The Keyed-Hash Message Authentication Code (HMAC)

Gigih Bagus Pambekti

13 Oktober 2013

Abstract

Message Authentication Code (MAC) yang berdasarkan pada fungsi hash kriptografi satu arah dikenal sebagai *Keyed Hash Message Authentication Code* (HMAC). Fungsi HMAC sering kali digunakan oleh pengirim pesan untuk menghasilkan sebuah nilai MAC yang dibentuk dengan cara memadatkan atau menyingkat kunci rahasia dan input pesan. MAC secara khusus dikirim kepada penerima pesan bersamaan dengan pesan. Algoritma ini secara luas digunakan untuk menyediakan fasilitas autentikasi pesan dan integrasi pesan pada komunikasi digital akan tetapi di lain sisi ini tidak mendukung prinsip non-repudiation. Hal ini dikarenakan oleh pemilikan kunci privat yang bisa diketahui oleh beberapa orang, sehingga tidak memungkinkan untuk mengetahui secara pasti orang yang membangkitkan nilai hash yang sebenarnya. Makalah ini akan membahas deskripsi, algoritma umum, dan contoh penggunaan dari *Keyed-hash Message Authentication Code* (HMAC).

Pendahuluan

Fungsi hash mengangkat mengenai inti dari kriptografi primitif. Prinsip ini diperkenalkan pada tahun 1950an. Fungsi hash merupakan fungsi yang menerima masukan (*input*) suatu string sembarang dan menghasilkan string keluaran (*output*) dengan panjang tetap yang pada umumnya string keluaran ini berukuran lebih kecil daripada ukuran string masukan. String keluaran fungsi hash ini dapat juga disebut sebagai nilai hash atau *message digest*. Salah satu sifat terpenting dari fungsi hash adalah tidak ada satupun menemukan *collision* yaitu dua buah pesan yang berbeda dapat dipetakan pada nilai hash yang sama. Fungsi hash ini biasanya digunakan untuk memverifikasi suatu dokumen atau arsip dengan arsip asli.

Suatu kegunaan aplikasi dari fungsi hash adalah untuk mendeteksi error. Penambahan *message digest* memperbolehkan penemuan error selama proses transmisi berlangsung. Di akhir penerimaan, nilai hash yang diterima pesan dikalkulasi ulang dan dibandingkan dengan nilai hash yang diterima. Apabila

keduanya tidak cocok, akan terjadi error. Pendeteksian ini hanya untuk *random error*. Seorang penyadap aktif mungkin saja menangkap sebuah pesan, kemudian memodifikasi sesuai apa yang dia inginkan dan mengirimkannya kembali dengan ditambahi catatan dengan *digest* yang dikalkulasi ulang untuk pesan yang termodifikasi.

Secara umum fungsi hash harus memiliki 2 sifat dasar yaitu :

1. Sifat kompresi, fungsi h memetakan suatu input x dengan panjang sembarang ke output y = h(x) dengan panjang tetap n; dan

2. Mudah dihitung, dengan diberikan h dan sebuah input x, maka y mudah dihitung.

Penggunaan fungsi hash, ini memungkinkan untuk menghasilkan sebuah *digital signature* dengan panjang yang fix sehingga memeprcayai keseluruhan pesan dan menjamin keaslian dari pesan. Untuk menghasilkan *digital signature* untuk pesan x, nilai hash x, diberikan dengan H(x), dihitung dan kemudian dienkripsi dengan kunci rahasia dari si pengirim.

Message Authentication Code (MAC) adalah fungsi hash satu-arah yang menggunakan kunci rahasia (*secret key*) dalam pembangkitan nilai hash. Algoritma *Message Authentication Code* merupakan algoritma kriptografi simetris primitif yang mengizinkan si pengirim dan si penerima dalam membagikan suatu kunci rahasia umum untuk meyakinkan secara pasti isi atau kandungan dari pesan yang ditransmisikan tersebut tidak dirusak atau diubah diantara kedua belah pihak. Algoritma MAC juga dapat digunakan untuk aplikasi yang membutuhkan perpaduan autentikasi dengan enkripsi.

Ide dasar dari MAC bahwa pihak musuh tanpa mengetahui tentang kuncinya seharusnya tidak mampu "memalsukan" hasil dari MAC untuk beberapa pesan baru, sama halnya ketika beberapa pesan sebelumnya dan keterkaitan dari hasil MACnya yang diketahui. MAC memiliki tiga buah algoritma yaitu pembangkit kunci (K), *sign* (S), dan verifikasi (V). Algotirma pembangkit kunci tidak memerlukan *input* dan *output* suatu kunci yang baru. Algoritma *sign* memerlukan suatu pesan dan kunci serta menghitung suatu tag yang disebut *authenticator* (pengautentikasi). Kemudian algoritma verifikasi memerlukan suatu pesan, kunci, *tag*, dan *output* 1 jika ini *tag* dianggap secara benar di-*generate* untuk *specied message* dan bernilai 0 untuk lainnya.

