Analisis dan Implementasi CubeHash

Stephen Herlambang - 13507040[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]norbert@students.itb.ac.id

Abstract*— Fungsi hash adalah fungsi yang mengubah pesan dengan panjang bervariasi menjadi suatu message digest yang pendek yang dapat digunakan untuk digital signature, message authentication, dan aplikasi-aplikasi lainnya. CubeHash adalah fungsi hash yang diikutsertakan dalam kompetisi NIST hash function oleh Daniel J. Bernstein. CubeHash r/b-h menghasilkan digest h-bit dengan pesan dengan panjang bervariasi. Makalah ini akan membahas mengenai konsep, analisis, implementasi, dan beberapa eksperimen dari algoritma CubeHash.

***Index Terms*—CubeHash, SHA-3, Hash**

I. PENDAHULUAN

Sebuah fungsi hash adalah prosedur deterministik yang mengambil suatu blok data dan mengembalikan bit string berukuran tetap yaitu nilai hash sehingga perubahan akan data akan mengubah nilai hash. Data yang akan di-*encode* sering disebut pesan (*message*), dan nilai hash disebut *digest*. Fungsi hash dalam kroptografi memiliki banyak aplikasi keamanan informasi, terutama dalam tanda tangan digital, kode autentikasi pesan (MAC), dan bentuk-bentuk otentikasi. Mereka juga dapat digunakan sebagai fungsi hash biasa, data indeks dalam tabel hash, untuk sidik jari, untuk mendeteksi data ganda atau mengidentifikasi file unik, dan sebagai checksum untuk mendeteksi data yang rusak (*corrupt*).

Salah satu fungsi hash yang terstandarisasi dan banyak digunakan adalah fungsi hash SHA (Secure Hash Algorithm). SHA adalah salah satu gungsi hash kriptografik yang dipublikasikan oleh National Insitute of Standards and Technology (NIST). Saat ini terdapat tiga generasi dari Secure Hash Algorithm tersebut, yaitu SHA-1, SHA-2, dan SHA-3.

SHA-3 saat ini masih dalam proses pengembangan. Lebih tepatnya, sedang terdapat kompetisi untuk menentukan algoritma yang paling cocok untuk dipakai sebagai SHA-3 ini. Kompetisi ini adalah *NIST hash function competition*. Kompetisi ini mirip dengan proses pengembangan dari *Advanced Encryption Standard (AES)*.

Kompetisi ini dimulai dengan pengumpulan pertama pada 31 Oktober 2008 dengan pengumuman kadidat yang lolos untuk ronde pertama pada 9 Desember 2008. Lalu, pada 24 Juli 2009, terdapat 14 kandidat yang lolos untuk masuk ke ronde 2. Salah satu dari peserta yang lolos ini adalah CubeHash. CubeHash adalah algoritma yang diajukan oleh Daniel J. Bernstein.

II. CUBEHASH

CubeHash r/b-h menghasilkan digest h-bit dengan pesan dengan panjang bervariasi. Digest bergantung pada dua parameter yaitu r dan b yang memungkinkan pemilihan rentang *tradeoff* antara sekuritas dan performansi. [1]

A. Cara Kerja

Masukan dari CubeHash adalah:

- parameter *r* dalam {1,2,3,...}, yaitu jumlah *round*, biasanya 16
- parameter *b* dalam {1,2,3,...,128}, jumlah bytes tiap blok pesan, biasanya 32
- parameter *h* dalam {8,16,24,...,512}, jumlah bit keluaran, biasanya 512
- pesan (*message*) *m*, string dengan ukuran antara 0 bit dan 2^{128}-1 bit.

CubeHash r/b-h memiliki lima langkah utama:

1. Inisialisasi *state* 128-byte (1024 bit) sebagai fungsi dari (h, b, r)
2. Konversi pesan masukan menjadi *padded message*, yang terdiri dari satu atau lebih blok sebesar b-byte.
3. Untuk setiap blok b-byte pada *padded message*, block di-xor dengan b-byte pertama dari *state*, dan kemudian *state* ditransformasikan melalui sejumlah r *round* yang identik.
4. Finalisasi (*finalize*) *state*
5. h/8 byte pertama dari state dipakai sebagai keluaran (*output*).

