Bahan kuliah IF4020 Kriptografi

Secure Socket Layer (SSL) dan Transport Layer Security (TLS)

Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
ITB - 2025


SSL
SSL
https://www.

Referensi

1. John Mitchell, *SSL / TLS Case Study*

2. SMU-CSE 5349/7349, *SSL/TLS*

Protokol

- **Protokol**: aturan yang berisi rangkaian langkah-Langkah, yang melibatkan dua atau lebih orang, yang dibuat untuk menyelesaikan suatu kegiatan.

- Protokol kriptografi adalah protokol yang menggunakan algoritma kriptografi.

- Orang yang berpartisipasi dalam protokol kriptografi memerlukan protokol tersebut misalnya untuk:
  - berbagi komponen untuk menghitung sebuah nilai rahasia,
  - membangkitkan rangkaian bilangan acak,
  - meyakinkan identitas orang lainnya (otentikasi),
  - mengenkripsi dan dekripsi pesan
  - dll

Contoh-contoh protokol kriptografi:

1. Secure Socket Layer (SSL)
2. IPSec (Internet Protocol Security)
3. Kerberos
4. Protokol pertukaran kunci Diffie-Hellman
5. Transport Layer Security (TLS)

- Contoh-contoh protokol kriptografi dalam praktek:
  1. Secure Socket Layer (SSL)
  2. IPSec (Internet Protocol Security)
  3. Kerberos
  4. Transport Layer Security (TLS)

- Protokol kriptografi dibangun dengan melibatkan beberapa algoritma kriptografi.

- Sebagian besar protokol kriptografi dirancang untuk dipakai antara dua entitas yang berkomunikasi.

- Tetapi ada juga beberapa protokol yang dirancang untuk dipakai oleh lebih dari dua entitas (misalnya pada aplikasi *teleconferencing*)

Apakah SSL / TLS itu?

- Protokol *Transport Layer Security* (TLS)
  - Standar *de facto* untuk keamanan Internet
  - "Tujuan utama protokol TLS adalah untuk memberikan privasi dan integritas data antara dua aplikasi yang berkomunikasi"
  - Dalam praktiknya, digunakan untuk melindungi informasi yang dikirim antara browser dan server Web

- Berdasarkan protokol *Secure Sockets Layers* (SSL) versi 3.0
  - Desain protokol yang sama, algoritma yang berbeda

- Di-*deploy* pada hamper setiap web browser

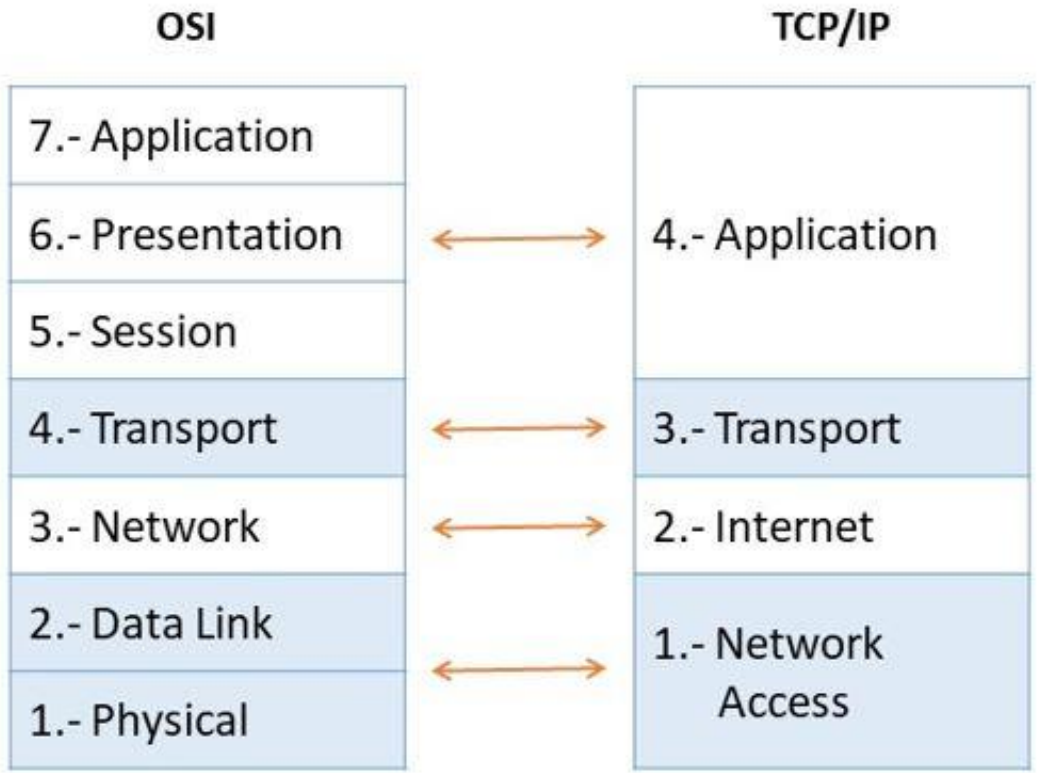
Sumber: John Mitchell, SSL / TLS Case Study

