ISSN : 2548-9704

PERBANDINGAN ALGORITMA MESSAGE DIGEST-5 (MD5) DAN GOSUDARSTVENNYI STANDARD (GOST) PADA HASHING FILE DOKUMEN

Marthin Benedict[1], Mohammad Andri Budiman[2], Dian Rachmawati[3]
Program Studi S1 Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi,
Universitas Sumatera Utara
[1]marthin_benedict@students.usu.ac.id, [2]mandrib@usu.ac.id, [3]dian.rachmawati@usu.ac.id

Abstrak

Dokumen elektronik memiliki sifat terbuka, artinya isi dokumen dapat dibaca dan diubah dengan mudah oleh pihak-pihak yang tidak berhak. Hal tersebut menyebabkan integritas dokumen menjadi tidak terjamin. Integritas dokumen elektronik dapat dijamin dengan menggunakan teknik kriptografi, salah satunya *hash* function. Ada banyak algoritma yang dapat digunakan untuk *hashing* file atau dokumen, dua algoritma diantaranya adalah algoritma *Message Digest*-5 (MD5) dan algoritma Gosudarstvennyi *Standard* (GOST). Secara garis besar, kedua algoritma mengambil panjang isi file atau pesan dalam *bit* lalu dibagi menjadi blok-blok bit setelah itu pada setiap blok akan dilakukan operasi matematika sehingga menghasilkan 128 *bit* nilai *hash* pada *Message Digest*-5 (MD5) dan 256 *bit* nilai *hash* pada Gosudarstvennyi Standard (GOST). Setelah itu, nilai *hash* diubah dalam bentuk heksadesimal sehingga Message Digest-5 (MD5) akan menghasilkan 32 karakter heksadesimal nilai *hash* dan 64 karakter heksadesimal nilai *hash* pada Gosudarstvennyi Standard (GOST).

Kata Kunci—Kriptografi, Fungsi Hash, Message Digest-5 (MD5), Gosudarstvennyi Standard (GOST), Dokumen Elektronik.

1. PENDAHULUAN

1.1 Latar Belakang

Pada era digital ini, dokumen banyak dibuat menggunakan media elektronik disebut dengan dokumen elektronik. Dokumen elektronik adalah informasi yang direkam atau disimpan dengan menggunakan perangkat komputer atau perangkat elektronik lain untuk menampilkan, menafsirkan atau memprosesnya. Dokumen-dokumen tersebut dapat berupa teks, grafik atau gambar yang dihasilkan oleh perangkat lunak dan disimpan melalui media *disc*. Salah satu perangkat lunak yang paling banyak digunakan untuk mengolah dokumen elektronik adalah Microsoft Office Word dengan ekstensi .docx. Dokumen elektronik memiliki sifat terbuka, artinya isi dokumen dapat dibaca dan diubah dengan mudah oleh pihak-pihak yang tidak berhak. Hal tersebut menyebabkan integritas dokumen menjadi tidak terjamin.

Kriptografi adalah ilmu yang bersandarkan pada teknik matematika untuk berurusan dengan keamanan informasi, seperti kerahasiaan, keutuhan data, dan otentikasi entitas [7]. Integritas dokumen elektronik dapat dijamin dengan menggunakan teknik kriptografi, salah satunya *hash* function. Ada banyak algoritma yang dapat digunakan untuk *hashingfile* atau dokumen, dua algoritma diantaranya adalah algoritma *Message Digest*-5 dan algoritma Gosudarstvennyi *Standard*.

Algoritma *Message Digest*-5 secara garis besar mengambil isi *file* atau pesan yang mempunyai panjang variabel diubah menjadi '*hash*' atau 'sidik jari' yang mempunyai panjang tetap yaitu 128 *bit*. Sidik jari ini tidak dapat dibalik untuk mendapatkan isi pesan atau *file*, dengan kata lain tidak ada orang yang dapat melihat pesan dari 'sidik jari' *Message Digest*-5 tersebut. Lebih tepatnya, *Message Digest*-5 memroses teks masukan ke dalam blok-blok *bit* sebanyak 512 *bit*, kemudian dibagi ke dalam 32 *bit* sub blok sebanyak 16 buah. Keluaran dari *Message Digest*-5 berupa 4 buah blok, masing-masing 32 *bit* yang mana akan menjadi 128 *bit* dari byte terendah A dan tertinggin byte D yang biasa disebut nilai '*hash*'.

Algoritma Gost merupakan blok *cipher* dari bekas Uni Sovyet, yang merupakan singkatan dari "Gosudarstvennyi *Standard*". Algoritma Gost merupakan blok *cipher* 64 *bit* dengan panjang kunci 256 *bit*. Algoritma ini mengiterasi algoritma enkripsi sederhana sebanyak 32 putaran (*round*). Untuk mengenkripsi pertama-tama *plaintext* 64 *bit* dipecah menjadi bagian kiri, L dan 32 *bit*

bagian kanan, R. subkunci (*subkey*) untuk putaran i adalah K_i, enkripsi dilakukan sebanyan 16 putaran (*round*). Pada setiap putaran, blok R (kanan) tidak akan mengalami perubahan apapun karena hanya akan dipindah menjadi blok L pada putaran selanjutnya. Namun blok R akan digunakan bersamaan dengan *subkey* kunci internal untuk diolah pada fungsi F dan akan di XOR-kan dengan blok L (kiri).

Kedua algoritma ini akan menghasilkan perbedaan waktu pemrosesan dan nilai '*hash*' atau 'sidik jari' pada *file* dokumen (.docx), sehingga akan dapat dibandingkan algoritma mana yang lebih baik.

1.2 Rumusan Masalah

Belum adanya penelitian yang membahas tentang perbandingan antara algoritma *hash function Message Digest* 5 (MD5) dan Gosudarstvennyi *Standard* (GOST) pada*hashingfile* .docx.

1.3 Batasan Masalah

1. Jenis *file* yang akan dicari nilai *hash*-nya adalah Microsoft Office Word 2007 ke atas (*.docx).
2. Parameter pembanding yang digunakan adalah waktu *hashing* (milisekon) dan kompleksitas algoritma (*Big* Θ) berdasarkan waktu *hashing*.
3. Bahasa pemrograman yang digunakan adalah C#.
4. Tidak melakukan penelitian tentang *collision*.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mencari algoritma yang lebih baik dalam *hashing file* dokumen dengan membandingkan waktu *hashing*(milisekon) dan kompleksitas (*Big* Θ) pada algoritma *Message Digest* 5 (MD5) dan Gosudarstvennyi *Standard* (GOST).

