Penggunaan Teori Bilangan Pada Algoritma RSA, Protokol Diffie-Hellman, dan Pencegahan Terhadap *Timing Attacks*

Tahir Arazi – NIM : 13505052

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15052@students.if.itb.ac.id

Abstrak

Teori bilangan ternyata dapat digunakan dalam pengembangan kriptografi. Makalah ini membahas mengenai penggunaan teori bilangan dalam implementasi kriptografi, yakni algoritma RSA, protokol Diffie-Hellman, dan pencegahan terhadap *Timing attalcs*. Algoritma RSA merupakan penerapan dari kriptografi asimetri, yakni jenis kriptografi yang menggunakan dua kunci yang berbeda : kunci public (*public key*) dan kunci pribadi (*private key*). Dengan demikian, maka terdapat satu kunci, yakni kunci publik, yang dapat dikirimkan melalui saluran yang bebas, tanpa adanya suatu kemanan tertentu. Hal ini bertolak belakang dengan kriptografi simetri yang hanya menggunakan satu jenis kunci dan kunci tersebut harus terus terjaga keamanan serta kerahasiaannya. Dalam kriptografi asimetri, dua kunci tersebut diatur sedemikian sehingga memiliki hubungan dalam suatu persamaan aritmatika modulo. Hubungan tersebut juga digunakan dalam protokol pertukaran kunci Diffie-Hellman. Protokol Diffie-Hellman membantu kerahasiaan dari pertukaran tersebut, sehingga jika terdapat seorang penyadap (*eavesdropper*), maka untuk mengetahui kunci yang ditukarkan harus melalui perhitungan yang sangat rumit dan panjang.

Makalah ini juga membahas aplikasi dari algoritma RSA, seperti dalam PGP (*Pretty Good Provacy*) dan protokol SSL (*Secure Socket Layer*). Fitur *digital signature* yang ditawarkan oleh program PGP dapat menggunakan algoritma RSA. Algortima RSA juga merupakan sutau pilihan dalam fase pertama, yakni negosiasi antara server dan klien, dalam protokol SSL.

Dalam dunia kriptografi terdapat dua profesi, yakni kirptografer dan kriptanalis (*cryptanalyst*). Seorang kriptografer akan berusaha untuk mencipatakan algoritma kriptografi sekompleks mungkin, sedangkan seorang kriptanalis akan berusaha untuk memecahkan algoritma tersebut agar bias mengetahui pesan yang dirahasiakan (mendekripsi dari *ciphertext* menjadi *plaintext*). Kriptanalis dapat berperan sebagai penguji dari suatu algoritma kriptografi atau menjadi penyerang utnuk mengetahui informasi rahasia yang sesungguhnya tidak berhak untuk ia ketahui. Suatu serangan dapat digolongkan sebagai serangan pasif (*passive attacks*), yakni serangan yang tidak merubah isi dari pesan, atau serangan aktif (*active attacks*), yang dapat mengubah isi dari suatu pesan.

Meskipun kriptografi asimetri menggunakan kunci publik yang dapat melewati saluran tanpa pengamanan, namun bukan berarti kriptografi ini tidak bebas dari sernagan (*attack*). Metode *timing attacks* memanfaatkan proses perhitungan suatu kunci, sehingga jika terdapat suatu kebocoran selama pemrosesan, maka informasi tersebut dapat dimanfaatkan untuk mengetahui kunci pribadi yang digunakan. Pencegahan terhadap *timing attacks* dapat dilakukan dengan cara memodifikasi persamaan aritmatika modulo yang digunakan, sehingga menambah kompleksitas dari perhitungan.

Kata kunci : RSA, Diffie-Hellman, *Pretty Good Privacy* (PGP), *Secure Socket Layer* (SSL), Kriptografi Asimetri (*asymmetric cryptography*), *Timing Attacks*.

1. Pendahuluan

Teori bilangan adalah cabang dari ilmu matematika yang mempelajari secara khusus sifat-sifat dari suatu bilangan, terutama integer-bilangan bulat yang dapat bernilai positif, negative, atau 0 (nol). Kriptografi menerapkan sifat-sifat serta operasi dari bilangan bulat positif pada aplikasinya. Bilangan bulat positif dapat dibagi menjadi dua macam, yakni bilangan komposit dan prima. Bilangan komposit adalah bilangan yang dapat habis dibagi dengan bilangan lain, selain 1 dan bilangan itu sendiri. Jika terdapat suatu bilangan yang hanya habis dibagi dengan 1 atau bilangan itu sendiri, maka bilangan tersebut merupakan bilangan prima.

Hal penting lainnya menyangkut Kriptografi adalah komputasi integer dengan Aritmatika Modulo. Operator yang digunakan pada aritmatika modulo adalah **mod** [1]. Operator mod memberikan sisa pembagian dari bilangan bulat. Misalkan *a* adalah bilangan bulat dan *m* adalah bilangan bulat > 0. Operasi *a* mod *m* memberikan sisa pembagian dari *a* dan *m*. Dapat dikatakan pula bahwa *a* mod *m* = *r*, sedemikian sehingga $a = mq + r$, dimana $0 \leq r < m$. Jika terdapat bilangan bulat *a* dan *b* sedemikian sehingga keduanya mempunyai sisa yang sama jika dibagi dengan bilangan bulat positif *m*, maka *a* dan *b* adalah kongruen dalam modulo m dan dilambangkan dengan $a \equiv b \pmod{m}$. Operator mod juga digunakan dalam persamaan berdasarkan teorema Fermat, yakni $a^{p-1} \equiv 1 \pmod{p}$, dengan *p* adalah bilangan prima dan *a* adalah bilangan bulat yang tidak habis dibagi dengan *p*. Aritmatika Modulo nantinya akan banyak dipergunakan pada pembahasan selanjutnya.

