimiliki oleh pemohon sertifikat.

Misalkan Alice mengakses homepage Bob untuk mendapatkan kunci publik Bob. Misalkan Carol berhasil memintas (*interception*) *request* Alice (*client*) ke homepage Bob (*server*), sehingga request tersebut masuk ke *homepage* Bob palsu (yang dibuat oleh Carol) (Tujuan memintas adalah agar Alice mengira Carol adalah Bob, sehingga Carol dapat memperoleh informasi rahasia dari Alice, misalnya kunci). Carol sudah meletakkan sertifikat digitalnya di dalam halaman web palsu, tapi jika Alice membaca sertifikat tersebut dia langsung paham bahwa dirinya sedang tidak berkomunikasi dengan Bob asli karena identitas Bob tidak terdapat di dalam sertifikat tersebut.

Misalkan Carol berhasil mengubah homepage Bob, mengganti kunci publik Bob di dalam sertifikat digital dengan kunci publiknya. Tetapi, jika Alice meng-hash sertifikat digital tersebut, dia memperoleh nilai hash yang tidak sama dengan nilai hash yang dihasilkan jika tandatangan digital diverifikasi dengan kunci publik CA. Karena Carol tidak mempunyai kunci privat CA, maka Carol tidak dapat membangkitkan tanda-tangan digital dari sertifikat Bob yang sudah diubah tersebut. Dengan cara ini, Alice dapat meyakini bahwa dia memiliki kunci publik Bob dan bukan kunci publik Carol. Lagipula, skema ini juga tidak membutuhkan CA harus *online* untuk melakukan verifikasi.

Adanya atribut waktu kadualarsa pada sertifikat digital dimaksudkan agar pengguna mengubah kunci publik (dan kunci privat pasangannya) secara periodik. Makin lama penggunaan kunci, makin besar peluang kunci diserang dan dikriptanalisis. Jika pasangan kunci tersebut diubah, maka sertifikat digital yang lama harus ditarik kembali (*revoked*). Pada sisi lain, jika kunci privat berhasil diketahui pihak lain sebelum waktu kadualarsanya, sertifikat digital harus dibatalkan dan ditarik kembali, dan pengguna harus mengganti pasangan kuncinya.

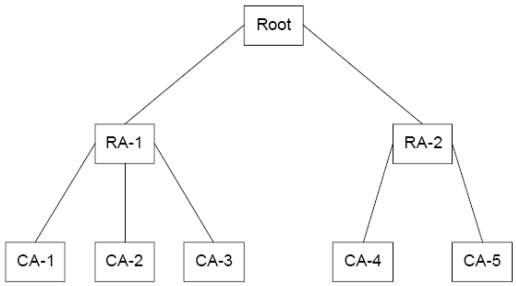
Bagaimana CA memberitahu ke publik bahwa sertifikat digital ditarik? CA secara periodik mengeluarkan CRL (*Certificate Revocation List*) yang berisi nomor seri sertifikat digital yang ditarik. Sertifikat digital yang sudah kadaluarsa otomatis dianggap sudah tidak sah lagi dan ia juga dimasukkan ke dalam CRL. Dengan cara ini, maka CA tidak perlu memberitahu perubahan sertifikat digital kepada setiap orang. Sayangnya, keberadaan CRL menyebabkan pengguna yang memakai sertifikat digital harus memiliki CRL untuk memvalidasi apakah sertifikat tersebut telah ditarik. Sebagai alternatif CRL adalah *Online Certificate Status Protocol* (OCSP), yang memvalidasi sertifikat secara real time. Standard untuk sertifikat telah ditetapkan dan disetujui oleh ITU. Standard tersebut dinamakan X.509 dan digunakan secara luas di internet.

Jika hanya ada satu CA untuk melayani sertifikat digital dari seluruh dunia, jelas CA tersebut akan kolaps karena load yang sangat besar. Solusi yang mungkin adalah mempunyai banyak CA, semua CA dijalankan oleh organisasi yang sama. Setiap CA bekerja dengan menggunakan kunci privat yang sama untuk menandatangani sertifikat. Tapi hal ini menimbulkan masalah; jika kunci privat dicuri, maka ribuan sertifikat digital harus diganti. Lagipula, organisasi mana yang akan mengoperasikan CA? Sukar membayangkan suatu otoritas yang dapat diterima seluruh dunia.

Untuk mengatasi masalah-masalah di atas, maka didefinisikan suatu cara yang berbeda untuk mensertifikasi kunci publik. Cara tersebut dinyatakan di dalam PKI (*Public Key Infrastructure*). PKI mengintegrasikan kriptografi kunci publik dengan sertifikat digital dan CA untuk mengotentikasi pihak-phak dalam suatu transaksi. PKI terdiri atas komponen-komponen:

- pengguna (pemohon sertifikat dan pemakai sertifikat)
- sertifikat digital
- CA
- Direktori (menyimpan sertifikat digital dan CRL)

PKI menyediakan cara penstrukturan komponen-komponen di atas dan mendefinisikan standard bermacam-macam dokumen dan protokol. Bentuk PKI yang sederhana adalah hirarkhi CA dalam struktur pohon pada Gambar 3.

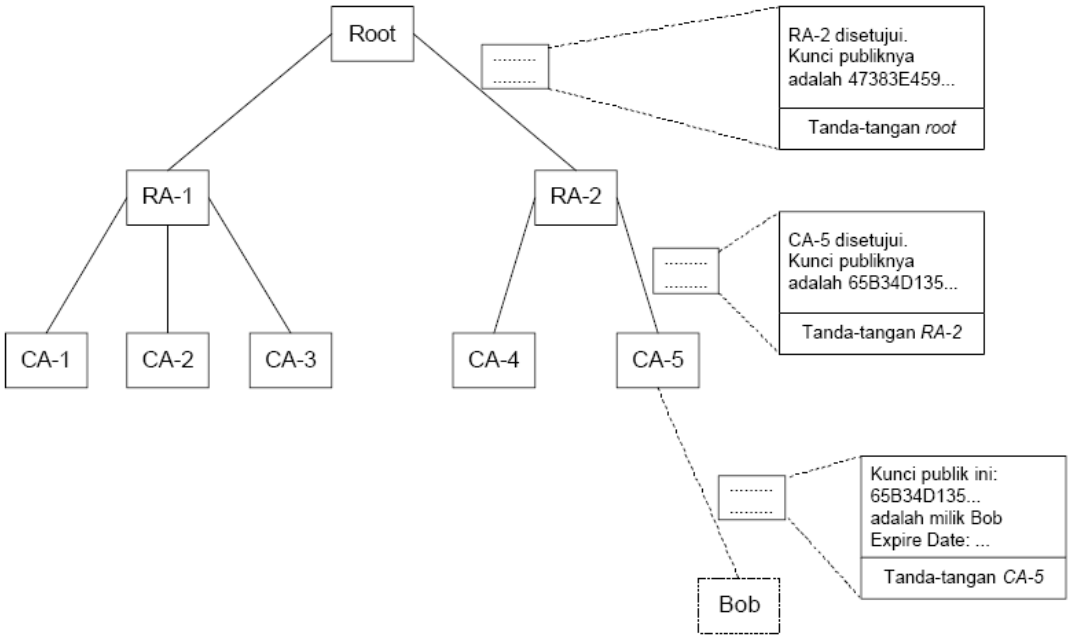


Gambar 3. Hirarkhi CA di dalam PKI

Aras ke-nol adalah *root*. *Root* merupakan *root* certificate authority, yang mana adalah *Internet Policy Registration Authority* (IPRA). *Root* mensertifikasi CA aras satu dengan menggunakan privat *root* yang disebut *root key*.

