Simulasi TLS *Handshake* dengan Menggunakan Algoritma RSA

Reihan Andhika Putra - 13519043
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: andhikareihan349@gmail.com

Abstrak—Perkembangan teknologi yang sangat pesat membuat kejahatan di dunia maya semakin sering terjadi. Contoh kejahatan yang marak belakangan ini adalah pencurian data pada website pribadi ataupun instansi. Salah satu cara mencegah kejahatan ini adalah dengan menggunakan SSL. SSL adalah mekanisme untuk melakukan enkripsi pada komunikasi antara *client* dan *server* sehingga keamanan data dapat lebih terjamin dan data susah untuk dicuri. SSL terletak di *transport layer* dan memiliki peran untuk mengirim data dan informasi yang sudah terenkripsi. SSL sendiri terdiri dari dua komponen utama yaitu SSL Handshaking (biasa disebut TLS Handshake) dan SSL Recording. SSL Handshaking adalah sub-protokol untuk membangun koneksi (kanal) yang aman untuk berkomunikasi, TLS Handshake dilakukan setelah TCP Handshake berhasil dilakukan antara pihak *client* dan *server*. Implementasi TLS Handshake bisa berbeda-beda tergantung algoritma kriptografi yang digunakan oleh *client* dan *server*. Pada makalah ini akan dilakukan simulasi TLS Handshake secara umum antara *client* dan *server* secara umum dengan menggunakan algoritma RSA. Algoritma RSA digunakan karena RSA menyediakan metode untuk menjamin kerahasiaan, integritas, keaslian, dan *non-repudiation* dari komunikasi elektronik. Supaya simulasi terlihat lebih nyata, maka simulasi akan dilakukan dengan *socket programming* dan transfer data akan dilakukan dalam bentuk *segment* sebagaimana transfer data terjadi pada protocol TCP.

Keywords— SSL, TLS Handshake, RSA, Server, Client

I. PENDAHULUAN

Perkembangan teknologi yang pesat membawa banyak perubahan pada pola hidup manusia. Pada awalnya manusia hanya dapat berkomunikasi satu sama lain dengan cara berbincang dan bertemu secara langsung. Sekarang manusia dapat berkomunikasi hanya dengan menggunakan aplikasi pada perangkat *smartphone* atau mengakses *web* dengan menggunakan laptop. Pada awalnya manusia hanya dapat menyimpan data/ingatan pada otak masing-masing atau menuliskannya di media kertas seperti daun, batu, kertas, dan media tradisional lainnya. Sekarang data tersebut bisa disimpan di penyimpanan eksternal seperti *hardisk*, *flashdisk*, atau penyimpanan online seperti *cloud*. Hal ini tentu mempermudah manusia dalam mengelola data yang dimilikinya sekaligus membagikan data tersebut kepada orang lain jika diperlukan. Masih banyak hal positif dari perkembangan teknologi yang membantu proses komunikasi dan penyimpanan data manusia.

Selain memberikan dampak positif, perkembangan teknologi yang pesat juga membawa dampak negatif bagi manusia karena perkembangan teknologi memunculkan banyak peluang baru untuk melakukan kejahatan terutama kejahatan dalam dunia maya. Baru-baru ini Indonesia seringkali mengalami kebobolan data oleh *hacker* terutama data kesehatan. Akhir Agustus lalu, sekitar 1,3 juta data pengguna aplikasi Health Alert Card (eHAC) buatan Kementerian Kesehatan Indonesia yang memuat data COVID-19 dibobol. Tiga bulan sebelumnya, data milik 279 juta warga Indonesia yang dikumpulkan bertahun-tahun oleh Badan Pengelola Jaminan Sosial Kesehatan juga bocor. Hal ini tentu merugikan banyak orang karena banyak data pribadi mereka yang bocor dan data tersebut bisa saja digunakan untuk kejahatan lain. Dari beberapa kasus diatas ternyata terdapat satu kesamaan yaitu data yang disimpan dan komunikasi di aplikasi tersebut dibiarkan telanjang (tanpa enkripsi). Hal ini seharusnya bisa diatasi dengan menerapkan SSL pada *service* yang disediakan di aplikasi tersebut. **[1]**

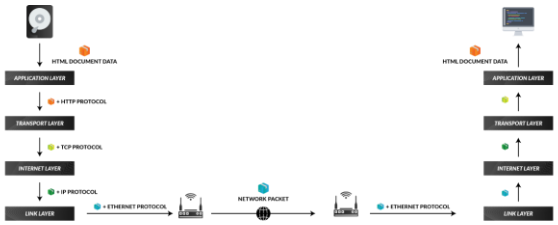
SSL adalah singkatan dari *Secure Socket Layer*. SSL menyediakan cara untuk mengotentikasi dan mengenkripsi komunikasi data melalui jaringan. SSL sangat dibutuhkan oleh *website* pada masa ini karena dengan SSL, transfer dan penyimpanan data yang terjadi pada *website* akan terenkripsi sehingga menyulitkan *hacker* dalam mencuri data. Bahkan Google Chrome pun melarang anda untuk mengakses website tanpa SSL dengan memberikan label kepada *website* tersebut sebagai *not secure*. Terdapat istilah lain yang sering didengar bersamaan dengan SSL yaitu TLS. TLS adalah singkatan dari Transport Layer Security. TLS dan SSL sama-sama mengamankan transfer data di dalam website. Namun, TLS menawarkan teknologi yang lebih baik dan lebih baru karena merupakan upgrade dari SSL. **[2]**

SSL disusun oleh dua sub-protokol yaitu SSL *Handshaking* dan SSL *Record*. SSL *Handshaking* adalah sub-protokol untuk membangun koneksi (kanal) yang aman untuk berkomunikasi, SSL *Handshaking* adalah proses yang paling kompleks yang ada di SSL. SSL *Handshaking* Bagian paling kompleks dari SSL. SSL *Handshaking* memungkinkan server dan klien untuk mengotentikasi satu sama lain. SSL *Handshaking* Digunakan sebelum data aplikasi apa pun dikirimkan. SSL *Record* adalah sub-protokol yang menggunakan kanal yang sudah aman. *SSL Record* membungkus seluruh data yang dikirim selama koneksi. Pada makalah ini akan dilakukan simulasi *SSL* Handshaking dengan menggunakan skema algoritma RSA.

