IMPLEMENTASI BLUM-BLUM-SHUB DAN *CHAOTIC FUNCTION* UNTUK MODIFIKASI *KEY GENERATING* PADA AES

IMPLEMENTATION OF BLUM-BLUM-SHUB AND CHAOTIC FUNCTION FOR MODIFYING KEY GENERATING ON AES

Muhammad Barja Sanjaya[1], Patrick Adolf Telnoni[2]

[1,2]Prodi D3 Manajemen Informatika, Fakultas Ilmu Terapan, Universitas Telkom
[1]**mbarja@tass.telkomuniversity.ac.id**, [2]**patrick.telnoni@tass.telkomuniversity.co.id**

Abstrak

Salah satu kriptografi yang masih tangguh menurut National Insitute of Standard and Technology (NIST) yakni *Advanced Encryption Standard* (AES). AES bekerja secara *block cipher* menyebabkan diperlukan waktu pemrosesan cukup tinggi. Terlebih, data yang akan diproses pada zaman sekarang membutuhkan *space memory* yang besar. Sehingga data yang akan diproses juga membutuhkan tambahan waktu. Oleh karena itu, dilakukan suatu penambahan modifikasi terhadap metode kriptografi dengan memasangkan metode *Chaotic Function* pada saat memproses pembangkitan kunci yang dihasilkan agar terjadi penurunan kebutuhan waktu proses saat mengenkripsi atau mendekripsi. Selain itu, juga ditambahkan fungsi pembangkit bilangan acak yang bekerja saat memproses pembangkitan kunci dengan metode *random number* Blum-Blum-Shub. Dari hasil penelitian, diperoleh hasil minimalisasi waktu pemrosesan sebesar 20.70% lebih cepat ketika dilakukan penambahan modifikasi dibandingkan tanpa modifikasi. Serta *memory usage* yang dibutuhkan saat proses kriptografi tetap sama yakni sebesar 15 kilo *bytes*.

Kata kunci : kriptografi, *Advanced Encryption Standard, block cipher, Chaotic Function, random number, memory usage*.

Abstract

One of the cryptography algorithms has high strength according to the National Institute of Standards and Technology (NIST), namely Advanced Encryption Standard (AES). AES operates as block cipher consumes higher processing computation. Moreover, nowadays data which will be processed also needs so bigger memory space that the computation takes an extra time. Thus, it is conducted a modification to the algorithm by installing/applying Chaotic Function operation while key generating processing is computed. This process is conducted in order to decrease the time processing consumption. Besides, it is also conducted an additional process of generating random number using Blum-Blum-Shub method to compute selected bits used for encryption key. Based on the research, time processing for encrypyting/decrypting is faster than the pure AES. The time consumption compared to the original AES gets decreased about 20.70%.However, the memory usage needed while conducting the modified one is still the same as the real AES, that is about 15 kilo bytes.

Keywords: cryptography, *Advanced Encryption Standard, block cipher, Chaotic Function, random number, memory usage*.

1. PENDAHULUAN

Seiring pesatnya perkembangan teknologi informasi, berbagai penelitian mengenai keamanan sistem pun berkembang. Salah satu keamanan sistem yang menjadi pendukung dalam perkembangan teknologi yakni kriptografi. Kriptografi merupakan ilmu untuk menyandikan suatu data yang di awal bisa terbaca dan dipahami menjadi data teracak yang tidak bisa dipahami [8]. Kriptografi digunakan untuk mengamankan suatu data agar pihak-pihak yang tidak memiliki wewenang tidak bisa mengakses serta membaca data tersebut [8]. Data tersebut diproses secara komputasi matematis dari data awal yang masih bisa dipahami (*plaintext*) menjadi data akhir berupa data acak yang sulit dipahami oleh manusia (*ciphertext*). Proses pengubahan data tersebut dikenal dengan istilah enkripsi. Sebaliknya, untuk mengembalikan dari data terenkripsi ke dalam bentuk awal disebut dengan istilah dekripsi.

Kriptografi dibedakan menjadi dua jenis berdasarkan dari sudut pandang penggunaan kunci yang digunakan untuk menyandikan data, yakni kriptografi asimetrik dan simetrik [9]. Kriptografi asimetrik menggunakan kunci yang berbeda pada proses enkripsi dan dekripsi. Sedangkan kriptografi simetrik menggunakan kunci yang sama untuk proses enkripsi dan dekripsi. Selain dibedakan berdasarkan kunci yang digunakannya, ada jenis kriptografi lain yakni kriptografi satu arah. Kriptografi jenis ini hanya bisa menyandikan *plaintext* menjadi deretan *bit-bit* (bisa juga dalam bentuk *hexa*) dan tidak bisa dilakukan proses dekripsi pada *cipher* tersebut. Kriptografi satu arah ini digunakan untuk menvalidasi, misalnya dengan mencocokkan data yang di-*generate* dari *server*.

Namun, dewasa ini metode-metode kriptografi tersebut sudah tersebar luas di internet. Hal ini menyebabkan masyarakat dengan mudah memperoleh informasi mengenai metode kriptografi serta mempelajarinya. Sangat jelas bahwa ketika metode kriptografi sudah tersebar akan mengakibatkan berkurangnya ketahanan metode kriptografi tersebut. Karena itu dilakukan suatu penelitian berupa modifikasi pada kriptografi yang bertujuan untuk tetap mempertahankan data agar tidak bisa diakses oleh pihak yang tidak berwenang dan juga mempercepat waktu pemrosesannya serta mempertahankan kestabilan kebutuhan *memory usage* yang dibutuhkan.

