# APLIKASI *IMAGE HASHING* MENGGUNAKAN ALGORITMA SHA256 DENGAN GDI+ .NET FRAMEWORK

Inas Luthfi – NIM : 13506019
*Program Studi Teknik Informatika, Institut Teknologi Bandung*
*Jl. Ganesha 10, Bandung*
E-mail : nassdzr@gmail.com

## Abstrak

Makalah ini membahas tentang penerapan *hashing* pada berkas citra digital untuk memastikan sebuah berkas citra tidak mengalami perubahan. Manipulasi berkas citra digital seringkali dilakukan untuk mengubah informasi dalam sebuah citra, manipulasi tersebut kadang sulit dibedakan melalui indra penglihatan manusia biasa.

Makalah ini akan membahas apa itu fungsi *hash* dan salah satu implementasi algoritmanya yaitu SHA-256 (*Secure Hash Algoritm*). SHA-256 merupakan salah satu algoritma *hashing* yang telah distandardisasi oleh *National Institute of Standards and Technology*. Algoritma ini teruji cukup aman dan dapat diimplementasikan pada rangkaian *byte* data seperti pada citra digital.

*Hashing* seringkali diterapkan untuk memvalidasi apakah suatu rangkaian *byte* data mengalami perubahan, terutama ketika dikirimkan melalui internet ataupun apabila dicurigai terdapat orang yang sengaja mengubah *byte* data (*tampered)*. Dalam makalah ini akan dibahas keuntungan menggunakan fungsi *hash* untuk melakukan pengecekan citra dibandingkan pengecekan per-*pixel*. Disertakan pula hasil implementasi *image hashing* dalam sebuah program berbasiskan .NET menggunakan pustaka GDI+.Net.

***Kata kunci**: Image Hashing*, *Secure Hash Algoritm*, *SHA-256*, *GDI+.Net*

## 1. Pendahuluan

Pada zaman digital seperti sekarang, berkas citra digital tentu bukanlah suatu hal yang aneh. Citra digital atau *image* memiliki bermacam jenis format dan pertukaran citra digital tersebut melalui media internet tentu bukanlah hal yang asing. Ketika sebuah citra digital tersebut tidak memiliki informasi yang berarti, tentu tidak menjadi masalah saat dipertukarkan melalui media internet. Namun lain halnya apabila berkas citra digital tersebut memiliki informasi yang sangat penting, misalnya *blueprint* reactor nuklir yang apabila dimanipulasi oleh pihak tak bertanggung jawab, dapat menyebabkan konstuksi yang salah. Untuk itulah diperlukan sebuah cara untuk memvalidasi bahwa suatu citra digital tidak dirubah.

*Hashing* adalah fungsi yang dapat mentransformasikan byte data menjadi sebuah string dengan panjang yang tetap. *Hashing* seringkali diterapkan untuk memvalidasi apakah suatu byte data mengalami perubahan, terutama ketika dikirimkan melalui internet, karena besar kemungkinan paket data hilang ditengah proses pengiriman. Dengan fungsi hash, integritas *byte* data dapat dijamin. Dalam makalah ini akan dibahas implementasi dan pengujian fungsi *hash* SHA-256 pada berkas citra digital

## 2. Fungsi Hash

Fungsi *Hash* memiliki sifat mentransformasikan data secara searah (*one-way*). Data ditransformasikan oleh fungsi *hash* dengan persamaan

$$Hash = func(Data)$$

Fungsi *hash* adalah fungsi yang menerima masukan *string* yang panjangnya sembarang, lalu mentransformasikannya menjadi *string* keluaran yang panjangnya tetap (*fixed*) (umumnya berukuran jauh lebih kecil daripada ukuran *string* semula)[1]. Dalam artian ini, *string* adalah data.

Sebuah lembaga bernama National Institute of Standards and Technology di Amerika adalah lembaga yang menstandardisasi fungsi *hash.* Lembaga ini telah mengeluarkan sebuah dokumen FIPS 180-2, dimana didalamnya telah ditetapkan lima jenis standar fungsi hash yaitu SHA-1 dan SHA-2 yang terdiri dari SAH-256, SHA-224, SHA-512, SHA-384. Panjang *string*

| Algorithm and variant | | Output size (bits) | Internal state size (bits) | Block size (bits) | Max message size (bits) | Word size (bits) | Rounds | Operations | Collisions found |
|---|---|---|---|---|---|---|---|---|---|
| **SHA-0** | | 160 | 160 | 512 | $2^{64} - 1$ | 32 | 80 | +,and,or,xor,rot | Yes |
| **SHA-1** | | 160 | 160 | 512 | $2^{64} - 1$ | 32 | 80 | +,and,or,xor,rot | None ($2^{52}$ attack) |
| **SHA-2** | *SHA-256/224* | 256/224 | 256 | 512 | $2^{64} - 1$ | 32 | 64 | +,and,or,xor,shr,rot | None |
| | *SHA-512/384* | 512/384 | 512 | 1024 | $2^{128} - 1$ | 64 | 80 | +,and,or,xor,shr,rot | None |