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah pembaca dapat mengetahui kegunaan dari algoritma *hash* function serta algoritma mana yang lebih baik antara *Message Digest* 5 (MD5) dan Gosudarstvennyi *Standard* (GOST).

2. TINJAUAN PUSTAKA

2.1 Kriptografi

Kriptografi merupakan studi tentang metode untuk mengirim pesan secara rahasia sehingga hanya penerima pesan yang dituju yang dapat menghilangkan penyamaran dan membaca atau memahami isi pesan yang sebenarnya. Berdasarkan etimologinya, kriptografi terdiri dari *kryptos* yang berarti tersembunyi, dan *graphein* yang berarti menulis. Pesan asli disebut dengan *plaintext*, dan pesan yang disamarkan disebut *ciphertext*. Pesan yang disamarkan dan dikirim ke penerima disebut dengan *cryptogram*. Proses mengubah *plaintext* menjadi *ciphertext* disebut *encryption* atau *enciphering*, dan proses kebalikan dari mengubah *ciphertext* menjadi *plaintext*, yang dilakukan oleh penerima yang memiliki pengetahuan untuk menghapus penyamaran, disebut *decryption* atau *deciphering*. Siapapun yang terlibat dalam kriptografi disebut *cryptographer*[5]. Kriptografi (keamanan kriptografi) adalah salah satu cara yang efektif untuk keamanan data [2].

2.2 Fungsi Hash (Hash Function)

Fungsi *hash* adalah fungsi yang menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi string keluaran yang panjangnya tetap (*fixed*) (umumnya berukuran jauh lebih kecil daripada ukuran string semula)[6].

Fungsi *hash* dapat diketahui oleh siapa pun, tak terkecuali, sehingga semuanya dapat memeriksa keutuhan dokumen atau pesan tertentu. Tak ada algoritma rahasia, dan umumnya tak ada pula kunci rahasianya. Jaminan dari keamanan nilai *hash* berangkat dari kenyataan bahwa hampir tidak ada dua *pre-image* yang memiliki nilai *hash* yang sama. Inilah yang disebut dengan sifat *collision free* dari suatu fungsi *hash* yang baik. Selain itu, sangat sulit untuk membuat suatu *pre-image* jika hanya diketahui nilai *hash*-nya saja.

Berikut diuraikan sifat-sifat fungsi *hash* kriptografi [1].

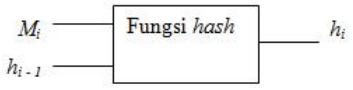
1. Tahan *pre-image* (*pre-image resistent*): Bila diketahuinilai *hashh*, sulit didapatkan (secara komputasi tidak layak) m dimana h = *hash*(m).

2. Tahan *pre-image* kedua (*second pre-image resistant*): Bila diketahui *input* m_1, sulit dicari *input* m_2 (tidak sama dengan m_1) yang menyebabkan *hash*(m_1) = *hash*(m_2).
3. Tahan tumbukan (*collision-resistant*): Sulit dicari dua input yang berbeda, m_1 dan m_2, yang menyebabkan *hash*(m_1) = *hash*i(m_2).

Selain itu, fungsi *hash* mempunyai sifat sebagai berikut.

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2 *H* menghasilkan nilai (*h*) dengan panjang tetap (*fixed-length output*).
3. *H*(*x*) mudah dihitung untuk setiap nilai *x* yang diberikan.
4. Untuk setiap *h* yang dihasilkan, sangat sulit dikembalikan nilai *x* sehingga *H*(*x*) = *h*.
5. Untuk setiap *x* yang diberikan, sangat sulit mencari $y \neq x$ sedemikian sehingga *H*(*y*) = *H*(*x*).
6. Sangat sulit mencari pasangan *x* dan *y* sedemikian sehingga *H*(*x*) = *H*(*y*).

Skema fungsi *hash* ditunjukkan pada Gambar 1



Gambar 1. Fungsi *Hash* [6]

2.3 Landasan Matematika

1. Logika Matematika dan Operator Logika

Logika matematika berisi pernyataan-pernyataan (dapat tunggal maupun gabungan). Pernyataan mempunyai sifat dasar yaitu dapat bernilai benar (pernyataan benar) atau bernilai salah (pernyataan salah), tetapi tidak mungkin memiliki sifat kedua-duanya. Kebenaran dan kesalahan sebuah pernyataan dinamakan nilai kebenaran dari pernyataan tersebut. Operator logika adalah operator yang digunakan untuk membandingkan dua kondisi logika, yaitu logika benar dan logika salah. Berikut adalah operator logika yang digunakan pada algoritma *Message Digest* 5 (MD5).

NOT ( ) :Bernilai kebalikan dari objek
AND (^) :Bernilai benar jika kedua objek bernilai benar
OR (∨) :Bernilai benar jika salah satu objek bernilai benar
XOR (.. :Bernilai benar jika diantara objek saling berlawanan nilainya

Tabel nilai operator logika dapat dilihat pada Tabel 1

1 bernilai benar
0 bernilai salah

TABEL I
Nilai Operator NOT, AND, OR dan XOR

A	B	NOT (A)	A AND B	A OR B	A XOR B
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

2. Konversi Bilangan Binari ke Bilangan Heksadesimal

Konversi dari bilangan binari ke bilangan heksadesimal dapat dilakukan dengan mengkonversikan tiap-tiap empat buah digit binari ke bilangan desimal lalu konversikan bilangan desimal tersebut ke bilangan heksadesimal [4].

Dengan nilai desimal dari setiap digitnya sebagai berikut.