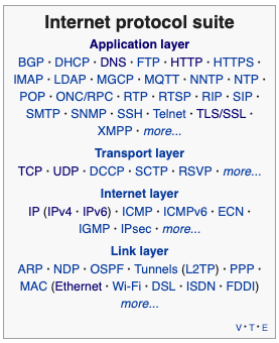
II. DASAR TEORI

A. *Model Komunikasi TCP/IP* [3]

TCP/IP adalah *standard* protokol yang digunakan untuk menghubungkan komputer dan jaringan dengan jaringan yang lebih besar, yaitu Internet. Berikut ini adalah *layer* pada TCP/IP beserta isi dari tiap *layer*-nya:



Gambar 1. Skema Komunikasi TCP/IP



Gambar 2. Protokol yang Bisa Diterapkan di Tiap *Layer*

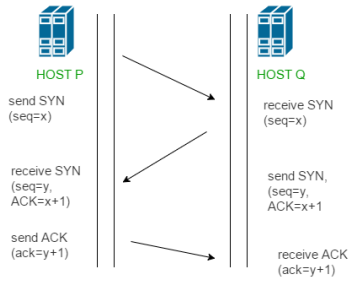
Pada gambar diatas, di bagian kiri, *application layer* akan mendapatkan data dari *internal server* kemudian membungkus data tersebut dengan *header* misalnya *header* protocol HTTP agar data bisa dikenal oleh aplikasi yang dapat memahami data tersebut misalnya *web browser*. Paket HTTP ini akan dikirim ke *transport layer*. *Transport layer* akan membungkus paket dengan *header* protokol TCP bersama dengan port sumber dan tujuan. Port sumber adalah port aplikasi yang mengirimkan data (*server*) dan port tujuan adalah port aplikasi penerima data (*client*).

Paket TCP ini diterima oleh *internet layer* yang akan membungkus paket dengan *header* yang berisi alamat IP sumber dan tujuan. Setelah mendapatkan *header* tersebut maka paket siap ditransmisikan melalui jaringan internet. *Layer* terakhir adalah *physical layer*. Pada bagian ini, *Network Interface Card* (NIC) akan mengambil paket dan menambahkan alamat MAC tujuan ke dalamnya. Paket kemudian akan ditransmisikan ke tujuan.

Setelah paket diterima oleh penerima, header dari paket akan dilepas dan paket yang semula kecil akan digabungkan menjadi objek yang lebih besar hingga data asli berhasil dipulihkan. Hal ini bisa terjadi karena pada setiap *layer* terdapat *header* yang memberikan informasi bagaimana paket dapat digabungkan dengan paket lain hingga didapatkan data yang sempurna. Setelah itu data bisa digunakan oleh aplikasi yang bersangkutan.

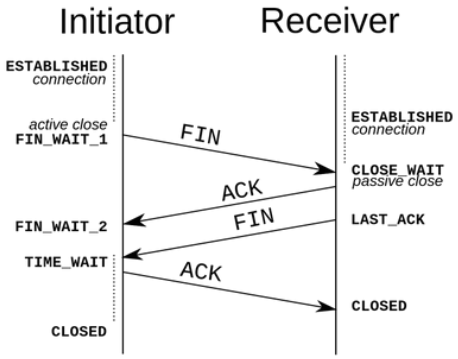
B. *Protokol HTTP* [3]

Protokol HTTP adalah protocol yang berjalan diatas TCP/IP atau UDP/IP. Sebelum data dikirim ke penerima, saluran komunikasi harus dibuka antara pengirim dan penerima. Hal ini dapat dilakukan dengan menggunakan komunikasi TCP/IP antara pengirim dan pengguna. Berikut adalah ilustrasi pembukaan saluran koneksi antara pengirim dan penerima dengan protocol TCP/IP.



Gambar 3. TCP/IP 3-*Way Handshake*

Proses diatas disebut dengan 3-*Way Handshake*. Setelah *handshake* selesai dan saluran komunikasi TCP dibuka, *client* atau *server* dapat mengirim dan menerima data melalui koneksi yang sama kecuali *client* atau *server* menutup koneksi. Setelah *server* tidak memiliki data lagi untuk dikirim, *server* akan mengirim paket kosong dengan *flag* FIN untuk menunjukkan bahwa pengiriman telah selesai dan *server* ingin menutup koneksi. *Client* dapat meng-*acknowledge* paket ini dan mulai untuk menutup koneksinya sendiri. Berikut adalah ilustrasi penutupan koneksi.



Gambar 4. TCP *Closing Connection*

HTTP adalah protokol yang tidak aman karena data yang digunakan untuk berkomunikasi dibiarkan telanjang tanap enkripsi. *Man-in-the-Middle Attack* dapat mendengarkan komunikasi TCP dan membaca data pribadi yang dikirimkan melalui *web*. Untuk itu, kita perlu menerapkan SSL pada *web* supaya *attacker* kesulitan dalam mencuri informasi yang ada di *web* kita.