CA aras satu disebut RA (*Regional Authorities*), yang bertindak sebagai policy creation authority, yaitu oganisasi yang membuat kebijakan untuk memperoleh sertifikat digital. Sebuah RA mungkin mencakup beberapa area geografis, seperti negara bagian, negara, atau benua. RA menandatangai sertifikat digital untuk CA di bawahnya dengan menggunakan kunci privat RA.

CA menandatangani sertifikat digital untuk individu atau organisasi dengan mengguankan kunci privat CA. CA bertanggung jawab untuk otentikasi sertifikat digital, sehingga CA harus memeriksa informasi secara hati-hati sebelum mengeluarkan sertifikat digital. Gambar 4 memperlihatkan rantai sertifikat di dalam PKI.



Gambar 4. Contoh rantai sertifikat digital

Misalkan Alice memerlukan kunci publik Bob untuk berkomunikasi, lalu dia mencari dan menemukan sertifikat Bob ditandatangani oleh CA-5. Alice kemudian mendatangani CA-5 dan meminta bukti legitimasi CA-5. CA-5 merespon dengan memperlihatkan sertifikat digital yang diperoleh dari RA-2, di dalamnya ada kunci publik CA-5 yang ditandangani oleh RA-2. Dengan menggunakan kunci publik CA-5, Alice dapat memverifikasi sertifikat digital Bob yang ditandatangani oleh CA-5 dan mendapatkan hasil bahwa sertifikat Bob sah. Langkah berikutnya, Alice mendatangi RA-2 dan meminta bukti legitimasi RA-5. RA-2 merespon dengan memperlihatkan sertifikat digital yang diperoleh dari *root*, di dalamnya ada kunci publik RA-2 yang ditandantangani oleh *root*. Alice memverifikasi sertifikat digital RA-2 dengan menggunakan kunci publik *root* dan mendapatkan hasil bahwa sertifikat tersebut sah. Sekarang Alice benar-benar yakin bahwa dia sudah memiliki kunci publik Bob. Rantai sertifikat yang menuju ke *root* seperti ini disebut *chain of trust* atau *certification path*. Tentu saja

kita masih mempunyai masalah siapa yang bertindak sebagai *root*. Solusinya bukanlah dengan mempunyai sebuah *root* tunggal, tetapi mempunyai banyak *root*, masing-masing dengan sejumlah RA dan CA-nya sendiri.

3. *Secure Sockets Layer*

Salah satu objek dalam cakupan keamanan adalah keamanan protokol yang dipergunakan dalam komunikasi dan transaksi. Protokol, dalam konteks ini, merupakan suatu set aturan yang dipergunakan oleh komputer-komputer (yang terhubung dalam suatu jaringan) untuk saling berkomunikasi. Protokol yang paling banyak dipergunakan dalam Web merupakan set protokol TCP/IP.

Sayangnya, protokol ini didesain tanpa memperhatikan faktor keamanan data. Oleh karena itu dibutuhkan suatu mekanisme tambahan untuk memperkokoh keamanan protokol ini tanpa harus menggantinya dengan yang lain. Dengan protokol (beserta mekanismenya) yang aman serta implementasi yang benar, tingkat keamanan komunikasi dan transaksi Web dapat ditingkatkan. Salah satu metode penerapan keamanan protokol ini adalah dengan menerapkan *Secure Socket Layer* (SSL).

SSL pada dasarnya sebuah set aturan yang memberitahukan step yang harus dilalui untuk meningkatkan tingkat keamanan komunikasi. Aturan SSL ini memiliki tiga properti dasar:

1. Enkripsi, untuk mengatasi *eavesdropping*. Enkripsi dilakukan setelah *handshake* awal untuk mendefinisikan kunci rahasia. Kriptografi simetris dilakukan untuk enkripsi data (contoh: DES, RC4).
2. Integritas data, untuk mengatasi manipulasi. Transfer data mencakup pengecekan integritas pesan dengan MAC yang telah diberi kunci. Fungsi –fungsi *secure hash* seperti SHA dan MD5 digunakan untuk komputasi MAC.
3. Otentikasi, untuk mengatasi impersonasi. Identitas *peer* dapat diotentikasi dengan kriptografi asimetris (contoh: RSA, Diffie-Hellman).

Perlu dipahami bahwa SSL hanya akan memproteksi data kita pada saat transmisi saja. Hal ini yang disebut *network security*. Protokol SSL sama sekali tidak akan memproteksi data - sebelum maupun sesudah dilakukan komunikasi.

Penerapan protokol SSL memiliki beberapa tujuan. Tujuan-tujuan tersebut, berdasarkan prioritasnya, adalah:

1. Keamanan kriptografis, SSL sebaiknya digunakan dalam koneksi antara dua pihak dalam web.
2. Interoperabilitas, *programmers* yang independen sebaiknya dapat mengembangkan aplikasi yang menggunakan protokol ini, yang akan memungkinkan pertukaran parameter-parameter kriptografis tanpa saling mengetahui kode mereka.
3. Ekstensibilitas, SSL bertujuan untuk menyediakan *framework* dimana metode-metode enkripsi baru dapat diintegrasikan. Hal ini akan sekaligus mencegah keperluan akan protokol baru yang notabene akan memerlukan waktu pengenalan dan mempunyai kelemahan-kelemahan baru. Selain itu, implementasi *security library* yang baru tidak diperlukan lagi.
4. Efisiensi relatif, karena operasi-operasi kriptografis, khususnya operasi-operasi kriptografi kunci publik, cenderung menggunakan *resource* CPU dalam jumlah yang signifikan, protokol SSL telah menginterasikan skema *session caching* (opsional) untuk mengurangi jumlah koneksi yang harus dimulai dari awal. Sebagai tambahan, aktivitas *network* juga dapat dikurangi

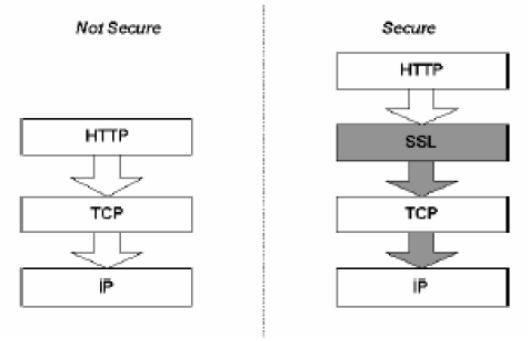
Mekanisme SSL ini terdiri dari beberapa komponen. Salah satu dari komponen yang berkaitan dengan sistem keamanan informasi dan dibahas dalam tulisan ini adalah komponen kriptografi yang terdapat di dalamnya. Komponen ini berfungsi melakukan pengolahan khusus terhadap data yang akan dikomunikasikan agar data tersebut tidak dapat diganggu oleh pihak luar. Komponen kriptografi dalam SSL tersusun dari beberapa algoritma matematis.