Salah satu metode kriptografi yang menjadi objek penelitian adalah kriptografi simetrik, metode *Advanced Encryption Standard* (AES). AES dipilih dikarenakan kesederhanaan sisi algoritmanya tanpa mengurangi sisi ketahanan dan kekuatannya. Hal ini pun disampaikan di [1][2][3] bahwa AES dengan kunci 256-bit akan tetap bertahan. Namun seiring berkembangnya zaman, kebutuhan akan *memory RAM* dan *processor* juga bertambah agar dapat menjadikan waktu untuk proses pun bertambah cepat. Hal tersebut bisa juga dihindari untuk dilaksanakan dengan cara melakukan modifikasi terhadap metode kriptografi supaya tetap dapat mengamankan data dalam waktu yang lebih cepat. Salah satu modifikasi pada kriptografi yang dilakukan yakni dengan menambahkan proses matematis pada saat men-*generate* bilangan acak yang akan digunakan sebagai kunci proses enkripsi dan dekripsi. Blum Blum Shub merupakan suatu metode yang berfungsi men-*generate* bilangan acak secara proses matematis dengan *output* yang dihasilkan adalah deretan angka *biner*. Selain itu perlu dilakukan penambahan metode *Chaotic Function* yang bekerja secara *linier XOR* pada saat *bit-bit* bilangan acak dihasilkan. Keunggulan lain dari metode *Chaotic Function* yang digunakan ini dikarenakan di [4] menjelaskan bahwa waktu proses bisa bertambah makin cepat.

2. DASAR TEORI DAN METODOLOGI

2.1 Dasar Teori

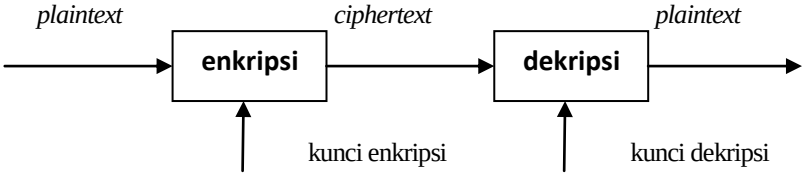
a. Kriptografi

Kriptografi merupakan cabang dari ilmu yang berperan dalam keamanan sistem dengan cara menyandikan suatu data agar menjadi acak dan sulit dipahami oleh pihak yang tidak berwenang. Dengan diterapkan kriptografi maka data yang memiliki informasi tingkat tinggi bisa

tetap aman terlindungi dari akses ilegal oleh pihak yang tidak memiliki wewenang. Menurut [9], kriptografi adalah pengetahuan dan seni menjaga pesan agar tetap aman. Pada konsep kriptografi terdapat empat prinsip dasar, yakni:

- *Confidelity*, prinsip kerahasiaan agar isi pesan yang dikirimkan tetap rahasia dan tidak diketahui oleh pihak lain (kecuaili pihak pengirim, pihak penerima atau pihak-pihak yang memiliki akses).
- *Data integrity*, prinsip keutuhan data agar mampu untuk mengenali dan mendeteksi adanya manipulasi data yang tidak sah oleh pihak lain yang tidak memiliki akses.
- *Authentication*, prinsip keotentikan, merupakan prinsip yang berkaitan dengan identifikasi baik otentikasi pihak-pihak yang terlibat dalam pengiriman data maupun otentikasi keaslian data/informasi.
- *Non-repudiation*, prinsip anti penyangkalan yakni prinsip yang mencegah suatu pihak untuk menyangkal aksi yang dilakukan sebelumnya (menyangkal bahwa pesan tersebut berasal darinya).

Dalam kriptografi, terdapat dua konsep utama yakni proses enkripsi dan dekripsi. Enkripsi adalah proses mengubah data informasi awal menjadi bentuk yang hampir tidak dikenali dengan menggunakan algoritma. Sedangkan dekripsi merupakan kebalikan dari proses enkripsi yakni mengubah kembali bentuk tersandikan tersebut menjadi informasi awal [8]. Berikut gambaran umum proses kriptografi:



Gambar 1. Proses Kriptografi [8]

Adapun istilah yang digunakan pada proses tersebut:

- *Plaintext* (M)
- *Ciphertext* (C)
- Enkripsi (E)
- Dekripsi (D)
- Kunci

b. *Advanced Encryption Standard* (AES)

AES termasuk dalam jenis algoritma kriptografi yang sifatnya simetris dan *block cipher*. Karena itu, algoritma ini memerlukan kunci yang sama saat melakukan proses enkripsi dan dekripsi, serta input dan *output*-nya berupa blok dengan jumlah *bit* tertentu.

AES mendukung berbagai variasi ukuran blok dan kunci yang akan digunakan. Namun AES mempunyai ukuran blok dan kunci yang tetap sebesar 128, 192, dan 256 bit [9]. Pemilihan ukuran blok data dan kunci akan menentukan jumlah proses yang harus dilalui untuk proses enkripsi dan dekripsi. Berikut adalah tabel jumlah perbandingan proses yang harus dilalui untuk masing-masing *input*-an.

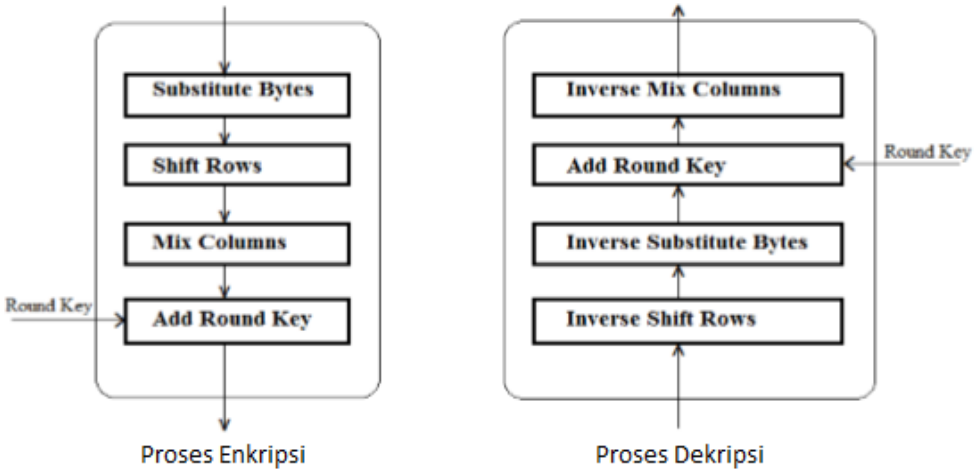
Tabel 1. Perbandingan Panjang Kunci, Ukuran Blok dan Jumlah Proses

