Pemberian Hiddentext Palsu pada Steganografi Visual

Okaswara Perkasa (13510051)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
okaswara@students.itb.ac.id

***Abstrak* — Makalah ini membahas mengenai algoritma baru steganografi visual yang memanfaatkan hiddetext palsu. Hiddentext palsu ini dapat digunakan untuk melindungi hiddentext yang sebenarnya. Makalah ini membahas salah satu contoh algoritma steganografi sederhana yang mengimplementasikan hiddentext palsu. Setelah itu, berbagai uji coba dilakukan, seperti batas informasi yang dapat disisipkan, serta melakukan steganalisis pada citra yang sudah disisipi dengan algoritma yang telah dirancang.**

***Kata kunci* — steganography, hiddentext palsu, steganalisis**

I. PENDAHULUAN

Secara umum, fokus utama dari pengembangan algoritma steganografi adalah membuat *stegotext* tidak terlihat mencurigakan oleh pihak ketiga. Jika dalam steganografi visual, ini berarti pesan tersebut tidak dapat dideteksi keberadaannya, baik secara perspektif maupun matematis.

Salah satu hal yang biasa dilakukan adalah dengan memfokuskan agar kualitas *covertext* tidak banyak berubah pada saat disisipi pesan. Namun, pada umumnya, *covertext* tidak terlalu bisa dipertahankan dengan baik kualitasnya, sehingga tetap berpotensi untuk menimbulkan kecurigaan.

Ketika suatu citra dinilai memiliki pesan tersembunyi didalamnya, steganalis akan langsung melakukan inspeksi terhadap citra dengan beberapa metode ekstraksi ataupun analisis yang umum. Jika suatu *stegotext* sudah dicurigai, maka besar kemungkinan pesan tersembunyi didalamnya akan terbongkar. Atau jikalau tidak dapat dibongkar, maka pemilik pesan dapat dipaksa untuk melakukan ekstraksi. Ini dapat dilakukan jika memang terdapat hukum yang mengatur tentang hal tersebut.

Namun, sebenarnya terdapat suatu metode yang dapat memberikan perlindungan tambahan pada *stegotext* yang telah dicurigai, yaitu dengan menambahkan suatu *hiddentext* palsu. *Hiddentext* palsu ini dapat berisi pesan yang tidak signifikan, menyangkal, ataupun tidak berhubungan sama sekali dengan *hiddentext* yang asli. Penyisipannya juga dibuat lebih sederhana, agar *hiddentext* palsu tersebut lebih mudah ditemukan pada saat melakukan steganalisis.

Saat steganalis melakukan analisis pada citra, atau saat pemilik pesan dipaksa untuk melakukan ekstraksi, mereka akan menemukan *hiddentext* palsu tersebut. Penemuan *hiddentext* palsu ini seolah-olah dapat dijadikan "bukti" bahwa citra tersebut ternyata tidak mengandung pesan yang "berbahaya", sehingga inspeksi terhadap citra dapat dihentikan.

Pesan yang asli menjadi tidak tersentuh.

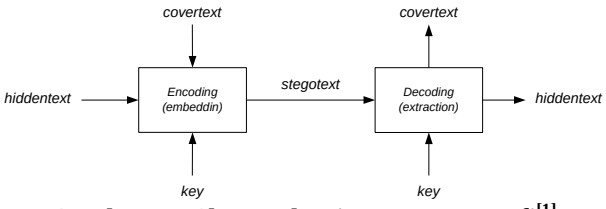
II. DASAR TEORI

A. Steganografi (Visual)

Steganografi adalah teknik untuk menyembunyikan pesan, tanpa menimbulkan kecurigaan akan keberadaan pesan tersebut oleh pihak ketiga. Tidak seperti kriptografi, fokus utama pada steganografi adalah agar pesan tersembunyi tidak terlihat dari luar.

Secara garis besar, terdapat sejumlah istilah yang biasa dipakai pada algoritma-algoritma steganografi[1], yakni:

1. *Hiddentext*: Pesan yang akan disembunyikan
2. *Covertext*: Media dimana pesan bersembunyi.
3. Stegokey: Key yang digunakan pada peyisipan ataupun pengambilan pesan
4. *Stegotext*: *Covertext* yang sudah mengandung *hiddentext*.



