Enkripsi Selektif Citra Digital dengan *Stream Cipher* Berbasiskan pada Fungsi Chaotik *Logistic Map*

Rinaldi Munir

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung (ITB)
Jl. Ganesha 10, Bandung 40132 Indonesia
email : rinaldi-m@stei.itb.ac.id

ABSTRACT

Enkripsi citra digital merupakan salah satu cara untuk melindungi pengaksesan citra dari pihak yang tidak berhak. Karena citra mengandung volume data yang besar, maka teknik enkripsi selektif bertujuan untuk mengurangi waktu enkripsi/dekripsi. Makalah ini memaparkan algoritma enkripsi selektif citra digital dengan stream cipher yang menggunakan fungsi chaotik logistc map untuk pembangkitan bit-bit kunci penyandian. Enkripsi selektif dilakukan dengan hanya mengenkripsi bit MSB (most significant bit) dari setiap pixel citra. Hasil eksperimen memperlihatkan algoritma yang diusulkan dapat mengenkripsi sembarang citra (citra grayscale atau citra berwarna) dengan baik, cepat, dan terbukti aman dari exhaustive attack, sehingga ia dapat digunakan untuk melindungi citra digital dari pengaksesan yang ilegal.

Key-words

Enkripsi citra, selektif, stream cipher, MSB, chaos

1. Pendahuluan

Citra digital merupakan salah satu bentuk informasi yang banyak digunakan secara luas. Citra digital juga merupakan penyusun video digital, sebab sebuah video terdiri dari ratusan hingga ribuan *frame* citra diam (*still images*).

Citra yang bersifat privat atau citra yang mengandung informasi rahasia perlu dilindungi dari pengakasesan oleh pihak-pihak yang yang tidak memiliki otoritas (*anauthorized*). Enkripsi citra merupakan teknik yang umum dilakukan untuk merahasiakan informasi di dalam citra dari pihak yang tidak berhak mengaksesnya.

Algoritma enkripsi tradisionil yang lazim digunakan untuk pesan teks seperti DES, AES, RSA, IDEA, dan lain-lain tidak cocok digunakan untuk mengenkripsi citra. Hal ini disebabkan karena data citra (atau video) umumnya bervolume relatif sangat besar dibandingkan dengan data teks, sehingga proses enkripsi citra dengan algoritma-algoritma tradisionil tersebut memerlukan komputasi lebih lama [1]. Oleh karena itu, enkripsi citra perlu dilakukan dengan metode yang lebih efisien. Hingga saat ini sudah banyak riset yang secara intensif mengembangkan algorima khusus untuk enkripsi citra digital. Beberapa algoritma enkripsi citra digital dapat dibaca di dalam [1-2], sedangkan performansi beberapa algoritma enkripsi citra dapat ditemukan di dalam [3].

Agar proses enkripsi/dekripsi citra berlangsung cepat, maka enkripsi citra dilakukan secara selektif. Enkripsi selektif artinya hanya mengenkripsi sebagian elemen citra namun efeknya terhadap keseluruhan citra. Enkripsi selektif dapat dilakukan baik dalam ranah spasial, ranah frekuensi, atau campuran keduanya. Beberapa teknik enkripsi selektif dapat ditemukan di dalam [7-8].

Selain faktor waktu, untuk mendapatkan teknik enkripsi yang memenuhi aspek sekuriti banyak para ilmuwan yang telah menerapkan sistem *chaos* di dalam kriptografi [6]. Karakteristik fungsi *chaos* adalah sensitivitasnya terhadap perubahan kecil parameter nilai awal (*sensitive dependence on initial condition*). Sensitivitas ini berarti bahwa perbedaan kecil pada nilai awal fungsi --setelah fungsi diiterasi sejumlah kali-- menghasilkan perbedaan yang sangat besar pada nilai fungsinya. Karakteristik ini penting di dalam keamanan dan bersesuaian dengan prinsip *diffusion* yang dikemukakan oleh Shannon [6].