Ada beberapa macam *Message Authentication Code* , salah satunya adalah Keyed-hash *Message Authentication Code* (HMAC). *Keyed-hash Message Authentication Code* adalah teknik *Message Authentication Code* yang dibuat oleh Mihir Bellare, Ran Canetti, Hugo Krawczyk pada tahun 1996. Hal ini didasari karena fungsi hash itu sendiri tidak memungkinkan penggunaan kunci ketika menghitung nilai *digest*-nya. HMAC sebaiknya digunakan pada suatu kombinasi dengan sebuah fungsi hash kriptografi yang telah ditetapkan di *Federal Information Processing Standard* (FIPS). HMAC menggunakan kunci rahasia untuk perhitungan dan verifikasi dari MAC.

Tujuan dari dibangunnya algoritma HMAC ini adalah:

1. Untuk menggunakan fungsi hash tanpa modifikasi, terutama fungsi hash yang telah tersedia pada perangkat lunak dan mudah didapatkan.

2. Untuk mempertahankan keaslian performa dari algoritma hash yang sudah ada tanpa menimbulkan degradasi yang signifikan.

3. Untuk menggunakan dan mengatur kunci secara mudah.

4. Untuk mendapatkan pengertian yang lebih dalam dari analisis kriptografi mengenai kekuatan mekanisme autentikasi yang berdasarkan fungsi hash.

5. Untuk mempermudah pengubahan atau penggantian fungsi hash yang digunakan apabila algoritma hash baru yang lebih cepat atau lebih aman ditemukan.

Teori yang Mendukung

1. Definisi

- *Message Authentication Code* (MAC): Jumlah penanda (kode) kriptografi yang dihasilkan dari data yang lewat melalui sebuah algoritma autentikasi pesan. Sebuah MAC dengan himpunan pesan M dan himpunan *tag* T merupakan triple algoritma I=(K, S, V) dengan sifat sebagai berikut:

1. Algoritma pembangkit kunci K merupakan suatu algoritma yang secara acak tidak membutuhkan input dan nilai balik dari kunci K. Sebut saja [K] adalah himpunan kunci MAC.

2. Algoritma *sign* S membutuhkan kunci K dari [K] dan pesan m dari M dan nilai balik *tag* t dari T.

3. Algoritma tersebut harus memenuhi sifat sebagai berikut: untuk beberapa $K \in [K]$, untuk beberapa pesan $m \in M$ dan untuk beberapa $t \in [S(K,m)]$, kita punya $V(K, m, t) = 1$.

algoritma verifikasi V membutuhkan sebuah kunci $K \in [K]$, suatu pesan $m \in M$ dan sebuah *tag* $t \in T$ serta nilai balik sebuah bit $b \in \{0,1\}$. Sebuah output $b=1$ berarti bahwa *tag* yang diterima bernilai valid dan bernilai 0 apabila *tag* ditolak. Algoritma ini merupakan algoritma yang deterministik.

Suatu MAC merupakan fungsi h yang memenuhi kondisi sebagai berikut:

1. Input X dapat berupa panjang yang berubah-ubah dan hasil h(K,X) memiliki panjang fix n-bit, dengan *key* K adalah secondary input dengan panjang k-bit yang ditentukan.

2. Diketahui h, K, dan sebuah input X, perhitungan h(K,X) harus mudah dihitung.

3. Diberikan sebuah *message* X (tanpa diketahui K), ini harus sulit untuk menentukan h(K,X). Begitu pula saat suatu himpunan besar dari pasangan $\{X_i,h(K,X_i)\}$ diketahui maka ini juga sulit untuk menentukan kunci K atau untuk menghitung h(K,X') untuk beberapa *message* baru $X' \neq X_i (\forall_i)$.

- *Keyed Hash-based Message Authentication Code* (HMAC): sebuah kode autentikasi pesan yang menggunakan kunci kriptografi yang disambungkan dengan sebuah fungsi hash.
- *Cryptographic key* (*key*): Suatu parameter yang digunakan dalam hubungan dengan algoritma kriptografi yang menjelaskan operasi secara khusus dari algoritma tersebut. kunci kriptografi digunakan oleh algoritma HMAC untuk menghasilkan suatu MAC pada data.
- *Secret key*: Suatu kunci kriptografi yang secara khusus atau unik terkait dengan satu atau lebih entitas. Penggunaan istilah "*secret*" dalam konteks ini tidak menggambarkan sebuah tingkat klasifikasi, hanya saja istilah ini menggambarkan kebutuhan untuk melindungi suatu kunci dari kebocoran atau perubahan.