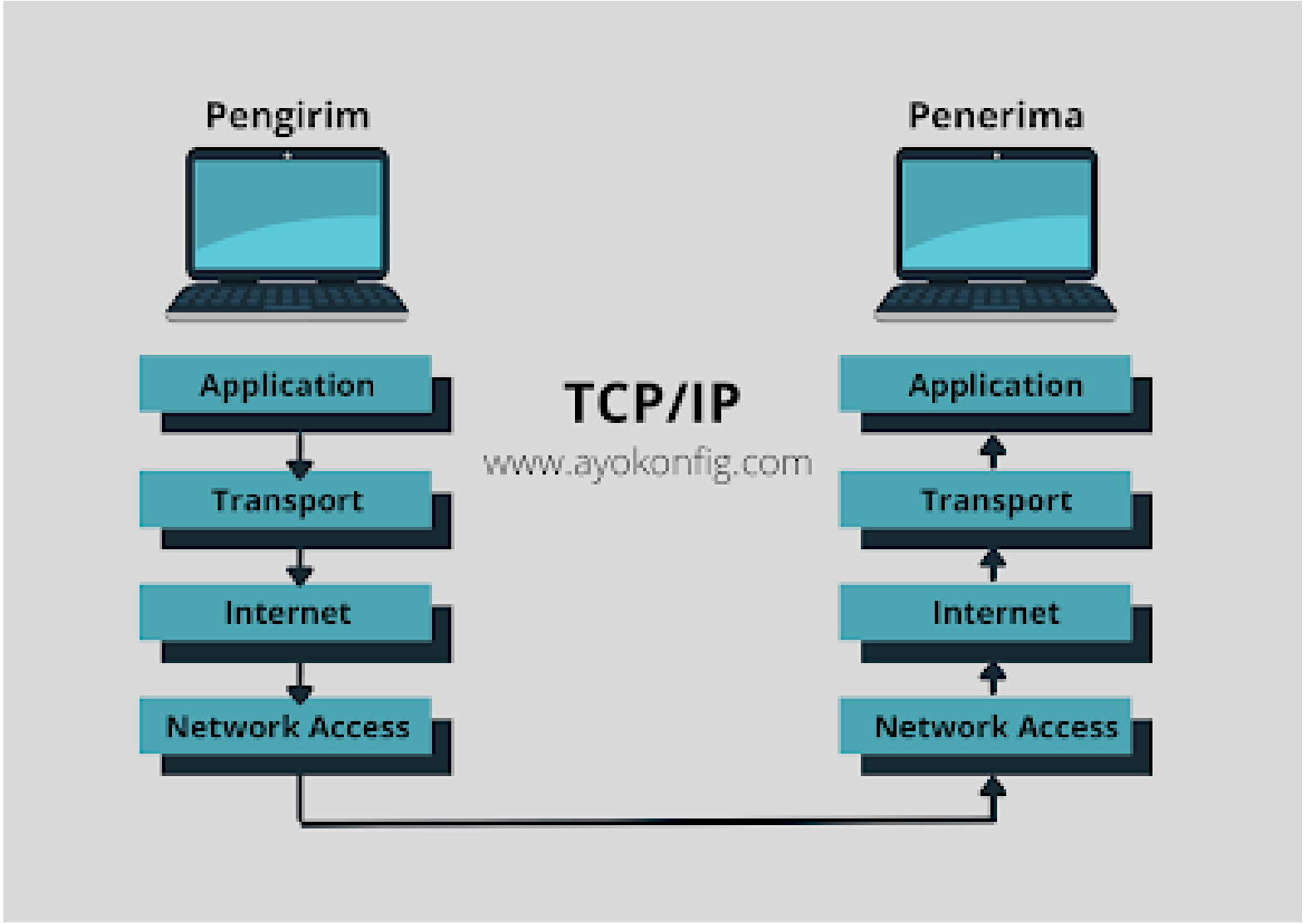
Sejarah Protokol SSL/TLS

- SSL 1.0
  - Desain internal Netscape, awal 1994?
  - Hilang tanpa ada kabar
- SSL 2.0
  - Diterbitkan oleh Netscape, November 1994
  - Beberapa masalah keamanan
- SSL 3.0
  - Dirancang oleh Netscape dan Paul Kocher, November 1996
- TLS 1.0
  - Standar internet berdasarkan SSL 3.0, Januari 1999
  - Tidak dapat dioperasikan dengan SSL 3.0
- TLS 1.1 (2006), TLS 1.2 (2008), TLS 1.3 (2018).

TCP/IP

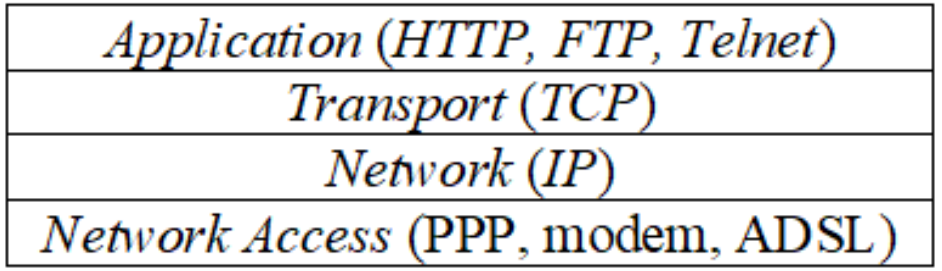
- *Transmission Control Protocol/Internet Protocol* (TCP/IP) adalah standard protokol yang digunakan untuk menghubungkan komputer dengan jaringan sehingga membentuk jaringan yang lebih besar (WAN atau Internet).





Sumber gambar: https://www.ayokonfig.com/2021/12/pengertian-tcpip-beserta-fungsi.html

Cara kerja TCP/IP



- Kebanyakan transmisi pesan di Internet dikirim sebagai kumpulan potongan pesan yang disebut **paket**.
- *IP* bertanggung jawab untuk merutekan paket (lintasan yang dilalui oleh paket).
- Pada sisi penerima, *TCP* memastikan bahwa suatu paket sudah sampai, menyusunnya sesuai nomor urut, dan menentukan apakah paket tiba tanpa mengalami perubahan.
- Jika paket mengalami perubahan atau ada data yang hilang, *TCP* meminta pengiriman ulang.
- Terlihat bahwa *TCP/IP* tidak memiliki pengamanan komunikasi yang bagus. Pesan ditransmisikan dalam bentuk plainteks.
- *TCP/IP* juga tidak dapat mengetahui jika pesan diubah oleh pihak ketiga (*man-in-the-middle attack*).

Keamanan Web

- *Secure Socket Layer* (*SSL*) adalah protokol yang digunakan untuk *browsing web* secara aman. *SSL* bertindak sebagai protokol yang mengamankan komunikasi antara *client* dan *server*.

- *SSL* beroperasi antara lapisan *Application* dan lapisan *Transport. SSL* seolah-olah berlaku sebagai lapisan baru (*security layer*) antara kedua lapisan tersebut.

Application (*HTTP, FTP, Telnet*)
Security (*SSL*)
Transport (*TCP*)
Network (*IP*)
Network Access (PPP, modem, ADSL)

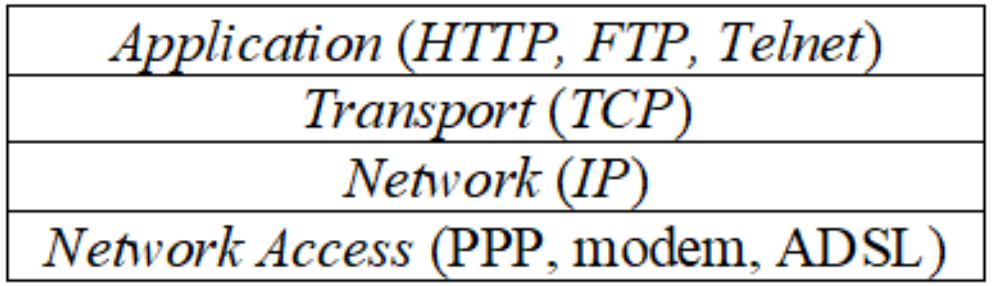
Gambar Lapisan (dan protokol) untuk *browsing* dengan *SSL*

- *SSL* dikembangkan pertama kali oleh *Netscape Communations* pada tahun 1994.

- SSL didefinisikan di dalam RFC2246: http://www.ietf.org/rfc/rfc2246.txt

- Implementasi *open-source* SSL tersedia di: http://www.openssl.org/

Cara kerja TCP/IP



- Kebanyakan transmisi pesan di Internet dikirim sebagai kumpulan potongan pesan yang disebut **paket**.
- *IP* bertanggung jawab untuk merutekan paket (lintasan yang dilalui oleh paket).
- Pada sisi penerima, *TCP* memastikan bahwa suatu paket sudah sampai, menyusunnya sesuai nomor urut, dan menentukan apakah paket tiba tanpa mengalami perubahan.
- Jika paket mengalami perubahan atau ada data yang hilang, *TCP* meminta pengiriman ulang.

