PENGGUNAAN ENKRIPSI DALAM PHP MENGGUNAKAN LIBRARY *MHASH*

Irvan Prama Defindal
NIM. 135 03 018

Program Studi Teknik Informatika STEI, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13018@students.if.itb.ac.id

Abstrak

PHP merupakan bahasa pemrograman yang paling sering digunakan dalam aplikasi berbasis web. Sebagai sebuah aplikasi *script* yang berjalan di sisi server (*server side script*), semua data akan dikirim melalui jaringan internet yang luas. Keamanan data memegang peranan penting dalam pengiriman data melalui jaringan. Selain adanya prosedur standar yang diterapkan oleh PHP untuk mengamankan data-data yang melewati jaringan , PHP juga menyediakan library khusus yang memungkinkan pengguna bahasa pemrograman tersebut mengenkripsi sendiri data. Dalam PHP dikenal 2 library enkripsi , yaitu *mcrypt* dan *mhash* yang bekerja dengan cara yang berbeda. Pada makalah ini , penulis hanya membahas fungsi-fungsi yang disediakan oleh *mhash*..

Seperti sebuah fungsi hash , library *mhash* akan memproduksi sederatan bit yang jumlahnya telah ditentukan sebelumnya yang dienkripsi menggunakan algoritma tertentu. Library ini banyak digunakan pada penyandian tandatangan digital *Mhash* ini mendukung berbagai macam algoritma , yaitu Adler32, CRC32 , CRC32B , Gost , Hava128, Haval160, Haval192, Haval256, Md4 , Md5, Ripemd160, Ripemd128, SHA1 , SHA256, Snefru , Tiger, Tiger128, dan Tiger 160. Sedangkan untuk pembangkitan kunci , library ini mendukung proses pembangkitan kunci dari Mcrypt, Asis , Hex , Pkdes , Simple , Salted, serta Isalted.

Kata Kunci :.

1. Pendahuluan

Berkembangnya dunia teknologi terutama dalam hal jaringan membuat teknologi informasi menjadi sebuah kebutuhan yang mendasar bagi semua orang. Semua perindustrian di dunia membutuhkan infrastruktur teknologi informasi yang memadai untuk mendukung semua aktifitas perusahaan.

Pengguna bahasa pemrograman kebanyakan kadang-kadang lupa akan pentingnya keamanan data yang dilewatkan melalui jaringan. Di samping itu, library enkripsi pada PHP bukan merupakan library dasar, sehingga perlu dilakukan beberapa langkah konfigurasi sebelum menggunakan library tersebut. Dengan informasi yang lebih mengenai fungsi enkripsi pada PHP , penggunaan fungsi-fungsi pada library *mhash* akan memudahkan programmer untuk mengamankan data yang bersifat penting dan rahasia.

Dengan adanya pengetahuan mengenai cara kerja dan seluk-beluk library ini, maka programmer bisa mengambil keputusan yang tepat dalam penggunaan fungsi-fungsi enkripsi data.

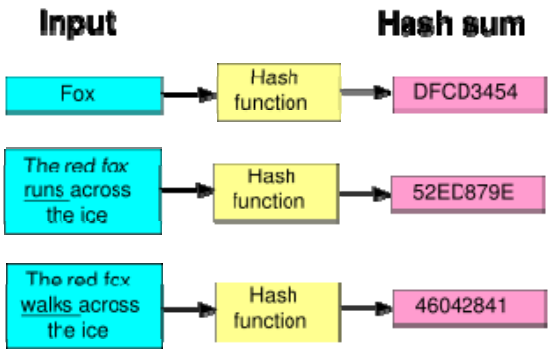
2. Fungsi Hash

2.1. Definisi dan Properti

Fungsi hash adalah metode yang digunakan untuk membangkitkan alamat di memory untuk kunci yang sudah dipesan . Fungsi yang terdapat pada *hash* memfasilitasi pengguna untuk menciptakan sebuah tantatangan digital dari sembarang data. Fungsi ini memotong dan mengkombinasikan data, baik melalui proses

substitusi maupun transposisi , untuk membuat tandatangan yang dinamakan dengan nilai *hash* . Nilai hash biasanya akan direpresentasikan dalam bilangan heksadesimal.

Sebuah fungsi hash yang baik akan menimalisir terjadinya *hash collisions* pada input dengan domain yang sama. Karena , *hash collision* memegang peranan penting dalam mempertahankan integritas informasi dan data menghadapi saluran dengan kebisingan tinggi dan media penyimpanan yang tidak selalu bias diandalkan. Pada table hash dan pemrosesan data, kolisi akan mengakibtakan perbedaan pada data , dan akan membuat pencarian *record* akan menghabiskan *cost* yang lebih



Gambar 1. Prosedur Standar Fungsi Hash

Sifat yang sangat fundamental bagi semua fungsi *hash* adalah apabila *2 hash* berbeda,yang diproses menggunakan fungsi hash tertentu, makan input dari *hash* merupakan hal yang berbeda. Oleh karena itu , maka fungsi *hash* merupakan fungsi deterministik. Begitu pun sebaliknya, sebuah fungsi *hash* tidak bersifat injektif, yaitu apabila 2 nilai *hash* bernilai sama, tidak akan menjamin sepenuhnya bahwa inputnya bernilai sama. Bila sebuah nilai *hash* dihitung dari sekumpulan data, maka perubahan pada 1 bit data akan menghasilkan nilai *hash* yang jauh berbeda karena fungsi *hash* semestinya mempunyai property yang bercampur..

