FILES AUTHENTICATION CODE (FAC): PENGEMBANGAN MAC UNTUK DIGUNAKAN DALAM MENJAMIN INTEGRITAS BERKAS

Raditya Arief – NIM 13507030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if17030@students.if.itb.ac.id

***Abstract*— Saat ini penggunaan berbagai jaringan untuk berkirim data sudah menjadi hal yang lumrah. Bahkan dalam beberapa kejadian, hal ini menjadi tumpuan agar data-data yang diinginkan dapat dikirim dengan selamat. Namun seiiring perkembangan zaman, dan semakin dibutuhkannya fasilitas untuk berkirim data, semakin banyak juga peluang kejahatan yang mungkin terjadi. Kejahatan seperti pemalsuan data sangat dihindari saat ini karena dapat berakibat data yang diterima tidak lagi valid, dan integritasnya dipertanyakan. Kemudian juga terdapat suatu jenis serangan yang dilakukan oleh virus komputer. Berbagai jenis virus terkadang memang tidak merusak data, dalam arti tidak merubah atau menghilangkan isi dari data tersebut, namun virus tersebut dapat menyisipkan sebuah kode jahat ke dalam data yang dikirim. Kode jahat ini akan tereksekusi saat data yang terinfeksi dijalankan, dan efek dari kode jahat ini sudah pasti akan berakibat buruk kepada korban.**

Oleh karena itu, dibutuhkan suatu teknik yang dapat memberi peringatan, atau setidaknya memberi suatu tanda, apabila terjadi suatu perubahan pada file-file yang telah dikirim. Saat konten suatu file saat diterima diketahui tidak sama dengan kontennya saat dikirim, maka integritas file tersebut dapat diragukan. Apabila sebuah sistem notifikasi seperti ini dapat diterapkan, maka para pengguna dapat lebih waspada dan lebih awas terhadap integritas suatu file saat dikirim. Saat ini, sudah terdapat sebuah teknik yang dapat digunakan untuk memeriksa keaslian sebuah data, yaitu sebuah teknik yang disebut dengan MAC.

Terkadang saat user berkirim data, mereka tidak hanya mengirimkan satu file, namun langsung beberapa file. File-file ini bisa tergabung dalam sebuah folder, atau bisa berada di dalam sebuah drive eksternal. Dengan menggunakan teknik MAC, user dapat memastikan apakah sebuah file tersebut dapat dijamin integritasnya, atau tidak. Namun terdapat permasalah dengan teknik MAC ini. MAC hanya dapat digunakan untuk memeriksa integritas suatu file satu-persatu, yang tentu akan memakan banyak waktu. Kemudian permasalahan lain, MAC tidak dapat digunakan untuk memeriksa apabila ternyata terdapat sebuah file yang disisipkan pada folder yang dikirim.

Maka dari itu, dapat diusulkan sebuah teknik baru yang disebut FAC. FAC ini nantinya akan memeriksa nilai hash dari seluruh file yang ada di dalam folder. Diharapkan dengan menggunakan FAC ini, integritas suatu folder dapat diketahui.

***Index Terms*— FAC, MAC, File, DES.**

I. PENDAHULUAN

Kriptografi merupakan secara umum adalah ilmu dan seni untuk menjaga kerahasiaan berita. Kemudian terdapat pengertian bahwa kriptografi adalah teknik matematika yang berhubungan dengan aspek keamanan informasi. Namun tidak semua aspek keamanan informasi ditangani oleh kriptografi.