B. Inisialisasi

Inisialisasi bekerja dengan cara sebagai berikut:

1. State sebesar 128-byte tersebut dilihat sebagai sekuens dari 32 buah *word* sebesar 4-byte : x_{00000}, x_{00001}, x_{00010}, x_{00011}, x_{00100}, ... , x_{11111}, dan tiap word diinterpretasikan sebagai integer 32-bit dalam bentuk *little-endian*.

2. Semua *word* dari state (x_{00000}, x_{00001}, $x_{00010,}$, x_{00100}, … , x_{11111}) masing-masing diinisialisasi.
3. x_{00000} diisi dengan nilai h/8.
4. x_{00001} diisi dengan nilai b.
5. x_{00010} diisi dengan nilai r.
6. x_{00100}, … , x_{11111} masing-masing diisi dengan 0.
7. State lalu ditransformasikan sebanyak 10r ronde.

C. Padding

Proses padding bekerja dengan cara sebagai berikut:

1. Pesan masukan ditambahkan dengan bit 1.
2. Pesan lalu ditambahkan lagi dengan 0 bit seminimum mungkin sampai tercapai panjang dengan kelipatan 8b bit.

Proses padding ini tidak memerlukan penyimpanan terpisah untuk panjang pesan, blok yang akan diproses, dan sebagainya. Implementasi memungkinkan hanya digunakannya satu buah integer antara 0 dan 8b untuk mencatat jumlah bit yang telah diproses dalam blok saat ini.

D. Finalisasi

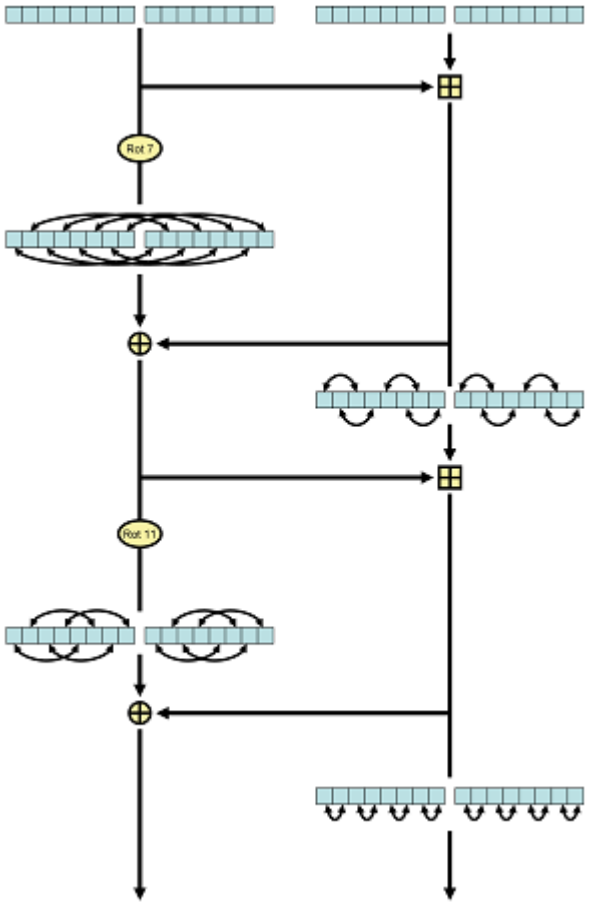
Finalisasi bekerja dengan cara sebagai berikut. *Word* terakhir pada state (x_{11111}) di-xor dengan 1. State kemudian ditransformasikan melalui sejumlah 10r ronde. Setelah itu keluaran yang dihasilkan adalah h/8 byte pertama dari state.

E. Transformasi

Setiap ronde (*round*) memiliki tahapan sebagai berikut:

1. Tambahkan x_{0jklm} ke x_{1jklm} modulo 2^{32}, untuk setiap (j, k, l, m).
2. Putar x_{0jklm} ke atas sebesar 7 bit, untuk setiap (j, k, l, m).
3. Tukar x_{00klm} dengan x_{01klm}, untuk setiap (k, l, m).
4. Xor x_{1jklm} ke x_{0jklm}, untuk setiap (j, k, l, m).
5. Tukar x_{1jk0m} dengan x_{1jk1m}, untuk setiap (j, k, m).
6. Tambahkan x_{0jklm} ke x_{1jklm} modulo 2^{32}, untuk setiap (j, k, l, m).
7. Putar x_{0jklm} ke atas sebesar 11 bit, untuk setiap (j, k, l, m).
8. Tukar x_{0j0lm} dengan x_{0j1lm}, untuk setiap (j, l, m).
9. Xor x_{1jklm} ke x_{0jklm}, untuk setiap (j, k, l, m).
10. Tukar x_{1jkl0} dengan x_{1jkl1}, untuk setiap (j, k, l).