Fungsi *hash* seharusnya mempunyai

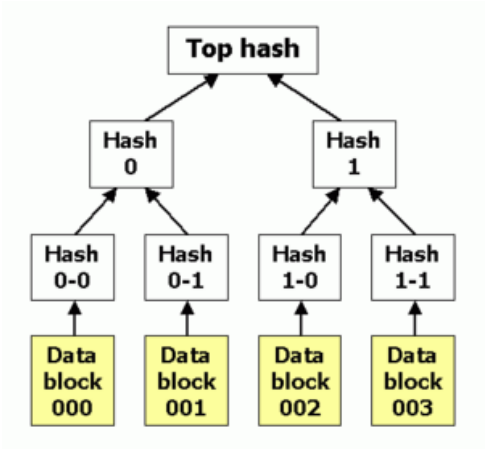
a. Domain yang tidak terbatas , seperti jumlah byte kumpulan karakter dengan panjang sembarang
b. Range yang terbatas, yaitu bit sekuensial dengan panjang yang sama

Dalam beberapa kasus , fungsi hash bisa diatur sedemikian rupa sehingga akan menciptakan fungsi satu-satu yang memetakan *domain* dan *range* dengan ukuran indentik. Fungsi *Hash* ini juga disebut dengan permutasi dimana *reversibility* diterapkan pada input dengan menerapkan serangkaian operasi campuran

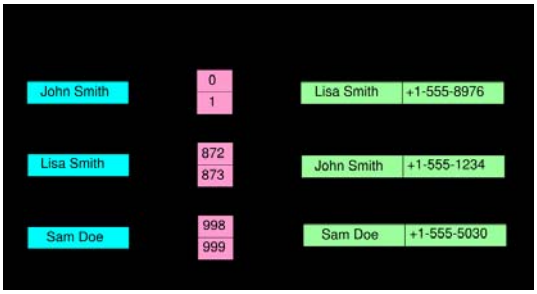
2.2. Representasi Hash

Fungsi hash dapat direpresentasikan dalam berbagai bentuk, seperti :

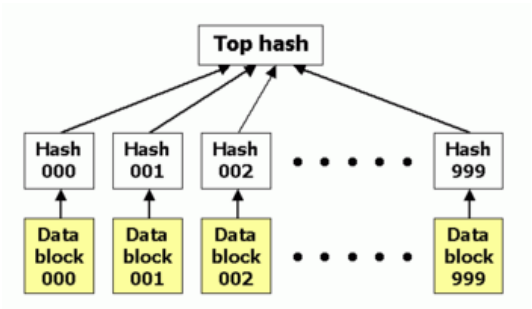
a. Pohon *Hash*



b. Table *Hash*



c. List *Hash*



3. Algoritma Mhash

Library Mhash pada PHP mendukung banyak metode algoritma sehingga, seharusnya layak dikaji. Yaitu :

3.1. Adler32

Merupakan algoritma *checksum* yang ditemukan oleh Mark Adler. Algoritma ini mampu menjalankan tes redundansi siklik sepanjang 32 bit untuk menjaga dari modifikasi mendadak terhadap data ,seperti distorsi yang terjadi pada saat transmisi data. Algoritma ini memegang peranan penting dalam perhitungan dalam perangkat lunak karena kecepatannya.

Nilai Algoritma Adler32 yang berjumlah 32bit didapatkan dengan menggabungkan 2 buah 16 bit checksum ; A dan B dengan aturan sebagai berikut :

A — Merupakan jumlah dari semua bytes dalam *string* . Diinisialisasi dengan 1 .

B — Merupakan jumlah individual dari A pada masing-masing langkah. Nilai B diinisialisasi dengan 0

Algoritma ini bisa disederhanakan dalam notasi :

$$A = 1 + D_1 + D_2 + \ldots + D_n \pmod{65521}$$

$$B = (1 + D_1) + (1 + D_1 + D_2) + \ldots + (1 + D_1 + D_2 + \ldots + D_n) \pmod{65521}$$

$$= n \times D_1 + (n-1) \times D_2 + (n-2) \times D_3 + \ldots + D_n + n \pmod{65521}$$

$$\textbf{Adler-32}(D) = B \times 65536 + A$$

Gambar 2. Notasi Algoritma Adler32

Dimana :
D : byte string yang akan dikalkulasi *checksum*
N : panjang dari D

Dapat dilihat bahwa tiap jumlah pada Adler32 selalu dikenai operasi modulus terhadap angka 65521, karena angka ini merupakan bilangan prima terbesar yang lebih kecil dari 2^{16}

Adler32 mempunyai kelebihan dibandingkan CRC karena lebih cepat bekerja pada perangkat lunak. Walaupun, untuk pesan berukuran kecil algoritma ini tidak bisa menangani 32 bit kosong yang tersisa.

Pada library *mhash* , algoritma ini diimplementasikan pada file *adler32.c.* Berikut potongan kode program *mhash* yang menggunakan adler32 :

```
void mhash_adler32(mutils_word32 *
adler, __const void *given_buf,
mutils_word32 len)
{
mutils_word32 s1 = (*adler) &
0x0000FFFF;
mutils_word32 s2 = ((*adler) >>
16) & 0x0000FFFF;
mutils_word32 n;
mutils_word8 *p = (mutils_word8 *)
given_buf;

for (n = 0; n < len; n++, p++)
{
      s1 += *p;
      if (s1 >= 65521)
/* using modulo took about 7 times
longer on my CPU! */
      {
      s1 -= 65521;
/* WARNING: it's meant to be >=
65521, not just > ! */
            }
      s2 += s1;
      if (s2 >= 65521)
/* same warning applies here, too
*/
      {
      s2 -= 65521;
            }
      }
      *adler = (s2 << 16) + s1;
}
```

3.2. CRC32 dan CRC32B

CRC merupakan singkatan dari Cyclic Redundancy Check , merupakan fungshi *hash* yang digunakan untuk menghasilkan *checksum* dari 1 blok data, seperti paket pada lalu lintas jaringan atau blok pada file komputer, *Checksum* digunakan untuk mengidentifikasi kerusakan pada saat penyimpanan dan setelah transmisi data dilakukan. CRC dikalkulasi dan digabungkan sebelum transmisi atau penyimpanan dan diverifikasi oleh penerima

untuk menyatakan bahwa tidak ada perubahan yang terjadi pada proses pengiriman. CRC lebih populer karena kesederhanaan untuk mengimplementasikan dalam perangkat keras binaru dan mudah dianalisis secara matematik. Selain itu, CRC juga bekerja baik dalam mendekteksi kerusakan umum yang disebabkan oleh kebisingan pada saluran pengiriman.