C. *SSL/TLS* [3]

HTTPS adalah singkatan dari *HyperText Transfer Protocol Secure*. HTTPS sering disalah artikan sebagai protocol HTTP yang lebih aman dan berdiri sendiri, padahal protokol HTTPS

tidak bisa sendirian melakukan enkripsi data. HTTPS bergantung pada lapisan protokol SSL atau TLS.

Baik lapisan protokol HTTP dan lapisan protokol TLS adalah bagian dari *application layer*. Peran lapisan TLS adalah untuk membuat koneksi yang aman antara *client* dan *server* dengan menggunakan TLS *Handshake* (dilakukan setelah TCP *3-Way Handshake)* dan mengenkripsi data HTTP menggunakan algoritma kriptografi kunci simetrik yang disepakati antara *client* dan *server*.

Karena data aplikasi dienkripsi, *Man-in-the-Middle Attack* mungkin memperoleh data tersebut tetapi tidak akan dapat memahaminya. Ketika protokol HTTP digunakan bersamaan dengan protokol TLS, maka akan disebut sebagai protokol HTTPS. SSL disusun oleh dua sub-protokol yaitu:

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.

D. *TLS/SSL Handshake* [5]

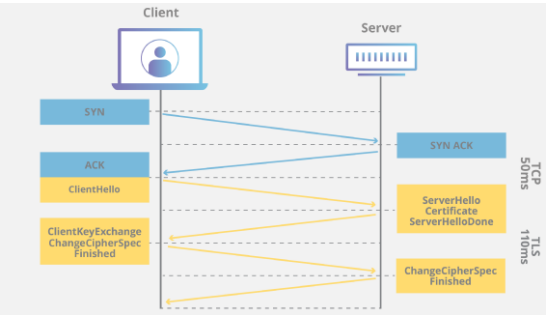
TLS *Handshake* adalah proses untuk memulai sesi komunikasi aman yang menggunakan enkripsi dengan algoritma kunci simetrik. Selama TLS *Handshake*, kedua pihak yang berkomunikasi akan bertukar pesan untuk saling mengakui, memverifikasi satu sama lain, menetapkan algoritma enkripsi yang akan mereka gunakan, dan memvalidasi *Session Key* yang dihasilkan oleh kedua pihak. TLS *Handshake* terjadi setiap kali pengguna masuk ke situs web menggunakan protocol HTTPS. TLS *Handshake* juga terjadi setiap kali terjadi panggilan API dan DNS melalui *query* HTTPS.

Saat melakukan TLS *Handshake*, *client* dan *server* bersama-sama akan melakukan hal berikut:

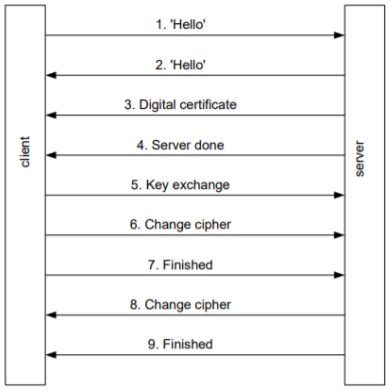
1. Menentukan versi TLS (TLS 1.0, 1.2, 1.3, dan lain-lain.) yang akan mereka gunakan.
2. Menentukan satu set algoritma kriptografi kunci simetrik yang akan digunakan untuk mengenkripsi dan dekripsi data selama komunikasi berlangsung.
3. Otentikasi identitas *server* melalui kunci publik server dan sertifikat digital yang dimiliki oleh server.
4. Membuat *shared Session Key* yang digunakan untuk enkripsi simetrik selama proses komunikasi berlangsung.

TLS *Handshake* adalah pertukaran serangkaian *datagram*, atau pesan, yang dilakukan antara *client* dan *server*. TLS *Handshake* melibatkan beberapa langkah.. Langkah-langkah yang tepat dalam proses TLS *Handshake* akan bervariasi dan bergantung pada jenis algoritma pertukaran kunci yang digunakan oleh *client* dan *server*. Algoritma pertukaran kunci yang paling sering digunakan untuk TLS *Handshake* adalah RSA. Skema TLS *Handshake* dengan menggunakan algoritma RSA adalah sebagai berikut:

1. *Client Hello*: *client* memulai handshake dengan mengirimkan pesan "*hello*" ke *server*. Pesan ini akan berisikan versi TLS, algoritma kriptografi kunci simetrik, dan metode kompresi data yang didukung *client*. Di dalam pesan ini juga terdapat serangkaian *byte* acak yang dikenal sebagai "*client random*".
2. *Server Hello*: sebagai balasan atas pesan *Client Hello*, *server* mengirimkan pesan yang berisi sertifikat SSL *server*, perjanjian versi TLS, algoritma kriptografi simetrik, dan metode kompresi yang dipilih *server* berdasarkan ketersediaan *client.* Di dalam pesan ini juga terdapat serangkaian *byte* acak yang dikenal sebagai "*server random*".
3. *Authentication*: *client* memverifikasi sertifikat SSL *server* dengan organisasi otoritas sertifikat yang mengeluarkan sertifikat tersebut. Apabila sertifikat terverifikasi maka *client* berinteraksi dengan pemilik domain yang sebenarnya.
4. *Premaster Secret*: *client* mengirimkan satu string *byte* acak lagi yang disebut sebagai "*premaster secret*". *Premaster secret* dienkripsi dengan kunci publik dan hanya dapat didekripsi dengan kunci privat oleh server. *Client* mendapatkan kunci publik dari sertifikat SSL *server*.
5. *Decrypt Premaster*: *server* mendekripsi *premaster secret* yang dikirimkan oleh *client* dengan menggunakan kunci privat yang dimilikinya.
6. *Generate Session Key*: baik *client* maupun server menmbuat *session key* dari konkatenasi antara *client random*, *server random*, dan *premaster secret*. *Sessio key* yang dibuat oleh *client* dan *server* haruslah sama.
7. *Client Cipher Spec*: *client* mengirimkan pesan "selesai" yang dienkripsi dengan *session key* menggunakan algoritma kriptografi kunci simetrik yang disepakati.
8. *Server Cipher Spec*: *server* menerima pesan "selesai" yang dikirimkan oleh *client* dan mencoba melakukan dekripsi dengan menggunakan *session key*. Apabila hasilnya benar maka *server akan* mengirimkan pesan "selesai" yang dienkripsi dengan *session key* menggunakan algoritma kriptografi kunci simetrik yang disepakati.
9. *Secure Connection Established*: *client* menerima pesan "selesai" yang dikirimkan oleh *server* dan mencoba melakukan dekripsi dengan menggunakan *session key*. Apabila hasilnya benar maka *TLS Handshake* selesai, dan komunikasi berlanjut menggunakan *session key*. *Client* mengirimkan pesan ACT kepada *server*.



