

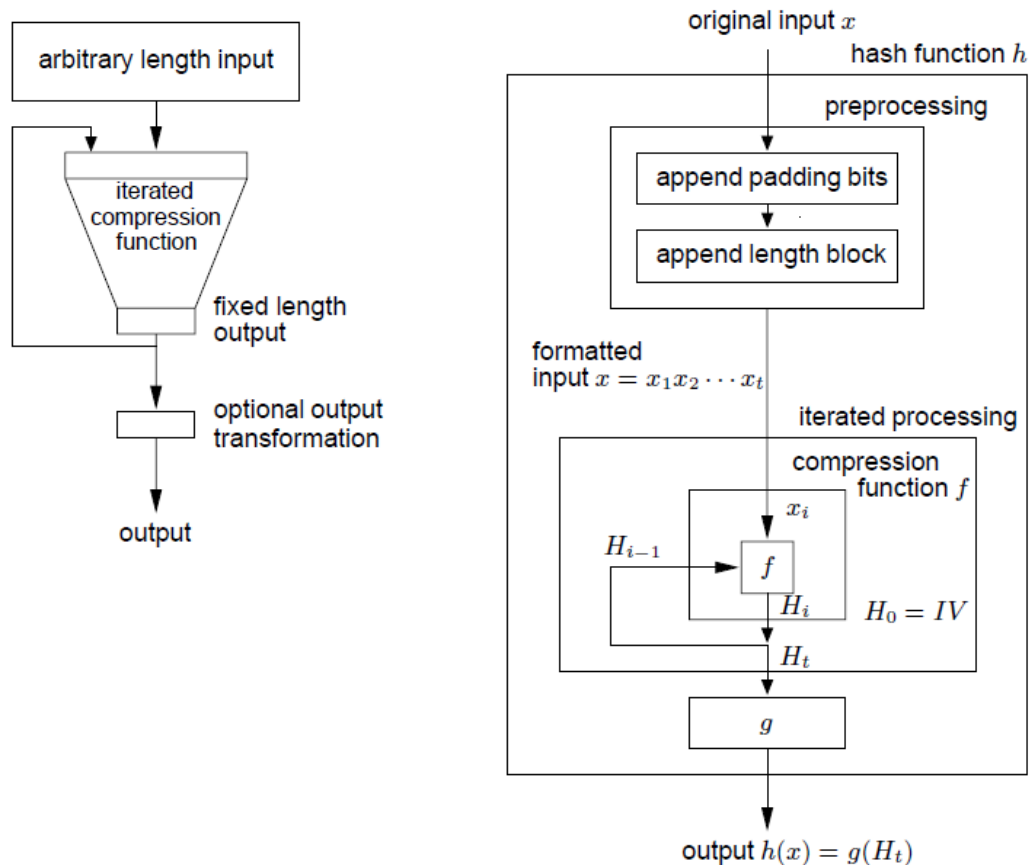
Hash Function

Basic constructions and general results

agung.nursilo¹ | jupri rahman²
www.noersilo.web.id¹ | www.jr.web.id²
noersilo@yahoo.com¹ | adempiliang@yahoo.com²

I. Model Umum untuk Fungsi Hash Berulang

Kebanyakan *unkeyed hash function* didesain dengan proses perulangan yang panjang inputnya tidak tetap dengan memproses blok input dengan panjang tetap.



Sebuah input hash x yang memiliki panjang yang tidak tetap, dibagi kedalam r bit block x_i dengan panjang tetap. Dalam proses tersebut melibatkan penambahan (padding) bit yang diperlukan untuk memenuhi panjang bit keseluruhan yang merupakan perkalian dari panjang block r dan berisi block atau sebagian dari block yang menunjukkan panjang bit yang tidak *padding*. Setiap block x_i akan masuk sebagai input pada fungsi hash f , fungsi kompresi h , dan hasil dari kompresi tersebut akan menjadi inputan bersama inputan berikutnya. Berikut prosesnya :

$$H_0 = IV; \quad H_i = f(H_{i-1}, x_i), \quad 1 \leq i \leq t; \quad h(x) = g(H_t).$$

H_{i-1} berfungsi sebagai chaining n bit variabel antara tahap $i - 1$ dan tahap i , dan H_0 adalah nilai dari IV. Transformasi output g digunakan dalam langkah terakhir untuk memetakan n -bit variabel chaining m -bit sebuah hasil g (H_t); g sering pemetaan identitas $g(H_t) = H_t$.