- *SSL* membangun hubungan (*connection*) yang aman antara pengirim dan penerima, sehingga pengiriman pesan antara dua entitas dapat dijamin keamanannya.

Application (*HTTP, FTP, Telnet*)
Security (*SSL*)
Transport (*TCP*)
Network (*IP*)
Network Access (PPP, modem, ADSL)

- Perlu dicatat bahwa *SSL* adalah protokol *client-server*, yang dalam hal ini *web browser* adalah *client* dan *website* adalah *server*.

- *Client* yang memulai komunikasi, sedangkan *server* memberi respon terhadap permintaan *client*.

- Protokol *SSL* tidak bekerja kalau tidak diaktifkan terlebih dahulu (biasanya dengan meng-klik tombol yang disediakan di dalam *web server*)

Komponen SSL

SSL disusun oleh dua sub-protocol (*layer*):

1. *SSL handshaking*, yaitu sub-protokol untuk membangun koneksi (kanal) yang aman untuk berkomunikasi,

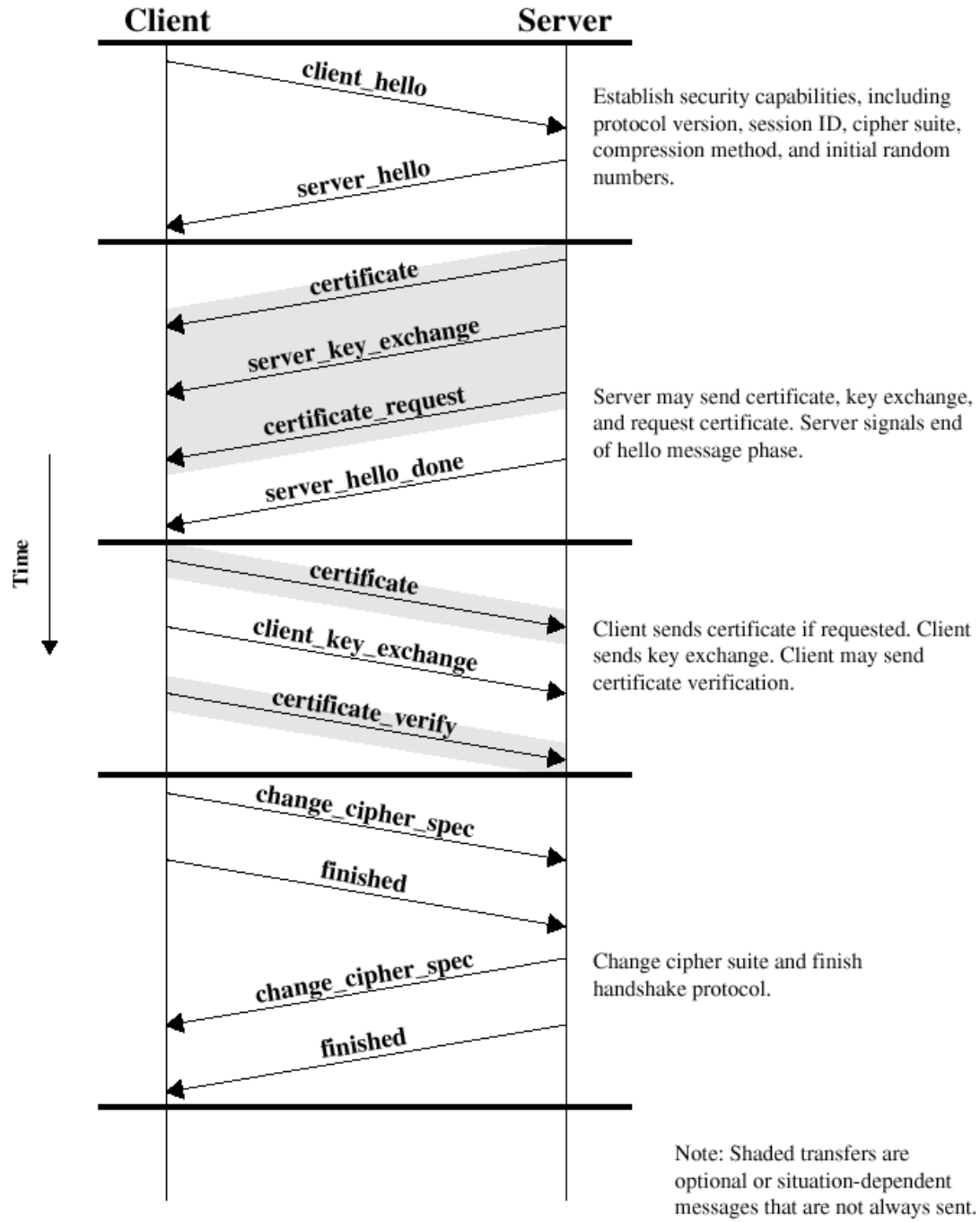
2. *SSL record*, yaitu sub-protokol yang menggunakan kanal yang sudah aman. *SSL Record* membungkus seluruh data yang dikirim selama koneksi.

Sub-protokol *handshaking*

- Merupakan bagian yang paling kompleks di dalam SSL.
- Proses yang dilakukan di dalam sub-protokol handshaking:
  - *Say* ’helllo’
  - *Client* dan *server* melakukan otentikasi satu sama lain (opsional)
  - Menggunakan kunci publik untuk membentuk kunci rahasia (untuk enkripsi dan dekripsi pesan dengan algoritma simetri)
  - Menegosiasikan algoritma enkripsi, fungsi hash, MAC, kompresi

- Subprotokol *handshaking* dilakukan sebelum data ditransmisikan antara client dan server

Sub-protokol *handshaking*



- **ClientHello**

Klien mengirim *ClientHello* yang berisi: versi maksimal yang didukung, daftar *cipher suites* yang didukung, daftar ekstensi (SNI, ALPN, supported groups/curves), dan *ClientRandom* (32 bytes random).

- **ServerHello**

Server merespons *ServerHello* memilih versi, *cipher suite*, dan mengirim *ServerRando*m.

- **ServerCertificate**

Server kirim sertifikat X.509 untuk membuktikan identitasnya. Klien akan memverifikasi sertifikat terhadap CA yang dipercaya.

- **ServerKeyExchange** (jika diperlukan)

Jika menggunakan Diffie-Hellman (ECDH/DHE), server mengirim parameter publik yang diperlukan untuk menghasilkan *shared secret*.

- **(Opsional) CertificateRequest**

Jika server meminta autentikasi klien, ia mengirim permintaan sertifikat.

- **ServerHelloDone**

Menandakan server selesai tahap hello/sertifikat.

- **ClientCertificate** (opsional)

Jika diminta, klien kirim sertifikatnya.

- **ClientKeyExchange**

Klien mengirim public key untuk *key-exchange* (contoh: client ECDH public key) atau, pada metode RSA (yang di TLS1.2 juga dipakai), klien mengenkripsi *premaster secret* dengan public key server.

