Fungsi *Hash* Kriptografis

Puthut Prabancono
Program Studi Teknik Informatika Institut Teknologi Bandung, Bandung, email: puthutp@students.itb.ac.id

Abstract *– Dalam ilmu komputer, fungsi hash dan tabel hash sering digunakan untuk penyimpanan dalam database. Selain untuk database, fungsi hash juga dapat digunakan dalam kriptografi. Dalam penggunaannya, fungsi hash sering menghasilkan nilai yang sama karena fungsi hash bukanlah fungsi satu-satu, karena itu diperlukan kebijakan resolusi bentrokan jika hal seperti ini terjadi. Makalah ini membahas macam-macam fungsi hash yang sering digunakan, metode kebijakan resolusi bila terjadi bentrokan, dan penggunaan fungsi hash untuk kriptografi.*

Kata Kunci*:hash, record, kriptografi*

1. PENDAHULUAN

Hashing adalah transformasi aritmatik sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya. Menurut bahasanya, *hash* berarti memenggal dan kemudian menggabungkan.

Hashing digunakan sebagai metode untuk menyimpan data dalam sebuah array agar penyimpanan data, pencarian data, penambahan data, dan penghapusan data dapat dilakukan dengan cepat. Ide dasarnya adalah menghitung posisi *record* yang dicari dalam array, bukan membandingkan *record* dengan isi pada array. Fungsi yang mengembalikan nilai atau kunci disebut fungsi *hash* (*hash* function) dan array yang digunakan disebut tabel *hash* (*hash* table). Secara teori, kompleksitas waktu (T(n)) dari fungsi *hash* yang ideal adalah O(1). Untuk mencapai itu setiap *record* membutuhkan suatu kunci yang unik.

Fungsi *hash* menyimpan nilai asli atau kunci pada alamat yang sama dengan nilai hashnya. Pada pencarian suatu nilai pada tabel *hash*, yang pertama dilakukan adalah menghitung nilai *hash* dari kunci atau nilai aslinya, kemudian membandingkan kunci atuau nilai asli dengan isi pada memori yang beralamat nomor hashnya. Dengan cara ini, pencarian suatu nilai dapat dilakukan dengan cepat tanpa harus memeriksa seluruh isi tabel satu per satu.

Selain digunakan pada penyimpanan data, fungsi *hash* juga digunakan pada algoritma enkripsi sidik jari digital (*fingerprint*) untuk mengautentifikasi pengirim dan penerima pesan. Sidik jari digital diperoleh dengan fungsi *hash*, kemudian nilai *hash* dan tanda pesan yang asli dikirim kepada penerima pesan. Dengan menggunakan fungsi *hash* yang sama dengan pengirim pesan, penerima pesan mentransformasikan pesan yang diterima. Nilai *hash* yang diperoleh oleh penerima pesan kemudian dibandingkan dengan nilai *hash* yang dikirim pengirim pesan. Kedua nilai *hash* harus sama, jika tidak, pasti ada masalah.

Hashing selalu merupakan fungsi satu arah. Fungsi *hash* yang ideal tidak bisa diperoleh dengan melakukan *reverse engineering* dengan menganalisa nilai *hash*. Fungsi *hash* yang ideal juga seharusnya tidak menghasilkan nilai *hash* yang sama dari beberapa nilai yang berbeda . Jika hal yang seperti ini terjadi, inilah yang disebut dengan bentrokan (*collision*). Kemungkinan terjadinya bentrokan tidak dapat dihindari seratus persen. Fungsi *hash* yang baik dapat meminimalkan kemungkinan terjadinya bentrokan.

2. MACAM-MACAM FUNGSI *HASH*

Fungsi *Hash* (dilambangkan dengan *h(k)*) bertugas untuk mengubah k (*key*) menjadi suatu nilai dalam interval [0....X], dimana "X" adalah jumlah maksimum dari *record-record* yang dapat ditampung dalam tabel. Jumlah maksimum ini bergantung pada ruang memori yang tersedia. Fungsi *Hash* yang ideal adalah mudah dihitung dan bersifat *random*, agar dapat menyebarkan semua *key*. Dengan *key* yang tersebar, berarti data dapat terdistribusi secara seragam bentrokan dapat dicegah. Sehingga kompleksitas waktu model *Hash* dapat mencapai O(1), di mana kompleksitas tersebut tidak ditemukan pada struktur model lain.

