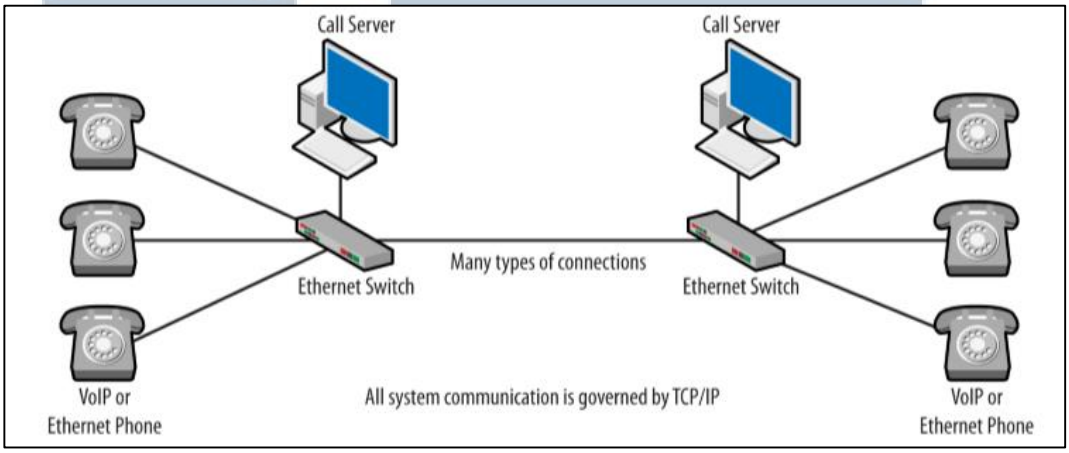
BAB II

LANDASAN TEORI

2.1 VoIP (Voice over Internet Protocol)

VoIP merupakan teknologi mengirimkan suara melalui sebuah jaringan berbasis IP. VoIP mengambil semua pesan signal dan meletakkannya ke dalam sebuah paket IP (Hartpence, 2013). Arsitektur sederhana dari aplikasi VoIP dapat digambarkan sebagai berikut.



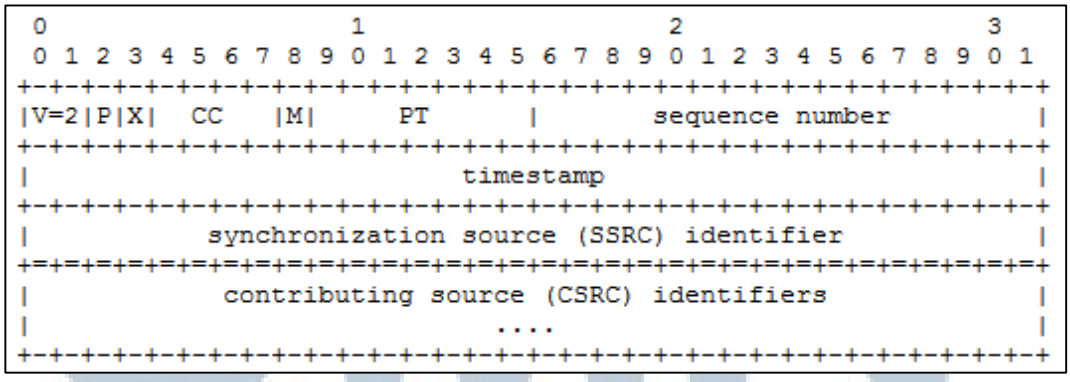
Gambar 2.1 Arsitektur Sederhana VoIP
(Sumber : Hartpence, 2013)

Dalam VoIP terdapat *encoding* pada saat paketisasi sinyal suara menggunakan *codec* yang berfungsi untuk kompresi dan dekompresi suara seperti contoh iLBC, G.711, dan G.729 (Purbo, 2007). Setelah melalui proses *encoding*, data suara kemudian dipecah menjadi paket yang lebih kecil, dan dikirim melalui kanal yang berbeda – beda antara pengirim dan penerima (Ridwan, dkk., 2011). Menurut Bur Goode (2002) pada *paper* berjudul "*Voice Over Internet Protocol (VoIP)*", *delay* yang masih dapat diterima pengguna adalah 0 – 400 ms jika melebihi maka delay tersebut dapat membuat kualitas aplikasi menjadi buruk. Menurut penelitian yang juga dilakukan oleh Fiona Fui-Hoon Nah (2003) dengan judul

"*A Study On Tolerable Waiting Time : How Long Are Web User Willing To Wait?",* jika waktu respon dari komputer telah melewati 10 detik maka pengguna akan melakukan kegiatan lain ketika menunggu komputer untuk menyelesaikan proses.

