# UJI KUALITAS DAN KETAHANAN ALGORITMA F5 PADA *STEGO IMAGE* TERHADAP *IMAGE DISTORTION*

**SKRIPSI**


Oleh:
**MADE AGUSTIA PERMATA WARDANI**
**0410960034 – 96**



**PROGRAM STUDI ILMU KOMPUTER**
**JURUSAN MATEMATIKA**
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**
**UNIVERSITAS BRAWIJAYA**
**MALANG**

**2008**

**UJI KUALITAS DAN KETAHANAN ALGORITMA F5 PADA *STEGO IMAGE* TERHADAP *IMAGE DISTORTION***

**SKRIPSI**

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer
dalam bidang Ilmu Komputer


Oleh:
**MADE AGUSTIA PERMATA WARDANI**
**0410960034 – 96**




**PROGRAM STUDI ILMU KOMPUTER**
**JURUSAN MATEMATIKA**
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**
**UNIVERSITAS BRAWIJAYA**
**MALANG**

**2008**

**LEMBAR PENGESAHAN SKRIPSI**

**UJI KUALITAS DAN KETAHANAN ALGORITMA F5 PADA *STEGO IMAGE* TERHADAP *IMAGE DISTORTION***

Oleh:
**MADE AGUSTIA PERMATA WARDANI**
**0410960034 – 96**

**Setelah dipertahankan di depan Majelis Penguji**
**pada tanggal 06 Agustus 2008**
**dan dinyatakan memenuhi syarat untuk memperoleh gelar**
**Sarjana Komputer dalam bidang Ilmu Komputer**

| Dosen Pembimbing I, | Dosen Pembimbing II, |
|---|---|
| Wayan F. Mahmudy, SSi, MT | Bayu Rahayudi, ST.MT |
| NIP. 132 158 724 | NIP. 132 318 424 |

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc
NIP. 132 126 049

**LEMBAR PERNYATAAN**
**Saya yang bertanda tangan di bawah ini :**

**Nama : Made Agustia Permata Wardani**
**NIM : 0410960034-96**
**Jurusan : Matematika**
**Program Studi : Ilmu Komputer**
**Penulis tugas akhir berjudul: UJI KUALITAS DAN KETAHANAN ALGORITMA F5 PADA *STEGO IMAGE* TERHADAP *IMAGE DISTORTION*.**

**Dengan ini menyatakan bahwa :**

1. **Isi dari tugas akhir yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Tugas Akhir ini.**
2. **Apabila dikemudian hari ternyata Tugas Akhir yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

**Demikian pernyataan ini dibuat dengan segala kesadaran.**

**Malang, 06 Agustus 2008**
**Yang menyatakan,**

**Made Agustia P W**
**NIM. 0410960034-96**

# UJI KUALITAS DAN KETAHANAN ALGORITMA F5 PADA *STEGO IMAGE* TERHADAP *IMAGE DISTORTION*

## ABSTRAK

Steganografi merupakan ilmu dan seni yang mempelajari cara penyembunyian informasi pada suatu media sedemikian rupa sehingga keberadaannya tidak terdeteksi oleh pihak lain yang tidak berhak atas informasi tersebut. Ada beberapa media yang dapat digunakan untuk menyisipkan pesan dalam teknik steganografi, salah satunya adalah citra digital yang sering disebut s*tego image*. Steganografi pada media citra digital digunakan untuk mengeksploitasi keterbatasan kekuatan sistem penglihatan manusia dengan cara menurunkan kualitas warna pada file gambar yang belum disisipi pesan rahasia. Teknik atau algoritma yang digunakan dalam tugas akhir ini adalah algoritma F5. Algoritma F5 ini menyisipkan bit data pesan ke dalam bit koefisien DCT kemudian membuat matriks encoding untuk mengurangi atau meminimalkan jumlah perubahan-perubahan yang diperlukan untuk menyisipkan suatu pesan dengan panjang tertentu. Penilaian sebuah algoritma steganografi yang baik dapat dinilai dari beberapa faktor salah satunya adalah *fidelity* yaitu dimana mutu atau kualitas citra penampung setelah ditambahkan pesan tidak jauh berbeda dengan kualitas citra penampung sebelum ditambahkan pesan. Dan *robustness* yaitu data yang disembunyikan harus tahan terhadap berbagai operasi manipulasi atau penambahan *image distortion* yang dilakukan pada citra *stego*. Oleh karena itu untuk mengetahui apakah algoritma F5 ini baik digunakan sebagai algoritma steganografi maka akan dilakukan uji kualitas dan uji ketahanan. Uji kualitas dilakukan dengan cara menghitung *Root Mean Square* (RMS) dari citra asli dan citra *stego*. Berdasarkan hasil uji coba kualitas maka didapatkan hasil yaitu rata-rata RMS dari citra input format BMP adalah 1,376% dan rata-rata citra input format JPEG adalah 1,418%. Sedangkan uji ketahanan maka citra *stego* akan diberi penambahan *image distortion* untuk mengetahui apakah data yang disembunyikan tetap valid jika diekstraksi. Dan hasil dari uji coba ketahanan ini adalah pesan dalam citra stego yang diberi penambahan efek *gaussian blur* dan intensitas warna tidak dapat diekstrak kembali. Sehingga algoritma F5 mempunyai kelemahan yaitu tidak tahan atau *robust* terhadap penambahan *image distortion*.

**TEST FIDELITY AND ROBUSTNESS OF ALGORITHM F5 AT STEGO IMAGE TO IMAGE DISTORTION**

## ABSTRACT

Steganography is a science and art of hiding message into a media such the way so the existence of the message is not recognize by human sensor system. The media itself does not suffer any significant damage. There are some media available for applied to insert message in steganography's technique, one of them is digital image which often called as stego image. Steganography at applied digital image media for exploiting limitation of strength of system of eyesight of human with reducing quality of color at picture file which not yet been inserted message of secret. F5 algorithm is a technique which is used in this undergraduate thesis. The F5 algorithm embeds message bits into randomly-chosen DCT coefficients and employs matrix embedding that minimizes the necessary number of changes to embed a message of certain length. The good assessments of Steganography's algorithm are fidelity or quality of cover image with secret messages is as good as cover image with no secret messages. And robustness that is hidden messages must robust from image distortions which are given in stego image. Therefore to know that F5 algorithm is good for Steganography, so it will be conducted a testing for fidelity and robustness. Fidelity testing will be conducted by calculating Root Mean Square (RMS) between original image and stego image. Based on the result of fidelity testing, the mean of RMS from input images BMP is 1,376% and mean of RMS form input images JPEG is 1,418%. And stego image will be given image distortions for robustness testing to know that secret messages which having been hidden still valid if stego image is extracted. The result of robustness testing is secret messages in stego image which is given a Gaussian blur effect and colors intensity can not be extracted. So, F5 algorithm has a weakness that is not robust from image distortions.

## KATA PENGANTAR

Puji syukur ke hadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya sehingga dapat menyelesaikan tugas akhir ini sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dalam bidang Ilmu Komputer.

Perkembangan dunia digital membuat banyak kalangan untuk menciptakan teknik untuk pengamanan data. Teknik pengamanan data pun bermacam-macam, mulai dari menyamarkan data tersebut maupun dengan menyembunyikannya ke dalam media lain. Teknik ini disebut Steganografi.

Penelitian dengan judul " UJI KUALITAS DAN KETAHANAN ALGORITMA F5 *STEGO IMAGE* TERHADAP *IMAGE DISTORTION*" bertujuan untuk melakukan pengujian dan analisa terhadap algoritma F5 yang digunakan sebagai teknik pengamanan data yaitu steganografi yang menggunakan media citra digital. Sehingga dengan mengetahui hasil uji coba dari algoritma ini diharapkan dapat dijadikan sebagai masukan salah satu metode untuk pengamanan data.

Dalam penyusunan tugas akhir ini, Penulis mengucapkan terima kasih kepada :

1. Wayan Firdaus M, S.Si, MT selaku pembimbing utama penulisan tugas akhir sekaligus Ketua Program Studi Ilmu Komputer, Jurusan Matematika, FMIPA Universitas Brawijaya.
2. Bayu Rahayudi, ST, MT selaku pembimbing pendamping dalam penulisan tugas akhir ini.
3. Drs. Agus Suryanto, M.Sc selaku Ketua Jurusan Matematika FMIPA Universitas Brawijaya.
4. Nurul Hidayat, S.Pd, M.Sc selaku dosen pembimbing akademis.
5. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
6. Kepada kedua Orang Tua Penulis dan saudara-saudara (mbak Novi dan Nia) yang tak pernah berhenti memberikan doa dan dukungannya kepada Penulis.

7. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya yang telah banyak membantu Penulis dalam pelaksanaan penyusunan tugas akhir ini.
8. Rekan-rekan di Program Studi Ilmu Komputer angkatan 2004 FMIPA Universitas Brawijaya yang telah banyak memberikan bantuannya demi kelancaran pelaksanaan penyusunan tugas akhir ini.
9. Dan semua pihak yang telah membantu dalam penyusunan laporan ini yang tidak dapat kami sebutkan satu per satu.

Dalam penyusunan laporan ini masih banyak kekurangan yang disebabkan oleh keterbatasan kemampuan dan pengalaman.

Semoga laporan ini dapat memberikan manfaat kepada pembaca dan bisa diambil manfaatnya, baik oleh mahasiswa maupun pihak-pihak lain yang tertarik untuk menekuni dan berkecimpung di pengembangan keamanan data digital khususnya teknik steganografi.

Malang, 06 Agustus 2008

Penulis

## DAFTAR ISI

Halaman

**BAB III METODOLOGI PENELITIAN**



**BAB IV HASIL DAN PEMBAHASAN**



**BAB V KESIMPULAN DAN SARAN**

## DAFTAR GAMBAR

Halaman

## DAFTAR TABEL

Halaman

## DAFTAR SOURCECODE

Halaman

# BAB I
# PENDAHULUAN

## 1.1. Latar Belakang

Dewasa ini penyembunyian pesan tidak hanya dapat dilakukan dengan menyamarkan pesan tersebut, melainkan dapat pula menyisipkan pesan tersebut ke dalam media lain. Sehingga orang tidak akan curiga terhadap pesan yang dikirimkan, karena pesan tersebut tidak terlihat. Yang terlihat hanyalah media penampung pesan tersebut. Teknik penyisipan pesan dalam media lain yang lebih besar ini dinamakan Steganografi.