2. Kriptografi

Kriptografi adalah seni dan ilmu dalam menuliskan pesan rahasia, artinya suatu informasi diubah sedemikian sehingga menjadi tidak dapat dimengerti oleh orang yang tidak diinginkan untuk mengetahui informasi tersebut. Namun, perubahan informasi tersebut harus dapat dikembalikan seperti semula (*reversible*) agar dapat dibaca oleh orang yang berhak [3]. Penyandiaan pesan tersebut selanjutnya dinamakan *cipher* atau *cryptosystem*. *Chiper* adalah sepasang fungsi yang tidak dapat dibalik, yakni f_k (*enciphering function*) dan g_k (*deciphering function*). Fungsi f_k memetakan elemen *x* dalam himpunan *S* menjadi elemen $f_k(x)$ dalam himpunan *T*, sehingga mencari pemetaan balikan (*inverse*) menjadi sangat sulit tanpa mengetahui *k'*. Elemen dari *S* disebut sebagai *plaintext* dan elemen dari *T* disebut sebagai *ciphertext*. Fungsi g_k adalah balikan (*inverse*) dari f_k. *k'* yang disebut juga sebagai *deciphering key* (kunci dekripsi). Jika *k* = *k'*, atau *k'* sangatlah gampang untuk dihitung dengan memanfaatkan nilai *k*, maka kriptografi yang dipakai disebut dengan *symmetric cryptography* (kriptografi simetri) dan kunci dari kriptografi ini disebut *secret key* (kunci rahasia). Namun, jika *k'* sangat sulit untuk diketahui, walaupun dengan mengetahui *k*, maka kriptografi yang dipakai disebut dengan *asymmetric cryptography* (kriptografi asimetri) dan *k* disebut sebagai *public key* (kunci publik) dan *k'* disebut dengan *private key* (kunci pribadi).

2.1 Kriptografi Simetri

Kriptografi simetri adalah metode enskripsi dimana pengirim dan penerima pesan memiliki kunci yang sama, atau dalam beberapa kasus kedua kunci berbeda namun mempunya relasi dengan perhitungan yang mudah. Studi modern terfokuskan pada *block cipher* dan *stream cipher* serta aplikasinya. *Block cipher* adalah aplikasi modern dari *Alberti's polyphabetic cipher*. *Block cipher* menerima masukan berupa blok *plaintext* dan sebuah kunci dan kemudian menghasilkan keluaran blok *ciphertext* dengan ukuran yang sama. Dikarenakan pesan yang dikirim hampir selalu lebih panjang dari *single block* (blok tunggal), maka diperlukan metode penggabungan beberapa blok.

Data Encryption Standard (DES) dan *Advanced Encryption Standard* (AES) adalah contoh *block ciphers* yang dijadikan standar kriptografi oleh pemerintahan Amerika Serikat. Walaupun AES telah diresmikan sebagai standar kriptografi terbaru, namun DES, khususnya varian *triple-DES*, masih banyak digunakan sebagai enkripsi ATM, keamanan surat elektronik (*e-mail*), dan *secure remote access*.

Stream cipher adalah lawan dari *block cipher*, yakni menciptakan arus kunci yang panjang dan sembarang (*arbitrarily long stream of key*) yang dikombinasikan dengan *plaintext* bit-per-bit (*bit-*

by-bit) dan karakter-per-karakter (*character-by-character*). Pada *stream cipher*, arus keluaran dibangkitkan berdasarkan keadaan internal (*internal state*) yang berubah-ubah seiring dengan jalannya *cipher*. Perubahaan tersebut diatur oleh kunci dan dibeberapa *stream cipher* diatur pula oleh *plaintext cipher*. *RC4* adalah contoh dari *stream cipher*.

2.2 Kriptografi Asimetri

Kriptografi simetri menggunakan kunci yang sama untuk enkripsi dan dekripsi [6]. Hal tersebut menyebabkan masalah yang signifikan, yakni kunci harus dikelola dengan sangat aman. Idealnya setiap kelompok yang terlibat komunikasi memiliki kunci yang berbeda. Kebutuhan akan variasi kunci meningkat, seiring dengan pertumbuhan jaringan, sehingga membutuhkan manajemen kunci yang kompleks untuk menjaga kerahasiaan tiap kunci. Masalah juga bertambah jika antara dua kelompok yang berkomunikasi tidak terdapat saluran aman (*secure channel*).

Pada tahun 1976, Whitfield Diffie dan Martin Hellman mengajukan konsep kriptografi asimetri yang meggunakan dua kunci yang secara matematika berhubungan satu sama lain, yakni kunci publik (*public key*) dan kunci pribadi (*private key*). Kunci publik dibangkitkan sedemikian sehingga kunci pribadi sangat sulit untuk dihitung, walaupun keduanya sesungguhnya berhubungan satu sama lain.

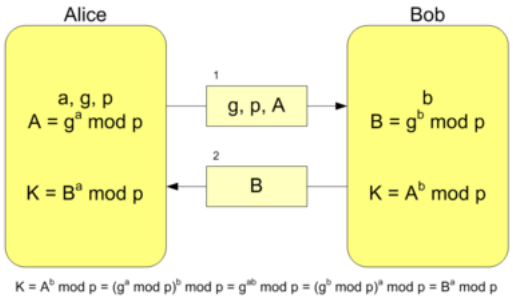
Dalam kriptografi asimetri, kunci publik dapat secara bebas disebarluaskan, sedangkan kunci pribadi harus senantiasa dijaga kerahasiaannya. Kunci publik digunakan untuk enkripsi, sedangkan kunci pribadi digunakan untuk dekripsi. Diffie dan Hellman membuktikan bahwa kriptografi asimetri adalah mungkin dengan menerapkan protokol pertukaran kunci Diffie-Hellman. Protokol ini akan dibahas lebih dalam pada pembahasan selanjutnya. Pada tahun 1978, Ronald Rivest, Adi Shamir, dan Len Adleman menemukan RSA, sebuah algoritma berdasarkan kriptografi asimetri. RSA juga akan dibahas secara mendalam pada bagian selanjutnya.