Ada empat tujuan mendasar dari ilmu kriptografi yang juga merupakan aspek keamanan informasi. Empat tujuan mendasar tersebut yaitu:

- **Kerahasiaan**: bagaimana suatu informasi yang dirahasiakan dijaga agar tidak dapat diketahui oleh siapapun kecuali mereka yang memiliki kunci
- **Integritas**: bagaimana suatu informasi dapat diketahui keasliannya. Apabila informasi tersebut ditambah, dikurangi, dirubah, ataupun dihilangkan, maka sistem ini haruslah dapat mendeteksi perubahan yang tidak sah tersebut
- **Autentikasi**: bagaimana sebuah sistem dapat mengautentikasi seluruh informasi yang dikirimkan saat ada pihak-pihak yang sedang berkomunikasi
- **Non-repudiasi**: adanya pencegahan terhadap usaha penyangkalan terhadap informasi yang telah dikirm oleh pihak pengirim

Tercatat ilmu kriptografi sudah ada sejak 4000 tahun lalu, digunakan oleh bangsa Mesir Kuno. Sejak saat itu, ilmu kriptografi terus berkembang secara paralel/terpisah sampai hari ini. Momen penggunaan ilmu kriptografi yang cukup terkenal sepanjang sejarah yaitu peran ilmu kriptografi pada perang dunia pertama dan kedua. Keberhasilan sekutu dalam memecahkan sandi kriptografi Nazi Jerman pada saat itu dianggap sebagai salah penyebab kemenangan pihak sekutu.

Saat ini ilmu kriptografi masih terus berkembang. Hanya saja, penggunaan ilmu ini sudah semakin meluas dan

jumlah kriptografer di seluruh dunia juga sudah bertambah seiring dengan meningkatnya popularitas ilmu kriptografi. Awalnya ilmu ini dominan digunakan pada bidang militer, diplomatik, atau pemerintahan secara umum. Namun kemudian ilmu ini juga digunakan oleh pihak swasta sebagai alat untuk melindungi informasi yang bersifat vital agar kerahasiaannya dapat dijaga.

Seiring perkembangan ilmu kriptografi dan jumlah kriptografer, bermunculan pula jenis-jenis teknik kriptografi yang baru. Secara umum, ilmu kriptografi saat ini dapat diklasifikasi sebagai berikut:

- Algoritma Sandi
    - Algoritma sandi kunci-simetris
        - Block-Cipher
        - Stream-Cipher
        - Algoritma-algoritma sandi kunci-simetris
    - Algoritma Sandi Kunci-Asimetris
        - Fungsi Enkripsi dan Dekripsi Algoritma Sandi Kunci-Asimetris
        - Algoritma -Algoritma Sandi Kunci-Asimetris
- Fungsi Hash Kriptografis
    - Sifat-Sifat Fungsi Hash Kriptografi
    - Algoritma-Algoritma Fungsi Hash Kriptografi

Fungsi sandi bertujuan untuk mengamankan informasi. Dua hal yang harus dimiliki oleh fungsi sandi adalah kekuatan konfusi/pembingungan, dan difusi/peleburan. Dengan menggunakan algoritma jenis ini, diharapkan suatu informasi akan aman dari mereka yang tidak berhak.

Kemudian fungsi kriptografi yang kedua, yaitu fungsi hash. Fungsi hash berbeda dengan fungsi sandi yang bertujuan mengamankan data, fungsi hash umumnya digunakan untuk keperluan autentikasi dan integritas data. Dengan menggunakan fungsi hash, maka manipulasi terhadap informasi dapat diketahui.

Dalam makalah kali ini penulis tidak akan membahas semua teknik-teknik kriptografi yang ada, penulis akan khusus membahas teknik kriptografi yang berhubungan dengan teknik MAC (Message Authentication Code).

II. MESSAGE AUTHENTICATION CODE

MAC adalah fungsi satu arah yang menggunakan kunci rahasia dalam pembangkitan nilai hash. MAC merupakan salah satu teknik kriptografi dalam klasifikasi yang menggunakan fungsi hash. Berbeda dengan fungsi MD5 atau SHA-1 yang tidak memerlukan kunci untuk membangkitkan nilai hash, MAC memerlukan kunci dalam membangkitkan nilai hash. Seperti kebanyakan teknik hash yang lain, MAC akan menghasilkan nilai hash yang panjangnya tetap (fixed). Biasanya MAC dilekatkan dengan pesan yang dikirim yang selanjutnya akan digunakan untuk autentikasi tanpa merahasiakan pesan. MAC bukanlah tanda tangan digital, namun hanya digunakan untuk menyediakan autentikasi pengirim.

$$MAC = C_K(M)$$

MAC = nilai hash
C = fungsi hash (atau algoritma MAC)
K = kunci rahasia

Dengan menggunakan sebuah fungsi hash, dan sebuah kunci rahasia, maka akan didapat nilai hash MAC.