2. *Cryptographic Keys*

Ukuran kunci, K, seharusnya berukuran minimal $\frac{L}{2}$ atau lebih besar daripada $\frac{L}{2}$ dimana L adalah ukuran dari *output* fungsi hash. (Catatan: kunci yang lebih besar dari L byte tidak secara signifikan meneambah kekuatan dari fungsi). Aplikasi yang menggunakan kunci yang panjangnya lebih besar dari B byte, dimana B adalah ukuran *input* hash, kunci hash pertama menggunakan h dan kemudian menggunakan hasil dari string L byte sebagai kunci K dari HMAC. Kunci seharusnya dipilih secara acak menggunakan metode pembangkitan kunci dan seharusnya diganti secara periodik. Kunci seharusnya diproteksi pada sebuah cara yang konsisten dengan nilai data harus di proteksi (yakni data yang diautentikasi menggunakan fungsi HMAC).

3. *Truncated Output*

Sudah diketahui secara pasti bahwa praktek menggunakan MAC yakni memotong *output*nya (yaitu ukuran panjang dari MAC yang digunakan kurang dari panjang dari keluaran (*output*) fungsi L dari MAC) sehingga sebuah aplikasi yang menggunakan standar ini mungkin saja akan memotong output dari HMAC. Ketika HMAC yang terpotong digunakan, t byte paling kiri dari perhitungan HMAC seharusnya digunakan sebagai MAC. Panjang keluaran, t, harus tidak boleh kurang dari empat byte (yaitu $4 \leq t \leq L$). Akan tetapi, t setidaknya berukuran $\frac{L}{2}$ byte (yaitu $\frac{L}{2} \leq t \leq L$) kecuali apabila suatu aplikasi atau protokol membentuk banyak percobaan yang tidak berguna. Contoh: *low bandwidth*

channel mungkin mencegah banyak percobaan pada MAC 4 byte, atau protokol yang mungkin diizinkan hanya untuk sebuah angka kecil dari percobaan MAC yang salah atau cacat.

4. Konstruksi Merkle-Damgard

Konstruksi Merkle-Damgard dapat mengonstruksi suatu collision-resistant fungsi hash dengan domain yang besar. Konstruksi ini didefinisikan sebagai : Misalkan F : $\{0,1\}^b \times \{0,1\}^n \rightarrow \{0,1\}^n$ adalah *collision resistant compression (hash) function*, dan IV = $H_1 \in \{0,1\}^n$sebagai *initial value*. IV ini merupakan sebuah string yang ditentukan dimana pernyataan nilai yang tetap tidak berarti disini.

Diberikan m = m_1. . . m_L, dimana setiap $m_i$adalah b-bit blok, nilai hash H(m) dihitung sebagai berikut: H_{i+1}:= F(m_i, H_i), untuk setiap i = 1, . . ., L-1 dan H(m) := H_{L+1}:= F(m_L||pad, H_L). *Padding* dihitung sebagai berikut : jika |m| bukan merupakan kelipatan dari b maka sebuah blok pad = 10 . . . 0|| msg-len ditambahkan, dimana msg-len merupakan representasi bitstring dari panjang pesan (panjang pesan menggunakan secara tepat sedikitnya 64 bit dari pesan yang dipadding), dan nilai 0 dipilih sehingga |m| + |pad| menjadi suatu kelipatan dari ukuran blok b.

Pembahasan

1. HMAC

Untuk menghitung MAC berdasarkan data teks menggunakan fungsi HMAC sebagai berikut operasi yang dihasilkan: $MAC(teks)_t = HMAC$(K, teks)$_t$= $HMAC_K$ (m)=h((K$\oplus$opad)||h((K$\oplus$ipad)||m)).

dengan K adalah kunci privat yang diketahui oleh pengirim dan penerima, h adalah fungsi hash yang digunakan, m adalah pesan yang akan diautentikasi, opad adalah 0x5c5c5c...5c (64 kali perulangan heksadesimal 5c) dan ipad adalah 0x363636...36 (64 kali perulangan heksadesimal 36) dengan panjang yang sama. Berikut adalah algoritma dari HMAC:

Langkah 1 : jika panjang K=B, set K0 = K. Jump: langkah 4

Langkah 2 : jika panjang K>B, hash K untuk memperoleh sebuah string L byte. (K=h(K))

Langkah 3 : jika panjang K<B, tambahkan nol pada akhir K untuk membentuk string B byte K0

Langkah 4 : lakukan XOR antara K0 dengan ipad untuk memperoleh string byte berukuran B. (K0 $\oplus$ipad)

Langkah 5 : menambahkan string teks ke dalam hasil string dari langkah 4 sebelumnya. ((K0$\oplus$ipad)|| teks)

Langkah 6 : terapkan H untuk string yang diperoleh dari langkah 5. (h(K0⊕ipad)|| teks)

Langkah 7 : XOR-kan K0 dengan opad. (K0⊕opad)

Langkah 8 : sisipkan hasil dari langkah 6 ke langkah 7. ((K0⊕opad)||h(K0⊕ipad)|| teks).

Langkah 9 : terapkan H dari langkah 8. (h(K0⊕opad)||h(K0⊕ipad)|| teks).

Langkah 10 : pilih t byte paling kiri dari hasil langkah 9 sebagai hasil MAC.

