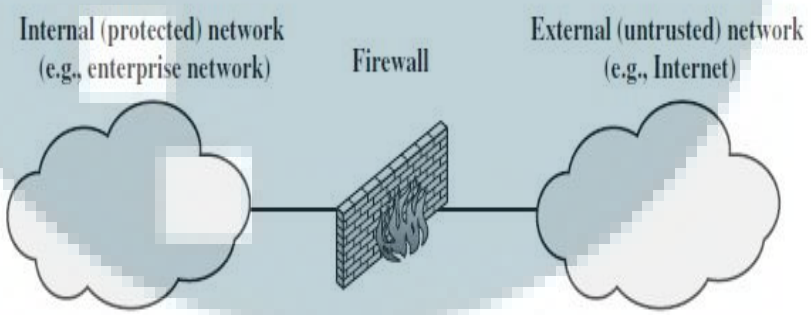
BAB II
DASAR TEORI

2.1 *Firewall*

Saat ini internet bukan lagi sebuah pilihan melainkan keharusan, namun internet tidak sepenuhnya aman bagi penggunanya karena berpotensi mengalami berbagai macam serangan. Ketika terjadi serangan, maka diperlukan peningkatan keamanan dengan membuat sistem perlindungan. Contoh sistem perlindungan yang umum digunakan yaitu *firewall*. *Firewall* diletakkan di antara jaringan dan internet untuk membangun koneksi yang aman dan menjadi dinding pembatas. Tujuan dinding pembatas adalah melindungi jaringan lokal dari serangan yang mungkin terjadi saat terhubung ke internet.



Gambar 2.1 Penggunaan *firewall* pada umumnya[8]

Dalam bukunya[8], Willliam Stalling menjelaskan bahwa *firewall* memiliki beberapa kelebihan diantaranya adalah:

1. *Firewall* melidungi jaringan dari pengguna yang tidak sah, melarang layanan yang rentan menimbulkan serangan untuk memasuki jaringan, memberikan perlindungan terhadap serangan berupa *IP spoofing* dan *routing attacks*. Penggunaan *firewall* memudahkan pengaturan keamanan karena mekanisme keamanan berada dalam sebuah sistem atau sekumpulan sistem.

2. *Firewall* mencatat lokasi saat dilakukan *monitoring* jaringan. Sistem *firewall* dapat memeriksa dan memberikan peringatan jika terdapat sesuatu yang membahayakan bagi jaringan.
3. *Firewall* merupakan sebuah *platform* yang nyaman bagi beberapa fungsi internet yang tidak berhubungan dengan keamanan, seperti *network address translator* (NAT) yang memetakan alamat pada jaringan lokal pada alamat internet dan fungsi pengaturan keamanan yang bertujuan untuk memeriksa dan mencatat penggunaan internet.
4. *Firewall* dapat berfungsi sebagai *platform* untuk IPsec. *Firewall* dapat digunakan untuk implementasi *virtual network*.

Namun selain memiliki kelebihan, *firewall* juga memiliki kelemahan, diantaranya adalah:

1. *Firewall* tidak dapat mengatasi serangan yang telah berhasil melewatinya.
2. *Firewall* tidak dapat melindungi dari ancaman yang berasal dari dalam sebuah jaringan internal, misalnya: jika ada individu yang berada di dalam jaringan internal bekerja sama dengan penyerang untuk melakukan serangan.
3. *Firewall* tidak dapat mendeteksi perangkat-perangkat internal yang terinfeksi, contohnya: perangkat-perangkat internal mungkin saja pernah terhubung dengan jaringan luar dan terinfeksi, ketika perangkat terhubung ke jaringan internal maka *firewall* tidak mendeteksi potensi serangan karena perangkat tersebut dapat melewati *firewall*.

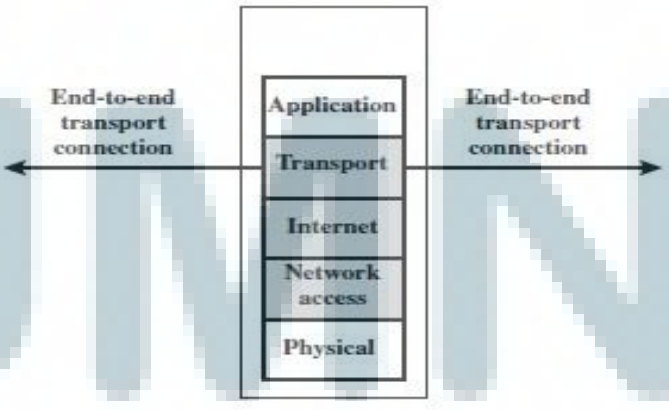
Pada prinsipnya *firewall* terdiri dari beberapa tipe, salah satunya adalah packet filtering *firewall* yang menerapkan beberapa peraturan untuk setiap paket IP yang masuk dan keluar dari *firewall* lalu memutuskan untuk meneruskan paket atau membuangnya.

Peraturan ditetapkan berdasarkan informasi yang terdapat pada paket IP, diantaranya adalah:

a. *Source IP address*: IP address sistem yang mengirimkan paket IP
b. *Destination IP address: IP address* sistem yang menjadi tujuan dari paket IP
c. *Source and destination transport-level address: transport level* (protokol TCP atau UDP) *port number*,untuk mendefinisikan layanan yang diinginkan misalnya *port* 22 untuk SSH.
d. *IP Protocol Field:* mendefinisikan jenis protokol yang digunakan.
e. *Interface:* untuk mendefinisikan interface mana yang digunakan untuk paket yang diterima dan interface mana yang digunakan untuk paket yang dikirimkan.