Checksum dari CRC adalah sisa pembagian biner tanpa bit *carry* dari sebuah *stream* bit pesan dengan *strean* bit yang sudah ditentukan dengan panjang n bit, dimana n merepresentasikan koefisien dari sebuah polinom. Secara umum , fungsi CRC bisa disederhanakan dalam notasi :

$$M(x) \cdot x^n = Q(x) \cdot G(x) + R(x)$$

Dimana :
M(x) adalah polinom pesan asli
G(x) merupakan polinom pembangkit *n* derajat

Sedangkan untuk turunan CRC , dapat dilihat dari tabel di bawah :

Nama	Polinom	Representasi
CRC-1	$x + 1$	0x1 or 0x1 (0x1)
CRC-5-CCITT	$x^5 + x^3 + x + 1$	0x0B or 0x15 (0x??)
CRC-5-USB	$x^5 + x^2 + 1$	0x05 or 0x14 (0x9)
CRC-7	$x^7 + x^3 +$	0x09 or 0x48 (0x11)
CRC-8-ATM	$x^8 + x^2 + x + 1$	0x07 or 0xE0 (0xC1)
CRC-8-CCITT	$x^8 + x^7 + x^3 + x^2 + 1$	0x87 or 0xE1
CRC-8-Dallas/Maxim	$x^8 + x^5 + x^4 + 1$	0x31 or 0x8C
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$	0xD3 or 0xCB (0x??)
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$	0x233 or 0x331 (0x????)
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	0x80F or 0xF01 (0xE03)
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	0x4599 or 0x4CD1
CRC-16-Fletcher	bukan CRC	Adler-32
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$	0x1021 or 0x8408 (0x0811)
CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$	0x8005 or 0xA001 (0x4003)
CRC-32-Adler	bukan CRC	See Adler-32
CRC-32-MPEG2	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	0x04C11DB7 or 0xEDB88320 Also used in IEEE 802.3
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (V.42)	0x04C11DB7 or 0xEDB88320 (0xDB710641)
CRC-32C (Castagnoli)[1]	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$	0x1EDC6F41 or 0x82F63B78 (0x05EC76F1)
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$	0x0000000000000 01B or 0xD800000000000 000 (0xB00000000000 0001)
CRC-64-ECMA-182	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^4 + x + 1$	0x42F0E1EBA9E A3693 or 0xC96C5795D787 0F42 (0x92D8AF2BAF 0E1E85)
CRC-128	Use superseded by hashes MD5 & SHA-1	
CRC-160	Use superseded by hashes MD5 & SHA-1	

Tabel 1. Polinom CRC yang sering digunakan

Pada library *mhash* , algoritma ini diimplementasikan pada file *crc32.c*. Berikut

potongan kode program *mhash* yang menggunakan crc32 :

```
void mhash_get_crc32(__const
mutils_word32 *crc, void *ret)
{
      mutils_word32 tmp;
      tmp = ~(*crc);
      /* transmit complement,
per CRC-32 spec  */
#if defined(WORDS_BIGENDIAN)
      tmp =
mutils_word32swap(tmp);
#endif
      if (ret != NULL)
      {
      mutils_memcpy(ret, &tmp,
sizeof(mutils_word32));
      }
}
```

Sedangkan untuk crc32b, terdapat pada module yang sama, namun memanggil fungsi yang berbeda :

```
Void mhash_crc32b(mutils_word32
*crc, __const void *given_buf,
mutils_word32 len)
{
      __const mutils_word8 *p;

      if ((crc == NULL) ||
(given_buf == NULL) || (len ==
0))
      {
            return;
      }

      for (p = given_buf; len >
0; ++p, --len)
      {
            (*crc) = (((*crc)
>> 8) & 0x00FFFFFF) ^
crc32_table_b[(*crc ^ *p) &
0xff];
      }
}
```

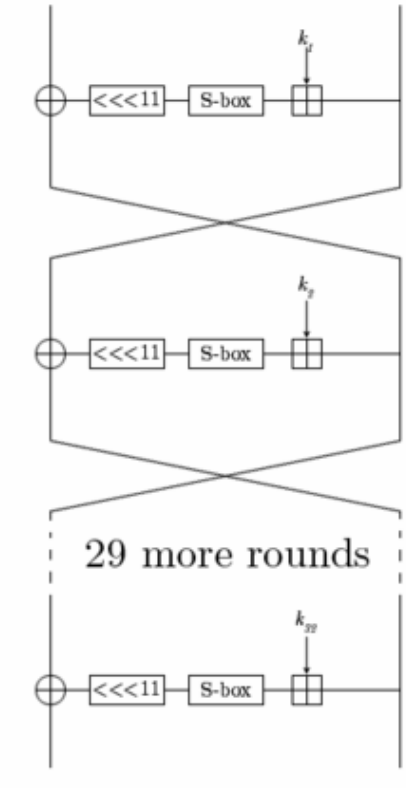
Kedua fungsi di atas mempunyai polinom yang berbeda , dimana CRC32b menggunakan konstanta *crc32_table_b[256]* dengan polinom 0xEDB88320L) Sedangkan CRC menggunakan konstanta crc32_table[256] dengan polinom 0x04c11db7) dan digunakan pada AUTODIN II, Ethernet, & FDDI

3.3. GOST

GOST atau lebih detail merupakan GOST 28147-89 merupakan standar blok kunci simetris yang dikembangkan pada tahun 1970. Standar ini pada jamannya merupakan *Top Secret* yang kemudian mengalami penurunan status menjadi *Secret* pada tahun 1990. Setelah pecahnya Uni Soviet , GOST menjadi publik dan diketahui rahasianya.

GOST mempunyai blok berukuran 64 bit dengan panjang kunci 512 bit. Secara konsep, strukturnya menggunakan Jaringan Feistel dengan jumlah putaran sebanyak 32 kali dimana 8 putaran terakhir menggunakan kata kunci dengan arah berlawanan dengan 24 putaran pertama.