2. Catatan Implementasi dari HMAC

Algoritma HMAC ini dispesifikasikan untuk fungsi hash FIPS yang direkomendasikan. Dengan perubahan yang sedikit saja, implementasi dari HMAC dapat secara mudah mengganti satu nilai fungsi hash h dengan fungsi hash baru h'. Sesuai dengan konsep dari algoritma hasil lanjutan dari fungsi kompresi blok B byte HMAC di atas, nilai dari K0⊕ipad(inner key) dan K0⊕opad (outer key) dapat dihitung sekali saja, yaitu ketika waktu generasi kunci K atau sebelum digunakan untuk pertama kalinya.

Hasil operasi XOR tersebut yang menghasilkan inner key dan outer key, dapat disimpan dan kemudian digunakan untuk menginisialisasi fungsi hash h setiap waktu saat sebuah pesan perlu diautentikasi menggunakan kunci yang sama. Setiap kali mengautentikasi pesan dengan kunci k yang sama, hasil operasi XOR tersebut menjaga aplikasi dari fungsi hash h pada dua blok berukuran B byte. Hal ini sangat berguna ketika harus melakukan autentikasi data dalam bentuk aliran yang pendek. Tapi, sesuai dengan prinsip keamanan, kedua hasil (i*nner key* dan *outer key*) tersebut harus dijaga dan dilindungi seperti halnya kunci. Hal ini jelas sekali karena *inner key* dan *outer key* tersebut merupakan kunci yang diubah menggunakan sebuah konstanta. Sehingga jika *inner key* dan *outer key* diketahui oleh pihak lain, pihak tersebut dapat dengan mudah mendapatkan kunci yang sebenarnya.

Keterbatasan dari Algoritma MAC

Keberhasilan verifikasi dari MAC tidak secara utuh menjamin bahwa pesan yang beriringan bernilai autentik atau asli. Terdapat suatu kesempatan dimana sebuah sumber tanpa kejelasan dari kunci dapat menampilkan suatu MAC yang "diakui" pada plainteks pesan yang akan melalui prosedur verifikasi. Contoh: suatu keputusan MAC t-bit yang diakui pada plainteks pesan mungkin saja berhasil diverifikasi dengan probabilitas harapan $(\frac{1}{2})^t$. Keterbatasan ini merupakan sifat yang melekat di beberapa algoritma MAC. Keterbatasan ini diperbesar jika sebuah aplikasi memperbolehkan suatu nonautentik pesan secara berulang ditampilkan untuk memverifikasi dengan MAC yang diakui. Masing-masing percobaan sendiri mengganti hanya dengan probabilitas

kecil $(\frac{1}{2})^t$, akan tetapi untuk percobaan yang berulang-ulang, probabilitasnya meningkat, akhirnya satu nilai dari MAC akan berhasil diverifikasi.

Dengan cara yang sama jika sebuah aplikasi mengizinkan suatu MAC yang ditampilkan dengan nonautentik pesan dimana probabilitasnya akan bertambah, MAC akan berhasil diverifikasi untuk satu buah pesan. Oleh karena itu, secara umum jika panjang MAC dipotong maka panjangnya, t, seharusnya dipilih sebesar kebutuhan prakteknya, dengan paling tidak setengah dari beberapa bit sebagai ukuran blok output L. Nilai minumum untuk t dikurangi sampai 32 bit untuk aplikasi dimana dari dua jenis dari percobaan yang berulang-ulang itu diuraikan pada paragraf sebelumnya cukup terbatas.

Sebagai contoh, aplikasi atau protokol yang mengontrol aplikasi, membolehkan memonitor dari semua plainteks pesan dan MAC tersebut dipresentasikan untuk pembuktian atau verifikasi, dan secara permanen menolak plainteks pesan atau beberapa MAC yang termasuk ke dalam banyaknya percobaan yang gagal. Contoh lain terjadi ketika suatu bandwith dari channel komunikasi cukup rendah untuk menghalangi atau menghindari dari banyaknya percobaan. Dari kedua kasus diatas, nilai maksimal dari ketidakberhasilan tersebut harus ditentukan dasar resiko kepada yang berhubungan dengan kesensifitasan data, panjang t dan penggunaan algoritma MAC .

Fungsi Hash untuk HMAC

Ada beberapa fungsi hash yang mungkin digunakan untuk algoritma HMAC. Fungsi hash tersebut adalah:

- SHA1 dan keluarga

Fungsi hash SHA1dan keluarga seperti SHA224, SHA256, SHA512 merupakan algoritma yang paling banyak digunakan dalam implementasi *Keyed-hash Message Authentication Code* (HMAC). Fungsi hash SHA1 menghasilkan output sebanyak 160 bit,SHA224 menghasilkan output sebanyak 224 bit, SHA256 menghasilkan output sepanjang 256, dan SHA512 menghasilkan output sepanjang 512 bit. Walaupun sering digunakan, akan tetapi pada beberapa tahun yang lalu pada fungsi hash SHA1 ini ditemukan adanya bentrokan atau *collision* yang dianggap sudah memasuki tahap fatal. hal ini dikemukakan oleh Christian Rechberger and Christophe De Cannière pada tahun 2006. Oleh karena itu, penggunaan SHA1 ini sudah tidak dianjurkan untuk digunakan kembali oleh kalangan kriptografi , sedangkan untuk keluarga fungsi hash SHA yang lain, serangan yang dinilai ampuh belum bisa dianggap menghancurkan fungsi ini, sehingga keluarga fungsi hash SHA yang lain masih layak digunakan untuk algoritma HMAC.