Gambar 1 – Skema algoritma steganografi[1]

Proses penyisipan *hiddentext* pada *covertext* sendiri disebut dengan *embedding.* Proses sebaliknya, mengembalikan *hiddentext* dari suatu *stegotext,* disebut dengan *extraction.*

Algoritma steganografi yang baik memiliki tiga kriteria[1], yakni:

1. *Imperceptible*: Keberadaan *hiddentext* tidak dapat dipersepsikan
2. *Fidelity*: *Covertext* tidak banyak berubah setelah dilakukan *embedding*.
3. *Recovery*: *Hiddentext* dapat dikembalikan dari *covertext.*

Steganografi visual adalah salah satu bentuk steganografi yang menggunakan media citra sebagai *covertext*.

Ada sejumlah cara yang dapat dilakukan untuk melakukan *embedding* informasi di dalam citra. Salah satunya adalah dengan menyimpan informasi tersebut pada LSb (*least significant bit*) untuk masing-masing channel warna yang ada pada setiap pixel[1].

B. PSNR

PSNR (*Peek Signal-to-Noise Ratio*) adalah salah satu cara untuk mengukur kualitas citra yang dihasilkan dari suatu transformasi, i.e. seberapa mirip citra tersebut dengan citra saat sebelum ditransformasi.

PSNR sendiri dihitung dengan rumus[2]:

$$PSNR = 20 \times \log_{10}\left(\frac{256}{rms}\right)$$

Dengan rms dihitung dengan rumus sebagai berikut:

$$rms = \sqrt{\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(I_{ij} - \hat{I}_{ij})^2}$$

Nilai PSNR > 30 pada suatu citra memiliki arti bahwa citra tersebut masih memiliki kualitas yang baik. Sedangkan nilai PSNR < 30 menandakan bahwa citra sudah cukup jauh berubah dari aslinya[2].

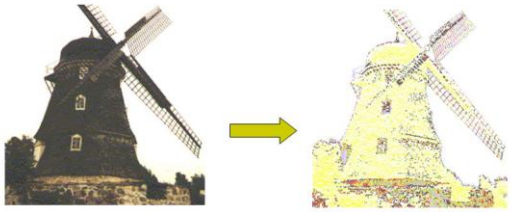
C. Enhanced LSb[4]

Enhanced LSb adalah salah satu metode steganalisis visual yang memanfaatkan indera pengelihatan manusia.

Hal ini dilakukan dengan mengubah nilai seluruh bit warna pada masing-masing channel warna dengan nilai LSb.

BLUE	GREEN	RED
1010010**1**	1001110**0**	1110011**1**

BLUE	GREEN	RED
11111111	**00000000**	**11111111**



Gambar 2 – Illustrasi *Enhanced LSb*[1]

Dengan melakukan *enhanced LSb*, LSb pada citra menjadi signifikan, sehingga bila terdapat pesan tersembunyi pada LSb, akan langsung terlihat adanya titik-titik warna acak pada citra.

Metode ini efektif untuk citra yang memiliki kontras yang tinggi, atau citra yang memiliki warna yang berbeda antara latar belakang dan objek pada citra. Misalnya pada logo atau illustrasi. Untuk citra dengan kontras rendah, seperti foto, metode ini kurang efektif karena kesulitan membedakan antara LSb yang dihasilkan oleh penyisipan pesan, dan LSb yang memang sudah ada pada *covertext*.[1]

D. Faketext

Faketext adalah istilah yang merujuk pada pesan palsu yang disisipkan pada suatu *covertext*.

Perhatikan bahwa istilah ini hanya berlaku pada makalah ini saja. Pembuatan istilah ini bertujuan untuk memudahkan dalam mereferensi konsep "*hiddentext* palsu" pada makalah.

IV. Deskripsi Algoritma

Sesuai yang telah dijabarkan sebelumnya, algoritma steganografi yang dirancang harus memiliki karakteristik sebagai berikut:

1. Dapat menyisipkan *faketext*, disamping dengan *hiddentext*.
2. Pada saat dilakukan steganalisis, *faketext* memiliki kemungkinan yang lebih besar untuk ditemukan terlebih dahulu.
3. Memilki ketiga kriteria algoritma steganografi yang baik, yakni *imperceptible*, *fidelity*, dan *recovery*.