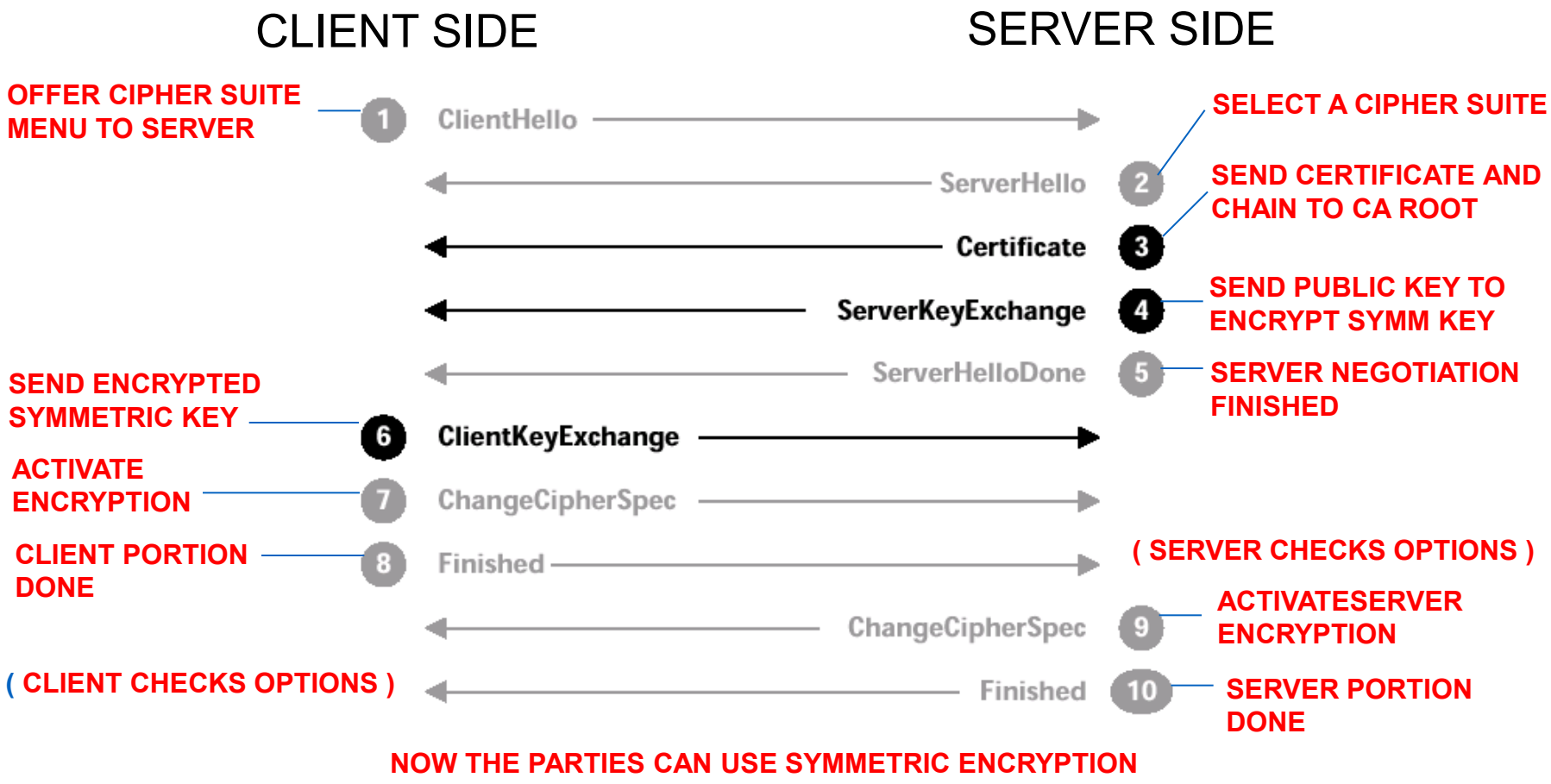
- **CertificateVerify** (jika klien mengautentikasi diri)

Klien menandatangani transkrip *handshake* untuk membuktikan kepemilikan private key.

- **ChangeCipherSpec** + **Finished**

Pesan *ChangeCipherSpec* mengindikasikan bahwa pesan berikutnya akan terenkripsi dengan kunci baru. *Finished* adalah pesan pertama yang terenkripsi dan berisi verifikasi integrity dari seluruh handshake. Jika verifikasi Finished berhasil di kedua pihak, handshake berhasil dan aplikasi boleh kirim data melalui **Record Protocol** terenkripsi.

SSL Messages



SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

Client Hello

- Protocol version
  - SSLv3(major=3, minor=0)
  - TLS (major=3, minor=1)
- Random Number
  - 32 bytes
  - First 4 bytes, time of the day in seconds, other 28 bytes random
  - Prevents replay attack
- Session ID
  - 32 bytes – indicates the use of previous cryptographic material
- CipherSuite cipher_suites
  - Cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)
- Compression algorithm

ClientHello (RFC)