Tahapan tersebut dapat digambarkan sebagai berikut:



Gambar 1 Transformasi internal pada CubeHash [3]

III. ANALISIS SERANGAN TERHADAP CUBEHASH

CubeHash r/b akan menjadi mudah untuk diserang jika nilai b terlampau besar. Sebagai contoh, CubeHash r/112 mengizinkan penyerang untuk menggunakan blok 112-byte terakhir dari masukan untuk mengatur 112 dari 128 byte *state*. Modifikasi pesan blok terakhir ini secara efektif mengurangi pengaruh dari blok masukan sebelumnya menjadi *"narrow pipe"* sebesar 16 byte, sehingga penyerang memiliki kemungkinan yang cukup besar untuk mendapatkan collision setelah melakukan iterasi hanya sebanyak 2^{64} pesan masukan. Serangan ini independen terhadap detil transformasi state, dan independen terhadap r.

Tetapi, penyerang akan kehilangan control jika nilai b menurun. Serangan umum memiliki kemungkinan sukses yang secara eksponen menurun dengan "*pipe size*" sebesar 1024 - 8b. Sebagai contoh, CubeHash r/32 memiliki pipe 768-bit, sehingga serangan akan sulit dilakukan.

Penyerang dapat mencoba melakukan serangan yang lebih baik dengan menggunakan serangan yang tidak umum (*non-generic*). Satu ronde pada transformasi CubeHash tidak terlalu kompleks. Kasus ekstrim pada CubeHash 1/b memiliki masukan satu bit yang hanay mempengaruhi sekitar sepuluh bit dari state pada awal blok berikutnya. Bagaimanapun, struktur *hypercube* dari

CubeHash mendistribusikan bit-bit tersebut secara meluas, dan jika b bernilai kecil penyerang tidak dapat mencegah defferential untuk menyebar ke seluruh state melalui beberapa blok berikutnya.

Sepuluh *internal round* pada CubeHash adalah filter keluaran, secara menyeluruh mencampurkan (*mixing*) bit-bit state sebelum pemotongan (*truncation*) terakhir untuk mendapatkan panjang keluaran yang diinginkan. Sepuluh ronde tampaknya telah sangat cukup untuk menyediakan proteksi penuh terhadap serangan linear (*linear attack*), serangan diferensial (*differential attack*), dan sebagainya. Dengan membalik bit state tambahan sebelum ronde terakhir ini, CubeHash membedakan blok terakhir pesan dari blok sebelumnya, dan dengan begitu membuat *slide attack*, *length-extension attack*, dan semacamnya sama sulitnya seperti *standard differential attack*. [2]

IV. IMPLEMENTASI CUBEHASH

Implementasi dilakukan pada bahasa C#. Hasil implementasi adalah sebagai berikut:

```
class hashState{
  int hashbitlen;
  int pos;
  myuint32[] x = new myuint32[32];
};
```

Kelas *hashState* ini merepresentasikan state, *hashbitlen* adalah panjang hash dalam bit, *pos* adalah posisi bit yang dibaca pada blok ini, dan *x* adalah isi state sebesar 32 *word* (*word* direpresentasikan dengan *unsigned integer 32*).

```
void transform(hashState state)
{
  int i;
  int r;
  myuint32[] y = new myuint32[16];

  for (r = 0; r < CUBEHASH_ROUNDS; ++r) {
    for (i = 0; i < 16; ++i)
        state.x[i + 16] += state.x[i];
    for (i = 0; i < 16; ++i)
        y[i ^ 8] = state.x[i];
    for (i = 0; i < 16; ++i)
        state.x[i] = ROTATE(y[i],7);
    for (i = 0; i < 16; ++i)
        state.x[i] ^= state.x[i + 16];
    for (i = 0; i < 16; ++i)
        y[i ^ 2] = state.x[i + 16];
    for (i = 0; i < 16; ++i)
        state.x[i + 16] = y[i];
    for (i = 0; i < 16; ++i)
        state.x[i + 16] += state.x[i];
    for (i = 0; i < 16; ++i)
        y[i ^ 4] = state.x[i];
    for (i = 0; i < 16; ++i)
        state.x[i] = ROTATE(y[i],11);
    for (i = 0; i < 16; ++i)
        state.x[i] ^= state.x[i + 16];
    for (i = 0; i < 16; ++i)
        y[i ^ 1] = state.x[i + 16];
    for (i = 0; i < 16; ++i)
        state.x[i + 16] = y[i];
  }
}
```

