Techno.COM, Vol. 15, No. 4, November 2016 : 292-302

ALGORITMA KRIPTOGRAFI GOST DENGAN IMPLEMENTASI MD5 UNTUK MENINGKATKAN NILAI *AVALANCHE EFFECT*

Aisyatul Karima,[1], Mohammad Nur Diyatan[2]
[1,2]Teknik Informatika, Fakultas Ilmu Komputer Universitas Dian Nuswantoro
Jl. Imam Bonjol No. 207 Semarang, telp/fax Telp. (024) 3517261
e-mail: aisyatul.karima@gmail.com[1], 1112011207059@mhs.dinus.ac.id[2]

Abstrak

Dalam penerapan algoritma kriptografi GOST selain menggunakan Fiestel Network dengan panjang kunci 256 bit, juga memerlukan perputaran sejumlah 32 kali dalam setiap prosesnya. Baik dalam proses enkripsi maupun dekripsi, algoritma ini mempunyai struktur yang sama, perbedaanya terletak pada penjadwalan kunci (key-schedule). Proses pembentukan kuncinya masih terhitung sederhana, hal inilah yang menyebabkan algoritma GOST rentan terhadap serangan. Nilai avalanche effect merupakan salah satu karakteristik yang menjadi acuan untuk menentukan baik atau tidaknya sebuah algoritma kriptografi. Perubahan sebuah bit pada plaintext atau kunci dapat menghasilkan perubahan beberapa bit dari ciphertext. Semakin banyak perubahan akan semakin baik pula algoritma tersebut. Untuk meningkatkan nilai avalanche effect pada GOST tersebut, diperlukan sebuah algoritma tambahan. Dalam penelitian ini akan ditambahkan algoritma MD5 dalam proses pembentukan kunci dalam algoritma GOST, dikarenakan MD5 mempunyai keunggulan dalam waktu pemrosesannya dibandingkan dengan algoritma SHA.Berdasarkan hasil eksperimen menunjukkan bahwa nilai rata-rata avalanche effect setelah penambahan MD5 memperoleh nilai yang lebih tinggi 52,34% dibandingkan dengan nilai rata-rata avalanche effect sebelum penambahan MD5 yang hanya mencapai angka 46,72% saja.

Kata kunci —Algoritma GOST, MD5, nilai avalanche effect, penjadwalan kunci

Abstract

GOST cryptography algorithm implementation uses the 256 bit key Fiestel Network and need 32 round each the process. This algorithm has the same structure in encryption and decryption process, the differences is on the key scheduling process. This algorithm uses the simple key scheduling process which is causes the GOST algorithm is vulnerable from attack. The avalanche effect value is the one of the characteristic that used to determine the algorithm is better than other cryptography algorithm. Bit alteration on plaintext or key produce the different ciphertext. The more changes the better the algorithm. In order to enhance the avalanche effect value on GOST algorithm requires the additional algorithm. In this study uses MD5 algorithm on key scheduling process of GOST algorithm, because MD5 has advantages in processing time compare with the SHA algorithm. According to the experimental result shows that the average of avalanche effect value with the MD5 algorithm reaches the value higher than only uses GOST algorithm. The avalanche effect value of GOST-MD5 is 52,34%, and the avalanche effect value of GOST is 46,72%

Keywords— GOST Algorithm, MD5 Algorithms, Avalanche Effect Value, Key Scheduling

1. PENDAHULUAN

Perkembangan isu keamanan data pada jaringan komputer meresahkan [1]. Kasus-kasus tentang keamanan data menjadi pekerjaan yang memerlukan keamanan dan penanganan biaya yang cukup besar. Seperti diantaranya keamanan pada sistem pertahanan, sistem perbankan dan sistem yang berskala besar lainnya memerlukan keamanan yang tinggi. Dengan menggunakan metode kriptografi kita dapat mengamankan informasi yang bersifat rahasia dan penting agar tidak dapat diketahui oleh pihak lain yang tidak mempunyai akses terhadap informasi tersebut [2]. Kriptografi merupakan suatu ilmu dan seni yang ditujukan untuk menjaga karahasiaan suatu pesan dengan cara mengubah pesan tersebut menjadi sandi-sandi yang tidak dapat dimengerti.