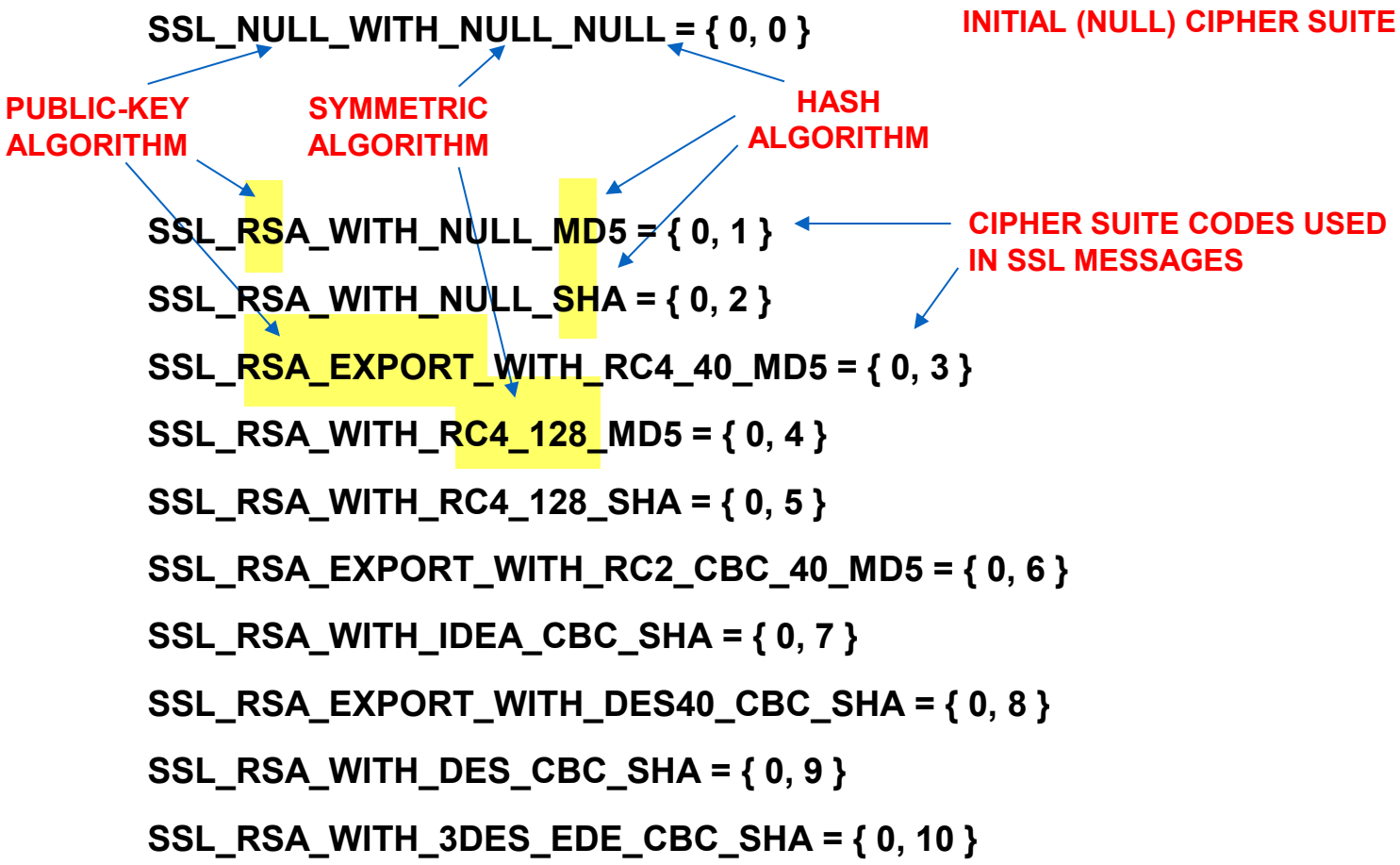
```
struct {
   ProtocolVersion client_version;
   Random random;
   SessionID session_id;
   CipherSuite cipher_suites;
   CompressionMethod compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

Session id (if the client wants to resume an old session)

Cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)

Client Hello - Cipher Suites



Server Hello

- Version
- Random Number
  - Protects against handshake replay
- Session ID
  - Provided to the client for later resumption of the session
- Cipher suite
  - Usually picks client’s best preference – No obligation
- Compression method

Certificates

- Sequence of X.509 certificates
  - Server’s, CA’s, …
- X.509 Certificate associates public key with identity
- Certification Authority (CA) creates certificate
  - Adheres to policies and verifies identity
  - Signs certificate
- User of Certificate must ensure it is valid

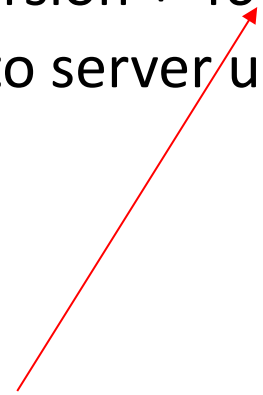
Validating a Certificate

- Must recognize accepted CA in certificate chain
  - One CA may issue certificate for another CA

- Must verify that certificate has not been revoked
  - CA publishes Certificate Revocation List (CRL)

Client Key Exchange

- Premaster secret
  - Created by client; used to “seed” calculation of encryption parameters
  - 2 bytes of SSL version + 46 random bytes
  - Sent encrypted to server using server’s public key



This is where the attack happened in SSLv2

Change Cipher Spec & Finished Messages

- Change Cipher Spec
  - Switch to newly negotiated algorithms and key material

- Finished
  - First message encrypted with new crypto parameters