2.2 RTP (Realtime Transport Protocol)

RTP merupakan protokol yang didesain untuk keperluan servis pengiriman data *end-to-end* dengan karakteristik yang *real-time,* seperti *audio* dan *video*. Protokol ini juga dilengkapi oleh tipe *payload, sequence numbering, timestamping,* dan *delivery monitor*. RTP biasanya berjalan diatas UDP (*User Datagram Protocol*) untuk dapat menggunakan servis *multiplexing* dan *checksum* (Schulzrinne, dkk., 2003). UDP sendiri merupakan *transaction oriented protocol* yang memberikan kemampuan program untuk mengirim pesan dengan mekanisme protokol yang minimum tetapi tidak memiliki jaminan pengiriman dan perlindungan pengiriman duplikasi (Postel,1980).



Gambar 2.2 RTP Header
(Sumber : ietf.org, 2003)

Gambar 2.2 merupakan bentuk susunan *header* dari paket protokol RTP yang terletak setelah header UDP (ietf.org, 2003).

1. Version (V) : 2 bit

Area ini merupakan identifikasi versi dari RTP yang biasanya didefinisikan sebagai dua.

2. Padding (P) : 1 bit

Jika *padding* digunakan , maka paket mengandung satu atau lebih tambahan *padding octet* pada akhir paket yang tidak termasuk ke dalam *payload.*

3. Extension (X) : 1 bit
4. CSRC Count (CC) : 4 bit
5. Payload Type (PT) : 7 bit

   Bagian ini berfungsi sebagai identifikasi format dari RTP *payload* dan menentukan interpretasinya oleh aplikasi.

6. Sequence Number : 16 bit

   Sequence number sendiri bertambah 1 setiap paket data RTP yang dikirim, dan dapat digunakan oleh penerima untuk mendeteksi paket yang hilang dan untuk mengembalikan urutan paket.

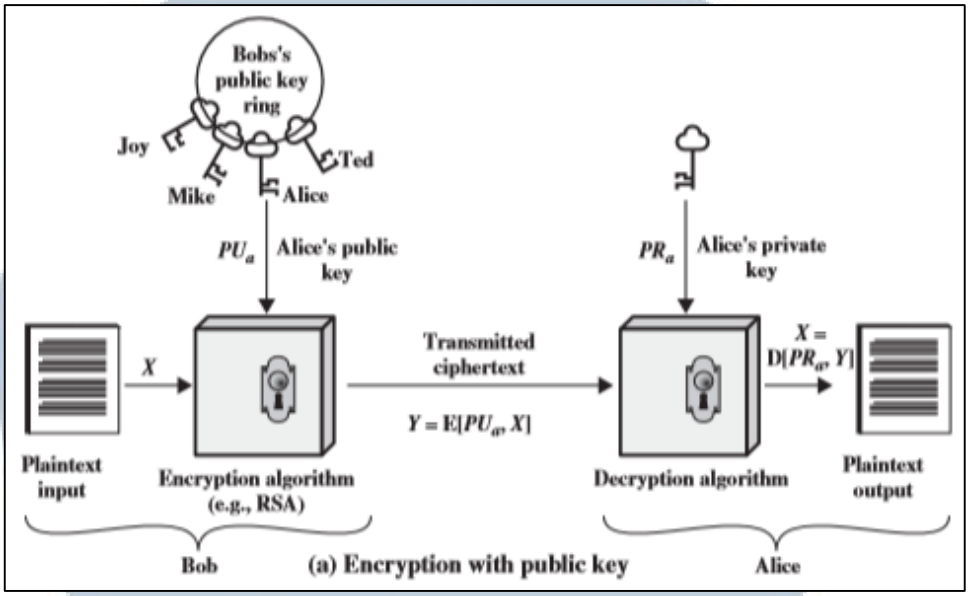
7. Timestamp : 32 bit

   Timestamp menggambarkan contoh *instant* dari oktet pertama pada paket data RTP. Contoh *instant* harus diturunkan dari jam yang bertambah bersamaan di dalam suatu waktu yang diggunakan untuk sinkronisasi dan penghitungan *jitter*.