Berbagai macam metode kriptografi saat ini yang ada, salah satunya adalah algoritma GOST (*Gosudarstvenny Standart*). Algoritma GOST adalah algoritma yang dikembangkan oleh pemerintah Uni Soviet pada saat perang dingin. Algoritma ini merupakan algoritma enkripsi sederhana yang memiliki panjang kunci 256-bit dan 64-bit *block cipher* [3]. Dalam prosesnya algoritma GOST menggunakan 8 buah sbox 4-bit yang mempunyai nilai berbeda-beda. GOST juga memiliki penjadwalan kunci (*key schedule*) sendiri karena GOST memiliki 32 putaran yang harus dilakukan dalam prosesnya. Kelebihan algoritma GOST berupa putaran yang panjang dan panjang kunci yang *default*, yaitu 256-bit. Sedangkan kekurangan algoritma GOST adalah struktur *round*-nya yang sederhana [4]. Algoritma GOST memiliki proses pembentukan kunci yang sangat sederhana [5], dimana kunci yang memiliki panjang 256-bit akan dibagi menjadi 32-bit subkunci : k0,k1,k2,…,k7. Hal tersebut menyebabkan algoritma ini rentan terhadap metoda kriptanalis seperti *Related-key Attack*.

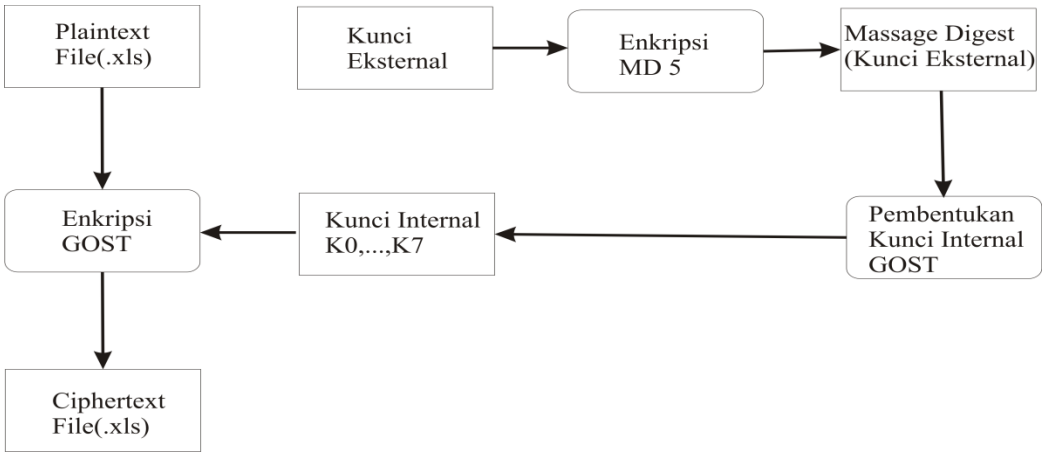
Algoritam MD5 (*Message-Digest Algortihm*) merupakan salah satu algoritma fungsi hash yang memiliki *hash value* 128-bit. MD5 didesain oleh profesor Ronald Rivest dari MIT pada tahun 1991 untuk menggantikan algoritma MD4 yang telah ditemukan kelemahannya. Menurut [6] algoritma MD5 memiliki waktu proses lebih cepat dibandingkan dengan algoritma SHA. Hal tersebut menjadi salah satu kelebihan dari algortima MD5.

