Analisis Fungsi dari Algoritma Hash Radio Gatun

Agung Dwi Lambang Gito Santosa - 13508086
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
gerrard_io@yahoo.co.id
if18086@students.if.itb.ac.id

Abstract—Radio Gatun adalah sebuah algoritma fungsi hash yang dibuat oleh Guido Bertoni, Joan Daemen, Michael Peeters dan Gilles Van Assche. Fungsi hash ini masih belum banyak dikenal pada kehidupan sehari-hari, padahala pada kenyataanya algoritma hash ini memiliki keunggulan dalam performansi dan cepat pada tingkat hardware. Pada makalah ini penulis ingin menganalisis fungsi dari algoritma ini dan memaparkan beberapa hal menarik yang ada dalam algoritma hash Radio Gatun.

***Kata kunci*—hash, performa, Radio Gatun.**

I. PENDAHULUAN

1.1. LATAR BELAKANG

Radio Gatun merupakan salah satu dari banyak hash fungsi yang ada selama ini, di perkenalkan oleh Bertoni , Daemen, Peeters dan Van Assche di the Second NIST Hash Workshop. Fungsi hash ini sangat menarik untuk dipelajari karena desainya yang tidak sama seperti fungsi hash tradisional. Fungsi hash ini tidak berdasarkan pada blockchipher seperti fungsi konstruksi Davies-Meyer dan tidak menggunakan paradigma Merkle-Damgard untuk mentransformasikan sebuah fungsi kompresi menjadi sebuah fungsi hash. fungsi hash ini mengembangkan desain sebelumnya yang digunakan oleh fungsi hash Panama.

1.2. TEORI SINGKAT

1.2.1. FUNGSI HASH KRIPTOGRAFI

Fungsi hash kriptografi adalah suatu prosedur yang mengambil sebuah blok data dan mengembalikan sebuah bit string yang unkurannya tetap, pengubahan data akan mengubah nilai hash value yang dihasilkan. data yang akan diproses disebut “message” dan hasilnya berupa sebuah “message digest”.

Fungsi hash kriptografi yang ideal memiliki 4 hal penting yang harus diperhatikan :

1. Mudah untuk di hitung hash value-nya untuk message apapun.
2. Tidak memungkinkan mengenerate message menggunakan hash value yang ada.
3. Tidak memungkinkan utnuk memodifikasi message tanpa mengubah nilai hashnya
4. Tidak memungkinkan terdapat 2 message yang memilki nilai hash yang sama.

Fungsi hash kriptografi memiliki banyak aplikasi keamanan informasi, seperti digital signatures, MAC, dan banyak yang lainnya. Mereka juga dapat digunakan sebagai fungsi hash biasa, untuk data indeks dalam tabel hash, untuk sidik jari, untuk mendeteksi duplikat data atau unik mengidentifikasi file, dan sebagai checksum untuk mendeteksi korupsi data yang disengaja.

Memang, dalam konteks keamanan informasi, nilai-nilai hash kriptografi kadang-kadang disebut (digital) sidik jari, checksum, atau hanya nilai hash, meskipun semua istilah ini berdiri untuk fungsi dengan sifat yang agak berbeda dan tujuan. Kebanyakan fungsi hash kriptografi dirancang untuk mengambil string dari setiap panjang sebagai masukan dan menghasilkan nilai hash fixed-panjang.

Sebuah fungsi hash kriptografi harus dapat menahan semuah tipe serangan dari kriptanalis. Sebagai syarat minimum, fungsi tersebut harus memiliki beberapa hal di bawah ini :