Panjang Kunci (*Nk*) Dalam words	Ukuran Blok Data (*Nb*) Dalam words	Jumlah Proses (*Nr*)
4	4	10
6	4	12
8	4	14

Secara garis besar proses enkripsi dan dekripsi di kriptografi AES meliputi *Add Round Key* dan juga melibatkan tiga transformasi, diantaranya: *Sub Bytes*, *Shift Rows*, dan *Mix Columns transformation*.

- *Add Round Key*
- *Sub Bytes Transformation*
- *Shift Rows Transformation*
- *Mix Columns Transformation*

Proses tersebut bisa digambarkan sebagai berikut:



Gambar 2. Skema Kriptografi *Advanced Encryption Standard* [5]

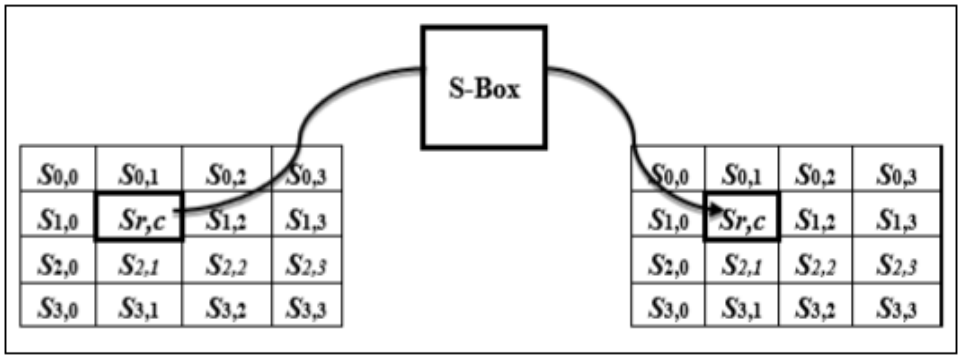
Berdasarkan Gambar 2, pertama kali proses AES dimulai dengan mensubstitusikan *byte plaintext* ke dalam bentuk *byte* lain dengan menerapkan fungsi *Substitution Byte* (bisa disebut juga *Sub Byte*). Proses tersebut tidak hanya memetakan *byte-byte plaintext* ke dalam bentuk *byte* lain namun juga mencari *inverse* dari *byte-byte* tersebut yang akan digunakan saat proses dekripsi. Proses *sub byte* mentransformasikan blok-blok *byte plaintext* ke dalam bentuk *byte* lain dengan menggunakan fungsi Matematika. Fungsi itu secara umum melibatkan perkalian dan penjumlahan pada bentuk matriks. Proses substitusi dilakukan 16 kali untuk tiap blok *plaintext*. Adapun algoritma proses substitusi *byte* dijelaskan pada Gambar 3 sebagai berikut:

```
Algoritma SubstitusiByte
Kamus
    Tabel S_box, S
    r : baris
    c : kolom
Deskripsi
    for r ← 0 to 3 do
        for c ← 0 to 3 do
            S_r,c ← subbyte (S_r,c)
        endfor
    endfor
    subbyte (b : byte) {
        a ← byte −1
        MatrixToByte (a,b)
        for i ← 0 to 7 do
            c_i ← b_i XOR b_(i+4)mod8 XOR b_(i+5)mod8 XOR b_(i+6)mod8 XOR b_(i+7)mod8
            d_i ← c_i XOR ByteToMatrix (0x63)
        endfor
        MatrixToByte (d, d)
        byte ← d
}
```

Gambar 3. Algoritma Transformasi Substitusi Byte [5]

Proses utama pada algoritma transformasi *Sub_byte* adalah men-*generate* blok-blok *plaintext* yang terbentuk dalam matriks berdimensi 8 baris dan 1 kolom menjadi ke bentuk matriks

persegi dengan ukuran 4x4 dan menggantikan nilai dari tiap elemen di matriks tersebut dengan nilai baru yang ditunjukkan di *S_box* berikut.



Gambar 4. Proses Substitusi Pada *S_Box* [5]

Pada saat pemrosesan transformasi, dilakukan juga pencarian nilai balik (*inverse*) dari tiap elemen di tabel *S_box*. Proses pencarian *inverse* yang digunakan adalah dengan menggunakan transformasi *affine* terhadap semua *inverse*-nya.

Setelah mensubstitusi *byte* saat proses pertama mengenkripsi, langkah kedua yang harus dilakukan adalah penggeseran baris per kolom (*Shift Rows*). Adapun jumlah baris yang digeser itu tergantung pada jumlah baris matriks tersebut. Algoritma untuk transformasi penggeseran baris diilustrasikan pada Gambar 5 sebagai berikut:

```
Algoritma ShiftRow
Kamus
    Tabel S
    r : baris
    c : kolom
Deskripsi
    for r ← 0 to 3 do
        shiftRow (Sr, r)
    endfor
    shiftRow (row, t) {
        for c ← 0 to 3 do
                row(c-n)mod4 ← tc
        endfor
    }
```

Gambar 5. Algoritma Transformasi *Shift Row* [5]

Algoritma untuk pemrosesan transformasi *Shift Row* dilakukan tiga kali. Pertama fungsi *shiftRow*(*row*, *t*) menyalin baris ke dalam matriks *t* sementara lalu menggeser satu baris di tiap kolom. Transformasi ini dilakukan untuk menghancurkan linieritas dari pola-pola *byte ciphertext*. Serta, untuk menghancurkan pola tersebut maka dilakukan proses ketiga yakni transformasi *Mixing Column*, yakni proses pengubahan tiap kolom di blok awal ke bentuk kolom baru. Proses transformasi tersebut sebenarnya perkalian matriks terhadap matriks persegi konstan. Berdasarkan referensi [13], perkalian matriks dilakukan secara GF (2^8) dengan modulus 283 (atau dalam *format biner* yakni 10001101, dalam *format polynomial* yakni $x^8 + x^4 + x^3 + x + 1$). Algoritma transformasi *Mix Column* ditunjukkan pada Gambar 6.