Salah satu fungsi *chaos* adalah persamaan logistik (*logistic map*). Komputasi *logistic map* relatif sederhana sehingga ia cocok digunakan untuk aplikasi enkripsi yang membutuhkan waktu proses yang cepat namun tetap aman secara kriptografis.

Makalah ini menyajikan sebuah usulan algoritma *stream cipher* untuk enkripsi selektif citra digital dengan mengaplikasikan fungsi *logistic map*. Fungsi *logistic map* berperan untuk membangkitkan bit-bit kunci (*keystream*) yang kemudian dienkripsi dengan data citra. Enkripsi selektif dilakukan pada ranah spasial dengan cara memilih bit tertentu saja dari *pixel* citra yang kemudian dienkripsi dengan *keystream* yang dibangkitkan dari fungsi chaotik *logistic map*.

2. Logistic Map

Salah satu fungsi *chaos* sederhana adalah persamaan logistik (*logistic map*) yang biasa dipakai di dalam sistem ekologi untuk mensimulasikan pertumbuhan spesies di dalam ekosistem. Persamaan logistik dinyatakan dalam bentuk iteratif sebagai

$$x_{i+1} = r\, x_i\, (1 - x_i) \qquad (1)$$

dengan $0 \le x_i \le 1$, $i = 0, 1, 2, ...$ Nilai awal (*seed*) persamaan iterasi adalah x_0 yang berperan sebagai kunci rahasia. Konstanta r menyatakan laju pertumbuhan fungsi dan $0 \le r \le 4$. Persamaan ini mulai menuju sifat chaotik ketika $r > 3.75$, dan ketika $r = 4$, fungsi secara total menjadi *chaos*.

Sebagai contoh, jika kita menggunakan $r = 4.0$ dan nilai awal $x_0 = 0.456$, maka kita memperoleh 100 bilangan acak: $x_1 = 4.0x_0(1 - x_0) = 0.992256$, $x_2 = 4.0x_1(1 - x_1) = 0.030736$, $x_3 = 4.0x_2(1 - x_2) = 0.119166$, … , $x_{99} = 4.0x_{98}(1 - x_{98}) = 0.914379$, $x_{100} = 4.0x_{99}(1 - x_{99}) = 0.313162$

Nilai-nilai acak yang dihasilkan di dalam barisan chaotik tersebut nilainya berada di antara 0 dan 1. Dengan mengubah nilai x_0 sedikit saja sebesar Δ (catatan: Δ adalah bilangan yang sangat kecil, misalnya 10^{-10}), barisan bilangan acak yang dihasilkan – setelah diiterasi beberapa kali-- berbeda signifikan dengan barisan sebelumnya.

3. *Stream Cipher*

Stream cipher adalah algoritma kriptografi yang mengenkripsi plainteks menjadi chiperteks bit per bit (atau *byte* per *byte*). *Stream cipher* pertama kali diperkenalkan oleh Vernam melalui algoritmanya yang dikenal dengan nama *Vernam Cipher*. Jika enkripsi dan dekripsi dilakukan bit per bit, maka persamaan enkripsi adalah

$$c_i = p_i \oplus k_i \qquad (2)$$

sedangkan dekripsi adalah

$$p_i = c_i \oplus k_i \qquad (3)$$

yang dalam hal ini p_i adalah bit plainteks ke-i, k_i adalah bit kunci, $\oplus$ adalah operator XOR dan c_i adalah bit cipherteks ke-i.

Barisan bit-bit kunci dihasilkan dari sebuah sebuah pembangkit yang dinamakan pembangkit arus-kunci (*keystream generator*). Arus-kunci (sering dinamakan juga *running key*) di-*XOR*-kan dengan arus-plainteks, $p_1, p_2, \ldots, p_i$, untuk menghasilkan arus-cipherteks $c_1, c_2, \ldots, c_i$ dengan persamaan (1). Di sisi penerima arus-kunci yang sama dibangkitkan lalu di-XOR-kan dengan cipherteks untuk menghasilkan plainteks semula dengan persamaan (3).