Ada beberapa macam fungsi *hash* yang relatif sederhana yang dapat digunakan dalam penyimpanan database:

1. Metode Pembagian Bersisa (*division-remainder method*)

   Jumlah lokasi memori yang tersedi dihitung, kemudian jumlah tersebut digunakan sebagai pembagi untuk membagi nilai yang asli dan menghasilkan sisa. Sisa tersebut adalah nilai hashnya.

   Secara umum, rumusnya h(k)= k mod m. Dalam hal ini m adalah jumlah lokasi memori yang tersedia pada array. Fungsi *hash* tersebut menempatkan *record* dengan kunci k pada suatu lokasi memori yang beralamat h(k).

Metode ini sering menghasilkan nilai *hash* yang sama dari dua atau lebih nilai aslinya atau disebut dengan bentrokan. Karena itu, dibutuhkan mekanisme khusus untuk menangani bentrokan yang disebut kebijakan resolusi bentrokan.

2. Melipat (*folding*)
   Metode ini membagi nilai asli ke dalam beberapa bagian, kemudian menambahkan nilai-nilai tersebut, dan mengambil beberapa angka terakhir sebagai nilai hashnya.

3. Transformasi Radiks (*radix transformation*)
   Karena nilai dalam bentuk digital, basis angka atau radiks dapat diganti sehingga menghasilkan urutan angka-angka yang berbeda. Contohnya nilai desimal (basis 10) bisa ditransformasikan kedalam heksadesimal (basis 16). Digit atas hasilnya bisa dibuang agar panjang nilai *hash* dapat seragam.

4. Pengaturan ulang digit (*digit rearrangement*)
   Metode ini mengubah urutan digit dengan pola tertentu. Contohnya mengambil digit ke tiga sampai ke enam dari nilai aslinya, kemudian membalikan urutannya dan menggunakan digit yang terurut terbalik itu sebagai nilai *hash*.

Fungsi *hash* yang bekerja dengan baik untuk penyimpanan pada database belum tentu bekerja dengan baik untuk keperluan kriptografi atau pengecekan kesalahan. Ada beberapa fungsi *hash* terkenal yang digunakan untuk keperluan kriptografi. Diantaranya adalah fungsi *hash* message-diggest, contohnya MD2, MD4, dan MD5, digunakan untuk menghasilkan nilai *hash* dari tanda tangan digital yang disebut message-diggest. Ada pula Secure *Hash* Algorithm (SHA), sebuah algoritma standar yang menghasilkan message-diggest yang lebih besar (60-bit) dan serupa dengan MD4.

3. BENTROKAN PADA FUNGSI *HASH*

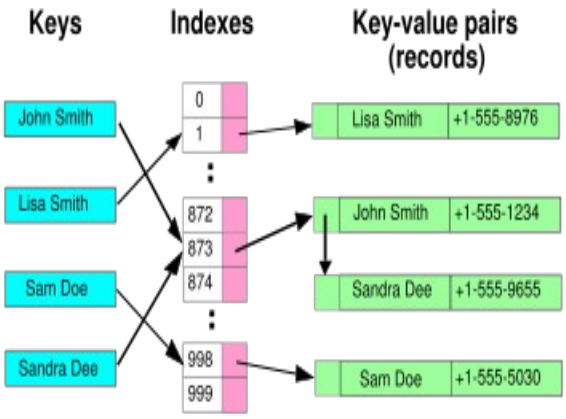
Fungsi *hash* bukan merupakan fungsi satu-ke-satu, artinya beberapa *record* yang berbeda dapat menghasilkan nilai *hash* yang sama / terjadi bentrokan. Dengan fungsi *hash* yang baik, hal seperti ini akan sangat jarang terjadi, tapi pasti akan terjadi. Jika hal seperti ini terjadi, *record-record* tersebut tidak bisa menempati lokasi yang sama.

Ada dua macam kebijakan resolusi bentrokan pada tabel *hash*, yaitu kebijakan resolusi bentrokan di luar tabel dan kebijakan resolusi bentrokan di dalam tabel. Harus diperhatikan juga teknik-teknik penempatan *record* agar mudah dicari jika dibutuhkan.