Proses terakhir pada enkripsi AES yakni dilakukannya penambahan kunci (dikenal dengan proses *Add Round Key*). Proses ini dilakukan untuk menjamin tidak adanya kunci yang sama di tiap iterasi. Pada transformasi *Add Round Key*, dilakukan proses men-*XOR* tiap kolom pada *state* dengan *key word* yang bersesuaian. Algoritma transformasi *Add Round Key* ditunjukkan pada Gambar 7.

```
Algoritma MixColumn
Kamus
    Tabel S
    c : kolom
Deskripsi
    for c ← 0 to 3 do
        mixColumn (Sc)
    endfor
    mixColumn (col) {
        copyColumn (col, t)
        col0 ← (0x02)•t0 XOR ((0x03)•t1) XOR t2 XOR t3
        col1 ← t0 XOR (0x02)•t1 XOR ((0x03)•t2) XOR t3
        col2 ← t0 XOR t1 XOR (0x02)•t2 XOR ((0x03)•t3)
        col3 ← ((0x03)•t0) XOR t1 XOR t2 XOR ((0x02)•t3)
    }
```

Gambar 6. Algoritma Transformasi *Mix Column* [5]

```
Algoritma AddRoundKey
Kamus
    Tabel S
    c : kolom
Deskripsi
    for c ← 0 to 3 do
        Sc ← Sc XOR wround+4c
    Endfor
```

Gambar 7. Algoritma Pembangkitan *Add Round Key* [5]

Berdasarkan Gambar 7, hal yang perlu diperhatikan pada waktu pembangkitan kunci di tiap ronde adalah jumlah ronde atau putaran (*round*) yang harus dilakukan. Jika *Nr* adalah jumlah putaran (*round*), fungsi *key-expansion* membuat N_{r+1} (128-*bit*) ronde kunci dari kunci *cipher* 128-*bit*. Kunci pertama digunakan untuk inisialisasi di ronde pertama. Kunci putaran sisanya digunakan untuk transformasi terakhir.

Untuk kriptografi AES versi 128-*bit*, ada 10 putaran dan 44 *word* seperti yang ditunjukkan pada tabel berikut:

Tabel 2. Penjadwalan Kunci yang diiterasi di tiap ronde [5]

Ronde Putaran	*Words*
Pre-ronde	$w_0\ w_1\ w_2\ w_3$
1	$w_4\ w_5\ w_6\ w_7$
2	$w_8\ w_9\ w_{10}\ w_{11}$
3	. . . . .
. . . .	
Nr	$w_{4Nr}\ w_{4Nr+1}\ w_{4Nr+2}\ w_{4Nr+3}$

c. Random Number

Bilangan acak adalah deretan nilai yang acak dan tidak dapat diprediksi secara keseluruhan. Untuk menghasilkan bilangan acak merupakan hal yang sulit, kebanyakan pembangkit bilangan acak (*Random Number Generator* = RNG) mempunyai beberapa bagian yang dapat diprediksi dan berhubungan [12]. Kebanyakan RNG mengulang *string* yang sama setelah melakukan *n* putaran. Sedangkan ada beberapa RNG lainnya menghasilkan nilai acak dengan berfokus pada suatu area tertentu dan mendistribusikannya secara seragam.

Random Number Generator (RNG) adalah suatu peralatan komputasional yang dirancang untuk menghasilkan suatu urutan nilai yang tidak dapat ditebak polanya dengan mudah, sehingga urutan nilai tersebut dapat dianggap sebagai suatu keadaan acak (*random*). RNG ini tidak dapat diterapkan dalam prakteknya. Bilangan acak yang dihasilkan oleh komputer sekalipun tidak benar-

benar acak dan kebanyakan bilangan acak yang diterapkan dalam kriptografi juga tidak benar-benar acak, tetapi hanya berupa acak semu. Ini berarti bahwa bilangan acak yang dihasilkan itu dapat ditebak susunan atau urutan nilainya. Dalam kriptografi, bilangan acak sering dibangkitkan dengan menggunakan pembangkit bilangan acak semu (*Pseudo Random Number Generator*) [9].

Suatu *Pseudo Random Number Generator* (PRNG) merupakan suatu algoritma yang menghasilkan suatu urutan nilai dimana elemen-elemennya bergantung pada setiap nilai yang dihasilkan [7]. *Output* dari PRNG tidak benar-benar acak, tetapi hanya mirip dengan properti dari nilai acak. Kebanyakan algoritma dari *Pseudo Random Number Generator* ditujukan untuk menghasilkan suatu sampel yang secara seragam terdistribusi. PRNG ini sering digunakan dalam kriptografi pada proses pembentukan kunci dari metoda kriptografi. Tingkat kerumitan dari PRNG ini menentukan tingkat keamanan dari metoda kriptografi. Semakin rumit (kompleks) PRNG yang digunakan maka semakin tinggi tingkat keamanan dari metoda kriptografi [10].

d. Blum-Blum-Shub

Blum-Blum Shub (BBS) merupakan suatu *Pseudo Random Number Generator* yang diajukan pada tahun 1986 oleh Lenore Blum, Manuel Blum dan Michael Shub. BBS memiliki bentuk persamaan:

$$X_{n+1} = X_n^2 \ \textbf{mod}\ m \qquad (1)$$

dengan m merupakan hasil dari perkalian dua buah bilangan prima besar p dan q, serta *output*-nya dalam *Least Significant Bit* dari X_n dimana hal yang sama sebagai *parity* dari X_n. Dua buah bilangan prima p dan q harus kongruen terhadap 3 mod 4 dan *Greatest Common Divisor* (GCD) harus kecil. *Generator* ini sering digunakan untuk aplikasi kriptografi, karena *generator* ini tidak begitu cepat.

