Studi Perbandingan Beberapa Fungsi Hash dalam Melakukan *Checksum* Berkas

Muhammad Dhito Prihardhanto - 13507118
Prodi Teknik Informatika
Sekolah Teknik Elektro dan Informatika (STEI)
Institut Teknologi Bandung Jalan Ganesha 10 Bandung 40132
email: muh_dhito@students.itb.ac.id

Abstrak **– *Fungsi hash saat ini banyak digunakan dalam melakukan cek integritas suatu data atau berkas. Istilah lainnya biasa disebut dengan checksum. Proses itu dilakukan dengan menghitung nilai hash dari satu berkas kemudian dibandingkan dengan nilai hash hasil dari berkas yang asli. Penerapan prinsip ini digunakan dalam aplikasi antivirus untuk memeriksa apakah suatu berkas terinfeksi oleh virus dan web browser yang memeriksa apakah suatu berkas yang diunduh telah lengkap dan tidak terjadi kerusakan. Dalam melakukan checksum itu dapat menggunakan algoritma-algoritma fungsi hash yang telah ada, di antaranya MD, SHA, atau CRC dengan berbagai variannya. Masing-masing algoritma itu memiliki kelebihan dan kelemahan masing-masing.***

Kata kunci: ***checksum, fungsi hash***

1. PENDAHULUAN

Operasi *checksum* perlu dilakukan untuk memeriksa integritas suatu berkas (*file*). Operasi *checksum* ini banyak dimanfaatkan dalam aplikasi komputer saat ini. Sebagai contoh, suatu *web browser* setiap melakukan *download* suatu file akan memeriksa pula kemungkinan terjadinya *corrupt* pada berkas tersebut dengan membandingkan hasil *checksum* terhadap berkas asli. Contoh lain adalah aplikasi antivirus. Aplikasi akan mendeteksi hasil *checksum* untuk dibandingkan dengan *database* virus yang dimilikinya.

Operasi *checksum* tersebut dapat dilakukan dengan berbagai algoritma. Algoritma yang paling sederhana adalah memeriksa apakah jumlah *byte* berkas yang diperbandingkan sama atau tidak. Namun, permasalahannya tidak sesederhana itu. Bisa jadi, ukuran sama tetapi ada *byte* yang dipertukarkan. Berangkat dari hal itu, maka perlu melibatkan fungsi *hash* untuk melakukan verikasi suatu berkas. Ada banyak algoritma fungsi *hash* yang ada saat ini yang dapat digunakan untuk melakukan *checksum* tersebut.

2. FUNGSI HASH

Fungsi *Hash* merupakan fungsi yang menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi suatu string keluaran yang panjangnya tetap (biasanya ukuran string keluaran jauh lebih kecil daripada ukuran semula). Persamaan fungsi *hash* adalah sebagai berikut:

Hash = func(Message)

Keluaran fungsi *hash* biasa disebut dengan nilai *hash* (*hash values*), kode *hash* (*hash codes*), *hash sums*, *checksums*, *hashes,* atau pesan-ringkas (*message digest*).

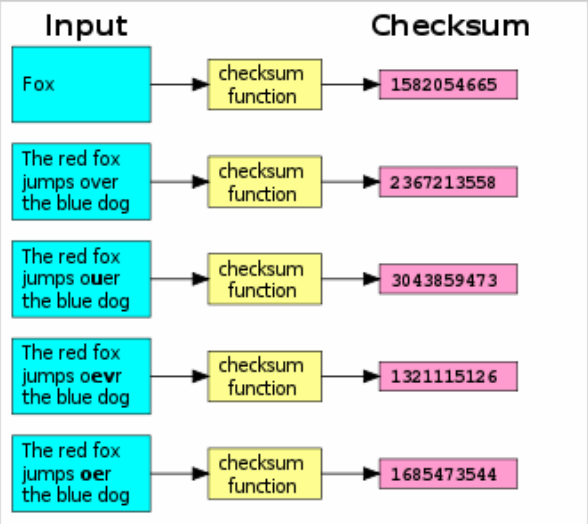
Saat ini suatu fungsi *hash* masih memungkinkan untuk memetakan dua pesan yang berbeda tetapi menghasilkan nilai *hash* yang sama. Hal itu disebut dengan kolisi (*collision*). Jadi, hingga saat ini perancangan fungsi *hash* yang baik masih menjadi topik riset yang terus dilakukan.

Fungsi *hash* sering dikaitkan (dan sering juga campur aduk) dengan istilah *checksum, check digits, fingerprints*, *randomizing functions*, *error correcting codes*, dan fungsi hash kriptografi. Meskipun semuanya berkaitan, tetapi pada dasarnya setiap istilah tersebut memiliki kegunaan dan kebutuhan masing-masing yang dirancang dan digunakan untuk hal yang berbeda.

2.1 CHECKSUM

Checksum atau *hash sum* adalah suatu data dengan ukuran tetap (*fixed-size datum*) yang dihitung dari suatu blok data digital dengan tujuan untuk mendeteksi kesalahan yang mungkin terjadi saat proses transmisi atau penyimpanan. Integritas data dapat diperiksa pada langkah selanjutnya dengan menghitung pula *checksum* dan membandingkannya dengan data yang satunya (data sumber/asli). Jika *checksum* tidak sama, maka hampir dipastikan bahwa data tersebut telah berubah, baik disengaja maupun tidak disengaja.