Diagram GOST bisa dilihat dari bawah :



Pada library *mhash* , algoritma ini diimplementasikan pada file *gosthash.c.* Sedikit

berbeda dengan algoritma sebelumnya , GOST menggunakan 4 tahap utama dalam penyandian :

a. Reset

Merupakan tahap inisialisasi dimana semua atribut dari tipe GostHash diberi nilai awal yang bersesuaian. Hal tersebut dapat dilihat dari potongan kode di bawah :

```
void gosthash_reset(GostHashCtx
* ctx)
{
      mutils_bzero(ctx->sum,
32);
      mutils_bzero(ctx->hash,
32);
      mutils_bzero(ctx->len,
32);
      mutils_bzero(ctx-
>partial, 32);
      ctx->partial_bytes = 0;
}
```

b. Utama

Pada tahap ini dipanggil fungsi *gosthash_byte* yang melakukan proses enkripsi dalam 32 putaran. Fungsi ini juga memanggil *gosthash_compress* yang melakukan kompresi terhadap bit yang tidak terpakai.

```
static void
gosthash_bytes(GostHashCtx *
ctx, __const mutils_word8 *
buf, mutils_word32 bits)
{
      mutils_word32 i;
      mutils_word32 j;
      mutils_word32 a, c, m[8];

/* convert bytes to a long
words and compute the sum */

j = 0;
c = 0;
for (i = 0; i < 8; i++) {
a = ((mutils_word32) buf[j]) |
(((mutils_word32) buf[j + 1])
<< 8) | (((mutils_word32) buf[j
+ 2]) << 16) |
(((mutils_word32) buf[j + 3])
<< 24);
j += 4;
m[i] = a;
c = a + c + ctx->sum[i];
if ((a==0xFFFFFFFFUL) && (ctx-
>sum[i]==0xFFFFFFFFUL)) {
      ctx->sum[i] = c;
      c = 1;
      } else {
      ctx->sum[i] = c;
      c = c < a ? 1 : 0;
      }
}
/*  compress  */

gosthash_compress(ctx->hash,m);

/* a 64-bit counter should be
sufficient  */

ctx->len[0] += bits;
if (ctx->len[0] < bits)
      ctx->len[1]++;
}
```

c. Update

Pada tahap ini, terjadi pencampuran bit data ke dalam buffer.

```
void osthash_update(GostHashCtx *
ctx, __const mutils_word8 * buf,
mutils_word32 len)
{
      mutils_word32 i, j;

      i = ctx->partial_bytes;
      j = 0;
while (i < 32 && j < len)
ctx->partial[i++] = buf[j++];

if (i < 32) {
      ctx->partial_bytes = i;
      return;
}
gosthash_bytes(ctx, ctx->partial,
256);

while ((j + 32) < len) {
gosthash_bytes(ctx,&buf[j],256);
      j += 32;
}
      i = 0;
while (j < len)
ctx->partial[i++] = buf[j++];
ctx->partial_bytes = i;
}
```

d. Final

Merupakan langkah untuk menghitung dan menyimpan pesan ringka berukuran 32 bit.

```
void gosthash_final(GostHashCtx
* ctx, mutils_word8 * digest)
{
      mutils_word32 i;
      mutils_word32 j;
      mutils_word32 a;

/* adjust and mix in the last
chunk */

if (ctx->partial_bytes > 0) {
mutils_bzero(&ctx->partial[ctx-
>partial_bytes], 32 - ctx-
>partial_bytes);
gosthash_bytes(ctx, ctx-
>partial, ctx->partial_bytes <<
3);
}
/* mix in the length and the
sum */

gosthash_compress(ctx->hash,
ctx->len);
gosthash_compress(ctx->hash,
ctx->sum);

/*convert the output to bytes*/
j = 0;
if (digest != NULL)
for (i = 0; i < 8; i++) {
a = ctx->hash[i];
digest[j] =(mutils_word8) a;
digest[j + 1] = (mutils_word8)
(a >> 8);
digest[j + 2] = (mutils_word8)
(a >> 16);
digest[j + 3] = (mutils_word8)
(a >> 24);
            j += 4;
      }
}
```

GOST mempunyai 4 S Box dengan masing-masing nama gost_sbox_1 , gost_sbox_2 , gost_sbox_3 dan gost_sbox_4.

3.4. HAVAL128,HAVAL160,HAVAL 192, dan HAVAL256

Haval merupakan jenis algoritma yang jarang digunakan namun didukung oleh *mhash.* Sedikit keunikan pada HAVAL adalah nilai *hash* yang dimiliki mempunyai panjang yang berbeda, yaitu antara 128,160,192, atau 256 bit. Algoritma ini memungkinkan pengguna menentukan jumlah putaran untuk membangkitkan nilai *hash* . Haval ditemukan oleh Yuliang Zheng dkk pada tahun 1992.

Nilai hash dari algoritma Haval biasanya direpresentasikan dalam 32, 40, 48, 56, atau 64 digit bilangan heksadesimal. Haval mendukung 15 tingkat keamanan yang berbeda. Selama ini belum ada serangan yang berhasil terhadap Haval yang telah diketahui. Berdasarkan percobaan kriptanalis secara brute force , pada putaran ketiga didapat data : 113.64 Mega bits per detik pada UltraSparc dengan komplier CC dan 92.6 Mega bits per detik pada Pentium Pro 180 yang dikompilasi dengan GCC.

Aturan penamaan versi Haval :

HAVAL – X – YYY

X = jumlah kalang
Y = jumlah bit

Pada library *mhash* , algoritma ini diimplementasikan pada file *haval.c.* Hampir sama dengan GOST , Haval melalui 3 tahap dalam penyandian :

a. Inisialisasi

Digunakan untuk memvalidasi masukan pada saat pemanggilan dan memberi nilai awal senarai pesan ringkas. Proses inisialisasi dapat dilihat dari potongan kode: :

```
mutils_error havalInit
(havalContext
*hcp,mutils_word32 passes,
mutils_word32 hashLength)
{
if (hcp == NULL)  {
return(-
MUTILS_INVALID_INPUT_BUFFER);
/* bad context */
}
/* check number of passes: */
if (passes != 3 && passes != 4
&& passes != 5)
{
return(-
MUTILS_INVALID_PASSES); /*
invalid number of passes */
}
```

b. Update

```
/* check hash length: */
if (hashLength != 128 &&
hashLength != 160 &&
hashLength != 192 &&
hashLength != 224 &&
hashLength != 256)
{
return(-MUTILS_INVALID_SIZE);
/* invalid hash length */
}
/* properly initialize HAVAL
context: */
mutils_bzero(hcp, sizeof
(havalContext));
hcp->passes = (mutils_word16)
passes;
hcp->hashLength =
(utils_word16) hashLength;
hcp->digest[0] = 0x243F6A88UL;
hcp->digest[1] = 0x85A308D3UL;
hcp->digest[2] = 0x13198A2EUL;
hcp->digest[3] = 0x03707344UL;
hcp->digest[4] = 0xA4093822UL;
hcp->digest[5] = 0x299F31D0UL;
hcp->digest[6] = 0x082EFA98UL;
hcp->digest[7] = 0xEC4E6C89UL;
return(MUTILS_OK); /* OK */
}
```