2.2.1 Protokol Diffie-Hellman

Protokol Diffie-Hellman memungkinkan dua pengguna untuk menukarkan kunci mereka melalui media tanpa keamanan dan kerahasiaan tertentu [7]. Protokol ini memiliki dua parameter, yakni *p* dan *g*. Keduanya publik dan dapat digunakan oleh semua pengguna yang berada dalam sistem. Parameter *p* adalah bilangan prima dan parameter *g*, biasanya disebut sebagai *generator* (pembangkit), adalah sebuah bilangan integer yang lebih kecil dari *p*, dengan sifat sebagai berikut : untuk setiap *n* dimana $1 \leq n \leq p\text{-}1$, terdapat pangkat *k* dari *g* sedemikian sehingga $n = g^k \bmod p$.

Untuk lebih memahami protokol ini, kita misalkan terdapat dua user Alice dan Bob. Runtunan komunikasi Alice dan Bob dijabarkan sebagai berikut :

1. Alice dan Bob sepakat untuk menggunakan bilangan prima *p*=23 dan integer *g*=5.
2. Alice memilih bilangan integer rahasia *a*=6, kemudian mengirim kepada Bob ($g^a \bmod p$).
   - $5^6 \bmod 23 = 8$
3. Bob memilih bilangan integer rahasia *b*=15, kemudian mengirimkan kepada Alice ($g^b \bmod p$).
   - $5^{15} \bmod 23 = 19$
4. Alice menghitung $(g^b \bmod p)^a \bmod p$.
   - $19^6 \bmod 23 = 2$
5. Bob menghitung $(g^a \bmod p)^b \bmod p$.
   - $8^{15} \bmod 23 = 2$



Gambar 1 Diffie-Hellman *Key Exchange*

Pada akhirnya Bob dan Alice akan memperoleh nilai yang sama, karena g^{ab} dan g^{ba} adalah

sama. Sesungguhnya hanya *a*, *b*, dan $g^{ab} = g^{ba}$ yang dijaga kerahasiaannya. Nilai yang lainnya dikirimkan secara terbuka. Saat Bob dan Alice menghitung kerahasiaan yang akan dipakai bersama-sama (*shared secret*), maka mereka dapat mempergunakan nilai tersebut sebagai kunci enkripsi, yang hanya diketahui oleh mereka, untuk mengirimkan pesan dalam saluran yang terbuka. Tentu saja, nilai *a*, *b*, dan *p* yang lebih besar akan lebih baik untuk menjaga kerahasiaan, karena akan mudah untuk mencoba semua kemungkinan *a* dan *b* yang memenuhi persamaan $g^{ab} \bmod 23$ (akan ada paling banyak 22 kemungkinan nilai, meskipun *a* dan *b* merupakan nilai yang besar). Jika *p* adalah bilangan prima yang panjangnya paling sedikit 300 digit, *a* dan *b* memiliki panjang sedikitnya 100 digit, maka algoritma terbaik saat ini pun tak akan mampu untuk mengetahui nilai *a* dengan memiliki informasi mengenai nilai *g*, *p*, dan $g^a \bmod p$, bahkan dengan menggunakan kemampuan menghitung semua manusia. Perlu diketahui pula bahwa sesungguhnya nilai *g* tidak perlu terlalu besar dan pada prakteknya nilai tersebut diantara 2 atau 5.

Berikut adalah tabel yang menunjukkan nilai yang diketahui ole Alice dan Bob, serta Eve sebagai *eavesdropper* (orang yang memperhatikan komunikasi antara Alice dan Bob). *Eavesdropper* akan dibahas pada bagian selanjutnya.

Alice	
knows	doesn't know
p = 23	b = 15
base g = 5	
a = 6	
$5^6 \bmod 23 = 8$	
$5^b \bmod 23 = 19$	
$19^6 \bmod 23 = 2$	
$8^b \bmod 23 = 2$	
$19^6 \bmod 23 = 8^b \bmod 23$	
s = 2	

Bob	
knows	doesn't know
p = 23	a = 6
base g = 5	
b = 15	
$5^{15} \bmod 23 = 19$	
$5^a \bmod 23 = 8$	
$8^{15} \bmod 23 = 2$	
$19^a \bmod 23 = 2$	
$8^{15} \bmod 23 = 19^a \bmod 23$	
s = 2	

Eve	
knows	doesn't know
p = 23	a = 6
base g = 5	b = 15
	s = 2
$5^a \bmod 23 = 8$	
$5^b \bmod 23 = 19$	
$19^a \bmod 23 = s$	
$8^b \bmod 23 = s$	
$19^a \bmod 23 = 8^b \bmod 23$	

Akan menjadi sulit untuk Alice menghitung kunci pribadi milik Bob, begitu pula sebaliknya. Jika perhitungan tersebut menjadi mudah, maka Eve dengan mudah dapat mensubstitusi kunci pribadi miliknya sendiri, memasukkan kunci publik milik Bob kedalam kunci pribadinya, menciptakan kunci rahasia palsu yang dipakai bersama (*fake shared secret key*), dan menyelesaikan perhitungan untuk mendapatkan kunci pribadi Bob.