Salah satu pengujian yang dilakukan dalam penelitian ini yaitu perhitungan nilai *avalanche effect*s. Pengujian ini dilakukan dengan cara membandingkan bit-bit file input dengan bit-bit file output dari suatu algoritma. Berdasarkan [7] pengukuran *avalanche effect* yang dijadikan sebagai parameter kekuatan suatu algoritma adalah dengan cara mencari perubahan satu bit pada pesan asli (*plaintext*) terhadap pesan tersandi yang dihasilkan (*ciphertext*). Berdasarkan latar belakang tersebut penulis akan menambahkan elemen MD5 pada saat pembentukan kunci dalam algoritma GOST. Dengan implementasi MD5 diharapkan mampu meningkatkan nilai *avalanche effects* guna meningkatkan keamanan data pada algoritma GOST.

2. METODE PENELITIAN

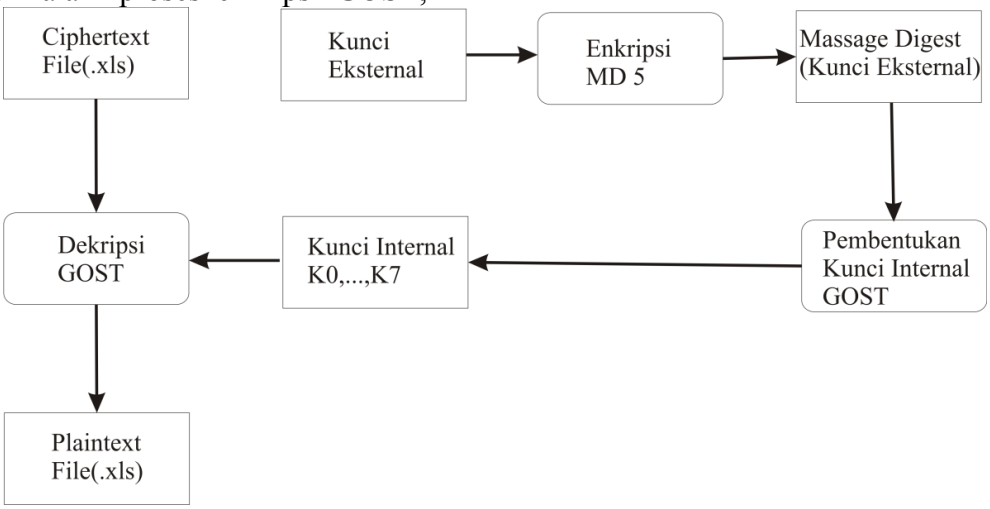
Dalam penelitian ini metode yang diusulkan adalah proses enkripsi dan dekripsi pada data file .xls dengan menggunakan algoritma GOST yang

sudah ditambahkan MD5 pada proses pembentukan kuncinya. Adapun metode penelitian yang digunakan dapat dilihat pada gambar 1 berikut ini.



Gambar 1. Metode Penelitian Proses Enkripsi

Pada gambar 1 diatas menggambarkan alur dari proses enkripsi algoritma GOST yang dikombinaksikan dengan MD5. Sebelum proses enkripsi dimulai, kunci akan dienkripsi dengan menggunakan algoritma MD5 terlebih dahulu. Dalam proses enkripsi GOST, data *plaintext* yang berbentuk file (.xls) diproses bersama kunci yang sudah melewati fungsi MD5. Hasil dari proses enkripsi tersebut adalah *ciphertext* yang berbentuk file (.xls).



Gambar 2. Metode Penelitian Proses Dekripsi

Pada gambar 2 diatas menggambarkan alur dari proses dekripsi algoritma GOST yang dikombinaksikan dengan MD5. Sama seperti proses enkripsi, sebelum proses dekripsi dimulai kunci akan dienkripsi dengan menggunakan algoritma MD5 terlebih dahulu. Dalam proses dekripsi GOST, data *ciphertext* yang berbentuk file (.xls) diproses bersama kunci yang sudah melewati fungsi MD5. Hasil dari proses dekripsi tersebut adalah *plaintext* yang berbentuk file (.xls).