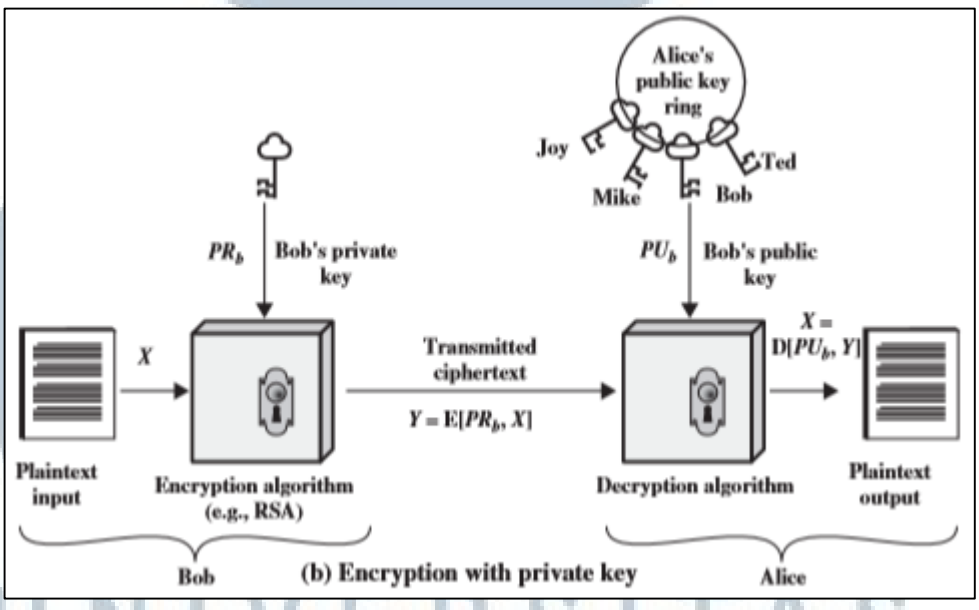
2.3 Public Key Cryptosystem

Public key crptosystem memiliki karakteristik dimana, setiap kunci yang berhubungan dapat digunakan untuk enkripsi dengan yang lainnya digunakan untuk dekripsi (Stallings, 2011). Sistem kriptografi kunci publik, dapat memiliki skema

kunci publik yang digunakan untuk enkripsi atau kunci pribadi yang digunakan untuk enkripsi yang tertuang dalam gambar di bawah.



Gambar 2.3 Skema enkripsi menggunakan kunci publik
(Sumber : Stallings, 2011)



Gambar 2.4 Skema enkripsi menggunakan kunci pribadi
(Sumber : Stallings, 2011)

Pada sistem kriptografi kunci publik, memiliki komponen sebagai berikut (Stallings, 2011) :

1. *Plaintext* merupakan pesan yang dapat dibaca atau data yang akan dienkripsi
2. *Encyption algorithm* merupakan algoritma enkripsi yang digunakan untuk merubah *plain text* ke *chiper text*
3. *Public and provate keys* merupakan sepasang kunci yang dipilih dan digunakan untuk melakukan enkripsi, dan yang lain diperlukan untuk dekripsi.
4. *Chiper text* adalah hasil pesan yang telah diacak yang berasal dari *plain text.*
5. *Decryption algorithm* adalah algoritma yang digunakan untuk melakukan dekripsi pada *chipertext* sehingga mengeluarkan pesan asli yang dapat dibaca.

Ada beberapa langkah penting dalam sistem kriptografi menggunakan kunci publik diantaranya adalah (Stallings, 2011) :

1. Setiap pengguna menghasilkan sepasang kunci yang digunakan untuk enkripsi dan dekripsi pesan.
2. Setiap pengguna meletakkan satu dari dua kunci di dalam *public register* atau file lain yang dapat diakses. Inilah yang disebut kunci publik, sedangkan kunci pengguna lain tetap *private.*
3. Ketika pengguna A ingin mengirim pesan ke pengguna B maka, pengguna A melakukan enkripsi pesan menggunakan kunci pubik pengguna B.
4. Saat pengguna B menerima pesan maka, pengguna B melakukan dekripsi menggunakan kunci pribadi pengguna B.

2.4 RSA (Rivest-Shamir-Adleman)

Algoritma enkripsi RSA merupakan algoritma enkripsi yang menggunakan skema kriptografi asimetris. Algoritma ini menerapkan skema kunci publik , dimana pengguna A dapat mengirim pesan terenkripsi ke pengguna B tanpa harus bertukar kunci rahasia (Ireland, 2010).

Langkah – langkah dalam menghasilkan kunci adalah (Ireland, 2010):

1. Hasilkan dua bilangan prima acak p dan q.
2. Lalu hitung n = pq dan (phi) φ = (p-1)(q-1).
3. Pilih bilangan bulat *e*, dengan 1 < *e* < phi, dimana FPB(e, phi) = 1.
4. Hitung eksponensial rahasia d, 1 < d < phi seperti ed ≡ 1 (mod phi).
5. Nilai kunci publik adalah (n, e) dan kunci *private* (d, p, q).