Steganografi merupakan salah satu cara untuk menyembunyikan suatu pesan / data rahasia di dalam data atau media lain yang tampak tidak mengandung apa-apa, kecuali bagi orang yang mengerti kuncinya (Penalosa, 2005). Sedangkan s*tego image* adalah hasil atau *output* dari proses steganografi dengan menggunakan media penyisipan berupa citra digital.

Banyak kalangan umum lebih mengenal kriptografi daripada steganografi. Walaupun steganografi dapat dikatakan mempunyai hubungan yang erat dengan kriptografi, tapi metoda ini sangat berbeda dengan kriptografi. Kriptografi mengacak pesan sehingga tidak dimengerti, sedangkan steganografi menyembunyikan pesan sehingga tidak terlihat. Menurut Penalosa dalam penelitiannya dengan judul "Steganografi Pada Citra Dengan Format GIF Menggunakan Algoritma GifShuffle" pada tahun 2005, dengan menggunakan steganografi maka orang tidak akan curiga kalau ternyata terdapat pesan rahasia. Tetapi ada harga yang harus dibayar dalam steganografi yaitu besarnya ukuran file yang disisipi pesan rahasia. Perlu dilakukan transfer data dalam jumlah besar padahal data penting yang diperlukan berukuran kecil.

Teknik steganografi pernah dilakukan oleh Setiana pada tahun 2006 dengan menggunakan metode *Least-Significant Bit* (LSB) *Insertion*, metode ini merupakan metode paling sederhana tetapi metode LSB ini paling tidak tahan terhadap proses yang dapat mengubah nilai-nilai intensitas pada citra dan metode ini paling mudah diserang. Selain LSB, metode yang juga pernah digunakan adalah metode *GifShuffle* yang digunakan pada citra digital dengan format GIF. Metode inipun cukup mudah digunakan karena hanya mengandung langkah-langkah singkat yang tidak rumit tetapi metode

ini memiliki kelemahan yaitu citra yang telah disisipi pesan tidak tahan terhadap perubahan. Karena setelah dilakukan beberapa perubahan terhadap citra diperoleh hasil bahwa pesan tidak lagi dapat diekstrak serta metode ini hanya cocok untuk menyisipkan pesan-pesan pendek yaitu pesan yang berukuran 209 bytes atau 209 karakter (Penalosa, 2005). Selain metode LSB dan *GifShuffle*, penyembunyian pesan dapat menggunakan *masking* dan *filtering* serta transformasi (Henry, 2000). Dan teknik atau algoritma yang digunakan dalam tugas akhir ini adalah algoritma F5. Algoritma F5 ini menyisipkan bit data pesan ke dalam bit koefisien DCT kemudian membuat matriks encoding untuk mengurangi atau meminimalkan jumlah perubahan-perubahan yang diperlukan untuk menyisipkan suatu pesan dengan panjang tertentu. (Fridrich, dkk, 2002).

Penilaian sebuah algoritma steganografi yang baik dapat dinilai dari beberapa faktor yaitu *imperceptible* atau keberadaan pesan dalam media penampung tidak dapat dideteksi. *Fidelity* yaitu dimana mutu atau kualitas citra penampung setelah ditambahkan pesan tidak jauh berbeda dengan kualitas citra penampung sebelum ditambahkan pesan. *Recovery* yaitu pesan yang disisipkan harus dapat diambil atau diekstrak kembali dan *robustness* yaitu data yang disembunyikan harus tahan terhadap berbagai operasi manipulasi atau penambahan *image distortion* yang dilakukan pada citra *stego*, seperti pengubahan kontras, penajaman, penghalusan (*smooth*) dan operasi geometri seperti rotasi, translasi (*flip*) dan *scaling* serta penambahan efek seperti efek *blur* dan efek *sepia*. Bila pada citra *stego* dilakukan operasi-operasi pengolahan citra tersebut, maka data yang disembunyikan seharusnya tidak rusak (tetap valid jika diekstraksi kembali).

Masalah yang timbul adalah apakah algoritma F5 baik atau layak untuk digunakan dalam steganografi. Oleh karena itu akan dilakukan pengujian terhadap algoritma F5. Pengujian tersebut meliputi pengujian terhadap kualitas citra apakah setelah disisipkan pesan mengalami penurunan kualitas atau tidak dan ketahanan citra *stego* untuk melihat apakah pesan yang disisipkan masih dapat diekstrak meskipun gambar mengalami beberapa perubahan.

Maka berdasarkan latar belakang yang telah dipaparkan dalam tugas akhir ini, penulis mengangkat judul yaitu "**UJI KUALITAS DAN KETAHANAN ALGORITMA F5 PADA *STEGO IMAGE* TERHADAP *IMAGE DISTORTION*."**

## 1.2. Rumusan Masalah

Masalah-masalah yang akan diteliti dan dipecahkan pada tugas akhir ini adalah:

1. Bagaimana mengimplementasikan steganografi sebagai salah satu teknik untuk pengamanan data dengan algoritma F5.
2. Bagaimana mengevaluasi kualitas citra setelah dilakukan *embedding* data terhadap citra tersebut.
3. Bagaimana mengevaluasi pengaruh *stego image* setelah dilakukan penambahan *image distortion*.

## 1.3. Batasan Masalah

Batasan masalah yang dibahas dalam tugas akhir ini adalah :

1. Citra yang akan disisipi pesan rahasia adalah "citra diam" (*still images*) dengan format BMP (*.bmp) dan JPEG (*.jpg).
2. Data yang akan disisipkan berupa data teks dokumen (*.txt).
3. Teknik atau algoritma yang digunakan untuk menyembunyikan data adalah algoritma F5.
4. Distorsi yang akan diberikan pada *stego image* yaitu berupa penambahan efek *gaussian blur* dan penambahan intensitas warna (penambahan nilai RGB)

## 1.4. Tujuan Penelitian

Tujuan dari penulisan tugas akhir ini adalah :

1. Mengimplementasikan teknik steganografi dengan algoritma F5 ke salah satu bahasa pemrograman.
2. Membuat perangkat lunak untuk melakukan pengujian terhadap *stego image*.
3. Membandingkan kualitas *stego image* dengan citra aslinya.
4. Melakukan pengujian ketahanan algoritma F5 terhadap beberapa penambahan *image distortion* yang dilakukan pada *stego image.*
5. Melakukan analisa terhadap *stego image* setelah dilakukan penambahan *image distortion.*
6. Menyampaikan seberapa aman steganografi pada media citra digital untuk menyimpan data atau pesan yang dirahasiakan.

## 1.5. Manfaat Penelitian

Manfaat yang dapat diambil dari penulisan tugas akhir ini adalah:

1. Memahami penerapan algoritma F5 untuk melakukan steganografi sebagai teknik pengamanan data digital.
2. Memberikan pandangan bahwa steganografi dengan menggunakan algoritma F5 memiliki tingkat keamanan yang cukup tinggi.

## 1.6 Metodologi Pemecahan Masalah

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan tugas akhir ini adalah:

1. Studi Literatur
   Studi ini dilakukan dengan cara mencari sekaligus mempelajari beberapa literatur dan artikel mengenai steganografi dan pemrograman *file* citra. Disamping itu juga mempelajari program aplikasi yang sudah ada untuk memberikan gambaran yang jelas mengenai aplikasi steganografi, sebagai acuan dalam perencanaan dan pembuatan Tugas Akhir.
2. Pendefinisian dan analisis masalah.
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi sistem
4. Uji coba dan analisis hasil implementasi
   Menguji coba perangkat lunak tersebut dan menganalisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian dievalusi.

## 1.7 Sistematika Penulisan

Adapun metodologi penulisan tugas akhir ini adalah sebagai berikut :

1. BAB I PENDAHULUAN
   Dalam bab ini membahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan metodologi pemecahan masalah serta sistematika penulisan.

2. BAB II TINJAUAN PUSTAKA

Menjelaskan dasar teori yang digunakan dalam menyusun tugas akhir, yaitu penjelasan mengenai steganografi, media penyisipan dalam teknik steganografi, penilaian algoritma steganografi, perbedaan steganografi dengan kriptografi, perbedaan steganografi dengan *watermarking*, konsep dasar citra digital, *pixel* dan warna, format citra BMP, format citra JPEG, *Discrete Cosine Transform* (DCT), kuantisasi, algoritma atau teknik yang digunakan yaitu algoritma F5, algoritma Huffman dan penambahan *image distortion* pada *stego image*.

3. BAB III METODOLOGI PENELITIAN

Dalam bab ini membahas mengenai proses penyisipan pesan dalam citra (*embedding/encoding*), proses penguraian pesan (*extraction*), perancangan uji coba yang meliputi citra uji, pengujian kualitas *stego image*, serta pengujian ketahanan *stego image* apabila diberi distorsi.

4. BAB IV HASIL DAN PEMBAHASAN

Dalam bab empat ini akan dibahas mengenai hasil implementasi dari perangkat lunak yang telah dibahas pada bab metodologi penelitian yaitu berupa implementasi program yang meliputi proses *embedding* dan proses *extraction* serta implementasi uji coba yang meliputi evaluasi kualitas citra dan evaluasi ketahanan citra *stego*.

5. BAB V KESIMPULAN DAN SARAN

Bab lima ini berisi kesimpulan dari hasil dan pembahasan serta saran.


UNIVERSITAS BRAWIJAYA

# BAB II
# TINJAUAN PUSTAKA

## 2.1 Steganografi

Seperti telah disebutkan sebelumnya steganografi adalah sebuah teknik penyisipan pesan dalam media yang lebih besar. Dalam bahasa Yunani kata ”Steganography” diterjemahkan sebagai ”Steganos” yang berarti tulisan tersembunyi. Sehingga dapat diartikan dalam terjemahan bebas bahwa Steganografi adalah ilmu dan seni menyisipkan informasi dengan cara menyisipkan pesan ke dalam pesan lain yang lebih besar.