Dalam pengembangan MAC ini, penulis mencoba menggunakan dua bahasa pemrograman, yaitu Java dan C#. Pada bahasa Java, penulis menemukan bahwa sudah terdapat suatu fungsi bagi fitur kriptografi MAC. Sedangkan untuk C#, penulis menggunakan fungsi SHA-1 sebagai fungsi hash. Pada penelitian yang penulis lakukan, penulis menemukan dua jenis fungsi hash yang sering digunakan yaitu MD5 (HMAC-MD5) dan SHA-1 (HMAC-SHA1).

III. FILES AUTHENTICATION CODE

Kemudian berdasarkan latar belakang yang telah dipaparkan sebelumnya, penulis akan mencoba mengembangkan suatu teknik untuk mengautentikasi isi dari sebuah drive atau folder. Teknik ini dinamakan Files Authentication Code (FAC). Secara umum, perbedaan antara FAC dan MAC hanya terdapat pada masukan fungsi. Apabila masukan dari fungsi hash MAC berupa *array of byte* dari file masukan, maka masukan fungsi hash FAC adalah *concatenation* dari seluruh *array of byte* yang terdapat pada folder atau drive tersebut.

Namun masih terdapat beberapa permasalahan yang harus diuji. Permasalahan tersebut antara lain sensitifitas algoritma terhadap perubahan sekecil apapun, apakah FAC dapat menyertakan seluruh file dalam folder sebagai input (termasuk file dengan status hidden), bagaimana pengaruh penambahan, pengurangan, dan manipulasi lainnya terhadap nilai hash FAC.

IV. KEJAHATAN FILE

Berikut akan dijelaskan sedikit mengenai bagaimana kejahatan file dilakukan dan jenis-jenisnya.

A. VIRUS

Virus merupakan salah satu jenis kejahatan file yang berupa penambahan suatu program jahat ke dalam folder. Nantinya virus ini dapat mengakibatkan suatu hal yang tidak diinginkan.

B. PENAMBAHAN FILE

Penambahan file dapat berupa suatu kegiatan memasukkan file tambahan yang bukan dibuat oleh pengirim. Dengan penambahan file ini, penerima pesan akan dibuat bingung dengan tidak mengetahui file mana yang dikirim oleh pengirim asli, dan mana yang bukan.

C. PENGURANGAN FILE

Pengurangan file dapat berupa manipulasi isi folder kiriman dengan menghilangkan semua atau sebagian file yang terdapat dalam folder.

D. PERUBAHAN NAMA FILE

Dengan perubahan nama file ini, maka informasi yang tadinya tersusun dengan rapih, dapat menjadi tidak teratur. Nama file ini memang tidak merubah secara keseluruhan isi dari file, namun hal ini tetap dapat berakibat fatal. Sebagai contoh, pada file-file notulensi rapat yang tergabung pada satu folder. Apabila nama-nama file notulensi ini ditukar, maka akan terjadi suatu mis-komunikasi mengenai pencatatan kejadian pada rapat.

E. PERUBAHAN ISI FILE

Perubahan pada isi file dapat berpengaruh besar, mulai dari perubahan informasi yang ingin dikirimkan, sampai hilangnya informasi.

F. PERUBAHAN BYTE FILE

Perubahan satu byte pada sebuah file mungkin terdengar tidak signifikan, namun dapat berakibat sebuah file tidak dapat dibuka, atau rusak sama sekali.

Kejahatan file yang sudah dipaparkan di atas akan menjadi sorotan penulis apakah file dengan menggunakan teknik FAC dapat dianalisa autentikasinya dengan akurat.

