KRIPTOGRAFI DAN PEMANFAATANNYA PADA RSA DAN MD5

Tiffany Adriana – NIM 13505068

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15068@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang kriptografi dan penggunaannya di berbagai bidang. Kriptografi merupakan ilmu dan seni yang mempelajari tentang cara menjaga kerahasiaan berita. Suatu berita dapat tetap terjaga kerahasiaannya dengan menggunakan suatu teknik yang disebut enkripsi. Melalui proses enkripsi, suatu berita dirubah menjadi bentuk lain yang tidak dapat langsung dibaca manusia maupun mesin. Dibutuhkan suatu kunci tertentu untuk dapat membaca berita yang telah dienkripsi. Proses ini dinamakan dekripsi.

Selain kriptografi secara umum, pada makalah ini akan dijelaskan mengenai beberapa algoritma dan fungsi yang memanfaatkan kriptografi. RSA yang merupakan algoritma pada enkripsi *pubic key* akan dijabarkan di sini. RSA merupakan salah satu algoritma yang paling maju dalam bidang kriptografi *public key*. RSA dipercaya dalam mengamankan dengan menggunakan kunci yang cukup panjang.

Pada makalah ini akan dijelaskan pula mengenai MD5 yang merupakan fungsi hash kriptografik. MD5 digunakan secara luas dengan *hash value* 128-bit. MD5 telah dimanfaatkan secara bermacam-macam pada aplikasi keamanan, dan MD5 juga umum digunkan untuk melakukan pengujian integritas sebuah file.

1. KRIPTOGRAFI

1.1 Defenisi

Secara umum, kriptografi adalah ilmu dan seni untuk menjaga kerahasiaan berita. Kriptografi (*cryptography*) dilakukan oleh seorang kriptografer. Ada empat tujuan utama dari kriptografi:

1. Kerahasiaan (*confidentiality*)
   Kriptografi digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka / mengupas informasi yang telah disandi. Kerahasiaan dijaga dengan melakukan enkripsi (penyandian).
2. Keutuhan (*integrity*)
   Berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak. Bentuk-bentuk manipulasi yang dapat dilakukan antara lain penyisipan, penghapusan, dan pensubstitusian data lain ke dalam data yang sebenarnya.
3. Autentikasi
   Berhubungan dengan identifikasi / pengenalan, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.
4. Non-repudasi
   Atau nirpenyangkalan, adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman / terciptanya suatu informasi oleh yang mengirimkan / membuat.

1.2 Algoritma Kriptografi

Suatu pesan yang tidak disandikan disebut sebagai *plaintext* ataupun dapat disebut juga sebagai *cleartext*. Proses yang dilakukan untuk mengubah plaintext ke dalam ciphertext disebut *encryption* atau *enciphering*. Sedangkan proses untuk mengubah ciphertext kembali ke plaintext disebut *decryption* atau *deciphering*.

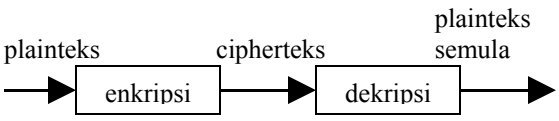
Algoritma kriptografi merupakan aturan untuk *enchipering* dan *dechipering*. Algoritma kriptografi dapat ditulis dalam suatu bentuk fungsi matematika yang digunakan untuk enkripsi dan dekripsi. Dasar matematis yang mendasari proses enkripsi dan dekripsi adalah relasi antara dua himpunan, yaitu yang berisi elemen teks terang / plaintext dan yang berisi elemen teks sandi / ciphertext. Enkripsi dan dekripsi merupakan fungsi transformasi antara himpunan-himpunan tersebut. Apabila elemen-elemen plaintext dinotasikan dengan P, elemen-elemen ciphertext dinotasikan dengan C, sedang untuk proses enkripsi dinotasikan dengan E, dekripsi dengan notasi D, maka rumus matematis untuk enchipering dan dechipering dapat ditulis sebagai berikut:

Enkripsi : $E(P) = C$

Dekripsi : $D(C) = P$

atau

$D(E(P)) = P$



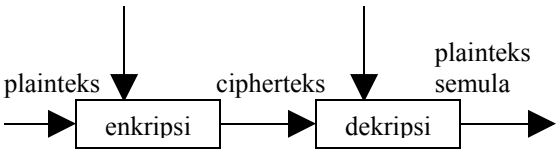
Kunci adalah parameter yang digunakan untuk transformasi *enciphering* dan *dechipering.* Proses enkripsi dan dekripsi diatur oleh satu atau beberapa kunci kriptografi. Secara umum, kunci-kunci yang digunakan untuk proses pengenkripsian dan pendekripsian tidak perlu identik, tergantung pada sistem yang digunakan. Dengan menggunakan kunci K, maka fungsi enkripsi dan dekripsi menjadi

$$E_K(P) = C$$
$$D_K(C) = P$$

dan kedua fungsi ini memenuhi

$$D_K(E_K(P)) = P$$

K K



Dengan demikian keamanan suatu pesan tergantung pada kunci ataupun kunci-kunci yang digunakan, dan tidak tergantung pada algoritma yang digunakan. Sehingga algoritma-algoritma yang digunakan tersebut dapat dipublikasikan dan dianalisis, serta produk-produk yang menggunakan algoritma tersebut dapat diproduksi massal. Tidaklah menjadi masalah apabila seseorang mengetahui algoritma yang kita gunakan. Selama ia tidak mengetahui kunci yang dipakai, ia tetap tidak dapat membaca pesan.

Algoritma kriptografi harus memiliki kekuatan untuk melakukan:

- Konfusi / pembingungan (*confusion*), dari plaintext sehingga sulit untuk direkonstruksikan secara langsung tanpa menggunakan algoritma dekripsinya
- Difusi / peleburan (*difusion*), dari plaintext sehingga karakteristik dari plaintext tersebut hilang.

sehingga dapat digunakan untuk mengamankan informasi. Pada implementasinya, sebuah Algoritma Kriptografi harus memperhatikan kualitas layanan / *Quality of Service* atau QoS dari keseluruhan sistem dimana Algoritma Kriptografi tersebut diimplementasikan. Algoritma kriptografi yang handal adalah algoritma kriptografi yang kekuatannya terletak pada kunci, bukan pada kerahasiaan algoritma itu sendiri.

1.3 Sistem Kriptografi (*Cryptosystem*)

Suatu cryptosystem terdiri dari sebuah algoritma, seluruh kemungkinan plaintext, ciphertext dan kunci-kunci. Secara umum berdasarkan kesamaan kuncinya, *cryptosystem* dibedakan menjadi :

- kunci-simetris / *symmetric-key*, sering disebut juga *cryptosystem* konvensional karena umumnya diterapkan pada *cryptosystem* klasik
- kunci-asimetris / *assymmetric-key*

Berdasarkan arah implementasi dan pembabakan jamannya dibedakan menjadi :

- *cryptosystem* klasik (*classic cryptography*)
- *cryptosystem* modern (*modern cryptography*)

Berdasarkan kerahasiaan kuncinya dibedakan menjadi :

- *cryptosystem* kunci rahasia (*private-key*)
- *cryptosystem* kunci publik (*public-key*)

Berikut akan dibahas mengenai symmetric-key cryptosystem dan assymmetric-key cryptosystem

1.3.1 Symmetric Cryptosystem

Dalam symmetric cryptosystem ini, kunci yang digunakan untuk proses enkripsi dan dekripsi pada prinsipnya identik, tetapi satu buah kunci dapat pula diturunkan dari kunci yang lainnya. Kunci-kunci ini harus dirahasiakan. Oleh karena itulah sistem ini sering disebut sebagai *secret-key ciphersystem*. Jumlah kunci yang dibutuhkan umumnya adalah :

$$ {}_nC_2 = \frac{n \,.\, (n-1)}{2} $$

dengan n menyatakan banyaknya pengguna.

