Penerapan MD5 untuk Pencarian File Duplikasi

Sandy Gunawan Tanuwijaya/13510025
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510025@itb.ac.id

***Abstrak*— MD5 merupakan sebuah jenis algoritma message digest yang dikembangkan oleh Ron Rivest pada tahun 1991. Algoritma MD5 dikembangkan sebagai perbaikan dari algoritma yang dikembangkan Ron Rivest sebelumnya yaitu MD4. Algoritma ini mengambil sebuah pesan input dengan panjang tertentu lalu membuat suatu hash/message digest sebesar 128-bit. Secara garis besar, pembentukan message digest tersebut meliputi penambahan bit-bit pengganjal/padding bits, penambahan nilai panjang pesan semula, inisialisasi penyangga/buffer MD, dan pengolahan pesan dalam blok ukuran 512 bit. Sekarang algoritma MD5 tidak dipakai lagi di luar aplikasi pemeriksaan integritas file karena sudah dianggap tidak aman lagi dan kolisinya sudah ditemukan.**

Algoritma MD5 digunakan untuk memeriksa integritas suatu data/file, seperti apakah suatu file yang diunduh dari server berada dalam keadaan utuh, atau apakah file yang disimpan di dalam file terkompresi tidak korup. MD5 hash juga dapat digunakan untuk menemukan adanya file duplikasi di dalam hard disk, dengan cara membandingkan nilai MD5 dari beberapa file itu sendiri. Jika file-file tersebut ternyata memiliki nilai MD5 yang sama, berarti file merupakan file terduplikasi. Aplikasi pencari file duplikasi ini dikembangkan dalam bahasa C#.

Kata kunci: MD5, hash, message digest, file duplikasi

I. PENDAHULUAN

Algoritma MD5 merupakan salah satu algoritma kriptografi fungsi *hash* yang banyak digunakan untuk berbagai macam aplikasi kriptografi, salah satunya adalah memeriksa integritas data dengan cara membentuk sebuah *hash/message digest* sebesar 128 bit dari data/pesan yang diinput. Algoritma MD5 juga digunakan pada aplikasi tanda tangan digital di mana sebuah file besar harus dikompres terlebih dahulu sebelum dienkripsi menggunakan kunci privat dari algoritma kriptografi kunci publik seperti RSA.

File duplikasi dapat memakan ruang hard-disk secara cepat dan tidak disadari. Oleh karena itu, dikembangkanlah aplikasi-aplikasi yang digunakan untuk mencari dan menghapus file duplikasi. Aplikasi tersebut biasanya bekerja dengan cara membandingkan atribut-atribut dari file yang dibaca, seperti nama, format file, ukuran, tanggal dibuat/modifikasi, maupun nilai *hash* MD5 atau CRC32.

II. DASAR TEORI

A. *Fungsi Hash MD5*

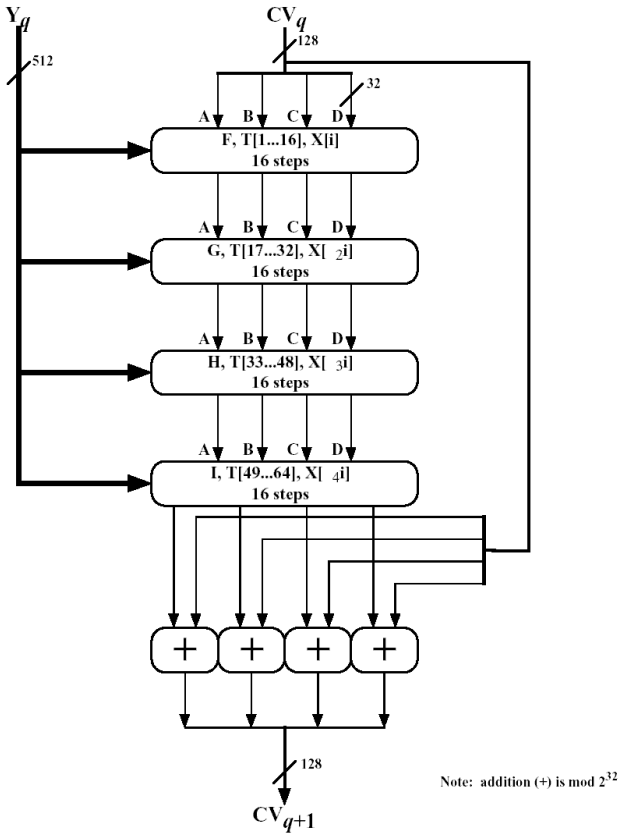


Figure 9.2 MD5 Processing of a Single 512-bit Block (MD5 Compression Function)

Gambar 2.1 Diagram cara kerja MD5 pada sebuah blok pesan berukuran 512 bit

MD5 dibuat oleh Ron Rivest dari MIT sebagai pengganti dari algoritma buatannya yang sebelumnya yaitu MD4.