2.2.2 Algoritma RSA

Algoritma RSA meliputi kunci publik dan kunci pribadi [5]. Kunci publik dapat diketahui oleh setiap orang dan digunakan untuk enkripsi pesan. Pesan yang telah dienkripsi hanya dapat di dekripsi dengan menggunakan kunci pribadi. Algoritma dari RSA adalah sebagai berikut :

1. Pilih dua buah bilangan prima sembarang *p* dan *q*
2. Hitunglah $n = pq$
   - *n* digunakan sebagai modulus untuk kunci publik dan kunci pribadi
3. Hitunglah nilai *totient* : $\phi(n) = (p-1)(q-1)$
   Catatan : persamaan ini diubah pada PKCS#1 v2.0 menjadi : $\lambda(n) = lcm(p-1, q-1)$
4. Pilih integer *e* sedemikian sehingga 1< *e* < $\phi(n)$ dan *e* realtif prima terhadap $\phi(n)$
   - *e* dirilis menjadi eksponen kunci publik
5. Hitunglah *d* yang memenuhi relasi kongruen $de \equiv 1 \ (\bmod \phi(n))$
   - *d* disimpan sebagai eksponen kunci pribadi

Kunci publik terdiri dari modulus *n* dan eksponen publik *e*. Kunci pribadi terdiri dari modulus *n* dan eksponen pribadi *d* yang harus dijaga kerahasiaannya. Bagian dari *n* yakni *p* dan *q*, juga harus dijaga kerahasiaannya karena kedua bilangan tersebut merupakan factor-faktor dari *n*. Untuk efisiensi sesungguhnya ada beberapa variasi lain yang dapat digunakan sebagai kunci pribadi, yakni :

- $d \bmod(p-1)$ dan $d \bmod(q-1)$ yang disebut sebagai *dmpl* dan *dmql*
- $q^{-1} \bmod(p)$ yang disebut sebagai *iqmp*

Untuk lebih jelasnya lagi, selanjutnya akan dijabarkan proses enkripsi dan dekripsi pesan yang dikirimkan seorang user Bob kepada Alice.

Enkripsi Pesan

1. Alice mengirimkan kunci publik *n* dan *e* kepada Bob dan menjaga kerahasiaan kunci pribadi miliknya. Bob kemudian bermaksud untuk mengirimkan pesan **M** kepada Alice.
2. Pertama-tama, Bob mengubah pesan **M** menjadi bilangan *m* < *n* dengan protokol yang telah disetujui, yakni *padding scheme*.
3. Bob kemudian menghitung *ciphertext c* dengan persamaan :
   - $c = m^e \bmod n$
4. Setelah mendapatkan nilai *c*, Bob mengirimkannya kepada Alice.

Dekripsi Pesan

1. Alice dapat menghitung nilai *m* dari *c* dengan menggunakan kunci pribadi miliknya, yakni *d*, dengan persamaan berikut :
   - $m = c^d \bmod n$
2. Dengan mengetahui *m*, Alice dapat mengetahui pesan **M**.
3. Persamaan $m = c^d \bmod n$ didapatkan dari langkah-langkah berikut :
   - $ed \equiv 1(\bmod p-1)$ dan $ed \equiv 1(\bmod q-1)$
   - Dengan menggunakan Teorema kecil Fermat (*Fermat's Little Theorem*) menghasilkan :
     $m^{ed} \equiv m(\bmod p)$ dan $m^{ed} \equiv m(\bmod q)$
   - Karena *p* dan *q* adalah dua bilangan prima yang berbeda, maka dengan menerapkan *Chinese Remainder Theorem* (CRT), dua kongruen diatas menghasilkan :
     $m^{ed} \equiv m(\bmod pq)$
     Yang ekivalen dengan
     $c^d \equiv m(\bmod n)$

Berikut adalah contoh perhitungan menggunakan Algoritma RSA.

1. Pilih dua bilangan prima
   - *p* = 61 dan *q* = 53
2. Hitung *n* = *pq*
   - *n* = 61 * 53 = 3233
3. Hitung nilai *totient* $\phi(n) = (p-1)(q-1)$
   - $\phi(n) = (61-1)(53-1) = 3120$
4. Pilih *e* > 1 yang relatif prima dengan 3120
   - *e* = 17
5. Pilih *d* yang memenuhi $de \equiv 1(\bmod \phi(n))$
   - *d* = 2753
   - 17 * 2753 = 46801 = 1 + 15 * 3120

Untuk mengenkripsi pesan **M** dengan *m* = 123, kita hitung :

$$c = 123^{17} \bmod 3233 = 855$$

Untuk mendekripsi *c* = 855, kita hitung :

$$m = 855^{2753} \bmod 3233 = 123$$

2.2.2.1 Aplikasi PGP

PGP (Pretty God Privacy) adalah program komputer yang menjalankan privasi dan autentifikasi [8]. PGP umunya digunakan untuk mengenkripsi pesan surat elektronik (*e-mail*) dan *attachments*, *digital signatures*, keamanan *file* dan *folder*, enkripsi pengiririman *file*,dan perlindungan untuk *files* dan *folders* dalam server jaringan.

Enkripsi PGP menggunakan kriptografi asimetri dan menjadikan kunci publik sebagai identitas pengguna (*user identities*). Versi pertama dari PGP memperkenalkan sistem yang disebut dengan *web of trust* yang kemudian sejak versi x.509 menggunakan pendekatan hirarki berdasarkan sertifikat otoritas (*certificate authority*). Versi terkini dari PGP meliputi kedua sistem diatas yang diatur dengan manajemen server yang terotomasi. Dikarenakan PGP menggunakan kriptografi asimetri, maka terdapat dua kunci yang digunakan, yakni kunci publik dan kunci pribadi. Kunci publik digunakan untuk enkripsi pesan yang akan dikirim (*plaintext*). Kunci publik didapatkan dari server PGP oleh para penggunanya. Penerima pesan yang telah dienkripsi akan menggunakan kunci pribadinya untuk medekripsi pesan yang diterima.

PGP juga dapat memeriksa apakah suatu pesan yang diterima masih dalam kondisi semula atau telah mengalami perubahan selama pengiriman. PGP juga memiliki kemampuan untuk mengetahui apakah pesan yang diterima adalah

pesan yang berasal dari pengirim yang dikehendaki. Sistem yang digunakan untuk melakukan hal tersebut disebut dengan *digital signature*. *Digital signature* yang dibuat dapat menggunakan algoritma RSA atau DSA. Dalam pembuatan *digital signature*, PGP menghitung *hash* dari *plaintext*, kemudian memanfaatkan *hash* tersebut serta kunci pribadi milik pengirim untuk membuat *digital signature*.