Dengan suatu algoritma yang baik serta implementasi yang benar, diharapkan SSL dapat berfungsi sesuai spesifikasi yang diharapkan: cepat, efisien, dan aman. Ketiga faktor inilah

yang akan sangat menentukan nilai keamanan dan keterandalan dalam penggunaan SSL sebagai mekanisme pengamanan di tingkat protokol.

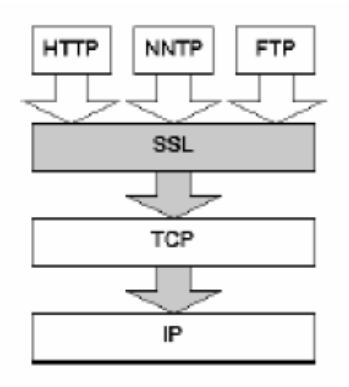
Pengembangan SSL selain oleh Netscape dilakukan oleh OpenSSL. OpenSSL merupakan komunitas yang didasarkan pada model inisiatif OpenSource. Bedanya, komunitas OpenSSL lebih memfokuskan diri pada apa yang disebut pengembangan SSL secara terbuka. Para desainer SSL memutuskan untuk membuat protokol terpisah untuk menangani masalah keamanan. Mereka menambahkan lapisan (*layer*) tambahan pada arsitektur protokol Internet.

Gambar 5 sebelah kiri menunjukkan susunan lapisan protokol dalam komunikasi Web. Gambar sebelah kanan menunjukkan penambahan lapisan SSL untuk memperkokoh keamanan. Dengan bertindak sebagai lapisan baru, SSL tidak banyak mengubah lapisan diatasnya dan dibawahnya. *Interface* aplikasi HTTP akan melihat SSL sebagai suatu lapisan yang hampir sama dengan lapisan TCP. Demikian juga halnya dengan lapisan TCP; ia akan melihat SSL sebagai suatu aplikasi lain yang menggunakan layanannya. Gambar 5 SSL merupakan lapisan terpisah dalam susunan protokol Internet



Gambar 5. Letak SSL dalam susunan protokol Internet

Selain hanya membutuhkan sedikit perubahan pada implementasi yang sudah ada, pendekatan ini memiliki keuntungan lain: SSL mampu mendukung aplikasi lain selain HTTP. Motivasi utama perancangan SSL adalah untuk meningkatkan keamanan Web, namun pada Gambar 6, SSL dapat juga dipergunakan untuk menambahkan keamanan pada aplikasi Internet lainnya seperti *Net News Transfer Protocol* (NNTP) dan *FileTransfer Protocol* (FTP).



Gambar 6. Penanganan keamanan aplikasi lain dengan SSL

3.1. Mekanisme Kerja SSL

Transmission Control Protocol/*Internet Protocol* (TCP/IP) mengatur transportasi dan pembentukan rute data di Internet. Protokol lain, seperti *HyperText Transport Protocol* (HTTP), *Lightweight Directory Access Protocol* (LDAP), atau *Internet Messaging Access Protocol* (IMAP), berjalan 'diatas' TCP/IP. Seluruh protokol tersebut menggunakan TCP/IP untuk mendukung aplikasi tipikal seperti menampilkan halaman web atau menjalankan server e-mail.

Protokol SSL berjalan diatas TCP/IP dan dibawah protokol-level-atas seperti HTTP atau IMAP. Ia menggunakan TCP/IP atas nama protokol-level-atas, dan dalam prosesnya memungkinkan sebuah SSL-*enabled* server untuk mengautentikasi dirinya sendiri kepada SSL-*enabled client*, serta memungkinkan client untuk mengautentikasi dirinya sendiri kepada server, dan memungkinkan kedua mesin untuk membangun sebuah koneksi terenkripsi.

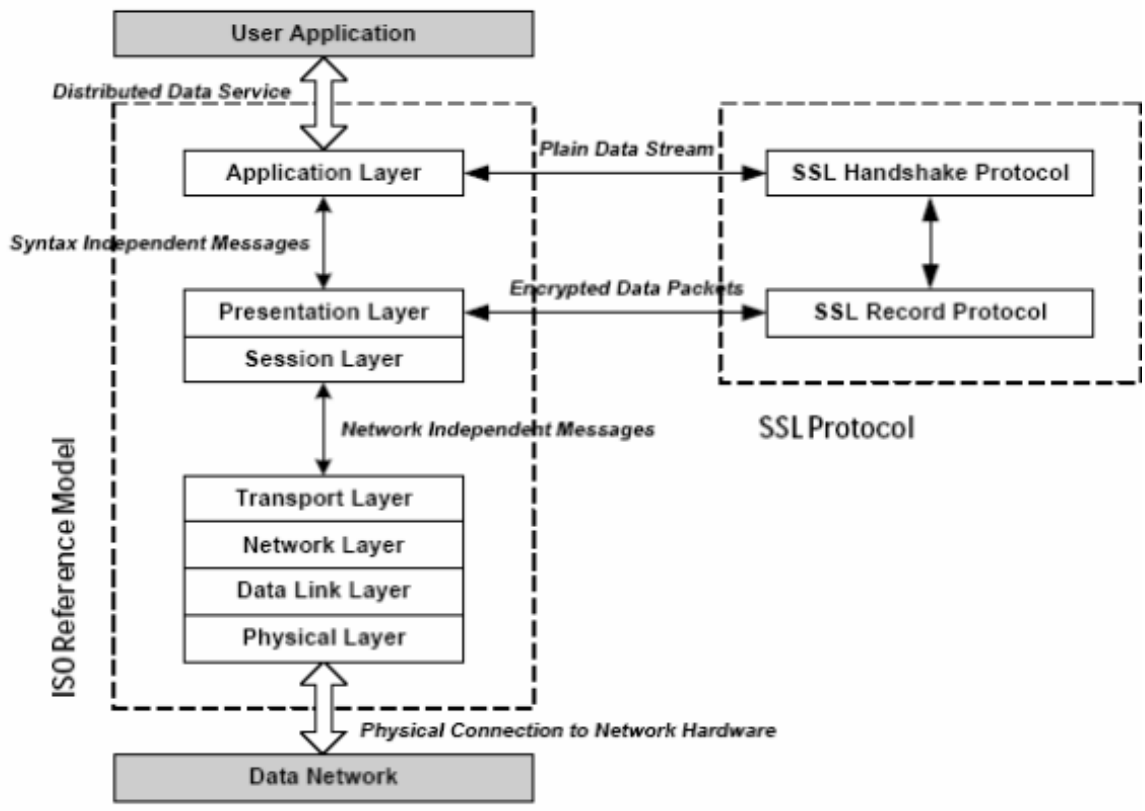
Hal diatas berimplikasi terhadap munculnya komponen fundamental dalam komunikasi di Internet dan di jaringan yang menggunakan TCP/IP :

• Autentikasi *server* SSL memungkinkan seorang pengguna untuk memastikan identitas *server*. Software SSL-*enabled client* dapat menggunakan teknik standar *public-key cryptography* untuk memeriksa keabsahan sertifikat *server* dan public ID yang dikeluarkan oleh *Certificate Authority* (CA). CA ini harus ada dalam daftar CA yang dipercayai oleh *client*. Konfirmasi ini bisa menjadi sangat penting jika, misalnya,

mengirimkan sebuah nomor kartu kredit melalui jaringan dan ingin memastikan identitas *server* penerima nomor tersebut.