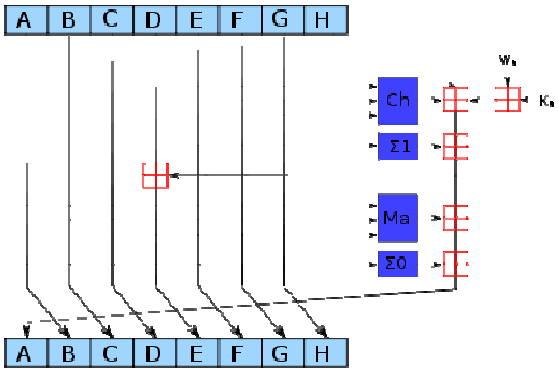
**Tabel 1. Perbandingan fungsi hash SHA-256**

transformasi dari setiap fungsi *hash* tersebut berbeda-beda, namun disini hanya fungsi *hash* SHA-256 yang akan digunakan.

Dapat dilihat pada Tabel 1 bahwa SHA-0 adalah fungsi hash yang lemah karena terlalu mudah mentransformasikan string yang sama dengan data yang berbeda (*Collision*). SHA-0 tidak menjadi standard.

SHA-256 menghasilkan *string* dengan panjang 256 bit. Hal ini menyebabkan algoritma *hash* SHA-256 lebih aman dari pada SHA-1.

SHA-256 terdiri atas 64 kali putaran dan menggunakan 8 buah 32 bit register. Hal ini berbeda dengan SHA-1 yang menggunakan lima buah 32 bit register yang terdiri atas 80 kali puaran. Pemrosesan pesan dilakukan dalam blok 512 bit, sehingga data dibagi ke dalam blok-blok sepanjang 512 bit.




**Gambar 1. Skema sebuah iterasi *hash* dalam SHA-2 *Family[2]***

### 2.1 *Image Hashing*

Ide dari *Image Hashing* sebenarnya sama dengan *hashing* biasa, namun perbedaannya adalah data input fungsi *hash* berbentu rangkaian *byte* yang bukanlah karakter string. Dengan begitu perbedaan satu pixel saja dalam berkas citra, akan menyebabkan keluaran fungsi *hash* yang berbeda.

### 2.2 Keuntungan *Image Hashing*

Keuntungan dari *Image Hashing* adalah kecepatan mendapatkan *hashed string* untuk mengecek integritas berkas citra. Apabila digunakan metode pengecekan per-pixel pada berkas citra waktu yang dihabiskan akan cukup lama.

## 3. Teknologi Implementasi

Dalam implementasi *Image Hashing* untuk menjamin integritas dari berkas citra, dalam makalah ini akan dipakai beberapa teknologi, yaitu .NET *Security Library* dan GDI+

### 3.1 .NET *Security Library*

Implementasi SHA-256 telah disediakan dalam pustaka .NET, sehingga penerapan hashing cukup memanggil fungsi dari pustaka ini. Berikut adalah contoh fungsi tersebut[3]:

```
SHA256Managed shaM = new SHA256Managed();
byte[] hash1 =
shaM.ComputeHash(btImage1);
```

Setelah menjalankan fungsi tersebut, kita akan mendapatkan rangkaian byte sebagai *message digest* atau *string hash*.

### 3.2. GDI+

GDI+ adalah pustaka dari Microsoft yang dipaketkan bersama teknologi .Net framework

dimana tujuan dari pustaka ini adalah memudahkan programmer untuk memanipulasi Bitmap dalam berkas citra digital [3]. GDI+ sendiri sebenarnya merupakan abstraksi dari kartu grafis dari sebuah komputer. Dengan menggunakan pustaka GDI+, manipulasi format berkas citra digital jpg, gif, tiff, dan bmp merupakan hal yang mudah. Misalnya untuk mengambil Bitmap dari sebuah file cukup[4]:

```
Bitmap bm = (Bitmap)
Image.FromFile("<filename>");
```

GDI+ digunakan untuk mengambil data citra per-pixel untuk kemudian digunakan sebagai data input *hashing* dan juga digunakan untuk komparasi perpixel.