Fungsi hash dibedakan oleh sifat preprocessing, fungsi kompresi, dan output transformasi.

II. Konstruksi Dasar dan Perluasan

Contoh perluasan ketidakamanan OWHF ke CRHF. Sebuah perulangan H OWHF menghasilkan n -bit nilai hash yang tidak collusion resistant (misalnya, ketika $2n/2$ birthday collusion attack dapat memungkinkan) kita dapat menggunakan h dari CRHF menggunakan sebagai output rangkaian terakhir dari dua n -bit variabel chaining, sehingga t -blok pesan nilai hash $H_{t-1} || H_t$ daripada H_t . Ini mengurangi masalah untuk menemukan sebuah tabrakan pada H_{t-1} untuk h .

Setiap fungsi f kompresi yang collision resistant dapat diperluas ke collision resistant fungsi hash h (mengambil panjang input tidak tetap). Hal ini menyatakan adanya hubungan yang penting antara collision resistant compression function dan collision resistant hash function. Perluasan dapat dilakukan secara efisien dengan menggunakan metode algoritma meta Merkle Hal ini akan mengurangi masalah seperti menemukan fungsi hash dengan menemukan seperti fungsi kompresi.

Algoritma Merkle's meta - metode untuk hashing

INPUT: fungsi f kompresi yang collision resistant

OUTPUT: *unkeyed hash function* h yang collision resistant

1. Misalkan pemetaan f $(n + r)$ -bit input untuk n -bit output (misal $n = 128$ dan $r = 512$). Buatlah sebuah fungsi hash h dari f , menghasilkan n -bit nilai hash.
2. Pecah input x dari bitlength b ke blok $x_1 x_2 \dots x_t$ setiap panjang bit r , padding blok terakhir x_t dengan 0-bit jika diperlukan .
3. Tentukan ekstra blok terakhir x_{t+1} , panjang-blok.
4. Jadikan 0^j mewakili bitstring dari j 0's, menentukan n -bit hash-nilai x yang akan $h(x) = H_{t+1} = f(H_t || x_{t+1})$ dihitung dari:

$$H_0 = 0_n; \quad H_i = f(H_{i-1} || x_i), \quad 1 \leq i \leq t + 1.$$

Bukti bahwa fungsi yang dihasilkan h adalah collision resistant berikut dengan argumen sederhana bahwa tabrakan untuk h akan berpengaruh untuk f untuk tahap i . Dimasukkannya panjang-blok, yang secara efektif meng-encode semua pesan yang dikodekan sedemikian sehingga tidak ada input diujung input lainnya, Menambahkan panjang blok yang disebut Merkle-Damgard Strengthening (MD-strengthening).

Algoritma MD-strengthening

Sebelum hashing pesan $x = x_1x_2 \dots x_t$ (di mana x_i adalah blok r bitlength sesuai untuk fungsi kompresi yang relevan) dari bitlength b , tambahkan panjang blok terakhir, x_{t+1} , mengandung (katakanlah) biner kanan b . (Hal ini mengandaikan $b < 2r$.)

Cascading fungsi hash

Jika salah satu h_1 atau h_2 adalah collision hash function, maka $h(x) = h_1(x) || h_2(x)$ adalah collision resistant hash function. Jika h_1 dan h_2 , n -bit adalah fungsi hash, lalu h $2n$ -bit menghasilkan output; pemetaan ini kembali ke n -bit output dengan n -bit collision resistant hash function (h_1 dan h_2 adalah kandidat) akan meninggalkan pemetaan keseluruhan collision resistant. Jika h_1 dan h_2 adalah independen, kemudian menemukan sebuah tabrakan untuk h perlu ditemukan sebuah tabrakan untuk kedua secara simultan (yaitu, input yang sama), yang mana yang bisa berharap akan membutuhkan produk dari upaya untuk menyerang mereka secara individual. Ini menyediakan cara yang sederhana namun kuat untuk (hampir pasti) meningkatkan kekuatan hanya menggunakan komponen yang tersedia.