Prosedur untuk memperoleh *checksum* dari suatu data biasa disebut dengan *checksum function* atau *checksum algorithm*. Suatu algoritma *checksum* yang baik akan menghasilkan nilai yang berbeda ketika data mengalami kerusakan. Jika ternyata nilai *checksum* sama, maka kemungkinan besar data bebas dari kerusakan.
Kembali ke istilah-istilah yang berkaitan dengan fungsi *hash* seperti yang telah disebutkan sebelumnya, *Checksum* pun juga berkaitan dengan istilah-istilah seperti *check digits* dan *parity bits* yang merupakan kasus khusus untuk memeriksa *checksum* pada blok data berukuran kecil (seperti nomor akun bank) dan *error-correcting codes* yang merupakan *checksum* khusus yang tidak hanya memeriksa kesalahan pada suatu data, tetapi juga mampu memulihkannya.



Gambar 1. Cheksum beberapa teks berbeda

2.2 AGORITMA FUNGSI *HASH*

Dalam makalah ini akan dilakukan pengujian *checksum* oleh beberapa algoritmafungsi *hash.* Algoritma itu saat ini memang banyak dgunakan sebagai algoritma untuk melakukan *checksum* pda beberapa aplikasi. Beberapa algoritma fungsi *hash* itu adalah:

2.2.1 AGORITMA MD5

Algoritma MD5 menerima masukan berupa pesan dengan ukuran sembarang dan menghasilkan *message digest* yang panjangnya 128 bit.

MD5 banyak digunakan dalam dunia perangkat lunak untuk membantu memberikan jaminan bahwa berkas yang ditransfer memang dalam keadaan utuh. Misalnya, *server* sering memberikan pra-perhitungan *checksum MD5* untuk berkas-berkas, sehingga pengguna dapat membandingkan *checksum* dari berkas yang diunduh. Sistem operasi UNIX sudah menyediakan MD5 dalam bentuk paket distribusi mereka, sedangkan pada sistem operasi Windows MD5 merupakan fitur yang terpisah (*trird-party application*).

Namun, dewasa ini adalah mudah bagi orang untuk menciptakan berkas salinan dengan *checksum* yang sama (kolisi). Jadi teknik ini menjadi kurang dapat melindungi terhadap beberapa bentuk gangguan berbahaya. Seringkali terdapat beberapa kasus *checksum* yang tidak dapat dipercaya dengan menggunakan MD5 ini, di mana MD5 hanya dapat menyediakan fungsionalitas pengecekan error, seperti menyatakan bahwa file yang diunduh mengalami kerusakan atau tidak lengkap.

2.2.2 AGORITMA SHA

SHA (*Secure Hash Algorithm*) dapat dikatakan sebagai kelanjutan dari pendahulunya, MD5, yang telah digunakan secara luas. SHA disebut aman karena ia dirancang sedemikian rupa sehingga secara komputasi tidak mungkin menemukan pesan yang berkoresponden dengan *message digest* yang diberikan. Fungsi *hash* yang paling umum digunakan adalah SHA-1 yang telah diimplementasikan di dalam berbagai aplikasi dan protokol keamanan seperti SSL, SSH, TLS, S/MIME, dan *Ipsec*. Ada banyak varian dari keluarga SHA ini.

Varian pertama, adalah SHA-1 yang menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari *message digest* yang dihasilkan oleh MD5 yang hanya 128 bit. SHA-1 ini digunakan antara lain dalam sistem control revisi terdistribusi seperti Git, Mercurial, dan untuk mendeteksi kerusakan data.

2.2.3 AGORITMA CRC

Cyclic Redundancy Check (CRC) atau *polynomial code checksum* adalah merupakan suatu fungsi *hash* yang dirancang untuk memeriksa perubahan yang terjadi pada suatu data komputer dengan cara menghasilkan suatu checksum dari fungsi hash-nya. Salah satu varian algoritma CRC adalah CRC32 di mana 32 melambangkan panjang checksum dalam bit. Bentuk CRC yang disediakan untuk algoritma sesuai dengan ide pembagian ”polinomial”. Dan hal ini digunakan untuk memperhitungkan checksum yang sama dari seluruh algoritma CRC.

Algoritma CRC cukup banyak digunakan sebagai algoritma untuk pemeriksaan byte dari sautu berkas. Algoritma ini mencari lewat seluruh jumlah byte dan

menghasilkan angka 32 bit untuk menggambarkan isi file. Sangat kecil sekali kemungkinan dua *stream* dari *byte* yang berbeda mempunyai CRC yang sama.

Algoritma CRC32 juga dapat diandalkan dalam memeriksa kesalahan yang mungkin terjadi pada urutan byte. Dengan CRC32 kemungkinan perubahan standar (penyimpangan dari penghitungan CRC terhadap file) yang terjadi dapat dikendalikan. Perkembangan teknologi dan informasi membawa perubahan besar dalam penggunaan metode Checksum CRC32. Banyak bermunculan *malware* dan juga perkembangan virus komputer yang semakin canggih membuat metode *checksum* CRC32 lantas digunakan untuk mengetahui mendeteksi virus dengan acuan nilai CRC32-nya. Nilai CRC32 adalah nilai yang didapat dari besar file dan nama file yang dibandingkan dengan tabel CRC32 yang sudah ada acuannya.