Untuk memenuhi ketiga karakteristik di atas, terdapat suatu algoritma sederhana yang dapat digunakan, yakni dengan menyimpan bit-bit pada *faketext* dan *hiddentext* pada LSb pixel-pixel pada citra secara independen.

Gambar 3 – Illustrasi alokasi bit pada gambar

Gambar 3 menunjukkan posisi alokasi bit-bit *faketext* dan *hiddentext* pada *covertext*. *Faketext* memilki alokasi yang berwarna biru, sedangkan *hiddentext* yang berwarna merah.

Secara umum, *faketext* disimpan dengan sederhana, yakni dituliskan secara sekuensial pada *covertext*. Sedangkan *hiddentext* disembunyikan dengan cara yang lebih rumit, yakni secara acak pada *covertext*.

Pada implementasinya, sebelum disisipkan, *hiddentext* terlebih dahulu diberikan suatu *super-encryption*, atau enkripsi yang terdiri dari sejumlah *substitution cipher* dan

transposition cipher. Implementasi yang dibuat menggunakan sepuluh kali operasi *substitution* dan *transposition* berturut-turut, untuk memastikan *hiddentext* tidak dapat terbaca secara langsung. Penyisipan pada *covertext* menggunakan suatu algoritma PRNG, agar posisi penyisipan dapat di-*generate* ulang pada saat ekstraksi.

Terdapat header untuk *hiddentext* yang digunakan untuk menyimpan panjang *hiddentext*. Header ini juga ikut dienkripsikan.

Sebaliknya, *faketext* langsung dituliskan pada *covertext*, ditulis dari bagian kiri atas citra, dan tanpa menggunakan enkripsi apapun. Ini dilakukan agar *faketext* menjadi mudah ditemukan pada saat steganalisis.

Faketext juga diberikan suatu *header* untuk menandakan panjang *faketext* yang tersimpan pada citra. *Header* ini juga tidak dienkripsi, dan diletakkan pada bagian depan *faketext*.

Selain itu, untuk memberikan kebebasan pada saat melakukan uji coba, bit yang digunakan oleh algoritma tidak hanya terbatas pada LSb saja. Seluruh posisi bit dapat dimanfaatkan untuk penyisipan data.

Namun, *embedding* tetap diprioritaskan pada bit-bit yang memiliki significance yang kecil. Posisi bit yang lebih *significant* akan digunakan ketika seluruh bit di posisi lebih dibawah sudah terisi. Hal ini perlu dilakukan untuk menjaga *fidelity* dari *covertext*.

Misalnya pada Gambar 3, seluruh *hiddentext* harus mengisi seluruh area yang berwarna putih terlebih dahulu sebelum dapat mengisi posisi bit selanjutnya.

Selain itu, *faketext* tetap disisipkan terlebih dahulu sebelum *hiddentext*, agar *faketext* tetap mudah ditemukan.

IV. Hasil Uji Coba dan Analisis

Secara garis besar, uji coba yang dilakukan adalah dengan meninjau tiga kriteria algoritma steganografi. *Imperceptible* diujicobakan dengan melihat tingkat persepsibilitas *faketext* dan *hiddentext*. *fidelity* dengan mengukur perubahan citra secara matematis, yakni PSNR. Pemeriksaan *recovery* dilakukan di setiap tahap ujicoba. Masukan yang digunakan pada uji coba adalah suatu string acak yang berisi karakter alfabet yang memiliki panjang tertentu.

Tujuan dari uji coba ini adalah untuk menentukan kualitas dari citra yang dihasilkan, serta mencari batas-batas jumlah pengisian bit yang dapat dilakukan.

Uji coba kedua adalah dengan mencoba melakukan steganalisis dengan sejumlah metode-metode yang sederhana. Tujuannya adalah untuk melihat seberapa baik algoritma yang dirancang dalam menyembunyikan *hiddentext*, serta dalam menonjolkan *faketext*.