Ringkasan algoritma RSA dapat dituliskan sebagai berikut (Ireland, 2010) :

1. n = pq (dimana p dan q merupakan bilangan prima yang berbeda). ...(2.1)
2. Phi, φ = (p-1)(q-1) ...(2.2)
3. *e* < n, seperti FPB(*e*, phi) = 1 ...(2.3)
4. $d = e^{-1} \bmod phi$ ...(2.4)
5. $c = m^e \bmod n, 1 < m < n$ ...(2.5)
6. $m = c^d \bmod n$ ...(2.6)

Dimana :

n adalah modulus

e adalah *public exponent*

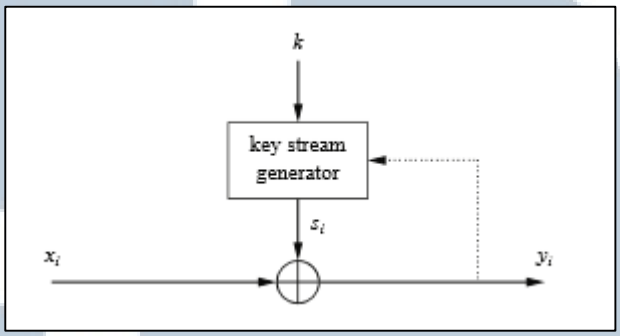
d adalah *secret exponent*

c adalah *cipher text*

m adalah *message*

2.5 Stream Cipher

Stream cipher merupakan teknik kriptografi dengan mengenkripsi bit secara individual (Paar dan Pelzl, 2010). Berikut merupakan ilustrasi dari *stream cipher*.



Gambar 2.5 Ilustrasi Proses Stream Cipher
(Sumber : Paar dan Pelzl, 2010)

Proses enkripsi dan dekripsi dari *stream cipher* melibatkan rumus :

Enkripsi :

$$y_i = e_{si}(x_i) \equiv x_i + s_i \bmod 2 \quad ...(2.7)$$

Dekripsi :

$$x_i = d_{si}(y_i) \equiv y_i + s_i \bmod 2 \quad ...(2.8)$$

2.6 Salsa20

Salsa20 merupakan algoritma enkripsi *stream cipher* yang didesain oleh Daniel J. Bernstein (Bernstein, 2012). Salsa20 membentuk 16 *words*, dengan setiap *word* memiliki ukuran 32 bit dan dapat digambarkan ke dalam 4 x 4 *matrix* sebagai berikut :

$$X = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_5 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}$$

Pada *matrix* yang paling kanan merupakan inisialisasi yang menggunakan empat *constant* yang telah ditentukan $c_0, \ldots, c_3$, kunci sepanjang 256 bit $k_{0,\ldots,}k_7$, *nonce* / IV (*Initialization Vector*) sepanjang 64 bit v_0, v_1, dan *counter* sepanjang 64 bit t_0, t_1. Tetapi jika kunci memiliki panjang 128 bit, maka nilai $k_i = k_{i+4}$ dimana, nilai *i* lebih besar atau sama dengan tiga (Maitra, dkk., 2005). Fungsi matematika pada Salsa20 terdiri dari beberapa tahap, diantaranya *quarterround* dimana pada tahap ini adalah pertambahan antara dua *word* dan melakukan operasi XOR sesudah dirotasi sebanyak 7, 9, 13, dan 18, lalu *rowround function* yang akan memodifikasi setiap baris *matrix* sesudah melakukan *columnround function* yang memodifikasi kolom pada *matrix*. Kemudian dilakukan *looping* sebanyak yang dispesifikasikan (Afdhila, dkk., 2016).

Berikut merupakan fungsi matematika dari *quarterround function,* jika y = (y0; y1; y2; y3), maka (Afdhila, dkk., 2016).

quarterround(y) = (z0; z1; z2; z3) ...(2.9)

dimana :

z1 = y1 XOR ((y0 + y3) <<< 7)

z2 = y2 XOR ((z1 + y0) <<< 9)

z3 = y3 XOR ((z2 + z1) <<< 13)

z0 = y0 XOR ((z3 + z2) <<< 18

Kemudian terdapat *rowround function*, langkah ini memodifikasi setiap baris pada *matrix*. Jika terdapat matrix y = (y0, y1, y2, y3, ... , y15) maka (Afdhila, dkk., 2016).