## 4. Implementasi

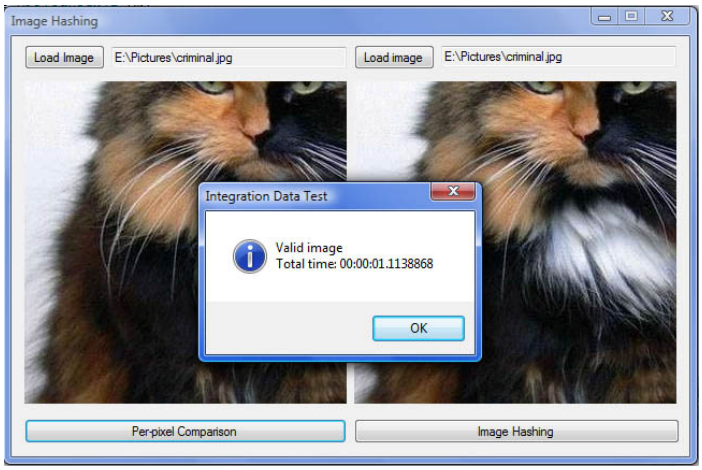
Dalam implementasi *Image Hashing* dibuatlah sebuah aplikasi untuk melakukan validasi integritas berkas citra berformat JPEG, GIF dan Windows Bitmap (BMP). Aplikasi ini dapat me-*load* dua buah berkas citra dan melakukan komparasi diantara keduanya sebagai simulasi pengiriman berkas melalui jaringan. Terdapat dua metode yang digunakan yaitu metode komparasi per-pixel dan metode *Image Hashing*.

Dalam implementasi ini, akan dicek performa metode komparasi per-pixel dan metode komparasi dengan hash melalui beberapa skenario

### 4.1 Skenario 1 : Berkas citra yang sama

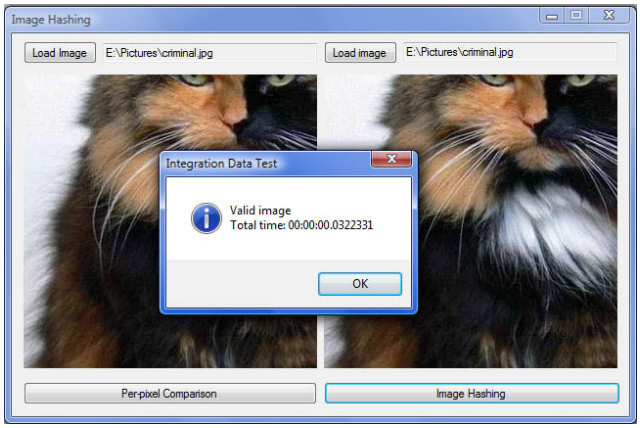
Dengan menggunakan berkas citra yang sama, berarti pengecekan harus memberikan hasil bahwa kedua berkas citra benar-benar sama.

Komparasi per-pixel:




Gambar 2. Komparasi per-pixel dengan gambar yang sama

Komparasi hashing:

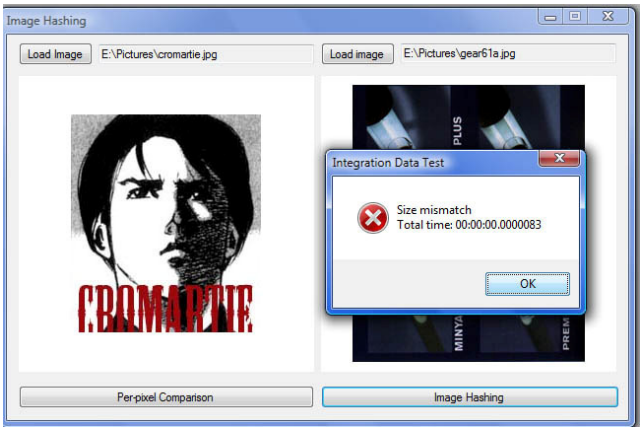



Gambar 3. Komparasi *Image Hashing* dengan gambar yang sama

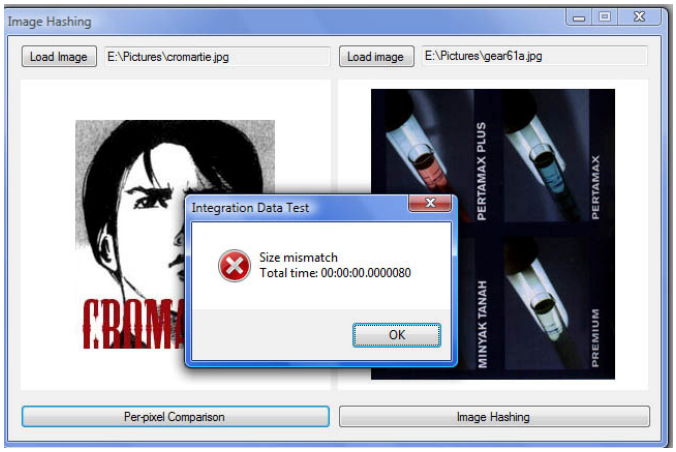
Dapat dilihat, dalam scenario ini, komparasi per-pixel membutuhkan waktu 1.1138868 detik dan komparasi *hashing* hanya membutuhkan waktu 0,0322331 detik

### 4.2 Skenario 2 : Berkas citra yang berbeda ukuran

Dalam scenario ini, kedua metode akan mengecek ukuran berkas terlebih dahulu sebelum melakukan proses komparasi. Oleh karena ukuran berkas citra sudah berbeda, maka proses sesungguhnya tidak akan dilewati.