Setiap paket yang melalui *firewall* akan diperiksa apakah sesuai dengan peraturan yang ditetapkan oleh *firewall* atau tidak. Jika tidak maka *firewall* akan melakukan kebijakan default yang dimilikinya. Kebijakan tersebut terdiri dari dua macam, yaitu:

a. *Default = discard*, artinya paket tersebut akan dibuang
b. *Default = forward*, artinya paket tersebut akan diteruskan

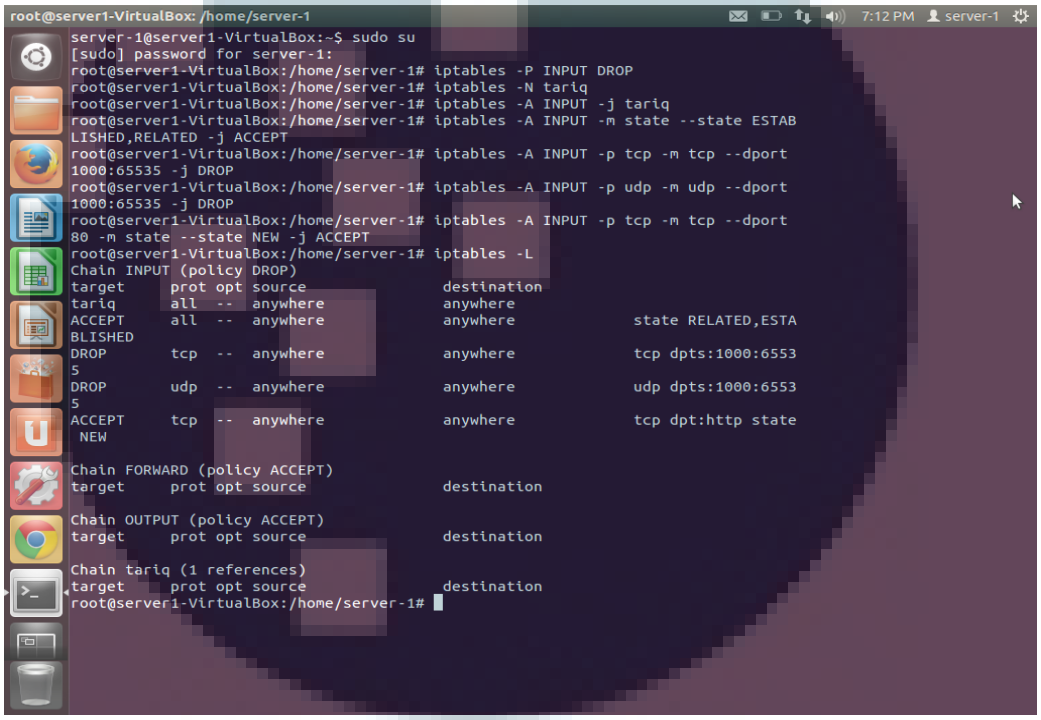


Gambar 2.2 Packet Filtering *Firewall*[8]

Dalam peneltian ini, tipe *firewall* yang digunakan adalah packet filtering *firewall*. Alasan pemilihan tipe ini adalah karena konsep *hybrid port knocking*adalah membatasi

setiap paket yang dikirim ke *firewall* kemudian memeriksa apakah paket memiliki *payload* (*image*) atau tidak. Jika tidak maka paket tersebut tidak akan diteruskan, namun jika iya maka paket akan diteruskan dan menjalani beberapa proses sebelum dapat mengakses *server*.

Gambar 2.3 menunjukkkan peraturan *firewall* menggunakan iptables untuk *hybrid port knocking*.



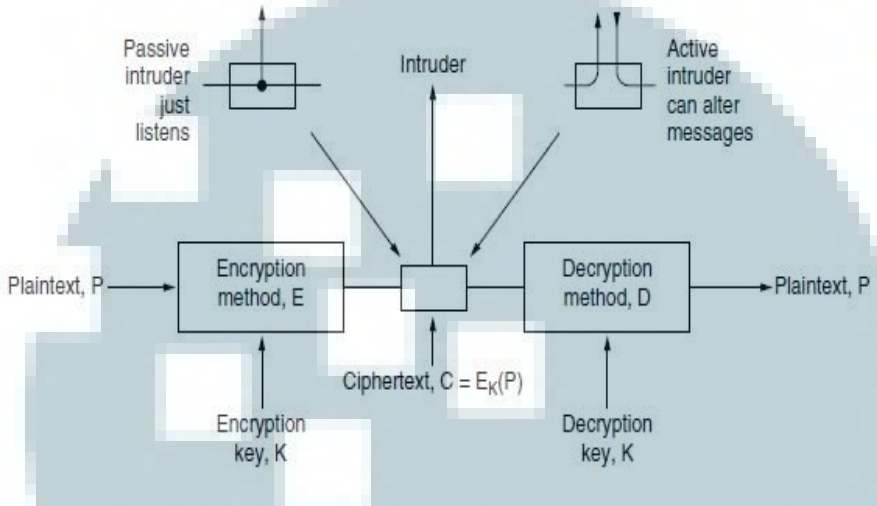
Gambar 2.3 iptables untuk *hybrid port knocking*

Berdasarkan gambar dapat dilihat bahwa peraturan *firewall* yang ditetapkan adalah membatasi semua paket yang memasuki *firewall*, membatasi paket yang mengirim tcp paket ke *port* number dengan *range* 1000-65535, membatasi paket yang mengirim udp paket ke *port* number dengan *range* 1000-65535, namun mengijinkan akses layanan http yang disediakan oleh *port* 80.