Langkah-langkah untuk membuat sebuah *message digest*:

1. Penambahan bit-bit pengganjal (*padding bits*).
2. Penambahan nilai panjang pesan semula.
3. Inisialisasi penyangga (*buffer*) *MD.*
4. Pengolahan pesan dalam blok berukuran 512 bit.

Secara teori, berdasarkan langkah-langkah tersebut, tidak mungkin untuk mendapat dua pesan dengan *message digest* yang sama, karena perubahan satu karakter pada pesan saja dapat mengubah nilai *hash* secara total. Contoh:

MD5("*The quick brown fox jumps over the lazy dog*") =
9e107d9d372bb6826bd81d3542a419d6

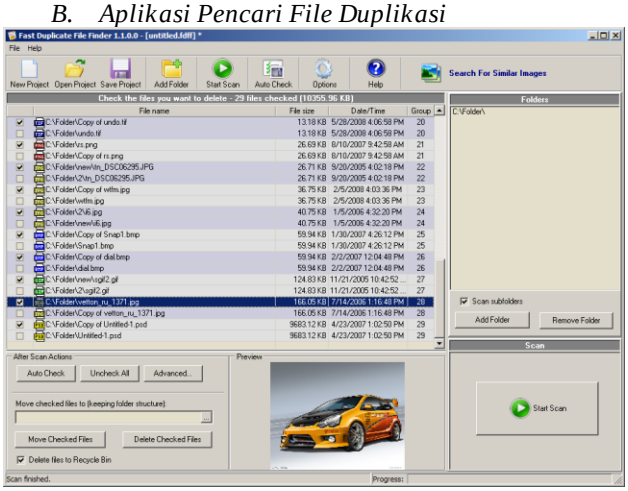
MD5("*The quick brown fox jumps over the lazy dog.*") =
e4d909c290d0fb1ca068ffaddf22cbd0

Pada tahun 1996, Hans Dobbertin dari RSA berhasil menemukan kolisi untuk algoritma MD5, namun masih belum terxmasuk serangan terhadap algoritma tersebut. Baru pada tahun 2005, algoritma MD5 ditemukan kolisinya oleh Arjen Lenstra, Xiaoyun Wang, and Benne de Weger dengan cara membuat dua sertifikat X.509 dengan kunci publik yang berbeda tetapi memiliki nilai *hash* MD5 ynag sama. Sehingga, sejak itu algoritma MD5 sudah dianggap tidak aman lagi untuk aplikasi kriptografi seperti sertifikat SSL atau tanda tangan digital, walaupun masih aman digunakan untuk hal-hal diluar keamanan digital seperti pemeriksaan integritas file atau pencarian file duplikasi.

Salah satu contoh penerapan MD5 untuk pemeriksaan integritas file antara lain pengecekan integritas file yang telah diunduh. Jika nilai *hash* MD5 file yang diunduh berbeda dari seharusnya, maka file tersebut korup/rusak. Penerapan yang serupa juga dapat dilakukan untuk memeriksa integritas file yang diunduh dari situs *mirror*, jika nilai *hash* MD5 berbeda maka terdapat kesalahan pada file yang diunduh dari situs *mirror* tersebut (rusak, terinfeksi virus, dan sejenis).

Untuk pencarian file duplikasi, sebenarnya dapat dilakukan dengan membandingkan nilai *hash* dari dua file atau lebih. Jika ternyata ditemukan file-file dengan nilai *hash* yang sama (walaupun nama, tanggal pembuatan/modifikasi maupun *format* dari file berbeda satu sama lain), maka dapat disimpulkan bahwa file tersebut merupakan file duplikasi.