Penerima pesan menggunakan kunci publik pengirim dan *digital signature* untuk memperoleh *message digest*. Kemudian penerima membandingkan *message digest* yang diperoleh dari perhitungan *plaintext* (hasil dekripsi *ciphertext*) dengan *message digest* dari *digital signature*. Jika ternyata cocok, maka dapat dipastikan bahwa pesan yang dikirim tidak mengalami perubahan.

Selain *digital signatures*, pada versi OpenPGP terdapat pula *trust signatures* yang mendukung pembuatan sertifikasi otoritas. *Trust signature* mengindikasikan dua keadaan, yakni kunci dimiliki oleh pemilik yang sesungguhnya dan pemilik kunci tersebut berhak untuk membuat kunci lain satu tingkat dibawahnya. *Level 0 signature* sama dengan *web of trust signature* yang berarti hanya kunci yang mendapat sertifikat dan tidak berhak untuk membuat kunci lain. *Level 1 signature* memiliki sertifikat untuk kunci dan berhak untuk membuat kunci pada tingkat 0. *Level 2 signature* tidak hany memiliki sertifikat untuk kunci, namun juga berhak untuk mengeluarkan sertifikasi otoritas.

2.2.2.2 Protokol SSL

Secure Socket Layer (SSL) adalah protokol kriptografi yang memungkinkan komunikasi yang aman di internet seperti saat menggunakan *web browser*, surat elektronik (*e-mail*), *internet faxing*, dan pengiriman data lainnya [9]. Protokol SSL memfalisitasi aplikasi server untuk berkomunikasi yang aman dari *eavesdropping* (akan dijelaskan pada bagian selanjutnya), *tampering* (gangguan), dan *message forgery* (pemalsuan pesan). SSL menggunakan otentifikasi dan privasi komunikasi dalam internet menggunakan kriptografi. Pada umumnya, otentifikasi hanya dilakukan pada server, sedangkan klien yang menggunakan *browser* tidak terotentifikasi. Hal ini berarti, pemakai browser dapat yakin kepada siapa ia 'berbicara'. Pada tingkatan keamanan berikutnya, otentifikasi dilakukan tidak hanya pada server, namun juga pada klien, yang dinamakan dengan otentifikasi mutual (*mutual authentification*). Otentifikasi mutual membutuhkan infrastruktur kunci publik (PKI). SSL terdiri dari tiga fase, yakni :

1. Negosiasi untuk algoritma yang mendukung
2. Enkripsi kunci publik, pertukaran kunci dan otentifikasi yang bersertifikat
3. *Cipher* simetri, enkripsi lalu lintas

Pada fase pertama server dan klien dapat memilih beberapa algrotima kriptografi, seperti :

- Kriptografi asimetri : RSA, Diffie-Hellman, atau Fortezza
- *Cipher* simetri : RC2, RC4, IDEA, DES, Triple DES, AES, atau Camellia
- Fungsi *one-way hash* : MD2, MD4, MD5, atau SHA

Cara kerja SSL pada dasarnya adalah pertukaran rekaman (*records*). Setiap rekaman dapat dimapatkan (*compressed*), terenkripsi, dan terpaketkan dengan kode pesan otentifikasi (*message authentication code* (MAC)). Ketika komunikasi mulai, rekaman mengeluarkan protokol lainnya, yakni *handshake protocol* yang memiliki *content_type*22.

Handshake protocol memiliki struktur sebagai berikut :

- Klien mengirimkan pesan *ClientHello* yang berisi spesifikasi *cipher* yang digunakan, metode kompresi, dan versi protokol yang mendukung. Klien juga mengirimkan sembarang (*random*) *bytes* yang akan digunakan kemudian.
- Kemudian klien menerima *ServerHello* dimana server memilih parameter koneksi dari pilihan yang diajukan oleh klien.
- Ketika parameter koneksi sudah diketahui, klien dan server bertukar sertifikat, tergantung *cipher* kunci publik yang dipilih. Beberapa sertifikat yang dipakai merupakan *OpenPGP based certificates*.
- Server dapat meminta sertifikat dari klien, sehingga koneksi tersebut dapat terotentifikasi secara mutual.
- Klien dan server bernegosiasi secara rahasia yang biasa disebut dengan '*master secret*', dapat menggunakan

protokol Diffie-Hellman atau hanya dengan mengenkripsi menggunakan kunci publik yang telah didekripsi menggunakan kunci pribadi. Data kunci lainnya melewati '*pseudorandom function*'.

SSL mempunyai ukuran keamanan yang bervariasi :

- Menomorkan semua rekaman dan menggunakan rutunan angka dalam MAC
- Menggunakan *message digest* yang dilengkapi dengan kunci, sehingga hanya dengan kunci tersebut dapat mengetahui MAC
- Proteksi terhadap beberapa serangan kriptografi, seperti *man in the middle attack* (akan dijelaskan pada bagian selanjutnya).
- Fungsi *pseudorandom* memecah masukan data menjadi dua dan memproses tiap bagian tersebut dengan algoritma *hash* yang berbeda (MD5 dan SHA), kemudian meng-XOR-kan (*exclusive or*) keduanya.

3. Serangan Terhadap Kriptografi

Serangan terhadap kriptografi berarti usaha untuk menemukan kunci atau *plaintext* dari *ciphertext*. Untuk lebih jelasnya, pembahasan mengenai serangan terhadap kriptografi dimulai dari penjelasan tentang kriptanalisis (*cryptanalysis*).