- MD5

Berkaitan dengan fungsi hash SHA1, fungsi MD5 merupakan fungsi hash yang banyak digunakan dalam pengimplementasian algoritma HMAC. MD5 sendiri adalah hash yang diciptakan oleh Ronald Rivest pada tahun 1991. MD5 ini

menghasilkan output hash sepanjang 128 bit dimana panjang bit MD5 lebih pendek daripada keluarga hash SHA. Hal ini menyebabkan keamanan yang diberikan oleh MD5 dinilai kurang terjamin dibandingkan dengan SHA dan keluarga. Adanya *collision* juga telah ditemukan melalui percobaan yang dilakukan oleh Xiaoyun Wang dan beberapa peneliti lainnya pada tahun 2005. Bahkan pada Maret 2006, peneliti bernama Vlastimil Klima telah berhasil menemukan algoritma yang bisa menemukan collision pada MD5 dalam waktu yang kurang dari satu menit. Oleh karena itu, penggunaan MD5 juga sudah tidak dianjurkan lagi.

- Tiger Hash

Tiger hash adalah fungsi kriptografi hash yang diciptakan oleh Ross Anderson dan Eli Biham pada tahun 1995. Fungsi hash ini menghasilkan output digest sepanjang 192 bit. Algoritma Tiger hash ini dikembangkan menggunakan paradigma Merkle-Damgard. *Collision* pada algoritma ini juga sudah ditemukan oleh peneliti bernama John Kelsey dan Stefan Lucks.

3. Contoh Penggunaan HMAC

a. Contoh tanpa pemotongan nilai MAC dengan kunci 64-byte

1. Text: 53616d70 6c652023 31
2. Key: 00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617 18191a1b 1c1d1e1f 20212223 24252627 28292a2b 2c2d2e2f 30313233 34353637 38393a3b 3c3d3e3f
3. K0: 00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617 18191a1b 1c1d1e1f 20212223 24252627 28292a2b 2c2d2e2f 30313233 34353637 38393a3b 3c3d3e3f
4. K0⊕ipad: 36373435 32333031 3e3f3c3d 3a3b3839 26272425 22232021 2e2f2c2d 2a2b2829 16171415 12131011 1e1f1c1d 1a1b1819 06070405 02030001 0e0f0c0d 0a0b0809
5. K0⊕ipad||Text: 36373435 32333031 3e3f3c3d 3a3b3839 26272425 22232021 2e2f2c2d 2a2b2829 16171415 12131011 1e1f1c1d 1a1b1819 06070405 02030001 0e0f0c0d 0a0b0809 53616d70 6c652023 31
6. Hash(K0⊕ipad||Text): bcc2c68c abbbf1c3 f5b05d8e 7e73a4d2 7b7e1b20
7. K0⊕opad: 5c5d5e5f 58595a5b 54555657 50515253 4c4d4e4f 48494a4b 44454647 40414243 7c7d7e7f 78797a7b 74757677 70717273 6c6d6e6f 68696a6b 64656667 60616263
8. (K0⊕opad)||Hash(K0⊕ipad||Text): 5c5d5e5f 58595a5b 54555657 50515253 4c4d4e4f 48494a4b 44454647 40414243 7c7d7e7f 78797a7b 74757677 70717273 6c6d6e6f 68696a6b 64656667 60616263 bcc2c68c abbbf1c3 f5b05d8e 7e73a4d2 7b7e1b20

9. Hash((K0⊕opad)||Hash(K0⊕ipad||Text)): 4f4ca3d5 d68ba7cc 0a1208c9 c61e9c5d a0403c0a

10. Maka nilai MAC-nya adalah (20 byte) : 4f4ca3d5 d68ba7cc 0a1208c9 c61e9c5d a0403c0a

b. Contoh dengan pemotongan nilai MAC dengan kunci 49-byte:

1. Pesan(Text) : 3616d70 6c652023 34

2. Key : 70717273 74757677 78797a7b 7c7d7e7f 80818283 84858687 88898a8b 8c8d8e8f 90919293 94959697 98999a9b 9c9d9e9f a0

3. K0 : 70717273 74757677 78797a7b 7c7d7e7f 80818283 84858687 88898a8b 8c8d8e8f 90919293 94959697 98999a9b 9c9d9e9f a0000000 00000000 00000000 00000000

4. K0⊕ipad : 46474445 42434041 4e4f4c4d 4a4b4849 b6b7b4b5 b2b3b0b1 bebfbcbd babbb8b9 a6a7a4a5 a2a3a0a1 aeafacad aaaba8a9 96363636 36363636 36363636 36363636

