MAKALAH IF2153
MATEMATIKA DISKRIT

PENGGUNAAN FUNGSI HASH DALAM KRIPTOGRAFI

Oleh:
Ricky Gilbert Fernando
13505077

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2007

DAFTAR ISI

PENGGUNAAN FUNGSI HASH DALAM KRIPTOGRAFI

Ricky Gilbert Fernando – 13505077
Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl Ganesha 10, Bandung
Email: if15077@students.if.itb.ac.id

Abstrak

Makalah ini menjelaskan tentang kegunaan fungsi hash dalam kriptografi. Keamanan data-data sensitif, seperti data-data perusahaan atau e-mail kita penting artinya untuk menjaga privasi seseorang. Kriptografi merupakan salah satu cara untuk menyimpan sebuah dokumen penting dimana dokumen yang telah terenkripsi hanya bisa bisa dibuka dengan 'kunci' tersendiri. Kriptografi memungkinkan keamanan sebuah data sebab peluang untuk mendapatkan kunci yang sesuai saja (untuk kasus kunci 64bit) membutuhkan 2^{64} = 18,446,744,073,709,551,616 percobaan dengan cara *brute force*. Dengan waktu selama itu, tidak ada orang yang berusaha untuk menjebolnya dengan cara ini. Lagipula cara ini kriptografi MD5 dan SHA belum ditemukan kelemahannya hingga saat ini.

Makalah ini terdiri dari beberapa bagian yang menjelaskan fungsi hash sebagai alat dalam proses kriptografi dokumen-dokumen sensitif. Fungsi hash merupakan sebuah alat untuk enkripsi data yang umumnya digunakan secara luas oleh publik dalam proteksi data.

Bagian awal makalah ini akan menjelaskan tentang arti dari fungsi hash dan kriptografi itu sendiri. Selain itu, terdapat pula fungsi hash yang khusus digunakan untuk proses kriptografi. Setelah bagian itu, diterangkan sebagian algoritma MD5 dan SHA yang merupakan bagian dari fungsi hash. Terdapat penjelasan tentang SHA-0, SHA-1 dan varian SHA lainnya yang lebih panjang. Setelah itu, terdapat penggunaan fungsi hash dalam kriptografi, seperti proses look-up tabel hash, error correction codes, checksum dan fungsi satu-arah. Di bagian akhir disediakan contoh-contoh kasus yang membahas hal-hal di atas dan diberikan source-code dalam membentuk fungsi hash dah algoritma SHA-1.

***Kata Kunci**: fungsi hash, kriptografi, algoritma, SHA, MD5, error correction codes, checksum, penyerangan, tanda tangan digital, sertifikat digital, tanda tangan pesan ganda.*

Pendahuluan

Makalah ini akan membahas fungsi hash yang sering digunakan dalam proses kriptografi dokumen-dokumen. Fungsi hash merupakan salah satu proses enkripsi cepat yang umumnya digunakan untuk kepentingan publik.

Apakah Fungsi Hash Itu?

Fungsi hash H adalah transformasi yang mengambil input dengan ukuran m dan mengembalikan sebuah string berukuran tetap yang disebut sebagai nilai hash h (di mana, h = H(m)). Fungsi hash sederhana ini memiliki berbagai jenis kegunaan komputasi, tetapi ketika digunakan untuk masalah kriptografi, fungsi hash selalu ditambahkan dengan sejumlah properti tambahan.

Yang dibutuhkan untuk fungsi kriptografi hash, yaitu:

1. input dengan panjang sembarang
2. hasilnya mempunyai keluaran dengan panjang yang *fixed*
3. H(x) umumnya mudah dikalkulasi untuk sembarang nilai x
4. H(x) adalah satu-arah
5. H(x) tidak pernah bermasalah dengan yang lain

Fungsi hash H merupakan fungsi satu-arah sebab sulit untuk dibalikkan yang berarti untuk nilai fungsi hash h, kita sulit menemukan nilai input x yang memenuhi persamaan H(x) = h

Selain itu, jika diberikan sebuah pesan x untuk dikomputasi dalam fungsi hash H, kita akan kesulitan dalam mencari pesan y yang akan menyamai nilai dari H(x) (berarti sulit menemukan nilai H(x) = H(y)) yang dapat kita katakan bahwa fungsi hash x tidak mungkin bertabrakan dengan fungsi hash y. Hal ini menyebabkan fungsi hash H adalah sebuah fungsi yang sulit untuk menemukan kesamaan antara 2 pesan (*strongly collision-free*).

Nilai dari fungsi hash menyatakan sebuah pesan atau dokumen yang lebih panjang yang berasal dari proses komputasi. Hal ini menarik sebab dengan fungsi hash, kita dapat membuat sebuah *digital fingerprint* untuk sebuah dokumen. Contoh yang paling terkenal dari fungsi hash adalah MD2, MD5 dan SHA.