3.1 Kriptanalisis

Tujuan dari kriptanalisis adalah untuk mengetahui kelemahan dari sebuah skema kriptografi. Kriptanalisis dapat diterapkan oleh seseorang yang bermaksud mengetahui informasi rahasia atau bias juga diterapkan oleh perancang suatu sistem keamanan.

Terdapat beberapa variasi dalam serangan kriptanalisis dan dapat diklasifikasikan dalam beberapa cara. Dalam serangan *ciphertext-only*, kriptanalisis mempunyai akses hanya kepada *ciphertext* (kriptosistem modern yang bagus pada umumnya kebal terhadap serangan ini). Pada serangan *known-plaintext*, kriptanalis mempunyai akses kepada *ciphertext* dan *plaintext* yang berkorespondensi. Pada serangan *chosen-plaintext*, kriptanalis memilih sebuah *plaintext* dan mempelajari *ciphertext* yang berkorespondensi. Terakhir, pada serangan *chosen-ciphertext*, kriptanalis memilih sebuah *ciphertext* dan mempelajari *plaintext* yang berkorespondensi.

Kriptanalisis dari *cipher* kunci simetri pada umumnya melibatkan serangan terhadap *block cipher* atau *stream cipher* yang lebih mangkus dari serangan manapun terhadap *cipher* sempurna (*the perfect cipher*). Sebagai contoh, sebuah serangan *brute force* terhadap DES memerlukan satu buah *plaintext* yang diketahui dan 2^{55} dekripsi, mencoba hamper kira-kira setengah dari kunci yang mungkin. Namun, ini bukanlah suatu kepastian, karena sebuah serangan kriptanalisis lanjar (*linear cryptanalysis attacks*) terhadap DES memerlukan 2^{43} *plaintext* yang diketahui dan 2^{43} operasi DES.

Kriptografi asimetri (atau biasa disebut algoritma kunci publik) pada dasarnya memiliki kekuatan pada kesulitan dalam perhitungan variasi masalah. Masalah yang paling terkenal adalah masalah faktorisasi integer (pada RSA) dan masalah logaritma diskret (*discrete logarithm problems*). Kebanyakan kripanalisis kunci publik memusatkan pada algoritma numeric untuk menyelesaikan masalah perhitungan. Sebagai contoh, algoritma terbaik saat ini untuk menyelesaikan logaritma diskret *elliptic curve-based* memakan waktu yang lebih banyak daripada algoritma terbaik untuk faktorisasi, setidaknya dengan bobot masalah yang sama. Maka dari itu, untuk memperoleh kekuatan dari serangan yang sama, teknik enkripsi dengan faktorisasi harus menggunakan kunci yang lebih besar dibandingkan teknik dengan kurva eliptis.

3.2 Jenis-jenis Serangan

Suatu serangan dapat digolongkan menjadi dua jenis serangan berdasarkan keterlibatan penyerang (kriptanalis) pada komunikasi, yakni serangan pasif dan serangan aktif [4].

Serangan pasif berarti seorang penyerang tidak berinteraksi dengan kelompok yang sedang berkomunikasi dan hanya memanfaatkan data yang diperoleh (seperti *ciphertext*). Ciri-ciri dari serangan pasif adalah penyerang tidak terlibat dalam komunikasi antara pengirim dan penerima, serta penyerang hanya melakukan penyadapan (*eavesdropping*) untuk memperoleh

data sebanyak-banyaknya. Penyadapan dapat berbentuk *wiretapping, electromagnetic eavesdropping*, dan *acoustic eavesdropping*.

Serangan aktif berarti penyerang tidak hanya mengintervensi komunikasi, tetapi juga ikut mempengaruhi sistem. Penyerang dapat saja menghapus sebagian *ciphertext*, mengubah *ciphertext*, menyisipkan potongan *ciphertext* palsu, membalas pesan lama, atau mengubah informasi yang tersimpan. Salah satu contoh serangan aktif adalah *man-in-the-middle attack* (MITM).

MITM adalah serangan dimana penyerang mampu untuk membaca, menambah dan mengubah sesuai kehendak pesan antara dua kelompok tanpa sepengetahuan kelompok tersebut. MITM biasanya dilakukan terhadap kriptografi asimetri dan bias juga pada protokol Diffi-Hellman.

Untuk lebih jelasnya kita misalkan Alice dan Bob sebagai dua orang yang akan berkomunikasi, dan Mallory berkehendak untuk menyadap komunikasi tersebut serta mengirimkan pesan yang salah kepada Bob. Untuk memulai komunikasi, pertama-tama Alice meminta Bob untuk mengirimkan kunci publik miliknya. Ketika Mallory dapat menyadap kunci publik yang dikirim, maka MITM akan dapat segera dimulai. Mallory dapat dengan mudah mengirimkan Alice kunci publik yang cocok dengan kunci pribadi milik Alice. Alice yang mempercayai bahwa kunci publik yang ia terima berasal dari Bob, mulai mengenkripsi kunci yang sebenarnya dikirm oleh Mallory dan kemudian mengirimkan kembali ke Bob pesan yang telah dienkripsi. Mallory lalu kembali menyadap komunikasi dan mendekripsi pesan tersebut, menyimpan salinan pesan, kemudian mengenkripsi pesan yang telah diubah menggunakan kunci publik yang dikirimkan oleh Bob dan kemudian mengirimkannya kepada Bob. Ketika Bob menerima *ciphertext*, maka ia akan percaya bahwa pesan tersebut berasal dari Alice.