Berdasarkan jumlah data per proses dan alur pengolahan data di dalamnya, Symmetric Cryptosystem ini dibedakan menjadi dua kelas, yaitu block-cipher dan stream-cipher.

a. Block-Cipher

Block-cipher adalah skema *cryptosystem* yang akan membagi-bagi plaintext yang akan dikirimkan dengan ukuran tertentu (disebut blok) dengan panjang t, dan setiap blok dienkripsi dengan menggunakan kunci yang sama. Pada umumnya, block-cipher memproses plaintext dengan blok yang relatif panjang lebih dari 64 bit, untuk mempersulit penggunaan pola-pola serangan yang ada untuk membongkar kunci. Untuk menambah kehandalan model *cryptosystem* ini, dikembangkan pula beberapa tipe proses enkripsi, yaitu :

- ECB, Electronic Code Book
- CBC, Cipher Block Chaining
- OFB, Output Feed Back
- CFB, Cipher Feed Back

b. **Stream-Cipher**

Stream-cipher adalah *cryptosystem* yang mengenkripsi data persatuan data, seperti bit, byte, nible atau per lima bit(saat data yang dienkripsi berupa data Boudout). Setiap mengenkripsi satu satuan data di gunakan kunci yang merupakan hasil pembangkitan dari kunci sebelum.

Beberapa Contoh dari Symmetric Cryptosystem

- DES - Data Encryption Standard
- blowfish
- twofish
- MARS
- IDEA
- 3DES - DES diaplikasikan 3 kali
- AES - Advanced Encryption Standard,

1.3.2 Assymmetric Cryptosystem

Skema ini adalah algoritma yang menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsinya. Skema ini disebut juga sebagai sistem kriptografi kunci publik karena kunci untuk enkripsi dibuat untuk diketahui oleh umum (*public-key*) atau dapat diketahui siapa saja, tapi untuk proses dekripsinya hanya dapat dilakukan oleh yang berwenang yang memiliki kunci rahasia untuk mendekripsinya, disebut *private-key*. Dapat dianalogikan seperti kotak pos yang hanya dapat dibuka oleh tukang pos yang memiliki kunci tapi setiap orang dapat memasukkan surat ke dalam kotak tersebut.

Keuntungan algoritma model ini, untuk berkorespondensi secara rahasia dengan banyak pihak tidak diperlukan kunci rahasia sebanyak jumlah pihak tersebut, cukup membuat dua buah kunci, yaitu *public key* bagi para korensponden untuk mengenkripsi pesan, dan *private key* untuk mendekripsi pesan. Berbeda dengan Symmetric Cryptosystem, jumlah kunci yang dibuat adalah sebanyak jumlah pihak yang diajak berkorespondensi.

Fungsi Enkripsi dan Dekripsi Assymmetric Cryptosystem

Apabila **A** dan **B** hendak bertukar informasi dan berkomunikasi, maka:

1. A dan B masing-masing membuat 2 buah kunci
   - **A** membuat dua buah kunci, *public-key* K_{public[A]} dan *private-key* K_{private[A]}
   - **B** membuat dua buah kunci, *public-key* K_{public[B]} dan *private-key* K_{private[B]}
2. Mereka berkomunikasi dengan cara:
   - **A** dan **B** saling bertukar *public-key*. **B** mendapatkan K_{public[A]} dari **A**, dan **A** mendapatkan K_{public[B]} dari **B**.
   - **A** mengenkripsi plaintext P ke **B** dengan fungsi C = E(P,K_{public[B]})
   - **A** mengirim ciphertext C ke **B**
   - **B** menerima C dari **A** dan membuka plaintext dengan fungsi P = D(C,K_{private[B]})

Hal yang sama terjadi apabila **B** hendak mengirimkan pesan ke **A**

1. **B** mengenkripsi plaintext P ke **A** dengan fungsi C = E(P,K_{public[A]})
2. **A** menerima C dari **B** dan membuka plaintext dengan fungsi P = D(C,K_{private[A]})

Contoh Assymetric Cryptosystem antara lain:

- Knapsack
- RSA - Rivert-Shamir-Adelman
- Diffie-Hellman

Setiap cryptosytem yang baik harus memiliki karakteristik sebagai berikut :

- Keamanan sistem terletak pada kerahasiaan kunci dan bukan pada kerahasiaan algoritma yang digunakan.
- Cryptosystem yang baik memiliki ruang kunci (keyspace) yang besar.
- Cryptosystem yang baik akan menghasilkan ciphertext yang terlihat acak dalam seluruh tes statistik yang dilakukan terhadapnya.
- Cryptosystem yang baik mampu menahan seluruh serangan yang telah dikenal sebelumnya

Namun demikian perlu diperhatikan bahwa bila suatu cryptosystem berhasil memenuhi seluruh karateristik di atas belum tentu ia merupakan sistem yang baik. Banyak cryptosystem lemah yang terlihat baik pada awalnya. Kadang kala untuk menunjukkan bahwa suatu cryptosystem kuat atau baik dapat dilakukan dengan menggunakan pembuktian matematika.

Hingga saat ini masih banyak orang yang menggunakan cryptosystem yang relatif mudah dibuka. Alasan dari penggunaan ini adalah mereka tidak mengetahui sistem lain yang lebih baik, serta kadang kala terdapat motivasi yang kurang untuk menginvestasikan seluruh usaha yang diperlukan untuk membuka suatu sistem.

1.4 Fungsi Hash Kriptografis

Fungsi hash adalah fungsi yang secara efisien mengubah string input dengan panjang berhingga menjadi string output dengan panjang tetap yang disebut nilai hash.

Fungsi hash Kriptografis adalah fungsi hash yang memiliki beberapa sifat keamanan tambahan sehingga dapat dipakai untuk tujuan keamanan data. Umumnya digunakan untuk keperluan autentikasi dan integritas data.

Sifat-Sifat Fungsi Hash Kriptografi

- Tahan preimej (*Preimage resistant*)
  Bila diketahui nilai hash h maka sulit (secara komputasi tidak layak) untuk mendapatkan m dimana h = hash(m).
- Tahan preimej kedua (*Second preimage resistant*)
  Bila diketahui input m1 maka sulit mencari input m2 (tidak sama dengan m1) yang menyebabkan hash(m1) = hash(m2).
- Tahan tumbukan (*Collision-resistant*)
  Sulit mencari dua input berbeda m1 dan m2 yang menyebabkan hash(m1) = hash(m2)

Beberapa contoh algoritma fungsi hash Kriptografi:

- MD4
- MD5
- SHA-0
- SHA-1
- SHA-256
- SHA-512

1.5 Cryptographic Protokol

Suatu protokol adalah serangkaian langkah yang melibatkan dua pihak atau lebih dan dirancang untuk menyelesaikan suatu tugas.

Dari definisi ini dapat diambil beberapa arti sebagai berikut :

- protokol memiliki urutan dari awal hingga akhir
- setiap langkah harus dilaksanakan secara bergiliran
- suatu langkah tidak dapat dikerjakan bila langkah sebelumnya belum selesai
- diperlukan dua pihak atau lebih untuk melaksanakan protokol
- protokol harus mencapai suatu hasil

Selain itu, suatu protokol pun memiliki karakteristik yang lain, yaitu :

- setiap orang yang terlibat dalam protokol harus mengetahui terlebih dahulu mengenai protokol dan seluruh langkah yang akan dilaksanakan
- setiap orang yang terlibat dalam protokol harus menyetujui untuk mengikutinya
- protokol tidak boleh menimbulkan kerancuan
- protokol harus lengkap

Cryptographic protocol adalah suatu protokol yang menggunakan kriptografi. Protokol ini melibatkan sejumlah algoritma kriptografi, namun secara umum tujuan protokol lebih dari sekedar kerahasiaan. Pihak-pihak yang berpartisipasi mungkin saja ingin membagi sebagian rahasianya untuk menghitung sebuah nilai, menghasilkan urutan random, atau pun menandatangani kontrak secara bersamaan.