A. Uji Coba Kualitas

Pertama kali dilakukan uji coba pada input yang normal, dengan jumlah *faketext* dan ciphertext sama, dan hanya menempati 1 LSb. Gambar 4 di sebelah kiri merupakan citra Lena[3] sebelum disisipkan pesan (*covertext*), sedangkan yang sebelah kanan adalah citra yang sudah disisipkan pesan (*stegotext*).

Gambar 4 – Ujicoba pengisian 1 LSb

Secara perseptif, hampir tidak ada perbedaan pada kedua citra. Nilai PSNR yang didapat adalah 46,4112. *Hiddentext* maupun *faketext* juga dapat diekstraksi kembali dari *stegotext*.

Berikut hasil pengambilan data untuk setiap jumlah pemakaian bit LSb terhadap nilai PSNR yang didapatkan. Data yang didapatkan menggunakan kasus normal, yakni panjang *faketext* dan *hiddentext* kira-kira memiliki jumlah yang sama.

Jumlah LSb	PSNR
1	46,4112
2	39,4061
3	33,0927
4	27,0112
5	20,9460
6	15,1087

Tabel 1 – Nilai PSNR terhadap jumlah LSb yang dipakai

Nilai PSNR menjadi < 30 pada saat pada pemakaian 4 LSb secara penuh. PSNR memiliki nilai sebesar 27,0112, dan secara visual citra sudah terlihat berbintik-bintik dibandingkan dengan citra aslinya, namun secara kasat mata masih belum terlihat adanya perbedaan. *recovery* tetap dapat dilakukan.

Jika dilanjutkan, kualitas citra akan semakin memburuk, dan tingkat perseptibilitas yang signifikan mulai terjadi pada jumlah LSb sebanyak 6. Pada saat itu, citra yang dihasilkan sudah sangat terlihat kerusakannya, dimana informasi warna sudah banyak yang menghilang.

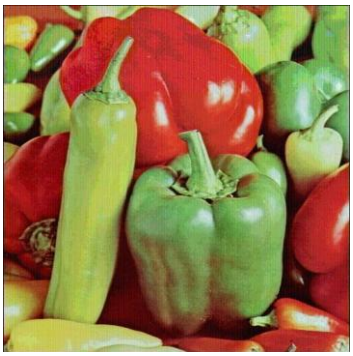
Gambar 5 – Ujicoba pengisian 6 LSb

Uji coba kedua adalah mencoba menentukan batas dari ukuran *faketext*.

Seperti yang telah diketahui bahwa *faketext* disisipkan dengan cara yang sederhana, yakni secara sekuensial tanpa terenkripsi. Jika penyisipan tidak hanya dilakukan pada 1 LSb, maka penyisipan tetap akan dilanjutkan pada posisi bit-bit selanjutnya.

Oleh karena itu, jika *faketext* tidak akan mengisi tepat suatu posisi bit secara penuh, maka terdapat perbedaan kualitas pada bagian sisi atas gambar dan bagian bawah gambar.

Setelah diujicobakan pada citra Peppers[3], perbedaan kualitas mulai sedikit terlihat pada saat pemanfaatan pemanfaatan LSb sebanyak 6 buah, dengan PSNR sebesar 23,213. Pada Gambar 5, dapat dilihat adanya perbedaan kualitas pada ¼ bagian atas dengan ¾ bagian bawah pada citra.



Gambar 6 – Penggunaan 6 LSb untuk menyimpan *faketext*

Perbedaan kualitas menjadi sangat terlihat pada saat bit ketujuh juga dimanfaatkan untuk penyisipan *faketext*, dengan hasil PSNR yang didapatkan adalah sebesar 17,197.

Gambar 7 – Penggunaan 7 LSb untuk menyimpan *faketext*

Perbedaan kualitas semakin meningkat sebanding dengan pemakaian jumlah LSb, karena posisi bit yang signifikan menjadi terisi, dan semakin signifikan posisi bit tersebut, maka pengubahan bit tersebut akan mengubah citra secara drastis.