Algoritma CRC32 ini adalah algoritma yang paling banyak digunakan oleh aplikasi-aplikasi antivirus. Cara – cara antivirus dalam mengenali sebuah virus melalui metode *checksum* CRC32 adalah sebagai berikut :

- Memilih berkas yang akan diperiksa
- Mengambil informasi dari berkas tersebut, yaitu nama dan ukuran
- Menghitung *checksum* berkas yang diambil dari ukuran berkas dengan metode CRC32.
- Hasil *checksum* tersebut akan dikumpulkan dalam *database signature checksum* CRC32 dari virus-virus yang telah dicari nilai checksum CRC32-nya.
- Antivirus menggunakan hasil *checksum* tersebut untuk mengenali bahwa program tersebut adalah virus.

3. IMPLEMENTASI

Dalam implementasi pengujian yang akan dilakukan, dibuatlah sebuah aplikasi untuk melakukan *checksum* dengan beberapa algoritma yang telah ditentukan. Dari *checksum* itu akan ditentukan integritas sautu berkas.

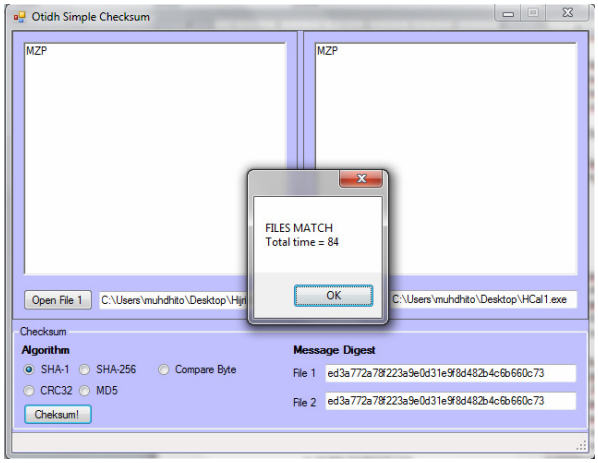
Aplikasi ini dibuat dengan me-*load* dua buah berkas, yaitu berkas yang diuji (file 1) dan berkas asli (file 2). Beberapa algoritma yang dapat dipilih dalam melakukan pengujian adalah SHA-1, SHA-256, MD5, CRC32, dan perbandingan dari pembacaan *byte*.

Implementasi algoritma SHA-1, SHA-256, dan MD5 menggunakan *library* yang telah disediakan di dalam .NET. Sementara itu untuk implementasi CRC32 menggunakan *source code* yang dibagi dalam situs www.geekpedia.com. Sedangkan untuk algoritma dengan perbandingan dari pembacaan *byte* menggunakan implementasi dengan langkah-langkah yang sama seperti algortima yang lain, yaitu pembacaan semua *byte* pada berkas terlebih dahulu, lalu sebagaimana hasil *checksum* yang lain yaitu berupa string, maka *byte* yang dibaca tersebut dikonversi dahulu menjadi string baru setelah itu dilakukan perbandingan string menggunakan fungsi yang sudah terdapat pada .NET.

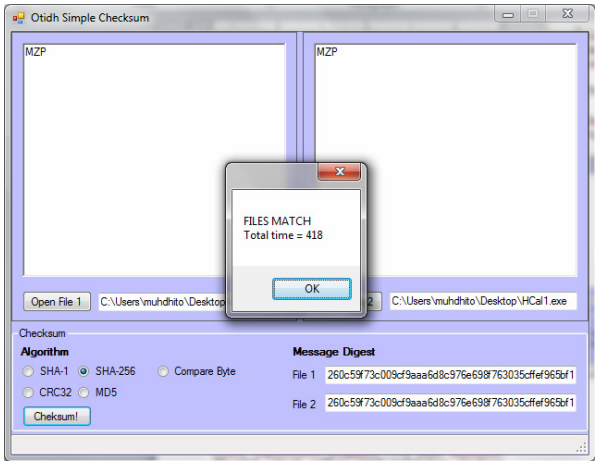
Dalam implementasi bertujuan untuk mengecek performa masing-masing algoritma (terutama dalam hal waktu) dengan menggunakan beberapa skenario. Skenario itu di antaranya:

3.1 SKENARIO 1: DUA BERKAS YANG SAMA

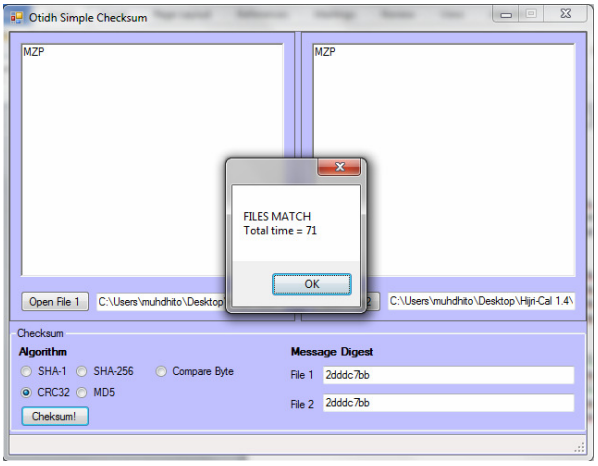
Dengan kedua berkas yang sama, maka seharusnya hasil *checksum* juga sama.