Bagaiman pun juga, *generator* ini mempunyai bukti keamanan yang kuat, dimana berhubungan dengan kualitas *generator* karena sulitnya faktorisasi *integer*. Berikut langkah-langkah algoritma dari BBS [11]:

1. Pilih dua bilangan prima p dan q, di mana p dan q keduanya kongruen terhadap 3 modulo 4.
   $p \equiv 3 \bmod 4$ dan $q \equiv 3 \bmod 4$.
2. Hasilkan bilangan bulat Blum n dengan menghitung $n = p \times q$.
3. Pilih lagi sebuah bilangan acak s sebagai umpan, bilangan yang dipilih harus memenuhi kriteria:
   a. $2 \le s < n$.
   b. s dan n adalah relatif prima.
4. Hitung nilai $x_o = s^2$ **mod** n.
5. Hasilkan bilangan *bit* acak dengan cara :
   a. Hitung $x_i = x_{(i-1)}^2$ **mod** n.
   b. Hasilkan $z_i =$ *bit* - *bit* yang diambil dari x_i. *Bit* yang diambil bisa merupakan LSB (*Least Significant Bit*) atau hanya satu *bit* atau sebanyak j bit (j tidak melebihi log_2 (log_2 n)).
   Bilangan *bit* acak dapat digunakan langsung atau di-*format* dengan aturan tertentu, sedemikian hingga menjadi bilangan bulat.

e. Chaotic Function

Teori *chaos* berasal dari teori sistem yang memperlihatkan kemunculan yang tidak teratur atau digunakan juga untuk menjelaskan kemunculan data acak. Sampai saat ini, teori *chaos* tersebut banyak diimplementasikan dalam kriptografi. Sistem *chaos* ini sangat berguna karena bisa menghasilkan bilangan semi acak yang tidak memiliki periode perulangan. Sistem *chaos* memiliki sifat yang sangat berharga yang bisa ditetapkan dalam kriptografi, yaitu peka pada perubahan kecil

pada kondisi awal sistem [4]. Salah satu dari dua prinsip Shanon yang dijadikan panduan dalam perancangan algoritma kriptografi adalah difusi (*diffusion*), yang artinya adalah menyebarkan pengaruh 1 *bit* (atau digit) *plaintext* ke seluruh *bit* (digit) *ciphertext* dengan maksud untuk menyembunyikan hubungan *statistic* antara *plaintext* dengan *ciphertext* [6]. Bisa juga artinya untuk menyebarkan 1 *bit* kunci (*key*) ke seluruh *bit ciphertext*. Apabila *bit* kunci diubah sedikit saja, maka *ciphertext* yang dihasilkan akan berbeda secara signifikan. Hal ini akan menyebabkan kriptanalisis akan kesulitan mendekripsikan *ciphertext* dengan menganalisis kunci dan mencari pola hubungan antara *plaintext* dengan *ciphertext*.

Meskipun sedemikian acak, sistem *chaos* tetap bersifat *deterministic*, yang berarti nilai-nilai acak yang dihasilkan bisa dibangkitkan kembali untuk menghasilkan nilai-nilai acak yang sama, dengan syarat nilai awal yang digunakan adalah fungsi *logistic* (*logistic map*). *Logistic map* berupa persamaan *iterative* yang dijabarkan sebagai berikut:

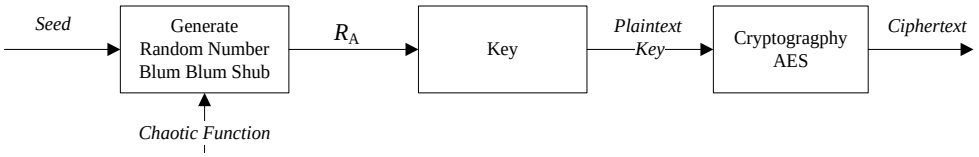
$$x_{i+1} = rx_i (1 - x_i) \qquad (2)$$

Pada persamaan di atas, variabel x_i akan menjadi nilai awal. Konstanta r berupa laju pertumbuhan dengan rentang nilai $0 \leq r \leq 4$. Ketika $r = 4$, iterasi bergantung sepenuhnya pada nilai awal x_i dan nilai-nilai yang dihasilkan muncul acak meskipun sistem ini *deterministic* [4]. Artinya, konstanta yang tepat agar nilai-nilai *chaos* benar-benar acak adalah konstanta dengan nilai 4.

Agar barisan nilai *chaotic* dapat dipakai untuk enkripsi dan dekripsi dengan *stream cipher*, maka nilai-nilai *chaos* tersebut dikonversi ke nilai *integer*. Ada beberapa teknik konversi dari nilai-nilai *chaotic* menjadi nilai *integer*. Salah satunya, nilai *chaos* dikalikan dengan 10 berulangkali sampai ia mencapai panjang angka (*size*) yang diinginkan, selanjutnya potong hasil perkalian tersebut untuk mengambil bagian *integer*-nya saja [4].