Firewall memiliki peranan penting pada teknik *hybrid port knocking*karena merupakan pembatas sekaligus *filter* terhadap paket yang dikirimkan oleh *client*. Paket yang memenuhi syarat dan telah melewati proses enkripsi, steganografi dan *mutual*

authentication kemudian akan diproses oleh *server* namun terlebih dahulu *server* akan meminta *firewall* untuk membukakan *port* yang diinginkan oleh *client*.

2.2 Kriptografi



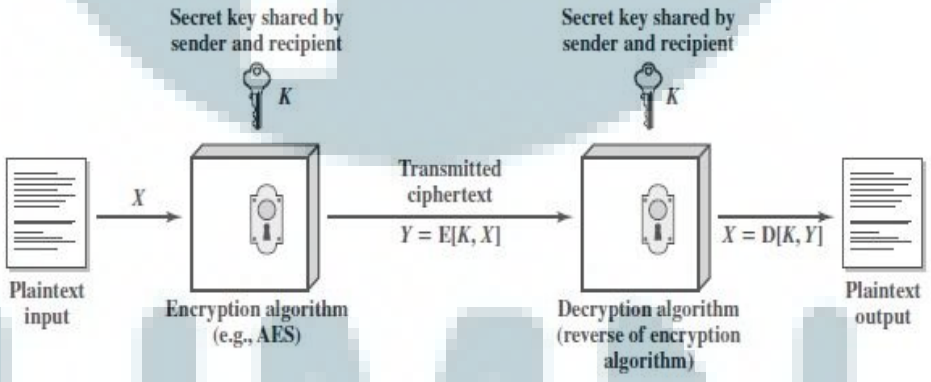
Gambar 2.4 Skema kriptografi [6]

Berdasarkan gambar di atas, pesan yang dienkripsi disebut dengan *plaintext*, kemudian ditransformasikan oleh sebuah fungsi dengan menggunakan sebuah kunci sebagai parameternya. Hasil dari proses enkripsi disebut *ciphertext*, kemudian dikirmkan kepada penerima pesan. Pesan tersebut kemudian dapat diketahui oleh penyerang dalam hal ini dinamakan sebagai *intruder* namun tidak dapat dengan mudah didekripsikan karena tidak memiliki kuncinya *(passive intruder).* Sebaliknya ada juga *intruder* yang tidak hanya mengetahui pesan, namun ia juga merekam kemudian mengirimkan kembali pesan yang telah dirubah isinya sehingga penerima akan menerima pesan yang salah *(active intruder)*.

Seni untuk memecahkan kode *ciphertext* disebut dengan kriptanalis, sedangkan seni untuk merancang kode *ciphertext* disebut dengan kriptografi. Secara keseluruhan semua prosesnya dikenal dengan kriptologi[6].

Dalam penelitian ini, digunakan enkripsi simetris untuk menyembunyikan isi pesan yang dikirim. Menurut William Stalling[8], skema enkripsi simetris memiliki 5 komponen pendukung, diantaranya adalah:

a. *Plaintext*, yaitu pesan asli atau data yang diolah dengan algoritma untuk menjadi input.
b. Algoritma enkripsi, yaitu algoritma enkripsi yang melakukan proses transposisi dan substitusi pada *plaintext*.
c. *Secret key*, yaitu input untuk algoritma. Ketepatan proses transposisi dan substitusi bergantung pada kunci ini.
d. *Ciphertext*, yaitu pesan acak yang dihasilkan sebagai output. *Ciphertext* bergantung pada *plaintext* dan *secret key*.
e. *Decryption algorithm*, yaitu algoritma yang menggunakan *ciphertext* dan *secret key* untuk menghasilkan pesan asli (*plaintext*).



Gambar 2.5 Model enkripsi simetris[8]

Gambar 2.5 memodelkan enkripsi simetris. Cara kerjanya adalah penerima mengirim *plaintext* sebagai input yang diproses dengan algoritma enkripsi untuk menghasilkan *ciphertext*. Pengirim kemudian *generate secret key* dan mengirimkan *secret key* kepada penerima dan mengirimkan *ciphertext* kepada penerima. *Ciphertext*

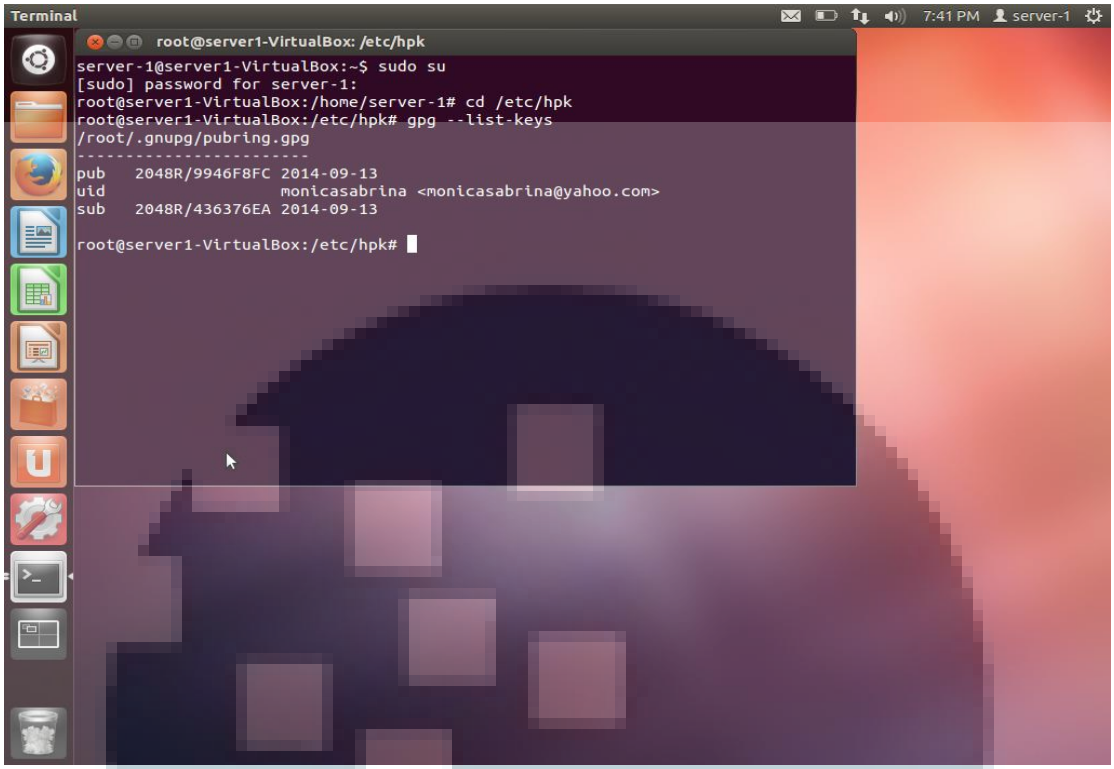
kemudian diproses oleh penerima menggunakan algoritma dekripsi dan *secret key* yang telah diterima sehingga menghasilkan *output* berupa *plaintext*.