Biner = 1 1 1 1
↓ ↓ ↓ ↓
Desimal = 8 4 2 1

Lalu dari bilangan desimal konversikan ke bilangan hexadesimal, dimana:

Desimal = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
↓↓↓↓↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
Heksadesimal=0 12 3 4 5 6 7 8 9 A B C D E F

Misalnya bilangan biner 11010100 dapat dikonversikan ke heksadesimal dengan cara:

1101→ D
0100→ 4
Jadi 11010100→ D4

3. Operasi Modulo (mod)

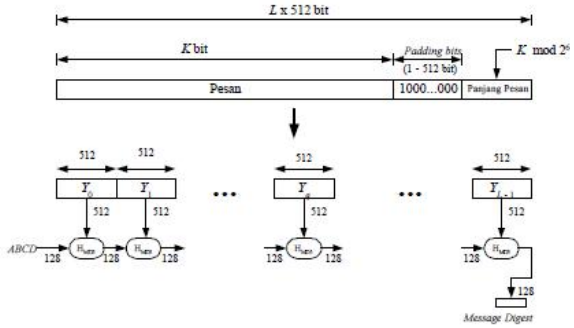
Operasi modulus adalah sebuah operasi yang menghasilkan sisa bagi positif dari suatu bilangan bulat terhadap bilangan bulat lain. Dalam bahasa pemrograman operasi ini umumnya dilambangkan dengan simbol %, mod atau modulo. Misalkan a dan m bilangan bulat (m > 0). Operasi a mod m (dibaca "a modulo m") memberikan sisa jika a dibagi dengan m.

Contoh hasil operasi dengan operator modulo:
(1) 20 mod 5 = 0 (20 = 5 × 4 + 0)

(2) 19 mod 3 = 1 (19 = 3 × 6 + 1)
(3) 0 mod 4 = 0 (0 = 4 × 0 + 0)
(4) 7 mod 9 = 7 (7 = 9 × 0 + 7)
(5) -38 mod 7 = 4 (-38 = 7(-6) + 4)
(6) -12 mod 3 = 0 (-12 = 3(-4) + 0)
Penjelasan untuk -38 mod 7 = -3, karena hasil modulo harus positif maka tambahkan 7 ke hasil modulo sehingga-3 + 7 = 4.

3 Algoritma Message Digest 5(MD5)

MD5 adalah fungsi *hash* satu arah yang dibuat oleh Ronald Rivest pada tahun 1991. MD5 merupakan fungsi *hash* satu arah yang merupakan perbaikan dari MD4 namun lebih kompleks dari MD4, keduanya mirip dari segi model dan juga menghasilkan 128-*bithash*[8]. MD5 memproses teks masukan ke dalam blok-blok *bit* sebanyak 512 *bit*, kemudian dibagi ke dalam 32 *bit* sub blok sebanyak 16 buah. Keluaran dari MD5 berupa 4 buah blok yang masing-masing 32 *bit* digabung menjadi 128 *bit* nilai *hash*[3]. algoritma MD5 diperlihatkan pada Gambar 2



Gambar 2 Pembuatan *Message Digest* dengan Algoritma MD5[6]

4 Algoritma Gosudarstvennyi Standard (GOST)

Algoritma GOSTberasal dari Russia, dan ditetapkan dalam standar GOST R 34.11-94. Salah satu tujuan GOSTadalah memperluas penerapan teknologi informasi saat membuat, mengolah dan menyimpan dokumen, untuk menjaga kerahasiaan, keutuhan dan keaslian isi dokumen tersebut. Algoritma GOST memiliki jumlah proses sebanyak 32 *round*, panjang kunci 256 *bit* dan menggunakan 64 *bitblock cipher* dengan panjang nilai *hash* 256 *bit* [8]. Metoda GOST juga menggunakan 8 buah *S-Box* yang permanen dan operasi XOR serta *Rotate Left Shift*.

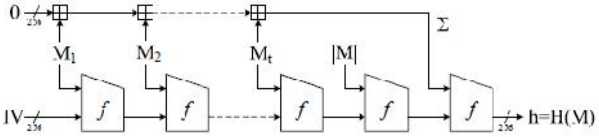
Pesan dengan blok-blok yang berukuran 256 akan diproses oleh fungsi GOST *hash* menjadi nilai *hash* 256 bit. Jika panjang pesan tidak mencapai kelipatan 256, pesan akan di-*padding* seminimal mungkin hingga kondisi tercapai (panjang pesan sama dengan kelipatan 256 bit) [9].

$$H_0 = IV \quad (1)$$

$$H_i = f(H_{i-1}, M_i) \text{ for } 0 < i \le t \quad (2)$$

$$H_{t+1} = f(H_t, |M|) \quad (3)$$

$$H_{t+2} = f(h_{t+1}, \sum) = h, \quad (4)$$



Gambar 3 Struktur Fungsi Gost *Hash*[9]

Setelah di-*padding*, pesan M akan dibagi menjadi tblok pesan ($M = M_1||M_2||M_3||..||M_t$). Nilai *hash* yang dihasilkan akan diproses seperti pada Gambar 3 Dimana $\sum = M_1 \boxplus M_2 \boxplus .. \boxplus M_t$, dan $\boxplus$ merupakan hasil modulo 2^{256} setelah penjumlahan. IV adalah *initialvalue* yang telah didefiniskan sebelum proses dilakukan dan |M| merepresentasikan sebagai bit panjang yang ditambahkan diakhir pesan. Seperti yang terlihat pada persamaan (4), *checksum* ($\sum$) merupakan hasil modulo dari penjumlahan semual blok-blok pesan. fungsi GOST *hash*. Penggunaan komputasi *checksum* ini merupakan bagian penting yang digunakan pada fungsi GOST *hash* dibandingkan MD5 dan SHA-1.

Fungsi kompresi pada fungsi f, pada dasarnya merupakan gabungan dari tiga bagian: *stateupdatetransformation*, *thekeygeneration* dan *theoutputtransformation*.

5 Kompleksitas Algoritma

Tergantung dari banyaknya input, beberapa algoritma dapat selesai diproses dalam waktu kurang dari satu detik. Namun, algoritma yang kompleks dapat memerlukan waktu beberapa menit hingga berhari-hari untuk menyelesaikan prosesnya. Algoritma bekerja berdasarkan input yang dimasukkan user. Ada dua macam kompleksitas algoritma, yaitu:

1. Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n.
2. Kompleksitas ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan n.