1. *Preimage resistance*
   Diberikan sebuah hash (H) yang harus sulit untuk dicari messagenya (M). sehingga H = hash(M). kosep ini berkaitan dengan *one-way function.* Fungsi yang tidak memiliki hal ini akan sangat mudah terkena *preimage attack.*
2. *Second preimage resistance*
   Diberikan sebuah masukan M1 yang sangat sulit mencari M2 dengan syarat M1 tidak samadengan M2. Sehingga hash(M1)= hash(M2). Hal ini berkaitan dengan *weak collision resistance*, dan fungsi yang kekurangan hal ini akan sangat mudah terkena *second perimage attack.*
3. *Collision resistance*
   Seharusnya akan sangat sulit menemukan 2 pesan yang berbeda yang memiliki nilai hash yang sama. Pasangan pesan yang memiliki nilai hash yang sama disebut *cryptographic hash collision.* Hal ini membutuhkan setidaknya nilai hash yang memiliki pangjang 2 kali dari yang dibutuhkan pada *preimage-resistance.* Jika tidak

collision bisa ditemukan dengan cara *birthday attack.*

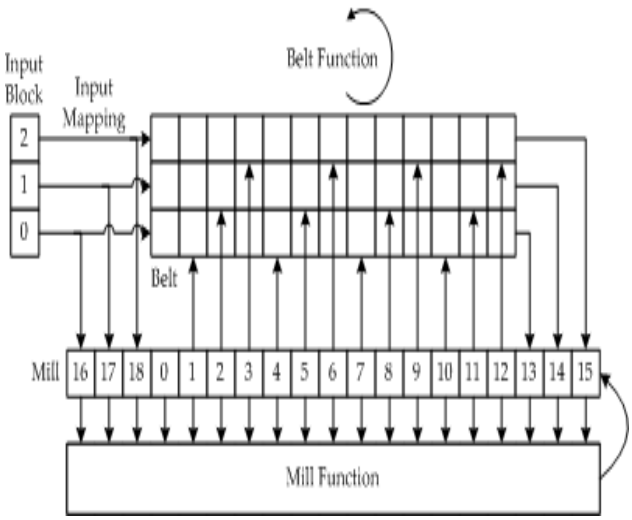
Hal ini membuktikan bahwa kita tidak dapat mengganti input tanpa mengubah message digestnya juga. Jadi dua string memiliki nilai hash yang sama, maka dapat dipastikan kedua string tersebut adalah sama.

Sebuah ilustrasi potensi penggunaan hash kriptografi adalah sebagai berikut: Alice memiliki soal matematika yang sulit untuk Bob, dan mengklaim dia telah memecahkan itu. Bob ingin mencoba sendiri, tapi dia ingin untuk memastikan bahwa Alice tidak berbohong. Oleh karena itu, Alice menuliskan solusi nya, akan menambahkan Nonce acak, menghitung hash dan memberitahu Bob nilai hash (sementara menjaga rahasia solusi dan Nonce). Dengan cara ini, ketika Bob datang dengan solusi sendiri beberapa hari kemudian, Alice dapat membuktikan bahwa ia punya solusi sebelumnya dengan mengungkapkan Nonce untuk Bob. (Ini adalah contoh skema komitmen sederhana, dalam praktek yang sebenarnya

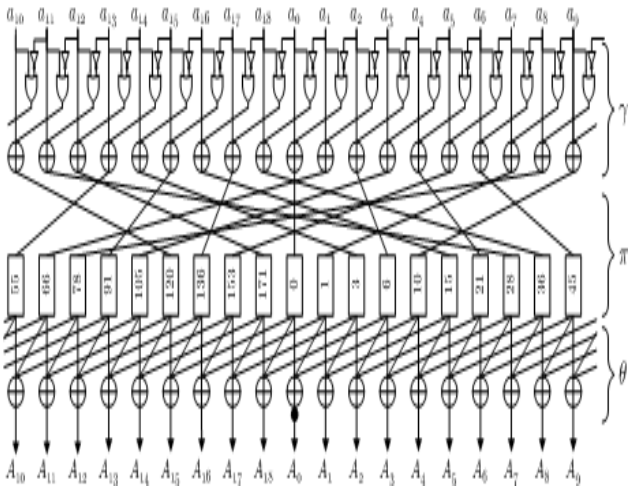
.