2.2.1 *Public-Key Algorithm*

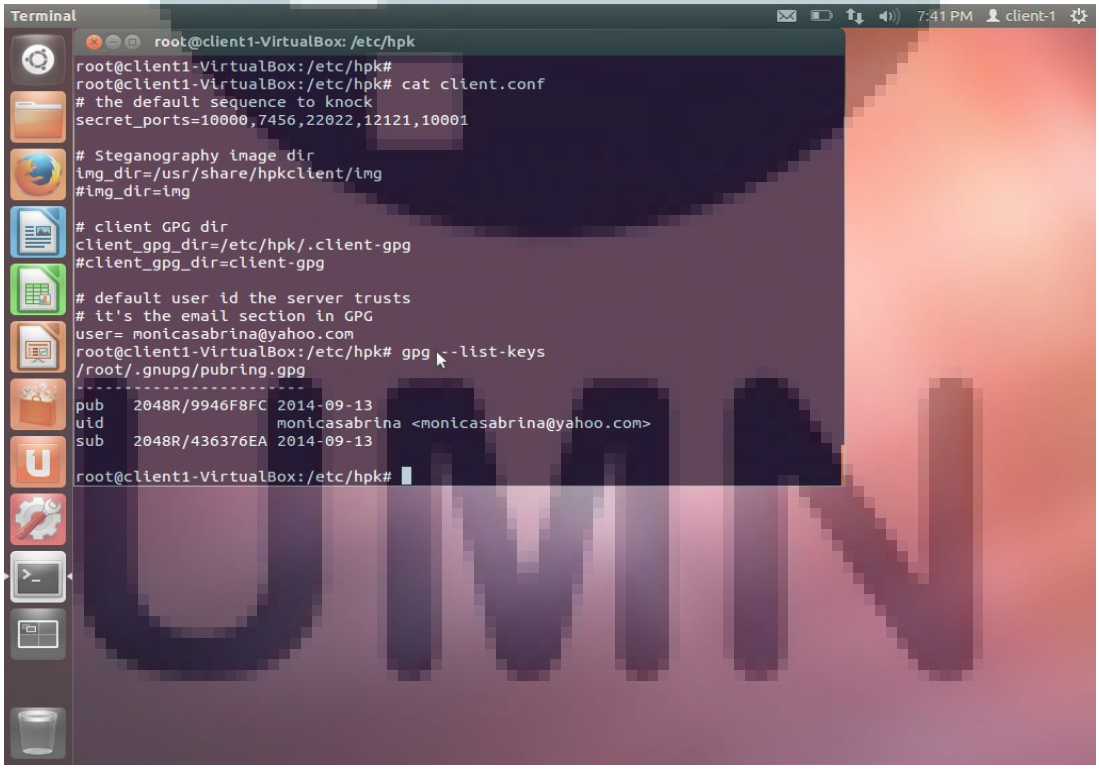
Definisi enkripsi *public key* adalah enkripsi dimana pengirim pesan merancang dua buah algoritma yang digunakan untuk mengubah *plaintext* menjadi *ciphertext* dan *generate public key* [6]. *Public key* akan diberikan kepada penerima pesan. Pesan yang berupa *ciphertext* dikirim dan didekripsikan oleh penerima menggunakan *secret key* nya. Dengan begitu pesan yang dikirim akan aman karena hanya pemegang *secret key* yang dapat mengetahui isi pesannya. *Public-key algorithm* misalnya RSA, Elgamal, DSA dan lain sebagainya.

Pada aplikasi *client-server* yang menggunakan metode *hybrid port knocking*, mekanisme kriptografi yang digunakan adalah enkripsi simetris berbasis GnuPG dengan gabungan *public-key algorithm* seperti RSA dan RSA, DSA dan Elgamal, serta RSA dan DSA. *Client* yang ingin berkomunikasi dengan *server* terlebih dahulu melakukan *generate key pair* (*private key* dan *public key*) dengan memilih *public key algorithm* yang diinginkan lalu *export public key* dan *private key* untuk di-im*port* oleh *server* dan status *keys* dirubah menjadi *trusted*.

Kerahasiaan pesan yang menggunakan metode enkripsi simetris sangat bergantung pada *private key* nya, sehingga *private key* ini harus dijaga keamanannya. Berdasarkan alasan ini maka baik di sisi *client* maupun *server*, direktori tempat penyimpanan kunci GnuPG harus diatur *permission*-nya menggunakan *chmod* 600 yang artinya hanya pemilik yang dapat mengakses direktori tersebut.