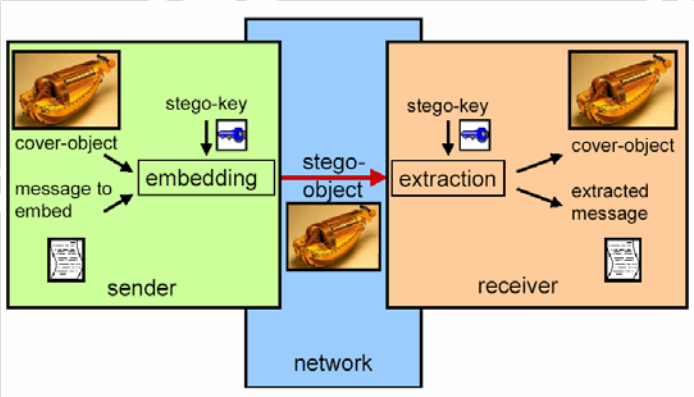
Sejarah mencatat bahwa steganografi pertama kali digunakan pada tahun 440 sebelum masehi. Ketika itu Demeratus mengirimkan berita tentang penyerangan yang akan dilakukan ke Yunani. Beliau mengirimkan pesan tersebut dalam sebuah kayu dan menutupinya dengan lilin. Sehingga pesan yang dikirimkannya tidak terlihat mencurigakan dan kerahasian pesan tetap terjaga (Irianto, 2004).

Tentunya metode Steganografi tersebut sudah sangat usang jika digunakan dalam dunia modern seperti sekarang ini. Pada awalnya metode ini berupa penggunaan tinta yang tidak nampak, pengaturan karakter, tanda tangan digital, saluran yang dikacaukan, dan spektrum komunikasi yang disebar. Teknik Steganografi yang digunakan dalam dunia modern sekarang ini sudah amat beragam. Beragam mulai dari algoritma yang digunakannya sampai pada media yang digunakannya.

Dalam bidang keamanan komputer, steganografi digunakan untuk menyembunyikan data rahasia saat enkripsi tidak dapat dilakukan atau bersamaan dengan enkripsi. Jadi, walaupun enkripsi berhasil dipecahkan (*decipher*) pesan / data rahasia tetap tidak terlihat. Selain itu, pada kriptografi pesan disembunyikan dengan “diacak” sehingga pada kasus-kasus tertentu dapat dengan mudah mengundang kecurigaan, sedangkan pada steganografi pesan “disamarkan” dalam bentuk yang relatif “aman” sehingga tidak terjadi kecurigaan itu (Henry, 2006).

Ada dua proses utama dalam steganografi yaitu penyisipan (*embedding*) dan penguraian (*extraction*) pesan atau data dalam media cover. *Embedding* merupakan proses menyisipkan pesan atau data ke dalam media cover, sedangkan *extraction* adalah proses menguraikan pesan yang tersembunyi dalam gambar stego (*Stego*

*image*). Pesan yang akan disembunyikan dalam sebuah gambar membutuhkan dua file. Pertama adalah gambar asli yang belum dimodifikasi yang akan menangani pesan tersembunyi, yang disebut gambar cover (*cover-object*). File kedua adalah informasi pesan yang akan disembunyikan (*message to embed*). Suatu pesan dapat berupa *plaintext*, *chipertext*, gambar lain, atau apapun yang dapat ditempelkan ke dalam bit stream. Ketika dikombinasikan, *cover image* dan pesan yang ditempelkan membuat gambar stego (*stego-object*). Adapun proses steganografi selengkapnya ditunjukkan pada Gambar 2.1.




**Gambar 2.1** *Steganographic System* (Jendricke, 2003)

Pihak yang terkait dengan steganografi antara lain *embeddor*, *extractor*, dan *stegoanalyst* (Mohanty, 1999). *Embeddor* adalah orang yang melakukan *embedding* dengan menggunakan aplikasi steganografi, *extractor* adalah orang yang melakukan ekstrak *stego image* dengan menggunakan aplikasi steganografi. Sedangkan proses untuk mendeteksi pesan yang tersembunyi dalam steganografi disebut steganalisis (Setiana, 2006).

### 2.1.1 Media Penyisipan dalam Teknik Steganografi

Beberapa contoh media penyisipan pesan rahasia yang digunakan dalam teknik Steganografi menurut Penalosa (2005) antara lain adalah :

1. Teks

   Dalam algoritma Steganografi yang menggunakan teks sebagai media penyisipannya biasanya digunakan teknik LSB (*Least Significant Bit*) sehingga teks yang telah disisipi

pesan rahasia tidak akan mencurigakan orang yang melihatnya.

2. Audio
   Format ini pun sering dipilih karena biasanya berkas dengan format ini berukuran relatif besar. Sehingga dapat menampung pesan rahasia dalam jumlah yang lebih besar pula.
3. Citra
   Format ini pun paling sering digunakan, karena format ini merupakan salah satu format file yang sering dipertukarkan dalam dunia internet. Alasan lainnya adalah banyaknya algoritma Steganografi untuk media penampung berupa citra.
4. Video
   Format ini memang merupakan format dengan ukuran yang relatif sangat besar namun jarang digunakan karena ukurannya yang terlalu besar sehingga mengurangi kepraktisannya dan juga kurangnya algoritma yang mendukung format ini.

### 2.1.2 Penilaian Algoritma Steganografi

Menurut penilaian sebuah algoritma steganografi yang baik dapat dinilai dari beberapa faktor yaitu :

1. *Imperceptible*
   Keberadaan pesan dalam media penampung tidak dapat dideteksi.
2. *Fidelity*
   Mutu media penampung setelah ditambahkan pesan rahasia tidak jauh berbeda dengan mutu media penampung sebelum ditambahkan pesan.
3. *Robustness.*
   Data yang disembunyikan harus tahan (*robust*) terhadap berbagai operasi manipulasi yang dilakukan pada citra penampung, seperti pengubahan kontras, penajaman, pemampatan, rotasi, perbesaran gambar, pemotongan (*cropping*), diputar (*flip*) dan sebagainya. Bila pada citra penampung dilakukan operasi-operasi pengolahan citra tersebut, maka data yang disembunyikan seharusnya tidak rusak (tetap valid jika diekstraksi kembali).

4. *Recovery*

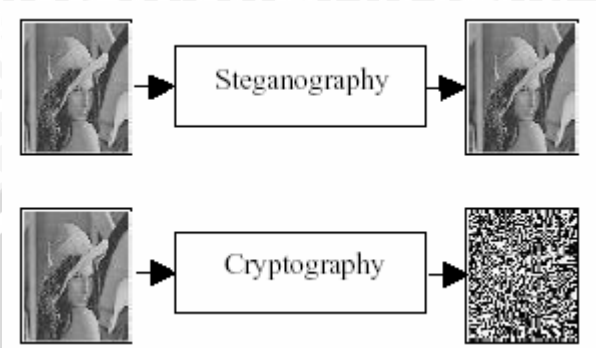
   Pesan rahasia yang telah disisipkan dalam media penampung harus dapat diungkap/diekstrak kembali. Hal ini merupakan syarat mutlak dalam sebuah algoritma steganografi, kerena ada banyak cara penyisipan pesan yang tidak terdeteksi namun sulit dalam pembacaan kembali.

### 2.1.3 Perbedaan Steganografi dengan Kriptografi

Menurut Setiana dalam penelitian tugas akhirnya dengan judul "Steganografi Pada File Citra Bitmap 24 bit Untuk Pengamanan Data Menggunakan Metode *Least Significant Bit* (LSB) *Insertion*" (2006), steganografi dan kriptografi mempunyai prinsip kerja yang berbeda, meskipun keduanya mempunyai hubungan yang dekat dalam dunia keamanan data. Pada kriptografi, menghasilkan sebuah *chipertext* dimana dengan itu seolah-olah dengan sengaja menunjukkan kepada orang lain bahwa ada sesuatu didalamnya, namun tidak dapat diketahui maknanya. Namun dengan bentuk chipernya, justru akan membuat data tersebut terancam oleh usaha-usaha yang dilakukan oleh orang lain untuk dapat membongkarnya dengan tujuan atau alasan apapun.

Steganografi dan kriptografi merupakan seni dan teknik yang dapat digunakan untuk melakukan pengamanan data digital. Namun keduanya tidaklah sama. Pada kriptografi, suatu data digital diamankan dengan cara mengenkripsi data menghasilkan sebuah data yang berupa sand. Secara visual data tersebut masih dapat terlihat atau diketahui, hanya saja data tersebut menjadi tidak dapat dimengerti. Berbeda dengan steganografi yang tujuannya adalah menyembunyikan data ke dalam sebuah media yang lain, sehingga data tersebut tidak terlihat.

Pada aplikasi steganografi modern, kehadiran steganografi bukanlah untuk menggantikan kedudukan kriptografi. Bahkan steganografi diciptakan untuk dapat memperkuat dan menambah satu lapis keamanan data digital.



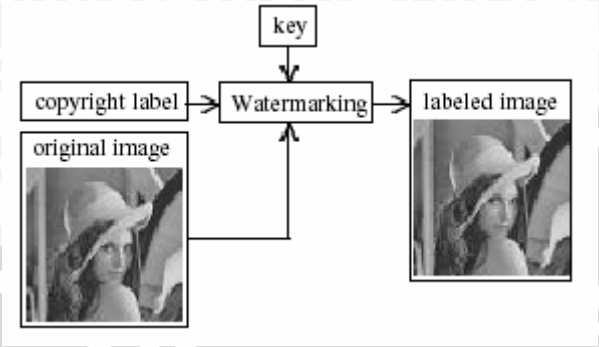

**Gambar 2.2** Ilustrasi Steganografi dan Kriptografi pada Citra (Supangkat, dkk, 2000)

### 2.1.4 Perbedaan Steganografi dengan *Watermarking*

*Watermarking* merupakan aplikasi dari steganografi, namun ada perbedaan antara keduanya. Jika ada steganografi informasi rahasia disembunyikan di dalam media digital dimana media penampung tidak berarti apa-apa, maka pada *watermarking* justru media digital tersebut yang akan dilindungi kepemilikannya dengan pemberian label hak cipta atau *watermark* (Munir, 2004) .