Prosedur ini digunakan untuk transformasi state. Di dalam prosedur inilah dilakukan sejumlah round (*cubehash_rounds*) transformasi terhadap state yang diberikan.

```
HashReturn Init(hashState state, int hashbitlen)
{
    int i;

    if (hashbitlen < 8)
        return HashReturn.BAD_HASHBITLEN;
    if (hashbitlen > 512)
        return HashReturn.BAD_HASHBITLEN;
    if (hashbitlen != 8 * (hashbitlen / 8))
        return HashReturn.BAD_HASHBITLEN;

    state.hashbitlen = hashbitlen;
    for (i = 0;i < 32;++i) state.x[i] = 0;
    state.x[0] = (myuint32)(hashbitlen / 8);
    state.x[1] = CUBEHASH_BLOCKBYTES;
    state.x[2] = CUBEHASH_ROUNDS;
    for (i = 0;i < 10;++i) transform(state);
    state.pos = 0;
    return HashReturn.SUCCESS;
}
```

Prosedur ini digunakan untuk inisialisasi yaitu pengecekan nilai panjang *hashbitlen*, lalu pengisian *state* awal, dan tranformasi sebanyak 10 kali.

```
HashReturn Update(hashState state, Byte[] data,
DataLength databitlen)
{
  int v = 0;
  while (databitlen >= 8) {
    myuint32 u = data[v];
    u <<= 8 * ((state.pos / 8) % 4);
    state.x[state.pos / 32] ^= u;
    v++;
    databitlen -= 8;
    state.pos += 8;
    if (state.pos == 8 * CUBEHASH_BLOCKBYTES) {
      transform(state);
      state.pos = 0;
    }
  }
  if (databitlen > 0) {
    myuint32 u = data[v];
    u <<= 8 * ((state.pos / 8) % 4);
    state.x[state.pos / 32] ^= u;
    state.pos += (int)databitlen;
  }
  return HashReturn.SUCCESS;
}
```

Fungsi yang merepresentasikan untuk setiap *byte* pada data, block di-xor dengan b-byte pertama dari *state*, dan kemudian *state* ditransformasikan.

```
HashReturn Final(hashState state, Byte[]
hashval)
{
  int i;
  myuint32 u;

  u = (myuint32)(128 >> (state.pos % 8));
  u <<= 8 * ((state.pos / 8) % 4);
  state.x[state.pos / 32] ^= u;
  transform(state);
  state.x[31] ^= 1;
  for (i = 0;i < 10;++i) transform(state);
  for (i = 0; i < state.hashbitlen / 8; ++i)
hashval[i] = (Byte)(state.x[i / 4] >> (8 * (i %
4)));
  return HashReturn.SUCCESS;
}
```

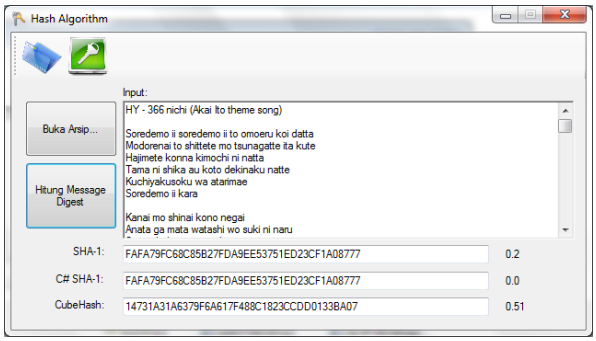
Fungsi yang merepresentasikan finalisasi, yaitu word

terakhir pada state di-xor dengan 1, kemudian state ditransformasikan sebanyak 10r ronde, dan keluaran yang dihasilkan adalah h/8 byte pertama dari state (disimpan dalam *hashval*).

```
public Byte[] Padding(Byte[] data)
{
    int length = data.Length + 1;
    int b = (int)CUBEHASH_BLOCKBYTES;
    if (b > 1) length += (b-(length % b));
    Byte[] result = new Byte[length];
    int i = 0;
    while (i < data.Length)
    {
        result[i] = data[i];
        i++;
    }
    result[i] = 128;
    i++;
    while (i < length)
    {
        result[i] = 0;
        i++;
    }
    return result;
}
```

Fungsi di atas digunakan untuk proses padding pada message (*data* sebagai *array of Byte*). Fungsi ini menambahkan 1000000 (1 yang diikuti 0, direpresentasikan dengan byte 128) dan sejumlah 0 seminimum mungkin.