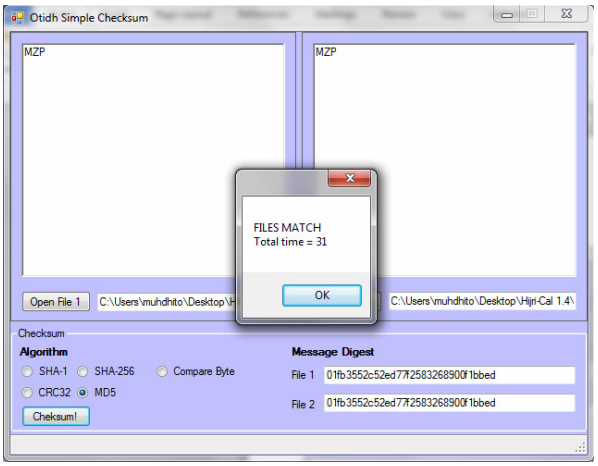
Gambar 2. Checksum dengan SHA1



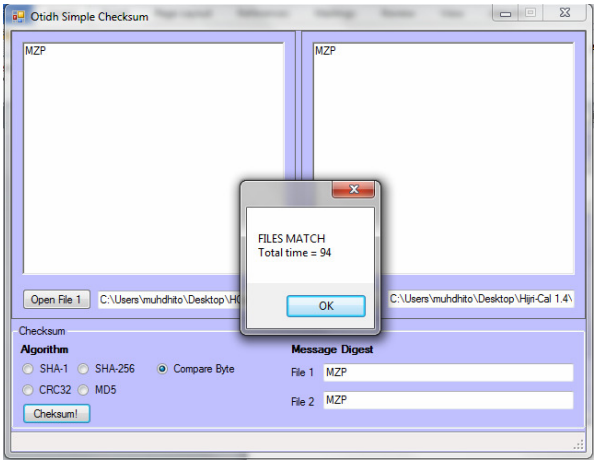
Gambar 3. Checksum dengan SHA256



Gambar 4. Checksum dengan CRC32



Gambar 5. Checksum dengan MD5

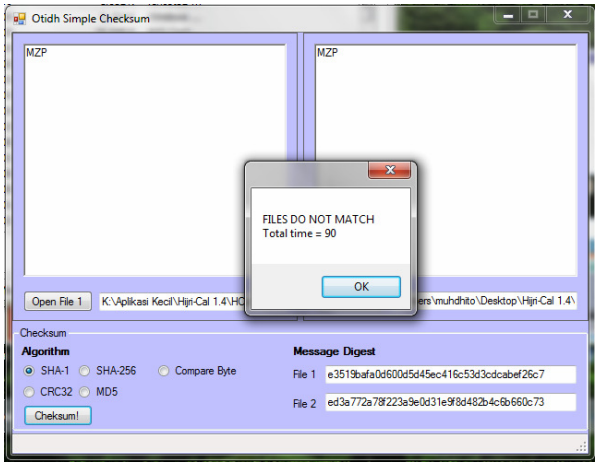


Gambar 6. Checksum dengan Compare Byte

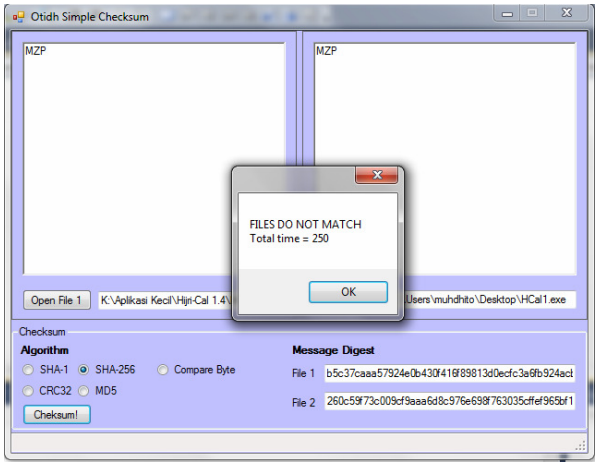
Dapat dilihat, dalam skenario ini, proses *checksum* tercepat adalah dengan menggunakan MD5, yaitu membutuhkan waktu 31 *milliseconds*. Sementara itu proses *checksum* terlama adalah dengan menggunakan algoritma SHA256, yaitu menghabiskan waktu 418 *milliseconds*.

3.2 SKENARIO 2: DUA BERKAS YANG SALAH SATUNYA TERINFEKSI VIRUS

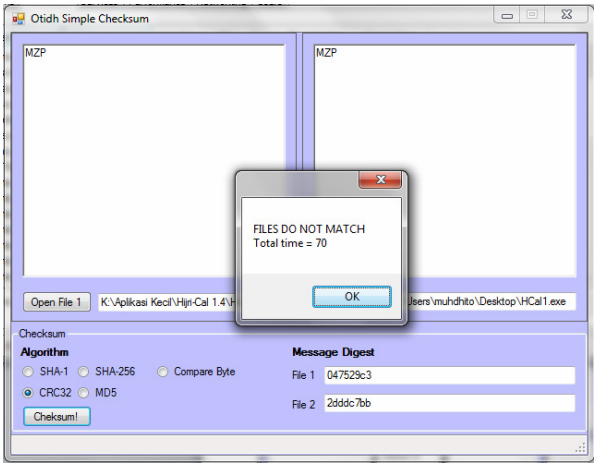
Dengan kedua berkas yang tidak sama, maka seharusnya hasil *checksum* juga tidak sama. Berkas satu adalah berkas yang telah terinfeksi dengan virus (diketahui setelah diperiksa dengan AVG Antivirus) sehingga memiliki ukuran lebih besar daripada berkas kedua yang asli.