3.1. Kebijakan resolusi bentrokan di luar tabel

Artinya tabel *hash* bukan lagi menjadi *array of records*, tetapi menjadi *array of pointers*. Setiap pointer menunjuk ke senarai berkait yang berisi *record* tersebut.

Metode seperti ini dinamakan *chaining*. Dalam bentuk sederhananya berupa senarai berkait dari *record-record* yang menghasilkan nilai *hash* yang sama. Penambahan *record* dapat dilakukan dengan menambah senarai berisi *record* tersebut. Untuk pencarian pada tabel, pertama-tama dicari nilai *hash* terlebih dahulu, kemudian dilakukan pencarian dalam senarai berkait yang bersangkutan. Untuk menghapus suatu *record*, hanya menghapus senarainya saja.



Gambar 1. Resolusi bentrokan dengan *chaining*

Kelebihan dari metode *chaining* ini *chaining* ini adalah proses penghapusan yang relarif mudah dan penambahan ukuran tabel *hash* bisa ditunda untuk waktu yang lebih lama karena penurunan kinerjanya berbanding lurus meskipun seluruh lokasi pada tabel sudah penuh. Bahkan, penambahan ukuran tabel bisa saja tidak perlu dilakukan sama sekali karena penurunan kinerjanya yang linier. Misalnya, tabel yang berisi *record* sebanyak dua kali lipat kapasitas yang direkomendasikan hanya akan lebih lambat dua kali lipat dibanding yang berisi sebanyak kapasitas yang direkomendasikan.

Kekurangan dari metode *chaining* ini sama dengan kekurangan dari senarai berkait. Operasi traversal pada senarai berkait memiliki performa *cache* yang buruk.

Struktur data lain dapat digunakan sebagai pengganti senarai berkait. Misalnya dengan pohon seimbang, kompleksitas waktu terburuk bisa diturunkan menjadi O(log n) dari yang sebelumnya O(n). Namun demikian, karena setiap senarai diharapkan untuk tidak panjang, struktur data pohon ini kurang efisien kecuali tabel *hash* tersebut memang didesain untuk jumlah *record* yang banyak atau kemungkinan terjadi bentrokan sangat besar yang mungkin terjadi karena masukan memang disengaja agar terjadi bentrokan.

3.2. Kebijakan resolusi bentrokan di dalam tabel

Berbeda dengan kebijakan resolusi bentrokan di luar tabel, pada kebijakan resolusi di dalam tabel data

disimpan di dalam *hash* tabel tersebut, bukan dalam senarai berkait yang bisa bertambah terus menerus. Dengan demikian data yang disimpan tidak mungkin bisa lebih banyak daripada jumlah ruang pada tabel *hash*.

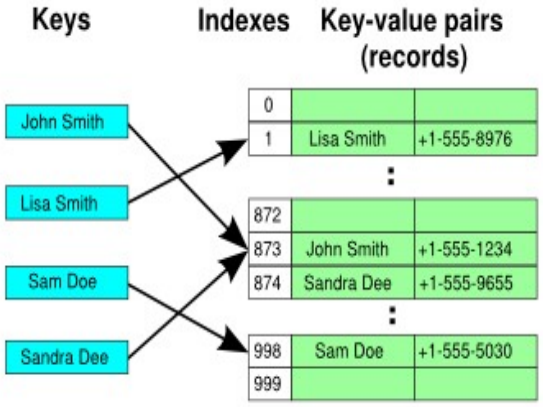
Jika suatu *record* akan dimasukkan ke dalam tabel *hash* pada lokasi sesuai nilai *hash*-nya dan ternyata lokasi tersebut sudah diisi dengan *record* lain maka harus dicari lokasi alternatif yang masih belum terisi dengan cara tertentu. Cara ini disebut *Open Addressing*.

Ada beberapa metode untuk menemukan lokasi baru yang masih kosong. Dalam proses menemukan lokasi baru ini harus menggunakan pola tertentu agar *record* yang disimpan tetap bisa dicari dengan mudah saat dibutuhkan kemudian.

Metode-metode yang sering digunakan adalah:

1. *Linear probing*

Dengan menambahkan suatu interval pada hasil yang diperoleh dari fungsi *hash* sampai ditemukan lokasi yang belum terisi. Interval yang biasa digunakan adalah 1.