4. *Most Significant Bit* (MSB)

Pixel-pixel citra di dalam struktur citra *bitmap* berukuran sejumlah *byte*. Pada citra 8-bit satu *pixel* berukuran 1 *byte* (8 bit), sedangkan pada citra 24-bit satu *pixel* berukuran 3 byte (24 bit).

Di dalam susunan *byte* terdapat bit yang paling tidak berarti (*least significant bit* atau *LSB*) dan bit yang paling berarti (*most significant bit* atau *MSB*). Contohnya pada *byte* 10111010, bit *LSB* adalah bit yang terletak paling kanan yaitu 0, sedangkan bit *MSB* adalah bit paling kiri yaitu 1.

Pengubahan 1 bit *LSB* tidak mempengaruhi nilai *byte* secara signifikan, namun pengubahan 1 bit *MSB* dapat mengubah nilai *byte* secara signifikan. Misalnya 10111010 jika diubah bit *MSB*-nya dari 1 menjadi 0 sehingga susunan *byte* menjadi 00111010, maka nilai *byte*-nya berubah dari 186 menjadi 122. Secara visual efek perubahan bit-bit *MSB* pada seluruh *pixel* citra menyebabkan citra tersebut menjadi "rusak" sehingga citra tidak dapat dipersepsi dengan jelas. Inilah yang menjadi dasar enkripsi citra yang bertujuan membuat citra tidak dapat dikenali lagi karena sudah berubah menjadi bentuk yang tidak jelas.

5. Usulan Algoritma Enkripsi Selektif

Pengubahan bit *MSB* menjadi dasar enkripsi selektif citra digital. Dengan hanya mengenkripsi bit *MSB* maka proses enkripsi menjadi lebih efisien, sebab tidak semua data citra dienkripsi dengan *stream cipher*.

Misalkan sebuah citra *grayscale* berukuran 256 × 256 *pixel* dan setiap *pixel* berukuran 1 *byte*, maka terdapat 256 × 256 × 8 bit = 524288 bit yang perlu dienkripsi dengan persamaan (2). Tetapi, jika hanya mengenkripsi bit-bit *MSB* saja, kita hanya perlu mengenkripsi 65536 bit atau hanya 12,5% dari volume citra keseluruhan. Jadi, proses komputasi dapat dihemat sebesar 87,5% dibandingkan bila mengenkripsi seluruh bit di dalam *pixel-pixel* citra.

Untuk citra berwarna dimana setiap *pixel* disusun oleh komponen warna *R* (*red*), *G* (*green*), dan *B* (*blue*), maka pengubahan bit *MSB* dilakukan pada setiap komponen warna. Pada citra 24-bit (*true image*), setiap *pixel* panjangnya 3 *byte* (masing-masing komponen warna *RGB* berukuran 1 *byte*), sehingga pada setiap *pixel* dilakukan 3 kali modifikasi bit *MSB*.

Untuk mengenkripsi bit-bit *MSB* dengan *stream cipher* diperlukan barisan bit kunci rahasia yang menjadi *keystream*. Bit-bit ini dibangkitkan dari fungsi *logistic map* dengan mengiterasikan persamaan (1). Nilai awal *logistic map* (x_0) berperan sebagai kunci enkripsi. Namun, keluaran dari *logistic map* tidak dapat langsung di-XOR-kan dengan bit-bit plainteks karena masih berbentuk bilangan riil antara 0 dan 1.

Agar barisan nilai chaotik dapat dipakai untuk enkripsi dan dekripsi dengan *stream cipher*, maka nilai-nilai *chaos* tersebut dikonversi ke nilai *integer*. Selanjutnya, 1 bit *LSB* diekstraksi dari nilai *integer* tersebut. Bit *LSB* inilah yang menjadi bit kunci untuk enkripsi.