• Autentikasi *client* SSL memungkinkan sebuah server untuk memastikan identitas pengguna. Dengan menggunakan teknik yang sama seperti yang digunakan pada autentikasi *server*, software SSL pada *server* dapat memeriksa keabsahan sertifikat *client* dan *public ID*. Peran CA dalam hal ini sama seperti yang telah disebutkan sebelumnya. Konfirmasi ini bisa menjadi sangat penting bagi *server* jika, misalnya, *server* tersebut adalah sebuah bank yang akan mengirimkan informasi finansial rahasia kepada pelanggannya. Pemeriksaan indentitas penerima, dalam hal ini, sangat penting.

• Enkripsi koneksi SSL membutuhkan kondisi dimana semua informasi yang dikirimkan antara client dan server dienkripsikan/didekripsikan oleh kedua belah pihak. Hal ini menyediakan tingkat kerahasiaan yang tinggi. Kerahasiaan merupakan hal penting bagi kedua pihak yang akan melakukan komunikasi pribadi. Sebagai tambahan, semua data yang dikirimkan melalui koneksi SSL dilindungi oleh mekanisme yang akan mendeteksi tampering – yaitu kemampuan untuk mendeteksi apakah data tersebut diubah selama dalam perjalanan.



Gambar 7. Letak SSL dalam model ISO Reference

3.2. *Handshake Protocol*

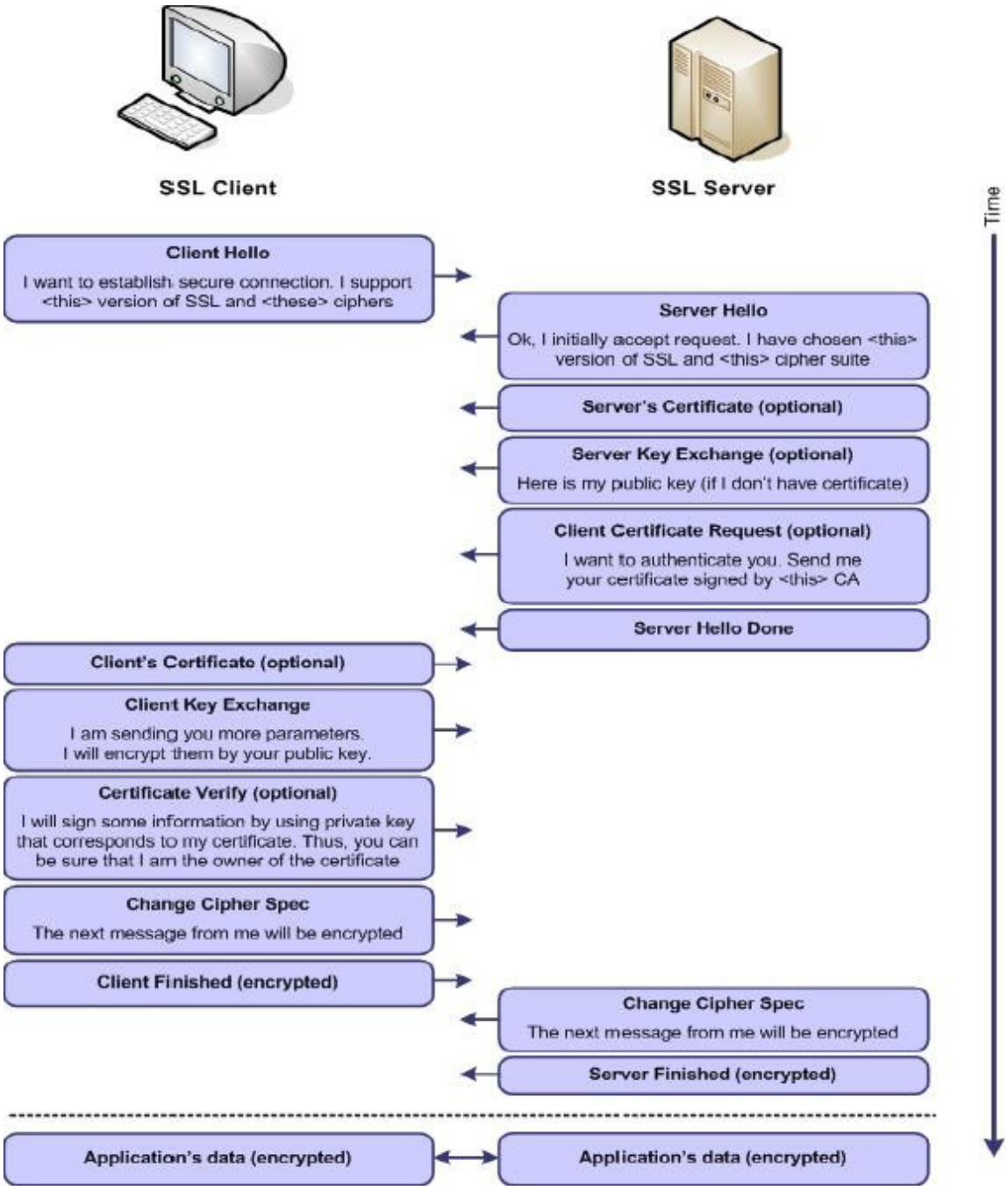
Parameter-parameter kriptografi dari session state dibuat oleh SSL Handshake Protocol, yang beroperasi pada bagian atas dari SSL Record Layer. Ketika sebuah client dan server SSL mulai berkomunikasi pertama kali, mereka membuat kesepakatan versi protokol, memilih algoritma-algoritma kriptografi, saling melakukan otentikasi (opsional), dan menggunakan teknik-teknik *public key encryption* untuk membangkitkan kode-kode yang digunakan bersama
Proses-proses ini dilakukan dalam handshake protocol, yang dapat diringkas menjadi sebagai berikut: *Client* mengirim pesan **client hello** yang akan direspon dengan pesan **server hello**. Jika tidak, akan terjadi *fatal error* dan koneksi akan gagal. Pesan **client hello** dan **server hello** tersebut digunakan untuk pengesetan atribut-atribut berikut: versi protokol, *session ID*, *cipher suite*, metode kompresi. Sebagai tambahan, dua nilai acak `ClientHello.random` dan `ServerHello.random` dibangkitkan dan dipertukarkan.

Setelah itu, *server* akan mengirim sertifikat yang akan diotentikasi. *Server* juga akan mengirim sebuah pesan *server key exchange* jika diperlukan. Jika *server* telah diotentikasi, *server* tersebut dapat meminta sertifikat dari *client*, bergantung pada atribut *cipher suite* yang dipilih. Sekarang *server* akan mengirim pesan **server hello done,** yang menunjukkan bahwa fase pesan hello dalam handshake telah selesai. Lalu *server* akan menunggu respon dari client.