1.2.2. RADIO GATUN

Fungsi hash ini tidak berdasarkan pada blockchipher seperti fungsi konstruksi Davies-Meyer dan tidak menggunakan paradigma Merkle-Damgard untuk mentransformasikan sebuah fungsi kompresi menjadi sebuah fungsi hash. fungsi hash ini mengembangkan desain sebelumnya yang digunakan oleh fungsi hash Panama. Radio Gatun dibagi menjadi dua bagian, *belt and mill*



Operasi yang digunakan dalam fungsi hash ini adalah operasi logikal seperti AND, XOR, NOT dan penggeseran kata. Sebernarnya ukuran kata yang dapat diproses oleh fungsi ini adalah antara 1 sampa 64. Dengan menggunakan 64-bit sebagai pemilihan default untuk 64-bit platforms. Untuk performa yang maksimal pada 32-bit platform kita sebaiknya menggunaknan radiogatun dengan 32-bit words.



Salah satu keuntungan menggunakan radiogatun ada pada performa yang baik. Dalam tingkat *software*, radiogatun cukup kompetitif jika dibandingkan dengan SHA-1. Juga radiogatung sangat cepat pada tingkat *hardware*.

Sebagai catatan bahawa radiogatun tidak menggunakan konstruksi *sponge* jadi tidak dapat disebut fungsi *sponge.* Secara spesifik perbedaan antara radiogatun dan konstruksi *sponge* adalah sebagai berikut :

1. Pada konstruksi *sponge* input dan output diproses pada state yang sama, hal ini tidak terjadi pada radiogatun.
2. Di radiogatun terdapat beberapa round kosong antara input dan output. Tetapi tidak ada round kosong pada fungsi *sponge.*

Di bawah ini adalah contoh penggunaan radiogatun :

```
RadioGatun[32]("") =
F30028B54AFAB6B3E55355D277711109A19BED
A7091067E9A492FB5ED9F20117
RadioGatun[32]("The quick brown fox
jumps over the lazy dog") =
191589005FEC1F2A248F96A16E9553BF38D0AE
E1648FFA036655CE29C2E229AE
RadioGatun[32]("The quick brown fox
jumps over the lazy cog") =
EBDC1C8DCD54DEB47EEEFC33CA0809AD23CD9F
FC0B5254BE0FDABB713477F2BD
RadioGatun[64]("") =
64A9A7FA139905B57BDAB35D33AA216370D5EA
E13E77BFCDD85513408311A584
RadioGatun[64]("The quick brown fox
jumps over the lazy dog") =
6219FB8DAD92EBE5B2F7D18318F8DA13CECBF1
3289D79F5ABF4D253C6904C807
RadioGatun[64]("The quick brown fox
jumps over the lazy cog") =
C06265CAC961EA74912695EBF20F1C256A338B
C0E980853A3EEF188D4B06FCE5
```

II. PROSES ANALISIS FUNGSI RADIO GATUN

Untuk proses analisis fungsi dari radiogatun, penulis menggunakan radiogatun32-bit sebagai acuan dalam mengganalisis fungsi RadioGatun.

Dibawah ini adalah file Rg32.h yang digunakan penulis sebagai header dari file Rg32.cpp :

```
#include <iostream>
#include <stdint.h>

class RG32
{
public:
        RG32 (char *data);
        RG32 (char *data, uint16_t spice);
        uint32_t Num ();
private:
        void init();
        void mill();
        void belt();
        void input_char(char a);
        uint32_t a[20]; // sebagai Mill
        uint32_t b[40]; // sebagai Belt
        uint32_t inputan; // masukan word
        uint32_t p[3]; // masukan words
        int place; // Place of byte in input word
        int mplace; // Word from mill to use in
output
        int pword; // Word (we input three at a
time) to add input to
};
```