Penggunaan kriptografi dalam sebuah protokol terutama ditujukan untuk mencegah atau pun mendeteksi adanya *eavesdropping* dan *cheating*.

Fungsi Protokol

Dalam kehidupan kita sehari-hari terdapat banyak sekali protokol tidak resmi, misalnya saja dalam permainan kartu, pemungutan suara dalam pemilihan umum. Akan tetapi tidak ada seorang pun yang memikirkan mengenai protokol-protokol ini, protokol-protokol ini terus berkembang, semua orang mengetahui bagaimana menggunakannya.

Saat ini, semakin banyak interaksi antar manusia dilakukan melalui jaringan komputer. Komputer ini tentu saja memerlukan suatu protokol formal agar dapat melakukan hal yang biasa dilakukan manusia tanpa berpikir. Bila kita berpindah dari satu daerah ke daerah lain dan mengetahui bahwa kartu pemilihan suaranya berbeda dengan yang biasa kita gunakan, kita dapat beradaptasi dengan mudah. Akan tetapi kemampuan ini belum dimiliki oleh komputer, sehingga diperlukan suatu protokol.

Protokol digunakan untuk mengabtraksikan proses penyelesaian suatu tugas dari mekanisme yang digunakan. Protokol komunikasi adalah sama meskipun diimplementasikan pada PC atau VAX. Bila kita yakin bahwa kita memiliki protokol yang baik, kita dapat mengimplementasikannya dalam segala benda mulai dari telepon hingga pemanggang roti cerdas.

Penyerangan terhadap protokol

Penyerangan cryptographic dapat ditujukan pada beberapa hal berikut :

- algoritma cryptographic yang digunakan dalam protokol
- teknik cryptographic yang digunakan untuk mengimplementasikan algoritma dan protokol
- protokol itu sendiri

Seseorang dapat mencoba berbagai cara untuk menyerang suatu protokol. Mereka yang tidak terlibat dalam protokol dapat menyadap sebagian atau seluruh protokol. Tindakan ini disebut penyerangan pasif, karena si penyerang tidak mempengaruhi atau mengubah protokol, ia hanya mengamati protokol dan berusaha untuk memperoleh informasi.

Selain itu, seorang penyerang dapat berusaha untuk mengubah protokol demi keuntungannya sendiri. Ia dapat mengirimkan pesan dalam protokol, menghapus pesan, atau bahkan mengubah informasi yang ada di dalam suatu komputer. Tindakan-tindakan ini disebut sebagai penyerangan aktif, karena ia membutuhkan suatu campur tangan aktif.

Seorang penyerang tidaklah hanya berasal dari lingkungan luar protokol, namun ia mungkin juga berasal dari dalam protokol itu sendiri, ia dapat merupakan salah satu pihak yang terlibat dalam protokol. Tipe penyerang semacam ini disebut sebagai *cheater*. *Passive cheater* mengikuti protokol, tetapi berusaha memperoleh informasi lebih banyak daripada yang diperbolehkan protokol bagi dirinya. *Active cheater* mengubah protokol dalam usahanya untuk berbuat curang.

Usaha untuk menjaga keamanan protokol akan semakin sulit apabila pihak-pihak yang terlibat umumnya merupakan active cheater, oleh karena itu suatu protokol yang baik harus mampu atau pun harus aman terhadap kemungkinan *passive cheating*.

1.6 Berbagai macam basic cryptanalytic attacks

Yang dimaksud *cryptanalytic attacks* adalah usaha-usaha yang dilakukan seseorang untuk memperoleh informasi ataupun data yang telah dienkripsi.

Tujuan *cryptanalytic attack* adalah untuk mengetahui beberapa plaintext yang sesuai dengan ciphertext yang ada dan berusaha menentukan kunci yang memetakan satu dengan yang lainnya. Plaintext ini dapat diketahui karena ia merupakan standar atau karena pendugaan. Jika suatu teks diduga berada di dalam suatu pesan, posisinya mungkin tidak diketahui, tetapi suatu pesan lazimnya cukup pendek sehingga memungkinkan *cryptanalyst* menduga plaintext yang diketahui dalam setiap posisi yang mungkin dan melakukan penyerangan pada setiap kasus secara paralel.

Suatu algoritma enkripsi yang kuat tidak hanya mampu bertahan terhadap serangan plaintext yang dikenal tetapi juga mampu bertahan terhadap *adaptive chosen plaintext*. Dalam penyerangan ini, cryptanalyst berkesempatan memilih plaintext yang digunakan dan dapat melakukannya secara berulang kali. Memilih plaintext untuk tahap N+1 setelah menganalisis hasil tahap N.

Secara ringkas terdapat tujuh macam basic cryptanalytic attacks berdasarkan tingkat kesulitannya bagi penyerang, dimulai dari yang paling sulit adalah :

- *Ciphertext-only attack*
  Dalam penyerangan ini, seorang cryptanalyst memiliki ciphertext dari sejumlah pesan yang seluruhnya telah dienkripsi menggunakan algoritma yang sama.
- *Known-plaintext attack*
  Dalam tipe penyerangan ini, cryptanalyst memiliki akses tidak hanya ke ciphertext sejumlah pesan, namun ia juga memiliki plaintext pesan-pesan tersebut.
- *Chosen-plaintext attack*
  Pada penyerangan ini, cryptanalyst tidak hanya memiliki akses atas ciphertext dan plaintext untuk beberapa pesan, tetapi ia juga dapat memilih plaintext yang dienkripsi.
- *Adaptive-chosen-plaintext attack*
  Penyerangan tipe ini merupakan suatu kasus khusus chosen-plaintext attack. Cryptanalyst tidak hanya dapat memilih plaintext yang dienkripsi, ia pun memiliki kemampuan untuk memodifikasi pilihan berdasarkan hasil enkripsi sebelumnya. Dalam *chosen-plaintext attack*, cryptanalyst mungkin hanya dapat memiliki plaintext dalam suatu blok besar untuk dienkripsi. Sedangkan dalam *adaptive-chosen-plaintext attack* ini ia dapat memilih blok plaintext yang lebih kecil dan kemudian memilih yang lain berdasarkan hasil yang pertama, proses ini dapat dilakukannya terus menerus hingga ia dapat memperoleh seluruh informasi.
- *Chosen-ciphertext attack*
  Pada tipe ini, cryptanalyst dapat memilih ciphertext yang berbeda untuk didekripsi dan memiliki akses atas plaintext yang didekripsi.
- *Chosen-key attack*
  Cryptanalyst pada tipe penyerangan ini memiliki pengetahuan tentang hubungan antara kunci-kunci yang berbeda.
- *Rubber-hose cryptanalysis*
  Pada tipe penyerangan ini, cryptanalyst mengancam, memeras, atau bahkan memaksa seseorang hingga mereka memberikan kuncinya.

Berdasarkan bagaimana cara dan posisi seseorang mendapatkan pesan-pesan dalam saluran komunikasi, penyerangan dapat dikategorikan menjadi:

1. *Sniffing*
   secara harafiah berarti mengendus,. Tentunya dalam hal ini yang diendus adalah pesan (baik yang belum ataupun sudah dienkripsi) dalam suatu saluran komunikasi. Hal ini umum terjadi pada saluran publik yang tidak aman. Sang pengendus dapat merekam pembicaraan yang terjadi.
2. *Replay attack*
   Jika seseorang bisa merekam pesan-pesan *handshake* (persiapan komunikasi), ia mungkin dapat mengulang pesan-pesan yang telah direkamnya untuk menipu salah satu pihak.
3. *Spoofing*

Penyerang bisa menyamar menjadi pihak pertama. Penyerang berusaha meyakinkan pihak-pihak lain bahwa tak ada yang salah dengan komunikasi yang dilakukan, padahal komunikasi itu dilakukan dengan sang penipu/penyerang. Contohnya jika orang memasukkan PIN ke dalam mesin ATM palsu – yang benar-benar dibuat seperti ATM asli – tentu sang penipu bisa mendapatkan PIN-nya dan copy pita magentik kartu ATM milik sang nasabah. Pihak bank tidak tahu bahwa telah terjadi kejahatan.