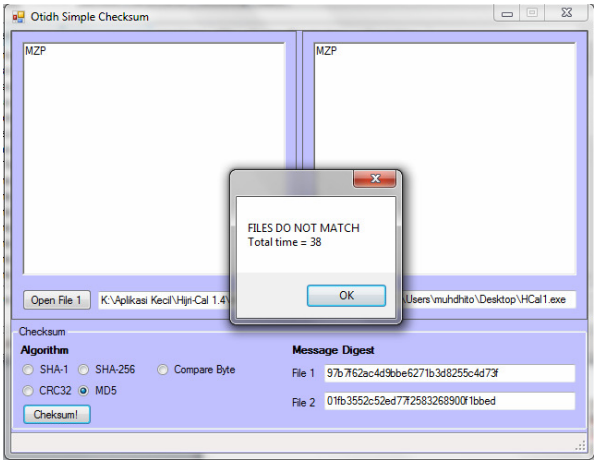
Gambar 7. Checksum dengan SHA1



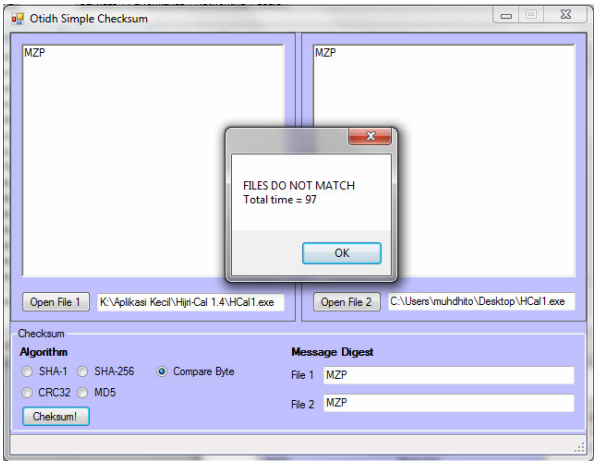
Gambar 8. Checksum dengan SHA256



Gambar 9. Checksum dengan CRC32



Gambar 10. Checksum dengan MD5



Gambar 11. Checksum dengan Compare Byte

Dapat dilihat, dalam skenario ini, proses *checksum* tercepat adalah masih sama yaitu dengan menggunakan MD5, yaitu membutuhkan waktu 38 *milliseconds*. Sementara itu proses *checksum* terlama adalah dengan menggunakan algoritma SHA256, yaitu menghabiskan waktu 250 *milliseconds*.

5. KESIMPULAN

Dari kedua scenario di atas, tampak MD5 menjadi algoritma tercepat dalam melakukan *checksum* dan algoritma SHA256 menjadi algoritma dengan proses terlama.

Namun, ada hal yang menarik, yaitu pada scenario pertama di mana kedua berkas identic, algoritma MD5 ternyata memakan waktu yang lebih lama daripada scenario kedua saat kedua berkas berbeda.

Sementara itu algoritma SHA256 meskipun menjadi algoritma dengan waktu terlama, ternyata dia bisa menjadi lebih cepat saat menghitung *checksum* untuk kedua berkas yang berbeda.

Secara umum tiga algoritma, yaitu SHA1, MD5, dan *Compare Byte* memproses lebih lambat saat kedua berkas berbeda, sementara itu algoritma CRC32 dan SHA256 malah membutuhkan waktu yang lebih rendah saat menghitung *checksum* untuk kedua berkas berbeda.

MD5 meskipun menjadi yang tercepat, tetapi berdasarkan literature yang ada sekarang jarang digunakan untuk proses *hashing* karena dinilai sudah tidak aman lagi dan terdapat kolisi. Sementara itu, algoritma CRC32 lebih banayak digunakan oleh aplikasi antivirus saat ini untuk men-*scan* berkas yang diduga terinfeksi *virus*.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2004. Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] http://antivirus.about.com/od/whatisavirus/a/virussignature.htm, diakses pada tanggal 28 April 2010

[3https://www.vicheck.ca/md5query.php?hash=c603dffd233f4c00e0ec6ffe85e52110, diakses pada tanggal 28 April 2010

[4] http://www.geekpedia.com/code113_Checksum-CRC32-Calculator.html, diakses pada tanggal 16 Mei 2010

[5] http://en.wikipedia.org/wiki/checksum, diakses pada tanggal 16 Mei 2010

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, 17 Mei 2010

Muhammad Dhito P.