  - Digest of negotiated master secret, the ensemble of handshake messages, sender constant
  - HMAC approach of nested hashing

SSL Encryption

- Master secret
  - Generated by both parties from premaster secret and random values generated by both client and server
- Key material
  - Generated from the master secret and shared random values
- Encryption keys
  - Extracted from the key material

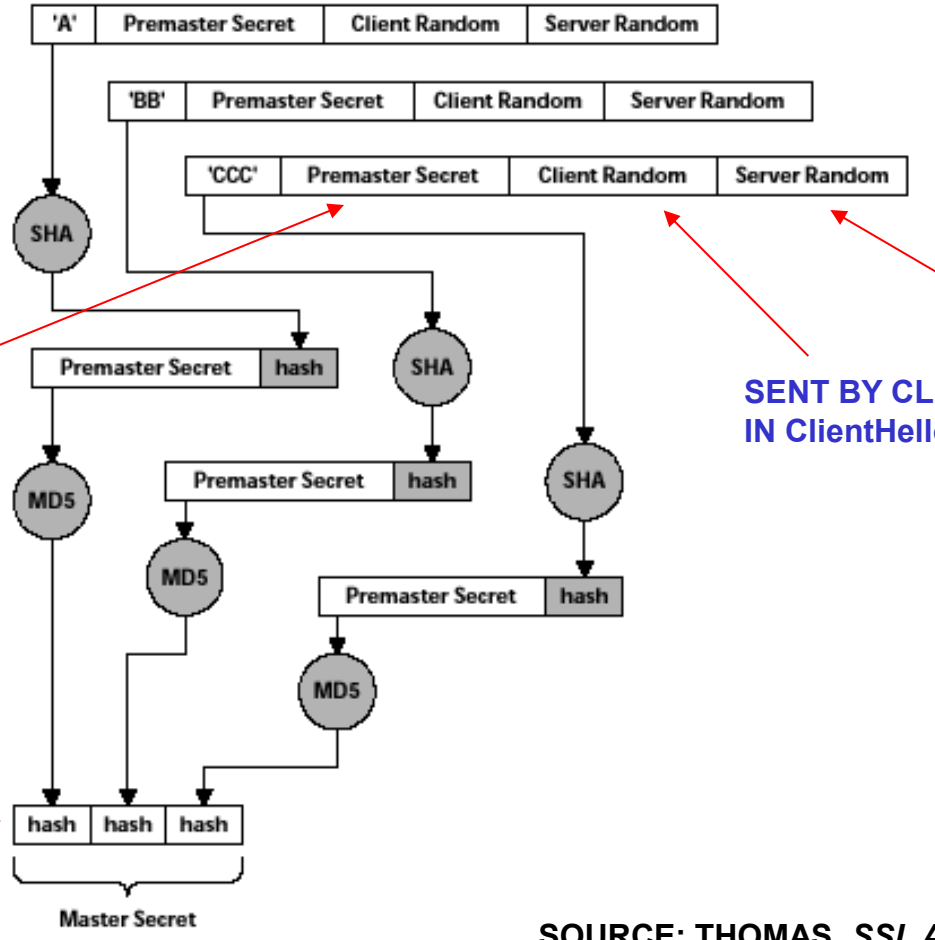
Generating the Master Secret

SERVER'S PUBLIC KEY IS SENT BY SERVER IN ServerKeyExchange

CLIENT GENERATES THE PREMASTER SECRET

ENCRYPTS WITH PUBLIC KEY OF SERVER

CLIENT SENDS PREMASTER SECRET IN ClientKeyExchange



SENT BY SERVER IN ServerHello

SENT BY CLIENT IN ClientHello

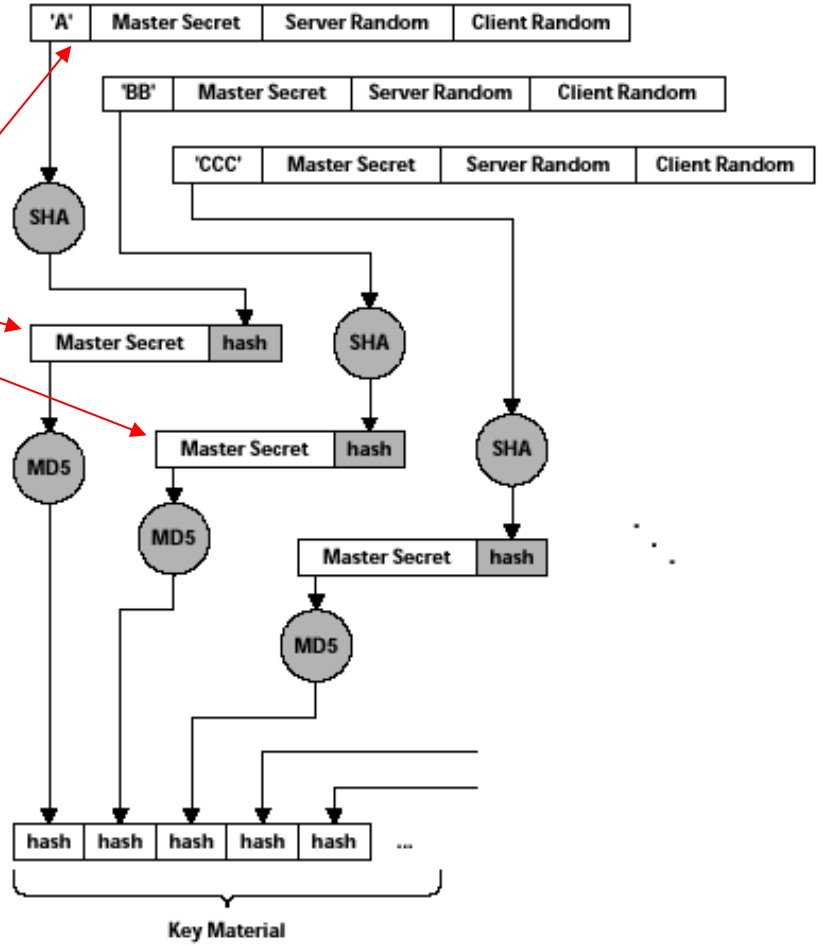
MASTER SECRET IS 3 MD5 HASHES CONCATENATED TOGETHER = 384 BITS

SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

Generation of Key Material

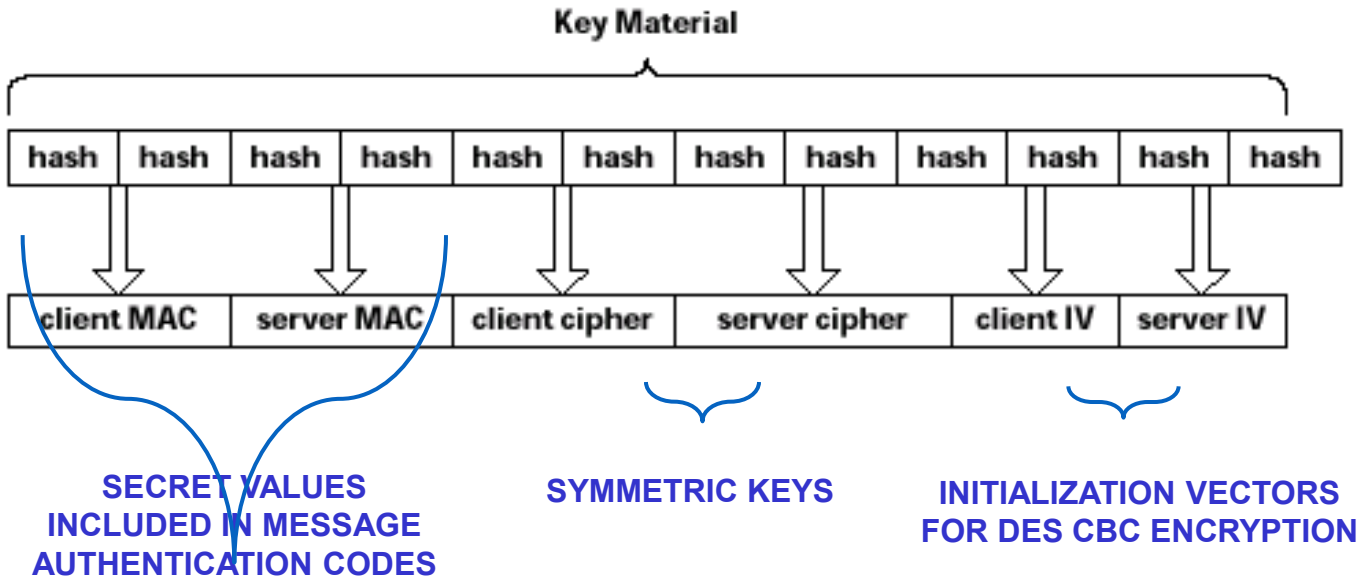
JUST LIKE FORMING THE MASTER SECRET

EXCEPT THE MASTER SECRET IS USED HERE INSTEAD OF THE PREMASTER SECRET



SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

Obtaining Keys from the Key Material



SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

- Sampai di sini, proses pembentukan kanal yang aman sudah selesai.

- Bila sub-protokol ini sudah terbentuk, maka *http://* pada *URL* berubah menjadi *https://* (*http secure*)

https://bankmandiri.co.id

ID | EN

Ketik untuk mencari

Log In

mandiri

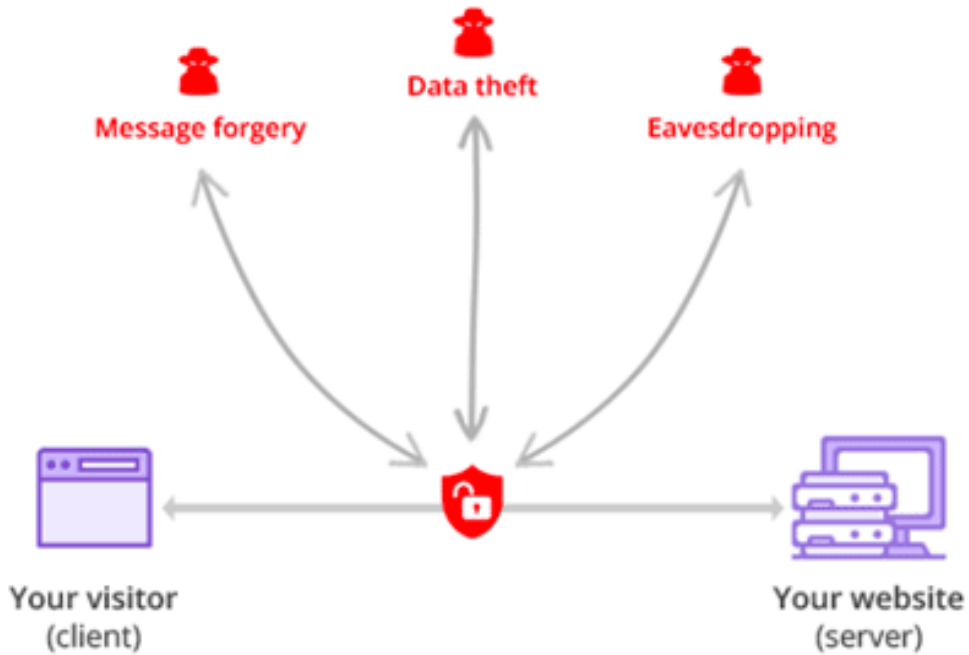
Perseorangan Bisnis Prioritas Tentang Mandiri Bantuan

Full Digital Banking World

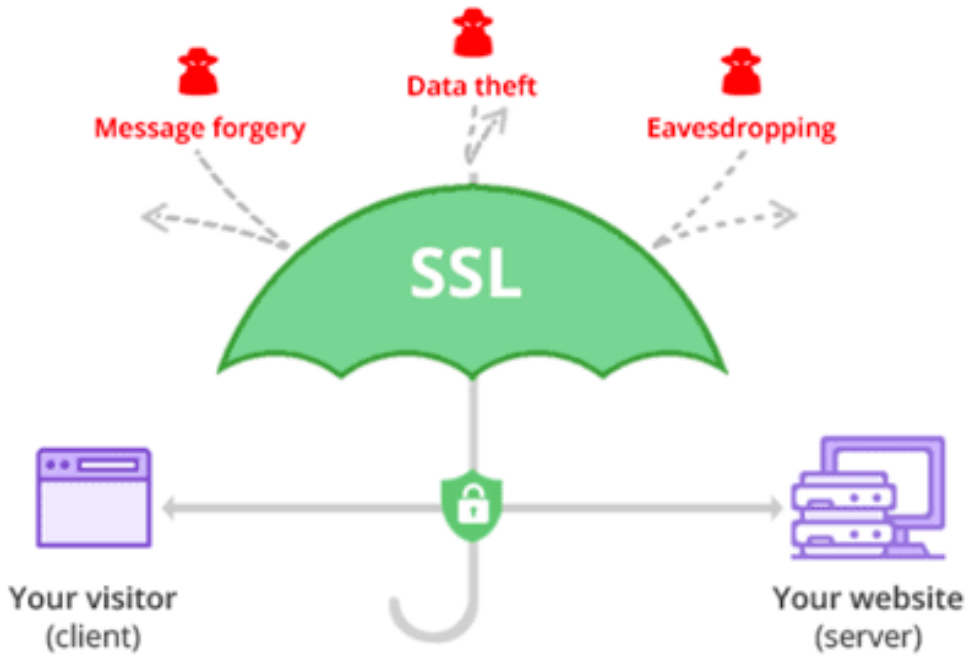
Financial Super App Livin' by Mandiri dan Wholesale Digital Super Platform Kopra by Mandiri