Gambar 2. Resolusi bentrokan dengan *Linear Probing*

2. *Quadratic probing / squared probing*

Hampir sama dengan linear probing, hanya saja pada quadratic probing, hasil yang diperoleh dari fungsi *hash* ditambahkan dengan kuadrat dari interval yang digunakan.

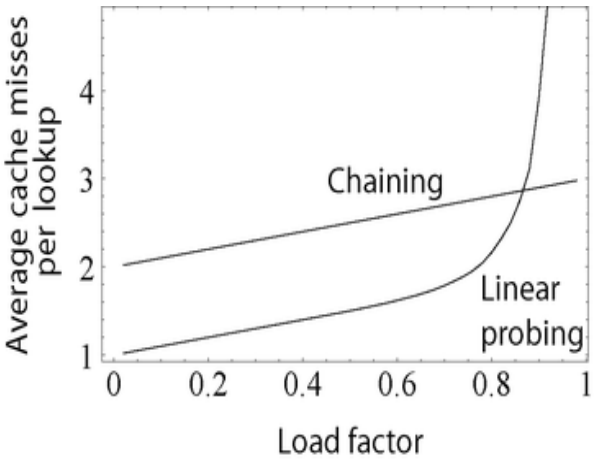
3. *Double hashing*

Pada metode *double hashing,* jika lokasi yang diperoleh dengan fungsi *hash* sudah terisi, maka dilakukan proses *hash* lagi sampai ditemukan lokasi yang belum terisi.

3.3. Perbandingan antara metode *chaining* dan *open addressing*

Keunggulan metode *chaining* dibanding *open addressing*:

1. Lebih mudah diimplementasikan dengan efektif dan hanya membutuhkan struktur data dasar.
2. Metode *chaining* tidak rawan terhadap data-data yang berkumpul di daerah tertentu. Metode *open addressing* membutuhkan algoritma *hash* yang lebih baik untuk menghindari pengumpulan data di sekitar lokasi tertentu.
3. Performa menurun secara linier. Meskipun semakin banyak *record* yang dimasukkan maka semakin panjang senarai berantai, tabel *hash* tidak akan penuh dan tidak akan menimbulkan peningkatan waktu pencarian *record* yang tiba-tiba meningkat yang terjadi bila menggunakan metode *open addressing.*
4. Jika *record* yang dimasukkan panjang, memori yang digunakan akan lebih sedikit dibandingkan dengan metode *open addressing*.



Gambar 3. Perbandingan waktu yang diperlukan untuk melakukan pencarian. Saat tabel mencapai 80% terisi, kinerja pada linear probing(*open addressing*)menurun drastis.

Untuk ukuran *record* yang kecil, keunggulan metode *open addressing* dibandingkan dengan *chaining* diantaranya

a. Ruang yang digunakan lebih efisien karena tidak perlu menyimpan pointer atau mengalokasi tempat tambahan di luar tabel *hash*.
b. Tidak ada waktu tambahan untuk pengalokasian memori karena metode *open addressing* tidak memerlukan pengalokasian memori.
c. Tidak memerlukan pointer.

Sebenarnya, penggunaan algoritma apapun pada tabel *hash* biasanya cukup cepat, dan persentase kalkulasi yang dilakukan pada tabel *hash* rendah. Penggunaan memori juga jarang berlebihan. Oleh karena itu, pada kebanyakan kasus, perbedaan antar algoritma ini tidak signifikan.

3.4. Metode-metode lain

Selain metode-metode yang sudah disebutkan di atas, ada juga beberapa metode lain.

3.4.1. *Coalesced hashing*
Gabungan dari *chaining* dan *openaddressing*. *Coalesced hashing* menghubungkan ke tabel itu sendiri. Seperti *open addressing*, metode ini memiliki keunggulan pada penggunaan tempat dan cache dibanding metode *chaining*. Seperti *chaining*, metode ini menghasilkan lokasi penyimpanan yang lebih menyebar, tetapi pada metode ini *record* yang disimpan tidak mungkin lebih banyak daripada ruang yang disediakan tabel.
3.4.2. *Perfect hashing*
Jika *record* yang akan digunakan sudah diketahui sebelumnya, dan jumlahnya tidak melebihi jumlah ruang pada tabel *hash*, *perfect hashing* bisa digunakan untuk membuat tabel *hash* yang sempurna, tanpa ada bentrokan.