Pada tahap ini dipanggil fungsi *haval_update* yang melakukan proses-proses :

i. Memperbarui hitungan bit yaitu dengan menambahkan tanda kurung untuk C++
ii. Melengkapi blok data dalam konteks
iii. Memproses konteks data blok untuk 3 kondisi yang berbeda

c. Final

Pada fungsi ini , dilakukan prosedur untuk :

i. Mengasersi assert (hcp->occupied < 128); invariant
ii. Menggabungkan *toggkle* ke blok data konteks
iii. Mengisi pesan kosong dengan null sehingga panjang bit 944
iv. Melakukan *append* terhadap data yang berada di ujung dan memproses blok pesan terakhir yang dipadding
v. Memecah 256bit pesan ringkas dalam ukuran tertentu

3.5. MD4

Merupakan algoritma yang dirancang oleh Prof Ronald Rivest dari MIT pada tahun 1990. Algoritma ini merupakan fungsi hash yang menekankan pada pengecekan integritas pesan. Pesan ringkas sepanjang 128 bit.

MD4 bekerja dalam 5 tahap :

a. Menambahkan bit di akhir sehingga panjangnya kongruen dengan 448 dan modulus dengan 512.
b. Menambahkan panjang sebagai keterangan dalam pesan
c. Inisialisasi MD Buffer dengan nilai-nilai :

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

d. Memproses pesan dalam blok berukuran 16 word

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XY v XZ v YZ
H(X,Y,Z) = X xor Y xor Z
```

e. Output : dimulai dengan byte terkecil dari A dan diakhiri dengan byte terbesar dari D

Pada library *mhash* , algoritma ini diimplementasikan pada file *md4.c.* Ada 3 prosedur utama dalam MD4:

a. MD4Init

Merupakan fungsi untuk melakukan inisialisasi terhadap MD4. Mengatur hitungan bit menjadi 0 dan alokasi buffer.

```
void MD4Init(struct MD4Context
*ctx)
{
      ctx->buf[0] = x67452301;
      ctx->buf[1] = xefcdab89;
      ctx->buf[2] = x98badcfe;
      ctx->buf[3] = x10325476;

      ctx->bits[0] = 0;
      ctx->bits[1] = 0;
}
```

b. MD4Update

Merupakan prosedur untuk memperbarui konteks yang mencerminkan penggabungan dari buffer lain yang penuh.

```
void MD4Update(struct
MD4Context *ctx, mutils_word8
__const *buf, mutils_word32
len)
{
register mutils_word32 t;

t = ctx->bits[0];
if ((ctx->bits[0] = t +
((mutils_word32) len << 3)) <t)
ctx->bits[1]++;
/* Carry from low to high */
ctx->bits[1] += len >> 29;

t = (t >> 3) & 0x3f;

if (t) {
mutils_word8 *p = (mutils_word8
*) ctx->in + t;

t = 64 - t;
if (len < t) {
mutils_memcpy(p, buf, len);
      return;
}
mutils_memcpy(p, buf, t);

mutils_word32nswap((mutils_word
32 *) ctx->in, 16,MUTILS_TRUE);
MD4Transform(ctx-
>buf,(mutils_word32 *) ctx-
>in);
      buf += t;
      len -= t;

while (len >= 64) {
mutils_memcpy(ctx->in,buf, 64);
mutils_word32nswap((mutils_word
32 *) ctx->in, 16,
MUTILS_TRUE);
MD4Transform(ctx-
>buf,(mutils_word32 *) ctx-
>in);
      buf += 64;
      len -= 64;
}

mutils_memcpy(ctx->in,
buf,len);
}
```

c. MD4Final

Melakukan proses *padding* terhadap pembatas 64 byte dengan *bit pattern*

d. MD4Transform

Merupakan inti dari algoritma MD4 dengan menggunakan konstanta F,G,H, FF, GG, dan HH dengan ketentuan sebagai berikut :

```
#define F(x, y, z) (((x) & (y)) |
((~x) & (z)))
#define G(x, y, z) (((x) & (y)) |
((x) & (z)) | ((y) & (z)))
#define H(x, y, z) ((x) ^ (y) ^
(z))