Tampilan antarmuka aplikasi adalah seperti di bawah ini:



Gambar 2 Tampilan Antarmuka Aplikasi

V. EKSPERIMEN CUBEHASH

Eksperimen dilakukan pada computer dengan prosesor AMD Turion 64x2 2.2 GHz (setara dengan Intel Core2Duo).

A. Perbandingan dengan SHA-1

Eksperimen pertama dilakukan dengan membandingkan CubeHash dengan algoritma lainnya dalam hal waktu pemrosesan. Algoritma pembanding yang digunakan adalah SHA-1 hasil implementasi dan SHA-1 yang telah disediakan oleh library C#.

Algoritma CubeHash diuji coba dengan menggunakan nilai r = 10, b = 32, dan h=160. Nilai h ditetapkan sejumlah 160 bit agar dapat dibandingkan dengan algoritma SHA-1.

Hasil eksperimen pertama dapat dilihat pada tabel di bawah ini:

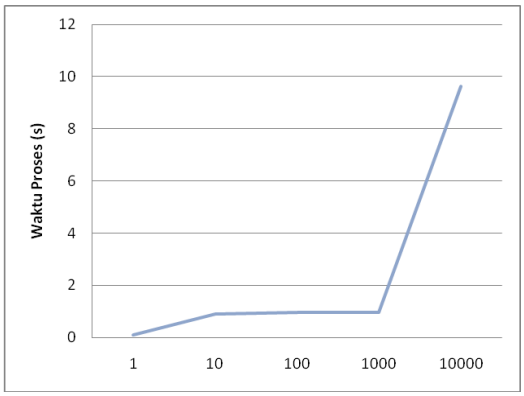
Tabel 1 Perbandingan CubeHash dan SHA-1

Besar File (kB)	Waktu Proses (s)		
	SHA-1	CubeHash	SHA-1 C#
1	0.1	0.1	0
10	0.7	0.9	0.1
100	0.78	0.95	0.1
1000	0.864	0.945	0.4
10000	9.68	9.634	0.5

Dari hasil eksperimen, dapat dilihat bahwa CubeHash algoritma hasil implementasi memiliki performansi yang kurang baik jika dibandingkan dengan algoritma SHA-1 yang disediakan oleh C#. Hal ini mungkin diakibatkan implementasi algoritma yang masih belum terlalu efisien.

Dari kedua algoritma hasil implementasi, terlihat bahwa tidak terdapat perbedaan yang sangat mencolok antara keduanya dari segi waktu proses. Waktu pemrosesan antara CubeHash dan SHA-1 meningkat sesuai dengan penambahan besar file yang diproses. Laju peningkatan antara kedua algoritma yang dibandingkan pun kurang lebih memiliki nilai yang sama.

Laju peningkatan waktu proses pada algoritma CubeHash dapat dengan lebih jelas direpresentasikan dengan grafik sebagai berikut:



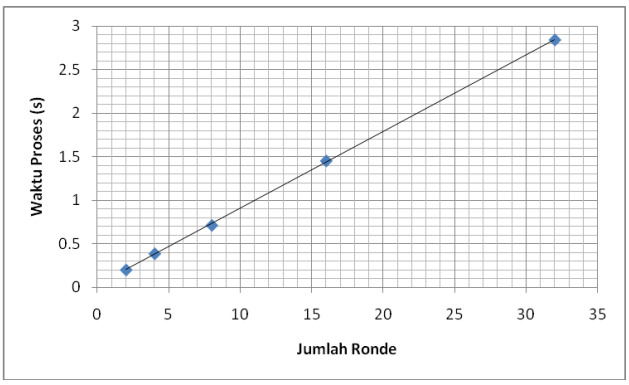
Gambar 3 Grafik Waktu Proses terhadap Besar File pada Algoritma CubeHash

Dari grafik dapat terlihat bahwa waktu proses dari algoritma CubeHash meningkat secara proporsional terhadap besar file.