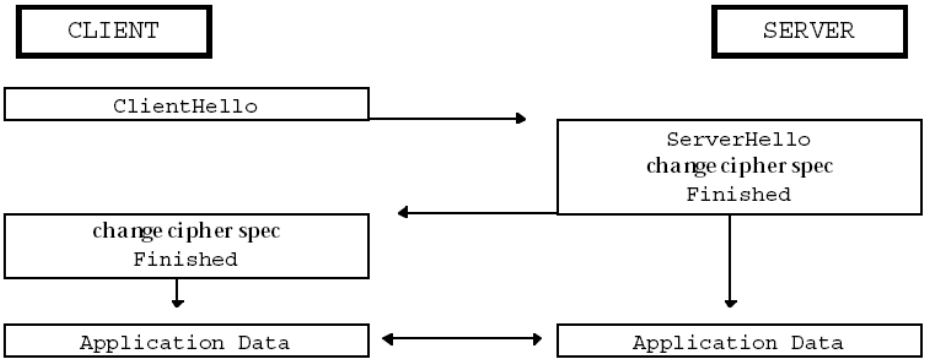
Jika *server* telah mengirim pesan **certificate request,** *client* harus mengirim pesan **certificate** atau pemberitahuan **no certificate**. Sekarang pesan **client key exchange** telah terkirim, dan isi pesan tersebut bergantung pada algoritma kunci publik yang dipilih dalam waktu antara **client hello** dan **server hello**. Jika *client* telah mengirim sertifikat yang dapat ditandatangani, pesan **certificate verify** yang telah ditandatangani dikirim untuk memverifikasi sertifikat tersebut secara eksplisit

Pada saat ini, sebuah pesan **change cipher spec** dikirim oleh *client*, dan *client* menyalin *pending* Cipher Spec ke *current* Cipher Spec. Selanjutnya *client* akan segera mengirim pesan **finished** dengan algoritma-algoritma, kunci-kunci, dan kode-kode rahasia baru. *Server* kemudian akan merespon dengan mengirim pesan **change cipher spec**, men-transfer *pending* Cipher Spec ke *current* Cipher Spec, dan mengirim pesan **finished** dengan Cipher Spec baru. Setelah ini, handshake selesai dan *client* dan *server* dapat mulai saling bertukar data pada *application layer*. Ilustrasi proses ini dapat dilihat pada Gambar 8.

Ketika *client* dan *server* akan menjalankan kembali *session* sebelumnya atau menduplikasi session yang sudah ada, aliran pesan yang terjadi adalah sebagai berikut: *Client* mengirim pesan `ClientHello` menggunakan *Session ID* dari *session* yang ingin dijalankan kembali. *Server* kemudian memeriksa *session cache*-nya dan mencari padanannya. Jika ditemukan dan *server* ingin tersambung kembali dengan *session state* tertentu, *server* akan mengirim sebuah pesan `ServerHello` nilai *Session ID* yang sama. Pada saat ini, *client* dan *server* harus saling mengirim pesan **change cipher spec** dan berlanjut ke pesan **finished**. Setelah itu, barulah *server* dan *client* dapat mulai bertukar data pada *application layer* (Gambar 9). Jika padanan *Session ID* tidak ditemukan, *server* akan membangkitkan *Session ID* baru *client* dan *server* harus kembali melakukan *handshake* secara utuh.



Gambar 8. Protokol handshake pada SSL



Gambar 9. Aliran pesan untuk penjalanan kembali sebuah session

4. *Public Key Infrastructure* pada *Secure Sockets Layer*

Ada empat operasi kriptografi yang diimplementasikan dalam protokol SSL: *digital signing*, *stream cipher encryption*, *block cipher encryption*, dan *public key encryption*. Kunci-kunci kriptografi diimplikasikan oleh *session state* yang berlaku saat itu.

Dalam *digital signing*, fungsi-fungsi hash satu arah digunakan untuk menandatangani algoritma. Sementara itu dalam RSA *signing*, struktur 36-byte dari dua nilai hash (satu SHA dan satu MD5) ditandatangani (dienkripsi dengan kunci privat). Dalam DSS, 20 bytes nilai hash SHA langsung diproses dengan Digital Signing Algorithm, tanpa hashing tambahan. Dalam *stream cipher encryption*, plainteks di XOR-kan dengan nilai yang dibangkitkan pembangkit bilangan *pseudorandom* yang telah diamankan dengan kunci kriptografi.

Dalam *block cipher encryption*, tiap blok plainteks terenkripsi menjadi sebuah blok cipherteks. Karena ukuran plainteks yang beragam, diperlukan padding pada akhit blok-blok pendek dengan pola-pola tertentu, biasanya dengan bit 0. Sementara itu pada *public key encryption*, fungsi-fungsi satu arah dengan "*trapdoors*" rahasia digunakan untuk mengenkripsi data yang akan menjadi output. Data yang dideskripsi dengan kunci publik dari pasangan kunci yang diberikan hanya dapat didekripsi dengan kunci privat, dan juga sebaliknya.

Dalam contoh berikut:

```
stream-ciphered struct {
uint8 field1;
uint8 field2;
digitally-signed opaque hash[20];
} UserType;
```

Isi dari nilai hash digunakan sebagai input untuk algoritma penandatanganan, lalu seluruh strukturnya dienkripsi dengan sebuah *stream cipher*.

PKI pada SSL diterapkan dalam otentikasi. Lebih spesifik lagi, algoritma-algoritma PKI digunakan untuk otentikasi client dan server dan mambangkitkan *shared keys* dan kode rahasia. Dalam proses pemenuhan kriteria pada properti ini, algoritma-algoritma yang digunakan adalah RSA (Rivest-Shamir-Adleman Algorithm) dan Diffie-Hellman. Berikut adalah implementasi tiap algoritma pada protokol SSL. Kode sumber diperoleh dari komunitas OpenSSL.

4.1. RSA

Ketika RSA digunakan untuk otentikasi *server* dan pertukaran kunci, kode rahasia dengan panjang 48-byte dibangkitkan oleh *client*, dienkripsi dengan kunci publik milik *server*, dan dikirimkan ke *server*. Lalu *server* menggunakan kunci privat miliknya untuk mendekripsi kunci rahasia tersebut. Kedua pihak lalu mengkonversi kode rahasia tersebut.

Struktur data untuk *methods* pada RSA:

```
typedef struct rsa_meth_st
        {
        const char *name;
        int (*rsa_pub_enc)(int flen,const
unsigned char *from,
          unsigned char *to,
          RSA *rsa,int padding);
        int (*rsa_pub_dec)(int flen,const
unsigned char *from,
          unsigned char *to,
          RSA *rsa,int padding);
        int (*rsa_priv_enc)(int flen,const
unsigned char *from,
          unsigned char *to,
          RSA *rsa,int padding);
        int (*rsa_priv_dec)(int flen,const
unsigned char *from,
          unsigned char *to,
          RSA *rsa,int padding);
        int (*rsa_mod_exp)(BIGNUM *r0,const
BIGNUM *I,RSA *rsa); /* Can be null */
        int (*bn_mod_exp)(BIGNUM *r, const
BIGNUM *a, const BIGNUM *p,
          const BIGNUM *m, BN_CTX *ctx,
BN_MONT_CTX *m_ctx); /* Can be null */
        int (*init)(RSA *rsa);        /*
called at new */
        int (*finish)(RSA *rsa);      /*
called at free */
        int flags;                    /*
RSA_METHOD_FLAG_* things */
        char *app_data;
        int (*rsa_sign)(int type,
          const unsigned char *m, unsigned
int m_length,
          unsigned char *sigret, unsigned int
*siglen, const RSA *rsa);
        int (*rsa_verify)(int dtype,
          const unsigned char *m, unsigned
int m_length,
          unsigned char *sigbuf, unsigned int
siglen, const RSA *rsa);

        } RSA_METHOD;
```