Gambar 4. Komparasi per-pixel dengan gambar yang berukuran berbeda

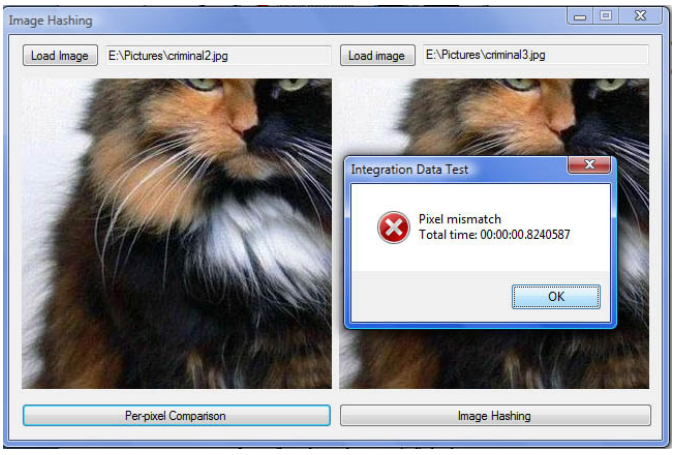



Gambar 5. Komparasi *Image Hashing* dengan gambar yang sama

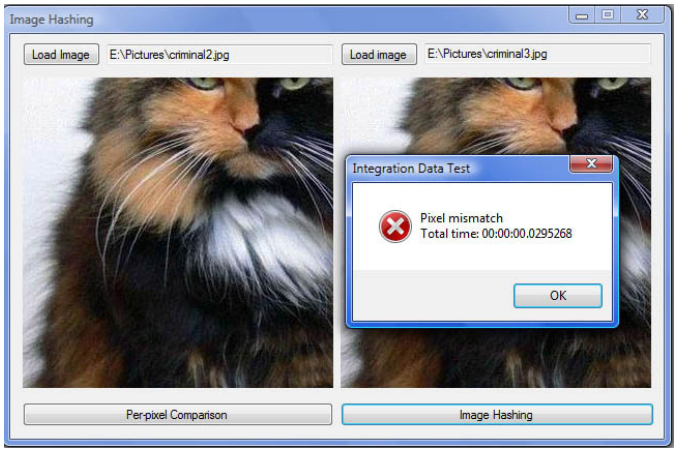
Dapat dilihat, dalam scenario ini performansi kedua metode adalah sama, komparasi per-pixel membutuhkan waktu 0,000008 detik dan komparasi *hashing* hanya membutuhkan waktu 0,0000083 detik (relative sama)

### 4.1 Skenario 3 : Berkas citra dengan ukuran yang sama namun dirubah pixelnya

Skenario terakhir, komparasi akan dilakukan pada berkas citra berukuran sama namun terdapat sedikit perubahan pixel pada berkas citra yang lain.




Gambar 6. Komparasi per-pixel dengan gambar yang berukuran sama namun telah dimanipulasi




Gambar 7. Komparasi *Image Hashing* dengan gambar yang berukuran sama namun telah dimanipulasi

Dapat dilihat, dalam scenario ini, komparasi per-pixel membutuhkan waktu 0,8240587 detik dan komparasi *hashing* hanya membutuhkan waktu 0,0295268 detik

## 5. Kesimpulan

SHA-256 adalah salah satu fungsi *hashing* yang cukup aman untuk digunakan mentransformasikan *byte* data menjadi *hash string*. Setelah dilakukan implementasi dan pengetesan performa, ternyata *image hashing* memiliki keuntungan dapat membuat proses komparasi antara berkas citra digital menjadi lebih cepat daripada proses komparasi yang menggunakan metode per-pixel.

## DAFTAR PUSTAKA

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

[2] Skema iterasi dalam fungsi hashing SHA-2 dan table komparasi antar algoritma SHA, *http://en.wikipedia.org/wiki/SHA1* Tanggal akses: 20 Mei 2009 pukul 15.10.

[3] GDI+ faq *http://www.bobpowell.net/faqmain.htm* Tanggal akses: 20 Mei 2009 pukul 15:20.

[3] Tutorial Codeproject, *Comparing Images using GDI+* http://www.codeproject.com/KB/GDI-plus/comparingimages.aspx Tanggal akses: 20 Mei 2009 pukul 17:30

[4] *Referensi GDI+, http://www.codeproject.com/KB/books/1861004990.aspx* Tanggal akses: 20 Mei 2009 pukul 15:20