B. Pengaruh Jumlah Ronde terhadap Waktu Proses

Eksperimen kedua dilakukan untuk menguji seberapa besar pengaruh jumlah ronde terhadap waktu proses pada algoritma CubeHash. Besar file yang digunakan untuk uji coba adalah 1000 kB.

Hasil eksperimen kedua dapat dilihat dengan representasi grafik di bawah ini:



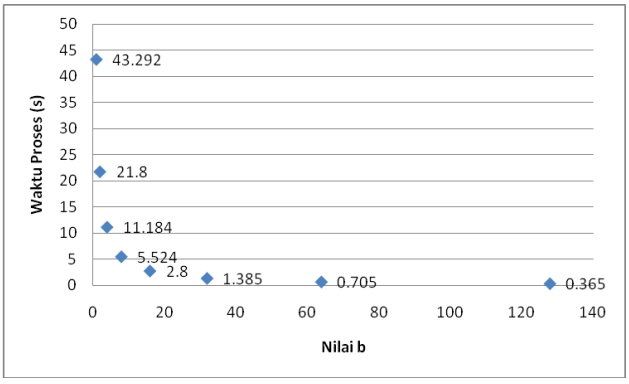
Gambar 4 Grafik Waktu Proses terhadap Jumlah Ronde pada Algoritma CubeHash

Dari grafik dapat dilihat bahwa peningkatan jumlah ronde akan berpengaruh terhadap meningkatnya waktu proses. Laju peningkatan waktu proses ini berbanding lurus terhadap peningkatan jumlah ronde yang digunakan pada algoritma CubeHash ini.

C. Pengaruh Nilai Parameter b terhadap Waktu Proses

Eksperimen ketiga dilakukan untuk menguji seberapa besar pengaruh nilai parameter b terhadap waktu proses pada algoritma CubeHash. Besar file yang digunakan untuk uji coba adalah 1000 kB. Nilai r tetap yaitu 16. Nilai b divariasikan dengan nilai 1, 2, 4, 8, 16, 32, 64, dan 128.

Hasil eksperimen ketiga dapat dilihat dengan representasi grafik di bawah ini:



Gambar 5 Grafik Waktu Proses terhadap Nilai Parameter b pada Algoritma CubeHash

Dari grafik dapat dilihat bahwa peningkatan jumlah nilai b akan berpengaruh terhadap menurunnya waktu proses. Laju penurunan waktu proses ini berbanding secara eksponensial terhadap peningkatan jumlah ronde yang digunakan pada algoritma CubeHash ini.

D. Pengaruh Modifikasi Satu Karakter

Eksperimen berikutnya adalah uji coba pengaruh pengubahan satu karakter terhadap hasil hash dari algoritma CubeHash ini.

String yang diuji coba adalah "Analisis dan Implementasi CubeHash".

Hasil hash dalam representasi heksa-desimal dari string tersebut adalah:
CFDCD8A5C3CED07598AF27D23FD876DC976A9A2A

Pengubahan 1 karakter pada string, sehingga string menjadi "Analisis dan Implementasi Cubehash" (Hash menjadi hash) akan membuat nilai hash menjadi:
AF34FA516B70D88E3361AE3F3FC8FAA1CA0D9A47

Penambahan 1 karakter pada string, sehingga string menjadi "Analisis dan Implementasi Cube Hash" (penambahan spasi antara Cube dan Hash) akan membuat nilai hash menjadi:
B13847D0D9A19750E066CFAC5B052D4746AE5284

Penghilangan 1 karakter pada string, sehingga string menjadi "Analisis dan Implementasi CubHash" (penghilangan karakter 'e' pada CubeHash) akan membuat nilai hash menjadi:
069787E3AB7A0E2EB757F89F4C7E9DCF4F78C9A7

Dari hasil eksperimen ini, dapat dilihat bahwa pengubahan, penambahan, dan penghilangan (*modify*, *add*, dan *delete*) satu karakter pada string asal (sebagai *message*) akan mengubah nilai hash menjadi jauh berbeda terhadap nilai hash dari pesan asal.