Walaupun demikian, penyimpanan *faketext* yang besar hingga menggunakan bit-bit pada MSb sebenarnya tidak selalu buruk. Perbedaan kualitas tersebut justru sebenarnya juga dapat "memancing" steganalis untuk melakukan ekstraksi pesan pada LSb dengan cara sekuensial, sehingga *faketext* nantinya terbongkar dan *hiddentext* tetap dalam keadaan aman.

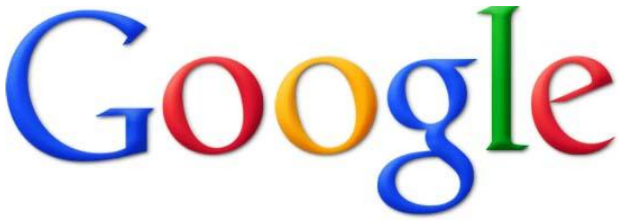
Aspek *recovery* berhasil dicakup pada algoritma, karena seluruh uji coba yang dilakukan berhasil mengembalikan *faketext* dan *hiddentext* seperti semula.

B. Uji Coba Steganalisis

Uji coba dilakukan dengan menggunakan steganalisis yang sederhana, seperti langsung melakukan ekstraksi pesan secara langsung dari bit LSb terkecil. Selain itu, metode steganalisis lain yang digunakan adalah dengan menggunakan *enhanced LSb*.

Ekstraksi secara langsung akan langsung mendapatkan *faketext*, beserta *header*-nya di bagian awal. Hal ini tentu saja mudah dipahami karena pengisian *faketext* sendiri menggunakan metode tersebut. *Faketext* mudah ditemukan dengan metode yang naif seperti mengambil LSb terkecil. Oleh karena itu, algoritma *hiddentext* palsu bekerja seperti seharusnya jika dilakukan steganalisis dengan cara mengekstraksi nilai LSb terkecil.

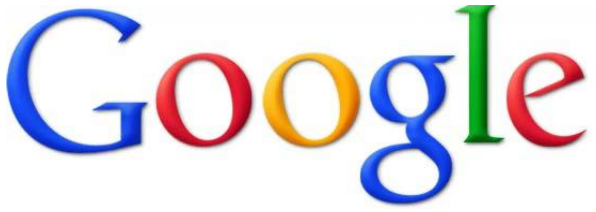
Uji coba selanjutnya dilakukan dengan melakukan steganalisis menggunakan *enhanced LSb*. *Covertext* yang digunakan adalah citra logo Google[5], dimana citra tersebut memiliki kontras yang tinggi.



Gambar 8 – Logo Google

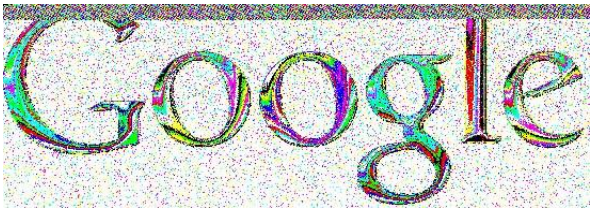
Gambar 9 menunjukkan citra logo Google setelah dilakukan *embedding* dengan menggunakan hanya 1 bit

LSb. Terlihat bahwa hampir tidak ada perubahan yang berarti pada citra.



Gambar 9 – Logo Google setelah *embedding*

Sayangnya, hasil analisis dengan menggunakan *enhanced LSb* secara eksplisit menunjukkan kebereadaan *hiddentext* (dan juga *faketext*).

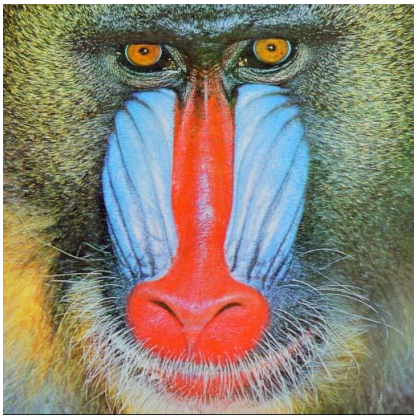


Gambar 10 – Hasil *enhanced LSb* logo Google

Noise pada bagian atas gambar menunjukkan adanya pesan tersembunyi pada bagian itu, pada hal ini adalah *faketext*. Namun terdapat pesan tersembunyi lain pada area yang lain, karena terlihat adanya noise pada bagian tersebut. Pesan tersembunyi tersebut adalah *hiddentext*. Oleh karena itu, algoritma *hiddentext* palsu berhasil dipecahkan jika menggunakan *enhanced LSb*, jika menggunakan *covertext* yang kontras.