Gambar 5. TCP dan TLS *Handshake*



Gambar 6. TLS *Handshake*

E. *Kriptografi Kunci Simetrik* [3]

Dalam algoritma kunci simetrik, enkripsi dan dekripsi data dilakukan dengan menggunakan kunci yang sama. Kunci ini biasa disebut sebagai *shared key*. Kriptografi kunci simetrik merupakan algoritma yang aman tetapi kita tidak dapat mengirim kunci yang digunakan ke khalayak umum. Kriptografi kunci simetrik umumnya lebih cepat daripada kriptografi kunci public dan dapat digunakan untuk mengenkripsi data dalam jumlah besar. Kriptografi kunci simetrik sering disebut sebagai *block cipher*.

Algoritma kunci simetrik biasa digunakan untuk membuka saluran enkripsi antara dua pihak yang terpercaya. Hanya kedua pihak ini yang akan tahu tentang data yang dibagikan di antara mereka karena tidak ada orang lain di jaringan yang memiliki akses ke *shared key*. Salah satu algoritma kunci simetris yang paling populer di web adalah AES (Advanced Encryption Standard).

F. *Kriptografi Kunci Asimetrik* [3]

Dalam algoritma kunci asimetris, kita memiliki dua kunci, satu untuk melakukan enkripsi dan satu lagi untuk melakukan dekripsi. Kunci publik digunakan untuk mengenkripsi data dan mengirim data tersebut ke penerima. Hanya kunci rahasia yang berkaitan yang dapat mendekripsi data tersebut.

Algoritma kriptografi kunci asimetrik yang paling populer adalah RSA (Rivest–Shamir–Adleman). RSA banyak digunakan di web untuk pertukaran kunci dan validasi tanda tangan digital. Algoritme kriptografi kunci asimetrik umumnya lebih lambat. Semakin besar panjang kunci atau data, semakin lama waktu yang dibutuhkan untuk mengenkripsi atau mendekripsi data.

Kriptografi kunci asimetrik tidak digunakan untuk enkripsi data dalam jumlah besar. Kriptografi kunci simetrik digunakan untuk mengenkripsi atau mendekripsi data dalam jumlah besar karena jauh lebih cepat dan efisien, sedangkan kriptografi kunci asimetrik digunakan untuk mentransfer kunci simetrik bersama.

G. *Algoritma RSA (Rivest-Shamir-Adleman)* [4]

Algoritma Rivest-Shamir-Adleman (RSA) merupakan algoritma yang ditemukan oleh tiga orang peneliti dari MIT, yakni Ronald Rivest, Adi Shamir, dan Len Adleman, pada tahun 1976. Algoritma ini merupakan merupakan algoritma kunci-publik yang paling terkenal dan paling banyak aplikasinya. Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor prima.

Ada beberapa komponen dari algoritma ini, diantaranya:

1. Komponen rahasia

Komponen rahasia adalah komponen yang hanya diketahui oleh individu yang akan menerima pesan. Komponen ini tidak boleh diberitahukan kepada khalayak umum. Komponen ini digunakan untuk melakukan dekripsi pesan yang sudah dienkripsi dengan komponen publik. Beberapa komponen rahasia pada algoritma RSA adalah bilangan prima *p* dan *q*, fungsi *Totient Euler (Φ(n) = (p-1)*(q-1))* untuk menentukan berapa bilangan yang relatif prima terhadap n, kunci dekripsi *d* *(d ≡ e -1 mod Φ(n))*, dan *plaintext* m yang merupakan pesan yang ingin dikirim.

2. Komponen publik

Komponen public merupakan komponen yang boleh diketahui dan boleh dibagikan ke khalayak umum. Komponen ini biasa digunakan untuk melakukan enkripsi dari *plaintext* yang akan dikirimkan ke penerima. Komponen ini dapat berupa deretan angka atau kombinasi angka dan karakter. Beberapa komponen publik diantaranya nilai *n (n = p*q)*, kunci enkripsi *e* (bilangan yang relatif prima terhadap *Φ(n)*), dan *ciphertext c* yang merupakan pesan yang telah terenkripsi.

Ada tiga prosedur yang dilakukan pada algoritma RSA, yakni.

1. Pembangkitan Kunci

Ada beberapa tahapan yang dilakukan untuk pembangkitan kunci RSA, yaitu:

a. Memilih dua buah bilangan prima *p* dan *q*.
b. Menghitung nilai *n = p*q*.
c. Menghitung nilai *Φ(n) = (p-1)*(q-1)*
d. Memilih sebuah bilangan prima e sebagai kunci publik yang nilainya relatif prima terhadap *Φ(n)*.
e. Menghitung nilai kunci privat d dengan persamaan *d ≡ e -1 mod Φ(n)*.