Ada beberapa teknik konversi dari nilai-nilai *chaotic* menjadi nilai *integer*. Teknik yang umum misalnya mengambil 3 angka terakhir pada bagian mantissa bilangan riil. Sebagai contoh, dari nilai chaotic 0.024568 diambil 3 angka terakhir dari bagian mantissanya yaitu 568.

Di dalam makalah ini, konversi nilai *chaos* ke *integer* dilakukan dengan menggunakan fungsi pemotongan yang diusulkan di dalam [5]. Caranya, 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. Secara matematis, nilai *chaos x* dikonversi ke *integer* dengan menggunakan persamaan berikut:

$$T(x, size) = \left\| x * 10^{count} \right\|, x \neq 0 \qquad (4)$$

yang dalam hal ini *count* dimulai dari 1 dan bertambah 1 hingga $x * 10^{count} > 10^{size-1}$. Hasilnya kemudian diambil bagian *integer* saja (dilambangkan dengan pasangan garis ganda pada persamaan 4). Sebagai contoh, misalkan x = 0.003176501 dan *size* = 4, maka dimulai dari *count* = 1 sampai *count* = 6 diperoleh

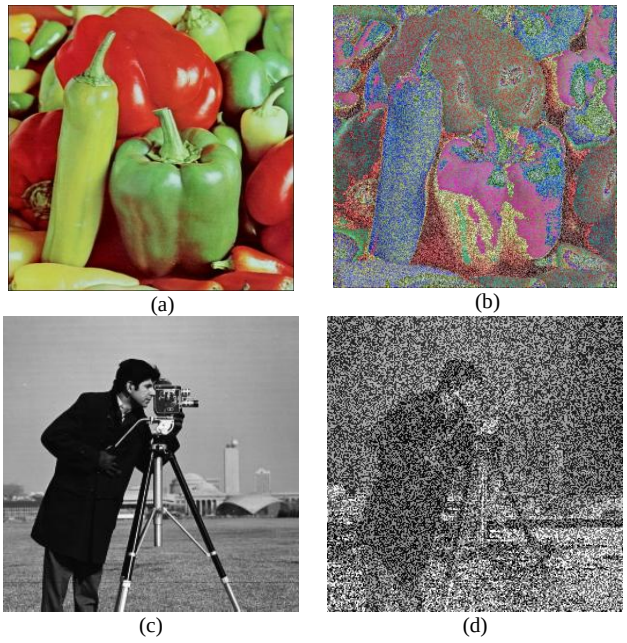
$$0.003176501 * 10^6 = 3176.501 > 10^3$$

kemudian ambil bagian *integer*-nya dengan

$$\|3176.501\| = 3176$$

Bit *LSB* dari representasi bit 3176 adalah 0 (perhatikanlah bahwa 3176 bilangan genap, maka bit LSB-nya pasti 0). Jadi, kita memperoleh sebuah bit kunci dari nilai chaotik 0.003176501 yaitu 0. Prosedur yang sama dilakukan untuk nilai-nilai chaotik lainnya.

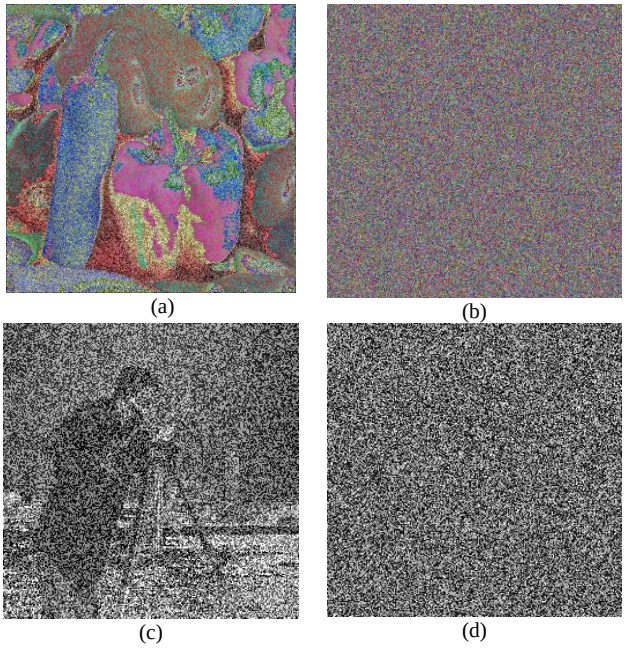
Bit-bit kunci yang dihasilkan dari *logistic map* selanjutnya di-XOR-kan dengan bit-bit *MSB* dari setiap byte *pixel*. Hasil pengubahan bit *MSB* sebenarnya belum terlalu aman, sebab citra keluaran masih memperlihatkan persepsi bayangan citra semula, seperti diperlihatkan pada enkripsi citra *peppers* dan citra *cameraman* pada Gambar 1. Tepi-tepi di dalam citra masih terlihat sehingga objek di dalam citra masih dapat dipersepsi secara visual meskipun gambar sudah tampak rusak.