V. PROGRAM

Disini penulis mencoba menggunakan dua buah program dengan dua bahasa pemrograman yang berbeda, yaitu pada Java dan pada C#.

Program pada Java ini menggunakan empat buat library Java:

- javax.crypto.Mac
- javax.crypto.KeyGenerator
- javax.crypto.SecretKey
- java.io.FileInputStream

Dengan menggunakan library ini, maka dapat di-*instance* sebuah tipe MAC,

```
Mac mac = Mac.getInstance("HmacSHA1");
```

Yang dimaksud dengan “HmacSHA1” adalah Hash MAC SHA-1. Tipe lain yang dapat digunakan yaitu HmacMD5.

Kemudian variabel *mac* harus dinisiasi dengan,

```
mac.init(key);
```

Dengan key yaitu berupa sebuah tipe SecretKey.

```
SecretKey key = kg.generateKey();
```

Dimana *kg* adalah sebuah tipe KeyGenerator,

```
KeyGenerator kg = KeyGenerator.getInstance("DES");
```

Lalu dengan menggunakan sebuah fungsi bernama FAC yang akan mengembalikan *array of bytes*,

```
byte[] macbytes = FAC(key, datafile, mac);
```

Dan kemudian variable *macbytes* ini dapat ditampilkan sebagai hasil dari fungsi hash. Permasalahan yang muncul disini adalah bagaimana masukkan dapat berupa gabungan *array of bytes* file-file yang ada di dalam sebuah folder.

Kemudian program yang digunakan untuk fungsi hash pada bahasa C# menggunakan fungsi SHA-1. Jadi permasalahan yang terdapat pada program C# hanya bagaimana penulis memasukkan gabungan *array of bytes* ke dalam fungsi hash.

VI. SOURCE CODE

Berikut akan dijabarkan source code program dari Java:

```
package uaskripto;

// File: src\jsbook\ch3\ComputeMAC.java
import javax.crypto.Mac;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import java.io.FileInputStream;
import java.security.KeyStore;
/**
 *
 * @author radit
 */
   public class Main {
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] unused) throws Exception{
    String datafile = "D:\\Test/Test2.txt";

    KeyGenerator kg = KeyGenerator.getInstance("DES");
    kg.init(56); // 56 is the keysize. Fixed for DES
    SecretKey key = kg.generateKey();

    /*Mac mac = Mac.getInstance("HmacSHA1");
    mac.init(key);

    FileInputStream fis = new FileInputStream(datafile);
    byte[] dataBytes = new byte[1024];
    int nread = fis.read(dataBytes);
    while (nread > 0) {
      mac.update(dataBytes, 0, nread);
      nread = fis.read(dataBytes);
    };
    byte[] macbytes = mac.doFinal();
    String digestB64 = new sun.misc.BASE64Encoder().encode(macbytes);
    System.out.println("MAC(in hex):: " + macbytes);
    System.out.println("MAC(in hex):: " + macbytes);*/

    Mac mac = Mac.getInstance("HmacSHA1");
    mac.init(key);

    byte[] macbytes = FAC(key, datafile, mac);
    System.out.println("MAC(in hex):: " + macbytes);

    macbytes = FAC(key, datafile, mac);
    System.out.println("MAC(in hex):: " + macbytes);
   }

    public static byte[] FAC(SecretKey key, String datafile, Mac mac) throws Exception
    {
      System.out.println("MAC(init):: " + key.toString());
      System.out.println("MAC(init):: " + mac.toString());

      FileInputStream fis = new FileInputStream(datafile);
      System.out.println("MAC(init):: " + fis.toString());
      byte[] dataBytes = new byte[1024];
      //int nread = fis.read(dataBytes);
      /*while (nread > 0) {
        mac.update(dataBytes, 0, nread);
        nread = fis.read(dataBytes);
      }*/
      byte[] macbytes = mac.doFinal();
      //String digestB64 = new sun.misc.BASE64Encoder().encode(macbytes);
      //System.out.println("MAC(in hex):: " + macbytes);

      return macbytes;
    }
}
```

Pada program Java ini, penulis belum berhasil menerapkan library Mac yang sudah ada dari java. Berdasarkan percobaan yang dilakukan penulis, belum berhasil didapatkan suatu cara untuk menyimpan kunci yang digunakan untuk menciptakan nilai hash, sehingga penulis belum dapat membandingkan nilai hash.