Gambar 2.6 GnuPG key di sisi *server*



Gambar 2.7 GnuPG key di sisi *client*

2.2.1.1 RSA

RSA merupakan algoritma yang dirancang oleh 3 orang yang bernama Ron Rivest, Adi Shamir, and Len Adleman. Nama RSA diambil dari nama belakang dari masing-masing perancang. Algoritma ini dipublikasikan untuk pertama kalinya pada tahun 1978. Seiring dengan pertambahan waktu, algoritma ini menjadi algoritma yang banyak digunakan pada implementasi sistem dengan pendekatan *public key algorithm*.

Skema RSA adalah sebuah *block cipher* dimana *plaintext* dan *ciphertext*-nya merupakan bilangan bulat antara 0 dan n-1[7]. Ukuran n pada umumnya adalah 1024 bits atau 309 digit decimals. Artinya nilai n tidak lebih dari 2^{1024}. *Plaintext* dienkripsi menjadi blok-blok, dimana setiap blok memiliki nilai biner lebih kecil dari beberapa n. Artinya ukuran blok harus kurang dari atau sama dengan $log2(n) + 1$.

Pada prakteknya ukuran blok adalah i bits, dimana 2^i 6 $n \leq 2^i+1$. Enkripsi dan dekripsinya ditunjukkan oleh rumus berikut, untuk beberapa blok *plaintext* M dan blok *ciphertext* C.

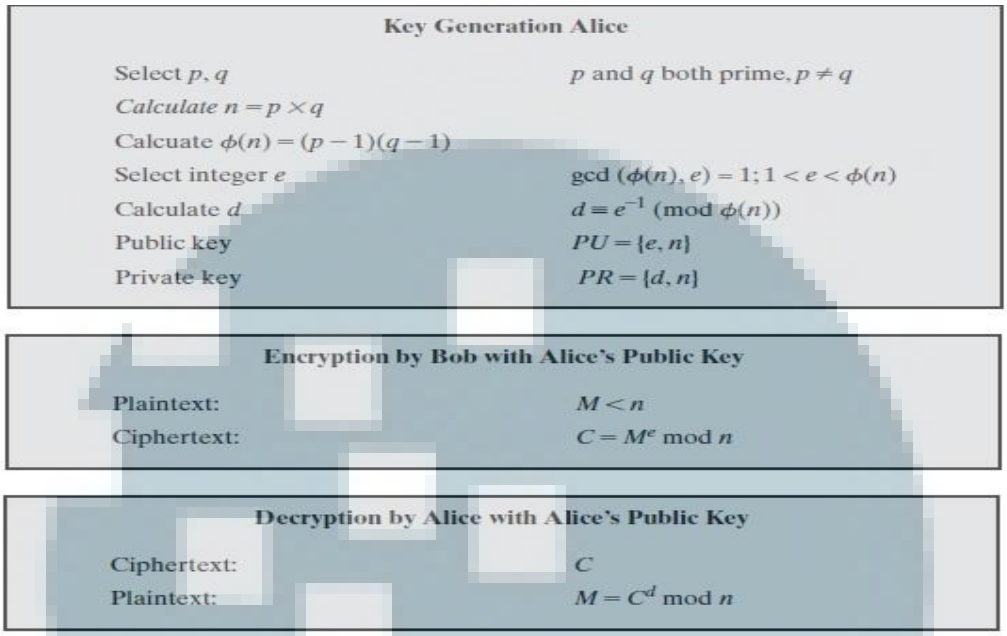
$$C = M^e \bmod n$$
$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Rumus 2.1 Enkripsi dan Dekripsi untuk beberapa blok *plaintext* M dan blok *ciphertext* C. [7]

Pengirim dan penerima harus mengetahui nilai n. Pengirim mengetahui nilai e, dan hanya penerima yang mengetahui nilai d sehingga *public key*-nya menjadi PU = {e,n} dan *private key*-nya PR = {d,n}. Untuk dapat memenuhi algoritma ini ada beberapa syarat yang harus dipenuhi:

1. Mengetahui nilai e,d,n sehingga $M^{ed} \bmod n = M$, untuk semua $M < n$
2. Mudah untuk menghitung $M^{ed} \bmod n$ dan $C^d \bmod n$ untuk semua nilai $M < n$
3. Nilai d tidak mudah diketahui, meskipun e dan n nilainya diketahui.

Gambar di bawah mendeskripsikan keseluruhan algoritma RSA yang diilustrasikan dengan dua tokoh yaitu Alice dan Bob.