Dari pengerjaan tahapan di atas, hasil yang diperoleh ada dua, yakni kunci publik *(e, n)* dan kunci privat *(d, n)*.

2. Enkripsi

Ada beberapa tahapan dalam melakukan enkripsi, yaitu:

a. Membagi *plaintext* ke dalam beberapa blok m_1, m_2, .., m_n dengan syarat $0 \leq m_i < n - 1$.
b. Menghitung nilai blok ciphertext c_i dengan persamaan $c_i = m_i^e \bmod n$
c. Menggabungkan kembali blok-blok *ciphertext* menjadi *ciphertext* yang utuh.

3. Dekripsi

Ada beberapa tahapan dalam melakukan dekripsi, yaitu:

a. Membagi *ciphertext* ke dalam beberapa blok c_1, c_2, .., c_n.

b. Menghitung kembali blok *plaintext* m_i dengan persamaan $m_i = c_i \, d \bmod n$.

c. Menggabungkan kembali blok-blok *ciphertext* menjadi *plaintext* yang utuh.

d. Jika *plaintext* berupa teks (bukan sederetan angka saja), buat algoritma untuk mengembalikan nilai pada *plaintext* menjadi pesan yang sesungguhnya

III. RANCANGAN SIMULASI

Simulasi yang penulis ajukan adalah simulasi dengan menggunakan program yang terdiri dari *client* dan *server* yang saling berkomunikasi melalui jaringan. Program akan ditulis dengan bahasa pemrograman Python dengan menggunakan kakas *socket* programming yang sudah tersedia. Untuk algoritma pertukaran kunci yaitu RSA akan menggunakan *library* bawaan Python yang sudah tersedia. Untuk algoritma enkripsi data pada saat komunikasi akan menggunakan AES yang juga diambil dari *library* bawaan Python. Penggunaan *library* bawaan bertujuan untuk mengurangi kompleksitas program yang dibuat. Program akan dibuat secara *verbose*, yaitu setiap langkah dalam komunikasi yang dilakukan di-output ke *stdout*, dalam hal ini adalah terminal.

Secara umum, program terdiri dari lima tahapan yaitu:

1. *Server* mendengarkan *request* dari *client* dan menambahkan *client* ke *list client*
2. Inisialisasi koneksi dengan melakukan TCP *3-Way Handshake* dan dilanjut dengan TLS *Handshake*
3. *Server* melakukan pengiriman data ke *client* secara sekuensial hingga semua data berhasil dikirimkan ke semua *client*
4. *Closing connection.*

Pada tahapan pertama, *client* akan melakukan pencarian server dengan mengirim *request* di *broadcast address*. *Server* akan mendengarkan *request* dari client dari *broadcast address*. Apabila server mendapatkan *request*, server akan menyimpan list client. Setiap kali *server* mendapatkan *client*, *server* akan menanyai ke pengguna untuk melanjutkan *listening* atau tidak. Apabila tidak, maka server akan mulai mengirimkan berkas secara sekuensial ke *list client*. *Server* dan *client* diasumsikan berada di satu *broadcast domain* yang sama dan akan berkomunikasi menggunakan *socket* UDP.

Pada tahapan kedua, *server* akan melakukan TCP *3-Way Handshake* dengan setiap *client* yang terkoneksi dengan *server*, lalu *client* akan melakukan TLS *Handshake* dengan server. TCP *3-Way Handshake* dan TLS *Handshake* dilakukan serentak artinya jika terdapat banyak *client* maka pada *client* pertamaa akan dilakukan TCP *3-Way Handshake* langsung diikuti dengan TLS *Handshake*, jika sudah selesai, proses akan diulangi untuk *client* yang lain. Proses dari TCP *3-Way Handshake* adalah sebagai berikut:

1. (SYN): Pada langkah pertama, *server* ingin membuat koneksi dengan *client*, sehingga mengirimkan segmen dengan SYN (*Synchronize Sequence Number*) yang menginformasikan *client* bahwa *server* kemungkinan akan memulai komunikasi.
2. (SYN + ACK): *Client* merespons permintaan *server* dengan bit sinyal SYN-ACK. Acknowledgement (ACK).
3. (ACK): Pada Langkah terakhir, *server* mengakui respon *client* dan koneksi sudah *established.*

Adapun proses dari TLS *Handshake* mengikuti apa yang dijelaskan di bagian dasar teori. Perhatikan lagi bahwa TLS *Handshake* yang diimplementasikan akan menggunakan algoritma *RSA* dan algoritma kriptografi simetrik yang digunakan adalah AES, keduanya menggunakan *library* bawaan Python. Setelah tahapan ini selesai maka koneksi yang *secure* sudah dibangun antara *client* dan *server*. Berikut adalah konstanta yang digunakan pada saat melakukan TLS *Handshake* beserta tabel yang berisikan data apa saja yang dikirim tiap proses di TLS *Handshake*

TABLE I. KONSTANTA HANDSHAKE TLS

```
TLS_VERSION = "1.0"
CRYPTO_ALGORITHM = "rsa"
DATA_COMPRESSION_METHOD = "vca"
SSL_CERTIFICATE = '''
-----BEGIN CERTIFICATE----
MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJ
KoZIhvcNAQEFBQAwVzELMAkGA1UEBhMCQkUxGTAX
BgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNV
BAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNp
Z24gUm9vdCBDQTAeFw05ODA5MDExMjAwMDBaFw0y
ODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkw
FwYDVQQKExBHbG9i
-----END CERTIFICATE-----
'''
```