3. HASIL DAN PEMBAHASAN

Proses enkripsi diawali dengan melakukan proses enkripsi dengan algoritma GOST dengan proses pembangkitan kunci terlebih dahulu. Untuk menciptakan kunci dengan panjang 256 bit digunakan algoritma MD5 sebanyak dua kali. Hal ini dikarenakan MD5 hanya menghasilkan *massage digest* dengan panjang 128 bit. Sebagai contoh dengan menggunakan kunci "saya" maka, pertama kunci "saya" dipecah menjadi dua bagian, yakni "sa" dan "ya".

sa = 0111001101100001 (16 bit)

Pesan tersebut ditambahkan dengan bit pengganjal sampai panjang pesan sama dengan 448 bit. Bit pengganjal diawali dengan 1 dan diikuti 0 untuk seterusnya.

16+bit pengganjal=448 → 448-16=bit pengganjal = 432 bit pengganjal.

488 bit pesan yang sudah ditambahkan bit pengganjal kemudian ditambahkan dengan 64 bit yang menyatakan panjang pesan. Sehingga komponen yang ada menjadi seperti berikut.

16 bit pesan + 432 bit pengganjal + 64 bit yang menyatakan panjang pesan.

Sesuai dengan aturan MD5, pesan tersebut akan dilakukan perhitungan sebanyak 64 kali dengan rumus dan operasi yang berbeda-beda. Misal operasi yang digunakan adalah persamaan berikut :

$$A = B + ((A + F(B,C,D) + M_i + K_i) <<< s). \quad (1)$$

Dimana :

- Mi menunjukan pesan ke-i dari blok 512 bit (nilai i=0 sampai 15).
- <<< s menunjukan bit akan digeser sebanyak s bit ke kiri.
- Konstanta K_i didapat dari tabel K_i.
- Dengan nilai :
  A : $01234567_{(16)}$
  B : $89ABCDEF_{(16)}$
  C : $FEDCAB98_{(16)}$
  D : $76543210_{(16)}$

Dari hasil perhitungan sebanyak 64 kali tersebut akan menghasilkan kunci sebagai hasil dari proses pembangkitan kunci yang sudah dilakukan. Sehingga hasil yang diperoleh adalah sebagai berikut :

"sa" = $C12E01F2A13FF5587E1E9E4AEDB8242D_{(16)}$ =
$1100000100101110000000011111001010100001001111111111010101011000011$
$1111000011110100111100100101011101101101110000010010000101101_{(2)}$

"ya" = $D74600E380DBF727F67113FD71669D88_{(16)}$ =
$1101011101000110000000001110001110000000110110111111011100100111111$
$1011001110001000100111111110101110001011001101001110110001000_{(2)}$

Proses selanjutnya adalah pembuatan delapan kunci K0,K2,…,K7 sebagai berikut :

K0 = 01001111100000000111010010000011

K1 = 00011010101011111111110010000101

K2 = 01010010011110010111100001111110

K3 = 10110100001001000001110110110111

K4 = 11000111000000000110001011101011

K5 = 11100100111011111101101100000001

K6 = 10111111110010001000111001101111

K7 = 00010001101110010110011010001110

Setelah pembuatan kunci, proses selanjutnya adalah dilakukan proses enkripsi dengan algoritma GOST. *Plaintext* yang sudah dirubah menjadi bentuk biner dipecah menjadi 2 bagian, yakni bagian R (*right*) dan L (*left*). Sehingga menghasilkan nilai R dan L sebagai berikut :

R = 00011110010100000011010010000110

L = 01010000001101001001111000110100

- Enkripsi iterasi ke-0
  a. Cari nilai L dan R dalam *plaintext*. Diketahui nilai L dan R sesuai diatas, dimana :

  d. Hasil *s-box* digabungkan kembali dan dilakukan *Rotate*

R(0) = 00011110010100000011010010000110

L(0) = 01010000001101001001111000110100

b. Nilai R(0) ditambah dengan K(0) lalu di lakukan modulus 2^{32}

R(0) = 00011110010100000011010010000110 = 508572806

K(0) = 01001111100000000111010010000011= 1333818499

R(0) + K(0) = 508572806 + 1333818499 = 1842391305 mod 232 = 1842391305

1842391305 = 01101101110100001010100100001001

c. Hasil 32 bit dipecah menjadi 8 kelompok yang isinya masing-masing 4 bit lalu masukan kedalam *s-box*