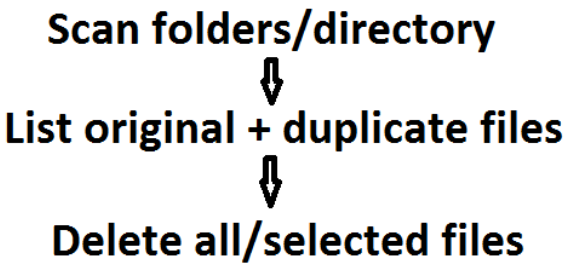
B. Aplikasi Pencari File Duplikasi



Gambar 2.2 Contoh tampilan sebuah aplikasi pencari file duplikasi

Cepat atau lambat, pengguna komputer pasti akan secara tidak sengaja dan tidak disadari menghasilkan file-file *junk* atau duplikasi yang dapat secara cepat memenuhi ruang *hard-disk*. Oleh karena itu, para pengembang aplikasi membuat berbagai macam aplikasi pencari file duplikasi, baik berbayar maupun gratis dan/atau *open source*. Aplikasi pencari file duplikasi umumnya mencari file duplikasi berdasarkan atribut-atribut file seperti nama, *format* atau ukuran file, tanggal file dibuat, *hash* MD5 atau CRC32, atau bahkan membandingkan seluruh isi file dalam bit per bit. Jika file yang dibaca ternyata memiliki salah satu atau lebih atribut yang sama dengan satu sama lain, berarti file-file tersebut merupakan file duplikasi, dan kebanyakan aplikasi pencari file duplikasi menyediakan pilihan untuk menghapus file duplikat secara langsung/permanen, atau memindahkan file duplikat ke *recycle bin* saja.

III. PERANCANGAN



Gambar 3.1 diagram proses pencarian file duplikasi

Tahap awal proses pencarian file duplikasi adalah melakukan pencarian pada seluruh direktori yang dipilih (*directory crawling*) dan membandingkan file-file pada direktori yang ditentukan berdasarkan kriteria-kriteria yang dipilih (ukuran, *format* dan nama file). File-file yang sesuai dengan kriteria akan didaftar – termasuk file 'asli' yang memiliki atribut sama sebagai perbandingan. Untuk kasus ini, terdapat tahap tambahan, setiap file yang dibaca akan dibandingkan nilai MD5nya. Setelah proses *directory crawling* selesai, maka aplikasi mendafarkan hasil pencarian file yang sesuai dengan kriteria pencarian. Terakhir, tergantung dari input user, program dapat langsung memindahkan file yang dipilih pengguna (baik salah satu dari dua/lebih duplikasi yang ditemukan maupun semuanya) ke *Recycle Bin* komputer maupun menghapus file tersebut secara permanen.

Program yang dibuat memiliki fitur-fitur standar dari program pencari file duplikasi, antara lain:

- Mampu mencari file duplikasi dengan *format* yang ditentukan pengguna
- Mampu mencari file dengan ukuran minimal/maximal sesuai dengan input pengguna (byte/KB/MB)
- Mampu mendaftar hasil pencarian file duplikasi pada *list* yang ada, termasuk nilai MD5 serta file asli
- Mampu memindahkan file yang dipilih ke *recycle bin* komputer atau langsung menghapus file tersebut secara permanen

IV. IMPLEMENTASI

Aplikasi dibuat dalam bahasa C#, menggunakan IDE *Microsoft Visual Studio 2010*. *MD5 hashing* dan pengoperasian file/direktori seperti melakukan *directory crawling* menggunakan *library* bawaan dari *Visual Studio 2010*.

Secara garis besar, antarmuka aplikasi dibuat mirip dengan antarmuka aplikasi pencari file duplikasi konvensional; bagian pertama adalah bagian untuk memasukkan input direktori dan *format* file apa saja yang ingin dicari, serta tombol untuk mulai melakukan pencarian. Bagian kedua adalah bagian untuk menentukan ukuran maksimum/minimum dari file yang ingin dicari. Ukuran dapat berkisar dari byte ke megabyte (MB). Bagian ketiga adalah *list* tempat aplikasi mendaftarkan hasil pencarian. File-file yang terdaftar pada *List* dapat dikategorikan berdasarkan nama, ukuran, *format* file, nilai *hash* MD5, serta tempat direktori file tersebut disimpan. Terakhir, bagian keempat memiliki dua tombol yang terpisah untuk menghapus file-file yang dipilih, baik secara permanen atau ke *Recycle Bin*.