LAMPIRAN SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Globalization;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        private string filename1 = "";
        private string filename2 = "";
        private StringBuilder filenameContent1;
        private StringBuilder filenameContent2;
        private Timer timer;
        private DateTime start;
        private DateTime end;
        private TimeSpan diff;
        private string waktu = "";

        private static readonly UInt32[] CRCTable
=
{
    0x00000000, 0x77073096, 0xee0e612c,
0x990951ba, 0x076dc419,
    0x706af48f, 0xe963a535, 0x9e6495a3,
0x0edb8832, 0x79dcb8a4,
    0xe0d5e91e, 0x97d2d988, 0x09b64c2b,
0x7eb17cbd, 0xe7b82d07,
    0x90bf1d91, 0x1db71064, 0x6ab020f2,
0xf3b97148, 0x84be41de,
    0x1adad47d, 0x6ddde4eb, 0xf4d4b551,
0x83d385c7, 0x136c9856,
    0x646ba8c0, 0xfd62f97a, 0x8a65c9ec,
0x14015c4f, 0x63066cd9,
    0xfa0f3d63, 0x8d080df5, 0x3b6e20c8,
0x4c69105e, 0xd56041e4,
    0xa2677172, 0x3c03e4d1, 0x4b04d447,
0xd20d85fd, 0xa50ab56b,
    0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6,
0xacbcf940, 0x32d86ce3,
    0x45df5c75, 0xdcd60dcf, 0xabd13d59,
0x26d930ac, 0x51de003a,
    0xc8d75180, 0xbfd06116, 0x21b4f4b5,
0x56b3c423, 0xcfba9599,
    0xb8bda50f, 0x2802b89e, 0x5f058808,
0xc60cd9b2, 0xb10be924,
    0x2f6f7c87, 0x58684c11, 0xc1611dab,
0xb6662d3d, 0x76dc4190,
    0x01db7106, 0x98d220bc, 0xefd5102a,
0x71b18589, 0x06b6b51f,
    0x9fbfe4a5, 0xe8b8d433, 0x7807c9a2,
0x0f00f934, 0x9609a88e,
    0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
0x91646c97, 0xe6635c01,
    0x6b6b51f4, 0x1c6c6162, 0x856530d8,
0xf262004e, 0x6c0695ed,
    0x1b01a57b, 0x8208f4c1, 0xf50fc457,
0x65b0d9c6, 0x12b7e950,
    0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf,
0x15da2d49, 0x8cd37cf3,
    0xfbd44c65, 0x4db26158, 0x3ab551ce,
0xa3bc0074, 0xd4bb30e2,
    0x4adfa541, 0x3dd895d7, 0xa4d1c46d,
0xd3d6f4fb, 0x4369e96a,
    0x346ed9fc, 0xad678846, 0xda60b8d0,
0x44042d73, 0x33031de5,
    0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c,
0x270241aa, 0xbe0b1010,
    0xc90c2086, 0x5768b525, 0x206f85b3,
0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822,
0xc7d7a8b4, 0x59b33d17,
    0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad,
0xedb88320, 0x9abfb3b6,
    0x03b6e20c, 0x74b1d29a, 0xead54739,
0x9dd277af, 0x04db2615,
    0x73dc1683, 0xe3630b12, 0x94643b84,
0x0d6d6a3e, 0x7a6a5aa8,
    0xe40ecf0b, 0x9309ff9d, 0x0a00ae27,
0x7d079eb1, 0xf00f9344,
    0x8708a3d2, 0x1e01f268, 0x6906c2fe,
0xf762575d, 0x806567cb,
    0x196c3671, 0x6e6b06e7, 0xfed41b76,
0x89d32be0, 0x10da7a5a,
    0x67dd4acc, 0xf9b9df6f, 0x8ebeeff9,
0x17b7be43, 0x60b08ed5,
    0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4,
0x4fdff252, 0xd1bb67f1,
    0xa6bc5767, 0x3fb506dd, 0x48b2364b,
0xd80d2bda, 0xaf0a1b4c,
    0x36034af6, 0x41047a60, 0xdf60efc3,
0xa867df55, 0x316e8eef,
    0x4669be79, 0xcb61b38c, 0xbc66831a,
0x256fd2a0, 0x5268e236,
    0xcc0c7795, 0xbb0b4703, 0x220216b9,
0x5505262f, 0xc5ba3bbe,
    0xb2bd0b28, 0x2bb45a92, 0x5cb36a04,
0xc2d7ffa7, 0xb5d0cf31,
    0x2cd99e8b, 0x5bdeae1d, 0x9b64c2b0,
0xec63f226, 0x756aa39c,
    0x026d930a, 0x9c0906a9, 0xeb0e363f,
0x72076785, 0x05005713,
    0x95bf4a82, 0xe2b87a14, 0x7bb12bae,
0x0cb61b38, 0x92d28e9b,
    0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21,
0x86d3d2d4, 0xf1d4e242,
    0x68ddb3f8, 0x1fda836e, 0x81be16cd,
0xf6b9265b, 0x6fb077e1,
    0x18b74777, 0x88085ae6, 0xff0f6a70,
0x66063bca, 0x11010b5c,
    0x8f659eff, 0xf862ae69, 0x616bffd3,
0x166ccf45, 0xa00ae278,
    0xd70dd2ee, 0x4e048354, 0x3903b3c2,
0xa7672661, 0xd06016f7,
    0x4969474d, 0x3e6e77db, 0xaed16a4a,
0xd9d65adc, 0x40df0b66,
    0x37d83bf0, 0xa9bcae53, 0xdebb9ec5,
0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330,
0x24b4a3a6, 0xbad03605,
    0xcdd70693, 0x54de5729, 0x23d967bf,
0xb3667a2e, 0xc4614ab8,
    0x5d681b02, 0x2a6f2b94, 0xb40bbe37,
0xc30c8ea1, 0x5a05df1b,
    0x2d02ef8d
};