4. *Man-in-the-middle*

Jika *spoofing* terkadang hanya menipu satu pihak, maka pada *man-in-the-middle*, saat A hendak berkomunikasi dengan B, C di mata A seolah-olah adalah B, dan C dapat pula menipu B sehingga C seolah-olah adalah A. C dapat berkuasa penuh atas jalur komunikas ini, dan bisa membuat berita fitnah.

Kabel koaksial yang sering dipergunakan pada jaringan sangat rentan terhadap serangan *vampire tap*, yakni perangkat keras sederhana yang bisa menembus bagian dalam kabel koaksial sehingga dapat mengambil data yang mengalir tanpa perlu memutuskan komunikasi data yang sedang berjalan. Seseorang dengan *vampire tap* dan komputer jinjing dapat melakukan serangan pada bagian apa saja dari kabel koaksial.

1.7 Analisis berbagai tipe penyerangan secara matematis

Suatu penyerangan pasif atas cryptosystem adalah semua metode untuk mengungkapkan informasi tentang plaintext dan ciphertextnya dengan tanpa mengetahui kunci.

Jika ditinjau secara matematis :
Diberikan fungsi F, G, dan H yang terdiri dari n variabel.
Diberikan sistem enkripsi E.
Diberikan suatu distribusi plaintext dan kunci.

Suatu penyerangan atas E dengan menggunakan G dengan mengasumsikan F membagi H dengan probabilitas p adalah suatu algoritma A dengan sepasang input f,g dan satu buah output h sedemikian hingga terdapat probabilitas p atas

$h = H(P_1, \ldots, P_n)$, jika kita memiliki
$f = F(P_1, \ldots, P_n)$ dan
$g = G(E_K(P_1), \ldots, EK(P_n))$.

Perlu diperhatikan bahwa probabilitas ini tergantung pada distribusi vektor-vektor $(K,P_1,\ldots,P_n)$.

Penyerangan akan merupakan suatu trivial bila terdapat probabilitas paling sedikit p untuk
$h = H(P_1, \ldots, P_n)$ jika
$f = F(P_1,\ldots,P_n)$ dan
$g = G(C_1,\ldots,C_n)$.

Di sini $C_1,\ldots,C_n$ terletak pada ciphertext yang mungkin, dan tidak memiliki hubungan tertentu dengan $P_1,\ldots,P_n$.

Dengan kata lain, suatu serangan akan merupakan trivial bila ia tidak benar-benar menggunakan enkripsi
$E_K(P_1),\ldots,E_K(P_n)$.

Dengan merumuskan penyerangan secara matematis, kita dapat secara tepat memformulasikan dan bahkan membuktikan pernyataan bahwa suatu cryptosystem itu kuat.

Kita katakan, sebagai contoh, bahwa suatu cryptosystem adalah aman terhadap seluruh penyerangan pasif jika sembarang penyerangan nontrivial terhadapnya tidak praktis. Jika kita dapat membuktikan pernyataan ini maka kita akan memiliki keyakinan bahwa cryptosystem kita akan bertahan terhadap seluruh teknik cryptanalytic pasif. Jika kita dapat mereduksi pernyataan ini hingga pada beberapa masalah yang tidak terpecahkan maka kita masih tetap memiliki keyakinan bahwa cryptosystem kita tidak mudah dibuka.

1.7.1 Ciphertext-only attack

Dengan menggunakan notasi di atas, suatu *ciphertext-only attack* adalah suatu penyerangan dengan F adalah konstanta. Diberikan hanya beberapa informasi $G(E_K(P_1),..E_K(P_n))$ tentang n ciphertext, penyerangan harus memiliki kesempatan menghasilkan beberapa informasi $H(P_1,\ldots,P_n)$ tentang plaintext. Penyerangan akan merupakan suatu trivial bila ia hanya menghasilkan
$H(P_1,\ldots,P_n)$ ketika diberikan
$G(C_1,\ldots,C_n)$ untuk $C_1,\ldots,C_n$ acak.

Sebagai contoh, misalkan
G (C) = C dan
H(P) adalah bit pertama P
Kita dapat secara mudah menulis suatu penyerangan, pendugaan, yang menduga bahwa H(P) adalah 1. Penyerangan ini adalah trivial karena tidak menggunakan ciphertext, probabilitas keberhasilannya adalah 50 %.

Di lain pihak, terdapat penyerangan atas RSA yang memproduksi satu bit informasi tentang P, dengan probabilitas keberhasilan 100 %, menggunakan C. Jika diberikan suatu C acak maka tingkat kesuksesan turun menjadi 50%. Inilah yang disebut penyerangan nontrivial.

1.7.2 Known-plaintext attack

Penyerangan known-plaintext klasik memiliki $F(P_1,P_2) = P1$, $G(C_1,C_2) = (C_1,C_2)$, dan $H(P_1,P_2)$ tergantung hanya pada P_2. Dengan kata lain, bila diberikan dua ciphertext C_1 dan C_2 dan satu dekripsi P_1, penyerangan known-plaintext seharusnya menghasilkan informasi tentang dekripsi P_2.

1.7.3 Brute-force attack

Umpamakan penyerangan known-plaintext berikut. Kita diberikan sejumlah plaintext $P_1,\ldots,P_{n-1}$ dan ciphertext $C_1,\ldots,C_{n-1}$. Kita juga diberikan sebuah ciphertext Cn. Kita jalankan seluruh kunci K. Bila kita temukan K sedemikian sehingga $E_K(P_1) = C_i$ untuk setiap I<n, kita cetak $D_K(C_n)$.

Jika n cukup besar sehingga hanya satu kunci yang bekerja, penyerangan ini akan sukses untuk seluruh input yang valid pada setiap waktu, sementara ia akan menghasilkan hasil yang tepat hanya sekali untuk input acak. Penyerangan ini adalah nontrivial, masalahnya ia sangat lambat bila terdapat banyak kemungkinan kunci.

2. BEBERAPA APLIKASI KRIPTOGRAFI

2.1 RSA

Dibidang kriptografi, **RSA** adalah sebuah algoritma pada enkripsi *public key*. RSA merupakan algoritma pertama yang cocok untuk *digital signature* seperti halnya enkripsi, dan salah satu yang paling maju dalam bidang kriptografi *public key*. RSA masih digunakan secara luas dalam protokol *electronic commerce*, dan dipercaya dalam mengamankan dengan menggunakan kunci yang cukup panjang.

2.1.1 Sejarah RSA

Algortima RSA dijabarkan pada tahun 1977 oleh Ron Rivest, Adi Shamir dan Len Adleman dari Massachusetts Institute of Technology, huruf **RSA** itu sendiri juga berasal dari inisial nama mereka (**R**ivest—**S**hamir—**A**dleman).

Clifford Cocks, seorang matematikawan Inggris yang bekerja untuk GCHQ, menjabarkan tentang sistem equivalen pada dokumen internal di tahun 1973. Penemuan Clifford Cocks tidak terungkap hingga tahun 1997 dikarenan alasan *top-secret classification*.

Algoritma tersebut dipatenkan oleh Massachusetts Institute of Technology pada tahun 1983 di Amerika Serikat sebagai U.S. Patent 4405829. Paten tersebut berlaku hingga 21 September 2000. Semenjak Algoritma RSA dipublikasikan sebagai aplikasi paten, regulasi di sebagian besar negara-negara lain tidak memungkinkan penggunaan paten. Hal ini menyebabkan hasil temuan Clifford Cocks di kenal secara umum, paten di Amerika Serikat tidak dapat mematenkannya.