Mungkin penggunaan yang umum dari fungsi kriptografi hash adalah pembuatan digital signatures. Karena fungsi hash umumnya lebih cepat daripada algoritma digital signature lainnya, fungsi hash lebih sering digunakan untuk mendapatkan nilai fungsi hash dengan mengkalkulasikan signature yang menghasilkan sebuah nilai hash yang lebih kecil daripada dokumen itu sendiri. Selain itu, publik dapat memberikan sebuah saran atau pendapat tanpa membeberkan isi dari pendapat yang terdapat di dalamnya. Cara ini digunakan dalam memberikan tanggal pada sebuah dokumen dimana dengan menggunakan fungsi hash, setiap orang dapat memberikan tanggal pada dokumen tanpa memperlihatkan isi dari dokumennya pada saat proses pemberian tanggal.

Karena fungsi hash erat berkaitan dengan kriptografi, bagian selanjutnya akan membahas sedikit tentang kriptografi.

Apakah Kriptografi Itu?

Kriptografi, secara umum adalah ilmu dan seni untuk menjaga kerahasiaan berita (Oleh Bruce Schneier - *Applied Cryptography*). Selain pengertian tersebut terdapat pula pengertian ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, keabsahan data, integritas data dan autentifikasi dat (Oleh A. Menezes, P. van Oorschot and S. Vanstone - *Handbook of Applied Cryptography*). Tapi tidak semua aspek keamanan informasi ditangani oleh kriptografi.

Ada empat tujuan mendasar dari ilmu kriptografi ini yang juga merupakan aspek keamanan informasi yaitu :

- 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.
- 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.
- 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.
- Non-repudiasi atau nirpenyangkalan adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman/terciptanya suatu informasi oleh yang mengirimkan / membuat

Pembakuan penulisan pada kriptografi dapat ditulis dalam bahasa matematika. Fungsi-fungsi yang mendasar dalam kriptografi adalah enkripsi dan dekripsi. Enkripsi adalah proses mengubah suatu pesan asli (*plaintext*) menjadi suatu pesan dalam bahasa sandi (*ciphertext*).

$C = E(M)$

dimana

M = pesan asli
E = proses enkripsi
C = pesan dalam bahasa sandi (untuk ringkasnya disebut sandi)

Sedangkan dekripsi adalah proses mengubah pesan dalam suatu bahasa sandi menjadi pesan asli kembali.

$M = D(C)$

D = proses dekripsi

Umumnya, selain menggunakan fungsi tertentu dalam melakukan enkripsi dan dekripsi, seringkali fungsi itu diberi parameter tambahan yang disebut dengan istilah kunci.

Secara garis besar kriptografi terdiri dari:

1. Algoritma Sandi
   1.1. Algoritma sandi kunci-simetris
      1.1.1. Block-Cipher
      1.1.2. Stream-Cipher
      1.1.3. Algoritma-Algoritma sanndi kunci-simetris
   1.2. Algoritma sandi kunci-asimetris
      1.2.1. Fungsi enkripsi dan dekripsi algoritma sandi kunci-asimetris
      1.2.2. Algoritma-algoritma sandi kunci asimetris
2. Fungsi Hash Kriptografis
   2.1. Sifat-sifat fungsi hash kriptografis
   2.2. Algoritma-Algoritma fungsi hash kriptografis

Dalam makalah ini, akan dibahas bagian fungsi hash kriptografis.

Fungsi Hash Kriptografis

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. 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 kriptografis:

1. *Preimage resistant*: bila diketahui nilai hash h maka sulit (secara komputasi tidak layak) untuk mendapatkan m dimana $h = \text{hash}(m)$.
2. *Second preimage resistant*: bila diketahui input m_1 maka sulit mencari input m_2 (tidak sama dengan m_1) yang menyebabkan $\text{hash}(m_1) = \text{hash}(m_2)$.
3. *Collision-resistant*: sulit mencari dua input berbeda m_1 dan m_2 yang menyebabkan $\text{hash}(m_1) = \text{hash}(m_2)$

Beberapa contoh algoritma fungsi hash kriptografi:

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

Algoritma MD5

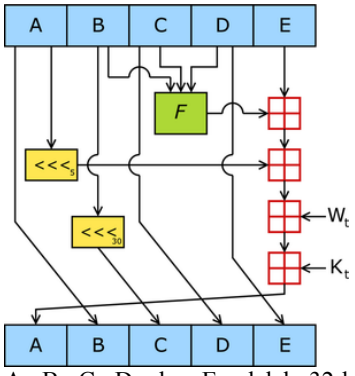
Algoritma MD5 adalah fungsi *hash* satu arah yang dibuat oleh Ron Rivest dan merupakan pengembangan dari algoritma MD4. Algoritma MD5 menerima masukan berupa pesan dengan ukuran sembarang dan menghasilkan sebuah *message digest* dengan panjang 128 bit. Langkah-langkah pembuatan *message digest* adalah sebagai berikut:

1. Penambahan bit-bit penyangga(*padding bits*).
   Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Bit-bit pengganjal terdiri dari sebuah bit 1 yang diikuti sejumlah bit 0.
2. Penambahan nilai panjang pesan semula.
   Pesan yang sudah ditambahi oleh bit pengganjal, diberi tambahan lagi sepanjang 64 bit agar panjang pesan menjadi 512-bit.
3. Inisialisasi *buffer* MD.
   MD5 memerlukan 4 buah penyangga yang masing-masing berukuran 32 bit. Keempat penyangga menampung hasil antara dan hasil akhir dari proses pembuatan *Message Digest* ini.
   Setiap penyangga diinisialisasi dengan nilai-nilai berikut (dalam notasi HEX) :
   A = 01234567
   B = 89ABCDEF
   C = FEDCBA98
   D = 76543210
4. Pengolahan pesan pada blok berukuran 512-bit.
   Pesan dibagi menjadi L buah blok, dimana masing-masing blok berukuran 512 bit. Setiap blok diproses bersama penyangga *MD* menjadi keluaran 128-bit. Proses ini terdiri dari 4 putaran dan masing-masing putaran melakukan operasi dasar *MD5* sebanyak 16 kali.

Algoritma SHA

Fungsi hash SHA (Secure Hash Algorithm), antara lain SHA-1, SHA-224, SHA-256, SHA-384 dan SHA-512 adalah lima fungsi hash kriptografis yang dibuat oleh National Security Agency (NSA) dan dinyatakan sebagai standar keamananan pemerintah USA. SHA-1 digunakan dalam proses sekuriti banyak program, seperti TLS and SSL, PGP, SSH, S/MIME dan IPsec. Fungsi ini dianggap sebagai pengganti fungsi hash MD5 yang lebih sering digunakan public. Tetapi pada kenyataannya kedua fungsi ini tetap digunakan sebagai proses enkripsi. Sedangkan 4 varian lainnya (SHA-224, SHA-256, SHA-384 dan SHA-512) biasanya disebut sebagai SHA-2. Hingga saat ini belum ada serangan terhadap SHA-2, tetapi karena kemiripannya dengan SHA-1, para peneliti khawatir dan mengembangkan kandidat baru penggantinya.

SHA-0 dan SHA-1



A, B, C, D, dan E adalah 32-bit kata. F adalah fungsi non-linear yang bervariasi. ; menyatakan rotasi ke kiri sebanyak n tempat di mana n berbeda untuk tiap operasi. menyatakan modulo tambahan 2^{32}. K_t adalah konstanta.

Contoh SHA-1:
SHA1("The quick brown fox jumps over the lazy dog") = 2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12

Bahkan dengan ada sedikit perubahan kecil, nilai dari SHA-1 akan berubah.
Contoh:
SHA1("The quick brown fox jumps over the lazy cog") = de9f2c7f d25e1b3a fad3e85a 0bd17d9b 100db4b3

Pada bagian belakang makalah, terdapat algoritma dari SHA-1

Longer Variants

Keluarga SHA dengan hasil yang lebih panjang adalah SHA-256, SHA-384 dan SHA-512.

SHA-224 mirip dengan SHA-256, kecuali:
- Nilai variabel inisial dari h0 ke h7 berbeda
- Hasilnya dibentuk dengan menghilangkan h7

SHA-512 mirip dari segi struktur, kecuali:
- Setiap nilai sepanjang 64.
- Terdapat 80 putaran, bukan 64
- Nilai inisial dan konstanta tambahan ditetapkan menjadi 64bit
- Jumlah perpindahan dan rotasi yang digunakan berbeda

SHA-384 mirip dengan SHA-512, kecuali:
- Nilai inisial h0 sampai h7 berbeda
- Hasilnya dibentuk dengan menghilangkan h6 dan h7.

Contoh:
SHA256("abc") = ba7816bf 8f01cfea 414140de 5dae2223 b00361a3 96177a9c b410ff61 f20015ad

SHA256("abcdbcdecdefdefgefghfghighijhijk ijkljklmklmnlmnomnopnopq") = 248d6a61 d20638b8 e5c02693 0c3e6039 a33ce459 64ff2167 f6ecedd4 19db06c1

SHA256("The quick brown fox jumps over the lazy dog") = d7a8fbb3 07d78094 69ca9abc b0082e4f 8d5651e4 6d3cdb76 2d02d0bf 37c9e592

Fungsi Hash untuk Proses Look-up Tabel Hash

Sebuah fungsi hash untuk look-up tabel hash seharusnya cepat dan menciptakan sesedikit mungkin tumbukan. Jika Anda mengetahui kuncinya, Anda dapat *hashing* sebelum memilih fungsi hashnya dan menghasilkan 0 tumbukan, yang disebut sebagai perfect-hashing. Di samping itu, yang terbaik yang dapat dilakukan adalah memetakan setiap kunci pada setiap nilai hash yang mungkin dan memastikan bahwa tidak ada lebih dari satu kunci yang memetakan ke nilai hash yang sama.