Enkripsi dengan kunci publik:

```
static int RSA_eay_public_encrypt(int flen,
const unsigned char *from,
             unsigned char *to, RSA *rsa,
int padding)
        {
        BIGNUM f,ret;
        int i,j,k,num=0,r= -1;
        unsigned char *buf=NULL;
        BN_CTX *ctx=NULL;
        if (BN_num_bits(rsa->n) >
OPENSSL_RSA_MAX_MODULUS_BITS)
                {
       RSAerr(RSA_F_RSA_EAY_PUBLIC_DECRYPT,
RSA_R_MODULUS_TOO_LARGE);
                return -1;
                }
        if (BN_ucmp(rsa->n, rsa->e) <= 0)
                {
 RSAerr(RSA_F_RSA_EAY_PUBLIC_DECRYPT,
RSA_R_BAD_E_VALUE);
                return -1;
                }
        /* for large moduli, enforce
exponent limit */
        if (BN_num_bits(rsa->n) >
OPENSSL_RSA_SMALL_MODULUS_BITS)
                {
                if (BN_num_bits(rsa->e) >
OPENSSL_RSA_MAX_PUBEXP_BITS)
                        {
        RSAerr(RSA_F_RSA_EAY_PUBLIC_DECRYPT
, RSA_R_BAD_E_VALUE);
                        return -1;
                        }
                }
                BN_init(&f);
        BN_init(&ret);
        if ((ctx=BN_CTX_new()) == NULL)
goto err;
        num=BN_num_bytes(rsa->n);
        if ((buf=(unsigned char
*)OPENSSL_malloc(num)) == NULL)
                {
```

```
        RSAerr(RSA_F_RSA_EAY_PUBLIC_ENCRYPT,E
RR_R_MALLOC_FAILURE);
                goto err;
                }
        switch (padding)
                {
        case RSA_PKCS1_PADDING:
        i=RSA_padding_add_PKCS1_type_2(buf,nu
m,from,flen);
                break;
#ifndef OPENSSL_NO_SHA
        case RSA_PKCS1_OAEP_PADDING:

i=RSA_padding_add_PKCS1_OAEP(buf,num,from,fle
n,NULL,0);
                break;
#endif
        case RSA_SSLV23_PADDING:

        i=RSA_padding_add_SSLv23(buf,num,from
,flen);
                break;
        case RSA_NO_PADDING:

        i=RSA_padding_add_none(buf,num,from,f
len);
                break;
        default:
        RSAerr(RSA_F_RSA_EAY_PUBLIC_ENCRYPT,R
SA_R_UNKNOWN_PADDING_TYPE);
                goto err;
                }
        if (i <= 0) goto err;
        if (BN_bin2bn(buf,num,&f) == NULL)
goto err;


        if (BN_ucmp(&f, rsa->n) >= 0)
                {
                /* usually the padding
functions would catch this */
        RSAerr(RSA_F_RSA_EAY_PUBLIC_ENCRYPT,R
SA_R_DATA_TOO_LARGE_FOR_MODULUS);
                goto err;
                }
        if (rsa->flags &
RSA_FLAG_CACHE_PUBLIC)
                {
                if
(!BN_MONT_CTX_set_locked(&rsa-
>_method_mod_n,

        CRYPTO_LOCK_RSA, rsa->n, ctx))
                        goto err;
                }
        if (!rsa->meth-
>bn_mod_exp(&ret,&f,rsa->e,rsa->n,ctx,
                rsa->_method_mod_n)) goto
err;
        j=BN_num_bytes(&ret);
        i=BN_bn2bin(&ret,&(to[num-j]));
        for (k=0; k<(num-i); k++)
                to[k]=0;
        r=num;
err:
        if (ctx != NULL) BN_CTX_free(ctx);
        BN_clear_free(&f);
        BN_clear_free(&ret);
        if (buf != NULL)
                {
                OPENSSL_cleanse(buf,num);
                OPENSSL_free(buf);
                }
        return(r);
        }
```

Dekripsi dengan kunci privat:

```
static int RSA_eay_private_decrypt(int
flen, const unsigned char *from,
            unsigned char *to, RSA *rsa,
int padding)
        {
        BIGNUM f,ret;
        int j,num=0,r= -1;
        unsigned char *p;
        unsigned char *buf=NULL;
        BN_CTX *ctx=NULL;
        int local_blinding = 0;
        BN_BLINDING *blinding = NULL;
        BN_init(&f);
        BN_init(&ret);
        ctx=BN_CTX_new();
```

```
        if (ctx == NULL) goto err;
        num=BN_num_bytes(rsa->n);
        if ((buf=(unsigned char
*)OPENSSL_malloc(num)) == NULL)
                {
        RSAerr(RSA_F_RSA_EAY_PRIVATE_DECRYPT,
ERR_R_MALLOC_FAILURE);
                goto err;
                }
        /* This check was for equality but
PGP does evil things
         * and chops off the top '0' bytes */
        if (flen > num)
                {
        RSAerr(RSA_F_RSA_EAY_PRIVATE_DECRYPT,
RSA_R_DATA_GREATER_THAN_MOD_LEN);
                goto err;
                }
        /* make data into a big number */
        if (BN_bin2bn(from,(int)flen,&f) ==
NULL) goto err;
        if (BN_ucmp(&f, rsa->n) >= 0)
                {
        RSAerr(RSA_F_RSA_EAY_PRIVATE_DECRYPT,
RSA_R_DATA_TOO_LARGE_FOR_MODULUS);
                goto err;
                }
        if (!blinding_helper(rsa, ctx))
                goto err;
        blinding = rsa->blinding;
                if (!(rsa->flags &
RSA_FLAG_NO_BLINDING))
                {
                if (blinding == NULL)
                        {
        RSAerr(RSA_F_RSA_EAY_PRIVATE_DECRYPT,
ERR_R_INTERNAL_ERROR);
                        goto err;
                        }
                }
        if (blinding != NULL)
                {
                if (1)
                        {
                        /* we need a local
one-time blinding factor */
                        blinding =
setup_blinding(rsa, ctx);
                        if (blinding ==
NULL)
                                goto err;
                        local_blinding = 1;
                        }
                }
        if (blinding)
                if (!BN_BLINDING_convert(&f,
blinding, ctx)) goto err;
        /* do the decrypt */
        if ( (rsa->flags &
RSA_FLAG_EXT_PKEY) ||
                ((rsa->p != NULL) &&
                (rsa->q != NULL) &&
                (rsa->dmp1 != NULL) &&
                (rsa->dmq1 != NULL) &&
                (rsa->iqmp != NULL)) )
                {
                if (!rsa->meth-
>rsa_mod_exp(&ret,&f,rsa)) goto err;
                }
        else
        {BIGNUM local_d;
        BIGNUM *d = NULL;
                if (!(rsa->flags &
RSA_FLAG_NO_EXP_CONSTTIME))
                        {
                        d = &local_d;
                        BN_with_flags(d,
rsa->d, BN_FLG_EXP_CONSTTIME);
                        }
                else
                        d = rsa->d;
                if (!rsa->meth-
>bn_mod_exp(&ret,&f,d,rsa->n,ctx,NULL))
                        goto err;
                }
        if (blinding)
                if
(!BN_BLINDING_invert(&ret, blinding, ctx))
goto err;
        p=buf;
        j=BN_bn2bin(&ret,p); /* j is only
used with no-padding mode */
        switch (padding)
```

```
                {
        case RSA_PKCS1_PADDING:

        r=RSA_padding_check_PKCS1_type_2(to,n
um,buf,j,num);
                break;
#ifndef OPENSSL_NO_SHA
         case RSA_PKCS1_OAEP_PADDING:

r=RSA_padding_check_PKCS1_OAEP(to,num,buf,j,n
um,NULL,0);
                 break;
#endif
        case RSA_SSLV23_PADDING:

        r=RSA_padding_check_SSLv23(to,num,buf
,j,num);
                break;
        case RSA_NO_PADDING:

        r=RSA_padding_check_none(to,num,buf,j
,num);
                break;
        default:

        RSAerr(RSA_F_RSA_EAY_PRIVATE_DECRYPT,
RSA_R_UNKNOWN_PADDING_TYPE);
                goto err;
                }
        if (r < 0)

        RSAerr(RSA_F_RSA_EAY_PRIVATE_DECRYPT,
RSA_R_PADDING_CHECK_FAILED);


err:
        if (ctx != NULL) BN_CTX_free(ctx);
        BN_clear_free(&f);
        BN_clear_free(&ret);
        if (local_blinding)
                BN_BLINDING_free(blinding);
        if (buf != NULL)
                {
                OPENSSL_cleanse(buf,num);
                OPENSSL_free(buf);
                }
        return(r);
        }
```