0110-1101-1101-0000-1010-1001-0000-1001

0110 = 6 → SBOX[1] → 0 = 0000

1101 = 13 → SBOX[2] → 7 = 0111

1101 = 13 → SBOX[3] → 0 = 0000

0000 = 0 → SBOX[4] → 7 = 0111

1010 = 10 → SBOX[5] → 9 = 1001

1001 = 9 → SBOX[6] → 6 = 0110

0000 = 0 → SBOX[7] → 13 = 1101

1001 = 9 → SBOX[8] → 2 = 0010

Left Shift (RLS) sebanyak 11 kali

Hasil penggabungan : 00000111000001111001011011010010
RLS : 00111100101101101001000000111000

e. Lakukan XOR hasil RLS dengan L(0)

RLS = 00111100101101101001000000111000
L(0) = 01010000001101001001111000110100 XOR

01101100100000100000111000001100 → R(1)

f. L(1) = R(0)
L(1) = 00011110010100000011010010000110

- Enkripsi iterasi ke-31
  Adapun hasil setelah 32 putaran, tepatnya hasil enkripsi pada iterasi ke-31 adalah sebagai berikut :
  Ciphertext : 11011001000000010111011000110101001001001110010011111111101101110
  ASCII : Ù v5Jr n

Adapun serangkaian proses enkripsi bisa dilihat dalam gambar 3 berikut ini. Berdasarkan gambar tersebut terdapat beberapa inputan yang berisi *plaintext* berupa file serta kunci yang akan digunakan dalam enkripsi. Selain itu terdapat juga menu enkripsi untuk proses enkripsi menggunakan GOST serta menu *save* untuk menyimpan file hasil enkripsi.

Gambar 3. Proses Enkripsi

Sedangkan hasil percobaan dari beberapa proses enkripsi yang sudah dilaksanakan dapat dilihat pada tabel 1 berikut yang merupakan hasil enkripsi dari beberapa file .xls. Dari hasil eksperimen pada tabel 1 dapat dilihat bahwa hasil file yang sudah dienkripsi memiliki rata-rata ukuran file lebih kecil dari aslinya. Dan dapat dilihat pula bahwa semakin besar ukuran file yang di enkripsi semakin lama waktu proses yang dibutuhkan.

Tabel 1: Eksperimen Enkripsi

No	Sebelum Enkripsi		Waktu Enkripsi (detik)	Sesudah Enkripsi	
	Nama File	Size		Nama File	Size
1	mahasiswa.xls	18 kb	11.8	mahasiswaenkripsi.xls	1 kb
2	summary.xls	21 kb	64.6	summaryenkripsi.xls	3 kb
3	fabric.xls	39 kb	311.5	fabricenkripsi.xls	14 kb
4	intransit.xls	34 kb	268.5	intransitenkripsi.xls	11 kb
5	inventory.xls	44 kb	385.2	inventoryenkrpsi.xls	19 kb

6	column.xls	48 kb	418.6	columnenkripsi.xls	9 kb
7	bar.xls	20 kb	18.4	barenkripsi.xls	1 kb
8	one.xls	33 kb	351.8	oneenkripsi.xls	17 kb
9	material.xls	30 kb	99.4	materialenkripsi.xls	5 kb
10	adjust	45 kb	395.8	adjustenkripsi.xls	15 kb

Adapun serangkaian proses dekripsi bisa dilihat dalam gambar 3 berikut ini. Berdasarkan gambar tersebut terdapat beberapa inputan yang berisi *ciphertext* berupa file serta kunci yang akan digunakan dalam proses dekripsi. Selain itu terdapat juga menu dekripsi untuk proses dekripsi menggunakan GOST serta menu *save* untuk menyimpan file hasil dari dekripsi.