#define FF(a, b, c, d, x, s) { \
(a) += F ((b), (c), (d)) + (x); \
(a) = rotl32 ((a), (s)); \
}
#define GG(a, b, c, d, x, s) { \
(a) += G ((b), (c), (d)) + (x) +
(mutils_word32)0x5a827999; \
(a) = rotl32 ((a), (s)); \
}
#define HH(a, b, c, d, x, s) { \
(a) += H ((b), (c), (d)) + (x) +
(mutils_word32)0x6ed9eba1; \
(a) = rotl32 ((a), (s)); \
     }
```

3.6. MD5

Merupakan penyempurnaan dari MD4 yang mempunyai banyak kelemahan. Prinsip dasar MD5 menggunakan fungsi

MD5compress(*I*,*X*)= MD5compress(*J*,*X*)

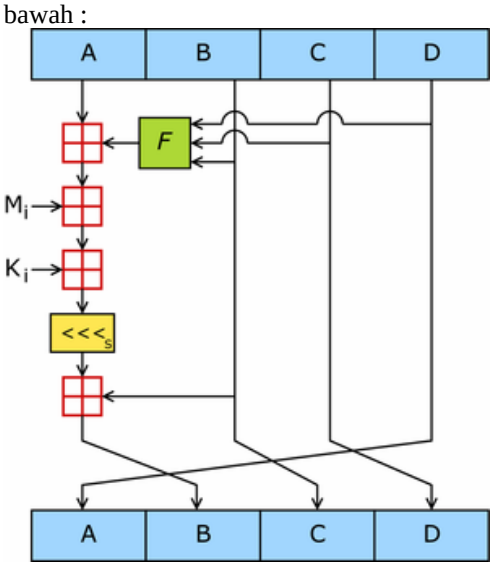
Sedangkan nilai F,G,H, dan I merupakan hasil komputasi dari :

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

Dimana :

$\oplus, \wedge, \vee, \neg$ berarti (secara berurutann) XOR , OR , AND, NOT

Untuk cara kerja, bisa dilihat dari bagan di bawah :



Sedangkan untuk fungsi di *mhash*, secara umum sama dengan MD4.

3.7. RIPEMD160 dan RIPEMD128

RIPEMD singkatan dari RACE Integrity Primitives Evaluation Message Digest merupakan algoritma untuk membentuk pesan ringkas sepanjang 160bit . Dibangun oleh Hans Dobbertin dan dirilis pada tahun 1996 yang merupakan pengembangan dari RIPEMD berdasarkan prinsip desain MD4 dan serupa dengan performa SHA-1.

Fungsi ini pada library *mhash* terdapat pada file *ripemd.c* . Tipe ripemd_ctx merupakan prototype dari semua ripemd dan mempunyai struktur sebagai berikut :

```
typedef struct ripemd_ctx {
  mutils_word32
digest[RIPEMD_STATESIZE];
mutils_word64 bitcount;
  mutils_word8
block[RIPEMD_DATASIZE];
  mutils_word32 index;
  mutils_word32 digest_len;
   } RIPEMD_CTX;
```

3.8. SHA1 dan SHA256

Algoritma ini merupakan algoritma yang sangat terkenal dan lazim digunakan. Pada libray *mhash*, sha mempunyai banyak sekali fungsi yang bisa dimanfaatkan. Yaitu (berdasarkan file) :

Mhash_sha1

*void sha_init(struct sha_ctx *ctx);*
*void sha_update(struct sha_ctx *ctx, mutils_word8 *buffer, mutils_word32 len);*
*void sha_final(struct sha_ctx *ctx);*
*void sha_digest(struct sha_ctx *ctx, mutils_word8 *s);*
*void sha_copy(struct sha_ctx *dest, struct sha_ctx *src);*

Mhash_sha256

*void sha256_init(struct sha256_ctx *ctx);*
*void sha256_update(struct sha256_ctx *ctx, __const mutils_word8 *data, mutils_word32 length);*
*void sha256_final(struct sha256_ctx *ctx);*
*void sha256_digest(__const struct sha256_ctx *ctx, mutils_word8 *digest);*

Mhash_sha256_sha224

*void sha256_init(struct sha256_sha224_ctx *ctx);*
*void sha224_init(struct sha256_sha224_ctx *ctx);*
*void sha256_sha224_update(struct sha256_sha224_ctx *ctx, __const mutils_word8 *data, mutils_word32 length);*
*void sha256_sha224_final(struct ha256_sha224_ctx *ctx);*
*void sha256_digest(__const struct sha256_sha224_ctx *ctx, mutils_word8 *digest);*
*void sha224_digest(__const struct sha256_sha224_ctx *ctx, mutils_word8 *digest);*

Mhash_sha512_sha384

*void sha512_init(struct sha512_sha384_ctx *ctx);*
*void sha384_init(struct sha512_sha384_ctx *ctx);*
*void sha512_sha384_update(struct sha512_sha384_ctx *ctx, __const mutils_word8 *data, mutils_word32 length);*
*void sha512_sha384_final(struct ha512_sha384_ctx *ctx);*
*void sha512_digest(__const struct sha512_sha384_ctx *ctx, mutils_word8 *digest);*
*void sha384_digest(__const struct sha512_sha384_ctx *ctx, mutils_word8 *digest);*

3.9. Snefru

Berasal dari nama salah seorang Farauk Mesir , ditemukan oleh Raplh Merkle. Fungsi hash ini mendukung output sebanyak 128bit dan 256bit. Setelah ditemukan adanya kelemahan, desain algoritma kemudian dimodifikasi dengan menambah iterasi pada kalang utama dari 2 menjadi 8. Secara brute force, serangan terhadap algoritma ini membutuhkan $2^{88.5}$

Adapun struktur tipenya adalah :

```
typedef struct snefru_ctx
{
/* buffer of data to hash */
mutils_word8
buffer[SNEFRU128_DATA_SIZE];
/* number of hashed bits */
mutils_word64 hashlen;
/* index to buffer */
mutils_word32 index;
/* the hashing state */
mutils_word32
hash[SNEFRU_BLOCK_LEN];
} SNEFRU_CTX;
```

Pada library *mhash*,algoritma ini mempunyai beberapa fungsi, diantaranya :

```
Void snefru_init(struct
snefru_ctx *ctx);
Void snefru128_update(struct
snefru_ctx *ctx, __const
mutils_word8 *data, mutils_word32
length);
Void snefru256_update(struct
snefru_ctx *ctx, __const
mutils_word8 *data, mutils_word32
length);
Void snefru128_final(struct
snefru_ctx *ctx);
Void snefru256_final(struct
snefru_ctx *ctx);
Void snefru128_digest(__const
struct snefru_ctx *ctx,
mutils_word8 *digest);
Void snefru256_digest(__const
struct snefru_ctx *ctx,
mutils_word8 *digest);
```

4. Pembangkit Kunci Mhash

Library *mhash*, seperti halnya algoritma enkripsi lain menggunakan kunci . Adapun struktur data dari kunci yang dibangkitkan dalah sebagai berikut :