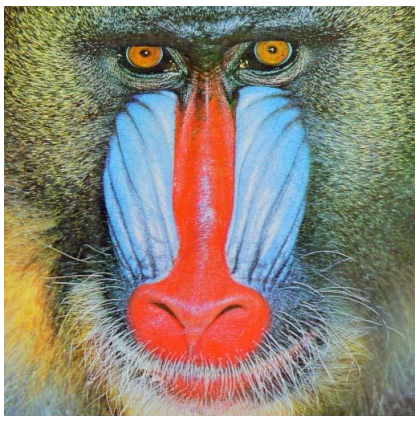
Namun bagaimana dengan *covertext* yang tidak kontras?

Untuk mengujinya, digunakan citra uji Baboon[3]. Citra ini dinilai tepat untuk digunakan karena memiliki penyebaran warna yang cenderung acak dan bervariasi.



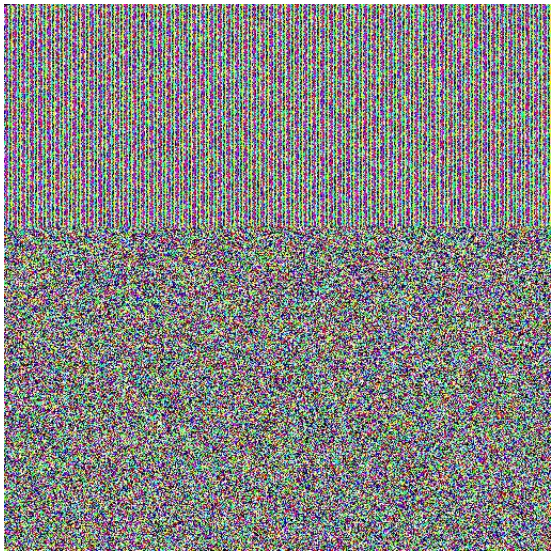
Gambar 11 – Citra Baboon

Gambar 12 menunjukkan citra setelah dilakukan *embedding* pesan. Sama seperti sebelumnya, proses *embedding* hanya menggunakan 1 LSb untuk penyimpanan pesan. Seperti citra logo Google, tidak terdapat perubahan yang signifikan pada citra.



Gambar 12 – Citra Baboon setelah *embedding*

Gambar 13 menunjukkan hasil steganalisis citra dengan menggunakan *enhanced LSb*. Terlihat bahwa terdapat suatu pola pada area atas citra.

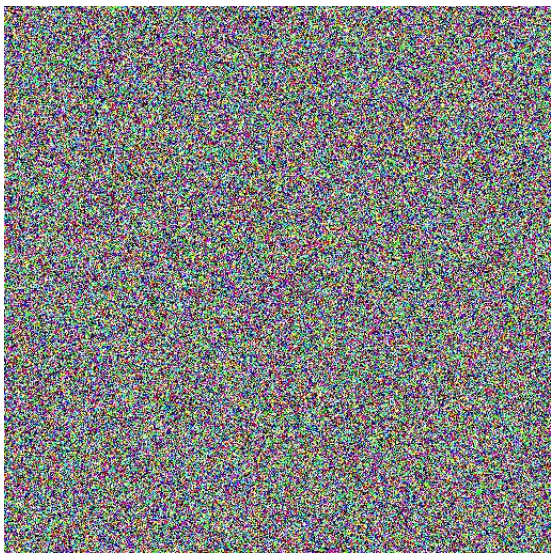


Gambar 13 – Hasil *enhanced LSb* citra Baboon

Pola pada bagian atas tersebut merupakan *faketext* yang tersembunyi pada citra. Sedangkan *hiddentext*, yang seharusnya berada pada bagian bawah citra, sama sekali tidak terlihat. Karena itu, pada kasus ini, algoritma steganografi *hiddentext* palsu bekerja dengan baik, karena berhasil menonjolkan *faketext* dan menyembunyikan *hiddentext*.