III. Detail Pemformatan dan Inisialisasi

Karena nilai hash tergantung pada bit string yang tepat, representasi data yang berbeda (misalnya, ASCII vs. EBCDIC) harus dikonversikan ke format yang biasa sebelum menghitung nilai hash.

Padding dan panjang block

Untuk metode *hashing* block per block, bit ekstra biasanya ditambahkan ke input string hash sebelum *hashing*, untuk menambahkannya menjadi sejumlah bit yang merupakan kelipatan dari ukuran block yang relevan. Bit padding tidak perlu ditransmisikan/ disimpan sendiri, asalkan pengirim dan penerima setuju pada suatu konvensi.

Algoritma Metode Padding 1

INPUT: data x ; ukuran blok data input bit dengan panjang n ke tahap pengolahan.

OUTPUT: data x' yang sudah *dipad*, dengan panjang bit kelipatan n .

1. Tambahkan ke x sedikit (bahkan nol) bit 0 untuk memperoleh string x' yang panjang bitnya merupakan kelipatan n .

Algoritma Metode Padding 2

INPUT: data x ; ukuran blok data input bit dengan panjang n ke tahap pengolahan.

OUTPUT: data x' yang sudah *dipad*, dengan panjang bit kelipatan n .

1. Tambahkan ke x satu bit 1.

2. Kemudian tambahkan sedikit (bahkan nol) bit 0 untuk memperoleh string x' yang panjang bitnya merupakan kelipatan n .

Metode Padding 1 ambigu – bit 0 di belakang data asli tidak dapat dibedakan dari data yang ditambahkan selama *padding*. Metode ini dapat diterima jika panjang data (sebelum *padding*) diketahui oleh penerima dengan cara lain. Metode Padding 2 tidak ambigu – setiap string x' yang telah *dipad* each cocok dengan string x unik yang tidak *dipad*. Ketika panjang bit dari data asli x sudah merupakan kelipatan n , maka Metode Padding 2 hasil pembuatan blok ekstra.

Penambahan panjang block sebelum hash mencegah terjadinya serangan collision dan pseudocollision yang menemukan pesan kedua dari panjang berbeda, termasuk trivial collisions untuk IVacak, serangan pesan panjang, dan serangan titik tetap.

IVs

Jika IV tetap, dipilih secara acak per komputasi fungsi hash, atau merupakan fungsi data input, IV yang sama harus digunakan untuk membangkitkan dan memverifikasi nilai hash. Jika tidak diketahui sebelumnya oleh yang memverifikasi, sebelumnya harus ditransfer dengan pesan.

IV. Sasaran Keamanan dan Serangan Dasar

Sebagai framework untuk mengevaluasi keamanan komputasi fungsi hash, tujuan perancang dan fungsi hash dan lawan harus dimengerti. Tabel berikut berisi tujuan desain untuk n bit fungsi hash (kunci MAC t bit). P_f menandakan probabilitas lawan menebak MAC dengan benar.

Hash type	Design goal	Ideal strength	Adversary's goal
OWHF	preimage resistance; 2nd-preimage resistance	2^n 2^n	produce preimage; find 2nd input, same image
CRHF	collision resistance	$2^{n/2}$	produce any collision
MAC	key non-recovery; computation resistance	2^t $P_f = \max(2^{-t}, 2^{-n})$	deduce MAC key; produce new (msg. MAC)

Diberikan fungsi hash tertentu, diharapkan dapat membuktikan batas bawah kompleksitas serangan di bawah skenario tertentu, dengan sedikit atau sekumpulan asumsi lemah mungkin. Walaupun demikian, hasil seperti itu langka. Biasanya pedoman terbaik yang tersedia mengenai keamanan fungsi hash tertentu adalah kompleksitas (paling efisien) serangan yang sudah diketahui yang dapat diaplikasikan, yang memberikan batas atas pada keamanan. Serangan *kompleksitas* 2^t adalah salah satu yang membutuhkan kira-kira 2^t operasi, masing-masingnya satuan kerja yang cocok (misalnya satu eksekusi fungsi kompresi atau satu enkripsi *cipher* yang mendasari). Kompleksitas penyimpanan serangan (misalnya penyimpanan yang dibutuhkan) juga harus dipertimbangkan.