```
typedef struct
__mhash_keygen_entry {
      mutils_word8 *name;
      keygenid id;
      mutils_boolean
      uses_hash_algorithm;
      mutils_boolean uses_count;
      mutils_boolean uses_salt;
      mutils_word32 salt_size;
      mutils_word32 max_key_size;
} mhash_keygen_entry;
```

Kunci ini bisa dibangkitkan secara otomatis menggunakan :

4.1. MCRYPT

Merupakan pembangkit kunci yang digunakan pada libray *mcrypt* .Fungsi yang dipanggil adalah *_mhash_gen_key_mcrypt* dengan masukan :

a. input *algorithm* bertipe hashid
b. input *keyword tidak bertipe
c. input *key_size* bertipe mutils_word32
d. input **salt* tidak bertipe
e. input *salt_size* bertipe mutils_word32
f. input *password bertipe mutils_word8
g. input plen bertipe mutils_word32

4.2. ASIS

Pembangkit kunci ini mengembalikan kata kunci dalam bentuk biner. Adapun fungsi yang dipanggil adalah :

```
mutils_error
_mhash_gen_key_asis(void
*keyword, mutils_word32 key_size,
mutils_word8 *password,
mutils_word32 plen)
{
#if defined(MHASH_ROBUST)
if ((keyword == NULL) ||
(key_size == 0) || (password ==
NULL) || (plen == 0))

return(-MUTILS_INVALID_SIZE);
#endif
if (plen > key_size)
plen=key_size;

mutils_bzero(keyword, key_size);
mutils_memcpy(keyword, password,
plen);
return(MUTILS_OK);
}
```

4.3. HEX

Fungsi yang tersedia adalah *_mhash_gen_key_hex* dengan masukan :

a. input **keyword* tidak bertipe
b. input *key_size* bertipe mutils_word32
c. input **password* bertipe mutils_word8
d. input *plen* bertipe mutils_word32

KEYGEN_HEX melakukan konversi terhadap kunci heksadesimal menjadi bilangan biner

4.4. S2K_SIMPLE

Fungsi yang disediakan adalah :

```
mutils_error
_mhash_gen_key_s2k_simple(hash
id algorithm,void *keyword,
mutils_word32 key_size,
mutils_word8 *password,
mutils_word32 plen)
```

Pembangkit kunci ini menggunakan algoritma OpenPGP(Pretty Good Privacy) RFC2440 Sederhana S2K.

4.5. S2K_SALTED

```
mutils_error
_mhash_gen_key_s2k_salted(hash
id algorithm,void *keyword,
mutils_word32 key_size,
mutils_word8 *salt,
mutils_word32 salt_size,
mutils_word8 *password,
mutils_word32 plen)
```

Pembangkit kunci ini menggunakan algoritma OpenPGP(Pretty Good Privacy) Salted S2K

4.6. S2K_ISALTED

Pembangkit kunci ini menggunakan algoritma OpenPGP(Pretty Good Privacy) Isalted S2K

```
mutils_error
_mhash_gen_key_s2k_isalted(
      hashid algorithm,
      mutils_word64 _count,
      void *keyword,
      mutils_word32 key_size,
      mutils_word8 *salt,
      mutils_word32 salt_size,
      mutils_word8 *password,
      mutils_word32 plen)
```

4.7. PKDES

Transformasi digunakan pada program enkripsi DES buatan Phil Karn. Berikut fungsi lengkap yang akan dijalankan :

```
mutils_error
_mhash_gen_key_pkdes(
void *keyword, mutils_word32
key_size, mutils_word8 *password,
mutils_word32 plen)
{
mutils_word8 *ptr = keyword;
mutils_word32 cnt;
mutils_word32 i;
mutils_word32 c;

#if defined(MHASH_ROBUST)
if (keyword == NULL)
return(-
MUTILS_SYSTEM_RESOURCE_ERROR);
#endif

if (plen > key_size)
return(-MUTILS_INVALID_SIZE);
mutils_bzero(keyword, key_size);
mutils_memcpy(keyword, password,
plen);

for (cnt = 0; cnt < key_size;
cnt++, ptr++) {
      c = 0;
      for (i = 0; i < 7; i++)
      if (*ptr & (1 << i))
            c++;
      if ((c & 1) == 0)
            *ptr |= 0x80;
      else
            *ptr &= ~0x80;
      }

      return(MUTILS_OK);
}
```

5. Library Mhash

Library Mhash mempunyai tipe utama yaitu mhash_hash_entry yang mempunyai properti :

a. Name
Tipe : mutils_word *
Menentukan nama dari fungis hash
b. Id
Tipe : hashid
Merupakan id yang diberikan kepada sebuah fungsi hash
c. block_size
Tipe : mutils_word32
Menyimpan informasi ukuran blok yang akan diproses oleh fungsi hash
d. hash_block
Tipe : mutils_word32
Merupakan blok yang dimiliki fungsi hash
e. state_size
tipe : mutlis_word32
f. init_func
tipe : INIT_FUNC
Menyatakan fungsi yang dipanggil pada saat inisialisasi
g. hash_func
tipe : HASH_FUNC
Adalah prosedur yang dijalankan untuk melakukan proses hashing
h. final_func
Tipe : FINAL FUNC
Menyimpan informasi nama prosedur yang dipanggil ketika Hash telah selesai dilaksanakan
i. deinit_func
Tipe : DEINIT_FUNC

Library Mhash yang bisa digunakan oleh pengguna adalah :

a. Libray untuk menentukan algoritma
i. **MHASH_ADLER32**
ii. **MHASH_CRC32**
iii. **MHASH_CRC32B**
iv. **MHASH_GOST**
v. **MHASH_HAVAL128**
vi. **MHASH_HAVAL160**
vii. **MHASH_HAVAL192**
viii. **MHASH_HAVAL256**
ix. **MHASH_MD4**
x. **MHASH_MD5**
xi. **MHASH_RIPEMD160**
xii. **MHASH_SHA1**
xiii. **MHASH_SHA256**
xiv. **MHASH_TIGER**
xv. **MHASH_TIGER128**
xvi. **MHASH_TIGER160**

b. Fungsi yang digunakan untuk hash

i. MHASH_COUNT

Untuk mendapatkan id hash tersedia yang tertinggi

```
WIN32DLL_DEFINE mutils_word32
mhash_count(void)
{
      hashid count = 0;

      MHASH_LOOP( count = MMAX(
p->id, count) );

      return count;
}
```

ii. MHASH_GET_BLOCK_SIZE

```
WIN32DLL_DEFINE mutils_word32
mhash_get_block_size(hashid type)
{
      mutils_word32 ret = 0;