2.1.2 Operasional

2.1.2.1 Pembangkitan Kunci

Semisal **A** berkeinginan untuk mengijinkan **B** untuk mengirimkan kepadanya sebuah pesan pribadi (*private message*) melalui media transmisi yang tidak aman (*insecure*). **A** melakukan langkah-langkah berikut untuk membuat sebuah *public key* dan *private key*:

1. Pilih dua bilangan prima $p \neq q$ secara acak dan terpisah untuk tiap-tiap p dan q. Hitung $N = p\,q$. N hasil perkalian dari p dikalikan dengan q.
2. Hitung $\varphi = (p\text{-}1)(q\text{-}1)$.
3. Pilih bilangan bulat (*integer*) antara satu dan φ ($1 < e < \varphi$) yang juga merupakan coprime dari φ.
4. Hitung d hingga $d\,e \equiv 1 \pmod{\varphi}$.

- bilangan prima dapat diuji probabilitasnya menggunakan *Fermat's little theorem*- a^(n-1) mod n = 1 jika n adalah bilangan prima, diuji dengan beberapa nilai a menghasilkan kemungkinan yang tinggi bahwa n ialah bilangan prima. *Carmichael numbers* (angka-angka Carmichael) dapat melalui pengujian dari seluruh a, tetapi hal ini sangatlah langka
- langkah 3 dan 4 dapat dihasilkan dengan algoritma *extended Euclidean*; lihat juga aritmetika modular
- langkah 4 dapat dihasilkan dengan menemukan integer x sehingga $d = (x(p\text{-}$

1)(q-1) + 1)/e menghasilkan bilangan bulat, kemudian menggunakan nilai dari d (mod (p-1)(q-1))
- langkah 2 PKCS#1 v2.1 menggunakan &lamda; = lcm(p-1, q-1) selain daripada φ = (p-1)(q-1))

Pada *public key* terdiri atas:
- N, modulus yang digunakan.
- e, eksponen publik (sering juga disebut eksponen enkripsi).

Pada *private key* terdiri atas:
- N, modulus yang digunakan, digunakan pula pada *public key*.
- d, eksponen pribadi (sering juga disebut eksponen dekripsi), yang harus dijaga kerahasiaannya.

Biasanya, berbeda dari bentuk *private key* (termasuk parameter CRT):
- p dan q, bilangan prima dari pembangkitan kunci.
- *d mod (p-1)* dan *d mod (q-1)* (dikenal sebagai *dmp1* dan *dmq1*).
- *(1/q) mod p* (dikenal sebagai *iqmp*).

Bentuk ini membuat proses dekripsi lebih cepat dan *signing* menggunakan Chinese Remainder Theorem (CRT). Dalam bentuk ini, seluruh bagian dari *private key* harus dijaga kerahasiaannya.

A mengirimkan *public key* kepada **B**, dan tetap merahasiakan *private key* yang digunakan. p dan q sangat sensitif dikarenakan merupakan faktorial dari N, dan membuat perhitungan dari d menghasilkan e. Jika p dan q tidak disimpan dalam bentuk CRT dari *private key*, maka p dan q telah terhapus bersama nilai-nilai lain dari proses pembangkitan kunci.

2.1.2.2 Proses enkripsi pesan

Misalkan **B** ingin mengirim pesan m ke **A**. **B** mengubah m menjadi angka $n < N$, menggunakan protokol yang sebelumnya telah disepakati dan dikenal sebagai *padding scheme*.

Maka **B** memiliki n dan mengetahui N dan e, yang telah diumumkan oleh **A**. **B** kemudian menghitung *ciphertext c* yang terkait pada n:

c = n^e \mod{N}

Perhitungan tersebut dapat diselesaikan dengan cepat menggunakan metode *exponentiation by squaring*. **B** kemudian mengirimkan c kepada **A**.

2.1.2.3 Proses dekripsi pesan

A menerima c dari **B**, dan mengetahui *private key* yang digunakan oleh **A** sendiri. **A** kemudian memulihkan n dari c dengan langkah-langkah berikut:

n = c^d mod{N}

Perhitungan diatas akan menghasilkan n, dengan begitu **A** dapat mengembalikan pesan semula m. Prosedur dekripsi bekerja karena

c^d ≡ (n^e)^d ≡ n^{ed} p mod{N}

Kemudian, dikarenakan $ed \equiv 1 \pmod{p-1}$ dan $ed \equiv 1 \pmod{q-1}$, hasil dari *Fermat's little theorem*.

n^{ed} ≡ n mod{p}

dan

n^{ed} ≡ n mod{q}

Dikarenakan p dan q merupakan bilangan prima yang berbeda, mengaplikasikan *Chinese remainder theorem* akan menghasilkan dua macam kongruen

n^{ed} ≡ n mod{pq}

serta

c^d ≡ n mod{N}

2.1.2.3 Contoh proses

Berikut ini merupakan contoh dari enkripsi RSA dan dekripsinya. Parameter yang digunakan disini berupa bilangan kecil. Kita membuat:

$p = 61$ — bilangan prima pertama (harus dijaga kerahasiannya atau dihapus secara hati-hati)
$q = 53$ — bilangan prima kedua (harus dijaga kerahasiannya atau dihapus secara hati-hati)
$N = pq = 3233$ — modulus (diberikan kepada publik)
$E = 17$ — eksponen publik (diberikan kepada publik)
$d = 2753$ — eksponen pribadi (dijaga kerahasiannya)

Public key yang digunakan adalah (*e*,*N*). *Private key* yang digunakan adalah *d*.

- Fungsi pada enkripsi ialah:
  encrypt(n) = n^e mod N = n^{17} mod 3233
  dimana *n* adalah *plaintext*

- Fungsi dekripsi ialah:
  decrypt(c) = c^d mod N = c^{2753} mod 3233
  dimana *c* adalah *ciphertext*

  - Untuk melakukan enkripsi *plaintext* bernilai "123", perhitungan yang dilakukan
    encrypt(123) = 123^{17} mod 3233 = 855

  - Untuk melakukan dekripsi *ciphertext* bernilai "855" perhitungan yang dilakukan
    decrypt(855) = 855^{2753} mod 3233 = 123

Kedua perhitungan diatas diselesaikan secara effisien menggunakan *square-and-multiply algorithm* pada *modular exponentiation*.

2.1.2.4 Padding schemes

Padding Scheme harus dibangun secara hati-hati sehingga tidak ada nilai dari *m* yang menyebabkan *masalah* keamanan. Sebagai contoh, jika kita ambil contoh sederhana dari penampilan ASCII dari *m* dan menggabungkan bit-bit secara bersama-sama akan menghasilkan *n*, kemudian pessan yang berisi ASCII tunggal karakter NUL (nilai numeris 0) akan menghasilkan *n*= 0, yang akan menghasilkan *ciphertext* 0 apapun itu nilai dari *e* dan *N* yang digunakan. Sama halnya dengan karakter ASCII tunggal SOH (nilai numeris 1) akan selalu menghasilkan *chiphertext* 1. Pada kenyataannya, untuk sistem yang menggunakan nilai *e* yang kecil, seperti 3, seluruh karakter tunggal ASCII pada pesan akan disandikan menggunakan skema yang tidak aman, dikarenakan nilai terbesar *n* adalah nilai 255, dan 255^3 menghasilkan nilai yang lebih kecil dari modulus yang sewajarnya, maka proses dekripsi akan menjadi masalah sederhana untuk mengambil pola dasar dari *ciphertext* tanpa perlu menggunakan modulus *N*. Sebagai konsekuensinya, standar seperti PKCS didesain dengan sangat hati-hati sehingga membuat pesan asal-asalan dapat terenkripsi secara aman. Dan juga berdasar pada bagian Kecepatan, akan dijelaskan kenapa *m* hampir bukanlah pesan itu sendiri tetapi lebih pada *message key* yang dipilh secara acak.