Namun darimana asal pola tersebut? Pola tersebut diduga berasal karena string masukan yang digunakan hanya berupa string dengan karakter alfabet. Karakter alfabet memiliki nilai ASCII yang saling berdekatan, sehingga jika dituliskan secara sekuensial, tanpa adanya enkripsi apapun, bit-bit warna yang dihasilkan akan cenderung memiliki pola yang sama.

Untuk memeriksa hipotesis tersebut, domain dari karakter string masukan diganti dengan seluruh 256 karakter ASCII. Ternyata memang pola tersebut langsung menghilang, seperti yang terlihat pada Gambar 14.



Gambar 14 – Hasil *enhanced LSb* dengan masukan *faketext* berupa 256 karakter ASCII

Untuk *hiddentext*, pesan tersembunyi tersebut tidak terlihat karena penyebarannya yang acak pada citra.

Selain itu, mengingat fenomena pola yang terjadi pada Gambar 13, proses enkripsi juga membantu dalam menyamarkan *hiddentext*. Karena dengan melakukan enkripsi, domain string yang disisipkan pada citra akan selalu menjadi 256 karakter ASCII, sesempit apapun domain string pada *hiddentext*.

Jadi, domain yang sempit pada masukan *faketext* (e.g. hanya karakter alphabet/alphanumeric) diperlukan untuk menghasilkan pola *faketext* seperti pada Gambar 13.

KESIMPULAN

Setelah dilakukan berbagai macam uji coba, dapat ditarik beberapa kesimpulan mengenai rancangan algoritma yang telah dijabarkan.

Jumlah LSb yang dapat dipakai tanpa merusak *covertext* secara signifikan pada kasus rata-rata adalah adalah sebanyak 3-4 LSb.

Faketext sendiri sebaiknya hanya mencakup maksimum 4-5 LSb agar tidak terdapat perbedaan kualitas antara bagian atas dan bagian bawah citra. Namun hal tersebut tidak mutlak, karena sebenarnya perbedaan kualitas tersebut juga dapat digunakan untuk menipu steganalis.

Algoritma dapat bekerja dengan baik ketika dilakukan steganalisis sederhana, seperti ekstraksi LSb.

Algoritma gagal menutup *hiddentext* jika dianalisis dengan *enhanced LSb*, dan menggunakan *covertext* yang memiliki kontras tinggi.

Jika kontras gambar rendah, algoritma bekerja seperti yang diharapkan. *Enhanced LSb* akan menemukan *faketext* dan gagal dalam menemukan *hiddentext*. Namun hal ini akan terjadi jika *faketext* memiliki domain karakter yang sempit, seperti hanya berisi alfabet saja. Oleh karena itu, *faketext* yang digunakan sebaiknya berupa string dengan domain yang sempit.

Terdapat sejumlah pengembangan yang dapat dilakukan pada teknik steganografi yang menggunakan *hiddentext* palsu ini. Salah satunya adalah dengan cara sengaja membuat *faketext* lebih terlihat menonjol, seperti pada perbedaan kualitas citra pada bagian atas dan bawah.

Untuk algoritma steganografi itu sendiri, dapat dilakukan pengembangan agar *faketext* dapat mencakup seluruh karakter ASCII. Hal ini dapat dilakukan misalnya dengan mengubah terlebih dahulu *faketext* ke Base64, setelah itu baru kemudian dilakukan *embedding*. Namun perlu diperhatikan bahwa dengan cara ini, *faketext* tidak dapat langsung terbaca isinya setelah diekstraksi dari *stegotext*.

REFERENSI

[1] Bahan Kuliah Steganografi – Rinaldi Munir, 24 Maret 2013

[2] Tugas Besar I IF3058 Kriptografi Sem. II Tahun 2012/2013, 24 Maret 2013

[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Koleksi/Citra%20Uji/CitraUji.htm, 24 Maret 2013

[4] Tugas akhir Yuli Anneria Sinaga, IF 2004 ITB, via [1], 25 Maret 2013

[5] www.google.com, 25 Maret 2013

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Maret 2013

Okaswara Perkasa (13510051)