SELENGKAPNYA >

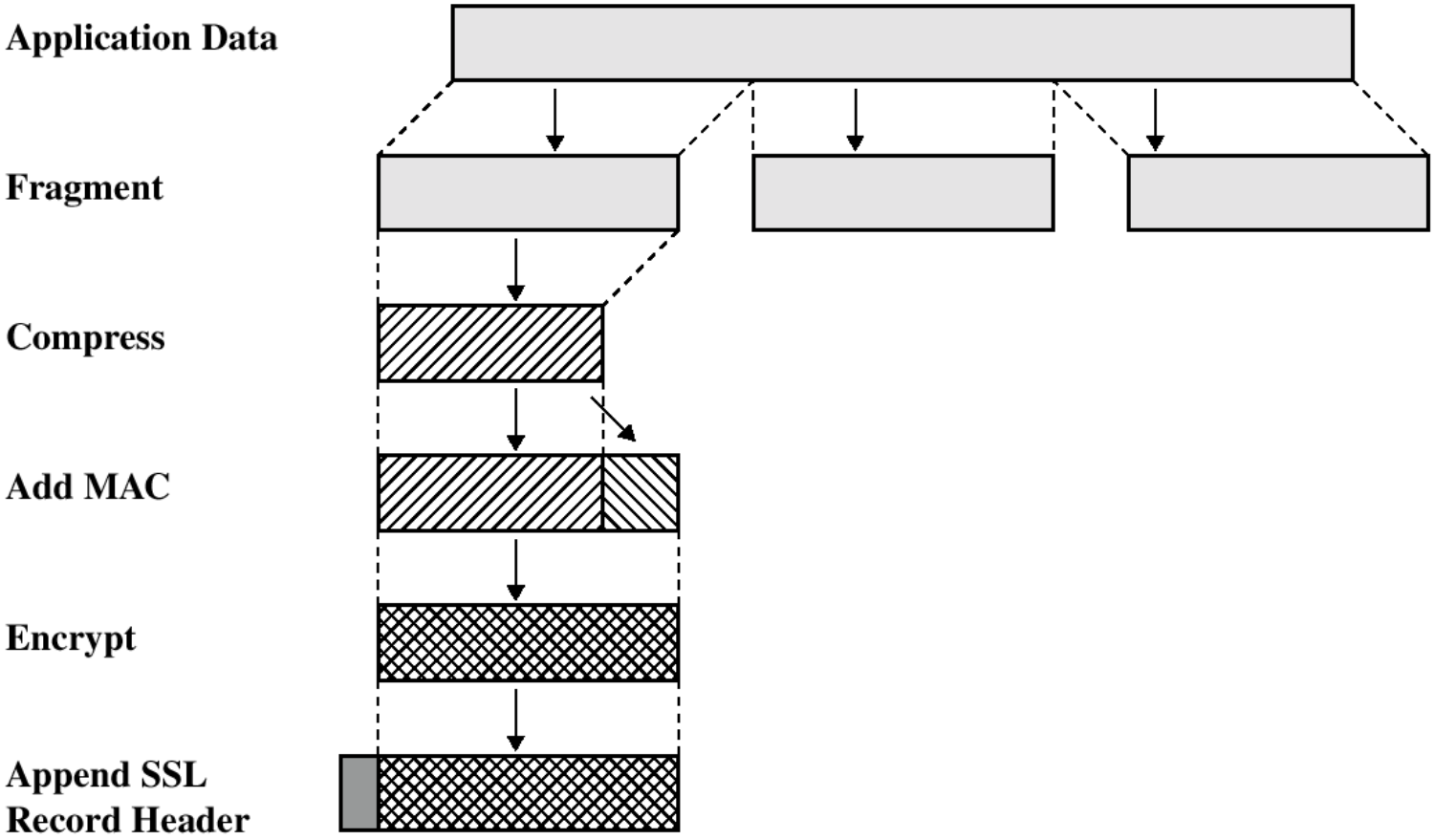
HTTP: No Encryption (no SSL)



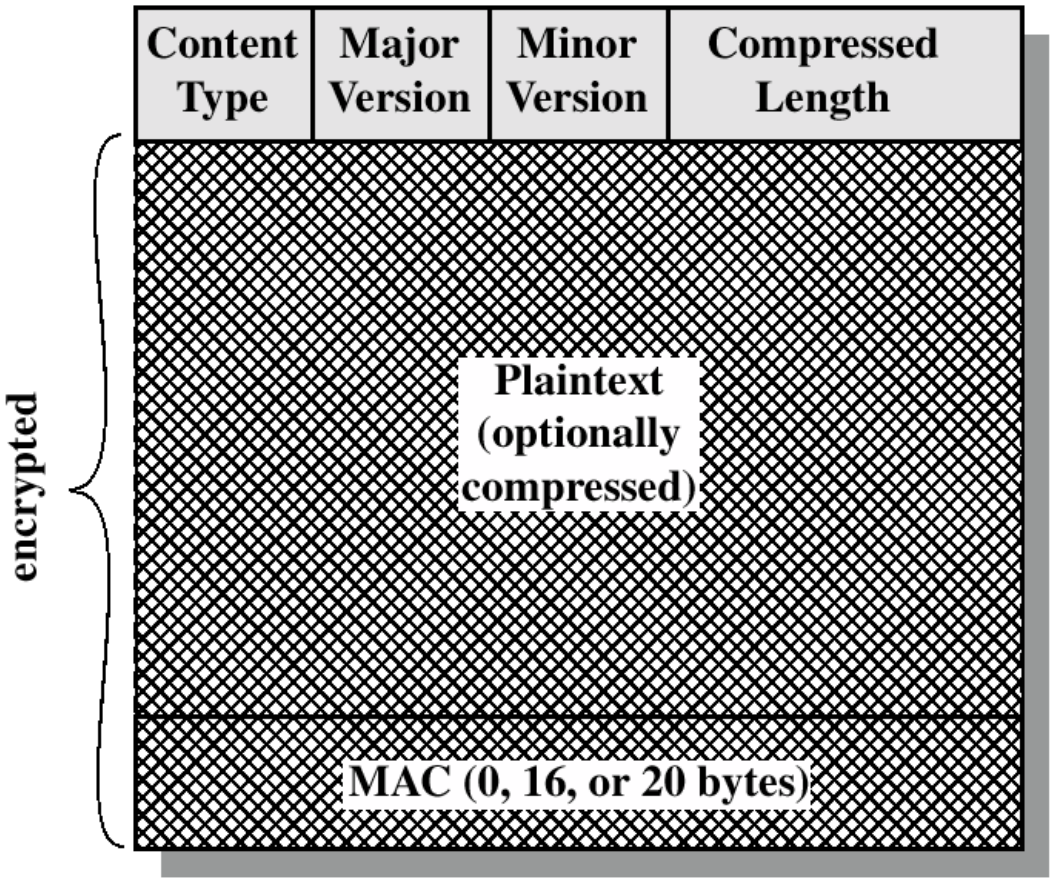
HTTPS: Secure Cheap SSL Connection



Sub-protokol *SSL record*



SSL Record Format



- Di tempat penerima, sub-protokol *SSL* Record melakukan proses berkebalikan: mendekripsi data yang diterima, mengotentikasinya (dengan *MAC*), men-dekompresinya, lalu merakitnya.

- Protokol SSL membuat komunikasi menjadi lebih lambat.

- Piranti keras, seperti kartu *peripheral component interconnect* (*PCI*) dapat dipasang ke dalam *web server* untuk memproses transaksi *SSL* lebih cepat sehingga mengurangi waktu pemrosesan

SSL Overhead

- 2-10 times slower than a TCP session

- Where do we lose time
  - Handshake phase
    - Client does public-key encryption
    - Server does private-key encryption (still public-key cryptography)
    - Usually clients have to wait on servers to finish
  - Data Transfer phase
    - Symmetric key encryption

TLS (*Transport Layer Security*)

- Pada Tahun 1996, *Netscape Communications Corp*. mengajukan *SSL* ke *IETF* (*Internet Engineering Task Force*) untuk standardisasi.
- Hasilnya adalah *TLS* (*Transport Layer Security*). *TLS* dijelaskan di dalam *RFC* 2246
- Untuk informasi lebih lanjut perihal *TLS*, kunjungi situs *IETF* di *www.ietf.org/rfc/rfc22*.
- *TLS* dapat dianggap sebagai *SSL* versi 3.1, dan implementasi pertamanya adalah pada Tahun 1999

Transport Layer Security (TLS)

- The same record format as the SSL record format.
- Defined in RFC 2246.
- Similar to SSL v3.
- Differences in the:
  - version number
  - message authentication code
  - pseudorandom function