Meskipun steganografi dan *watermarking* tidak sama, namun secara prinsip proses penyisipan informasi ke dalam data digital tidak jauh berbeda. Beberapa metode yang sudah ditemukan untuk penyisipan *watermark* adalah metode LSB, metode adaptif, metode *spread spectrum*, dan sebagainya.

Data *watermark* yang lazim disisipkan ke dalam data digital adalah teks, citra atau suara. *Watermark* berupa teks mengandung kelemahan karena kesalahan satu bit akan menghasilkan hasil teks yang berbeda pada waktu verifikasi (ekstraksi). *Watermark* berupa suara atau citra lebih disukai karena kesalahan pada beberapa bit *watermark* tidak menghasilkan perubahan yang berarti pada waktu verifikasi. Hasil ekstraksi *watermark* yang mengandung kesalahan tersebut masih dapat dipersepsi secara visual (atau secara pendengaran jika *watermark*-nya berupa suara). Citra yang sering digunakan sebagai *watermark* biasanya logo atau lambang.




**Gambar 2.3** Proses *Watermarking* pada Citra (Supangkat,dkk, 2000)

## 2.2 Konsep Dasar Citra Digital

Secara harafiah, citra (*image*) adalah gambar pada bidang dwimatra (dua dimensi). Citra dapat dikelompokkan menjadi citra tampak dan tidak tampak. Contoh dari citra tampak adalah foto, lukisan, dsb. Sedangkan citra tak tampak misalnya citra digital, fungsi matematis ataupun citra distribusi panas ditubuh manusia, dimana apabila ingin dilihat oleh mata manusia maka harus ditampilkan dalam monitor atau dicetak diatas kertas (Latifawariq, 2007).

Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*continoue*) dari intensitas cahaya pada bidang dwimatra. Sumber cahaya menerangi objek, objek memantulkan kembali sebagian dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh alat-alat optik, misalnya mata pada manusia, kamera, pemindai (*scanner*), dan sebagainya, sehingga bayangan objek yang disebut citra tersebut terekam.

Diantara citra tersebut, hanya citra digital yang dapat diolah menggunakan komputer. Agar dapat diolah dengan komputer digital, maka suatu citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Representasi citra dari fungsi malar (kontinu) menjadi nilai-nilai diskrit disebut *digitalisasi*. Citra yang dihasilkan inilah yang disebut citra digital (*digital image*). Pada umumnya citra digital berbentuk empat persegi panjang, dan dimensi ukurannya dinyatakan sebagai tinggi x lebar (atau lebar x panjang).

Citra digital yang tingginya *N*, lebarnya *M*, dan memiliki *L* derajat keabuan dapat dianggap sebagai fungsi :

$$f(x,y)\begin{cases}0 \le x \le M \\ 0 \le y \le N \\ 0 \le f \le L\end{cases} \qquad (2.1)$$

Citra digital yang berukuran *N* x *M* lazim dinyatakan dengan matriks yang berukuran *N* baris dan *M* kolom sebagai berikut :

$$\begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M) \\ f(1,0) & f(1,1) & \dots & f(1,M) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix}$$

Indeks baris (*i*) dan indeks kolom (*j*) menyatakan suatu koordinat titik pada citra, Sedangkan *f*(*i*,*j*) merupakan intensitas (derajat keabuan) pada titik (*i*,*j*) (Munir, 2004).

### 2.2.1 *Pixel* dan Warna

Suatu *pixel* (kependekan dari *picture element*) adalah titik kecil sebagai representasi gambar pada memori komputer. Jumlah *pixel* pada sebuah gambar disebut dengan resolusi. Sebagai contoh, misalkan sebuah citra berukuran 256 x 256 dan direpresentasikan secara numerik dengan matriks yang terdiri dari 256buah baris (di-indeks dari 0 sampai 255) dan 256 buah kolom (di-indeks dari sampai 255) seperti contoh berikut :

$$\begin{bmatrix} 0 & 134 & 145 & \dots & \dots & 231 \\ 0 & 167 & 201 & \dots & \dots & 197 \\ 220 & 187 & 189 & \dots & \dots & 120 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 221 & 219 & 210 & \dots & \dots & 156 \end{bmatrix}$$

*Pixel* pertama pada koordinat (0,0) mempunyai nilai intensitas 0 yang berarti warna *pixel* tersebut hitam, *pixel* kedua pada koordinat (0,1) mempunyai intensitas 134 yang berarti warnanya antara hitam dan putih, dan seterusnya (Munir 2004).

Titik-titik warna pada gambar digital juga disebut *pixel*. Pada gambar berwarna setiap pixelnya memiliki warna dan kecerahannya masing-masing. Kedalaman warna yang dapat direpresentasikan dengan pixel tergantung pada jumlah bit per pixelnya (bpp), kedalaman warna yang biasa dipakai pada gambar digital antara lain

8 bpp (256 warna), 16 bpp (65.536 warna, dikenal dengan *Highcolor*), dan 24 bpp (16.777.216 warna, dikenal dengan *Truecolor*).

Warna adalah aspek dari penglihatan, tanggapan psikofisikal yang mencakup reaksi fisik mata dan tanggapan interpretasi otak terhadap karakteristik gelombang cahaya di atas tingkat kecerahan tertentu (pada titik yang lebih rendah mata dapat merasakan perbedaan tingkat kecerahan tetapi tidak mampu membedakan warna (Scruggs, 2003).

Model warna adalah cara untuk merepresentasikan warna dan hubungan warna satu sama lain. Sistem pemroses gambar yang berbeda menggunakan model warna yang berbeda pula untuk alasan yang berbeda (Setiana, 2006).

### 2.2.2 Format Citra Bitmap

Citra disimpan di dalam berkas (*file*) dengan format tertentu. Format citra yang baku di lingkungan sistem operasi Microsoft Windows dan IBM OS/2 adalah berkas *bitmap* (BMP). Saat ini format BMP memang "kalah" populer dibandingkan format JPG atau GIF. Hal ini karena berkas BMP pada umumnya tidak dimampatkan, sehingga ukuran berkasnya relatif lebih besar daripada berkas JPG maupun GIF. Hal ini juga yang menyebabkan format BMP sudah jarang digunakan.

Meskipun format BMP tidak mangkus dari segi ukuran berkas, namun format BMP mempunyai kelebihan dari segi kualitas gambar. Citra dalam format BMP lebih bagus daripada citra dalam format yang lainnya, karena citra dalam format BMP umumnya tidak dimampatkan sehingga tidak ada informasi yang hilang. Terjemahan bebas *bitmap* adalah pemetaan bit. Artinya, nilai intensitas *pixel* di dalam citra dipetakan ke sejumlah bit tertentu. Peta bit yang umum adalah 8, artinya setiap *pixel* panjangnya 8 bit. Delapan bit ini merepresentasikan nilai intensitas *pixel*. Dengan demikian ada sebanyak $2^8 = 256$ derajat keabuan, mulai dari 0 sampai 255.

Citra dalam format BMP ada tiga macam: citra biner, citra berwarna, dan citra hitam-putih (*grayscale*). Citra biner hanya mempunyai dua nilai keabuan, 0 dan 1. Oleh karena itu, 1 bit sudah cukup untuk merepresentasikan nilai *pixel*. Citra berwarna adalah citra yang lebih umum. Warna yang terlihat pada citra *bitmap* merupakan kombinasi dari tiga warna dasar, yaitu merah, hijau, dan

biru. Setiap *pixel* disusun oleh tiga komponen warna : R (*red*), G (*green*), dan B (*blue*). Kombinasi dari ketiga warna RGB tersebut menghasilkan warna yang khas untuk pixel yang bersangkutan. Pada citra 256 warna, setiap *pixel* panjangnya 8 bit, tetapi komponen warna RGB-nya disimpan di dalam tabel RGB yang disebut **palet**. Setiap komponen RGB panjangnya 8 bit, jadi ada 256 nilai keabuan untuk warna merah, 256 nilai keabuan untuk warna hijau, dan 256 nilai keabuan untuk warna biru. Nilai setiap *pixel* tidak menyatakan derajat keabuannya secara langsung, tetapi nilai *pixel* menyatakan indeks tabel RGB yang memuat nilai keabuan merah (R), nilai keabuan hijau (G), dan nilai keabuan biru (B) untuk *pixel* yang bersangkutan. Pada citra hitam-putih, nilai R = G = B untuk menyatakan bahwa citra hitam-putih hanya mempunyai satu kanal warna. Citra hitam-putih umumnya adalah citra 8-bit.

Citra yang lebih kaya warna adalah citra 24-bit. Setiap pixel panjangnya 24 bit, karena setiap pixel langsung menyatakan komponen warna merah, komponen warna hijau, dan komponen warna biru. Citra 24-bit disebut juga citra 16 juta warna, karena ia mampu menghasilkan $2^{24}$ = 16.777.216 kombinasi warna.

Berkas citra 24-bit (16,7 juta warna) tidak mempunyai palet RGB, karena nilai RGB langsung diuraikan dalam data *bitmap*. Setiap elemen data *bitmap* panjangnya 3 *byte*, masing-masing *byte* menyatakan komponen R, G, dan B. Contoh format citra 24-bit (16 juta warna) kira-kira seperti pada gambar 2.4. Pada contoh format citra 24-bit tersebut *pixel* pertama mempunyai R = 20, G = 19, B = 21, pixel kedua mempunyai R = 24, G = 24, B = 23. Demikian seterusnya (Munir, 2004).

```
<header berkas>
<header bitmap>

<data bitmap>
 20   19   21   24   24   23   24   ...
```

**Gambar 2.4** Format citra 24-bit (16 juta warna)

### 2.2.3 Format Citra JPEG

Menurut Daryanto (2007) dalam jurnal dengan judul "Analisa Perbandingan Standar-Standar Kompresi Pada Gambar". Joint Photograpic Experts (JPEG, dibaca *jay-peg*) di rancang untuk

kompresi beberapa *full-color* atau *gray-scale* dari suatu gambar yang asli, seperti pemandangan asli di dunia ini. JPEGs bekerja dengan baik pada *continous tone images* seperi *photographs* atau semua pekerjaan seni yang menginginkan yang nyata, tetapi tidak terlalu bagus pada ketajaman gambar dan seni pewarnaan seperti penulisan, kartun yang sederhana atau gambar yang mengunakan banyak garis. JPEG sudah mendukung untuk 24-bit *color depth* atau sama dengan 16,7 juta warna ($2^{24}$ = 16.777.216 warna). *Progressive JPEGs (p-JPEGs)* adalah tipe dari beberapa persen lebih kecil dibandingkan *baseline JPEGs* tetapi keuntungan dari JPEG dan tipe-tipenya telihat pada langkah-langkahnya sama seperti *interlaced GIFs*.