Potongan kode dari algoritma MD5, algoritma ini menggunakan operasi bawaan dari Visual Basic 2010 untuk menghasilkan *output hash/message digest* dari string yang dibaca

```
public static string MD5HashString(string
stringToHash)
{
      byte[] result;
      string HashString;

System.Security.Cryptography.MD5 md5 = new
System.Security.Cryptography.MD5CryptoServ
iceProvider();
char[] cctx = stringToHash.ToCharArray();
byte[] bctx =
System.Text.UTF8Encoding.UTF8.GetBytes(cct
x);

result = md5.ComputeHash(bctx);
System.Text.StringBuilder output = new
System.Text.StringBuilder(2 +
(result.Length * 2));

foreach (byte b in result)
{
output.Append(b.ToString("x2"));
}
HashString = output.ToString().ToUpper();
return HashString;
}
```

Untuk melakukan *directory crawling*, hal yang pertama dilakukan adalah melakukan *scan* pada folder yang dipilih:

```
private void ScanFolder(DirectoryInfo
Folder)
{
try
{
        List<FileInfo> fileZ = new
List<FileInfo>();
        List<FileInfo> nofileZ = new
List<FileInfo>();

foreach (string pattern in m_Pattern)
fileZ.AddRange(Folder.GetFiles(pattern,
SearchOption.TopDirectoryOnly));

foreach (string nopattern in
m_SkipPattern)
nofileZ.AddRange(Folder.GetFiles(nopattern
, SearchOption.TopDirectoryOnly));

if (nofileZ.Count != 0)
{
       List<int> toRemove = new
List<int>();
       for (int i = 0; i < fileZ.Count;
i++)
             for (int j = 0; j <
nofileZ.Count; j++)
             {
             if (nofileZ[j].Name ==
fileZ[i].Name)
             {
                    toRemove.Add(i);
                    break;
             }
}
foreach (int index in toRemove)
fileZ.RemoveAt(index);
}
foreach (FileInfo fi in fileZ)
if (fi.Length > skipLessThan && fi.Length
< skipMoreThan)
{
         alFiles.Add(fi);
         m_FilesFound = alFiles.Count;
}
}
catch (Exception IOExec) { }

}
```

Lalu dilanjutkan dengan melakukan *crawling*.

```
private void doCrawl()
{
DirectoryInfo df;
foreach (string path in m_Path)
{
       df = new DirectoryInfo(path);
       BrowseFolders(df);
}

fiZ = new FileInfo[alFiles.Count];
alFiles.CopyTo(fiZ);
OnFoldersDone(EventArgs.Empty);
}
```

Setelah melakukan *directory crawling*, aplikasi melakukan proses pencarian yang lebih spesifik, berdasarkan batas ukuran file minimum dan maksimum yang telah ditentukan.

```
private void SearchFiles()
{
long[] limits = ParseMinMaxSizes();
long skipLessThan = limits[0], skipMoreThan
= limits[1];

string sExt = textBox2.Text, sFolder =
textBox1.Text;

m_fileInfos = null;
alFiles = new List<string[]>();

DCSearch = new DirectoryCrawler(sFolder,
sExt);

firstFolder = DCSearch.PathOne.ToLower();

DCSearch.DeleteToFolder = m_deleteFolder;
DCSearch.FolderChaged += new
EventHandler(DCSearch_FolderChanged);
DCSearch.FoldersDone += new
EventHandler(DCSearch_FoldersDone);

setText(this, "Searching files....");

DCSearch.Crawl(skipLessThan, skipMoreThan);
}
```

Berikut adalah potongan kode dari algoritma perbandingan file

```
public int Compare(object x, object y)
{
if ((x == null) && (y == null))
      {
            return 0;
      }
else
      {
            long xLength =
((System.IO.FileInfo)x).Length;
            long yLength =
((System.IO.FileInfo)y).Length;

          int retval =
((System.IO.FileInfo)x).Length.CompareTo(((S
ystem.IO.FileInfo)y).Length);
          return retval;
      }
}
```