Referensi untuk fungsi hash ini adalah "The Art of Computer Programming" edisi 3 "Sorting and Searching" bab 6.4 karya Knuth. Dia menyarankan sebuah fungsi hash

```
for (hash=len; len--;)
{
 hash = ((hash<<5)^(hash>>27))^*key++;
}
hash = hash % prime;
```

Sayangnya, fungsi hash tersebut kurang bagus. Masalahnya adalah pencampuran tiap karakter. Fungsi tersebut hanya merotasikan bit, tetapi benar-benar mencampur semuanya. Setiap bit input hanya mempengaruhi 1 bit hash hingga proses % terakhir. Jika dua input menghasilkan nilai bit hash yang sama, mereka akan saling menghilangkan. Selain itu, proses % bisa berlangsung sangat lambat (230 lebih lambat daripada proses penjumlahan).

Pada bagian belakang disediakan source code dalam membentuk fungsi hash.

Error Correction Codes

Error correction codes adalah *checksum* dari panjang m+2d+1 yang menghasilkan tidak ada suatu bilangan selain d yang dapat mengubah nilainya. Jika d atau nilai yang lebih kecil berganti, *checksum* hasil dan teks yang telah diubah bisa digunakan untuk mendeduksikan nilai m yang sebenarnya. Nilai dari *checksum* bisa berada antara nilai d. Jika sebaliknya, maka tidak berhasil. Sebenarnya sangat mudah untuk menghasilkan 2 dokumen dengan *checksum* yang sama.

Error correction codes telah menjadi lebih baik. Kode-kode dari Mc'Eliece's Turbo Codes dan David MacKay's Competing

Research on Gallager Codes bisa menyederhanakan kode rumit seperti, Goppa, Viterbi dan Reed Solomon.

Checksums, untuk Mengidentifikasi Dokumen

Pada bagian ini, ada sebuah kasus. Ada dua buah dokumen yang seharusnya sama, tapi kita tidak yakin. Salah satu dari dua dokumen tersebut tidak dapat diakses, sedangkan yang lain adalah versi remote atau versi sebelumnya sehingga tidak dapat dibandingkan secara langsung.

Solusinya adalah kita harus mendapatkan *checksum* dari kedua dokumen. Jika *checksum* keduanya sama, kita yakin bahwa kedua dokumennya sama. Sebenarnya sangat jarang sekali mendapatkan dua dokumen mempunyai *checksum* yang sama secara kebetulan (dengan kemungkinan 2^{-128} atau lebih kecil). Di sini kita mengasumsikan bahwa tidak ada seorangpun yang berusaha untuk menghasilkan sebuah dokumen dengan *checksum* yang sama. Asumsi yang sama digunakan untuk error correction codes yang lebih ampuh daripada *checksum*.

Fungsi Satu-Arah (Checksum Kriptografis)

Fungsi ini seperti checksum, tapi fungsi ini didesain untuk membingungkan pencuri data. Cara ini dianggap telah gagal jika seseorang berhasil menemukan nilai hash yang sama. Jika nilai hash mempunyai n bit, dibutuhkan 2^n kali untuk menemukan kunci yang sama dengan nilai hash dan sqrt(2^n) kali untuk menemukan dua kunci yang memetakan beberapa nilai hash.

Jenis Penyerangan

Di bawah ini terdapat sejumlah nama yang memudahkan identifikasi setiap orang yang terlibat dalam kasus-kasus nanti.

Kode & nama	**Penjelasan**
A: Anto	Pihak pertama
B: Badu	Pihak kedua
C: Chandra	Pihak ketiga
E: Edi	Pihak penyadap informasi yang tidak diperuntukkan kepadanya (*eavesdropper*)
M: Maman	Pihak yang tidak hanya menyadap informasi, namun juga mengubah informasi yang disadap (*malacious person*)
T: Tari, Tata, Tania	Pihak yang dipercaya oleh pihak pertama, kedua dan ketiga (*trusted person*)

Dijelaskan beberapa macam penyerangan terhadap pesan yang sudah dienkripsi:

1. *Ciphertext only attack*, penyerang hanya mendapatkan pesan yang sudah tersandikan saja.
2. *Known plaintext attack*, dimana penyerang selain mendapatkan sandi, juga mendapatkan pesan asli. Terkadang disebut pula *clear-text attack*.
3. *Choosen plaintext attack*, sama dengan *known plaintext attack*, namun penyerang bahkan dapat memilih penggalan mana dari pesan asli yang akan disandikan.