TABLE II. DATA YANG TERLIBAT DALAM TLS HANDSHAKE

Nama Proses	Alur	Data yang Terlibat
1. *Client Hello*	*client => server*	versi TLS, algoritma kriptografi kunci simetrik, metode kompresi data yang didukung *client*, dan *client random*
2, *Server Hello*	*server => client*	versi TLS, algoritma kriptografi kunci simetrik yang disetujui, metode kompresi data yang didrtujui, *server public key, server random, dan server* SSL *Certificate*
3. *Authentication*	*client*	-
4. *Premaster Secret*	*client => server*	*RSA* (*premaster secret*)
5. *Decrypt Premaster*	*server*	-
6. Generate Session Key	*server + client*	*session key*
7. Client Cipher Spec	*client => server*	AES(*session key*, ”selesai”)

8. Server Cipher Spec	*server* => *client*	AES(*session key*, ”selesai”)
9. Secure Connection Established	*client*	AES(*session key*, ”selesai”)

Pada tahapan ketiga, *server* akan mulai mengirimkan data kepada *client.* Setiap data yang dipertukarkan antara *client* dan *server* akan dienkripsi dengan algoritma kriptografi simetrik yang disetujui sebelumnya yaitu AES. **Namun pada program yang saya implementasikan, enkripsi tersebut tidak dilakukan karena makalah ini lebih berfokus pada *handshake* dan *closing connection.*** Algoritma pengiriman data yang digunakan juga tidak akan dibahas pada makalah ini. *Server* akan mengirimkan data ke *client* secara berurutan setelah bagian-bagiannya dikonversikan sebagai segmen. Setiap segmen memiliki *sequence number* yang menandakan urutan dari setiap segmen. Pengiriman diasumsikan berjalan dengan lancer tanpa adanya *packet loss, duplicate,* maupun *reorder*. Berikut merupakan anatomi dari segmen yang dikirim[6]:

TABLE III. ANATOMI SEGMEN DALAM SIMULASI

Octet Offset	**0**	**1**	**2**	**3**
0	Sequence Number			
4	Acknowledgment Number			
8	Flags	[empty]	Checksum	
12	Data (maksimal 32768 Bytes)			
...				
32777				

Berikut merupakan keterangan dari setiap bagian dari segmen[6]:

1. *Sequence Number* adalah angka urutan dari segmen yang dikirim.
2. *Acknowledgment Number* adalah angka yang dikirimkan beserta segmen ini yang merupakan *Sequence Number* segmen sebelumnya yang diterima (*Previous Sequence Number*) yang menandakan *Sequence Number* tersebut sudah diterima oleh pihak tersebut.
3. *Flags* merupakan penanda apakah segmen ini merupakan dari jenis segmen-segmen pada komunikasi TCP. Apabila bit ke-n adalah 1, maka segmen merupakan salah satu dari jenis berikut:
   a. SYN merupakan *flag* yang menandakan bahwa segmen ini merupakan permulaan dari TCP *3-Way Handshake* yang dilakukan. SYN terletak di *bit* ke-1 dari *byte Flags.*
   b. ACK merupakan *flag* yang menandakan bahwa segmen ini merupakan balasan (*acknowledgement*) dari suatu proses. ACK terletak di *bit* ke-4 dari *byte Flags*.
   c. FIN merupakan *flag* yang menandakan bahwa segmen ini merupakan permulaan dari proses *tearing down connection*. FIN terletak di *bit* ke-0 dari *byte Flags*.
   d. Apabila segmen hanya membawa data, semua *bit* di *byte Flags* adalah 0.
4. *Checksum* merupakan bagian data yang menandakan *signature* dari segmen ini. Detailnya tidak terlalu dibahas di makalah ini.
5. *Data* untuk setiap segmen merupakan bagian dari berkas yang akan dikirim.

Pada tahapan keempat, akan dilakukan *closing connection* antara *server* dengan semua *client* yang terhubung. Protokol *closing connection* akan mengikuti *protocol* yang dilakukan oleh TCP. Skema *closing connection* adalah sebagai berikut:

1. *Server* mengirimkan segmen FIN yang menandakan *server* ingin melakukan *tearing down connection* dengan *client. Server* kemudian menunggu ACK dari *client.*
2. *Client* mengirimkan ACK ke *server.*
3. *Server* menerima ACK dari *client* dan menunggu *client* mengirimkan segmen FIN.
4. *Client* mengirim FIN ke *server.*
5. *Server* menerima FIN dari *client* kemudian mengirimkan ACK ke *server*. Saat *client* menerima ACK ini maka *client* akan menutup koneksinya. *Server* akan menutup koneksinya juga setelah dilakukan prosedur diatas untuk semua *client.*

IV. HASIL PROGRAM DAN SIMULASI

Program yang sudah dibuat dapat diakses di https://github.com/AndhikaRei/TLS-Handshake. Agar proses yang terjadi dimulai dari *handshake* hingga *closing connection* dapat terlihat dengan jelas maka program implementasi dibuat dengan *verbose*. Program *client* dan *server* dapat dijalankan melalui *command line* yang bisa dibaca di *readme* yang ada di link tertera. Berikut adalah *screenshot* beberapa percobaan pada program

A. Percobaan Pertama

Client random: *isgp46h*
Premaster secret: *kgn0s*
Server random: *5tjc5d*
Session key: *gp46h5tjc5dkgn0s*
Status TLS *Handshake:* sukses
Status Operasi Lainnya*:* sukses
Dokumentasi:

1. *Client*

```
Begin TLS handshake with server
Generating Client Hello message
Client random string isgp46h
[Segment SEQ=1  CTL=DATA] Sent   <- Client hello segment
[Segment SEQ=2  CTL=DATA] Received   <- Server hello segment
Server hello message validated
Server random message is 5tjc5d
Client premaster key is kgn0s
[Segment SEQ=2  CTL=DATA] Sent   <- Client encrypted premaster key segment
Session key with server is gp46h5tjc5dkgn0s
[Segment SEQ=0  CTL=DATA] Sent   <- Client change cipher spec
[Segment SEQ=0  CTL=DATA] Received
[Segment SEQ=3  CTL=DATA] Received   <- Server change cipher spec
Secure Connection Established
[Segment SEQ=3  CTL=ACK] Sent
Secure connection with server established
```

Gambar 7. Dokumentasi *Client* Percobaan-1

2. *Server*

```
Begin TLS Handshake with client-127.0.0.1:1336
[Segment SEQ=1  CTL=DATA] Received   <- Client hello segment
Client-127.0.0.1:1336 hello message validated
Client-127.0.0.1:1336 random string is gp46h
Constructing Server Hello message
[Segment SEQ=2  CTL=DATA] Sent   <- Server hello segment
[Segment SEQ=2  CTL=DATA] Received   <- Client premaster key segment
Client-127.0.0.1:1336 premaster key is kgn0s
Session key with client-127.0.0.1:1336 is gp46h5tjc5dkgn0s
[Segment SEQ=0  CTL=DATA] Received   <- Client change cipher spec segment
[Segment SEQ=3  CTL=DATA] Sent   <- Server change cipher spec segment
[Segment SEQ=1  CTL=DATA] Received
[Segment SEQ=3  CTL=ACK] Received
TLS Handshake with client-127.0.0.1:1336 success
```

Gambar 8. Dokumentasi *Server* Percobaan-1

B. Percobaan Kedua

Client random: *is40t53*
Premaster secret: *vc188*
Server random: *rarprr*
Session key: *40t53rarprrvc188*
Status TLS *Handshake:* sukses
Status Operasi Lainnya*:* sukses
Dokumentasi:

1. *Client*

```
Begin TLS handshake with server
Generating Client Hello message
Client random string is40t53
[Segment SEQ=1  CTL=DATA] Sent   <- Client hello segment
[Segment SEQ=2  CTL=DATA] Received   <- Server hello segment
Server hello message validated
Server random message is rarprr
Client premaster key is vc188
[Segment SEQ=2  CTL=DATA] Sent   <- Client encrypted premaster key segment
Session key with server is 40t53rarprrvc188
[Segment SEQ=0  CTL=DATA] Sent   <- Client change cipher spec
[Segment SEQ=0  CTL=DATA] Received
[Segment SEQ=3  CTL=DATA] Received   <- Server change cipher spec
Secure Connection Established
```

Gambar 9. Dokumentasi *Client* Percobaan-2

2. *Server*

```
Begin TLS Handshake with client-127.0.0.1:1336
[Segment SEQ=1  CTL=DATA] Received   <- Client hello segment
Client-127.0.0.1:1336 hello message validated
Client-127.0.0.1:1336 random string is 40t53
Constructing Server Hello message
[Segment SEQ=2  CTL=DATA] Sent   <- Server hello segment
[Segment SEQ=2  CTL=DATA] Received   <- Client premaster key segment
Client-127.0.0.1:1336 premaster key is vc188
Session key with client-127.0.0.1:1336 is 40t53rarprrvc188
[Segment SEQ=0  CTL=DATA] Received   <- Client change cipher spec segment
[Segment SEQ=3  CTL=DATA] Sent   <- Server change cipher spec segment
[Segment SEQ=1  CTL=DATA] Received
[Segment SEQ=3  CTL=ACK] Received
TLS Handshake with client-127.0.0.1:1336 success
```

Gambar 10. Dokumentasi *Server* Percobaan-2

C. Percobaan Ketiga

Client random: *is1eo7e*
Premaster secret: *gzrb3*
Server random: *2i7a9p*
Session key: *1eo7e2i7a9pgzrb3*
Status TLS *Handshake:* sukses
Status Operasi Lainnya*:* sukses
Dokumentasi:

1. *Client*

```
Begin TLS handshake with server
Generating Client Hello message
Client random string is1eo7e
[Segment SEQ=1  CTL=DATA] Sent   <- Client hello segment
[Segment SEQ=2  CTL=DATA] Received   <- Server hello segment
Server hello message validated
Server random message is 2i7a9p
Client premaster key is gzrb3
[Segment SEQ=2  CTL=DATA] Sent   <- Client encrypted premaster key segment
Session key with server is 1eo7e2i7a9pgzrb3
[Segment SEQ=0  CTL=DATA] Sent   <- Client change cipher spec
[Segment SEQ=0  CTL=DATA] Received
[Segment SEQ=3  CTL=DATA] Received   <- Server change cipher spec
Secure Connection Established
[Segment SEQ=3  CTL=ACK] Sent
Secure connection with server established
```

Gambar 11. Dokumentasi *Client* Percobaan-3

2. *Server*

```
Begin TLS Handshake with client-127.0.0.1:1336
[Segment SEQ=1  CTL=DATA] Received   <- Client hello segment
Client-127.0.0.1:1336 hello message validated
Client-127.0.0.1:1336 random string is 1eo7e
Constructing Server Hello message
[Segment SEQ=2  CTL=DATA] Sent   <- Server hello segment
[Segment SEQ=2  CTL=DATA] Received   <- Client premaster key segment
Client-127.0.0.1:1336 premaster key is gzrb3
Session key with client-127.0.0.1:1336 is 1eo7e2i7a9pgzrb3
[Segment SEQ=0  CTL=DATA] Received   <- Client change cipher spec segment
[Segment SEQ=3  CTL=DATA] Sent   <- Server change cipher spec segment
[Segment SEQ=1  CTL=DATA] Received
[Segment SEQ=3  CTL=ACK] Received
TLS Handshake with client-127.0.0.1:1336 success
```