        public Form1()
        {
            InitializeComponent();
```

```
            filenameContent1 = new
StringBuilder();
            filenameContent2 = new
StringBuilder();
            timer = new Timer();
            start = new DateTime();
            end = new DateTime();
        }

        private void timer_Tick(object sender,
EventArgs e)
        {

            /*
            lblTgl.Text =
DateTime.Now.ToString("dddd, dd MMMM yyyy",
KulturIDN);
            lblJam.Text =
DateTime.Now.ToString("HH:mm:ss", KulturIDN);*/
        }


        private void timer_Stop(object sender,
EventArgs e)
        {
            CultureInfo KulturIDN = new
CultureInfo("id-ID");
            waktu = timer.Interval.ToString();
            timer.Enabled = false;
        }

        private void button1_Click(object sender,
EventArgs e)
        {
            if (radioButton1.Checked)
                doSHA1();
            else if (radioButton2.Checked)
                doSHA256();
            else if (radioButton3.Checked)
                doMD5();
            else if (radioButton4.Checked)
                doCRC32();
            else if (radioButton5.Checked)
                doCompareByte();


        }

        private void doSHA1()
        {
            FileStream file1 = new
FileStream(filename1, FileMode.Open);
            FileStream file2 = new
FileStream(filename2, FileMode.Open);
            StringBuilder sb1 = new
StringBuilder();
            StringBuilder sb2 = new
StringBuilder();
            SHA1Managed shaM = new SHA1Managed();
            Boolean match = true;
            ;
            start = DateTime.Now;
            byte[] retVal1 =
shaM.ComputeHash(file1);
            byte[] retVal2 =
shaM.ComputeHash(file2);


            //StringBuilder sb1 = new
StringBuilder();
            for (int i = 0; i < retVal1.Length;
i++)
            {

sb1.Append(retVal1[i].ToString("x2"));
            }
            //StringBuilder sb2 = new
StringBuilder();
            for (int i = 0; i < retVal2.Length;
i++)
            {

sb2.Append(retVal2[i].ToString("x2"));
            }
            if
(!sb1.ToString().Equals(sb2.ToString()))
                match = false;
            ;
            end = DateTime.Now;
            diff = end.Subtract(start);
            waktu = diff.Milliseconds.ToString();
            file1.Close();
            file2.Close();
            hashBox1.Text = sb1.ToString();
            hashBox2.Text = sb2.ToString();
            if (match)
                MessageBox.Show("FILES
MATCH\nTotal time = " + waktu);
            else
                MessageBox.Show("FILES DO NOT
MATCH\nTotal time = " + waktu);

        }

        private void doSHA256()
        {
            FileStream file1 = new
FileStream(filename1, FileMode.Open);
            FileStream file2 = new
FileStream(filename2, FileMode.Open);
            StringBuilder sb1 = new
StringBuilder();
            StringBuilder sb2 = new
StringBuilder();
            SHA256Managed shaM = new
SHA256Managed();
            Boolean match = true;
            ;
            start = DateTime.Now;
            byte[] retVal1 =
shaM.ComputeHash(file1);
            byte[] retVal2 =
shaM.ComputeHash(file2);


            //StringBuilder sb1 = new
StringBuilder();
            for (int i = 0; i < retVal1.Length;
i++)
            {

sb1.Append(retVal1[i].ToString("x2"));
            }
            //StringBuilder sb2 = new
StringBuilder();
            for (int i = 0; i < retVal2.Length;
i++)
            {

sb2.Append(retVal2[i].ToString("x2"));
```

```
            }
            if
(!sb1.ToString().Equals(sb2.ToString()))
                match = false;
            ;
            end = DateTime.Now;
            diff = end.Subtract(start);
            waktu = diff.Milliseconds.ToString();
            file1.Close();
            file2.Close();
            hashBox1.Text = sb1.ToString();
            hashBox2.Text = sb2.ToString();
            if (match)
                MessageBox.Show("FILES
MATCH\nTotal time = " + waktu);
            else
                MessageBox.Show("FILES DO NOT
MATCH\nTotal time = " + waktu);
        }

        private void doMD5()
        {
            FileStream file1 = new
FileStream(filename1, FileMode.Open);
            FileStream file2 = new
FileStream(filename2, FileMode.Open);
            StringBuilder sb1 = new
StringBuilder();
            StringBuilder sb2 = new
StringBuilder();
            Boolean match = true;
            MD5 md5 = new
MD5CryptoServiceProvider();

            start = DateTime.Now;
            byte[] retVal1 =
md5.ComputeHash(file1);
            byte[] retVal2 =
md5.ComputeHash(file2);