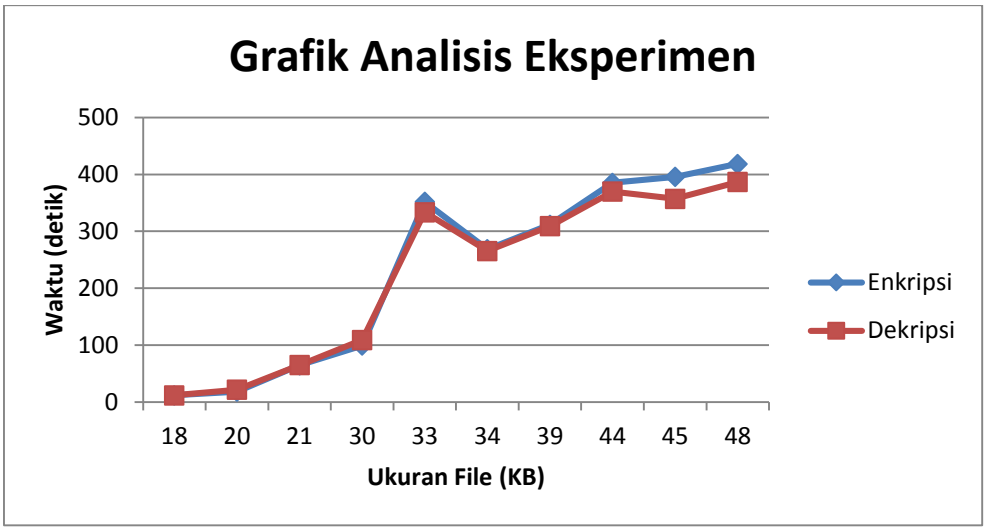
Gambar 4. Proses Dekripsi

Sedangkan hasil percobaan dari beberapa proses dekripsi yang sudah dilaksanakan dapat dilihat pada tabel 2 berikut yang merupakan hasil dekripsi dari beberapa file .xls. Dari hasil eksperimen pada tabel 2 dapat dilihat bahwa ukuran file yang terlah didekripsi menjadi bertambah besar dari sebelum proses dekripsi, tetapi jika dibandingkan dengan file aslinya masih lebih kecil ukuran file-nya.

Tabel 2. Eksperimen Dekripsi

No	Sebelum Dekripsi		Waktu Dekripsi (detik)	Sesudah Dekripsi	
	Nama File	Size		Nama File	Size
1	mahasiswaenkripsi.xls	1 kb	11.9	mahasiswadekripsi.xls	16 kb
2	summaryenkripsi.xls	3 kb	64.9	summarydekripsi.xls	11 kb
3	fabricenkripsi.xls	14 kb	309.1	fabricdekripsi.xls	26 kb
4	intransitenkripsi.xls	11 kb	265.4	intransitdekripsi.xls	21 kb
5	inventoryenkripsi.xls	19 kb	369.9	inventorydekripsi.xls	32 kb
6	columnenkripsi.xls	9 kb	386.7	columndekripsi.xls	35 kb
7	barenkripsi.xls	1 kb	21.7	bardekripsi.xls	6 kb
8	oneenkripsi.xls	17 kb	333.7	Onedekripsi	20 kb

9	materialenkripsi.xls	5 kb	109.2	materialdekripsi.xls	16 kb
10	adjustenkripsi.xls	15 kb	357.1	adjustdekripsi.xls	33 kb



Gambar 5. Grafik Eksperimen

Berdasarkan tabel hasil percobaan di atas dapat disimpulkan melalui grafik pada gambar 5. Berdasarkan gambar 5 tersebut, nampak bahwa semakin besar file yang dienkripsi dan dekripsi maka waktu yang diperlukan untuk proses enkripsi dan dekripsi relative lebih lama pula. Sedangkan untuk perbandingan waktu antara waktu proses enkripsi dan dekripsi relatif sama, tidak memiliki perbedaan yang jauh.