Tergantung isi input, waktu proses dapat bervariasi:
Best Case : Omega dilambangkan (Ω)
Average Case : Theta dilambangkan (Θ)
Worst Case : Big-O dilambangkan (O)

3. ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

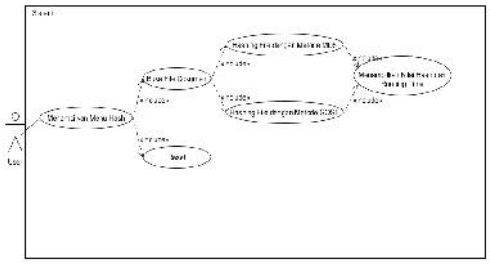
Pada analisis masalah, sebab dan akibat diidentifikasi sehingga nantinya sistem yang akan dibangun dapat bekerja sesuai dengan tujuan utama sistem tersebut dibangun. Permasalahan yang ingin dipecahkan pada penelitian ini adalah untuk mengetahui algoritma mana yang lebih baik antara algoritma *Message Digest*-5 (MD5) dan algoritma Gosudarstvennyi *Standard* (Gost) pada *hashingfile* dokumen. *Hashingfile* menggunakan algoritma Message *digest*-5 (MD5) dan Gosudarstvennyi *Standard* (Gost) dipengaruhi oleh setiap bit pada *file* yang dilakukan proses *hashing*. Jenis *file* yang digunakan yaitu *.docx.

3.2 Perancangan Sistem

Penulis menggunakan *Unified Modeling Language* (UML) pada bagian ini sebagai bahasa standar pemodelan yang digunakan dan berfungsi untuk merancang sistem. Jenis UML yang digunakan pada penelitian ini terdiri dari *use case diagram, activity diagram,sequence diagram, dan flowchart sistem*.

3.3 Use Case Diagram

Use case diagram merupakan gambaran proses pada sistem dari sudut pandang *user*. *Use case diagram* untuk sistem yang akan dibangun dapat dilihat pada Gambar 4

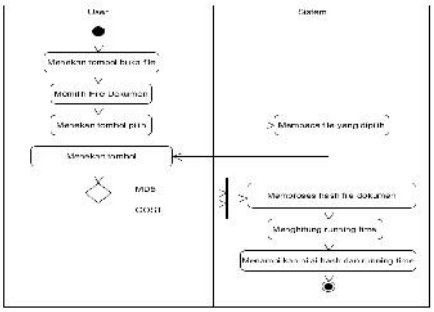


Gambar 4 *Use Case Diagram*

Use Case pada Gambar 4 di atas terlihat di mana *user* melakukan proses *hashing file* dokumen pada menu *hash*. Proses yang ada pada menu *hash* yaitu, pertama-tama *user* meng*input* atau membuka *file* dokumen di mana pada penelitian ini ekstensi yang digunakan adalah *.docx. Setelah meng*input file* dokumen selanjutnya *user* melakukan proses *hashing* dengan algoritma *Message Digest*-5 (MD5) dan Gosudarstvennyi *Standard* (GOST) namun tidak dalam waktu yang bersamaan untuk mencegah ketidakakuratan waktu *hashing* atau *running time*. Setelah itu nilai *hash* dari kedua algoritma akan ditampilkan berikut dengan waktu proses *hashing* pada masing-masing algoritma. Spesifikasi *Use Case* dapat diuraikan sebagai berikut.

3.4 *Activity Diagram*

Activity diagram merupakan gambaran alir aktivitas dalam sistem yang akan dibangun, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* pada sistem yang akan dibangun dapat dilihat pada Gambar 5



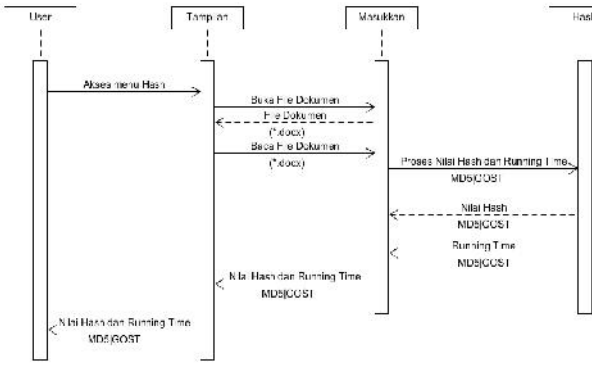
Gambar 5 *Activity Diagram* proses *Hash*

3.5 *Sequence Diagram*

Sequence Diagram adalah bentuk pemodelan sistem yang menggambarkan hubungan antar objek atau objek yang saling berinteraksi melalui pesan dalam eksekusi. *Sequence Diagram* untuk sistem yang akan dibangun pada penelitian ini adalah sebagai berikut.

3.6 Sequence Diagram pada Proses Hash

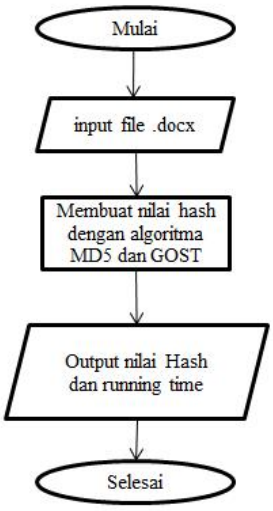
Sequence diagram pada Proses *Hash* dapat dilihat pada Gambar 6



Gambar 6*Sequence Diagram* pada Proses *Hash*

3.7 Flowchart Sistem

Flowchart dari sistem yang akan dibangun dapat dilihat pada Gambar 7 berikut.



Gambar 7*Flowchart* Sistem

4. IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi

Pada tahap ini sistem dibangun sesuai dengan fungsi utamanya menggunakan bahasa pemrograman C# dan IDE visual studio 2015. Tampilan sistem diberikan nuansa hijau dengan alasan, agar *user* merasa nyaman melihat tampilan sistem dan disesuaikan dengan warna logo Universitas Sumatera Utara. Terdapat tiga halaman tampilan pada sistem yang dibangun, yakni halaman Home, halaman Hash, dan halaman About.

4.2 Pengujian

Pengujian terhadap aplikasi perbandingan algoritma *Message* Digest-5 (MD5) dan Gosudarstvennyi Standard (GOST) pada *file* dokumen yang telah dibangun dilakukan dengan menguji kinerja atau *running time* tiap metode dengan beberapa file dokumen yang memiliki besar ukuran file berbeda. File yang sama akan diuji satu persatu dengan metode Message Digest 5 (MD5) dan Gosudarstvennyi Standard (GOST), kemudian hasilnya akan dibandingkan.

1. Pengujian Metode Message Digest-5 (MD5)

Pengujian dengan metode Message Digest 5 (MD5) akan dilakukan dengan menggunakan tiga sample dan pengujian setiap sample dilakukan lima kali percobaan yang akan menghasilkan lima nilai running time berbeda. Setelah melakukan lima kali percobaan, rata-

rata dari kelima nilai running time akan didapat dengan menekan button rata-rata.