2.1.3 Pengesahan pesan

RSA dapat juga digunakan untuk mengesahkan sebuah pesan. Misalkan **A** ingin mengirim pesan kepada **B**. **A** membuat sebuah *hash value* dari pesan tersebut, di pangkatkan dengan bilangan *d* dibagi *N* (seperti halnya pada deskripsi pesan), dan melampirkannya sebagai "tanda tangan" pada pesan tersebut. Saat **B** menerima pesan yang telah "ditandatangani", **B** memangkatkan "tanda tangan" tersebut dengan bilangan *e* dibagi *N* (seperti halnya pada enkripsi pesan), dan membandingkannya dengan nilai hasil dari *hash value* dengan *hash value* pada pesan tersebut. Jika kedua cocok, maka **B** dapat mengetahui bahwa pemilik dari pesan tersebut adalah **A**, dan pesan pun tidak pernah diubah sepanjang pengiriman.

Harap dicatat bahwa *padding scheme* merupakan hal yang esensial untuk mengamankan pengesahan pesan seperti halnya pada enkripsi pesan, oleh karena itu kunci yang sama tidak digunakan pada proses enkripsi dan pengesahan.

2.1.4 Keamanan

Penyerangan yang paling umum pada RSA ialah pada penanganan masalah faktorisasi pada bilangan yang sangat besar. Apabila terdapat faktorisasi metode yang baru dan cepat telah dikembangkan, maka ada kemungkinan untuk membongkar RSA.

Pada tahun 2005, bilangan faktorisasi terbesar yang digunakan secara umum ialah sepanjang 663 bit, menggunakan metode distribusi mutakhir. Kunci RSA pada umumnya sepanjang 1024—2048 bit. Beberapa pakar meyakini bahwa kunci 1024-bit ada kemungkinan dipecahkan pada waktu

dekat (hal ini masih dalam perdebatan), tetapi tidak ada seorangpun yang berpendapat kunci 2048-bit akan pecah pada masa depan yang terprediksi.

Semisal **E**, seorang *eavesdropper* (pencuri dengar—penguping), mendapatkan *public key* *N* dan *e*, dan ciphertext *c*. Bagimanapun juga, **E** tidak mampu untuk secara langsung memperoleh *d* yang dijaga kerahasiannya oleh **A**. Masalah untuk menemukan *n* seperti pada $n^e=c$ mod N di kenal sebagai permasalahan RSA.

Cara paling efektif yang ditempuh oleh **E** untuk memperoleh *n* dari *c* ialah dengan melakukan faktorisasi *N* kedalam *p* dan *q*, dengan tujuan untuk menghitung $(p-1)(q-1)$ yang dapat menghasilkan *d* dari *e*. Tidak ada metode waktu polinomial untuk melakukan faktorisasi pada bilangan bulat berukuran besar di komputer saat ini, tapi hal tersebut pun masih belum terbukti.

Masih belum ada bukti pula bahwa melakukan faktorisasi *N* adalah satu-satunya cara untuk memperoleh *n* dari *c*, tetapi tidak ditemukan adanya metode yang lebih mudah (setidaknya dari sepengatahuan publik).

Bagaimanapun juga, secara umum dianggap bahwa **E** telah kalah jika *N* berukuran sangat besar.

Jika *N* sepanjang 256-bit atau lebih pendek, *N* akan dapat difaktorisasi dalam beberapa jam pada Personal Computer, dengan menggunakan perangkat lunak yang tersedia secara bebas. Jika *N* sepanjang 512-bit atau lebih pendek, *N* akan dapat difaktorisasi dalam hitungan ratusan jam seperti pada tahun 1999. Secara teori, perangkat keras bernama TWIRL dan penjelasan dari Shamir dan Tromer pada tahun 2003 mengundang berbagai pertanyaan akan keamanan dari kunci 1024-bit. Santa disarankan bahwa *N* setidaknya sepanjang 2048-bit.

Pada thaun 1993, Peter Shor menerbitkan Algoritma Shor, menunjukkan bahwa sebuah komputer quantum secara prinsip dapat melakukan faktorisasi dalam waktu polinomial, mengurai RSA dan algoritma lainnya. Bagaimanapun juga, masih terdapat pedebatan dalam pembangunan komputer quantum secara prinsip.

2.1.5 Pertimbangan praktis

2.1.5.1 Pembangkitan kunci

Menemukan bilangan prima besar *p* dan *q* pada biasanya didapat dengan mencoba serangkaian bilangan acak dengan ukuran yang tepat menggunakan probabilitas bilangan prima yang dapat dengan cepat menghapus hampir semua bilangan bukan prima.

p dan *q* seharusnya tidak "saling-berdekatan", agar faktorisasi fermat pada *N* berhasil. Selain itu pula, jika *p*-1 atau *q*-1 memeiliki faktorisasi bilangan prima yang kecil, *N* dapat difaktorkan secara mudah dan nilai-nilai dari *p* atau *q* dapat diacuhkan.

Seseorang seharusnya tidak melakukan metoda pencarian bilangan prima yang hanya akan memberikan informasi penting tentang bilangan prima tersebut kepada penyerang. Biasanya, pembangkit bilangan acak yang baik akan memulai nilai bilangan yang digunakan. Harap diingat, bahwa kebutuhan disini ialah "acak" ***dan*** "tidak-terduga". Berikut ini mungkin tidak memenuhi kriteria, sebuah bilangan mungkin dapat dipilah dari proses acak (misal, tidak dari pola apapun), tetapi jika bilangan itu mudah untuk ditebak atau diduga (atau mirip dengan bilangan yang mudah ditebak), maka metode tersebut akan kehilangan kemampuan keamanannya. Misalnya, tabel bilangan acak yang diterbitkan oleh Rand Corp pada tahun 1950-an mungkin memang benar-benar teracak, tetapi dikarenakan diterbitkan secara umum, hal ini akan mempermudah para penyerang dalam mendapatkan bilangan tersebut. Jika penyerang dapat menebak separuh dari digit *p* atau *q*, para penyerang dapat dengan cepat menghitung separuh yang lainnya (ditunjukkan oleh Donald Coppersmith pada tahun 1997).

Sangatlah penting bahwa kunci rahasia *d* bernilai cukup besar, Wiener menunjukkan pada tahun 1990 bahwa jika *p* diantara *q* dan $2q$ (yang sangat mirip) dan *d* lebih kecil daripada $N^{1/4}/3$, maka *d* akan dapat dihitung secara effisien dari *N* dan *e*. Kunci enkripsi $e = 2$ sebaiknya tidak digunakan.

2.1.5.2 Kecepatan

RSA memiliki kecepatan yang lebih lambat dibandingkan dengan DES dan algoritma simetrik lainnya. Pada prakteknya, **B** menyandikan pesan rahasia menggunakan algoritma simetrik, menyandikan kunci simetrik menggunakan RSA, dan

mengirimkan kunci simetrik yang dienkripsi menggunakan RSA dan juga mengirimkan pesan yang dienkripasi secara simetrik kepada **A**.

Prosedur ini menambah permasalahan akan keamanan. Singkatnya, Sangatlah penting untuk menggunakan pembangkit bilangan acak yang kuat untuk kunci simetrik yang digunakan, karena **E** dapat melakukan *bypass* terhadap RSA dengan menebak kunci simterik yang digunakan.

2.1.5.3 Distribusi kunci

Sebagaimana halnya chiper, bagaimana public key RSA didistribusi menjadi hal penting dalam keamanan. Distribusi kunci harus aman dari *man-in-the-middle attack* (penghadang-ditengah-jalan). Anggap **E** dengan suatu cara mampu memberikan kunci arbitari kepada **B** dan membuat **B** percaya bahwa kunci tersebut milik **A**. Anggap **E** dapan "menghadang" sepenuhnya transmisi antara **A** dan **B**. **E** mengirim **B** public key milik **E**, dimana **B** percaya bahwa public key tersebut milik **A**. **E** dapat menghadap seluruh ciphertext yang dikirim oleh **B**, melakukan dekripsi dengan kunci rahasia milik **E** sendiri, menyimpan salinan dari pesan tersebut, melakukan enkripsi menggunakan public key milik **A**, dan mengirimkan ciphertext yang baru kepada **A**. Secara prinsip, baik **A** atau **B** tidak menyadari kehadiran **E** diantara transmisi mereka. Pengamanan terhadap serangan semacam ini yaitu menggunakan sertifikat digitalatau kompnen lain dari infrastuktur public key.