(a) (b) (c) (d)

Gambar 1. (a) dan (c) citra semula, (b) dan (d) citra setelah pengubahan bit-bit MSB

Oleh karena itu, prosedur tambahan diperlukan untuk membuat gambar hasil enkripsi *stream cipher* menjadi acak sempurna. Prosedur tersebut mengacak *pixel-pixel* hasil proses sebelumnya dengan teknik permutasi, sehingga susunannya menjadi teraduk acak seperti diperlihatkan pada Gambar 2.



(a) (b) (c) (d)

Gambar 2. (a) dan (c) citra hasil pengubahan bit-bit MSB, (b) dan (d) citra dari (a) dan (c) masing-masing setelah pengacakan dengan teknik permutasi

Proses dekripsi merupakan kebalikan dari proses enkripsi. Mula-mula citra yang terenkripsi dipermutasi kembali dengan matriks permutasi yang sama, selanjutnya bit-bit *MSB* diekstraksi dari *pixel-pixel* citra kemudian di-XOR-kan dengan bit-bit kunci melalui persamaan (3). Citra hasil dekripsi tepat sama seperti citra semula.

Algoritma enkripsi dan dekripsi selektif yang diuraikan di atas dapat diringkas dalam urutan langkah-langkah (*step*) sebagai berikut:

(a) Enkripsi
Masukan: citra, x_0
Keluaran: citra terenkripsi

Step 1: Baca *pixel-pixel* citra dan simpan di dalam sebuah matriks. Jika citra tersebut citra 24-bit, maka setiap komponen *R*, *G*, dan *B* disimpan di dalam tiga matriks berbeda.
Step 2: Ekstraksi bit-bit *MSB* dari setiap *byte pixel* citra, misalkan bit-bit tersebut adalah $p_1, p_2, \ldots, p_n$.
Step 3: Iterasikan *logistic map* dengan nilai awal x_0 (kunci rahasia). Konversikan setiap nilai chaotic menjadi *integer* dengan persamaan (4). Ekstraksi 1 bit *LSB* dari setiap nilai *integer* tersebut, misalkan bit-bit *LSB* yang dihasilkan adalah $k_1, k_2, \ldots, k_n$.
Step 4: Enkripsi setiap p_i dengan k_i menggunakan persamaan (2). Hasil enkripsi adalah $c_1, c_2, \ldots, c_n$. Letakkan kembali bit-bit hasil enkripsi pada posisi *MSB pixel* semula.
Step 5: Acak *pixel-pix*el citra dengan sebuah matriks permutasi rahasia. Hasil pengacakan ini adalah citra terenkripsi.