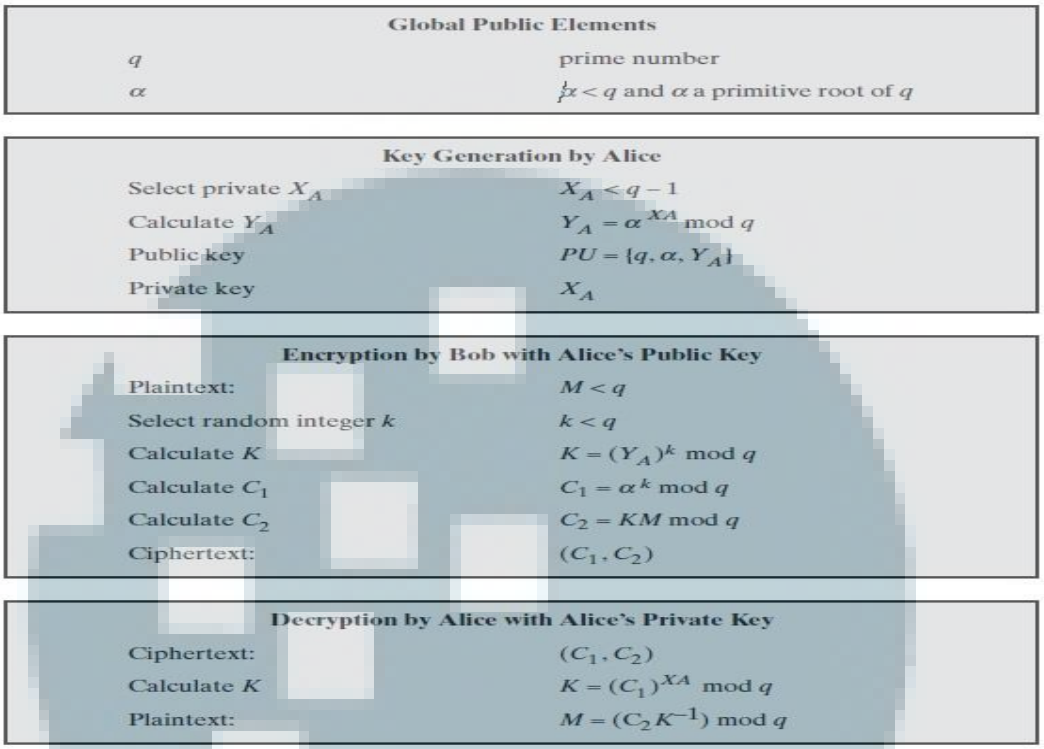
Gambar 2.8 Algoritma RSA[7]

2.2.1.2 ElGamal

ElGamal merupakan *public key algorithm* yang memiliki elemen global berupa bilangan pokok q dan α. Proses yang terjadi pada ElGamal terdiri dari beberapa langkah yang akan dibahas berikut ini[7]:

1. Pengirim *generate* bilangan bulat secara *random* yaitu k.
2. Pengirim *generate one-time key* yaitu K menggunakan komponen *public key* Y_A, q dan k yang dimiliki penerima.
3. Pengirim mengenkripsi k menggunakan kompoen *public key* α menghasilkan C_1. C_1 menyediakan informasi yang diperlukan Alice untuk mendekripsi K.
4. Pengirim mengenkripsi pesan *plaintext* M menggunakan K
5. Penerima mendekripsi K dari C_1 yang telah diterima menggunakan *private key* nya.

6. Penerima menggunakan K^{-1} untuk memperoleh *plaintext* dari C_2.



Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice	
Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$PU = \{q, \alpha, Y_A\}$
Private key	X_A

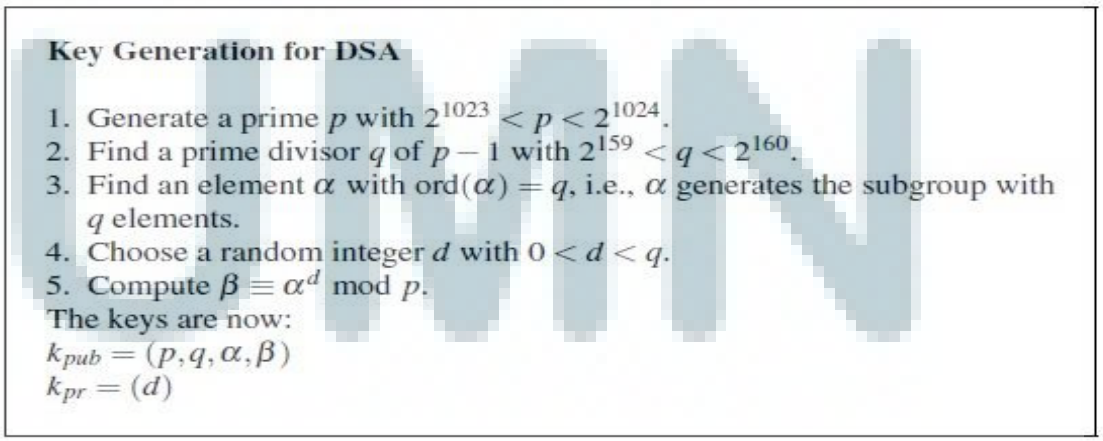
Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Gambar 2.9 Proses Algoritma ElGamal[7]

2.2.1.3 DSA

DSA *(Digital Signature Algorithm*) diusulkan oleh *National Institute of Standards and Technology* (NIST). Kelebihannya adalah ukuran *signature*-nya hanya 320 bit[9]. Penjelasan lebih lanjut dapat dilihat pada gambar di bawah



Key Generation for DSA

1. Generate a prime p with $2^{1023} < p < 2^{1024}$.
2. Find a prime divisor q of $p - 1$ with $2^{159} < q < 2^{160}$.
3. Find an element α with $\text{ord}(\alpha) = q$, i.e., α generates the subgroup with q elements.
4. Choose a random integer d with $0 < d < q$.
5. Compute $\beta \equiv \alpha^d \bmod p$.

The keys are now:

$k_{pub} = (p, q, \alpha, \beta)$

$k_{pr} = (d)$

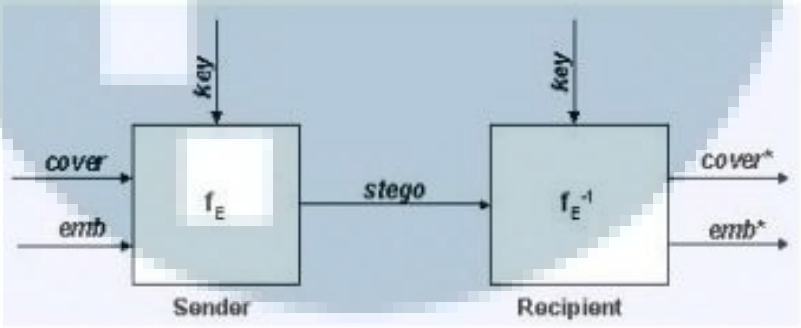
Gambar 2.10 Proses DSA[9]

Untuk dapat melakukan *generate public key* maka perlu mencari nilai dari elemen p, q, α , dan β. Nilai elemen-elemen tersebut diperoleh dengan ketentuan yang terdapat pada gambar 2.10. Sebaliknya untuk *private key*, perlu untuk mencari nilai d dengan melakukan langkah no 5 pada gambar.