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* [DHMM 96]: 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* [DHMM 96]: Penyerang – misalnya Maman – bisa menyamar

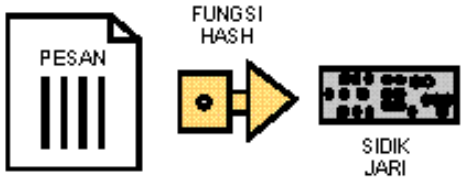
menjadi Anto. Semua orang dibuat percaya bahwa Maman adalah Anto. Penyerang berusaha meyakinkan pihak-pihak lain bahwa tak ada salah dengan komunikasi yang dilakukan, padahal komunikasi itu dilakukan dengan sang penipu/penyerang. Contohnya jika orang memasukkan PIN ke dalam mesin ATM palsu – yang benar-benar 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* [Schn 96]: Jika *spoofing* terkadang hanya menipu satu pihak, maka dalam skenario ini, saat Anto hendak berkomunikasi dengan Badu, Maman di mata Anto seolah-olah adalah Badu, dan Maman dapat pula menipu Badu sehingga Maman seolah-olah adalah Anto. Maman dapat berkuasa penuh atas jalur komunikas ini, dan bisa membuat berita fitnah.

Contoh Fungsi Hash Satu-Arah

Kini akan dibahas mengenai keutuhan pesan saat dikirimkan. Bagaimana jika Anto mengirimkan surat pembayaran kepada Badu sebesar 1 juta rupiah, namun di tengah jalan Maman (yang ternyata berhasil membobol sandi entah dengan cara apa) membubuhkan angka 0 lagi dibelakangnya sehingga menjadi 10 juta rupiah? Di mata Tari, pesan tersebut harus utuh, tidak diubah-ubah oleh siapapun, bahkan bukan hanya oleh Maman, namun juga termasuk oleh Anto, Badu dan gangguan pada transmisi pesan (*noise*). Hal ini dapat dilakukan dengan fungsi *hash* satu arah (*one-way hash function*), yang terkadang disebut sidik jari (*fingerprint*), *hash*, *message integrity check*, atau *manipulation detection code*.

Saat Anto hendak mengirimkan pesannya, dia harus membuat sidik jari dari pesan yang akan dikirim untuk Badu. Pesan (yang besarnya dapat bervariasi) yang akan di-*hash* disebut *pre-image,* sedangkan outputnya yang memiliki ukurannya tetap, disebut *hash-value* (nilai *hash*). Kemudian, melalui saluran komunikasi yang aman, dia mengirimkan sidik jarinya kepada Badu. Setelah Badu menerima pesan si Anto – tidak peduli lewat saluran komunikasi yang mana – Badu kemudian juga membuat sidik jari dari pesan yang telah diterimanya dari Anto. Kemudian Badu membandingkan sidik jari yang dibuatnya dengan sidik jari yang diterimanya dari Anto. Jika kedua sidik jari itu identik, maka Badu dapat yakin bahwa pesan itu utuh tidak diubah-ubah sejak dibuatkan sidik jari yang diterima Badu. Jika pesan pembayaran 1 juta rupiah itu diubah menjadi 10 juta rupiah, tentunya akan menghasilkan nilai *hash* yang berbeda.



Gambar 1. Membuat sidik jari pesan

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.

Jaminan dari keamanan sidik jari berangkat dari kenyataan bahwa hampir tidak ada dua pre-image yang memiliki *hash*-value yang sama. Inilah yang disebut dengan sifat *collision free* dari suatu fungsi *hash* yang baik. Selain itu, sangat sulit untuk membuat suatu pre-image 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 *pre-image*, sebuah kunci rahasia juga menjadi input bagi fungsi MAC.

Contoh Tanda Tangan Digital

Badu memang dapat merasa yakin bahwa sidik jari yang datang bersama pesan yang diterimanya memang berkorelasi. Namun

bagaimana Badu dapat merasa yakin bahwa pesan itu berasal dari Anto? Bisa saja saat dikirimkan oleh Anto melalui saluran komunikasi yang tidak aman, pesan tersebut diambil oleh Maman. Maman kemudian mengganti isi pesan tadi, dan membuat lagi sidik jari dari pesan yang baru diubahnya itu. Lalu, Maman mengirimkan lagi pesan beserta sidik jarinya itu kepada Badu, seolah-oleh dari Anto.

Untuk mencegah pemalsuan, Anto membubuhkan tanda tangannya pada pesan tersebut. Dalam dunia elektronik, Anto membubuhkan tanda tangan digitalnya pada pesan yang akan dikirimkan untuk Badu sehingga Badu dapat merasa yakin bahwa pesan itu memang dikirim oleh Anto.

Sifat yang diinginkan dari tanda tangan digital diantaranya adalah:

1. 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.
2. 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.
3. Tanda tangan itu dapat diperiksa dengan mudah.
4. Tanda tangan itu dapat diperiksa oleh pihak-pihak yang belum pernah bertemu dengan penandatangan.
5. Tanda tangan itu juga sah untuk kopi dari dokumen yang sama persis.