JPEG adalah algoritma kompresi secara *lossy*. JPEG bekerja dengan mengubah gambar spasial dan merepresentasikan kedalam pemetaan frekuensi *Discrete Cosine Transform (DCT)* dengan memisahkan antara informasi frekuensi yang rendah dan tinggi dari sebuah gambar. Informasi frekuensi yang tinggi akan diseleksi untuk dihilangkan yang terikat pada pengaturan kualitas yang digunakan. Waktu Kompresi dan dekompresi dilaksanakan dengan simetris. JPEG Group's (IJG) decoder lebih ditingkatkan kemampuannya dibadingkan dengan encodernya. Manakala, ketika diperlihatkan 8 bits, mengurangi kuantisasi warna yang lambat. Banyak para penjual JPEG menawarkan untuk mempercepat hasil dari JPEG, kuantisasi warna dan kualitas dengan mengimplementasikan IJG.

JPEG dirancang untuk mengeksploitasi tingkatan dari mata kita, yakni bahwa mata kita tidak akan dapat membedakan perubahan yang terang dan warna dibandingkan dengan perbedaan suatu jarak apakah jauh atau dekat. Untuk itu JPEG sangat baik digunakan pada fotografi dan monitor 80-bit. JPEG sebenarnya hanyalah algoritma kompresi, bukan merupakan nama format file. File yang biasa disebut JPEG pada jaringan sebenarnya adalah JFIF (*JPEG File Interchange Format*).

### 2.3 *Discrete Cosine Transform* (DCT)

Pada format gambar JPEG, masing-masing komponen warna menggunakan transformasi DCT (*discrete cosine transform*) untuk mentransformasi blok-blok gambar 8x8 pixel kedalam 64 masing-masing koefisien DCT.

Koefisien-koefisien DCT tersebut adalah *F(u,v)* dari suatu blok 8*x*8 dari citra piksel *f(x,y)* dinyatakan pada Persamaan 2.2 :

$$F(u,v) = \frac{2}{8} C(u)C(v) \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right] \quad (2.2)$$

Pada Persamaan 2.2, *F(u,v)*, berbentuk matriks 2-dimensi 8*x*8 dimana : *u, v, x, y* = 0, 1, 2,…,7

*x, y* adalah koordinat spatial dari domain asal.

*u,v* adalah koordinat frekuensi pada domain transformasi atau koefisien-koefisien DCT.

*C(u), C(v)* = 1/√2 , untuk u, v = 0

*C(u), C(v)* = 1 , untuk lainnya.

Dalam perhitungan DCT ini, apabila dilihat Persamaan 2.2 adalah :

1. Citra berupa blok array 2D, citra dengan domain spatial atau nilai piksel memiliki fungsi *f(x,y)*, sedangkan dalam domain frekuensi atau koefisien DCT memiliki fungsi *F(u,v)*.
2. Panjang dan lebar citra sama yaitu 8.
3. Looping untuk *x,y* = 0,1,2,…,7 dan looping *u,v* = 0,1,2,…,7.
4. Nilai *C(u), C(v)* = 1/√2, untuk *u,v* = 0 serta *C(u), C(v)* = 1, untuk lainnya.
5. Setelah itu dilakukan perhitungan seperti pada persamaan 2.2 maka akan didapat blok array baru dalam domain frekuensi atau koefisien DCT (Irianto, 2004).

**2.4 Kuantisasi**

Proses kuantisasi merupakan proses untuk mengurangi jumlah bit yang diperlukan dalam hal penyimpanannya. Proses kuantisasi diterapkan pada keluaran proses DCT. Kuantisasi dilakukan dengan cara membagi keluaran proses DCT dengan suatu nilai yang ditetapkan dalam matriks kuantisasi (Murinto, 2005).

Proses kuantisasi dilakukan dengan cara :

$$F_Q(u,v) = Round\left( \frac{F[u,v]}{q[u,v]} \right) \quad (2.3)$$

dimana :

(*q[u,v]*) = tabel kuantisasi.

Pemilihan tabel kuantisasi (*q[u,v)*) akan menentukan hasil kualitas kompresi dan dekuantisasi dari suatu citra. Dalam proses

dekompresi JPEG tiap elemen dari *FQ(u,v)* dikalikan dengan *q(u,v)* untuk mendapatkan kembali suatu pendekatan bagi *F(u,v)*, blok citra *f(i,j)* dapat dibalikkan kembali dengan menggunakan suatu invers 2-D DCI (IDCT). Adapun persamaannya adalah (Murinto, 2005) :

$$f(i,j) = \frac{1}{4} \sum_{u} \sum_{v} C(i,v) C(j,v) F(u,v) \qquad (2.4)$$

**2.5 *Embedding* dengan Algoritma F5**

Algoritma steganografi F5 diperkenalkan oleh peneliti dari Jerman yaitu Plitzman dan Westfeld pada tahun 2001. Tujuan dari penelitian mereka adalah untuk mengembangkan konsep-konsep dan mempraktekkan bagaimana proses embedding pesan untuk gambar format JPEG (Fridrich, dkk, 2002). Algoritma steganografi F5 dapat pula disebut sebagai metode substraksi. F5 mengurangi bit koefisien DCT dengan data pesan, F5 mengurangi nilai absolutnya dalam proses yang disebut matriks encoding. Menurut deskripsi dari algoritma F5 versi 11, program aplikasi menerima 5 inputan yaitu:

- Faktor kualitas dari *stego image* – Q;
- Input file (BMP, JPEG);
- Output file name;
- Password atau *stego-key*;
- Pesan yang akan disembunyikan

Matriks *encoding* menghitung kode hamming yang sesuai (1, ($2^k$-*1*), *k*) dengan menghitung ukuran blok pesan *k* dari panjang pesan dan jumlah koefisien-koefisien non DC yang tidak nol. Kode Hamming (1, $2^k$*-1*, *k*) mengkode pesan rahasia *k*-bit dari *m* kata pesan kedalam *n*-bit kata kode a dengan $n=2^k-1$. Dan dapat me-*recover* dari *single bit* yang *error* dalam kata kode.

F5 menggunakan fungsi decoding $f(a) = \oplus n_{i-1} a_i \cdot i$ dan jarak Hamming *d*. Dengan *matrix encoding*, *embedding* pesan *k*-bit ke dalam *n*-bit kata kode akan mengubahnya maksimum dengan satu bit. Dengan kata lain, dapat ditemukan kata kode yang sesuai *a'* untuk setiap kata kode *a* dan setiap kata pesan *m* sehingga m=*f*(*a'*) and $d(a, a') \leq 1$

Pertama, koefisien-koefisien DCT dipermutasi dengan kunci *pseudo-random number generator* (PRNG), lalu diatur kedalam kelompok-kelompok n, angka nol dan koefisien DC dilompati. Pesan tersebut kemudian dipecah menjadi blok-blok k-bit. Untuk setiap

blok pesan m, diperoleh kata kode n-bit dengan penggabungan LSB koefisien nilai absolut. Jika blok pesan m dan pengkodean *f(a)* adalah sama, maka blok pesan dapat diembed tanpa adanya perubahan, sebaliknya bila digunakan $s = m \oplus f(a)$ untuk menentukan koefisien mana yang perlu diubah (nilai absolutnya dikurangi dengan satu). Jika koefisien menjadi nol, akan terjadi pengerutan, dan dibuang dari kelompok koefisien. Kelompok tersebut diisi dengan koefisien yang bukan nol berikutnya dan proses berulang sampai semua pesan dapat diembed.

Untuk pesan-pesan yang lebih kecil, matrik *encoding* membuat F5 mengurangi jumlah perubahan pada gambar. Untuk contohnya, pada $k = 3$, tiap perubahan mengembed 3,43 bit pesan sedangkan total ukuran kode lebih dari dua kalinya. Karena F5 mengurangi koefisien DCT, jumlah koefisien yang berdekatan tidak lagi invariant, dan test $x^2$ tidak dapat mendeteksi pesan yang terembed secara F5 (Irianto, 2004).

Adapun algoritma F5 ditunjukkan pada Gambar 2.5.

**Input :** message, shared secret, cover image
**Output :** stego image
Inisialisasi PRNG dengan shared secret
Permutasi koefisien DCT dengan PRNG
Tentukan k dari kapasitas image
Hitung panjang kode word n ← $2^k$ - 1
**While** data left embed **do**
    Get next k-bit blok pesan
    **Repeat**
    G ← {n non zero koefisien AC}
    S ← k-bit hash f dari LSB pada G
    S ← s + k-bit blok pesan
    **If** s= 0 **then**
        Decrement nilai absolut koefisien DCT dari $G_s$
        Insert $G_s$ ke dalam stego image
    **End if**
    **Until** s = 0 atau $G_s$ = 0
    Insert koefisien DCT dari G kedalam stego image
**End while**

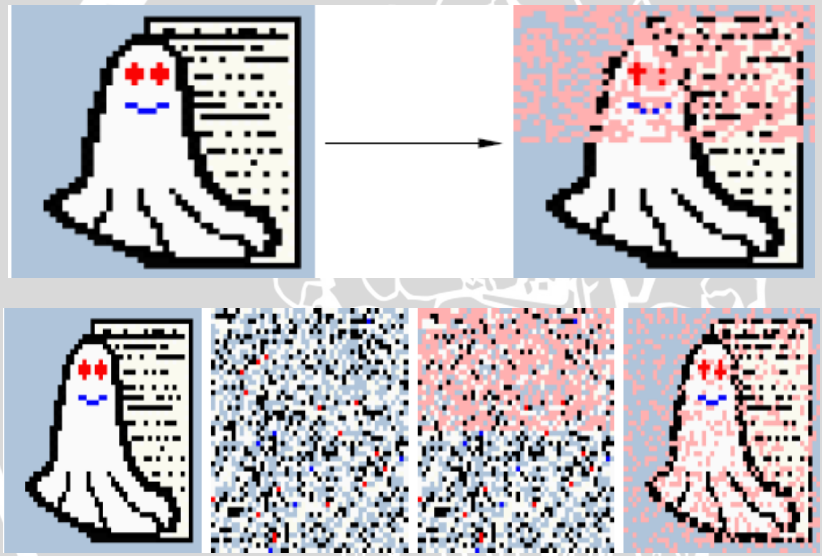
**Gambar 2.5** Algoritma F5 menggunakan substraksi dan matriks encoding untuk mengembed data ke dalam koefisien DCT (Provos & Honeyman).