3.4.3. *Probabilistic hashing*
Kemungkinan solusi paling sederhana untuk mengatasi bentrokan adalah dengan mengganti *record* yang sudah disimpan dengan *record* yang baru, atau membuang *record* yang baru akan dimasukkan. Hal ini bisa berdampak tidak ditemukannya *record* pada saat pencarian. Metode ini digunakan untuk keperluan tertentu saja.

3.4.4. *Robin Hood hashing*
Salah satu variasi dari resolusi bentrokan *double hashing*. Ide dasarnya adalah sebuah *record* yang sudah dimasukkan bisa digantikan dengan *record* yang baru jika nilai pencariannya (*probe count* – bertambah setiap menemukan termpat yang sudah terisi) lebih besar daripada nilai pencarian dari *record* yang sudah dimasukkan. Efeknya adalah mengurangi kasus terburuk waktu yang diperlukan untuk pencarian.

4. FUNGSI *HASH* KRIPTOGRAFIS

Fungsi *hash* Kriptografis adalah fungsi *hash* yang memiliki beberapa sifat keamanan tambahan sehingga dapat dipakai untuk tujuan keamanan data [1]. Umumnya digunakan untuk keperluan autentikasi dan integritas data. Fungsi *hash* adalah fungsi yang secara efisien mengubah string input dengan panjang berhingga menjadi string output dengan panjang tetap yang disebut nilai *hash*.

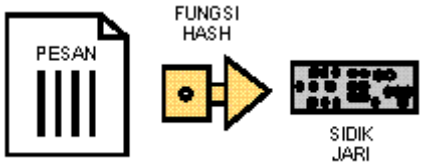
Sifat-Sifat Fungsi *Hash* Kriptografi
a. Tahan preimej (*Preimage resistant*): bila diketahui nilai *hash h* maka sulit (secara komputasi tidak layak) untuk mendapatkan *m* dimana $h = hash(m)$.
b. Tahan preimej kedua (*Second preimage resistant*): bila diketahui input m_1 maka sulit mencari input m_2 (tidak sama dengan m_1) yang menyebabkan $hash(m_1) = hash(m_2)$.
c. Tahan tumbukan (*Collision-resistant*): sulit mencari dua input berbeda m_1 dan m_2 yang menyebabkan $hash(m_1) = hash(m_2)$

4.1. Pengertian Kriptografi
Kriptografi adalah ilmu yang mempelajari bagaimana membuat suatu pesan yang dikirim pengirim dapat disampaikan kepada penerima dengan aman [2]. Selain pengertian tersebut terdapat pula pengertian ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, keabsahan data, intergritas data, serta autentikasi data [3].

Ada empat tujuan mendasar dari ilmu kriptografi ini yang juga merupakan aspek keamanan informasi yaitu :
1. Kerahasiaan, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka/mengupas informasi yang telah disandi.
2. Integritas data, adalah 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, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam data yang sebenarnya.
3. Autentikasi, adalah 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-repudiasi, atau nirpenyangkalan adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman/terciptanya suatu informasi oleh yang mengirimkan/membuat.

4.2 Sidik Jari
Kini akan dibahas mengenai keutuhan pesan saat dikirimkan. Saat pengirim pesan hendak mengirimkan pesannya, dia harus membuat sidik jari dari pesan yang akan dikirim untuk penerima pesan. Pesan (yang besarnya dapat bervariasi) yang akan di-*hash* disebut preimej, sedangkan outputnya yang memiliki ukurannya tetap, disebut nilai *hash*. Kemudian, melalui saluran komunikasi yang aman, dia mengirimkan sidik jarinya kepada penerima. Setelah penerima menerima pesan si pengirim – tidak peduli lewat saluran komunikasi yang mana – penerima kemudian juga membuat sidik jari dari pesan yang telah diterimanya dari pengirim. Kemudian kedua sidik jari dibandingkan. Jika kedua sidik jari itu identik, maka penerima dapat yakin bahwa pesan itu utuh tidak diubah-ubah sejak dibuatkan sidik jari yang diterimanya. Jika pesan sudah diubah, tentunya akan menghasilkan nilai *hash* yang berbeda.