E. Perbandingan Variasi CubeHash

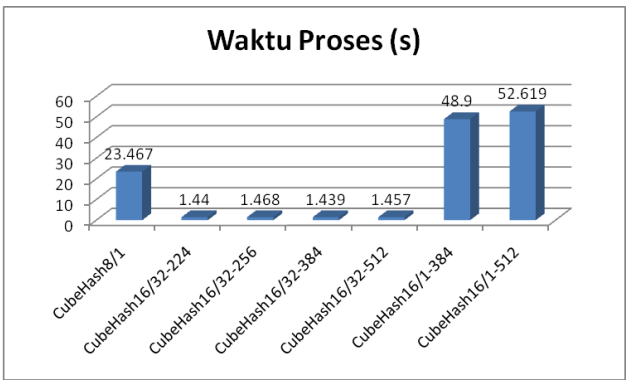
CubeHash yang diajukan sebagai calon SHA-3 pada kompetisi NIST adalah CubeHash8/1 (CubeHash dengan nilai r=8, dan b=1).

Sebenarnya terdapat beberapa variasi lainnya yang juga diajukan oleh pembuat CubeHash (Daniel J. Bernstein). Variasi-variasi terseubt adalah sebagai berikut:

- CubeHash16/32-224 untuk SHA-3-224,
- CubeHash16/32-256 untuk SHA-3-256,
- CubeHash16/32-384 untuk SHA-3-384-normal,
- CubeHash16/32-512 untuk SHA-3-512-normal,
- CubeHash16/1-384 untuk SHA-3-384-formal, dan
- CubeHash16/1-512 untuk SHA-3-512-formal.

Menurut Daniel J. Bernstein, CubeHash16/32 kira-kira 16 kali lebih cepat daripada CubeHash8/1, dengan mudah menyusul baik SHA-256 dan SHA-512. Meskipun cukup cepat, CubeHash16/32 tetap memiliki rentang sekuritas yang cukup baik. [1]

Eksperimen dilakukan dengan menggunakan file berukuran 1000 kB. Hasil eksperimen dapat dilihat dengan grafik sebagai berikut:



Gambar 6 Hasil Perbandingan Variasi CubeHash

Berdasarkan hasil eksperimen yang ditunjukkan pada grafik, dapat dilihat bahwa waktu proses pada CubeHash16/32 (dari keempat varian) memiliki rata-rata waktu proses sebesar 1.451 detik, sedangkan CubeHash8/1 memiliki waktu proses sebesar 23.467 detik.

Dengan begitu, waktu proses CubeHash16/32 pada eksperimen (23.467 / 1.451) = 16.173 kali lebih cepat daripada CubeHash8/1. Dengan begitu, hasil eksperimen cukup sesuai dengan pendapat Daniel J. Bernstein (kira-kira 16 kali lebih cepat).

VI. KESIMPULAN

CubeHash merupakan salah satu fungsi hash yang cukup sederhana dan cukup efektif. Algoritma ini memiliki tingkat proteksi yang cukup baik terhadap serangan-serangan yang ada.

Waktu proses dari algoritma CubeHash meningkat secara proporsional sesuai besar file (*message*) yang diproses, berbanding lurus terhadap nilai jumlah ronde dari transformasi (r), dan berbanding terbalik secara eksponensial terhadap nilai parameter b. Modifikasi terhadap pesan asli sebesar 1 karakter mengubah nilai hash pada algoritma CubeHash menjadi jauh berbeda terhadap nilai hash dari pesan asal. Terdapat beberapa variasi dari CubeHash sesuai dengan perubahan parameter r, b, dan h. Varian CubeHash16/32 terbukti kira-kira 16 kali lebih cepat daripada CubeHash8/1.

REFERENSI

[1] Daniel J. Bernstein, CubeHash specification (2.B.1), http://cubehash.cr.yp.to/submission2/spec.pdf, Waktu Akses: 28 April 2010, 21.54.

[2] Daniel J. Bernstein, CubeHash attack analysis (2.B.5), http://cubehash.cr.yp.to/submission/attacks.pdf, Waktu Akses: 16 Mei 2010, 20.34.

[3] Eric Brier and Thomas Peyrin, Cryptanalysis of CubeHash, http://thomas.peyrin.googlepages.com/Brier-Peyrin-ACNS2009.pdf, Waktu Akses: 16 Mei 2010, 20.35.

[4] Introduction to CubeHash, http://cubehash.cr.yp.to/index.html, Waktu Akses: 28 April 2010, 21.58.

[5] Cryptographic Hash Algorithm Competition, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, Waktu Akses: 28 April 2010, 21.16.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 16 Mei 2010

ttd

Stephen Herlambang - 13507040