*F5* adalah *Steganographic system* yang menawarkan kapasitas penyimpanan data yang besar. Dengan *F5*, penggunanya dapat memasukkan data yang berukuran sampai 13% dari ukuran gambar keseluruhan.

Selain itu *F5* menawarkan efisiensi yang sangat baik dalam perubahan bit. Efisiensinya dapat mencapai 3,7 bit untuk setiap perubahan. Dan jika *F5* diimplementasi tanpa *Matrix Encoding* efisiensinya dapat mencapai dua kali lipat.

Penggunaan *Matrix Encoding* membuat tes $\chi$2-test tidak dapat memeriksa keberadaan dari data yang disisipkan.

Implementasi *Permutative Straddling* menyebabkan penyisipan data yang "terlalu" banyak menyebabkan gambar digital "rusak" secara merata. Sehingga tidak mudah dikenali dengan melihatnya.



**Gambar 2.6** Gambar atas contoh dari continuous embedding tanpa *Permutative Straddling* menyebabkan gambar rusak pada bagian awalnya. sedangkan Gambar bawah dengan *Permutative Straddling* pada *F5* yang menyebabkan gambar menjadi "rusak" secara merata (Adythia, 2007).

*F5* merupakan penyempurnaan dari sistem *F4*. Pada pelaksanaan *F5*, permutasi dikenakan kata yang dimasukkan pengguna. Kemudian *Pseud one time pad* untuk distribusi pesan secara rata,

Matriks *Encoding* dengan *embedding rate* yang minimal. Terakhir dilakukan *Core Embedding* seperti pada *F4*.

## 2.6 Algoritma Huffman

Dalam ilmu komputer dan teori informasi, kode Huffman adalah algoritma pengkodean entropi untuk kompresi data *lossless*. Istilah ini merujuk kepada penggunaan tabel kode yang memiliki panjang bervariasi (*variable length code*) dimana tabel kode tersebut diturunkan dengan cara tertentu berdasarkan prakiraan probabilitas kemunculan setiap nilai dalam sumber data. Kode Huffman dikembangkan oleh David A. Huffman pada saat mengambil gelar Ph.D. di MIT.

Kode Huffman menggunakan metode spesifik untuk merepresentasikan setiap simbol, menghasilkan *prefix-free code* (*string* dari bit representasi sebuah simbol tidak pernah menjadi prefiks [awalan] dari sebuah simbol lain). Yang merepresentasikan karakter yang lebih sering muncul dengan *bit string* yang lebih pendek daripada karakter yang jarang muncul dalam suatu sumber data. Kompresi Huffman adalah metode paling efisien dari metode lain yang sejenis karena pemetaan lain simbol dari sumber data menjadi string unik menghasilkan file *output* yang lebih kecil ketika frekuensi simbol sesuai dengan frekuensi yang digunakan untuk menghasilkan kodenya. Kemudian ditemukan metode untuk melakukan pemetaan ini dalam waktu linear dengan mengurutkan probabilitas *input*. Untuk suatu set simbol dengan distribusi probabilitas tersebar dan jumlah elemen sebesar kelipatan pangkat dua, kode Huffman setara dengan *binary block encoding* sederhana. Penggunaan kode Huffman begitu luas sampai-sampai semua kode *prefix free* disinonimkan dengan Huffman walaupun algoritmanya tidak sesuai dengan Huffman.

Walaupun kode Huffman optimal untuk pengkodean simbol demi simbol dengan probabilitas distribusi input yang diketahui, kadang-kadang keefesienannya dilebih-lebihkan. Sebagai contoh kompresi hasil kode aritmatika dan LZW sering memiliki efisiensi yang lebih baik dari Huffman pada kasus probabilitas input tidak diketahui (Alam, 2008).

## 2.7 Penambahan *Image Distortion* pada *Stego Image*

Penilaian sebuah algoritma steganografi yang baik dapat dinilai dari beberapa faktor. Salah satunya adalah *robustness* yaitu data yang disembunyikan harus tahan (*robust*) terhadap berbagai operasi manipulasi yang dilakukan pada citra penampung, seperti pengubahan kontras, penajaman, pemampatan, rotasi, perbesaran gambar, pemotongan (*cropping*), diputar (*flip*) dan sebagainya. Bila pada citra penampung dilakukan operasi-operasi pengolahan citra tersebut, maka data yang disembunyikan seharusnya tidak rusak (tetap valid jika diekstraksi kembali).

Oleh karena itu untuk menguji ketahanan (*robustness*) pesan yang disembunyikan dalam citra maka akan dilakukan beberapa operasi manipulasi pengolahan citra terhahap citra penampung atau *cover image* tersebut. Operasi pengolahan citra atau distorsi yang akan dilakukan yaitu :

Efek *Gaussian* Blur

*Gaussian* blur yaitu perubahan warna tiap-tiap *pixel* dalam citra agar membentuk distribusi normal.

*Mask* yang digunakan pada *Gaussian* blur memiliki nilai seragam untuk semua bobot dan berbentuk piramida atau *Gaussian*. Bobot pada *mask Gaussian* blur mengikuti distribusi normal sebagaimana dinyatakan dalam persamaan (2.5). σ adalah nilai deviasi standar distribusi normal yang digunakan. Makin besar nilai σ makin banyak titik tetangga yang diikutkan dalam perhitungan.

$$f(u,v) = G(u,v) = \frac{1}{2\pi\sigma^2} e\left(-\left[\frac{u^2+v^2}{2\sigma^2}\right]\right) \quad (2.5)$$

dimana : $u,v = 0,1,2,\ldots,N-1$

$\sigma$ = standar deviasi (Achmad dan Kartika, 2005).

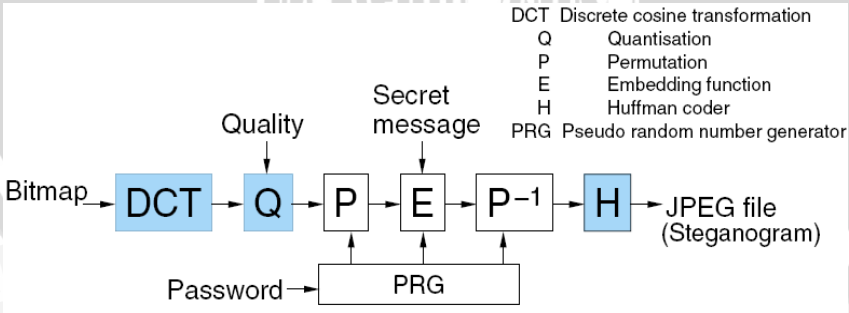
# BAB III
# METODOLOGI PENELITIAN

Pada dasarnya, proses penyembunyian pesan rahasia dalam sistem steganografi dimulai dengan identifikasi bit-bit *redundant* dari cover mediumnya (yang dapat dimodifikasi tanpa merusak integritas medium yang bersangkutan). Proses *embedding* menggunakan algoritma F5 menghasilkan suatu *medium stego* (dalam kasus ini medium tersebut berupa *cover image*) melalui penyisipan bit data pesan dalam koefisien DCT kemudian membuat matriks *encoding* untuk mengurangi atau meminimalkan jumlah perubahan-perubahan yang diperlukan untuk menyisipkan suatu pesan dengan panjang tertentu sehingga menghasilkan sebuah *stego image*. Penambahan *stego key* atau *shared secret* digunakan untuk memperkuat pengamanan data pada data rahasia.

Dalam bab ini akan dibahas mengenai dua proses utama dalam teknik steganografi yaitu proses penyisipan atau *embedding* pesan dalam citra dan proses penguraian atau *extraction* pesan, serta membahas tentang penjelasan mengenai distorsi atau operasi manipulasi pengolahan citra yang akan diberikan pada citra yang telah disisipkan pesan Dan proses pengujian kualitas dan ketahanan terhadap *stego image*.

## 3.1. Proses Penyisipan/*Embedding* Pesan dalam Citra

Proses penyisipan pesan dalam citra mengunakan algoritma F5 dapat dilihat dalam Gambar 3.1.




**Gambar 3.1** Proses Penyisipan Pesan Dalam Citra (Westfeld, 2001)

Dalam proses penyisipan pesan ini menggunakan algoritma F5, langkah-langkah penyisipan pesan dengan menggunakan algoritma F5 (sesuai dengan Bab 2 Gambar 2.5) yaitu pertama inisialisasi PRNG (*pseudo-random number generator*) dengan *shared secret* atau *stego key*. Kemudian dapatkan koefisien-koefisien DCT atau domain frekuensi dari citra. Untuk mengubah citra menjadi koefisien-koefisien DCT atau domain frekuensi menggunakan perhitungan DCT seperti pada Bab 2 Persamaan 2.2.

Hasil transformasi citra dengan DCT akan menghasilkan nilai positif dan negatif. Oleh karena itu nilai hasil transformasi akan diabsolutkan hal ini disebut proses kuantisasi (sesuai pada Bab 2 Persamaan 2.3). Komponen DC dari koefisien DC (daerah [0,0] – *zero frequency*) akan selalu bernilai lebih tinggi dari nilai koefisien yang lain.