Oleh karena itu, penulis akan memfokuskan menggunakan program dengan bahasa C#. Pada program ini, digunakan fungsi hash SHA-1 yang menggunakan generator kunci terpisah. Berikut adalah source code fungsi hash SHA-1:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;

namespace DigitalSignature
{
  class SHA1
  {
    private static uint mA = 0x67452301;
    private static uint mB = 0xEFCDAB89;
    private static uint mC = 0x98BADCFE;
    private static uint mD = 0x10325476;
    private static uint mE = 0xC3D2E1F0;

    public static byte[] padding(byte[] inputMsg)
    {
      ulong msgLength = (ulong)inputMsg.Length;
      ulong blockNum = (ulong)Math.Floor((double)((msgLength + 8) / 64) + 1);
      ulong padLength = ((blockNum * 64) - (msgLength + 8));
      byte[] tLength = BitConverter.GetBytes(msgLength * 8);
      Array.Reverse(tLength);
      byte[] tMsg = new byte[(blockNum * 64)];
      for (ulong x = 0; x < msgLength; x++)
      {
        tMsg[x] = inputMsg[x];
      }
      tMsg[msgLength] = 128;
      for (ulong i = 1; i < padLength; i++)
      {
        tMsg[(msgLength + i)] = 0;
      }
```

```
            for (ulong j = 0; j < 8; j++)
            {
               tMsg[msgLength + padLength + j] = tLength[j];
            }
            return tMsg;
         }

         public static uint[][] MsgBlock(byte[] bMsg)
         {
            ulong msgLength = (ulong)bMsg.Length;
            ulong bLength = msgLength / 64;
            uint[][] mBlock = new uint[bLength][];
            byte[] temp = new byte[4];
            for (ulong i = 0; i < bLength; i++)
            {
               mBlock[i] = new uint[16];
               for (int j = 0; j < 16; j++)
               {
                  for (int k = 0; k < 4; k++)
                  {
                     temp[k] = bMsg[((4 * j) + k)];
                  }
                  mBlock[i][j] = ByteToInt(temp);
               }
            }
            return mBlock;
         }

         public static void tohex(byte[] test)
         {
            //Console.WriteLine(conv.Length);
         }

         public static uint ByteToInt(byte[] input)
         {
            uint tW = ((uint) input[0]) << 24;
            tW |= ((uint)input[1]) << 16;
            tW |= ((uint)input[2]) << 8;
            tW |= ((uint)input[3]);
            return tW;
         }

         public static string Enkripsi(byte[] pMessage)
         {
            mA = 0x67452301;
            mB = 0xEFCDAB89;
            mC = 0x98BADCFE;
            mD = 0x10325476;
            mE = 0xC3D2E1F0;

            byte[] tPadding = padding(pMessage);
            uint[][] tUintMessage = MsgBlock(tPadding);
            for (int i = 0; i < tUintMessage.Length; i++)
            {
               HSHA(tUintMessage[i]);
            }
            return (ConvertToHexString(mA) + ConvertToHexString(mB) + ConvertToHexString(mC) + ConvertToHexString(mD) + ConvertToHexString(mE));
         }

         public static void HSHA(uint[] pBlock)
         {
            uint[] tListW = new uint[80];
            uint temp, tA, tB, tC, tD, tE;

            for (int t = 0; t < 16; t++)
            {
               tListW[t] = pBlock[t];
            }

            for (int t = 16; t < 80; t++)
            {
               tListW[t] = CircularShiftLeft(1, tListW[t - 3] ^ tListW[t - 8] ^ tListW[t - 14] ^ tListW[t - 16]);
            }

            tA = mA;
            tB = mB;
            tC = mC;
            tD = mD;
            tE = mE;