Serangan pada seukuran bit MDC

Diberikan pesan x dengan panjang tetap dengan n bit hash $h(x)$, metode naïf untuk mendapatkan sebuah input yang bertabrakan dengan x adalah mengambil sembaran bit string x' (dari batas panjang bit) dan mengecek jika $h(x')=h(x)$. Biayanya mungkin sedikit sebagaimana evaluasi fungsi kompresi dan memori dapat

diabaikan. Dengan asumsi kode hash mendekati variable acak seragam, probabilitas pasangannya adalah 2^{-n} . Implikasinya adalah The implication of this is Fact 9.33, which also indicates the effort required to find collisions if x may itself be chosen freely. Definition 9.34 is motivated by the design goal that the best possible attack should require no less than such levels of effort, i.e., essentially brute force.

Untuk n bit fungsi hash h , seseorang bisa berharap serangan dengan tebakan untuk menemukan *preimage* atau *2nd preimage* dalam 2^n operasi hash. Selama lawan dapat memilih pesan, *birthday attack* memungkinkan menabrakan pasangan pesan x , x' dengan $h(x) = h(x')$ untuk ditemukan di sekitar $2^{n/2}$ operasi dan memori yang dapat diabaikan.

Sebuah n bit An n -bit unkeyed hash function memiliki *keamanan ideal* jika (1) diberikan sebuah output hash, untuk menghasilkan setiap preimage dan 2nd preimage membutuhkan kira-kira 2^n operasi; dan (2) untuk menghasilkan tabrakan membutuhkan kira-kira $2^{n/2}$ operasi.

Serangan pada Ruang Kunci MAC

Usaha yang mungkin untuk menentukan (mendapatkan) kunci MAC adalah menggunakan *exhaustive search*. Dengan sebuah pasangan text MAC yang diketahui, seorang penyerang dapat menghitung n bit MAC pada text tersebut dengan semua kemungkinan kunci dan kemudian mengecek dilai MAC yang mana yang cocok dengan pasangan yang diketahui tadi. Untuk t bit ruang kunci membutuhkan 2^t operasi MAC

Serangan pada ukuran bit MAC

Pemalsuan MAC melibatkan pembuatan input x sebarang dan MAC yang benar berkorepondensi tanpa mendapatkannya dari orang yang tahu akan kunci. Untuk n bit algoritma MAC, baik menebak MAC untuk input yang diberikan maupun menebak *preimage* mempunyai probabilitas sukses sekitar 2^{-n} , sebagaimana pada MDC. Perbedaannya, menebak nilai MAC tidak dapat diverifikasi offline tanpa mengetahui pasangan text MAC atau kuncinya atau “black-box” yang menyediakan MAC untuk input dibutuhkan.

9.3.5 Ukuran Bit yang dibutuhkan untuk keamanan praktis

Anggap sebuah fungsi hash menghasilkan n bit nilai hash dan *representative benchmark* berasumsi bahwa 2^{80} (tapi tidak lebih sedikit) operasi dapat diterima di atas kemampuan komputasi. Maka pernyataan berikut dapat berlaku pada n .

1. Untuk OWHF, $n \geq 80$ dibutuhkan. *Exhaustive off-line attack* membutuhkan setidaknya 2^n operasi; hal ini dapat dikurangi dengan komputasi awal.
2. Untuk CRHF, $n \geq 160$ dibutuhkan. *Birthday attack* dapat diterapkan
3. Untuk MAC, $n \geq 64$ dengan kunci MAC 64-80 bit cukup untuk kebanyakan aplikasi dan *environment*

Source : Handbook of Applied Cryptography
www.noersilo.web.id