Fungsi *hash* untuk membuat sidik jari tersebut dapat diketahui oleh siapapun, tak terkecuali, sehingga siapapun dapat memeriksa keutuhan dokumen atau pesan tertentu. Tak ada algoritma rahasia dan umumnya tak ada pula kunci rahasia.



Gambar 4. Membuat sidik jari pesan

Jaminan dari keamanan sidik jari berangkat dari kenyataan bahwa hampir tidak ada dua preimej yang memiliki nilai *hash* yang sama. Inilah yang disebut dengan sifat bebas bentrokan dari suatu fungsi *hash* yang baik. Selain itu, sangat sulit untuk membuat suatu preimej jika hanya diketahui *hash*-valuenya saja.

Contoh algoritma fungsi *hash* satu arah adalah MD-5 dan SHA. *Message authentication code* (MAC) adalah salah satu variasi dari fungsi *hash* satu arah, hanya saja selain preimej, sebuah kunci rahasia juga menjadi input bagi fungsi MAC.

4.3. Tanda Tangan Digital

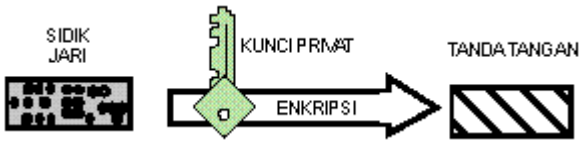
Penerima pesan memang dapat merasa yakin bahwa sidik jari yang datang bersama pesan yang diterimanya memang berkorelasi. Namun belum tentu pesan yang diterimanya adalah pesan sebenarnya yang dikirim. Bisa saja pesan tersebut sudah diterima orang lain saat melalui saluran komunikasi yang tidak aman, yang kemudian mengganti isi pesan dan membuat sidik jari lain seolah-olah itu adalah pesan yang sebenarnya..

Untuk mencegah pemalsuan, pengirim membubuhkan tanda tangannya pada pesan tersebut. Dalam dunia elektronik, pengirim membubuhkan tanda tangan digitalnya pada pesan yang akan dikirimkan agar penerima dapat merasa yakin bahwa pesan itu memang yang sebenarnya.

Sifat yang diinginkan dari tanda tangan digital diantaranya adalah:
a. Tanda tangan itu asli (otentik), tidak mudah ditulis/ditiru oleh orang lain. Pesan dan tanda tangan pesan tersebut juga dapat menjadi barang bukti, sehingga penandatangan tak bisa menyangkal bahwa dulu ia tidak pernah menandatanganinya.
b. Tanda tangan itu hanya sah untuk dokumen (pesan) itu saja. Tanda tangan itu tidak bisa dipindahkan dari suatu dokumen ke dokumen lainnya. Ini juga berarti bahwa jika dokumen itu diubah, maka tanda tangan digital dari pesan tersebut tidak lagi sah.
c. Tanda tangan itu dapat diperiksa dengan mudah.
d. Tanda tangan itu dapat diperiksa oleh pihak-pihak yang belum pernah bertemu dengan penandatangan.
e. Tanda tangan itu juga sah untuk kopi dari dokumen yang sama persis.

Meskipun ada banyak skenario, ada baiknya kita perhatikan salah satu skenario yang cukup umum dalam penggunaan tanda tangan digital. Tanda tangan digital memanfaatkan fungsi *hash* satu arah untuk menjamin bahwa tanda tangan itu hanya berlaku untuk dokumen yang bersangkutan saja. Bukan dokumen tersebut secara keseluruhan yang ditandatangani, namun biasanya yang ditandatangani adalah sidik jari dari dokumen itu beserta *timestamp*-nya dengan menggunakan kunci privat. *Timestamp* berguna untuk menentukan waktu pengesahan dokumen.

Keabsahan tanda tangan digital itu dapat diperiksa oleh penerima. Pertama-tama pesan yang diterima dibuat sidik jarinya. Kemudian tanda tangan digital didekripsi untuk mendapatkan sidik jari yang asli. Bandingkan kedua sidik jari tersebut. Jika kedua sidik jari tersebut sama, maka dapat diyakini bahwa pesan tersebut ditandatangani pengirim yang sebenarnya.