Rowround(y) = (z0, z1, z2, z3, ... , z15) ...(2.10)

dimana :

(z0, z1, z2, z3) = quarterround(y0, y1, y2, y3)

(z5, z6, z7, z4) = quarterround(y5, y6, y7, y4)

(z10, z11, z8, z9) = quarterround(y10, y11, y8, y9)

(z15, z12, z13, z14) = quarterround(y15, y12, y13, y14)

Kemudian terdapat *columnround function*, langkah ini memodifikasi setiap kolom di dalam *matrix*. Jika terdapat matrix x = (x0, x1, x2, x3, ... , x15) maka (Afdhila, dkk., 2016).

columnround(x) = (y0, y1, y2, y3, ... , y15) ...(2.11)

dimana :

(y0, y4, y8, y12) = quarterround(x0, x4, x8, x12)

(y5, y9, y13, y1) = quarterround(x5, x9, x13, x1)

(y10, y14, y2, y6) = quarterround(x10, x14, x2, x6)

(y15, y3, y7, y11) = quarterround(x15, x3, x7, x11)

Kemudian proses dilanjutkan dengan *doubleround* function yang merupakan fungsi untuk melakukan *looping* sebanyak yang dispesifikasikan. Terakhir ada fungsi *littleendian* yang memiliki operasi seperti berikut.

Jika b = (b0, b1, b2, b3) maka (Afdhila, dkk., 2016),

Littleendian(b) = $b0 + 2^8 b1 + 2^{16} b2 + 2^{24} b3$

2.7 Jitter

Jitter adalah variasi antara waktu yang seharusnya sampai dengan waktu ketika paket datang. Dalam RFC3550 *jitter* dapat dijelaskan dengan melakukan analisa pada persamaan berikut (Schulzrinne,2003) :

$$J_{(i)} = J_{(i-1)} + \frac{D_{(i-1,i)} - J_{(i1)}}{16} \quad ...(2.12)$$

Dimana :

$J_{(i)}$ adalah *mean jitter* dari paket ke *i*.

$J_{(i-1)}$ adalah *mean jitter* dari paket ke (*i - 1*).

$D_{(i-l,\ i)}$ adalah *delay* antara paket *(i)* dan *(i-l).*

Jitter juga merupakan variasi *delay* antar paket. *Jitter* dipengaruhi oleh variasi beban *traffic* dan besarnya tumbukan antar paket atau *congestion* yang ada pada jaringan. Semakin besar beban trafik maka semakin besar juga kemungkinan terjadinya *congestion* yang menyebabkan nilai *jitter* semakin besar (Fatoni, 2011)

2.8 ARP Spoofing

ARP spoofing adalah proses mengirim *ARP request* atau *ARP reply* untuk mengelabui komputer korban bahwa *hacker* merupakan *gateway router*. Kemudian *hacker* juga akan mengirim *ARP request* atau *ARP reply* untuk mengelabui *gateway router* bahwa komputer *hacker* adalah komputer korban. Teknik ini memanfaatkan kelemahan dari protokol ARP yang tidak memiliki mekanisme untuk melakukan cek identitas. Titik lemah lainnya adalah, *ARP table* pada setiap *host* akan berubah ketika menerima *ARP request* atau *ARP reply*. Isi pada *ARP table* akan bertahan selama 15 detik, oleh karena itu sebuah *program* yang digunakan untuk melakukan *ARP spoofing* harus mengirimkan paket ARP tanpa berhenti (Chomsiri, 2008).

2.9 Fast Exponential Modulus Arithmetic

Pada halaman *web* yang ditulis oleh Aptitude For Dummies, terdapat penjelasan untuk menghitung *exponential modulus arithmetic* secara cepat. Dalam *video* yang dibagikan pada halaman *web* youtube.com, jika terdapaat persamaan m^e mod n, maka nilai e akan dirubah ke dalam bentuk angka biner. Kemudian inisiasi bilangan d adalah 1. Dalam memulai perhitungan pertama pointer menunjuk pada deret biner e paling kiri. Jika pada pointer bilangan biner e menunjuk angka 1 maka perhitungan yang dilakukan adalah d^2 mod n terlebih dahulu, kemudian d*m

mod n. Tetapi jika menunjuk angka 0 maka perhitungan yang dilakukan hanya d^2 mod n. Kemudian jika pointer sudah menunjuk baris biner paling kanan dan menunjuk angka 1 maka hasil yang digunnakan adalah hasil perhitungan dari d*m mod n. Tetapi jika menunjuk angka 0 maka hasil yang dipakai adalah hasil perhitungan d*m mod n terakhir (aptitudefordummies, 2014).