      MHASH_ALG_LOOP(ret = p-
>blocksize);
      return ret;
}
```

Untuk mendapatkan ukuran blok dari hash tertentu

iii. MHASH_GET_HASH_NAME

Untuk mendapatkan nama dari sebuah fungsi hash

```
WIN32DLL_DEFINE mutils_word8
*mhash_get_hash_name(hashid type)
{
      mutils_word8 *ret = NULL;
      /* avoid prefix */
MHASH_ALG_LOOP(ret = p->name);

if (ret != NULL)  {
ret += sizeof("MHASH_") - 1;
}
      return mutils_strdup(ret);
}
```

iv. MHASH_KEYGEN_S2K
Untuk membangkitkan sebuah kunci

v. MHASH
Untuk menentukan nilai hash

```
mutils_boolean mhash(MHASH td,
__const     void     *plaintext,
mutils_word32 size)
{
if (td->hash_func!=NULL)
     {
     td->hash_func(td->state,
     plaintext, size);
       }
     return(MUTILS_OK);
}
```

5 Fungsi utama di atas memanfaatkan 2 konstanta berupa fungsi lain , yaitu:

1. MHASH_ALG_LOOP

```
#define MHASH_LOOP(b) \
__const mhash_hash_entry *p; \
for (p = algorithms; p->name !=
NULL; p++) { b ; }
```

2. **MHASH_ALG LOOP**

```
#define MHASH_ALG_LOOP(a) \
      MHASH_LOOP(  if  (p->id  ==
type) { a; break; } )
```

6. Penerapan Mhash

Penulis makalah mencobakan fungsi yang didukung oleh library *mhash* dan mendapatkan hasil sebagai berikut :

MHASH_ADLER32

```
<?php
$input = "Wikipedia";
$hash = mhash(MHASH_ADLER32,
$input);
echo "MHASH_ADLER32 :" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_ADLER32 : 11E60398
```

MHASH_CRC32

```
<?php
$input = "defindal";
$hash = mhash(MHASH_CRC32,
$input);
echo "MHASH_CRC32 :" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_CRC32 : E477F48
```

MHASH_HAVAL256

```
<?php
$input = "";
$hash = mhash(MHASH_HAVAL256,
$input);
echo "MHASH_HAVAL256:" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_HAVAL256:
be417bb4dd5cfb76c7126f4f8eeb1553a
449039307b1a3cd451dbfdc0fbbe330
```

MHASH_MD4

```
<?php
$input = "abc";
$hash = mhash(MHASH_MD4, $input);
echo "MHASH_MD4:" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_MD4 :
a448017aaf21d8525fc10ae87aa6729d
```

MHASH_MD5

```
<?php
$input = "The quick brown fox
jumps over the lazy cog";
$hash = mhash(MHASH_MD5, $input);
echo "MHASH_MD5:" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_MD5 :
1055d3e698d289f2af8663725127bd4b
```

MHASH_RIPEMD256

```
<?php
$input = "abc";
$hash = mhash(MHASH_RIPEMD256,
$input);
echo "MHASH_RIPEMD256:" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_RIPEMD256:
afbd6e228b9d8cbbcef5ca2d03e6dba10
ac0bc7dcbe4680e1e42d2e975459b65
```

MHASH_RIPEMD320

```
<?php
$input = "abc";
$hash = mhash(MHASH_RIPEMD320,
$input);
echo "MHASH_RIPEMD3206:" .
bin2hex($hash) . "<br />";
?>
```

Output :

```
MHASH_RIPEMD320:
de4c01b3054f8930a79d09ae738e92301
e5a17085beffdc1b8d116713e74f82fa9
42d64cdbc4682d
```

MHASH_SHA1

```
<?php
$input = "Let us meet at 9 o'
clock at the secret place.";
$hash = mhash (MHASH_SHA1,
$input);

print "The hash is ".bin2hex
($hash)."\n";
?>
```

Output :

```
The hash is
d3b85d710d8f6e4e5efd4d5e67d041f9c
ecedafe
```

Semua output merupakan bilangan heksa. Untuk mengkonversi menjadi heksa, digunakan fungsi php *bin2hex.*

```
<?php
$hash = MHASH_MD5;

print mhash_get_hash_name
($hash);
?>
```

Output :

```
MD5
```

7. Kesimpulan

Dari makalah ini dapat diambil kesimpulan :

a. Mhash merupakan library yang sangat kaya dengan fungsi yang mendukung berbagai macam algoritma
b. Library *mhash* mendukung proses otentifikasi pesan yang tidak dapat dilakukan oleh library *mcrypt*
c. Tidak ada sistem yang sepenuhnya aman di dunia ini karena berkembangnya ilmu pengetahuan berimbang antara yang baik dan buruk
d. Keamanan berbanding terbalik dengan biaya yang diperlukan dan waktu yang dihabiskan

8. Pustaka

http://www.adastral.ucl.ac.uk/~helger/crypto/
http://en.wikipedia.org/wiki/GOST_28147-89
http://en.wikipedia.org/wiki/Cyclic_redundancy_check
http://id2.php.net/manual/en/ref.mhash.php
http://www.ietf.org/html.charters/openpgp-charter.html
http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html
http://labs.calyptix.com/crypto/haval.php
http://md5.rednoize.com/
http://mhash.sourceforge.net/
http://pajhome.org.uk/crypt/md5/
http://planeta.terra.com.br/informatica/paulobarreto/hflounge.html
http://tools.ietf.org/html/rfc3309
http://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html
http://tools.ietf.org/html/rfc1950
http://www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html#GOST
http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html
http://www.cryptodox.com/MD5

http://www.cryptography.com/cnews/hash.html
http://www.faqs.org/rfcs/rfc1320.html
http://www.md5hashes.com/
http://www.uic.rsu.ru/doc/web/php-4.0.3pl1/function.mhash-count.html
http://www.uic.rsu.ru/doc/web/php-4.0.3pl1/function.mhash-get-hash-name.html
http://xyssl.org/code/source/md5/
http://vipul.net/gost/
http://www.zorc.breitbandkatze.de/crc.html
Zheng, Yuliang. HAVAL — A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract).1993 : Australia.