2.2 Metodologi

Pada sub bagian berikut dijelaskan mengenai rancangan sistem penambahan modifikasi pada metode kriptografi yang diteliti. Secara garis besar, proses yang dilakukan pada penelitian ini digambarkan dengan *block diagram* berikut:



Gambar 8. Rancangan Blok Diagram

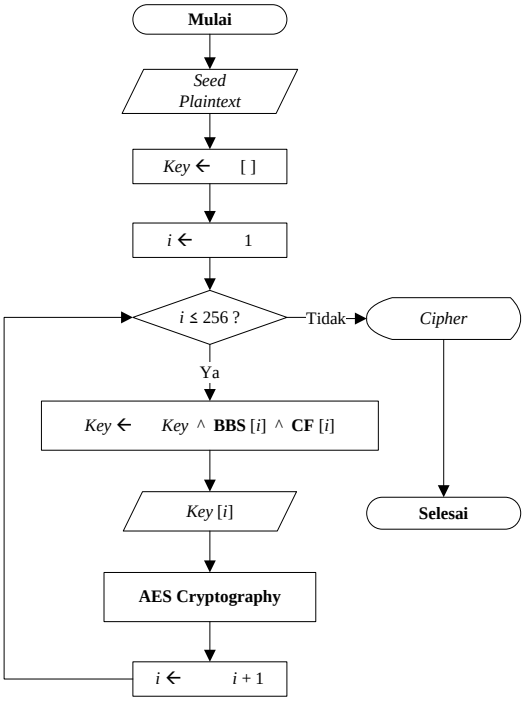
Dari Gambar 8, berikut urutan blok diagram proses yang dilakukan, yakni:
1. Men-*generate Random Number Blum-Blum-Shub*
2. Men-*generate key*
3. Enkripsi dan Dekripsi

Detail proses yang dilakukan pada penelitian ini pada saat memproses men-*generate key* sebelum dilakukan enkripsi atau dekripsi dijelaskan sebagai berikut:

Penjelasan mengenai *flowchart* sistem pada Gambar 9, yakni:

1. Lakukan *input seed* sebagai suatu bilangan dan *plaintext* sebagai data yang akan diproses. *Seed* merupakan bilangan *integer* dengan range $0 < seed < 2^8$. Nilai 2^8 atau setara angka desimal 256 (satuan *bit*) digunakan untuk representasi dari panjang kunci yang digunakan di kriptografi AES (AES 256-*bit*).

2. Siapkan *array* (ukuran 1 dimensi) untuk menampung hasil kunci yang dibangkitkan di tiap ronde / putaran.
3. Lakukan proses kriptografi AES dimulai dari iterasi kunci ke-*i* pertama dengan *key* yang dihasilkan dengan penambahan *XOR* terhadap *bit-bit* hasil *Random Number* Blum Blum Shub dan penambahan metode *Chaotic Function* yang bertujuan untuk mengacak *bit-bit* kunci yang telah dihasilkan tersebut di setiap iterasi / ronde / putaran.
4. Lakukan proses di poin 3 pada iterasi untuk blok-blok *byte plaintext* berikutnya sampai blok terakhir.



Gambar 9. Rancangan Modifikasi Kriptografi AES

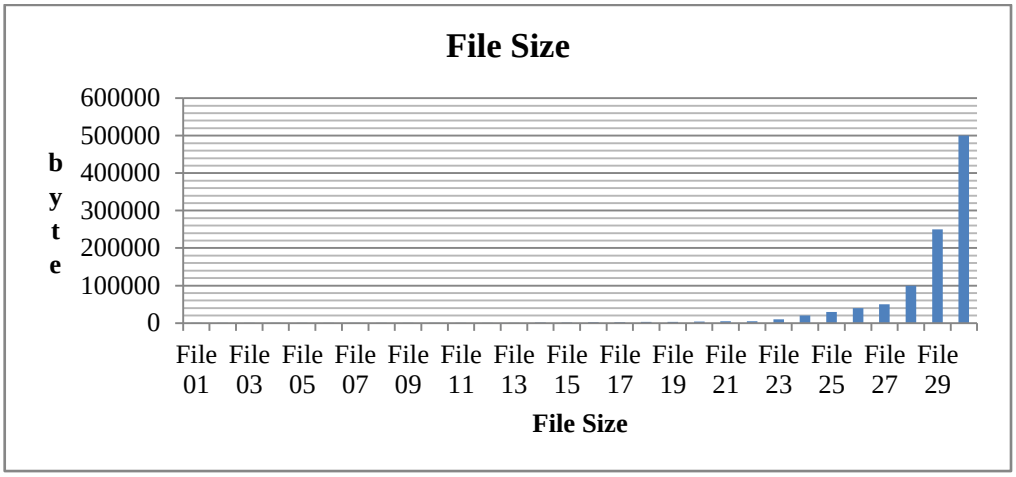
3. Pembahasan

Pada sub ini dijelaskan mengenai pengujian yang dilakukan pada data bertipe *text* dalam *format file* Notepad. Data tersebut diurutkan berdasarkan kuantitas atau besarnya ukuran *file*. Adapun ukuran *file* yang diproses ditunjukkan pada tabel sebagai berikut:

Tabel 3. File size data yang diuji (dalam satuan B *byte*)

File 1	File 2	File 3	File 4	File 5	File 6	File 7	File 8	File 9	File 10
10~50	50~100	100~150	150~200	200~250	250~300	300~350	350~450	450~550	550~650
File 11	File 12	File 13	File 14	File 15	File 16	File 17	File 18	File 19	File 20
650~750	750~850	850~950	950~1050	1050~1500	1500~2000	2000~2500	2500~3000	3000~3500	3500~4000
File 21	File 22	File 23	File 24	File 25	File 26	File 27	File 28	File 29	File 30
4000~4500	4500~5 K	5 ~ 10 K	10 ~ 20 K	20 ~ 30 K	30 ~ 40 K	40 ~ 50 K	50 ~ 100 K	100~250 K	250~500 K

Grafik berikut menunjukkan pertumbuhan ukuran *file* yang akan diproses.