DSA tidak menyediakan mekanisme enkripsi dan dekripsi sehingga dalam penggunaannya perlu digabungkan dengan algoritma lain seperti RSA dan ElGamal.

2.3 Steganografi

Steganografi adalah seni untuk menyembunyikan informasi dan usaha untuk menyembunyikan keberadaan pesan yang dikirm[10]. Pesan disembunyikan ke dalam sebuah media pembawa sehingga tidak mudah diketahui oleh orang lain. Media pembawa yang umum digunakan diantaranya adalah gambar, audio, video, teks dan lain sebagainya.



Gambar 2.11 Proses Steganografi [10]

Sebuah pesan berisi informasi yang berupa *plaintext*, *ciphertext*, gambar dan lain-lain yang di-*embbed* ke dalam sebuah *bit stream*. Lalu media pembawa dan pesan yang di-*embbed* disatukan dan membentuk *stego-carrier*. Pesan yang disembunyikan membutuhkan *stego-key*, yaitu informasi rahasia seperti *password* yang diperlukan untuk meng-*embbed* informasi.

Formula untuk merepresentasikan proses steganografi adalah *cover medium* + *embedded message* + *stego-key* = *stego-medium*.

2.4 *Mutual authentication*

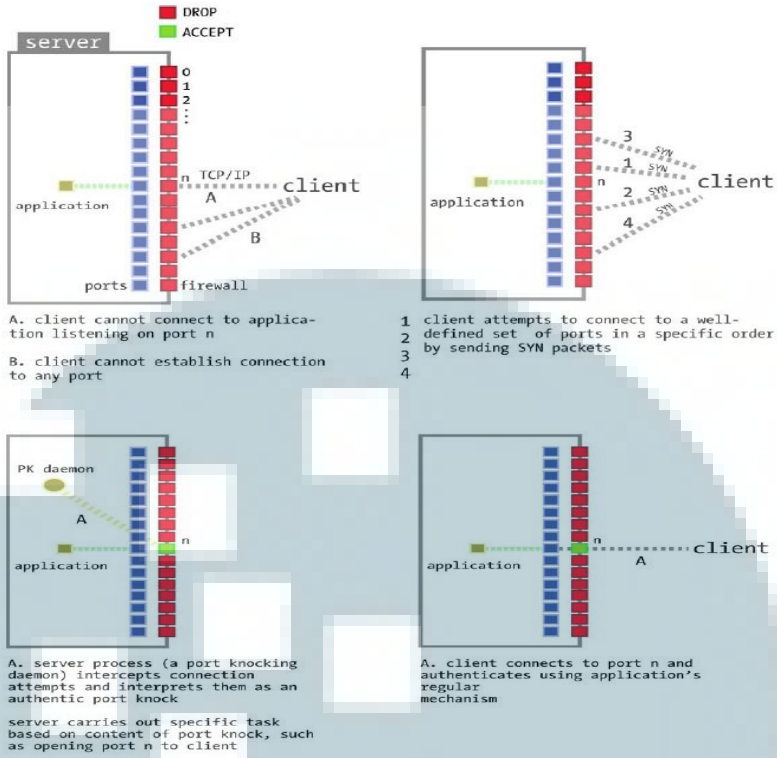
Mutual authentication adalah mekanisme autentikasi yang dilakukan pada kedua belah pihak, misalnya *server* melakukan autentikasi terhadap *client* dan *client* melakukan autentikasi terhadap *server*[11]. Pada umumnya *mutual authentication* dimplemetasikan pada sistem berbasis *digital certificates*.

Penggunaan *mutual authentication* dalam *hybrid port knocking*adalah untuk mengautentikasi *client* dan *server* yang sedang berkomunikasi. Jika dalam proses autentikasi, *client* bukan merupakan *client* yang benar maka *server* akan memblokir ip address *client* tersebut. Begitu juga jika *server* bukan merupakan *server* yang benar maka *client* akan memutuskan komunikasi dengan *server*.

2.5 Penelitian sebelumnya

Metode *port knocking*telah banyak dikembangkan dan diimplementasikan sebelumnya. Diantaranya adalah *port knocking*, *knockknock*, dan *hybrid port knocking*.

*Port knocking*atau yang dikenal dengan *traditional port knocking*merupakan sistem yang mekanisme autentikasi penggunanya berbasis *firewall* dan memanfaatkan *port* tertutup untuk melakukan autentikasi sehingga memungkinkan terjadinya komunikasi pada jaringan yang semua *port*-nya ditutup[1]. Komunikasi tersebut memanfaatkan *firewall log*, dimana semua percobaan koneksi dicatat. *Log* tersebut mengawasi *sequence port* tertentu jika ada yang meng-*encode* informasi yang berisi perintah untuk mengubah peraturan *firewall* misalnya membuka atau menutup *port* bagi sebuah alamat IP.