4.2. Diffie-Hellman

Pada protokol SSL, dilakukan komputasi Diffie-Hellman konvensional. Parameter-parameter algoritma ini ditentukan oleh *server*, dapat hanya digunakan untuk satu kali komputasi atau menjadi bagian dari sertifikat milik *server*.

Struktur data untuk *methods* pada algoritma Diffie-Hellman:

```
typedef struct dh_method {
        const char *name;
        /* Methods here */
        int (*generate_key)(DH *dh);
        int (*compute_key)(unsigned char
*key,const BIGNUM *pub_key,DH *dh);
        int (*bn_mod_exp)(const DH *dh,
BIGNUM *r, const BIGNUM *a, const BIGNUM
*p, const BIGNUM *m, BN_CTX *ctx,
BN_MONT_CTX *m_ctx); /* Can be null */
        int (*init)(DH *dh);
        int (*finish)(DH *dh);
        int flags;
        char *app_data;
} DH_METHOD;
```

Implemantasi fungsi komputasi kunci:

```
static int compute_key(unsigned char *key,
const BIGNUM *pub_key, DH *dh)
        {
        BN_CTX *ctx;
        BN_MONT_CTX *mont=NULL;
        BIGNUM *tmp;
        int ret= -1;
        if (BN_num_bits(dh->p) >
OPENSSL_DH_MAX_MODULUS_BITS)
                {
        DHerr(DH_F_DH_COMPUTE_KEY,DH_R_MODU
LUS_TOO_LARGE);
                goto err;
                }
        ctx = BN_CTX_new();
        if (ctx == NULL) goto err;
        BN_CTX_start(ctx);
        tmp = BN_CTX_get(ctx);


        if (dh->priv_key == NULL)
```

```
                {
        DHerr(DH_F_DH_COMPUTE_KEY,DH_R_NO_PRI
VATE_VALUE);
                goto err;
                }
        if (dh->flags & DH_FLAG_CACHE_MONT_P)
                {
                mont = BN_MONT_CTX_set_locked(
                                (BN_MONT_CTX
**)&dh->method_mont_p,
                                CRYPTO_LOCK_DH,
dh->p, ctx);
                if ((dh->flags &
DH_FLAG_NO_EXP_CONSTTIME) == 0)
                        {
        BN_set_flags(dh->priv_key,
BN_FLG_EXP_CONSTTIME);
                        }
                if (!mont)
                        goto err;
                }
        if (!dh->meth->bn_mod_exp(dh, tmp,
pub_key, dh->priv_key,dh->p,ctx,mont))
                {
        DHerr(DH_F_DH_COMPUTE_KEY,ERR_R_BN_LI
B);
                goto err;
                }
        ret=BN_bn2bin(tmp,key);
err:
        if (ctx != NULL)
                {
                BN_CTX_end(ctx);
                BN_CTX_free(ctx);
                }
        return(ret);
        }
```

Kode sumber OpenSSL yang digunakan memiliki nomor versi openssl-0.9.7l. Hingga saat penulisan, versi OpenSSL tersebut belum berada dalam status *stable*. OpenSSL, seperti yang telah dijelaskan di atas, merupakan usaha kolaboratif untuk membangun suatu *toolkit* OpenSource yang dapat memfungsikan *Secure Socket Layer* (SSL v2/v3) dan untuk membangun suatu *cryptography library* umum yang kuat. Dengan demikian, OpenSSL dituntut untuk dapat difungsikan sepenuhnya seperti implementasi SSL lainnya dan memiliki keamanan yang kuat, yang memenuhi berbagai macam tingkat kebutuhan akan komunikasi data yang aman. Dari implementasi pada versi OpenSSL tersebut, terlihat bahwa implementasi PKI pada OpenSSL telah memenuhi spesifikasi SSL yang dibuat oleh pelopor perancang dan pengembang SSL, yaitu Netscape.

5. Prospek *Secure Sockets Layer*

Pengetesan suatu algoritma atau protokol tidak harus selalu dilakukan oleh tim pengembang. Contohnya adalah pengetesan protokol SSL ini. Pada tahun 1995, Netscape mempublikasikan semacam sayembara untuk membobol protokol ini. Sayembara ini disebut SSL Challenge. Hasil dari usaha pembobolan tersebut menjadi umpan balik bagi para perancang protokol untuk kemudian diperbaiki. Berikut adalah usaha-usaha pembobolan *security* yang terkait dengan SSL selengkapnya:

14 Juli 1996: Hal Finney melakukan *posting* dari SSL challenge: sebuah *record* dari "*secure*" Netscape *session* yang di enkrip menggunakan algoritma RC4-128-EXPORT-40.

15 Agustus 1996: Dua kelompok berhasil memecahkan SSL Challenge tersebut. Kelompok yang pertama adalah David Byers dan Eric Young, bekerjasama dengan Adam Back berhasil memecahkannya pada pukul 10:43. Dua jam kemudian, Damien Doligez berhasil memecahkan pada pukul 12:23.

Menurut Damien, untuk memecahkan SSL dengan 40-bit *key* dibutuhkan sekitar 40 buah *high end* Pentium PC yang di kaitkan dalam sebuah jaringan komputer sehingga dapat dipecahkan secara paralel untuk memperkuat kemampuan komputasinya. Perlu diketahui bahwa Damien menggunakan sekitar 120 komputer secara paralel untuk memecahkannya. Waktu yang dibutuhkan untuk memecahkan *challenge* tersebut dengan kekuatan 120 komputer adalah sekitar 8 hari (estimasi maksimum akan dibutuhkan waktu 15 hari).

17 Agustus 1996: Netscape mengirimkan respon resmi mereka. Walaupun sebagian besar orang kurang setuju (karena underestimate) dengan angka US$10,000 untuk biaya pembobolan algoritma RC4-128 di SSL mereka.