2.1.5.4 Penyerangan waktu

Kocher menjelaskan sebuah serangan baru yang cerdas pada RSA di tahun 1995: jika penyerang, **E**, mengetahui perangkat keras yang dimiliki oleh **A** secara terperinci dan mampu untuk mengukur waktu yang dibutuhkan untuk melakukan dekripsi untuk beberapa *ciphertext*, **E** dapat menyimpulkan kunci dekripsi *d* secara cepat. Penyerangan ini dapat juga diaplikasikan pada skema "tanda tangan" RSA. SAlah satu cara untuk mencegah penyerangan ini yaitu dengan memastikan bahwa operasi dekripsi menggunakan waktu yang konstan untuk setiap *ciphertext* yang diproses. Cara yang lainnya, yaitu dengan menggunakan properti multipikatif dari RSA. Sebagai ganti dari menghitung $c^d \bmod N$, **A** pertama-tama memilih nilai bilangan acak *r* dan menghitung $(r^e c)^d \bmod N$. Hasil dari penghitungan tersebut ialah $rm \bmod N$ kemudian efek dari *r* dapat dihilangkan dengan perkalian dengan inversenya. Nilai baru dari *r* dipilih pada tiap *ciphertext*. Dengan teknik ini, dikenal sebagai *message blinding* (pembutaan pesan), waktu yang diperlukan untuk proses dekripsi tidak lagi berhubungan dengan nilai dari *ciphertext* sehingga penyerangan waktu akan gagal.

2.1.5.5 Penyerangan ciphertext adaptive

Pada tahun 1998, Daniel Bleichenbacher menjelaskan penggunaan penyerangan *ciphertext* adaptive, terhadap pesan yang terenkripsi menggunakan RSA dan menggunakan PKCS #1 v1 *padding scheme*. Dikarenakan kecacatan pada skema PKCS #1, Bleichenbacher mampu untuk melakukan serangkaian serangan terhadap implementasi RSA pada protokol Secure Socket Layer, dan secara potensial mengungkap kunci-kunci yang digunakan. Sebagai hasilnya, para pengguna kriptografi menganjurkan untuk menggunakan *padding scheme* yang relatif terbukti aman seperti *Optimal Asymmetric Encryption Padding*, dan Laboratorium RSA telah merilis versi terbaru dari PKCS #1 yang tidak lemah terdapat serangan ini.

2.2 MD5

Dalam kriptografi, **MD5 (*Message-Digest algortihm 5*)** ialah fungsi hash kriptografik yang digunakan secara luas dengan *hash value* 128-bit. Pada standart Internet (RFC 1321), MD5 telah dimanfaatkan secara bermacam-macam pada aplikasi keamanan, dan MD5 juga umum digunkan untuk melakukan pengujian integritas sebuah file.

MD5 di desain oleh Ronald Rivest pada tahun 1991 untuk menggantikan *hash function* sebelumnya, MD4. Pada tahun 1996, sebuah kecacatan ditemukan dalam desainnya, walau bukan kelemahan fatal, pengguna kriptografi mulai menganjurkan menggunakan algoritma lain, seperti SHA-1 (klaim terbaru menyatakan bahwa SHA-1 juga cacat). Pada tahun 2004, kecacatan-kecacatan yang lebih serius ditemukan menyebabkan penggunaan algoritma tersebut

dalam tujuan untuk keamanan jadi makin dipertanyakan.

2.2.1 Sejarah dan kriptoanalisis

MD5 adalah salah satu dari serangkaian algortima *message digest* yang didesain oleh Profesor Ronald Rivest dari MIT (Rivest, 1994). Saat kerja analitik menunjukkan bahwa pendahulu MD5 — MD4 — mulai tidak aman, MD5 kemudian didesain pada tahun 1991 sebagai pengganti dari MD4 (kelemahan MD4 ditemukan oleh Hans Dobbertin).

Pada tahun 1993, den Boer dan Bosselaers memberikan awal, bahkan terbatas, hasil dari penemuan *pseudo-collision* dari fungsi kompresi MD5. Dua vektor inisialisasi berbeda *I* dan *J* dengan beda 4-bit diantara keduanya.

$$MD5compress(I,X) = MD5compress(J,X)$$

Pada tahun 1996 Dobbertin mengumumkan sebuah kerusakan pada fungsi kompresi MD5. Dikarenakan hal ini bukanlah serangan terhadap fungsi *hash* MD5 sepenuhnya, hal ini menyebabkan para pengguna kriptografi menganjurkan pengganti seperti WHIRLPOOL, SHA-1 atau RIPEMD-160.

Ukuran dari *hash* — 128-bit — cukup kecil untuk terjadinya serangan *brute force birthday attack*. MD5CRK adalah proyek distribusi mulai Maret 2004 dengan tujuan untuk menunjukka kelemahan dari MD5 dengan menemukan kerusakan kompresi menggunakan *brute force attack*.

Bagaimanapun juga, MD5CRK berhenti pada tanggal 17 Agustus 2004, saat [[kerusakan *hash*]] pada MD5 diumumkan oleh Xiaoyun Wang, Dengguo Feng, Xuejia Lai dan Hongbo Yu [1][2]. Serangan analitik mereka dikabarkan hanya memerlukan satu jam dengan menggunakan IBM P690 cluster.

Pada tanggal 1 Maret 2005, Arjen Lenstra, Xiaoyun Wang, and Benne de Weger mendemontrasikan[3] kunstruksi dari dua buah sertifikat X.509 dengan *public key* yang berbeda dan *hash* MD5 yang sama, hasil dari demontrasi menunjukkan adanya kerusakan. Konstruksi tersebut melibatkan *private key* untuk kedua *public key* tersebut. Dan beberapa hari setelahnya, Vlastimil Klima menjabarkan[4] dan mengembangkan algortima, mampu membuat kerusakan Md5 dalam beberapa jam dengan menggunakan sebuah komputer notebook. Hal ini menyebabkan MD5 tidak bebas dari kerusakan.

Dikarenakan MD5 hanya menggunakan satu langkah pada data, jika dua buah awalan dengan *hash* yang sama dapat dibangun, sebuah akhiran yang umum dapat ditambahkan pada keduanya untuk membuat kerusakan lebih masuk akal. Dan dikarenakan teknik penemuan kerusakan mengijinkan pendahuluan kondisi *hash* menjadi arbitari tertentu, sebuah kerusakan dapat ditemukan dengan awalan apapun. Proses tersebut memerlukan pembangkitan dua buah file perusak sebagai file templat, dengan menggunakan blok 128-byte dari tatanan data pada 64-byte batasan, file-file tersebut dapat mengubah dengan bebas dengan menggunakan algoritma penemuan kerusakan.

2.2.2 Efek nyata dari kriptoanalisis

Saat ini dapat diketahui, dengan beberapa jam kerja, bagaimana proses pembangkitan kerusakan MD5. Yaitu dengan membangkitkan dua byte *string* dengan *hash* yang sama. Dikarenakan terdapat bilangan yang terbatas pada keluaran MD5 (2^{128}), tetapi terdapat bilangan yang tak terbatas sebagai masukannya, hal ini harus dipahami sebelum kerusakan dapat ditimbulkan, tapi hal ini telah diyakini benar bahwa menemukannya adalah hal yang sulit.