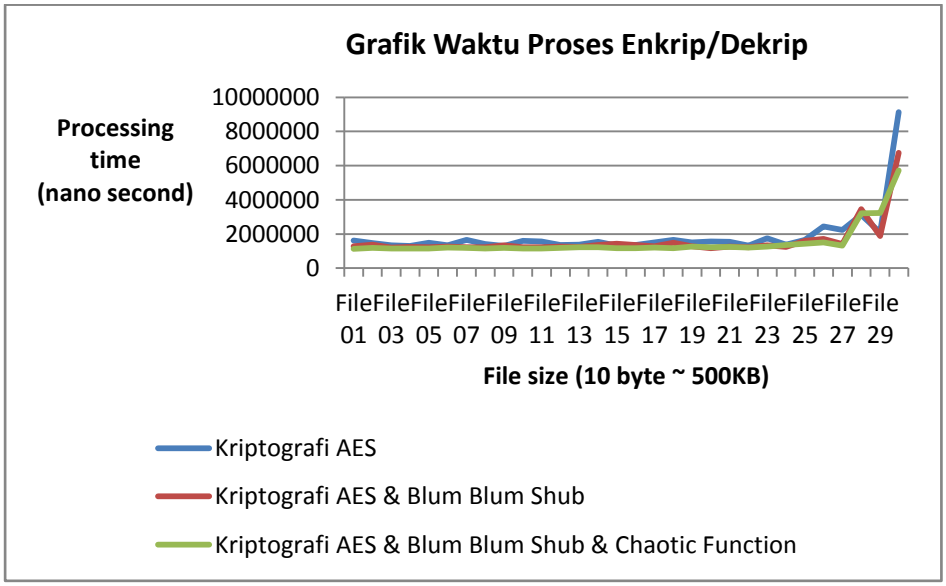


Gambar 10. Grafik Pertumbuhan Ukuran File Data *Plaintext*

a. Pengujian dan Analisis

Pada tahapan ini dijelaskan mengenai proses pengujian pada sistem usulan yang telah dimodifikasi pada bagian pembangkitan kunci. Adapun hasil pengujian tersebut dikelompokkan menjadi dua bagian utama antara lain pertama mengenai pengujian pada performansi waktu proses kriptografi yang melibatkan enkripsi dan dekripsinya. Sedangkan pengujian kedua memfokuskan pada kompleksitas *memory* yang digunakan saat pemrosesan kriptografi dilakukan.

Adapun hasil pengujian pertama dari segi waktu proses kriptografi ditunjukkan dengan grafik sebagai berikut:



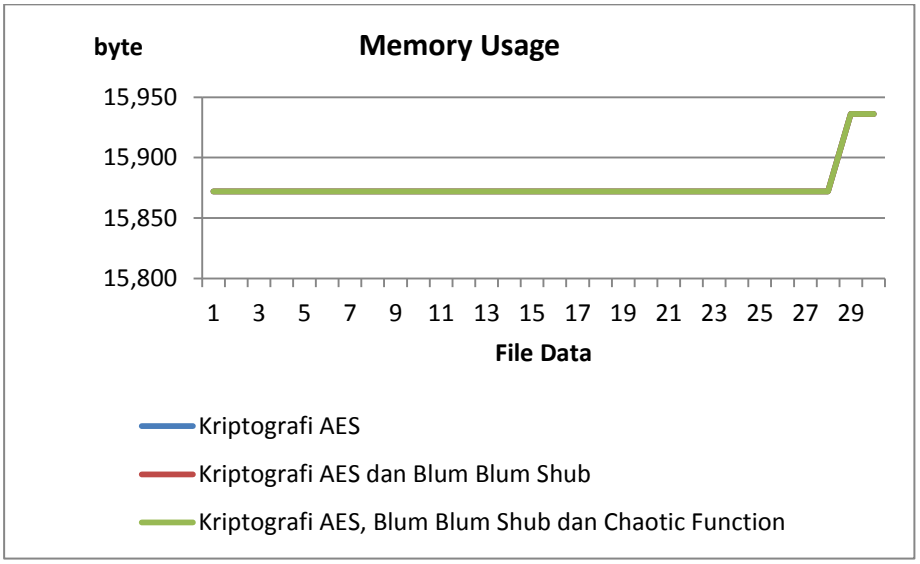
Gambar 11. Grafik Waktu Proses Enkripsi/Dekripsi

Dari grafik tersebut diperoleh analisis bahwa kriptografi dengan modifikasi, yakni dengan menambahkan proses *chaotic function* sangat meminimalisir waktu yang dibutuhkan saat proses. Hal ini terjadi dikarenakan sifat *Chaotic Function* menurut [4] tentang *linierity* pada proses apapun. Sehingga, waktu proses pun bisa dipercepat seiring ukuran data bertambah selain dari segi keamanan meningkat. Adapun tingkat percepatan proses kriptografi dengan modifikasi terhadap kriptografi murni mencapai 20.72122966%. Tingkat percepatan tersebut lebih dari seperlima lebih

cepat dibandingkan dengan kriptografi murni. Hal ini menunjukkan adanya penghematan saat proses komputasi dilakukan.

Hal serupa juga bisa dilihat pada *memory* yang dibutuhkan saat proses berlangsung. Di penelitian ini diperoleh hasil *memory* proses pada metode kriptografi dengan modifikasi menunjukkan kesamaan antara *resourse memory* metode kriptografi tanpa modifikasi dan dengan modifikasi. Pada semua metode kriptografi di penelitian yang dilakukan, meski ada penambahan proses pada pembangkitan *key* pada kriptografi AES namun *resource memory* yang dibutuhkan untuk memproses data pertama sampai dengan data ke-28 adalah sama yakni sebesar 15.872 *bytes* (15 Kilo *byte*), namun pada pemrosesan data ke-29 dan ke-30 menunjukkan adanya penambahan *memory* menjadi 15.936 *bytes*.