2. *Percobaan Pertama Pada Pengujian Metode Message Digest 5 (MD5)*

Hasil yang diperoleh dari pengujian pertama dengan metode Message Digest 5 (MD5) menggunakan *file* sample1.docx pada aplikasi yang telah dibangun adalah sebagai berikut:
Ukuran File = 12532 bytes
Hash = FF7FE44B5F83C897271C2B69EB1DF509
Pada percobaan pertama didapatkan nilai running time 3,7422 milisekon
Pada percobaan kedua didapatkan nilai running time 2,2236 milisekon
Pada percobaan ketiga didapatkan nilai running time 1,7498 milisekon
Pada percobaan keempat didapatkan nilai running time 1,9448 milisekon
Pada percobaan kelima didapatkan nilai running time 1,8931 milisekon
Rata-rata nilai running time dari lima percobaan adalah 2,3107 milisekon

3. *Percobaan Kedua Pada Pengujian Metode Message Digest 5 (MD5)*

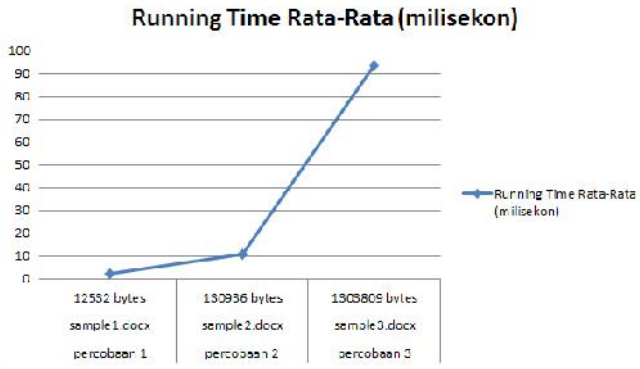
Hasil yang diperoleh dari pengujian pertama dengan metode Message Digest 5 (MD5) menggunakan *file* sample2.docx pada aplikasi yang telah dibangun adalah sebagai berikut:
Ukuran File = 130936 bytes
Hash = E29D771A455AAA9AF4B822C5B6781A2F
Pada percobaan pertama didapatkan nilai running time 10,731 milisekon
Pada percobaan kedua didapatkan nilai running time 9,6328 milisekon
Pada percobaan ketiga didapatkan nilai running time 11,4293 milisekon
Pada percobaan keempat didapatkan nilai running time 12,5219 milisekon
Pada percobaan kelima didapatkan nilai running time 10,3705 milisekon
Rata-rata nilai running time dari lima percobaan adalah 10,9371 milisekon

4. *Percobaan Ketiga Pada Pengujian Metode Message Digest 5 (MD5)*

Hasil yang diperoleh dari pengujian pertama dengan metode Message Digest 5 (MD5) menggunakan *file* sample3.docx pada aplikasi yang telah dibangun adalah sebagai berikut:
Ukuran File = 1303809 bytes
Hash = 3EC2BA2B64E24FC802C7878B5857F6F1
Pada percobaan pertama didapatkan nilai running time 103,0555 milisekon
Pada percobaan kedua didapatkan nilai running time 90,1812 milisekon
Pada percobaan ketiga didapatkan nilai running time 86,2589 milisekon
Pada percobaan keempat didapatkan nilai running time 99,6169 milisekon
Pada percobaan kelima didapatkan nilai running time 90,0011 milisekon
Rata-rata nilai running time dari lima percobaan adalah 93,82272 milisekon

5. *Grafik Fungsi Running Time Rata-Rata Metode MD5*

Grafik fungsi running time rata-rata percobaan 1, 2, dan 3 metode MD5 dapat dilihat pada Gambar 11 berikut.



Gambar 11 Grafik Running Time Rata-Rata Percobaan 1, 2, dan 3 Metode MD5

Pada grafik Gambar 11 terlihat running time rata-rata metode MD5 percobaan 1, 2, dan 3 berbanding lurus dengan ukuran file yang diproses. Dimana running time akan mengalami penambahan berdasarkan ukuran file yang diproses. Maka perhitungan jumlah byte yang diproses per milisekon pada setiap percobaan dapat dilihat pada Tabel II berikut.

TABEL II
Tabel Bytes per Milisekon Pada Metode MD5

Percobaan	Ukuran File (bytes)	Running Time Rata-Rata (milisekon)	bytes per milisekon
1	12532	2.3107	5423.464751
2	130936	10.9371	11971.72925
3	1303809	93.82272	13896.51675
		rata-rata	10430.57025

Pada Tabel II terlihat ukuran file percobaan pertama yang diproses setiap milisekon adalah 5423,464751 bytes, lalu pada percobaan ke dua ukuran file yang diproses setiap milisekon adalah 11971,72925 bytes, dan pada percobaan ke tiga ukuran file yang diproses setiap milisekon adalah 13896,51675 bytes. Sehingga didapatkan rata-rata ukuran file yang diproses setiap milisekon pada percobaan satu, dua, dan tiga adalah 10430,57025 bytes.

6. Pengujian Metode Gosudarstvennyi Standard (GOST)

Pada pengujian dengan metode Gosudarstvennyi Standard sama seperti metode Message Digest-5 (MD5) dilakukan dengan menggunakan tiga sample dengan sample yang sama seperti pada pengujian metode Message Digest-5 (MD5) dan pengujian setiap sample dilakukan lima kali percobaan yang akan menghasilkan lima nilai running time berbeda. Setelah melakukan lima kali percobaan, rata-rata dari kelima nilai running time akan didapat dengan menekan button rata-rata.