  - alert codes
  - cipher suites
  - client certificate types
  - certificate_verify and finished message
  - cryptographic computations
  - padding

OpenSSL

- OpenSSL adalah *toolkit* kriptografi *open-source* yang mengimplementasikan protokol SSL/TLS

- OpenSSL menyediakan fungsi kriptografi dasar (enkripsi, hashing, tanda tangan, manajemen kunci, X.509 certificates, dll.).

- Paket ini terdiri dari pustaka (libcrypto, libssl) dan utilitas baris-perintah openssl untuk tugas kriptografi.

- Berasal dari fork SSLeay (akhir 1990-an). Sejak itu menjadi salah satu implementasi TLS/SSL paling banyak dipakai (web server, mail server, perangkat IoT, library lain).

- Komponen utama OpenSSL
  1. libcrypto — implementasi algoritma kriptografi (AES, RSA, SHA, HMAC, dsb.)
  2. libssl — lapisan protokol SSL/TLS yang menggunakan libcrypto
  3. openssl (CLI) — program command-line untuk membuat kunci, CSR, sertifikat, mengonversi format, debugging TLS, dll.
- OpenSSL memakai lisensi Apache License 2.0 — ini membuat kompatibilitas lisensi dan penggunaan komersial menjadi lebih mudah dibanding model lisensi lama

Cara umum menggunakan OpenSSL

- Dari CLI: membuat private key, certificate signing request (CSR), self-signed cert, konversi format (PEM/DER/PFX), memeriksa sertifikat, tes koneksi TLS, dsb.

- Contoh singkat:

```
# buat private key
openssl genpkey -algorithm RSA -out key.pem -pkeyopt rsa_keygen_bits:2048

# buat CSR
openssl req -new -key key.pem -out req.csr

# lihat sertifikat
openssl x509 -in cert.pem -text -noout
```

Dokumentasi CLI lengkap tersedia di manual OpenSSL:
https://docs.openssl.org/3.5/man1/openssl/?utm_source=chatgpt.com