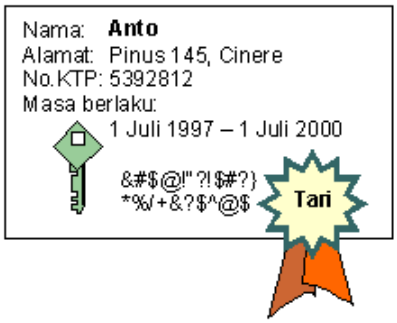
Meskipun ada banyak skenario, ada baiknya kita perhatikan salah satu skenario yang cukup umum dalam penggunaan tanda tangan digital. Tanda tangan digital memanfaatkan fungsi *hash* satu arah untuk menjamin bahwa tanda tangan itu hanya berlaku untuk dokumen yang bersangkutan saja. Bukan dokumen tersebut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

Gambar 2. Pembuatan tanda tangan digital

Keabsahan tanda tangan digital itu dapat diperiksa oleh Badu. Pertama-tama Badu membuat lagi sidik jari dari pesan yang diterimanya. Lalu Badu mendekripsi tanda tangan digital Anto untuk mendapatkan sidik jari yang asli. Badu lantas membandingkan kedua sidik jari tersebut. Jka kedua sidik jari tersebut sama, maka dapat diyakini bahwa pesan tersebut ditandatangani oleh Anto.

Contoh Sertifikat Digital

Masalah di atas dapat dipecahkan dengan penggunaan sertifikat digital. Tari tidak lagi setiap saat menjadi penukar kunci, namun Tari 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 Tari dalam memberikan tanda tangan, sehingga orang-orang dalam jaringan itu merasa aman menggunakan kunci publik yang telah ditandatangani Tari.



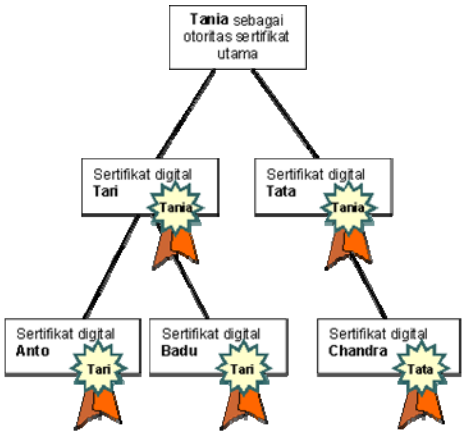
Gambar 3. Contoh sertifikat digital

Jika Maman berhasil mencuri sertifikat digital yang dipertukarkan antara Anto dan Badu, serta menggantinya dengan sertifikat digital milik dirinya sendiri, maka Anto dan Badu dapat segera melihat bahwa sertifikat digital yang diterimanya bukan 'lawan bicara' yang semestinya.

Bagaimana jika Chandra – yang berada di luar jaringan Tari – hendak berkomunikasi dengan Anto? Chandra memiliki juga sertifikat, tetapi tidak ditandatangani oleh Tari, melainkan oleh Tata, seseorang yang dipercaya dalam jaringan tempat Chandra berada. Tari dan Tata adalah otoritas sertifikat (*certificate authority*), yaitu pihak-pihak yang berwenang memberikan sertifikat. Namun Anto tidak mengenal dan tidak mempercayai Tata. Masalah ini dapat diselesaikan jika ada otoritas sertifikat (OS) yang kedudukannya lebih tinggi dari Tata dan Tari – katakanlah Tania. Tania memberikan pengesahan kepada Tata dan Tari. Jadi ada hirarki dari sertifikat digital. Jika Tania berada pada kedudukan hirarki yang paling tinggi, maka Tania disebut otoritas sertifikat utama (*root certificate authority*).

Anto mempercayai tanda tangan Tari. Namun karena Tari sendiri keberadaannya disahkan oleh Tania, tentunya Anto harus mengakui otoritas Tania. Jika Tania memberikan pengesahan kepada OS lain dibawahnya, seperti Tata, maka dengan merunut struktur hirarki percabangan OS, Anto dapat memeriksa kebenaran sertifikat digital milik Chandra yang disahkan oleh Tata.

Serangan terhadap sistem yang memiliki pengamanan dengan sertifikat digital sulit dilakukan. Jelas Edi tidak mendapatkan apa-apa kalaupun ia memainkan ulang percakapan antara Anto dan Chandra. Edi membutuhkan kunci privat untuk bisa membuka pesan-pesan yang dipertukarkan, padahal kunci privat itu tidak ada di dalam sertifikat digital.



Gambar 4. Contoh hirarki otoritas sertifikat digital

Penukaran sertifikat digital Chandra dengan sertifikat digital Maman akan segera diketahui, karena sertifikat digital itu pasti berbeda. Sedangkan jika sertifikat yang dipertukarkan antara Chandra dan Anto tidak diganti, tetapi yang diganti oleh Maman adalah pesan yang dipertukarkan, maka tentu ada ketidakcocokan dalam pemeriksaan tanda tangan digital.