7. Percobaan Pertama Pada Pengujian Metode Gosudarstvennyi Standard (GOST)

Hasil yang diperoleh dari pengujian pertama dengan metode Gosudarstvennyi Standard (GOST) menggunakan *file* sample1.docx pada aplikasi yang telah dibangun adalah sebagai berikut

Ukuran File = 12532 bytes
Hash = FC6E81907B9B36F774DE273C69A38C15889 37373071F6C15A75583889 56D66B3
Pada percobaan pertama didapatkan nilai running time 32.6413 milisekon
Pada percobaan kedua didapatkan nilai running time 23,3573 milisekon
Pada percobaan ketiga didapatkan nilai running time 23.0152 milisekon
Pada percobaan keempat didapatkan nilai running time 22.9263 milisekon
Pada percobaan kelima didapatkan nilai running time 23.2804 milisekon
Rata-rata nilai running time dari lima percobaan adalah 25.0441 milisekon

8. Percobaan Kedua Pada Pengujian Metode Gosudarstvennyi Standard (GOST)

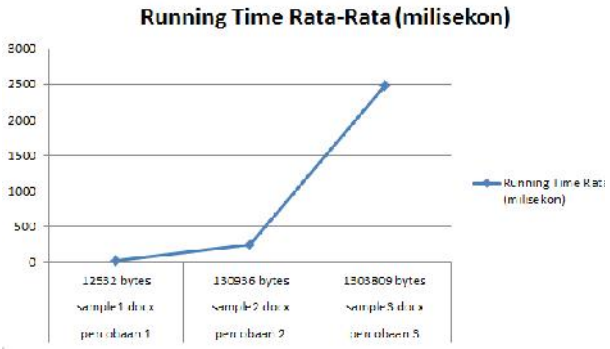
Hasil yang diperoleh dari pengujian kedua dengan metode Gosudarstvennyi Standard (GOST) menggunakan *file* sample2.docx pada aplikasi yang telah dibangun adalah sebagai berikut:
Ukuran File = 130936 bytes
Hash = EB394A20DC0C11B816FF2A7A99D462D5D B93B444A291C90A8CBB1 D47E268C057
Pada percobaan pertama didapatkan nilai running time 250.1703 milisekon
Pada percobaan kedua didapatkan nilai running time 246.1321 milisekon
Pada percobaan ketiga didapatkan nilai running time 252.0027 milisekon
Pada percobaan keempat didapatkan nilai running time 242.2252 milisekon
Pada percobaan kelima didapatkan nilai running time 241.2921 milisekon
Rata-rata nilai running time dari lima percobaan adalah 246.36448 milisekon

9. Percobaan Ketiga Pada Pengujian Metode Gosudarstvennyi Standard (GOST)

Hasil yang diperoleh dari pengujian ketiga dengan metode Gosudarstvennyi Standard (GOST) menggunakan *file* sample3.docx pada aplikasi yang telah dibangun adalah sebagai berikut:
Ukuran File = 1303809 bytes
Hash = 074C240FB86FE320AF05765A60FAADA8C BA0C1E472F33021FFDFD8 19269A4C57
Pada percobaan pertama didapatkan nilai running time 2459.191 milisekon
Pada percobaan kedua didapatkan nilai running time 2458.1591 milisekon
Pada percobaan ketiga didapatkan nilai running time 2438.5171 milisekon
Pada percobaan keempat didapatkan nilai running time 2512.4033 milisekon
Pada percobaan kelima didapatkan nilai running time 2505.9575 milisekon
Rata-rata nilai running time dari lima percobaan adalah 2474.8456 milisekon

10. Grafik Fungsi Running Time Rata-Rata Metode GOST

Grafik fungsi running time rata-rata percobaan 1, 2, dan 3 metode GOST dapat dilihat pada Gambar 12 berikut.



Gambar 12 Grafik Running Time Rata-Rata Percobaan 1, 2, dan 3 Metode GOST

Pada grafik Gambar 12 sama seperti metode MD5 running time rata-rata metode GOST percobaan 1, 2, dan 3 berbanding lurus dengan ukuran file yang diproses. Dimana running time akan mengalami penambahan berdasarkan ukuran file yang diproses. Maka perhitungan jumlah byte yang diproses per milisekon pada setiap percobaan dapat dilihat pada Tabel III berikut.

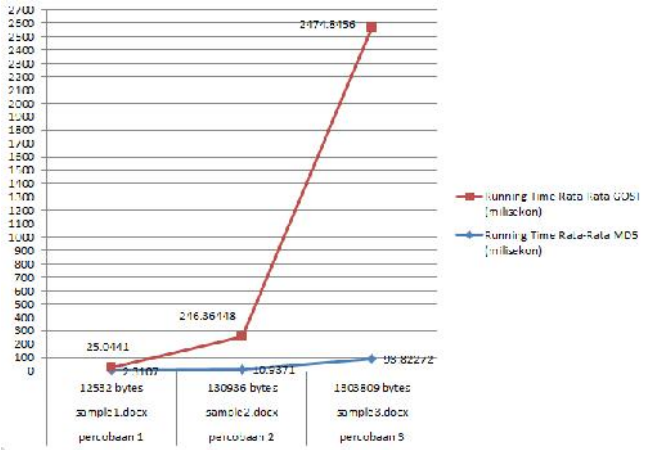
TABEL III
Tabel Bytes per Milisekon Pada Metode GOST

Percobaan	Ukuran File (bytes)	Running Time Rata-Rata (milisekon)	bytes per milisekon
1	12532	25.0441	500.3972992
2	130936	246.36448	531.4727188
3	1303809	2474.8456	526.8243805
		rata-rata	519.5647995

Pada Tabel III terlihat ukuran file percobaan pertama yang diproses setiap milisekon adalah 500,3972992 bytes, lalu pada percobaan ke dua ukuran file yang diproses setiap milisekon adalah 531,4727188 bytes, dan pada percobaan ke tiga ukuran file yang diproses setiap milisekon adalah 519,5647995 bytes. Sehingga didapatkan rata-rata ukuran file yang diproses setiap milisekon pada percobaan satu, dua, dan tiga adalah 519,5647995 bytes.

11. Perbandingan Running Time Rata-Rata Metode Message Digest-5 (MD5) dan Gosudarstvennyi Standard (GOST)

Setelah dilakukan pengujian terhadap masing-masing metode Message Digest-5 (MD5) dan Gosudarstvennyi Standard (GOST) pada tiga sample dokumen elektronik yaitu microsoft word (*.docx) dengan ukuran file yang berbeda sebanyak lima kali percobaan setiap sample dan didapatlah nilai rata-rata running time pada setiap sample. Setelah didapat nilai rata-rata running time setiap sample pada kedua metode hashing, maka dibuatlah grafik garis untuk kedua metode. Agar lebih terlihat perbandingan running time rata-rata pada kedua metode maka nilai running time rata-rata tersebut digabungkan sehingga diperoleh penggabungan grafik pada Gambar 13 berikut.