(b) Dekripsi
Masukan: citra terenkripsi, x_0
Keluaran: citra semula

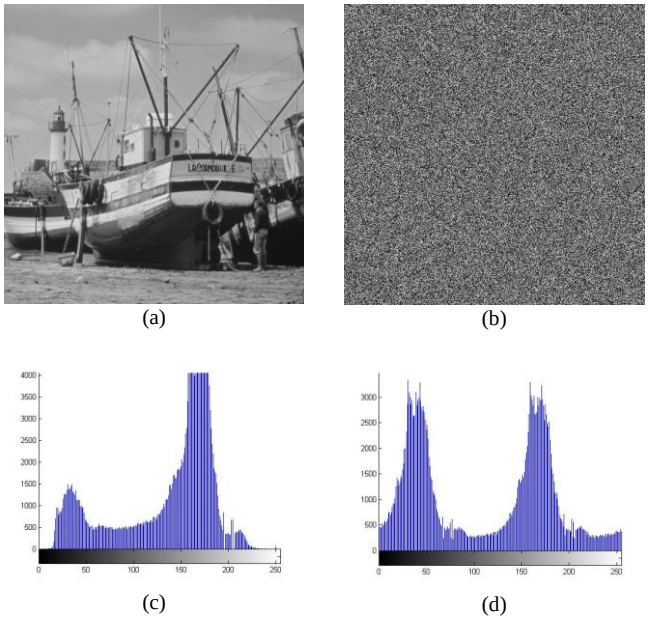
Step 1: Baca *pixel-pixel* citra dan simpan di dalam sebuah matriks. Jika citra tersebut citra 24-bit, maka setiap komponen *R*, *G*, dan *B* disimpan di dalam tiga matriks berbeda.
Step 5: Lakukan *invers* permutasi, yaitu mengembalikan susunan *pixel-pixel* yang teracak menjadi susunan semula.
Step 2: Ekstraksi bit-bit *MSB* dari setiap *byte pixel* citra, misalkan bit-bit tersebut adalah $c_1, c_2, \ldots, c_n$.
Step 3: Iterasikan *logistic map* dengan nilai awal x_0 (kunci rahasia). Konversikan setiap nilai chaotic menjadi *integer* dengan persamaan (4). Ekstraksi 1 bit *LSB* dari setiap nilai *integer* tersebut, misalkan bit-bit *LSB* yang dihasilkan adalah $k_1, k_2, \ldots, k_n$.
Step 4: Dekripsi setiap p_i dengan k_i menggunakan persamaan (2). Hasil denripsi adalah $p_1, p_2, \ldots, p_n$. Letakkan kembali bit hasil enkripsi pada posisi *MSB pixel* semula.

5. Hasil-Hasil Eksperimen dan Pembahasan

Algoritma enkripsi selektif yang diusulkan ini diprogram dengan kakas Matlab. Pengujian dilakukan pada sejumlah citra, baik citra *grayscale* maupun citra berwarna. Semua citra dapat dienkripsi menjadi *encrypted-image* dan dapat didekripsi kembali menjadi citra semula. Parameter nilai yang digunakan adalah $x_0 = 0.5743$, *size* = 4. Matriks permutasi dibangkitkan dengan fungsi `randperm` yang sudah tersedia di dalam Matlab. Umpan (*seed*) fungsi `randperm` adalah jumlah semua barisan chaotik yang dibangkitkan dengan nilai awal x_0. Diantara citra-citra uji tersebut, hasil-hasil eksperimen terhadap citra *boat* dan *Lena* didiskusikan di bawah ini.