            for (int i = 0; i < 80; i++)
            {
               uint tF = F(i, tB, tC, tD);
               uint tCLS5 = CircularShiftLeft(5, tA);
               uint tK = K(i);

               temp = tF + tCLS5 + tE + tListW[i] + tK;
               tE = tD;
               tD = tC;
               tC = CircularShiftLeft(30, tB);
               tB = tA;
               tA = temp;
            }

            mA = mA + tA;
            mB = mB + tB;
            mC = mC + tC;
            mD = mD + tD;
            mE = mE + tE;
         }

         private static uint CircularShiftLeft(int pBits, uint pWord)
         {
            return (pWord << pBits) | (pWord >> (32 - pBits));
         }

         public static uint F(int pT, uint pB, uint pC, uint
```

```
pD)
        {
            uint tTemp = 0x00000000;

            if ((pT >= 0) && (pT <= 19))
            {
                tTemp = ((pB & pC) | ((~pB) & pD));
            }
            else if ((pT >= 20) && (pT <= 39))
            {
                tTemp = pB ^ pC ^ pD;
            }
            else if ((pT >= 40) && (pT <= 59))
            {
                tTemp = (pB & pC) | (pB & pD) | (pC & pD);
            }
            else if ((pT >= 60) && (pT <= 79))
            {
                tTemp = pB ^ pC ^ pD;
            }

            return tTemp;
        }

        public static uint W(List<uint> pListW, int pT)
        {
            uint tW = pListW[pT - 16] ^ pListW[pT - 14] ^
pListW[pT - 8] ^ pListW[pT - 3];
            return tW;
        }

        public static uint K(int pT)
        {
            uint tK = 0x00000000;

            if ((pT >= 0) && (pT <= 19))
            {
                tK = 0x5A827999;
            }
            else if ((pT >= 20) && (pT <= 39))
            {
                tK = 0x6ED9EBA1;
            }
            else if ((pT >= 40) && (pT <= 59))
            {
                tK = 0x8F1BBCDC;
            }
            else if ((pT >= 60) && (pT <= 79))
            {
                tK = 0xCA62C1D6;
            }

            return tK;
        }

        public static string ConvertToHexString(uint
value)
        {
            StringBuilder tBuilder = new StringBuilder();
            tBuilder.Append(Convert.ToString(value,
16).PadLeft(8, '0'));