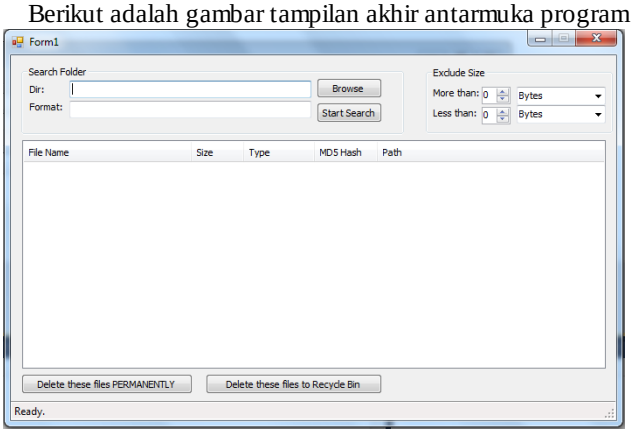
Dan untuk pencarian duplikasi menggunakan perbandingan nilai *hash* MD5:

```
private void FindDuplicate()
{
DCFileInfo = new FileInfoProvider();

DCFileInfo.FileProgressed += new
FileInfoEventHandler(DCFileInfo_FileProgress
ed);

DCFileInfo.FileDone += new
FileInfoEventHandler(DCFileInfo_FileDone);

DCFileInfo.AllFilesRead += new
FileInfoEventHandler(DCFileInfo_AllFilesRead
);

setText(this, "Comparing MD5 of files....");

DCFileInfo.PopulateInfos(m_fileInfos);
}
```

Akhirnya, aplikasi akan mendaftarkan file-file apa saja yang memiliki nilai MD5 yang sama. Berikut adalah potongan kode dari algoritma tersebut:

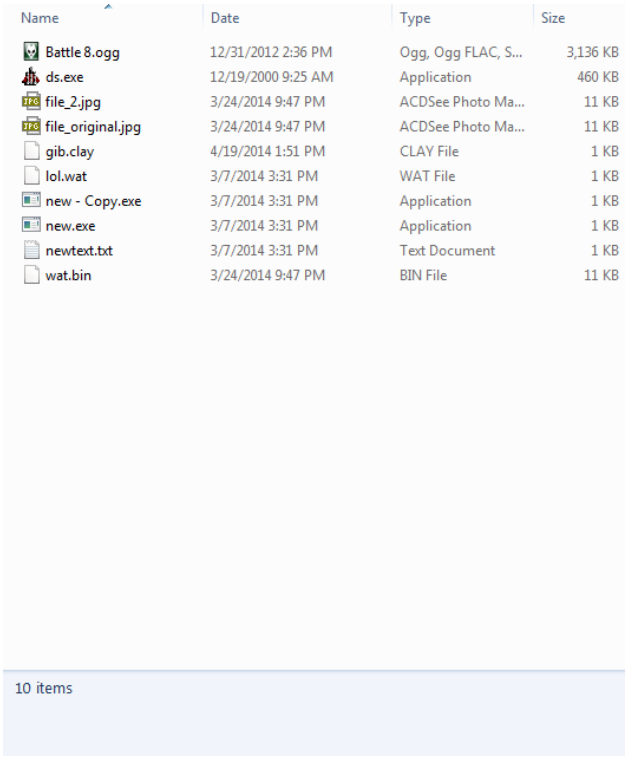
```
private void addListItem(object lvitem, bool
check)
{
if (lstFiles.InvokeRequired)
    lstFiles.BeginInvoke(new
AddListItem(addListItem), new object[] { lvitem,
check });
else
    {
    ListViewItem lv = new
ListViewItem((string[])lvitem);
    lv.Checked = check;
    lv.BackColor =
Colorize(long.Parse(lv.SubItems[1].Text.Replace(
" ", string.Empty)));
    lstFiles.Items.Add(lv);
    }
}
```

Berikut adalah gambar tampilan akhir antarmuka program



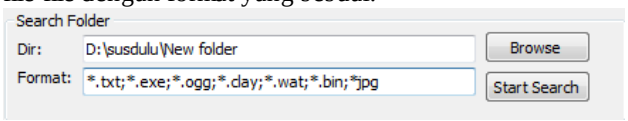
V. PERCOBAAN

Sebagai percobaan, sebuah folder kosong diisi dengan sebuah file yang berperan sebagai file asli, beserta beberapa buah kopian dari file tersebut. Salah satu file duplikasi diberi nama file yang berbeda, sedangkan file lainnya diberi *format* yang berbeda.



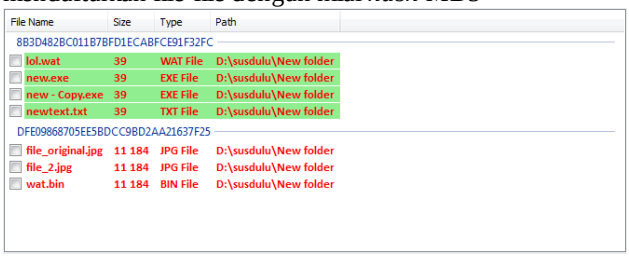
Gambar 5.1 Isi folder pada kondisi awal

Pada aplikasi, kolom format diisi sesuai dengan format dari file-file di dalam folder sehingga aplikasi dapat mencari file-file dengan format yang sesuai.