Dalam evaluasi performa ke-1 *plaintext* yang digunakan adalah "enkripsi" dan "encripsi" dengan kunci "wordpass" seperti tampak pada Tabel 3 berikut ini.

Tabel 3: Perhitungan Nilai Avelanche dengan GOST-MD5

Algoritma : GOST-MD5	
Key : wordpass	
Plaintext	*Ciphertext*
enkripsi	10111011101001001010011110011001010011111111101101010100110111
encripsi	10100101010110111111111010110101000001111000110001010101001011

Dalam algoritma GOST-MD5 bit yang berubah sebanyak 39 buah dari 64 total bit yang ada, sehingga dapat dihitung avalanche effect nya sebagai berikut.

$$\text{Avalanche effect (GOST-MD5)} = \frac{\textit{Jumlah bit berubah}}{\textit{Total bit}} x\ 100\% \quad (2)$$

$$= \frac{39}{64} \text{ x } 100\%$$

$$= 60.94\ \%$$

Berdasarkan perhitungan diatas, diketahui nilai *avalanche effect* untuk GOST-MD5 sebesar 60,94 %. Sedangkan untuk perhitungan nilai

avalanche effect untuk algoritma GOST dapat dilihat pada tabel 4 berikut.

Tabel 4: Perhitungan Nilai Avalanche dengan GOST

Algoritma : GOST	
Key : wordpass	
Plaintext	*Ciphertext*
enkripsi	0000101110100110100001001010010010011110100101010010011101101011111100
encripsi	0110101111010011111000100001100101010000010100101010010001001010111

Bit yang berubah dalam algoritma GOST sebanyak 30 bit dari 64 total bit yang ada. Dimana dapat dihitung avalanche effect nya sebagai berikut.

$$\text{Avalanche effect (GOST)} = \frac{\text{Jumlah bit berubah}}{\text{Total bit}} \times 100\%$$

$$= \frac{30}{64} \times 100\%$$

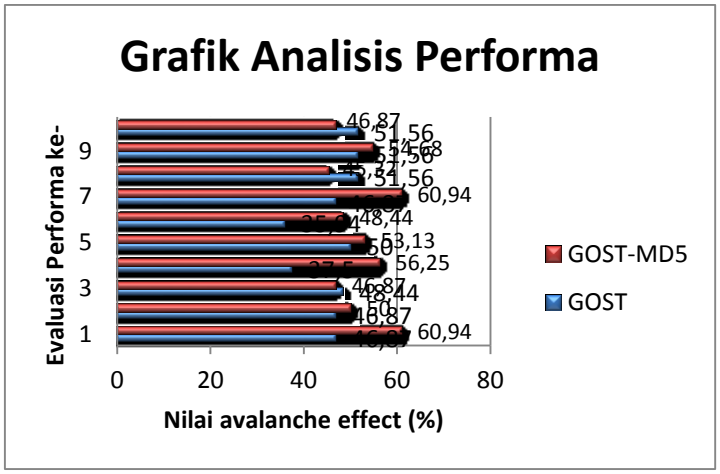
$$= 46.87\ \%$$

Nilai *avalanche effect* yang didapat dari algoritma GOST adalah sebesar 46,87 %. Dimana nilai tersebut lebih kecil dibandingkan dengan nilai *avalanche effect* GOST-MD5. Adapun untuk mengetahui perbandingan nilai *avalanche effect* antara GOST sebelum dan sesudah ditambah MD5 dijelaskan dalam tabel 5 berikut.

Tabel 5 : perbandingan avalanche effect GOST dan GOST-MD5

Evaluasi ke	Nilai *avalanche effect* (%)	
	GOST	GOST-MD5
1	46.87	60.94
2	46.87	50.00
3	48.44	46.87
4	37.50	56.25
5	50.00	53.13
6	35.94	48.44
7	46.87	60.94
8	51.56	45.32
9	51.56	54.68
10	51.56	46.87
Rata-rata	**46.72**	**52.34**

Berdasarkan hasil dari tabel 5 tersebut dapat dibuat grafik yang menggambarkan seluruh avaluasi performa yang telah dilakukan. Penjelasan evaluasi performa dapat dilihat pada gambar 6 berikut.