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.

Contoh Tanda Tangan Pesan Ganda

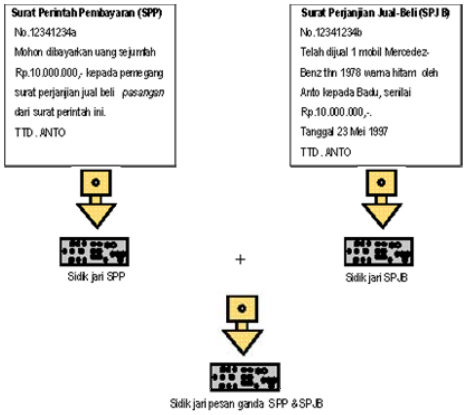
Andaikan Anto membuat perjanjian jual-beli dengan Badu. Untuk masalah pembayaran, Anto menginstruksikan bank untuk memberikan kepada Badu sejumlah uang sesuai dengan perjanjian jual-beli, namun Anto tidak ingin agar bank mengetahui isi perjanjian jual-beli itu.

1. Anto membuat sidik jari dari SPP (yaitu *Hash*(SPP)) dan sidik jari SPJB (yakni *Hash*(SPJB)).
2. Kemudian, Anto membuat sebuah sidik jari baru dari gabungan kedua sidik jari sebelumnya (*Hash* ((*Hash*(SPP) + *Hash*(SPJB))). Hasil *hash* tersebut dinamakan sidik jari pesan ganda SPP & SPJB.
3. Anto menyerahkan surat perjanjian jual belinya kepada Badu. Selain itu Anto juga menyerahkan surat perintah pembayaran beserta sidik jari pesan ganda SPP & SPJB kepada bank.
4. Saat Badu ingin mengambil uang di bank, Badu membuat sidik jari dari surat perjanjian jual beli (SPJB). Badu menyerahkan sidik jari SPJB kepada bank.
5. Bank membuat sidik jari dari surat perintah pembayaran (SPP).
6. Bank menggabungkan sidik jari SPP dengan sidik jari SPJB yang diterimanya dari Badu, kemudian meng-*hash*-nya sehingga dihasilkan sidik jari pesan ganda SPP & SPJB.
7. Jika sidik jari pesan ganda SPP & SPJB yang baru dibuat itu sama dengan yang telah diberikan oleh Anto, maka bank menjalankan kewajibannya kepada Badu.

Jika sidik jari pesan ganda SPP & SPJB dienkripsi dengan kunci privat Anto, maka akan menjadi tanda tangan pesan ganda (*dual-signature*) Anto untuk kedua perjanjian tersebut.

Source Code Fungsi Hash

Di bawah ini terdapat source code untuk menghasilkan sebuah nilai hash dari sebuah dokumen.



Gambar 5. Pembuatan sidik jari pesan ganda

```
/*
This works on all machines, is identical to hash() on little-endian machines, and it is much faster than hash(), but it requires -- that buffers be aligned, that is, ASSERT(((ub4)k)&3 == 0), and -- that all your machines have the same endianness.
*/

void  hash2( k, len, state)
register ub1 *k;
register ub4  len;
register ub4 *state;
{
  register ub4 a,b,c,d,e,f,g,h,length;

/*
Use the length and level; add in the golden ratio.
*/
  length = len;
  a=state[0];       b=state[1];       c=state[2];
d=state[3];
  e=state[4];       f=state[5];       g=state[6];
h=state[7];

/*
handle most of the key
*/
  while (len >= 32)
  {
    a += *(ub4 *)(k+0);
    b += *(ub4 *)(k+4);
    c += *(ub4 *)(k+8);
    d += *(ub4 *)(k+12);
    e += *(ub4 *)(k+16);
    f += *(ub4 *)(k+20);
```

```
    g += *(ub4 *)(k+24);
    h += *(ub4 *)(k+28);
    mix(a,b,c,d,e,f,g,h);
    mix(a,b,c,d,e,f,g,h);
    mix(a,b,c,d,e,f,g,h);
    mix(a,b,c,d,e,f,g,h);
    k += 32; len -= 32;
  }

/*
handle the last 31 bytes
*/

  h += length;
  switch(len)
  {
  case 31: h+=(k[30]<<24);
  case 30: h+=(k[29]<<16);
  case 29: h+=(k[28]<<8);
  case 28: g+=(k[27]<<24);
  case 27: g+=(k[26]<<16);
  case 26: g+=(k[25]<<8);
  case 25: g+=k[24];
  case 24: f+=(k[23]<<24);
  case 23: f+=(k[22]<<16);
  case 22: f+=(k[21]<<8);
  case 21: f+=k[20];
  case 20: e+=(k[19]<<24);
  case 19: e+=(k[18]<<16);
  case 18: e+=(k[17]<<8);
  case 17: e+=k[16];
  case 16: d+=(k[15]<<24);
  case 15: d+=(k[14]<<16);
  case 14: d+=(k[13]<<8);
  case 13: d+=k[12];
  case 12: c+=(k[11]<<24);
  case 11: c+=(k[10]<<16);
  case 10: c+=(k[9]<<8);
  case 9 : c+=k[8];
  case 8 : b+=(k[7]<<24);
  case 7 : b+=(k[6]<<16);
  case 6 : b+=(k[5]<<8);
  case 5 : b+=k[4];
  case 4 : a+=(k[3]<<24);
  case 3 : a+=(k[2]<<16);
  case 2 : a+=(k[1]<<8);
  case 1 : a+=k[0];
  }
  mix(a,b,c,d,e,f,g,h);
  mix(a,b,c,d,e,f,g,h);
  mix(a,b,c,d,e,f,g,h);
  mix(a,b,c,d,e,f,g,h);

/*
report the result
*/
  state[0]=a;       state[1]=b;       state[2]=c;
state[3]=d;
  state[4]=e;       state[5]=f;       state[6]=g;
state[7]=h;
}
```