Sedangkan file Rg32.cpp berisi :

```
RG32::RG32(char *in)
{
        int c = 0;
        RG32::init();
        while(*in != 0)
        {
                RG32::input_char(*in);
                in++;
        }
        RG32::input_char(1); // 1 at end
        if(pword == 0 && inputan == 0)
        {
                for(c = 0; c < 16; c++)
                {
                        RG32::belt();
                }
                return;
        }
        p[pword % 3] = inputan;
        for(c = 0; c < 3; c++) // Map memasukan
ke belt dan mill
        {
                b[c * 13] ^= p[c];
                a[16 + c] ^= p[c];
        }
        for(c = 0; c < 17; c++)
        {
                RG32::belt();
        }

}

uint32_t RG32::Num()
{
        uint32_t out;
        if(mplace < 1 || mplace > 2) {
                RG32::belt();
                mplace = 1;
        }
        out = a[mplace];
        mplace++;
        // penggantian edian untuk pengetesan
vektor
        out = (out << 24) | ((out & 0xff00) << 8)
| ((out & 0xff0000) >> 8) |
                (out >> 24);
        return out;
}

void RG32::init ()
{
        int z;
        for(z = 0; z < 20; z++)
        {
                a[z] = 0;
        }
        for(z = 0; z < 40; z++)
        {
                b[z] = 0;
        }
        place = 0;
        inputan = 0;
        pword = 0;
        mplace = 0;
}

void RG32::mill ()
{
        uint32_t A[19];
        uint32_t x;
        int i = 0, y = 0, r = 0, z = 0, q = 0;
        for(i = 0; i < 19; i++)
        {
                y = (i * 7) % 19;
                r = ((i * (i + 1)) / 2) % 32;
                x = a[y] ^ (a[ ((y + 1) % 19) ] |
(~a [ ((y + 2) % 19) ]));
                A[i] = (x >> r) | (x << (32 -
r));
        }
        for(i = 0; i < 19; i++)
        {
                y = i;
                z = (i + 1) % 19;
                q = (i + 4) % 19;
                a[i] = A[y] ^ A[z] ^ A[q];
        }
        a[0] ^= 1;
}

void RG32::belt ()
{
        uint32_t q[3];
        int s = 0, i = 0, v = 0;
        for(s = 0; s < 3; s++)
        {
                q[s] = b[((s * 13) + 12)];
        }
        for(i = 12; i > 0; i--)
        {
                for(s = 0; s < 3; s++)
                {
                        v = i - 1;
                        if(v < 0)
                        {
                                v = 12;
                        }
                        b[(s * 13) + i] = b[(s *
13) + v];
                }
        }
        for(s = 0; s < 3; s++)
        {
                b[(s * 13)] = q[s];
        }
        for(i = 0; i < 12; i++)
        {
```

```
                s = (i + 1) + ((i % 3) * 13);
                b[s] ^= a[(i + 1)];
        }
        RG32::mill();
        for(i = 0; i < 3; i++)
        {
                a[(i + 13)] ^= q[i];
        }
}

void RG32::input_char(char input)
{
        int q = 0, c = 0, r = 0, done = 0;

        q = input;
        q &= 0xff;
        place++;

        if(place == 1)
        {
                inputan |= q;
                return;
        }
        else if(place == 2)
        {
                inputan |= q << 8;
                return;
        }
        else if(place == 3)
        {
                inputan |= q << 16;
                return;
        }
        else if(place > 5) //Untuk berjaga – jaga
seharusnya tidak akan terjadi
        {
                place = 0;
                inputan = 0;
                return;
        }

        inputan |= q << 24;

        p[pword] = inputan;
        inputan = 0;
        place = 0;
        pword++;
        if(pword < 3)
        {
                return;
        }
        else if(pword > 3)
        {
                pword = 0;
                return;
        }

        //  inputan  and  pword  telah  selesai;
jalankan belt
        for(c = 0; c < 3; c++) {
                b[c * 13] ^= p[c];
                a[16 + c] ^= p[c];
                p[c] = 0;
        }
        RG32::belt();
        pword = 0;
        place = 0;
        return;

}

// fungsi untuk menggenerate number yang benar
int show_num(char *num) {
        RG32 *state;

        state = new RG32(num);
        printf("%s %x\n",num,state->Num());
        delete state;
}
```