Gambar 13 Grafik Perbandingan Running Time Rata-Rata Metode MD5 dan Metode GOST

Grafik pada Gambar 13 memperlihatkan perbedaan running time rata-rata metode Message Digest-5 (MD5) dan metode Gosudarstvennyi Standard (GOST) pada ketiga sample file dokumen dengan ekstensi file (*.docx) yang telah dilakukan sebelumnya.

12. Kompleksitas Algoritma Message Digest-5 (MD5)

Perhitungan Kompleksitas algoritma Message Digest-5 dapat dilihat pada Tabel IV berikut.

TABEL IV
Kompleksitas Algoritma Message Digest-5 (MD5)

Code	C	#	C#
var*int*[64] s, K	C1	1	C1
s[0..15] := { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }	C1	1	C1
s[16..31] := { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }	C1	1	C1
s[32..47] := { 4, 11, 16,	C1	1	C1

23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }			
s[48..63] := { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }	C1	1	C1
for i **from** 0 **to** 63	C2	64	64C2
K[i] := floor(2^{32} × abs(sin(i + 1)))	C1	64	64C2
end for	C3	1	C3
var*int* a0 := 0x67452301	C1	1	C1
var*int* b0 := 0xefcdab89	C1	1	C1
var*int* c0 := 0x98badcfe	C1	1	C1
var*int* d0 := 0x10325476	C1	1	C1
append "1" bit **to** message	C4	1	C4
append "0" bit **until** message length in bits ≡ 448 (mod 512)	C4	1	C4
append original length in bits **mod** (2 **pow** 64) **to** message	C4	1	C4
for each*512-bit* chunk **of** message	C2	N	C2N
break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15	C1	N	C1N
var*int* A := a0	C1	N	C1N
var*int* B := b0	C1	N	C1N
var*int* C := c0	C1	N	C1N
var*int* D := d0	C1	N	C1N
for i **from** 0 **to** 63	C2	64N	64C2N
if 0 ≤ i ≤ 15 **then**	C4	16N	16C4N
F := (B **and** C) **or** ((**not** B) **and** D)	C1	16N	16C1N
g := i	C1	16N	16C1N
else if 16 ≤ i ≤ 31	C4	16N	16C4N
F := (D **and** B) **or** ((**not** D) **and** C)	C1	16N	16C1N
g := (5×i + 1) **mod** 16	C1	16N	16C1N
else if 32 ≤ i ≤ 47	C4	16N	16C4N
F := B **xor** C **xor** D	C1	16N	16C1N
g := (3×i + 5) **mod** 16	C1	16N	16C1N
else if 48 ≤ i ≤ 63	C4	16N	16C4N
F := C **xor** (B **or** (**not** D))	C1	16N	16C1N
g := (7×i) **mod** 16	C1	16N	16C1N
dTemp := D	C1	64N	64C1N
D := C	C1	64N	64C1N
C := B	C1	64N	64C1N
B := B + **leftrotate**((A + F + K[i] + M[g]), s[i])	C1	64N	64C1N
A := dTemp	C1	64N	64C1N
end for	C3	N	C3N
a0 := a0 + A	C1	N	C1N
b0 := b0 + B	C1	N	C1N
c0 := c0 + C	C1	N	C1N
d0 := d0 + D	C1	N	C1N
end for	C3	1	C3
var*char* digest[16] := a0 **append** b0 **append** c0 **append** d0	C1	1	C1
leftrotate (x, c)	C5	1	C5
return (x << c) **binary or** (x >> (32-c));	C6	1	C6

$\sum$ = T(n)
= 10C1 + 128C2 + 2C3 + 3C4 + C2N + 9C1N + 64C2N + 4(16C4N) + 8(16C1N) + 5(64C1N) + C3N + C5 + C6
= 10C1 + 128C2 + 2C3 + 3C4 + C2N + 9C1N + 64C2N + 64C4N + 128C1N + 320C1N + C3N + C5 + C6
= Θ(N)

13. Kompleksitas Algoritma Gosudarstvennyi Standard (GOST)

Perhitungan Kompleksitas algoritma Gosudarstvennyi Standard (GOST) dapat dilihat pada Tabel V berikut.

TABEL V
Kompleksitas Algoritma Gosudarstvennyi Standard (GOST)

code	C	#	C#
sbox = (10, 4, 5, 6, 8, 1, 3, 7, 13, 12, 14, 0, 9, 2, 11, 15),	C1	1	C1
(5, 15, 4, 0, 2, 13, 11, 9, 1, 7, 6, 3, 12, 14, 10, 8),	C1	1	C1
(7, 15, 12, 14, 9, 4, 1, 0, 3, 11, 5, 2, 6, 10, 8, 13),	C1	1	C1
(4, 10, 7, 12, 0, 15, 2, 8, 14, 1, 6, 5, 13, 11, 9, 3),	C1	1	C1
(7, 6, 4, 11, 9, 12, 2, 10, 1, 8, 0, 14, 15, 13, 3, 5),	C1	1	C1
(7, 6, 2, 4, 13, 9, 15, 0, 10, 1, 5, 11, 8, 14, 12, 3),	C1	1	C1
(13, 14, 4, 1, 7, 0, 5, 10, 3, 12, 8, 15, 6, 2, 9, 11),	C1	1	C1
(1, 3, 10, 9, 5, 11, 4, 15, 8, 6, 7, 14, 13, 0, 2, 12),	C1	1	C1