Gambar 2.12 Cara kerja *port knocking*[2]

Cara kerja *port knocking*[2] ditunjukkan oleh gambar 2.12, dimana pada awalnya *client* tidak dapat terhubung ke aplikasi yang listen *port* tertentu (n) dan *client* tidak dapat terhubung dengan *port* manapun. Lalu *client* melakukan usaha koneksi berupa *port knock sequence* dengan mengirimkan SYN packet. *Server* selanjutnya akan memroses usaha koneksi yang diterima dan melakukan autentikasi *port knock*. *Server* melakukan tugasnya sesuai dengan isi *port knock*, misalnya: untuk membuka *port* n untuk *client*. *Client* terhubung ke *port* n dan dapat mengakses layanan di *server* melalui *port* tersebut.

Kelebihan *port knocking*adalah sistemnya tersembunyi karena penyerang tidak mudah menentukan apakah *server* sedang mendengarkan *port knock* atau tidak. *Port knocking*juga merupakan sistem yang fleksibel karena autentikasinya berupa *port knock sequence*. Namun masih terdapat kekurangan yaitu *port knocking*tidak dapat digunakan untuk melindungi *port* yang menyediakan layanan yang bersifat publik seperti *web* atau *mail services*. Jika *port knocking*diimplementasikan pada sebuah *host* untuk melindungi

port (ssh, telnet, ftp, dan lain-lain) sedangkan *port* lainnya dibiarkan terbuka untuk publik maka hanya tersisa dua pilihan. Pilihannya adalah menempatkan *host* yang menyediakan *port* untuk layanan publik pada jaringan DMZ (*Demilitarized Zone*), dimana jaringan DMZ berupa jaringan yang dilengkapi dengan firewall untuk membatasi lalu lintas jaringan baik yang masuk maupun keluar pada *server* sehingga *server* terlindungi. Pilihan lainnya untuk mengatasi kekurangan *port knocking*adalah memisahkan *server* yang berisi data penting dan rahasia dengan *server* yang digunakan untuk mengakses layanan publik.

*Port knocking*kemudian dikembangkan lagi oleh seseorang bernama Moxie Marlinspike yang menamakan aplikasinya "*knockknock*"[3]. *"knockknock*" merupakan aplikasi yang sederhana karena beberapa alasan yang diungkapkan oleh pembuatnya, yaitu:

1. Sebuah aplikasi seharusnya berukuran kecil dan tidak memiliki banyak persyaratan.
2. Aplikasi tidak dijalankan di *kernel*
3. Aplikasi tidak dirancang untuk *bind* ke *socket*.
4. Tidak menggunakan *software* seperti libpcap untuk meng-*capture* paket dan memeriksa setiap paket.
5. Aplikasi tidak menggunakan protokol UDP
6. Aplikasi tidak *generate request port knock* dengan jelas karena hal ini membuat penyerang dapat mengetahui *port* mana yang akan dibuka.
7. Aplikasi tidak mengirimkan lebih dari satu paket ke jaringan.
8. Menggunakan skema IND-CCA *secure*

Cara kerja *knockknock* [3] adalah sebagai berikut:

1. *Server meng-execute* aplikasi *python "knockknock-daemon"* dan *client meng-execute* aplikasi *python "knockknock".*
2. *"Knocknock-daemon"* hanya terhubung dengan *kern.log.* "*Knocknock-daemon"* tidak *bind* ke *socket*, tidak menggunakan libpcap (*software* yang digunakan untuk meng-*capture* paket), tidak memeriksa setiap paket dan tidak mengirimkan sesuatu ke jaringan.
3. Ketika *client* akan membuka *port*, maka *client* harus *meng-execute "knockknock"* yang mengirimkan *single SYN packet* ke *server*. IP dari paket dan TCP *header*-nya akan di-*encode* sesuai dengan skema IND- CCA *secure* untuk mengenkripsi *request* yang berisi perintah membuka *port* dari alamat IP tersebut.
4. Informasi tentang paket dicatat di *kern.log* dan diproses oleh *"knockknock-daemon*" yang bertugas untuk melakukan validasi.
5. *Client* terhubung ke *server* melalui *port* yang telah dibuka
6. Setelah *request client* telah dipenuhi oleh *server*, *port* ditutup kembali dan tidak mengijinkan adanya koneksi lain.

Aplikasi *knockknock* dapat mengatasi beberapa serangan seperti *eavesdropping, replay attack* dan lain sebagainya sebab penyerang tidak menyadari bahwa SYN *packet* yang dikirim oleh aplikasi *python "knockknock"* merupakan *port knock request*. Tetapi jika penyerang menyadarinya, ia tidak dapat dengan mudah menentukan *port* yang akan dibuka karena informasinya telah dienkripsi.

Hybrid port knocking(HPK)[4] adalah pengembangan *port knocking*yang menggabungkan teknik *port knocking*, steganografi dan *mutual authentication*. Penggabungan tiga teknik tersebut bertujuan untuk lebih meningkatkan keamanan karena

pada prinsipnya HPK dirancang untuk dapat mengatasi serangan yang sering terjadi seperti *Denial of Service* dan *TCP replay attack*.

Pada HPK, *client* mengirimkan *payload* yang berisi *image* melalui protokol TCP. Payload tersebut membawa informasi tentang *client's request* misalnya *request* untuk membuka *port*. HPK memiliki beberapa kelebihan yaitu tidak hanya dapat memroses *request* untuk membuka atau menutup *port*, namun juga dapat mengirimkan *request remote command*, HPK dapat mengatasi serangan *Denial of Service* karena terdapat sistem deteksi untuk mencegah *host* yang akan melakukan serangan, HPK dapat mengatasi serangan TCP *replay attacks* karena menggunakan teknik kriptografi dan steganografi untuk mengirimkan paket serta menggunakan *mutual authentication* untuk memeriksa apakah *client* dan *server* merupakan *client* dan *server* yang sah.