Sebagai hasilnya bahwa *hash* MD5 dari informasi tertentu tidak dapat lagi mengenalinya secara berbeda. Jika ditunjukkan informasi dari sebuah *public key*, *hash* MD5 tidak mengenalinya secata berbeda jika terdapat *public key* selanjutnya yang mempunyai *hash* MD5 yang sama.

Bagaimanapun juga, penyerangan tersebut memerlukan kemampuan untuk memilih kedua pesan kerusakan. Kedua pesan tersebut tidak dengan mudah untuk memberikan serangan *preimage*, menemukan pesan dengan *hash* MD5 yang sudah ditentukan, ataupun *serangan preimage kedua*, menemukan pesan dengan *hash* MD5 yang sama sebagai pesan yang diinginkan.

Hash MD5 lama, yang dibuat sebelum serangan-serangan tersebut diungkap, masih dinilai aman untuk saat ini. Khususnya pada *digital signature* lama masih dianggap layak pakai. Seorang user boleh saja tidak ingin membangkitkan atau mempercayai *signature* baru menggunakan MD5 jika masih ada kemungkinan kecil pada teks (kerusakan dilakukan dengan melibatkan pelompatan beberapa bit pada bagian 128-byte pada masukan *hash*) akan memberikan perubahan yang berarti.

Penjaminan ini berdasar pada posisi saat ini dari kriptoanalisis. Situasi bisa saja berubah secara tiba-tiba, tetapi menemukan kerusakan

dengan beberapa data yang belum-ada adalah permasalahan yang lebih susah lagi, dan akan selalu butuh waktu untuk terjadinya sebuah transisi.

2.2.3 Pengujian Integritas

Ringkasan MD5 digunakan secara luas dalam dunia perangkat lunak untuk menyediakan semacam jaminan bahwa file yang diambil (*download*) belum terdapat perubahan. Seorang user dapat membandingkan MD5 sum yang dipublikasikan dengan *checksum* dari file yang diambil. Dengan asumsi bahwa *checksum* yang dipublikasikan dapat dipercaya akan keasliannya, seorang user dapat secara yakin bahwa dile tersebut adalah file yang sama dengan file yang dirilis oleh para developer, jaminan perlindungan dari *Trojan Horse* dan virus komputer yang ditambahkan pada perangkat lunak. Bagaimanapun juga, seringkali kasus yangterjadi bahwa *checksum* yang dipublikasikan tidak dapat dipercaya (sebagai contoh, *checksum* didapat dari channel atau lokasi yang sama dengan tempat mengambil file), dalam hal ini MD5 hanya mampu melakukan *error-checking*. MD5 akan mengenali file yang didownload tidak sempurna, cacat atau tidak lengkap.

2.2.4 Algortima

Satu operasi MD5 — MD5 terdiri atas 64 operasi, dikelompokkan dalam empat putaran dari 16 operasi. *F* adalah fungsi nonlinear; satu fungsi digunakan pada tiap-tiap putaran. M_i menujukkan blok 32-bit dari masukan pesan, dan K_i menunjukkan konstanta 32-bit, berbeda untuk tiap-tiap operasi.

MD5 memproses variasi panjang pesan kedalam keluaran 128-bit dengan panjang yang tetap. Pesan masukan dipecah menjadi dua gumpalan blok 512-bit; Pesan ditata sehingga panjang pesan dapat dibagi 512. Penataan bekerja sebagai berikut: bit tunggal pertama, 1, diletakkan pada akhir pedan. Proses ini diikuti dengan serangkaian nol (0) yang diperlukan agar panjang pesan lebih dari 64-bit dan kurang dari kelipatan 512. Bit-bit sisa diisi dengan 64-bit integer untuk menunjukkan panjang pesan yang asli. Sebuah pesan selalu ditata setidaknya dengan 1-bit tunggal, seperti jika panjang pesan adalah kelipatan 512 dikurangi 64-bit untuk informasi panjang (panjang mod(512) = 448), sebuah blok baru dari 512-bit ditambahkan dengan 1-bit diikuti dengan 447 bit-bit nol (0) diikuti dengan panjang 64-bit.

Algortima MD5 yang utama beroperasi pada kondisi 128-bit, dibagi menjadi empat *word* 32-bit, menunjukkan *A*, *B*, *C* dan *D*. Operasi tersebut di inisialisasi dijaga untuk tetap konstan. Algoritma utama kemudian beroperasi pada masing-masing blok pesan 512-bit, masing-masing blok melakukan pengubahan terhadap kondisi.Pemrosesan blok pesan terdiri atas empat tahap, batasan *putaran*; tiap putasan membuat 16 operasi serupa berdasar pada fungsi non-linear *F*, tambahan modular, dan rotasi ke kiri.

2.2.5 Pseudocode

Pseudocode pada algoritma MD5 adalah sebagai berikut.

```
//Catatan: Seluruh variable tidak pada
32-bit dan dan wrap modulo 2^32 saat
melakukan perhitungan

//Mendefinisikan r sebagai berikut
var int[64] r, k
r[0..15]
           :={7,12,17,22,7,12,17,22,7,12
           ,17,22,7,12,17,22}
r[16..31]:={5,9,14,20,5,9,14,20,5,9,14,20
           ,5,9,14,20}
r[32..47]:={4,11,16,23,4,11,16,23,4,11,16
           ,23,4,11,16,23}
r[48..63]:={6,10,15,21,6,10,15,21,6,10,15
           ,21, 6,10,15,21}

//Menggunakan bagian fraksional biner
dari integral sinus sebagai konstanta:
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × 2^32)

//Inisialisasi variabel:
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

//Pemrosesan awal:
append "1" bit to message
append "0" bits until message length in
bits ≡ 448 (mod 512)
append bit length of message as 64-bit
little-endian integer to message

//Pengolahan pesan paada kondisi gumpalan
512-bit:
for each 512-bit chunk of message break
chunk into sixteen 32-bit little-endian
words w(i), 0≤i≤15

//Inisialisasi nilai hash pada gumpalan
ini:
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3

//Kalang utama:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
```

```
            f := (b and c) or ((not b)
                 and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d)
                 and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16

        temp := d
        d := c
        c := b
           b := ((a + f + k(i) + w(g))
leftrotate r(i)) + b
        a := temp

        //Tambahkan hash dari gumpalan
sebagai hasil:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d

var int digest := h0 append h1 append h2
append h3 //(diwujudkan dalam little-
endian)
```

Catatan: Meskipun rumusan dari yang tertera pada *RFC 1321*, berikut ini sering digunakan untuk meningkatkan efisiensi:
(0 ≤ i ≤ 15): f := d **xor** (b **and** (c **xor** d))
(16 ≤ i ≤ 31): f := c **xor** (d **and** (b **xor** c))

2.2.6 Hash-hash MD5

Hash-hash MD5 sepanjang 128-bit (16-byte), yang dikenal juga sebagai *ringkasan pesan*, secara tipikal ditampilkan dalam bilangan heksadesimal 32-digit. Berikut ini merupakan contoh pesan ASCII sepanjang 43-byte sebagai masukan dan *hash* MD5 terkait:

MD5("The quick brown fox jumps over the lazy dog") =
9e107d9d372bb6826bd81d3542a419d6

Bahkan perubahan yang kecil pada pesan akan (dengan probabilitas lebih) menghasilkan *hash* yang benar-benar berbeda, misalnya pada kata "dog", huruf d diganti menjadi c:

MD5("The quick brown fox jumps over the lazy cog") =
1055d3e698d289f2af8663725127bd4b

Hash dari panjang-nol ialah:
MD5("") =
d41d8cd98f00b204e9800998ecf8427e

DAFTAR PUSTAKA

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

[2] Wikipedia. (2006). ***http://id.wikipedia.org/wiki/Kriptografi***. Tanggal akses: 30 Desember 2006 pukul 16:00.

[3] Wikipedia. (2006). ***http://id.wikipedia.org/wiki/MD5***. Tanggal akses: 30 Desember 2006 pukul 16:00.