Untuk menjaga keamanan dari serangan MITM maka dapat menggunakan beberapa teknik sebagai berikut :

- Menggunakan otentifikasi mutual
- Menggunakan sandi lewat (*password*)
- Menggunakan pengenal suara (*voice recognition*)

3.3 *Timing Attacks*

Sistem kripto sering kali menghabisakan periode waktu yang berbeda dalam mengolah setiap masukan yang berbeda. Hal tersebut antara lain disebabkan oleh optimasi kemampuan untuk mengenyampingkan beberapa operasi yang tidak perlu, kecepatan *cache*, instruksi *processor* dalam waktu yang tak tentu, dan masalah-masalah lainnya [2]. Karakteristik kemampuan bergantung pada kunci enkripsi yang dipakai dan data masukan (*plaintext* dan *ciphertext*). Data atau kunci dapat saja sewaktu-waktu bocor dari saluran waktu (*time channel*) pada saat proses berlangsung, namun pada umumnya data yang bocor hanya berisi sebagian kecil informasi dari sistem kripto, seperti bobot *Hamming* dari sebuah kunci. *Timing attacks* dapat memanfaatkan kebocoran tersebut sehingga bias mendapatkan keseluruhan informasi tentang kunci pribadi.

3.3.1 Kriptanalisis dari *Modular Exponentiator* Sederhana

Operasi kunci publik untuk algoritma RSA dan protokol Diffie-Hellman mengandung unsur perhitungan $R = y^x \bmod n$ dimana *n* publik dan *y* dapat ditemukan oleh penyadap (*eavesdropper*). Tujuan dari penyerang adalah untuk mengetahui kunci rahasia/pribadi *x*. Untuk keperluan penyerang, korban harus melakukan perhitungan terhadap $y^x \bmod n$ untuk beberapa nilai dari *y*, dimana *y*, *n*, dan waktu penghitungan diketahui oleh penyerang. Informasi tersebut sessungguhnya dapat diperoleh dengan menyadap sebuah protokol interaktif. Diasumsikan bahwa penyerang mengetahui desain dari sistem yang menjadi sasaran.

Algoritma pertama yang digunakan penyerang untuk menghitung $R = y^x \bmod n$ dimana *x* memilki panjang *w* bit, adalah sebagai berikut :

```
Let s_0 = 1.
For k = 0 upto w−1:
  If (bit k of x) is 1 then
      Let R_k = (s_k . y) mod n.
  Else
      Let R_k = s_k .
  Let s_{k+1} = R_k^2 mod n.
```

EndFor.
Return (R_{w-1}).

Penyerang membiarkan seseorang yang mengetahui bit eksponen 0..(*b*-1) untuk menemukan *b*. Untuk mengetahui keseluruhan eksponen, mulai dengan *b*=0 dan ulangi penyerangan hingga keseluruhan eksponen diketahui.

Dikarenakan bit *b* pertama diketahui, penyerang dapat menghitung iterasi *b* yang pertama dari kalang (*loop*) For untuk menemukan nilai dari s_b. Iterasi selanjutnya memerlukan bit pertama yang tidak diketahui. Jika bit ini adalah sebuah himpunan, maka $R_b = (s_b.y) \bmod n$ akan dihitung. Jika bernilai 0 (nol), maka langkah tersebut akan dilewati.

Jika sasaran menggunakan fungsi muliplikasi modular (*modular multiplication*), maka prosesnya akan lebih cepat dibandingkan menggunakan eksponensiasi modular (*modular exponentiation*). Untuk beberapa nilai s_b dan *y*, perhitungan $R_b = (s_b.y) \bmod n$ akan sangat lambat dan dengan mengetahui desain yang dipakai oleh sasaran, penyerang akan segera mengetahui nilai-nilai tersebut. Jika jumlah dari waktu proses eksponensiasi modular adalah sangat cepat dan $R_b = (s_b.y) \bmod n$ lambat, maka nilai *b* dapat dipastikan adalah 0 (nol). Jika operasi $R_b = (s_b.y) \bmod n$ lambat begitu pula dengan proses eksponensiasinya, maka mungkin saja bit eksponennya merupakan suatu himpunan. Ketika penyerang mengetahui bit eksponen *b*, maka dapat disimpulkan bahwa keseluruhan waktu operasi adalah lambat kapanpun nilai dari $s_{b+1} = R_b^2 \bmod n$. Pengukuran waktu ini juga dapat digunakan kembali untuk mendapatkan nilai bit eksponen selanjutnya.

3.3.2 Serangan Pada Umumnya

Serangan sesungguhnya dapat dianggap sebagai sinyal deteksi masalah. ‘Sinyal’ terdiri dari variasi pengaturan waktu berkaitan dengan bit eksponen dan *noise* dari ketidaktepatan pengukuran dan variasi pengaturan waktu berkaitan dengan bit eksponen yang tidak diketahui. Persamaan dari *Timing Attack* adalah sebagia berikut :

- Diberikan *j* buah pesan yakni : $y_0, y_1, ..., y_{j-1}$ dengan pengukuran pengaturan waktu $T_0, T_1, ..., T_{j-1}$ peluang bahwa perkiraan nilai x_b untuk bit eksponen *b* pertama adalah benar, proposional dengan :

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

  dimana $t(y_i, x_b)$ adalah waku yang dibutuhkan untuk iterasi mencari *b* yang pertama dari perhitungan $y_i^x \bmod n$ menggunakan bit eksponen x_b dan F adalah peluang fungsi distribusi $T - t(y, x_b)$ terhadap nilai *y* dan x_b yang benar. Dikarenakan definisi dari *F* adalah peluang distribusi dari $T_i - t(y_i, x_b)$ jika x_b adalah benar, maka *F* merupakan fungsi terbaik untuk memprediksi $T_i - t(y_i, x_b)$. Perlu diketahui bahwa pengukuran pengaturan waktu dan nilai intermediate *s* dapat digunakan untuk estimasi nilai *F* yang lebih tepat.

- Diberikan prediksi yang tepat untuk x_{b-1}, terdapat dua kemungkinan nilai dari x_b. Peluang bahwa x_b adalah benar dan $x_b^{'}$ adalah tidak benar, dapat dicari dengan persamaan berikut :

$$\frac{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))}{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b)) + \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b^{'}))}$$

Dalam praktiknya, sesungguhnya persamaan diatas tidak terlalu berguna karena mencari nilai *F* akan memerlukan usaha yang sangat besar. Namun, penyederhanaan dari persamaan tersebut tidak akan dibahas, karena jauh menyimpang dari teori bilangan.