5. K0⊕ipad||Text : 46474445 42434041 4e4f4c4d 4a4b4849 b6b7b4b5 b2b3b0b1 bebfbcbd babbb8b9 a6a7a4a5 a2a3a0a1 aeafacad aaaba8a9 96363636 36363636 36363636 36363636 3616d70 6c652023 34

6. Hash(K0⊕ipad || Text): bf1e889d 876c34b7 bef3496e d998c8d1 16673a2e

7. K0⊕opad : 2c2d2e2f 28292a2b 24252627 20212223 dcdddedf d8d9dadb d4d5d6d7 d0d1d2d3 cccdcecf c8c9cacb c4c5c6c7 c0c1c2c3 fc5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c

8. (K0⊕opad)||Hash(K0⊕ipad||Text): 2c2d2e2f 28292a2b 24252627 20212223 dcdddedf d8d9dadb d4d5d6d7 d0d1d2d3 cccdcecf c8c9cacb c4c5c6c7 c0c1c2c3 fc5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c bf1e889d 876c34b7 bef3496e d998c8d1 16673a2e

9. Hash((K0⊕opad) || Hash(K0⊕ipad || Text)): 9ea886ef e268dbec ce420c75 24df32e0 751a2a26

10. Maka nilai MAC-nya adalah (karena hanya 12 byte yang diambil) : 9ea886ef e268dbec ce420c75

c. Contoh HMAC SHA-1 dengan 64-Byte Kunci

1. Text: "Sample #1"

2. Key: 00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617 18191a1b 1c1d1e1f 20212223 24252627 28292a2b 2c2d2e2f 30313233 34353637 38393a3b 3c3d3e3f

3. K0 : 00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617 18191a1b 1c1d1e1f 20212223 24252627 28292a2b 2c2d2e2f 30313233 34353637 38393a3b 3c3d3e3f

4. K0⊕ipad: 36373435 32333031 3e3f3c3d 3a3b3839 26272425 22232021 2e2f2c2d 2a2b2829 16171415 12131011 1e1f1c1d 1a1b1819 06070405 02030001 0e0f0c0d 0a0b0809

5. (Key⊕ipad)||text: 36373435 32333031 3e3f3c3d 3a3b3839 26272425 22232021 2e2f2c2d 2a2b2829 16171415 12131011 1e1f1c1d 1a1b1819 06070405 02030001 0e0f0c0d 0a0b0809 53616d70 6c652023 31

6. Hash((Key⊕ ipad)||text): bcc2c68c abbbf1c3 f5b05d8e 7e73a4d2 7b7e1b20

7. K0⊕opad: 5c5d5e5f 58595a5b 54555657 50515253 4c4d4e4f 48494a4b 44454647 40414243 7c7d7e7f 78797a7b 74757677 70717273 6c6d6e6f 68696a6b 64656667 60616263

8. (K0⊕ opad) || Hash((Key⊕ipad)||text): 5c5d5e5f 58595a5b 54555657 50515253 4c4d4e4f 48494a4b 44454647 40414243 7c7d7e7f 78797a7b 74757677 70717273 6c6d6e6f 68696a6b 64656667 60616263 bcc2c68c abbbf1c3 f5b05d8e 7e73a4d2 7b7e1b20

9. HMAC(Key, Text) = Hash((K0⊕opad) || Hash((Key⊕ipad)||text)): 4f4ca3d5 d68ba7cc 0a1208c9 c61e9c5d a0403c0a

10. 20-byte HMAC(Key, Text): 4f4ca3d5 d68ba7cc 0a1208c9 c61e9c5d a0403c0a

d. Contoh HMAC SHA-1 dengan 49-Byte Kunci, terpotong untuk 12-Byte HMAC

1. Text: "Sample #4"

2. Key: 70717273 74757677 78797a7b 7c7d7e7f 80818283 84858687 88898a8b 8c8d8e8f 90919293 94959697 98999a9b 9c9d9e9f a0

3. K0: 70717273 74757677 78797a7b 7c7d7e7f 80818283 84858687 88898a8b 8c8d8e8f 90919293 94959697 98999a9b 9c9d9e9f a0000000 00000000 00000000 00000000

4. K0 ⊕ ipad: 46474445 42434041 4e4f4c4d 4a4b4849 b6b7b4b5 b2b3b0b1 bebfbcbd babbb8b9 a6a7a4a5 a2a3a0a1 aeafacad aaaba8a9 96363636 36363636 36363636 36363636

5. (Key⊕ipad)||text: 46474445 42434041 4e4f4c4d 4a4b4849 b6b7b4b5 b2b3b0b1 bebfbcbd babbb8b9 a6a7a4a5 a2a3a0a1 aeafacad aaaba8a9 96363636 36363636 36363636 36363636 53616d70 6c652023 34