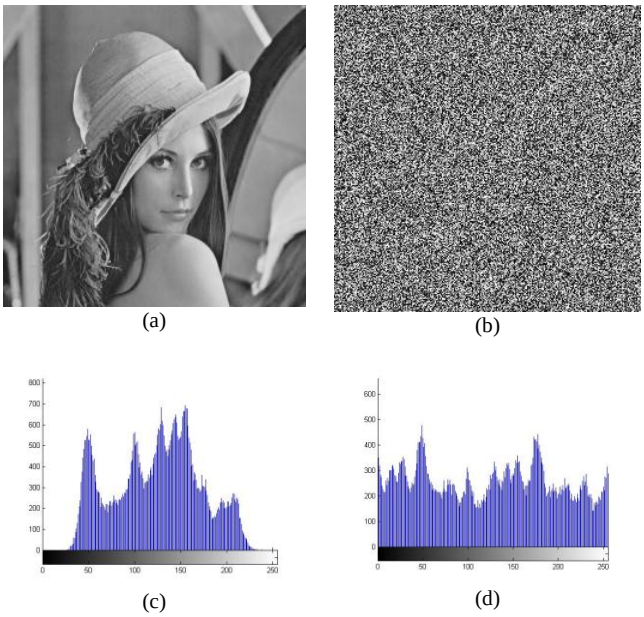
(a) Analisis histogram citra

Citra *boat* orijinal dan histogramnya serta citra terenkripsi dan histogramnya diperlihatkan pada Gambar 3. Terlihat bahwa citra terenkripsi sudah tidak dapat dikenali lagi, demikian pula histogramnya juga terlihat tidak sama. Histogram citra terenkripsi terlihat berimbang pada kedua sisi (kiri dan kanan). Histogram bergeser ke kiri karena nilai-nilai *pixel* yang tinggi banyak yang berubah menjadi lebih kecil akibat perubahan bit *MSB* dari 1 menjadi 0, demikian juga sebaliknya. Histogram yang terlihat seimbang ini akan menyulitkan kriptanalis dalam melakukan analisis statistik untuk menemukan kunci enkripsi karena jumlah *pixel* yang bernilai tinggi dan jumlah *pixel* yang bernilai rendah seimbang banyaknya.



Gambar 3. (a) citra *boat* orijinal, (b) citra *boat* terenkripsi, (c) histogram citra semula, (d) histogram citra terenkripsi

Hasil yang serupa terlihat pada citra *Lena* yang memperlihatkan histogram citra hasil enkripsi terlihat seimbang antara kiri dan kanan sehingga menyulitkan kriptanalisis melakukan analisis statistik (lihat Gambar 4).

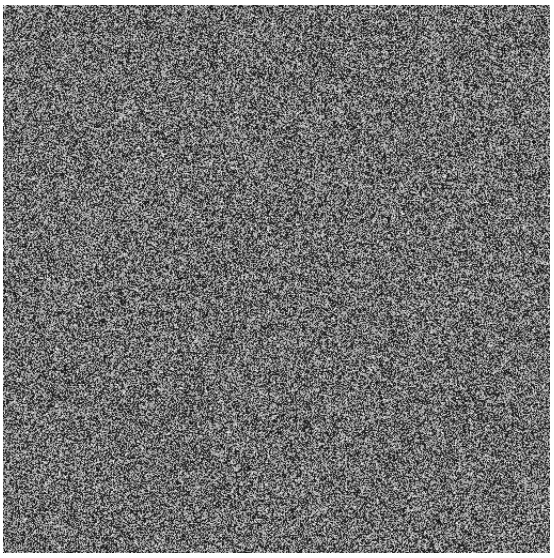


Gambar 4. (a) citra *Lena* orijinal, (b) citra *Lena* terenkripsi, (c) histogram citra semula, (d) histogram citra terenkripsi

(b) Analisis perubahan nilai awal *logistic map*

Sifat fungsi *chaos* yang terpenting adalah kepekaaannya terhadap perubahan kecil nilai awal iterasi. Jika nilai awal fungsi diubah sedikit saja, nilai-nilai acak yang dihasilkannya berubah secara signifikan setelah *logistic map* diiterasi sejumlah kali. Hal ini sangat bagus untuk keamanan sebab lawan yang mencoba menemukan kunci akan frustasi karena perbedaan nilai awal fungsi *chaos* yang sekecil apapun tidak menghasilkan citra semula. Nilai awal *logistic map* (yaitu x_0) berlaku sebagai kunci rahasia untuk enkripsi dan dekripsi.