Gambar 12. Dokumentasi *Server* Percobaan-3

V. KESIMPULAN

Algoritma RSA dapat digunakan untuk melakukan prosedur *sharing session key* dalam TLS *Handshake*. Tahapan yang dilakukan oleh TLS *Handshake* jika menggunakan RSA adalah *Client Hello (client* mengirim pesan *hello* ke *server)*, *Server Hello* (*server* membalas pesan *hello client* dengan pesan *hello* yang dimiliki), *Authentication* (*client* memverifikasi sertifikat SSL *server*), *Premaster Secret* (*client* mengirimkan *premaster secret* yang dienkripsi dengan RSA), *Decrypt Premaster* (*server* mendekripsi *premaster secret)*, *Generate Session Key* (baik *client* maupun server membuat *session key)*, *Client Cipher Spec* (*client* mengirimkan pesan yang dienkripsi dengan *session key*), *Server Cipher Spec* (*server* memvalidasi pesan dari *client* dan juga mengirim pesan yang dienkripsi dengan *session key)*, *Secure Connection Established* (*client* memvalidasi pesan yang dikirim *server* lalu *secure connection* antara *client* dan *server* terbentuk)

Berdasarkan eksperimen yang dilakukan pada bagian sebelumnya, dapat ditarik kesimpulan bahwa program yang dibuat sukses untuk melakukan simulasi TLS *Handshake*. Dari tiga percobaan yang dilakukan, tiga-tiganya berhasil dimenangkan. Dari 3 percobaan yang dilakukan semua simulasinya berhasil menampilkan segmen yang benar, TCP 3-*Way Handshake*, pengiriman data, dan *closing connection* juga sukses sehingga harapannya simulasi yang dilakukan bisa menggambarkan dengan jelas bagaimana skema TLS *Handshake* pada protokol HTTPS diatas TCP.

PRANALA REPOSITORY GITHUB

Kode program dapat diakses melalui pranala berikut https://github.com/AndhikaRei/TLS-Handshake

REFLEKSI DAN APRESIASI

Penulis mengucapkan puji syukur kepada Allah SWT karena berkat kehendak-Nya penulis dapat menyelesaikan tugas pembuatan makalah ini dengan tepat waktu. Terima kasih juga kepada dosen pengajar mata kuliah IF4020 Kriptografi K-01, Dr. Ir. Rinaldi, M.T. Terimakasih juga kepada dua teman saya dari Teknik Informatika ITB 2019 yaitu Hokki Suwanda dan Reyhan Emyr Arrosyid karena kode program yang saya buat merupakan kode program pengembangan dari kode program yang saya kerjakan bersama mereka berdua. Terakhir, penulis ingin berterima kasih kepada semua pihak yang telah berkontribusi pada pembuatan makalah ini terutama kepada seluruh pemilik referensi yang telah penulis cantumkan karena pembuatan makalah ini tidak mungkin selesai tanpa adanya bantuan dari referensi tersebut.

Penulis sadar bahwa dalam penyusunan makalah ini terdapat banyak kekurangan seperti banyak penyederhanaan kasus, kurang rincinya percobaan, dan kurang lengkapnya simulasi yang dibuat. Penulis memohon maaf jika ada kesalahan baik dalam penulisan karena penulis masih dalam proses belajar untuk membuat makalah dengan baik dan benar. Akhir kata, penulis berharap pembuatan makalah ini dapat dikembangkan sehingga dapat bermanfaat untuk banyak orang.

REFERENCES

[1] Wibawa, S. W. (2021, September 3). *Bocornya 1,3 Juta Data eHac, Mengapa Terjadi Dan bahayanya Bagi Pasien Halaman all*. KOMPAS.com. Retrieved December 10, 2021, from https://www.kompas.com/sains/read/2021/09/03/080000023/bocornya-1-3-juta-data-ehac-mengapa-terjadi-dan-bahayanya-bagi-pasien?page=all.

[2] PT. Timedoor Indonesia. (n.d.). *APA ITU SSL? seberapa penting SSL untuk website?* Apa itu SSL? Seberapa penting SSL untuk website? Retrieved December 10, 2021, from https://id.timedoor.net/blogs/Apa-itu-SSL-Seberapa-penting-SSL-untuk-website/.

[3] Hiwarale, U. (2020, September 1). *A brief overview of the TCP/IP model, SSL/TLS/HTTPS protocols and SSL certificates*. Medium. Retrieved December 11, 2021, from https://medium.com/jspoint/a-brief overview-of-the-tcp-ip-model-ssl-tls-https-protocols-and-ssl-certificates-d5a6269fe29e.

[4] Munir, R. (2020). Algoritma RSA. Retrieved from http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Algoritma-RSA-2020.pdf

[5] Cloudflare, Inc. (n.d.). *What happens in a TLS handshake? | SSL handshake | cloudflare*. What happens in a TLS handshake? | SSL handshake. Retrieved December 11, 2021, from https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/.

[6] Sister (2021), Tugas Besar Jaringan Komputer 2. Retrieved December 13, 2021, from https://docs.google.com/document/d/10PTuxke98TE26YStT4KrSyZ5t-EWSVjVIRUPjLm238w/edit

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bojonegoro, 12 April 2021

Reihan Andhika Putra
13519043