Gambar 6. Grafik Analisis Performa

Berdasarkan evaluasi performa yang telah dilaksanakan diketahui nilai rata-rata avalanche effect GOST sebesar 46.72% dan nilai rata-rata avalanche effect kombinasi GOST-MD5 sebesar 52.34%. Avalanche effect merupakan salah satu karakteristik yang menjadi acuan untuk menentukan baik atau tidaknya sebuah algoritma kriptografi. Dari hasil eksperimen nampak bahwa nilai rata-rata avalanche effect setelah ditambahkan GOST-MD5 lebih besar dibandingkan dengan nilai rata-rata avalanche effect sebelum menggunakan MD5. Perubahan satu buah bit pada *plaintext* atau kunci dapat menghasilkan perubahan beberapa bit dari *ciphertext*, semakin banyak perubahan akan semakin baik pula algoritma tersebut. Hasil perubahan tersebut dinamakan sebagai avalanche effect. Sebuah algoritma kriptografi akan memenuhi kriteria avalanche effect jika satu buah bit input mengalami perubahan, maka probabilitas semua bit berubah adalah setengahnya.

4. KESIMPULAN

Berdasarkan penelitian yang telah dilaksanakan dengan proses pengujian dan sebagainya, maka dapat ditarik kesimpulan sebagai berikut.

a. Berdasarkan evaluasi performa yang telah dilaksanakan diketahui bahwa algoritma GOST yang sudah ditambahkan dengan MD5 cukup unggul dibandingkan dengan GOST yang belum ditambah MD5 dilihat dari nilai *avalanche effect*. Dengan nilai rata-rata avalanche effect kombinasi GOST-MD5 sebesar 52.34% dan nilai rata-rata avalanche effect GOST sebesar 46.72%. berdasarkan angka tersebut menunjukkan bahwa implementasi MD5 pada algoritma GOST mampu meningkatkan nilai *avalanche effect*.
b. Berdasarkan hasil ekperimen yang telah dilaksanakan diketahui bahwa semakin besar file yang digunakan maka waktu yang dibutuhkan untuk mengenkripsi dan dekripsi semakin lama pula dan ukuran file setelah dilakukan proses enkripsi dan dekripsi menjadi lebih kecil dari ukuran file aslinya.

5. SARAN

Saran-saran untuk untuk penelitian lebih lanjut adalah untuk meningkatkan nilai *avalanche effect* bisa dikembangkan lagi menggunakan algoritma yang berbeda sehingga diperoleh nilai *avalanche effect* yang

lebih tinggi lagi. Selain itu, aplikasi enkripsi dan dekripsi GOST yang sudah ditambah MD5 dapat dikembangkan lagi sehingga mampu mempercepat waktu yang dibutuhkan baik untuk enkripsi maupun dekripsi.

DAFTAR PUSTAKA

[1] A. Kristanto, Keamanana Data pada Jaringan Komputer, Edisi Pertama, Jakarta : Gava Media, 2003.

[2] R. Munir, Kriptografi, Bandung: Informatika Bandung, 2006.

[3] A. Dony, Pengantar Ilmu Kriptografi Teori Analisis dan Implementasi, Yogyakarta : ANDI, 2008.

[4] Yudhistira, "Analisa Perbandingan Algoritma AES dan GOST", Institut Teknologi Bandung, 2011.

[5] P. Novlentina." Studi Teknis Dekripsi dan Enkripsi File dengan Menggunakan Algoritam GOST pada CFB (Cipher Feedback)", Universitas Sumatera Utara, 2011.

[6] G. Piyush dan K. Kendeep, " A Comparative Analysis of SHA and MD5 Algorithm", International Jurnal Computer Science and Information Technologies (IJCSIT), 2014.

[7] Implementasi Dan Analisis Enkripsi XML Untuk Data RSS Feed. Hanum, Ulya Karima. 2011.