Dalam proses *embedding* pesan dalam citra menggunakan algoritma F5 ini pertama-tama dimasukkan (*input*) file citra. Setelah itu masukkan *password* atau *shared secret* sebagai kunci steganografi. Setelah itu ambil *pixel* dari citra tersebut, kemudian lakukan perhitungan DCT (sesuai dengan Persamaan 2.2 pada Bab 2) untuk memperoleh koefisien DCT atau domain frekuensinya serta hasil tabel kuantisasinya. Inisialisasi PRNG dengan *password* yang telah dimasukkan. Setelah itu dilakukan permutasi pada koefisien DCT, lalu dilakukan proses *embedding* bit data pesan ke dalam koefisien DCT yang telah dipermutasikan. Setelah proses *embedding* selesai, lakukan invers permutasi terhadap koefisien DCT yang telah disisipkan pesan. Sehingga didapatkan hasil berupa *stego image*. Selanjutnya sesuai pada Gambar 3.1 maka dilakukan *huffman coding* untuk mengkompresi citra sehingga output citra atau *stego image* merupakan citra dengan format JPEG (*.jpg).

Berikut ini contoh *embedding* atau proses *encoding* pada citra digital dengan menggunakan algoritma F5. Citra dengan ukuran 8x8 piksel dengan masing-masing nilai pikselnya adalah sebagai berikut :

| Pixel | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 68 | 105 | 101 | 94 | 95 | 95 | 104 | 97 |
| 1 | 74 | 116 | 117 | 112 | 110 | 107 | 116 | 111 |
| 2 | 64 | 108 | 112 | 108 | 107 | 104 | 112 | 107 |
| 3 | 69 | 106 | 106 | 101 | 102 | 101 | 108 | 99 |
| 4 | 62 | 108 | 105 | 103 | 108 | 112 | 115 | 118 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 82 | 146 | 147 | 148 | 152 | 156 | 147 | 160 |
| 6 | 96 | 166 | 172 | 172 | 174 | 176 | 166 | 172 |
| 7 | 95 | 170 | 176 | 170 | 171 | 178 | 172 | 178 |

Setelah didapatkan nilai masing-masing pikselnya, buat koefisien nilai DCT untuk masing-masing nilai piksel. Contoh perhitungan untuk mencari nilai koefisien DCT sesuai dengan Bab 2 Persamaan 2.2 :

$$F(0,0) = \frac{2}{8} * C(0) * C(0) * \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2*0+1)*0*\frac{22}{7}}{2*8} \cos \frac{(2*0+1)*0*\frac{22}{7}}{2*8} \right]$$

$$F(0,0) = \frac{2}{8} * \frac{1}{\sqrt{2}} * \frac{1}{\sqrt{2}} * \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2*0+1)*0*\frac{22}{7}}{2*8} \cos \frac{(2*0+1)*0*\frac{22}{7}}{2*8} \right]$$

$$F(0,0) = 976{,}12$$

$$F(0,1) = \frac{2}{8} * C(0) * C(1) * \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2*0+1)*0*\frac{22}{7}}{2*8} \cos \frac{(2*1+1)*1*\frac{22}{7}}{2*8} \right]$$

$$F(0,1) = \frac{2}{8} * \frac{1}{\sqrt{2}} * 1 * \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2*0+1)*0*\frac{22}{7}}{2*8} \cos \frac{(2*1+1)*1*\frac{22}{7}}{2*8} \right]$$

$F(0,1) = -191{,}26$ … dan seterusnya.

Sehingga didapatkan hasil koefisien DCT citra dengan ukuran 8x8 piksel adalah sebagai berikut :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 976,12 | −76,79 | −61,25 | −63,11 | −56,38 | −41,20 | −25,37 | 10,48 |
| −191,26 | 32,09 | 24,48 | 14,11 | 8,23 | 17,67 | −4,05 | 9,86 |
| 86,17 | −1,827 | −7,83 | −7,49 | −6,29 | −3,79 | −0,03 | −1,79 |
| −7,51 | −5,354 | 4,12 | 3,34 | 3,38 | −4,837 | 1,21 | −3,03 |
| −53,13 | 5.04 | 11,53 | 6,48 | −1,63 | 3,564 | −0,58 | 2,16 |
| 3,98 | 6,62 | −4,36 | 2,91 | 0,09 | 1,79 | −0,16 | 0,99 |
| −2,00 | −5,22 | 0,21 | −2,33 | −0,50 | −2,49 | 0,83 | −1,19 |
| −16,36 | −3,89 | 0,53 | −0,01 | 0,16 | 0,02 | −0,38 | 0,26 |

Algoritma F5 ini menggantikan LSB (jika koefisien (d) lebih besar dari 0 (d>0) maka LSB(d) = d mod 2 dan jika d<0 maka LSB(d) = 1-d mod 2) setelah itu ganti bit-bit LSB dengan representasi biner data pesan. Kemudian F5 akan mengurangi nilai koefisien DCT dengan bit data pesan. Jika nilai kuantisasi koefisien DCT adalah 0 maka lompati (*skip*) ke nilai kuantisasi koefisien DCT selanjutnya tetapi apabila nilai bit data pesan = 1 maka kurangi nilai kuantisasi koefisien DCT dengan 1, sedangkan apabila nilai bit data pesan = 0 maka tambahkan nilai kuantisasi koefisien DCT dengan 1. Dan ulangi penyisipan bit data pesan pada 1 blok (8*x*8 piksel) dan jika 1 blok telah selesai disisipkan maka lanjutkan pada blok selanjutnya (Westfeld, 2001).

Kemudian buat tabel kuantisasi sesuai dengan Bab 2 Persamaan 2.3 terhadap koefisien DCT tersebut. Sehingga didapatkan hasil tabel kuantisasinya adalah sebagai berikut :

| Pixel | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 122 | -5 | -3 | -3 | -2 | -2 | -1 | 0 |
| 1 | -12 | 2 | 1 | 1 | 0 | 1 | 0 | 0 |
| 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Misal pesan yang akan disisipkan adalah huruf A dengan representasi biner A = 01000001 maka akan dihasilkan.

| 122 | -5 | -3 | -3 | -2 | -2 | -1 | 0 |
|---|---|---|---|---|---|---|---|
| ↓ 0 | ↓ 1 | ↓ 0 | ↓ 0 | ↓ 0 | ↓ 0 | ↓ 0 | SKIP |
| 123 | -6 | -4 | -2 | -1 | -1 | 0 | 0 |

-12

↓ 1

-13

Ket :

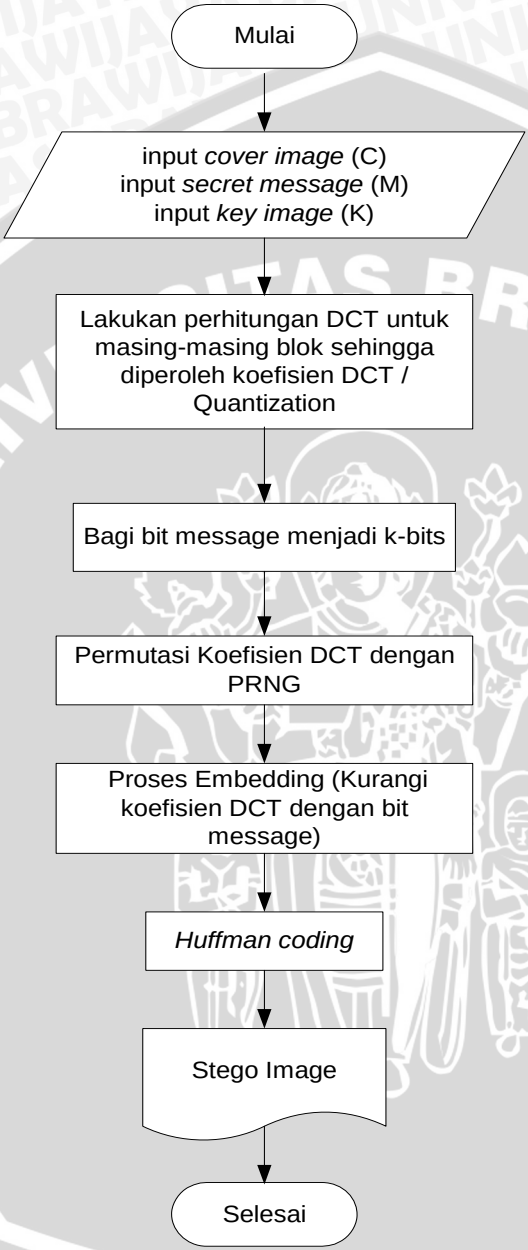
1. *Shrinkage* atau *skip* jika nilai kuantisasi koefisien DCT = 0.
2. Apabila nilai bit data pesan = 1 maka kurangi nilai kuantisasi koefisien DCT dengan 1.
3. Apabila nilai bit data pesan = 0 maka tambahkan nilai kuantisasi koefisien DCT dengan 1.

Maka hasil kuantisasi koefisien DCT setelah dikurangi dengan bit data pesan adalah sebagai berikut :

| Pixel | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 123 | -6 | -4 | -2 | -1 | -1 | 0 | 0 |
| 1 | -13 | 2 | 1 | 1 | 0 | 1 | 0 | 0 |
| 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Dan setelah itu lakukan invers DCT untuk mendapatkan koefisien DCT atau domain spatialnya sesuai dengan Bab 2 Persamaan 2.4.

Proses penyisipan data dalam citra dapat digambarkan dengan diagram alir pada Gambar 3.2 :




**Gambar 3.2** Diagram Alir Proses Penyisipan Pesan Dalam Citra

Berikut ini penjelasan diagram alir penyisipan dalam citra sesuai dengan Gambar 3.2 :

1. Masukan citra yang akan disisipkan, citra ini dapat berupa citra dengan format bmp dan jpg. Kemudian masukan file data teks yang akan disisipkan dan *key* yang berlaku sebagai kunci steganografi dalam aplikasi ini.
2. Bagi citra input menjadi blok-blok 8x8 *pixel*, setelah itu lakukan perhitungan DCT sesuai pada Bab 2 Persamaan 2.2 sehingga akan didapatkan koefisien DCT. Dan lakukan perhitungan kuantisasi sesuai pada Bab 2 Persamaan 2.3
3. Bagi bit data pesan menjadi grup-grup *k*-bit.
4. Lakukan permutasi koefisien DCT tersebut dengan PRNG (*Pseudo Random Number Generator*).
5. Proses *Embedding* yaitu kurangi nilai kuantisasi koefisien DCT dengan bit data pesan.
6. Lakukan kompresi pada citra tersebut dengan menggunakan kompresi Huffman.
7. Setelah proses *embedding* data selesai akan dihasilkan sebuah citra stego (*stego image*) dengan format jpg (*.jpg).