3.3.3 Mencegah *Timing Attacks*

Unutngnya terdapat suatu teknik untuk mencegah terjadinya *timing attalcs*. Antara lain kita bisa mengadapatasi teknik *blinding signatures* untuk mencegah penyerang mengetahui masukan dalam fungsi eksponensiasi modular. Caranya adalah sebelum menghitung fungsi tersebut, pilih sembarang pasang (v_i, v_f) sedemikian sehingga $v_f^{-1} = v_i^{\,x} \bmod n$. Untuk protokol Diffie-Hellman, akan lebih mudah untuk memilih sembarang v_i daripada menghitung $v_f = v_i^{-1} \bmod n$. Untuk RSA akan lebih cepat untuk memilih sembarang v_f yang relative prima terhadap *n* daripada menghitung $v_i = (v_f^{-1})^e \bmod n$, dimana *e* adalah eksponen publik. Sebelum operasi eksponensiasi modular, pesan masukan harus dikalikan terlebih dahulu dengan $v_i \pmod n$ dan setelah itu hasilnya akan dikoresksi dengan mengalikan dengan $v_f \pmod n$. Sistem akan menolak pesan yang sama dengan 0 (mod *n*).

Menghitung balikan dari mod *n* akan lambat, mak menjadi tidak praktis untuk membangkitkan pasangan sembarang baru (v_i, v_f) untuk setiap eksponensiasi baru. Perhitungan $v_f = (v_i^{-1})^x \bmod n$ itu sendiri dapat saja dimanfaatkan oleh *timing attalcs*. Namun, pasangan (v_i, v_f) sebaiknya tidak digunakan kembali, dikarenakan dapat saja disadap pada saat *timing attalcs* dan menjadikan nilai eksponen yang rahasia rentan untuk diketahui. Solusi yang mangkus untuk menyelesaikan masalah ini adalah memperbaharui nilai v_f dan v_i sebelum langkah eksponensiasi berikutnya dengan perhitungan $v_i^{'} = v_i^2$ dan $v_f^{'} = v_f^2$.

Jika (v_i, v_f) dijaga kerahasiaannya, maka penyerang tidak akan memiliki informasi yang berguna tentang masukan dalam eksponensiasi modular. Akibatnya kebanyakan penyerang akan mempelajari pengaturan distribusi operasi eksponensiasi secara umum. Dalam prakteknya, distribusi mendekati normal dan 2^w eksponen tidak akan dapat dibedakan. Namun, desain eksponensiator yang buruk secara teoritis akan memiliki distribusi yang hamper tepat berkaitan dengan bit-bit eksponen, sehingga teknik *blinding* tidak akan 100% dapat mencegah *timing attalcs*.

Walaupun dengan *blinding*, distribusi akan memperlihatkan waktu operasi rata-rata, yang mana dapat digunakan untuk mengetahui bobot *Hamming* dari eksponen. Teknik ini dapat membantu dalam mencegah serangan yang memanfaatkan informasi yang bocor selama eksponensiasi modular dikarenakan radiasi elektromagnetik, fluktuasi performa sistem, perubahan dalam konsumsi tenaga (*power consumption*), dan lain-lain, mengingat bit-bit eksponen berubah pada setiap operasi.

4. Kesimpulan

Dari berbagai penjelasan mengenai penggunaan teori bilangan pada RSA, Diffie-Helman, dan *timing attalcs*, dapat disimpulkan bahwa :

1. Teori bilangan dapat dipergunakan untuk membentuk suatu algoritma kriptografi dengan memanfaatkan aritmatika modulo.
2. Aritmatika modulo yang digunakan dalam protokol Diffie-Hellman memungkinkan komunikasi antara dua kelompok aman dari penyadapan oleh *eavesdropper*.
3. Kriptografi asimeteri dengan algoritma RSA memungkinkan sebuah kunci publik untuk bebas diketahui oleh pihak manapun. Hal ini merupakan suatu keuntungan lebih dibandingkan dengan kriptografi simetri yang senantiasa harus menjaga kerahasiaan dari sebuah kunci.
4. SSL dan PGP merupakan contoh dari implementasi kriptografi asimetri, yang dapat menggunakan algoritma RSA dalam prosesnya.
5. Meskipun memakai dua kunci yang berbeda, tetap saja algoritma RSA dan protokol Diffie-Hellman, yang didasari oleh kriptografi asimetri, rentan terhadap penyerangan.
6. Modifikasi pada aritmatika modulo yang dipakai, khusunya pada perhitungan eksponensiasi modular, dapat membantu untuk mencegah serangan.

Daftar Pustaka

[1] Munir, Rinaldi. (2004). Diktat Kuliah IF2151 Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Kocher, Paul C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other System. Cryptography Research Inc, San Fransisco.

[3] Bishop, David. (2003). Introduction to Cryptography with Java™ Applets. Jones and Barlett Publisher, Massachusetts.

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

[5] RSA. (2006). *http://en.wikipedia.org/wiki/RSA*. Tanggal akses: 28 Desember 2006 pukul 19.49.

[6] Cryptography. (2006). *http://en.wikipedia.org/wiki/Cryptography*. Tanggal akses: 29 Desember 2006 pukul 19.23.

[7] Diffie-Hellman. (2006). *http://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange*. Tanggal akses: 29 Desember 2006 pukul 14.19.

[8] Pretty Good Privacy. (2006). *http://en.wikipedia.org/wiki/Pretty_Good_Privacy*. Tanggal akses: 29 Desember 2006 pukul 14.19.

[9] Transport Layer Security. (2006). *http://en.wikipedia.org/wiki/Transport_Layer_Security*. Tanggal akses: 29 Desember 2006 pukul 13.11.