Gambar 5.3 Isi kolom format

Setelah pencarian selesai, maka aplikasi akan mendaftarkan file-file dengan nilai *hash* MD5



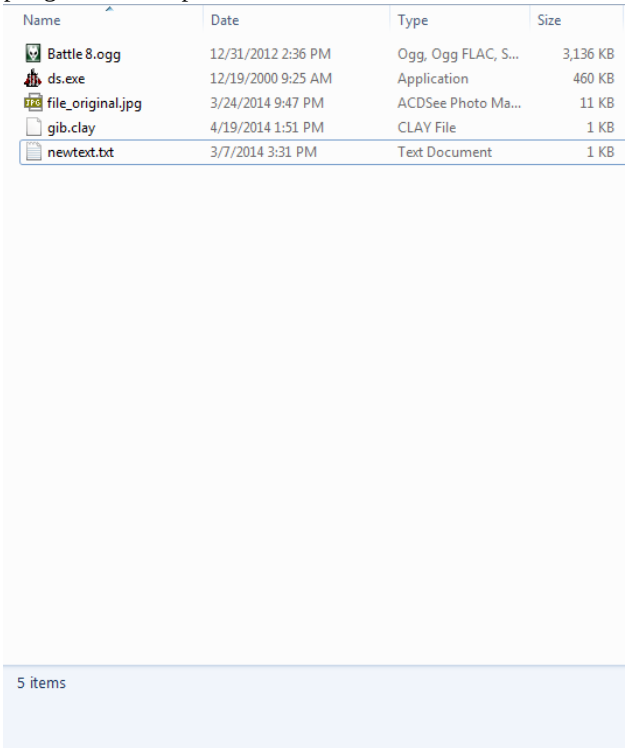
Gambar 5.4 Hasil pencarian

Ditemukan beberapa file dengan nilai MD5 yang sama, yang menandakan file-file tersebut merupakan duplikasi dari salah satu file asli yang memiliki nilai MD5 serupa. Dapat dilihat juga bahwa tidak peduli apa nama maupun format file, karena file-file tersebut memiliki nilai MD5 sama, maka mereka merupakan file duplikasi. Pada titik ini, pengguna dapat memilih file mana yang ingin dihapus dengan mencentang kotak centang di sisi kiri daftar hasil pencarian.

Misalnya, pada kasus ini semua file kecuali file *file_original.jpg* dan *newtext.txt* yang dianggap sebagai file asli akan dihapus secara permanen

Gambar 5.5 Tombol untuk menghapus file yang dipilih secara permanen

Berikut adalah keadaan terakhir dari folder hasil pengeksekusian perintah:



Gambar 5.6 Isi folder setelah file-file duplikasi dihapus

VI. KESIMPULAN

Penggunaan algoritma MD5 untuk mencari file duplikasi dapat dilakukan dengan membandingkan nilai *hash* MD5 dari file-file tersebut. Jika file ditemukan memiliki nilai MD5 yang sama maka file tersebut merupakan duplikasi dari salah satu atau lebih file-file dengan MD5 yang sama.

Selain untuk pengecekan integritas file, algoritma MD5 merupakan salah satu teknik efektif untuk melakukan pencarian file duplikasi.

REFERENSI

[1] http://voices.yahoo.com/top-5-duplicate-file-finders-windows-mac-12177910.html diakses tanggal 4 Mei 2014 pukul 16.40

[2] http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html diakses tanggal 4 Mei 2014 pukul 18.20

[3] http://www.pcworld.com/article/2032515/how-to-find-and-remove-duplicate-files.html diakses tanggal 4 Mei 2014 pukul 18.50

[4] http://www.fourmilab.ch/md5/ diakses tanggal 4 Mei 2014 pukul 18.58

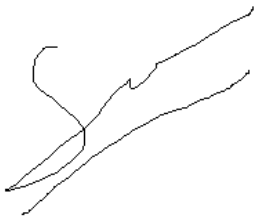
[5] http://tools.ietf.org/html/rfc1321 diakses tanggal 15 Mei 2014 pukul 17.00

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, 18 Mei 2014

ttd



Sandy Gunawan Tanuwijaya