            //StringBuilder sb1 = new
StringBuilder();
            for (int i = 0; i < retVal1.Length;
i++)
            {

sb1.Append(retVal1[i].ToString("x2"));
            }
            //StringBuilder sb2 = new
StringBuilder();
            for (int i = 0; i < retVal2.Length;
i++)
            {

sb2.Append(retVal2[i].ToString("x2"));
            }
            if
(!sb1.ToString().Equals(sb2.ToString()))
                match = false;
            //return sb.ToString();

            end = DateTime.Now;
            diff = end.Subtract(start);
            waktu = diff.Milliseconds.ToString();
            file1.Close();
            file2.Close();
            hashBox1.Text = sb1.ToString();
            hashBox2.Text = sb2.ToString();
            if (match)
                MessageBox.Show("FILES
MATCH\nTotal time = " + waktu);
            else
                MessageBox.Show("FILES DO NOT
MATCH\nTotal time = " + waktu);
        }

        private void doCRC32()
        {
            FileStream file1 = new
FileStream(filename1, FileMode.Open);
            FileStream file2 = new
FileStream(filename2, FileMode.Open);
            System.Text.ASCIIEncoding
AsciiEncoding = new System.Text.ASCIIEncoding();
            StringBuilder sb1 = new
StringBuilder();
            StringBuilder sb2 = new
StringBuilder();
            Boolean match = true;
            byte[] Buffer1 = null;
            byte[] Buffer2 = null;

            start = DateTime.Now;
            Buffer1 = new
byte[(int)file1.Length];
            file1.Read(Buffer1, 0,
(int)file1.Length);
            Buffer2 = new
byte[(int)file2.Length];
            file2.Read(Buffer2, 0,
(int)file2.Length);
            byte[] retVal1 = Calculate(Buffer1);
            byte[] retVal2 = Calculate(Buffer2);


            for (int i = 0; i < retVal1.Length;
i++)
            {

sb1.Append(retVal1[i].ToString("x2"));
            }

            for (int i = 0; i < retVal2.Length;
i++)
            {

sb2.Append(retVal2[i].ToString("x2"));
            }
            if
(!sb1.ToString().Equals(sb2.ToString()))
                match = false;

            end = DateTime.Now;
            diff = end.Subtract(start);
            waktu = diff.Milliseconds.ToString();
            file1.Close();
            file2.Close();

            hashBox1.Text = sb1.ToString();
            hashBox2.Text = sb2.ToString();
            if (match)
                MessageBox.Show("FILES
MATCH\nTotal time = " + waktu);
            else
                MessageBox.Show("FILES DO NOT
MATCH\nTotal time = " + waktu);
        }

        private void doCompareByte()
```

```
        {
            FileStream file1 = new
FileStream(filename1, FileMode.Open);
            FileStream file2 = new
FileStream(filename2, FileMode.Open);
            ASCIIEncoding AsciiEncoding = new
System.Text.ASCIIEncoding();
            StringBuilder sb1 = new
StringBuilder();
            StringBuilder sb2 = new
StringBuilder();
            Boolean match = true;
            byte[] Buffer1 = null;
            byte[] Buffer2 = null;
            ;
            start = DateTime.Now;
            Buffer1 = new
byte[(int)file1.Length];
            file1.Read(Buffer1, 0,
(int)file1.Length);
            Buffer2 = new
byte[(int)file2.Length];
            file2.Read(Buffer2, 0,
(int)file2.Length);


sb1.Append(AsciiEncoding.GetString(Buffer1));

sb2.Append(AsciiEncoding.GetString(Buffer2));
            if
(!sb1.ToString().Equals(sb2.ToString()))
                match = false;
            ;
            end = DateTime.Now;
            diff = end.Subtract(start);
            waktu = diff.Milliseconds.ToString();
            file1.Close();
            file2.Close();
            hashBox1.Text = sb1.ToString();
            hashBox2.Text = sb2.ToString();
            if (match)
                MessageBox.Show("FILES
MATCH\nTotal time = " + waktu);
            else
                MessageBox.Show("FILES DO NOT
MATCH\nTotal time = " + waktu);
        }

        private void openButton1_Click(object
sender, EventArgs e)
        {
            OpenFileDialog Open = new
OpenFileDialog();
            Open.Title = "Open File";
            Open.AddExtension = true;
            if (Open.ShowDialog() ==
DialogResult.OK)
            {
                filename1 = Open.FileName;
                filenameBox1.Text = filename1;

filenameContent1.Append(ReadFile.ByteArrayToStrin
g(ReadFile.FileToByteArray(filename1)));
                fileArea1.Text =
filenameContent1.ToString();
            }
        }

        private void openButton2_Click(object
sender, EventArgs e)
```

```
        {
            OpenFileDialog Open = new
OpenFileDialog();
            Open.Title = "Open File";
            Open.AddExtension = true;
            if (Open.ShowDialog() ==
DialogResult.OK)
            {
                filename2 = Open.FileName;
                filenameBox2.Text = filename2;

filenameContent2.Append(ReadFile.ByteArrayToStrin
g(ReadFile.FileToByteArray(filename2)));
                fileArea2.Text =
filenameContent2.ToString();
            }
        }


        public static byte[] Calculate(byte[]
Value)
        {
            UInt32 CRCVal = 0xffffffff;
            for (int i = 0; i < Value.Length;
i++)
            {
                CRCVal = (CRCVal >> 8) ^
CRCTable[(CRCVal & 0xff) ^ Value[i]];
            }
            CRCVal ^= 0xffffffff; // Toggle
operation
            byte[] Result = new byte[4];

            Result[0] = (byte)(CRCVal >> 24);
            Result[1] = (byte)(CRCVal >> 16);
            Result[2] = (byte)(CRCVal >> 8);
            Result[3] = (byte)(CRCVal);

            return Result;
        }
    }}
```