Sejak saat ini mulai ada usaha-usaha dari para *cyberpunk* untuk bersatu & membentuk sebuah jaringan "*key cracking ring*" untuk melihat seberapa cepat kita kita dapat membongkar sesion yang di enkrip dengan menggunakan banyak mesin di Internet secara paralel dari *ring* tersebut.

19 Agustus 1996: Hal Finney kembali melakukan *posting* dari SSL Challenge yang kedua kepada para *cyberpunk* untuk "*key cracking ring*". Kegiatan ini di koordinir oleh Adam Back dan Piete Brooks.

24 Agustus 1996: "*Key cracking ring*" yang merupakan kumpulan komputer yang bekerja secara paralel melalui jaringan Internet mulai bekerja pada *challenge* tanggal 19 Agustus yang dilakukan pada pukul 18:00 GMT. Dalam waktu kurang dari 32 jam hasilnya diperoleh!

17 September 1996: Ian Goldberg dan David Wagner berhasil membobol pseudo-random number generator dari Netscape Navigator 1.1. Mereka berhasil masuk ke session key dalam waktu beberapa jam dari sebuah workstation.

Seperti kita lihat di atas bahwa untuk memecahkan sebuah *secure session* SSL dengan 40-bit key ternyata membutuhkan waktu yang lama (berhari-hari) dengan kekuatan 120 komputer melakukan komputasi secara paralel. Hal ini menunjukan betapa sulitnya untuk memecahkan sebuah *secure session*. Dengan versi SSL saat ini yang lebih aman dengan panjang 128-bit maka semakin sulit untuk menembus proteksi yang akan ada. Terbukti sampai saat ini belum ada berita yang menyatakan keberhasilan pembobolan SSL dengan panjang kunci 128-bit ini.

Dilihat dari penerapan PKI pada SSL, analisis dapat didasarkan pada penggunaan standar X.509 untuk sertifikasi. Banyak kritik yang dialamatkan ke standar ini. PKI dengan standar ini mengasumsikan *single global namespace* dan *namespace* tersebut memiliki interoperabilitas dengan *multiple namespace*. Hal ini dapat menimbulkan konflik. Selain itu, dalam revokasi, pemberian *timestamps* seharusnya menjadi salah satu fitur.

Dengan beberapa kualifikasi, arsitektur-X.509v3 didesain untuk bekerja dalam pandangan keamanan "*absolute trust*" yang sederhana. Ini bertolak belakanng dengan pandangan keamanan yang seharusnya, yaitu pandangan "*risk management*". Banyak skema operasional yang hanya mempunyai satu layer CA, dan basis kepercayaan terhadap CA-CA tersebut adalah *blind trust*.

Lebih lanjut lagi, aplikasi standar X.509 untuk PKI pada lingkup umum, seperti sertifikasi Verisign yang terimplementasi pada *web browsers* komersial adalah implementasi versi relaxed, yang kurang mengindahkan kaidah-kaidah dalam standar X.509 yang resmi. Selain itu, ditemukan juga kerumitan dan waktu tunggu yang cukup signifikan pada beberapa kasus pembangkitan kunci, akuisisi sertifikat, dan manajemen sertifikat, bahkan oleh teknisi ahli. Kendati begitu, kelemahan CA tersebut cukup tertutupi dengan keamanan fisik dan elektronik dari fasilitas-fasilitas yang digunakan untuk membangkitkan sertifikat.

Saat ini telah ada suksesor dari SSL, yaitu TLS (*Transport Layer Security*). TLS ini merupakan pengembangan dari SSL, sehingga dapat diaplikasikan pada transaksi *client-server* apa saja. Satu perbedaan yang penting adalah bahwa TLS mengaplikasikan algoritma *Keyed-Hashing for Message Authentication Code* (HMAC) yang menggantikan algoritma MAC pada SSL. Cara menghasilkan nilai pengecekan integritas pada HMAC sama dengan MAC, hanya saja fungsi hash pada HMAC telah mengalami konstruksi, sehingga lebih sulit untuk dipecahkan dibandingkan fungsi hash pada MAC pada SSL.

TLS sangat umum digunakan sebagai *setting* dalam *email programs*. Selain perbedaan diatas, TLS mempunyai kelebihan yaitu kemampuannya untuk bekerja pada *ports* yang berbeda-beda. Walaupun hanya memiliki sedikit perbedaan, TLS tidak memiliki interoperabiitas dengan SSL. Artinya mesin atau aplkasi yang menggunakan protokol TLS tidak dapat beroperasi dengan mesin atau aplikasi yang menggunakan protokol SSL.

Suksesi SSL ke TLS ini dilakukan untuk "mengimbangi" HTTP 1.1 dan untuk mendukung semua *ciphers* yang populer. Untuk

clients, SSL 3.0 masih menjadi standar. Walaupun protokol default-nya adalah TLS, pada prakteknya protokol akan mengalami *downgrade* secara rutin ke SSL 3.0.

Dari fakta-fskta diatas, dapat dikatakan bahwa SSL masih banyak digunakan sebagai protokol *default.* Selain itu, dalam praktiknya, protokol ini masih tergolong aman untuk transaksi dan transmisi data *client-server* di Internet

6. Simpulan

Dari kajian yang dilakukan penerapan PKI pada SSL diperoleh beberapa simpulan:

1. Berbagai algoritma *security* yang dikembangkan untuk mengamankan transmisi data melalui jaringan komputer publik Internet mampu untuk mengamankan saluran transmisi sehingga cukup aman.

2. SSL hanya akan memproteksi data kita pada saat transmisi saja. Hal ini yang disebut *network security*. Protokol SSL sama sekali tidak akan memproteksi data sebelum maupun sesudah dilakukan komunikasi.

3. Penerapan PKI akan manimbulkan konsekuensi pemakaian *resource* yang cukup besar, namun memiliki tingkat kerumitan komputasi yang tinggi, yang meningkatkan keamanan algoritma.

4. Penerapan PKI pada SSL menjamin keamanan transmisi data pada Internet dengan baik. Kelemahan-kelemahan yang ada, walaupun cukup fundamental, pada prakteknya dapat ditolerir.

5. SSL masih banyak digunakan, karena perbedaannya dengan suksesornya (TLS) tidak terlalu signifikan.

6. Untuk membobol sebuah session yang telah di amankan, dibutuhkan banyak waktu & komputer yang bekerja secara paralel.

7. Tidak ada sistem yang benar-benar aman. Optimasi algoritma, perbaikan mekanisme keamanan dan evaluasi harus selalu dilakukan, apapun sistem keamanannya.

DAFTAR PUSTAKA

[1] Clarke, Roger. (2000). Conventional Public Key Infrastructure: An Artefact Ill-Fitted to the Needs of the Information Society. Xamax Consultancy Pty Ltd.

[2] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] Netscape Communications Corporation (1996). SSL 3.0 Specification. Netscape.

[4] Purbo, Onno. (2000). Usaha Mengamankan Internet bagi Dunia Usaha. Computer Network Research Group ITB.

[5] Quast, William. (2006). SSL and TLS: Secure Communication over Unsecure Medium.

[6] Rahardjo, Budi. (2005). Keamanan Sistem Informasi Berbasis Internet.

[7] Simalango, Indra. (2004). Perancangan Cryptoboard dengan Mengimplementasikan OpenSSL-0.9.7 pada Single Board Computer. Departemen Teknik Elektro, Institut Teknologi Bandung.