Algoritma SHA-1

Pseudocode algoritma SHA-1:
Catatan: semua variabel adalah 32bit unsigned dan wrap modulo 2^{32} ketika dikalkulasi.

Inisialisasi Variabel:
h0 := 0x67452301
h1 := 0xEFCDAB89
h2 := 0x98BADCFE
h3 := 0x10325476
h4 := 0xC3D2E1F0

Proses Pendahulu:
Tambahkan bit '1' ke depan
Tambahkan k bit '0' dimana k adalah bilangan terkecil positif yang menghasilkan panjang pesan sama dengan 448 (mod 512)
Tambahkan panjang pesan (sebelum proses pendahulu) sebagai integer 64bit big endian.

Proses perubahan pesan menjadi 512bit:
Membagi pesan menjadi 512bit bagian
for setiap bagian
  membagi bagian menjadi 16 kata 32bit big endian w(i), $0 \le i \le 15$

  Perbesar ukurannya menjadi 80 kata 32bit
for i **from** 16 to 79
    w(i) := (w(i-3) **xor** w(i-8) **xor** w(i-14) **xor** w(i-16)) **leftrotate** 1

  Inisialisasi nilai hash untuk setiap bagian:
  a := h0
  b := h1
  c := h2
  d := h3
  e := h4

  Main Loop:
  for i **from** 0 to 79
    if $0 \le i \le 19$ **then**
      f := (b **and** c) **or** ((**not** b) **and** d)
      k := 0x5A827999
    else if $20 \le i \le 39$
      f := b **xor** c **xor** d

```
            k := 0x6ED9EBA1
        else if 40 ≤ i ≤ 59
            f := (b and c) or (b and d) or (c and
d)
            k := 0x8F1BBCDC
        else if 60 ≤ i ≤ 79
            f := b xor c xor d
            k := 0xCA62C1D6

        temp := (a leftrotate 5) + f + e + k +
w(i)
        e := d
        d := c
        c := b leftrotate 30
        b := a
        a := temp
```

Tambahkan nilai hash bagian ke hasilnya:

```
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
```

Hasilkan nilai hash final (big-endian):

digest = hash = h0 **append** h1 **append** h2 **append** h3 **append** h4

Catatan: tidak menggunakan formula dari FIPS PUB 180-1, ekspresi berikut dapat diganti menjadi range yang sesuai dari pseudocode di atas untuk efisiensi yang lebih baik.

(0 ≤ i ≤ 19): f := d **xor** (b **and** (c **xor** d))
(alternatif)
(20 ≤ i ≤ 39): f := b **xor** c **xor** d
(tidak berubah)
(40 ≤ i ≤ 59): f := (b **and** c) **or** (d **and** (b **or** c))
(alternatif 1)
(40 ≤ i ≤ 59): f := (b **and** c) + (d **and** (b **xor** c))
(alternatif 2, menurut Colin Plumb)
(40 ≤ i ≤ 59): f := (b **and** c) **or** (d **and** (b **xor** c))
(alternatif 3, menurut Jeffrey Riaboy)
(60 ≤ i ≤ 79): f := b **xor** c **xor** d
(tidak berubah)

DAFTAR PUSTAKA

1. http://id.wikipedia.org/ Tanggal akses: 26 Desember 2006 pukul: 08.00
2. http://www.geocities.com/ Tanggal akses: 26 Desember 2006 pukul: 08.00
3. http://en.wikipedia.org/ Tanggal akses: 26 Desember 2006 pukul: 08.00
4. http://www.butleburtle.net/ Tanggal akses: 2 Januari 2007 pukul: 17.00
5. http://www.x5.net/ Tanggal akses: 26 Desember 2006 pukul: 08.00