Terdapat beberapa fungsi diantaranya adalah fungsi mill dan belt sebagai fungsi utama dalam radiogatun.

```
RadioGatun[32]("cobacoba") = 191589005FEC1F2A240E73A66E9553BF38D0AEE1648FFA03665
5CE29C2E229AE
RadioGatun[32]("") = 005CE29C24E2251F2A2485E73A29C2873663E9553BF38D0AEE16A24848F
FA036
RadioGatun[32]("cobacob") = 129854BA939E9C239F940255B3BB31C6A7A8E8D873493DBC1923
4810047736AB3
Press any key to continue . . .
```

Gambar diatas adalah hasil dari eksekusi program, pada contoh yang pertama penulis mencoba untuk memasukan kata “cobacoba” ,sedangkan contoh kedua penulis mencoba memasukan string kosong, dan contoh ketiga penulis mencoba memasukan kata “cobacob”.

Dari hasil diatas dapat kita lihat untuk contoh 1 dan 3 memiliki stirng yang hampir sama, tetapi menghasilkan nilai hash yang sangat berbeda. Jadi radiogatun telah untuk sementara ini radiogatun telah memenuhi syarat untuk menjadi suatu fungsi hash.

Dan juga jika kita masukan nilai hash dari contoh 1 sebagai message akan menghasilkan nilai hash yang berbeda (bukan “cobacoba”) ini membukatikan bahwa *one-way function* berlaku pada algoritma ini.

Tetapi apakah fungsi hash ini benar-benar aman dari serangan-serangan kriptananlis ? dalam paper “two attack on radiogatun” oleh Dmitry Khovratovich dan Alex Birykov,dua serangan yang tidak merusak keamanan radiogatun, satu dengan compleksitas 2^(18*w) dan yang lain dengan comlpeksitas 2^(23,1*w).

Selain itu pada paper “Analysis of the Collision Resistance of RadioGatun using Algebraic Techniques” oleh Charles Bouillaguet dan Pierre Alain Fouque, mereka menunjukan cara mendapatkan collision pada radiogatun 1-bit, serangan yang mereka lakukan membutuhkan 2^24,5 operasi. Tetapi serangan yang mereka lakukan tidak dapat diperluas menggunakan selain 1–bit. Jadi serangan ini juga belum efektif untuk meruntuhkan keamanan dari radiogatun.

Serangan yang paling efektif dilakukan ke radiogatun adalah pada paper “Cryptanalysis of RadioGatun” oleh Thimas Fuhr dan Thomas Peyrin, dimana serangan dilakukan dengan dengan kompleksitas 2^(11*w), tetapi serangan ini tetap tidak dapat menembus tingkat keamanan dari radiogatun.

IV. KESIMPULAN

Dari proses analisis dari berbagai sumber dan beberapa tes pada fungsi hash radio gatun penulis menyimpulkan bahwa radio gatun merupakan salah satu fungsi hash yang

aman. Tetapi pada aplikasi sehari-hari masih belum banyak dikenal dan kalah bersaing dengan fungsi sejenis seperti SHA-1 dll. Penulis berharap dengan membaca makalah ini pembaca akan lebih tertarik dengan fungsi hash radiogatun dan menggembangkanya pada kehidupan sehari – hari.

REFERENSI

http://en.wikipedia.org/wiki/RadioGat%C3%BAn tanggal akses 8 Mei 2011

http://en.wikipedia.org/wiki/Cryptographic_hash_function tanggal akses 8 Mei 2011

http://www.samiam.org/rg32/ tanggal akses 8 Mei 2011

http://radiogatun.noekeon.org/ tanggal akses 8 Mei 2011

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, 9 Mei 2011

Agung DLGS 13508086