Grafik *memory usage* berikut menggambarkan *memory* proses yang dibutuhkan dari data pertama sampai data terakhir.



Gambar 12. Grafik *Memory Usage*

Terlihat pada Gambar 12, yakni semua metode kriptografi memakai *memory* yang sama sebesar 15.872 *bytes* saat memproses data pertama sampai dengan data ke-28. Dan pada pemrosesan data ke-29 dan ke-30. Terjadi peningkatan kebutuhan *memory usage* sampai 15.936 *bytes*, atau bertambah 64 *bytes*. Ukuran data ke-28 yakni 50~100 Kilo *byte* sedangkan ukuran data ke-29 dan ke-30 sebesar 100~250 Kilo *byte* dan 250~500 Kilo *byte*. Persentase peningkatan *memory* ini dapat dikategorikan ke dalam skala sangat minor, yakni hanya 0.40323%. Sementara perubahan ukuran data yang diproses adalah ((100 ~ 250 K) pada data ke-29 dan (50~100 K)) pada data ke-28, atau kurang lebih setara 166.667% lebih banyak. Adapun jumlah memory yang dibutuhkan saat proses kriptografi sebelum dan sesudah dilakukan modifikasi berjumlah sama dikarenakan kriptografi AES bekerja secara *block cipher* dengan susunan matriks di tiap blok yang sudah ditetapkan. Sehingga *byte-byte plaintext* yang masuk ke proses enkripsi atau pun *byte-byte ciphertext* yang masuk ke proses dekripsi diproyeksikan / ditransformasikan ke dalam bentuk matriks-matriks bongkahan yang sama yakni sebesar 16 *byte* di tiap bongkahan. Serta proses yang dilakukan oleh AES adalah tetap melibatkan empat proses utama yang membutuhkan *resource memory* yang tetap. Karenanya, modifikasi yang dilakukan pun menguntungkan dan tetap mempertahankan performansi AES.

4. Kesimpulan

4.1. Kesimpulan

Berdasarkan dari analisis dan pengujian pada penelitian yang telah dilakukan, disimpulkan bahwa:

a. Penambahan proses *random number* Blum-Blum-Shub dan *Chaotic Function* pada *key generating* sebagai bentuk modifikasi pada kriptografi AES dapat mempercepat waktu proses enkripsi dan dekripsi. Jumlah waktu yang dapat dihemat saat proses enkripsi atau dekripsi antara kriptografi AES modifikasi dibandingkan AES murni yakni 20.72122966% lebih cepat.
b. Modifikasi yang dilakukan pada kriptografi AES tidak mengakibatkan kebutuhan *memory* saat proses enkripsi atau dekripsi (*memory usage*) menjadi bertambah, namun tetap sama.

4.2. Saran Untuk Penelitian Berikutnya

a. Pada penelitian berikutnya akan dilakukan pemilihan blok-blok pada matriks *sub byte transformation* untuk proses enkripsi dan dekripsi.
b. Data yang akan dijadikan objek penelitian adalah data berupa multimedia *audio* yang sudah dilakukan pengkodean dan pengkompresan, misal *file format .mp3* dan *.flac*.

Daftar Pustaka:

[1] Li-Chang Lo, Johnny. Bishop, Judith. Eloff, J. H. P. 2010. "*SMSSec: an End-to-End Protocol for Secure SMS*". Computer Science, University of Petrocia, South Africa.
[2] Soleymani, Ali. Md Ali, Zulkarnaen and Nordin, Md Jan. 2012. "*A Survery on Principal Aspects of Secure Image Transmission*". World Academy of Science, Engineering and Technology 66.
[3] Shah, Jolly and Saxena, Vikas. 2011. "*Performance Study on Image Encryption Schemes*". IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No. 1, July 2011. ISSN (Online): 1694-0814. Department of CS & IT, Jaypee Institute of Information Technology. Nodia, Uttar Pradesh 201307, India.
[4] Dharmaadi, I Putu Arya. 2012. Tugas Akhir "*Partially Image Encryption with Combination Method of RC4 Stream Cipher and Chaotic Function*". Bandung, IT Telkom.
[5] J. Daemen and V. Rijmen. 2002. "*The Design of Rijndael: AES – The Advanced Encryption Standard*". Springer Verlag.
[6] Forouzan, Behrouz. A. 2008. "*Cryptography and Network Security*". International Edition. New York. MacGraw-Hill Companies, Inc.
[7] Stinson, D. R. 1995. "*Cryptography: Theory and Practice*". Florida: CRC Press, Inc.
[8] Menezes, A. and Van Oorschot, P. and Vanstone, S. 1997. "Handbook of Applied Cryptography". Florida: CRC Press Inc.
[9] Schneier, B. 1996. "*Applied Cryptography*". John Wiley and Sons Inc.
[10] Junod, Pascal. 1999. "*Cryptography Secure Pseudorandom Bits Generation: The Blum-Blum-Shub Generator*". August.
[11] Sidorenko, Andrey and Schoenmakers, Berry. 2005. "*Concrete Security of The Blum-Blum-Shub Pseudorandom Generator*". Cryptography and Coding: 10th IMA International Conference. Springer-Verlag: Computer Science 3796. Eindhoven University of Technology. P.O. box 513, 5600MB Eindhoven. The Netherlands.
[12] Barmawi, Ari M. and Barja Sanjaya, Muhammad. 2014. "*Cryptosaic: Cryptograpy for Mosaicing*". Bandung, Department of Informatic Engineering, Universitas Telkom.