6. Hash((Key⊕ipad)||text): bf1e889d 876c34b7 bef3496e d998c8d1 16673a2e

7. K0⊕opad: 2c2d2e2f 28292a2b 24252627 20212223 dcdddedf d8d9dadb d4d5d6d7 d0d1d2d3 cccdcecf c8c9cacb c4c5c6c7 c0c1c2c3 fc5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c

8. (K0⊕opad) || Hash((Key⊕ipad)||text): 2c2d2e2f 28292a2b 24252627 20212223 dcdddedf d8d9dadb d4d5d6d7 d0d1d2d3 cccdcecf c8c9cacb c4c5c6c7 c0c1c2c3 fc5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c bf1e889d 876c34b7 bef3496e d998c8d1 16673a2e

9. HMAC(Key, Text) = Hash((K0⊕opad) || Hash((Key⊕ipad)||text)): 9ea886ef e268dbec ce420c75 24df32e0 751a2a26

10. 12-byte HMAC(Key, Text): 9ea886ef e268dbec ce420c75

4. Aspek Keamanan pada HMAC

Ada beberapa hal yang perlu diperhatikan dalam aspek keamanan dari HMAC. Hal-hal tersebut antara lain:

a. Keamanan kunci Keamanan

HMAC sangat tergantung kepada tingkat kerahasiaan dari kunci. Maka itu, pengguna harus menjaga kunci ini dari tindakan yang membahayakan. Standar HMAC ini juga tidak menjamin bahwa implementasinya aman. Keamanan adalah tanggung jawab sepenuhnya dari pengguna standar HMAC untuk memastikan implementasinya aman.

b. Keamanan fungsi hash

Faktor yang sangat penting dalam aspek keamanan dari HMAC adalah keamanan fungsi hash yang digunakan itu sendiri. Hal ini disebabkan satu-satunya cara HMAC gagal adalah ketika fungsi hash yang digunakan juga gagal. Fungsi hash sendiri sangat rentan terhadap *collision*.

c. Pemalsuan

Keamanan dari MAC berarti keamanannya dari pemalsuan. MAC dikatakan gagal jika seorang penyusup yang tidak memiliki kunci K, bisa menemukan beberapa pesan Text bersama dengan nilai MAC-nya. Penyusup tersebut diasumsikan dapat mengumpulkan sejumlah contoh dari teks dan nilai MAC-nya yang valid dengan melakukan observasi terhadap jalur aliran data antara pengirim dan penerima. Bahkan sang penyusup bisa melakukan *chosen message attack* dengan mempengaruhi pemilihan pesan yang akan dihitung nilai MAC-nya oleh pengirim. Karena itulah, keamanan dari MAC ini bisa dianggap sebagai ukuran kemungkinan terjadinya pemalsuan yang berhasil dengan melakukan serangan seperti itu.

5. Serangan pada HMAC

a. Birthday attack

Birthday attack adalah sebuah jenis serangan di dalam konteks kriptografi yang menggunakan perhitungan matematis. Tujuan dari serangan ini adalah mencari 2 buah pesan yang hampir sama yang memiliki nilai fungsi hash yang sama. Setelah salah satu pesan ditandatangani, pihak penyusup menggunakan nilai fungsi hash tersebut untuk ditempelkan pada pesan yang lain.

Sebagai contoh: Alice ingin menjebak Bob untuk menandatangani kontrak yang telah dicurangi olehnya. Kemudian Alice menyiapkan kontrak yang benar M dan kontrak yang salah M'. Selanjutnya, Alice akan menemukan posisi pada pesan M yang bisa diubah tanpa mengubah artinya, seperti menambahkan tanda baca koma, baris kosong, dua spasi dan sebagainya. Dengan mengombinasikan perubahan tesebut, Alice dapat menciptakan banyak variasi dari M. Begitu pula dengan tindakan sebelumnya, Alice juga melakukan perubahan pada kontrak M'. Kemudian dia menghitung nilai fungsi hash pada kedua pesan kontrak tersebut sehingga nilai kontrak yang benar dan kontrak yang salah memiliki nilai fungsi hash yang sama. Kemudian dia memberikan kontrak yang benar untuk ditandatangani oleh Bob. Bob yang tidak mengetahui hal ini kemudian memberikan tanda tangan pada kontrak yang benar. Setelah itu, Alice mengambil tanda tangan dari kontrak tersebut dan menambahkannya kepada kontrak yang salah. Dengan demikian secara tidak langsung Bob telah menandatangani kontrak tersebut.