Gambar 5. Pembuatan tanda tangan digital

4.3 Sertifikat Digital

Pengiriman pesan dengan tanda tangan digital dapat dilakukan jika pengirim pesan dan penerima pesan sudah saling mengenal dan mengetahui kunci untuk enkripsi dan dekripsi pesan.

Jika pengirim pesan dan penerima pesan belum pernah mengenal atau tidak saling mengetahui kunci untuk enkripsi dan dekripsi, perlu adanya pihak ketiga. Pihak ketiga ini diasumsikan telah memiliki saluran komunikasi yang aman dengan pengirim pesan dan penerima pesan. Pengirim pesan menggunakan saluran ini untuk mengirim pesan kepada penerima. Skenario ini tetap membutuhkan kunci-kunci kriptografi lagi (baik itu kunci simetris ataupun kunci asimetris) untuk pengamanan saluran komunikasi antara pihak ketiga dengan pengirim atau penerima pesan.

Masalah di atas dapat dipecahkan dengan penggunaan sertifikat digital. Pihak ketiga tidak lagi setiap saat menjadi penukar kunci, namun cukup menandatangani kunci publik milik setiap orang di jaringan tersebut. Sebenarnya dalam sertifikat tersebut tak hanya berisi kunci publik, namun dapat berisi pula informasi penting lainnya mengenai jati diri pemilik kunci publik, seperti misalnya nama, alamat, pekerjaan,

jabatan, perusahaan dan bahkan *hash* dari suatu informasi rahasia. Semua orang mempercayai otoritas pihak ketiga dalam memberikan tanda tangan, sehingga orang-orang dalam jaringan itu merasa aman menggunakan kunci publik yang telah ditandatanganinya.

Jika seseorang berhasil mencuri sertifikat digital yang dipertukarkan, serta menggantinya dengan sertifikat digital milik dirinya sendiri, dapat segera terlihat bahwa sertifikat digital yang diterima bukan 'lawan bicara' yang semestinya.

Serangan terhadap sistem yang memiliki pengamanan dengan sertifikat digital sulit dilakukan. Secara teoritis keunggulan dari tanda tangan digital adalah kemampuan untuk melakukan proses otentikasi secara *off-line*. Pemeriksa cukup memiliki kunci publik dari OS utama untuk mengetahui sah-tidaknya kunci publik dari lawan bicaranya. Selain itu untuk meningkatkan keamanan, kunci publik OS utama bisa saja diintegrasikan dalam program aplikasi. Namun kenyataannya, karena ada kemungkinan sertifikat digital tersebut hilang, tercuri atau identitas pemilik sertifikat berubah (perubahan alamat surat elektronik atau nomor KTP misalnya), maka sertifikat digital perlu diperiksa keabsahannya dengan melihat daftar sertifikat terbatalkan (*certificate revocation list*) yang disimpan oleh OS.

4.4. Fungsi *hash* MD5

MD5 adalah salah satu fungsi *hash* yang digunakan untuk keperluan kriptografi.

Dalam kriptografi, MD5 (*Message-Digest algortihm 5*) ialah fungsi *hash* kriptografik yang digunakan secara luas dengan nilai *hash* 128-bit [4]. Pada standard 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, salah satu pembuat algoritma RSA, 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.

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

5. KESIMPULAN

Fungsi *hash* adalah fungsi yang bisa digunakan untuk berbagai keperluan. Fungsi *hash* yang baik digunakan untuk penyimpanan pada database belum tentu baik juga untuk keperluan kriptografi.

Jika dalam database fungsi *hash* digunakan untuk memudahkan penyimpanan *record* dalam tabel *hash*, pada kriptografi, fungsi *hash* digunakan untuk memastikan kebenaran pesan yang dikirim dengan cara membandingkan nilai-nilai *hash* yang diperoleh.

Dalam sebuah tabel *hash*, bentrokan adalah hal yang hampir tidak dapat dihindari, terutama jika *record* yang dimasukkan pada tabel *hash* relatif banyak.

DAFTAR REFERENSI

[1]http://www.geocities.com/amwibowo/resource/komparasi/komparasi.html#daftar, Senin 31 Desember 2007, 10.50 WIB
[2]http://id.wikipedia.org/wiki/Kriptografi, Jumat 28 Desember 2007, 10.22 WIB
[3]http://id.wikipedia.org/wiki/MD5, Jumat 28 Desember 2007, 10.24