            return tBuilder.ToString();
        }
    }
}
```

Oleh karena belum ditemukan cara untuk mengambil seluruh isi dari folder, maka penulis menggunakan cara manual dengan meng-*hardcode* semua file yang terdapat dalam sebuah folder ujicoba untuk dipantau nilai hashnya selama masa percobaan.

VII. HASIL PENGUJIAN

Sebuah folder bernama “TEST” berisi tiga buah file yang akan diuji coba:

1. File text bernama Test1.txt
2. File text bernama Test2.txt, status file ini dalam keadaan *hidden*
3. File gambar staticmap.png

Ketiga file ini akan diujicobakan dalam kondisi berbeda nantinya. Nilai hash awal saat kondisi masih asli yaitu:

EFC4C*C299A*15A70E*24826A*10B9CB*A8E4F*360153*36947B*3D6FC7*188EAE*

Dan berikut adalah nilai hash saat pengujian dilakukan dalam berbagai kondisi dimana kejahatan file berpontensi terjadi seperti yang sudah dijabarkan sebelumnya.

- Ujicoba penambahan file
  Kondisi ini dicoba dengan menambahkan suatu file teks bernama Test3.txt. Nilai hash yang dihasilkan menunjukkan bahwa nilai ini berbeda dengan nilai asli.
  232827*463D7*2E6166*215279*1DBAA1*6849D*2F6ECC*12F4A2*16D478*3C314A*

- Ujicoba pengurangan file
  Kondisi ini dicoba dengan tidak menyertakan file staticmap.png ke dalam pengujian. Nilai hash yang didapat adalah:
  301D78*391C9E*391FA8*358D9D*2C2DE0*142087*1D42E4*19D130*2774C9*316FC2*

- Pengubahan nama file
  Pengubahan nama file dilakukan dengan hanya mengubah nama file secara sederhana. Hasil nilai hash yang didapat ternyata sama dengan nilai hash awal. Hal ini disebabkan nama file tidak menjadi sebuah variabel bagi nilai hash.

EFC4C*C299A*15A70E*24826A*10B9CB*A8E4F
*360153*36947B*3D6FC7*188EAE*

- Pengubahan isi file
  Pengubahan isi file dilakukan dengan mengubah beberapa karakter pada file teks Test1.txt. Nilai hash menunjukkan nilai yang sama:
  EFC4C*C299A*15A70E*24826A*10B9CB*A8E4F
  *360153*36947B*3D6FC7*188EAE*
  Hal ini cukup mengejutkan penulis karena isi dari file seharusnya merupakan nilai yang paling dipertimbangkan dari perhitungan nilai hash. Oleh karena itu, penulis mencoba untuk melakukan perhitungan ulang dengan memperbanyak perubahan pada isi file tersebut. kemudian nilai hash yang didapat adalah sebagai berikut:
  13360D*34D2B1*36E063*2A8BC9*1A6084*1198
  3F*181FEB*1C39D*24BA4F*29F558*
  Hal ini akan lebih banyak dibahas pada bagian kesimpulan.

- Pengubahan byte
  Pengubahan byte dilakukan pada file gambar staticmap.png dengan menggunakan aplikasi notepad++. Nilai hash menunjukkan hasil yang sama dengan nilai hash asli.
  EFC4C*C299A*15A70E*24826A*10B9CB*A8E4F
  *360153*36947B*3D6FC7*188EAE*
  Hal yang terjadi kurang lebih sama dengan pengubahan isi file. Kemudian penulis mencoba untuk merubah lebih banyak byte dari file tersebut. Nilai hash yang didapatkan adalah:
  1651CF*39B674*108EFF*9056C*202F3B*19483*
  DC8CC*1475A9*24596D*37A1B4*

VIII. KESIMPULAN

Dari hasil test program FAC dengan bahasa C# tersebut, dapat ditarik beberapa kesimpulan. Pada percobaan dengan beberapa kemungkinan terjadinya kejahatan file, program ini gagal pada percobaan mengganti nama file. Kemudian juga terdapat catatan penting mengenai sensitifitas nilai hash program ini. Pada pengubahan dalam tingkat yang masih sedikit, pengubahan byte dan pengubahan isi byte tidak dapat diketahui oleh nilai hash FAC. Namun setelah diubah kembali dengan perubahan yang lebih besar, hasilnya baru terlihat dengan perbedaan nilai hash.

Dapat ditarik beberapa kesimpulan dari hal ini. Walaupun ternyata program ini dapat mendeteksi perubahan dalam suatu folder, namun sensitifitas pendeteksian perubahannya masih haru dipertanyakan, karena ternyata masih belum dapat melihat perubahan satu karakter pada pengubahan isi file, dan juga tidak dapat melihat perubahan satu byte pada kasus pengubahan byte. Walaupun begitu, penulis rasa program FAC ini sudah dapat menjadi pondasi yang cukup apabila ternyata program yang lebih mangkus ingin dikembangkan, dengan catatan harus terdapat suatu pengembangan pada hal sensitifitas algoritma hash-nya.

REFERENCES

[1] http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html, 16 Maret 2010, 22.11 WIB.
[2] http://www.informit.com/articles/article.aspx?p=170967&seqNum=6, 16 Maret 2010, 22.11 WIB.
[3] http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html#Mac, 16 Maret 2010, 22.14 WIB.
[4] http://www.informit.com/articles/article.aspx?p=170967&seqNum=3, 16 Maret 2010, 22.14 WIB.
[5] http://id.wikipedia.org/wiki/Kriptografi, 17 Maret 2010, 00.37 WIB

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, 29 April 2010

ttd

Raditya Arief, 13507030