Ada beberapa cara untuk menghindari serangan semacam ini. Yang pertama adalah dengan memilih panjang *output* dari fungsi hash sepanjang mungkin. Sehingga kemungkinan dari serangan ini secara komputasi tidak masuk akal. Cara yang kedua adalah Bob harus mengubah beberapa bagian dari kontrak yang disodori oleh Alice sebelum ditandatangani. Akan tetapi, cara yang kedua ini juga tidak menyelesaikan permasalahan dari *Birthday attack* ini. Hal ini disebabkan bahwa Alice bisa mencurigai Bob melakukan *Birthday attack* kepada dirinya. Serangan *Birthday attack* ini merupakan serangan yang paling berbahaya yang bisa dilakukan kepada sebuah *Message Authentication Code*.

b. Collision attack

Collision attack adalah celah pada kriptografi fungsi hash yang menghasilkan nilai hash yang sama untuk dua buah pesan yang jauh berbeda. Pada fungsi hash MD5 dan SHA1, celah *collision* ini telah ditemukan oleh beberapa orang peneliti kriptografi. Contohnya *collision* pada SHA1 ditemukan oleh Xiaoyun Wang, Yiqun Lisa Yin dan Hongbu Yo pada bulan Februari 2005.

c. Preimage attack

Preimage attack memiliki jenis serangan yang hampir sama dengan *collision attack*. Bedanya, pada *preimage* dibutuhkan 2^n operasi supaya serangan pada

nilai hash sebanyak n-bit sukses. Sedangkan pada *collision attack*, jumlah operasi yang dibutuhkan adalah $2^{\frac{n}{2}}$ operasi. Ada dua jenis *preimage attack* yang biasa dilakukan. Pertama, mencari sebuah pesan sehingga nilai hash dari pesan tersebut sama dengan nilai H yang telah didefiniskan. Yang kedua adalah mencari pesan M2 sehingga nilai hash-nya sama dengan nilai hash dari pesan M1 yang telah didefinisikan.

d. Brute Force attack

Jenis serangan *brute force* ini adalah salah satu jenis serangan yang paling umum dalam dunia kriptografi. Ciri-ciri dari serangan *brute force* adalah mencoba semua kemungkinan yang ada untuk mendapatkan nilai tertentu. Pada HMAC ini, serangan ini akan mencoba kemungkinan pesan yang benar untuk sebuah nilai hash yang diketahui.

e. Rainbow attack

Metode Rainbow adalah salah satu seni untuk serangan *brute force*. Penggunaan metode *Rainbow* sekarang hanya digunakan untuk menge-*crack* LM *password,* akan tetapi secara mudah dapat dimofikasi untuk beberapa *one-way function* yang diketahui. Ide dasar dari metode *Rainbow* adalah basis dari pasangan-pasangan pra-perhitungan dari input dan nilai output. Jika *collision* terjadi (*output* dengan nilai ganda), hanya satu pasang saja yang akan disimpan. Pada PC modern , hal ini dapat dihitung dengan waktu sekitar 50 jam. Ukuran basis dari kamus bahasa Inggris dengan seluruh varian (huruf besar dan huruf kecil, permutasi kata, dsb.) lebih kecil dari 500 MB. Dengan basis 1 GB menjamin keberhasilan sebesar 99,99 persen. Serangan metode Rainbow sangat cepat karena serangan ini hanya membandingkan *message digest* dengan nilai dari basis.

6. Kelebihan dan Kekurangan HMAC

Kelebihan:

1. Keamanannya terjamin
2. Fungsi hash bisa diganti dengan yang lain
3. Adanya pemotongan *output* mengurangi kemungkinan serangan terhadap algoritma HMAC

Kekurangan:

1. Keamanannya sangat bergantung kepada fungsi hash yang digunakan
2. Operasi yang digunakan masih sederhana

Kesimpulan

Keyed-hash Message Authetication Code adalah salah satu algoritma *Message Authetication Code* yang umum digunakan. Biasanya HMAC ini diaplikasikan pada aplikasi web yang membutuhkan koneksi yang aman, contohnya jual-beli *online*. HMAC ini memiliki keunggulan pada pemakaian kunci kriptografi dan juga *truncated output* (pemotongan nilai MAC). HMAC juga memungkinkan penggantian fungsi hash yang digunakan secara mudah. Walaupun begitu, keamanan dari HMAC ini sangat bergantung kepada fungsi hash yang digunakan. Jika fungsi hash yang dipakai mempunyai banyak celah keamanan, maka HMAC yang diimplementasikan juga akan memiliki banyak celah yang bisa dimanfaatkan oleh pihak penyusup.

REFERENSI

1. National Institute of Standards and Technology, *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Publication 198, March 6, 2002.

2. National Institute of Standards and Technology, *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Publication #HMAC, MONTH DAY, 2001.

3. Taufik Ramadhany, Paper *Keyed-hash Message Authentication Code (HMAC).*

4. Dwi Destrya Sofiana , Ferdiansyah, dan Wahyu Haryadi Sihombing, paper *MD5-MAC.*

5. Spasic, Boban, paper *Hash Functions.*

6. Prof. Michael Backes, *Lecture Notes for CS-578 Cryptography (SS2007): MACs and Hash Functions*, Saarland University.

7. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, Paper *Keyed Hash Functions*, Centre for Computer Security Research Department of Computer Science University of Wollongong, Wollongong NSW 2522, Australia.

8. Bart Van Rompay, 2004, *Analysis and Design Of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers,* Katholieke Universiteit Leuven Faculteit Toegepaste Wetenschappen Arenbergk asteel, B-3001 Heverlee (Belgium).