BLOCKSIZE = 32	C1	1	C1
C2 = 32 * b'\x00'	C1	1	C1
C3 = hexdec(b'ff00ffff000000ffff0000ff00ffff0000ff00ff00ff00ffff00ff00ff00')	C1	1	C1
C4 = 32 * b'\x00'	C1	1	C1
digest_size = 32	C1	1	C1
A(x): x4, x3, x2, x1 = x[0:8], x[8:16], x[16:24], x[24:32]	C1	1	C1
return b"".join((strxor(x1, x2), x4, x3, x2))	C2	1	C2
P(x): return bytearray((	C2	1	C2
x[0], x[8], x[16], x[24], x[1], x[9], x[17], x[25], x[2],	C2	1	C2
x[10], x[18], x[26], x[3], x[11], x[19], x[27], x[4], x[12],	C2	1	C2
x[20], x[28], x[5], x[13], x[21], x[29], x[6], x[14], x[22],	C2	1	C2
x[30], x[7], x[15], x[23], x[31],	C2	1	C2
u = hin	C1	1	C1
v = m	C1	1	C1
w = strxor(hin, m)	C1	1	C1
k1 = P(w)	C1	1	C1
u = strxor(A(u), C2)	C1	1	C1
v = A(A(v))	C1	1	C1
w = strxor(u, v)	C1	1	C1
k2 = P(w)	C1	1	C1
u = strxor(A(u), C3)	C1	1	C1
v = A(A(v))	C1	1	C1
w = strxor(u, v)	C1	1	C1
k3 = P(w)	C1	1	C1
u = strxor(A(u), C4)	C1	1	C1
v = A(A(v))	C1	1	C1
w = strxor(u, v)	C1	1	C1
k4 = P(w)	C1	1	C1
h4, h3, h2, h1 = hin[0:8], hin[8:16], hin[16:24], hin[24:32]	C1	1	C1
s1 = ns2block(encrypt(sbox, k1[::-1], block2ns(h1[::-1])))[::-1]	C1	1	C1
s2 = ns2block(encrypt(sbox, k2[::-1], block2ns(h2[::-1])))[::-1]	C1	1	C1
s3 = ns2block(encrypt(sbox, k3[::-1], block2ns(h3[::-1])))[::-1]	C1	1	C1
s4 = ns2block(encrypt(sbox, k4[::-1], block2ns(h4[::-1])))[::-1]	C1	1	C1
s = b"".join((s4, s3, s2, s1))	C1	1	C1
x = s	C1	1	C1
for _ in range(12):	C3	12	12C3
x = _chi(x)	C1	12	12C1
x = strxor(x, m)	C1	12	12C1
x = _chi(x)	C1	12	12C1
x = strxor(hin, x)	C1	12	12C1
for _ in range(61):	C3	61	61C3
x = _chi(x)	C1	61	61C1
return x	C2	61	61C1
l = 0	C1	1	C1
checksum = 0	C1	1	C1
h = 32 * b'\x00'	C1	1	C1
m = self.data	C1	1	C1
for i in xrange(0, len(m), BLOCKSIZE):	C3	N	C3N
part = m[i:i + BLOCKSIZE][::-1]	C1	N	C1N
l += len(part) * 8	C4	N	C4N
checksum = addmod(checksum, int(hexenc(part), 16), 2 ** 256)	C1	N	C1N
if len(part) < BLOCKSIZE:	C5	N	C5N
part = b'\x00' * (BLOCKSIZE - len(part)) + part	C1	N	C1N
h = _step(h, part, self.sbox)	C1	N	C1N
h = _step(h, 24 * b'\x00' + pack(">Q", l), self.sbox)	C1	N	C1N
checksum = hex(checksum)[2:].rstrip("L")	C1	1	C1
if len(checksum) % 2 != 0:	C5	1	C5
checksum = "0" + checksum	C1	1	C1
checksum = hexdec(checksum)	C1	1	C1
checksum = b'\x00' * (BLOCKSIZE - len(checksum)) + checksum	C1	1	C1
h = _step(h, checksum, self.sbox)	C1	1	C1
return h	C2	1	C2

$\sum$ = T(n)
= 46C1 + 7C2 + 12C3 + 4(12C1) + 61C3 + 2(61C1) + C3N + 5C1N + C4N + C5N + C5

= 46C1 + 7C2 + 12C3 + 48C1 + 61C3 + 122C1 + C3N + 5C1N + C4N + C5N + C5 = Θ(N)

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil studi literatur, analisis, perancangan, implementasi, dan pengujian sistem serta kompleksitas yang telah didapat, maka kesimpulan yang didapat adalah sebagai berikut:

1. Hasil hash dari algoritma Message Digest-5 (MD5) dan Gosudarstvennyi Standard memiliki panjang karakter yang berbeda, dimana panjang hash pada MD5 berjumlah 32 karakter hexadesimal sedangkan GOST memiliki panjang hash berjumlah 64 karakter hexadesimal.
2. Nilai running time pada algoritma Message Digest-5 (MD5) lebih kecil dibandingkan dengan nilai running time algoritma Gosudarstvennyi Standard (GOST). Artinya proses pada algoritma MD5 lebih cepat dibandingkan proses pada algoritma GOST.
3. Kompleksitas big Θ algoritma Message Digest-5 (MD5) adalah Θ(N) dan big Θ algoritma Gosudarstvennyi Standard (GOST) adalah Θ(N). Kompleksitas kedua algoritma sama-sama Θ(N) yang artinya bernilai linier.

5.2 Saran

Saran-saran yang untuk penelitian maupun pengembangan berikutnya adalah:

1. Sistem ini membandingkan file dokumen, dimana ukuran file tidak terlalu besar. Sebaiknya penelitian selanjutnya menggunakan file dengan ukuran data yang lebih besar agar lebih terlihat perbandingan running time-nya.
2. Sistem ini menggunakan dua algoritma yaitu Message Digest-5 (MD5) dan Gosudarstvennyi Standard (GOST), jadi untuk pengembangan selanjutnya sebaiknya menggunakan tiga atau lebih algoritma untuk dapat melihat perbandingan kedua algoritma hash dengan algoritma hash yang lain.

DAFTAR PUSTAKA

[1] Asmayunita. 2014. *Aplikasi Otentikasi Dokumen Menggunakan Algoritma Gost Digital Signatur*. Universitas Sumatera Utara.

[2] Dolmatov, V. 2010. *GOST R 34.11-94: Hash Function Algorithm*. (*Online*) https://tools.ietf.org/html/rfc5831 (8 Mei 2016).

[3] Harahap, R. 2010. Sistem Pengamanan Data Teks Menggunakan Algoritma Message Digest-5. Universitas Sumatera Utara.

[4] Hartono, J. 1999. *Pengenalan Komputer*. Andi: Yogyakarta.

[5] Mollin, R. A. 2007. *An Introduction to Cryptography Second Edition*. Chapman & Hall/CRC: Florida.

[6] Munir, R. 2006. *Kriptografi*. Informatika, Bandung.

[7] Sadikin, R. 2012. *Kriptografi untuk Keamanan Jaringan*. Yogyakarta: Penerbit Andi.

[8] Schneier, B. 1996. Applied Cryptography: Protocols,
Algorithms, and Sorce Code in C Second Edition. Wiley: New York.

[9] Wagner, D. (Editor). 2008. Cryptanalysis of the GOST Hash Function. In Mendel *et al*. *Advance in Cryptology – CRYPTO 2008*. pp. 162 – 178. Springer: New York.