Pada eksperimen ini nilai awal *logistic map* diubah sebesar Δ sehingga menjadi $x_0 + \Delta$, kemudian citra didekripsi dengan kunci $x_0 + \Delta$. Misalkan $\Delta = 10^{-10}$ sehingga nilai awal *logistic map* adalah 0.5734000001. Gambar 5 memperlihatkan citra hasil dekripsi yang ternyata tetap teracak. Perubahan kecil nilai awal *chaos* membuat nilai chaotik yang dihasilkan berbeda signifikan. Karena nilai-nilai ini dipakai sebagai bit-bit kunci dan sebagai umpan untuk pembangkitan permutasi (yang berkoresponden dengan posisi pengacakan), maka resultan dari keduanya memberikan hasil dekripsi yang tidak benar dan tetap teracak.



Gambar 5. Citra *boat* hasil dekripsi dengan pengubahan nilai awal *logistic map* sebesar $\Delta = 10^{-10}$. Hasil dekripsi tetap acak.

Hasil eksperimen ini membuktikan bahwa karakteristik *chaos* yang sensitif terhadap nilai awal memang memberikan keamanan yang bagus dari serangan *exhaustive attack*. Eksperimen ini juga menyiratkan bahwa perubahan sangat kecil pada kunci menyebabkan hasil dekripsi salah, apalagi jika kunci dekripsi yang diberikan berbeda jauh nilainya dengan kunci yang sebenarnya.

6. Kesimpulan dan *Future Research*

Di dalam makalah ini telah dipresentasikan algoritma enkripsi selektif citra digital berbasis *chaos*. Enkripsi selektif dilakukan pada hanya pada bit-bit *MSB* dari *pixel-pixel* citra. Fungsi *chaos* yang digunakan adalah *logistic map*. Nilai awal *logistic map* berperan sebagai kunci rahasia.

Hasil eksperimen memperlihatkan bahwa algoritma enkripsi ini dapat mengenkripsi citra dengan baik dan mendekripsinya kembali tepat sama seperti citra semula. Keamanan algoritma telah ditunjukkan dari analisis histogram yang terlihat seimbang antara *pixel-pixel* bernilai rendah dengan *pixel-pixel* yang bernilai tinggi, sehingga menyulitkan kriptanalisis dengan analisis statistik. Eksperimen perubahan parameter nilai *chaos* memperlihatkan bahwa algoritma ini aman dari serangan *exhaustive attack*.

Penelitian untuk *future research* dapat dikembangkan untuk mencari formula agar histogram citra hasil enkripsi terlihat *flat* (datar). Histogram yang *flat* merupakan indikasi tidak adanya hubungan statistik antara *pixel-pixel* citra orijinal dengan *pixel-pixel* citra terenkripsi, sehingga serangan dengan menggunakan analisis statistik menjadi sangat sulit dilakukan.

REFERENSI

[1] Lala Krikor, Sama Baba, Thawar Arif, Zyad Shaaban, *Image Encryption Using DCT and Stream Cipher*, European Journal of Scientific Research, Vol. 32, No. 1 (2009), pp 47-57

[2] Ibrahim S I Abuhaiba, Maaly A S Hassan, *Image Encryption Using Differential Evolution Approach*, Signal and Image Processing: An International Joernal (SIPIJ) Vol 2, No. 1, March 2011.

[3] Jolly Shah, Vikas Saxena, *Performance Study on Image Encryption Schemes*, IJSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011.

[4] Jonathan M.Blackledge, Musheer Ahmad, Omar Farooq, *Chaotic Image Encryption Algorithm Based on Frequency Domain Scrambling*, School of Electrical Engineering Systems, Dublin Institute of Technology, 2010

[5] James Lampton, *Chaos Cryptography: Protecting data Using Chaos*, Missisippi School for Mathematics and Science.

[6] Bruce Schneier, *Applied Cryptography*, John Wiley & Sons, 1996.

[7] Nidhi S Kulkarni, Balasubramanian, Indra Gupta, *Selective Encryption of Multimedia Images*, Proceeding of XXXII National Systems Conference NSC 2008.

[8] Roman Pfarrhofer, Andreas Uhl, *Selective Encryption Using JBIG*, Lecture Notes in Computer Science, 2005, Volume 3677/2005, 98-107.