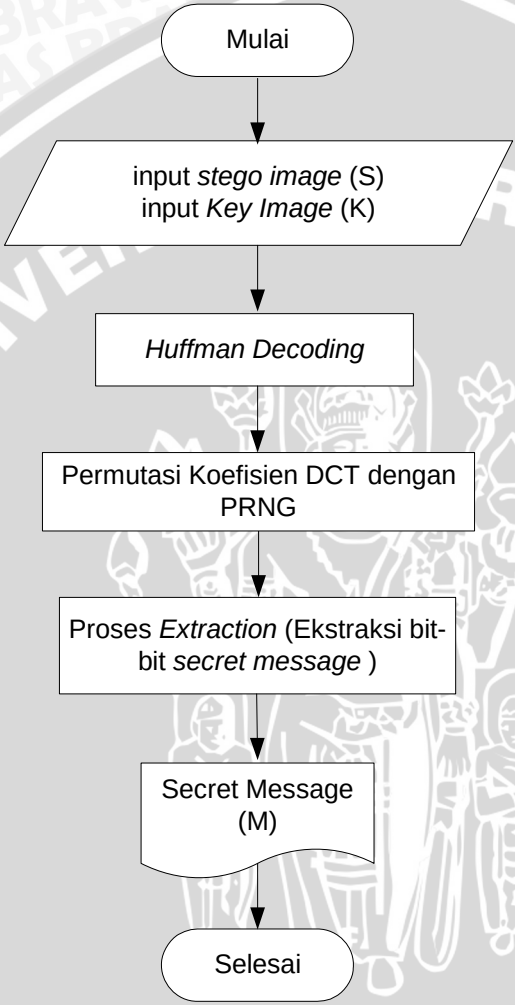
### 3.2. Proses Penguraian/*Extraction* Pesan

Pada dasarnya proses penguraian pesan atau *extraction* adalah proses pembalikan dari proses penyisipan pesan dalam citra atau embedding.

Proses penguraian pesan pada stego image dengan algoritma F5 ini sesuai dengan Gambar 3.3 yaitu :

1. Masukkan s*tego image* dan *key*.
2. Kemudian lakukan dekompres pada citra stego tersebut dengan menggunakan *Huffman decoder*.
3. Lakukan permutasi koefisien DCT dengan PRNG.
4. Setelah itu lakukan proses ekstraksi pesan dalam citra stego tersebut.
5. Sehingga didapatkan pesan rahasia tersebut.

Proses penguraian pesan dapat digambarkan dengan diagram alir Gambar 3.3.




**Gambar 3.3** Diagram Alir Proses Penguraian Pesan

### 3.3 Perancangan Uji Coba

Pengujian yang akan dilakukan dalam sistem ini dilihat dari kualitas (*fidelity*) citra *stego* digital yang dihasilkan dibandingkan dengan citra asli, dan ketahanan (*robutsness*) citra *stego* setelah dilakukan operasi manipulasi pengolahan citra atau distorsi. Dalam uji ketahanan ini, citra *stego* akan diberikan distorsi seperti pada subbab 3.3. Pengujian ketahanan ini dilakukan untuk mengetahui apakah data yang telah disisipkan dapat diekstrak kembali atau tidak.

#### 3.3.1 Citra Uji

Spesifikasi citra yang akan diujikan yaitu:

1. Citra dengan format BMP, JPEG.
2. Terdiri dari citra RGB 24 bit.
3. 2 citra uji untuk proses steganografi, dengan komposisi 1 citra format BMP dan 1 citra format JPEG.
4. Banyaknya citra uji yang digunakan untuk uji kualitas adalah 60 dengan komposisi : 30 format BMP dan 30 format JPEG.
5. Pesan yang digunakan yaitu pesan ukuran 648 *Bytes*, 368 *Bytes*, 3,79 *KB*, 6,80 *KB*, 8,64 *KB*.
6. Satu citra stego yang digunakan untuk uji ketahanan.

#### 3.3.2 Pengujian Kualitas (*Fidelity*) *Stego Image*

Pada pengujian kulitas citra *stego* ini akan menggunakan *Root Mean Square (*RMS*)*. RMS akan memandang citra asli sebagai *f(x,y)* sedangkan citra *stego* sebagai *g(x,y)*, dengan ukuran citra *M x N*.

Pengukuran RMS menunjukan rata-rata perbedaan antara intensitas citra asli *f(x,y)* dan intensitas citra *stego g(x,y)*. Dalam perhitungan RMS ini, pertama-tama citra input akan dijadikan citra *grayscale* yaitu masing-masing nilai RGB pada satu piksel ditambahkan dan dihitung rata-ratanya. RMS dalam tugas akhir ini dinyatakan dalam satuan persen, seperti persamaan berikut (Sulistyanto, 2002) :

$$RMS(\%) = \sqrt{\frac{1}{MxN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [g(x,y) - f(x,y)]^2} \; x\,100\% \qquad (3.1)$$

Ket : $M$ = panjang citra, $N$ = lebar citra

Citra yang akan diuji yaitu 30 citra asli format BMP dan 30 citra asli format JPEG kemudian citra-citra tersebut disisipkan pesan dengan ukuran-ukuran yang berbeda sesuai dengan ukuran atau size

dari citra. Setelah itu citra akan diuji kualitasnya dengan menggunakan RMS (Persamaan 3.1). Sehingga bentuk tabel pengujian akan disajikan sebagai berikut:

**Tabel 3.1** Tabel Hasil Uji Kualitas Citra *Stego*

| No. | Nama Citra | Dimensi (*Pixel*) | Ukuran Pesan (KB) | | RMS (%) | |
|---|---|---|---|---|---|---|
| | | | BMP | JPEG | BMP-JPG | JPG-JPG |
| | | | | | | |

Hasil pengujian ini akan menunjukkan kualitas citra stego dan hasil pengujian tersebut dapat menjadi acuan atau ukuran apakah citra format BMP atau format JPEG yang merupakan citra yang lebih baik digunakan untuk menyisipkan pesan dengan algoritma F5.

### 3.3.3 Pengujian Ketahanan (*Robustness*) *Stego Image* terhadap *Image Distortion*

Sebagaimana telah disebutkan sebelumnya bahwa sebuah algoritma stegenografi harus memenuhi empat faktor. Apabila ketiga faktor telah dipenuhi yaitu *imperceptible, fidelity, recovery* maka akan dilanjutkan dengan pengujian ketahanan terhadap *stego image* (*Robustness*). Untuk menguji *stego image* tersebut maka akan dilakukan beberapa operasi manipulasi pengolahan citra (distorsi).

Dalam subbab ini akan dibahas tentang pengujian mutu berkas citra yang telah disisipi pesan (*stego image*). Apakah mengalami penurunan kualitas atau tidak, dan apakah informasi masih dapat diperoleh dari citra yang telah diubah-ubah tersebut.

**Gambar 3.4** Gambar Stego yang digunakan

### 3.3.3.1 Penambahan Efek *Gaussian Blur*

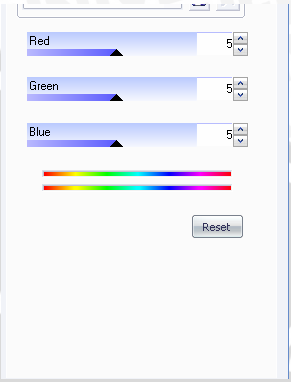
Pengujian kemudian dilanjutkan untuk melihat apakah pesan yang disisipkan masih dapat diekstrak meskipun *stego image* mengalami beberapa perubahan. Dalam pembuatan operasi manipulasi pengolahan citra ini menggunakan *software ACDSee 9 Photo Manager*. Pengujian yang dilakukan yaitu penambahan efek *Gaussian blur* (sesuai pada Bab 2 Persamaan 2.5) :

**Gambar 3.5** Citra diberi efek *Gaussian Blur*

Maka setelah diberi penambahan efek *gaussian blur* maka citra stego tersebut akan diekstraksi. Kemudian nilai piksel citra stego dengan penambahan efek *gaussian blur* akan dibandingkan dengan nilai piksel citra stego tanpa penambahan efek. Hal ini dimaksudkan untuk mengetahui apakah terjadi perubahan nilai piksel atau warna antara kedua citra stego tersebut.

### 3.4.3.2 Penambahan Intensitas Warna

Kemudian untuk mengetahui apakah citra stego dapat tahan atau *robust* dengan penambahan intensitas warna (hal ini dimaksudkan untuk mengetahui apakah pesan dalam citra stego dapat diekstrak kembali jika nilai pikselnya diubah) maka dengan menggunakan *ACDSee 9 Photo Manager* nilai RGB dari citra stego akan dtambah. Berikut ini penambahan intensitas warna pada citra stego :




**Gambar 3.6** Citra stego dengan penambahan RGB sebanyak 5.

Dan berikut perbandingan nilai piksel citra stego tanpa penambahan intensitas warna dengan citra stego dengan penambahan RGB sebanyak 5 :

**Tabel 3.2** Nilai *pixel* citra stego tanpa penambahan RGB dan citra stego dengan penambahan RGB sebanyak 5.

| Koordinat | | Sebelum pic1hasilsteg.jpg | | | Sesudah pic1hasilstegRGB5r.jpg | | |
|---|---|---|---|---|---|---|---|
| X | Y | R | G | B | R_ | G_ | B_ |
| 0 | 0 | 135 | 136 | 102 | 139 | 145 | 109 |
| 1 | 0 | 80 | 81 | 49 | 80 | 85 | 53 |
| 2 | 0 | 24 | 24 | 0 | 22 | 26 | 0 |
| . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . |
| 147 | 149 | 18 | 8 | 0 | 17 | 4 | 2 |
| 148 | 149 | 12 | 0 | 6 | 12 | 0 | 3 |
| 149 | 149 | 15 | 0 | 0 | 17 | 0 | 0 |

Kemudian citra stego tersebut akan diekstraksi apakah pesan citra stego dengan penambahan intensitas warna masih valid atau tidak rusak (pesan dapat diekstraksi).

# BAB IV
# HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas mengenai implementasi dari representasi rancangan ke bahasa pemrograman yang dimengerti oleh komputer. Bab ini akan membahas hasil implementasi yang dihasilkan oleh perangkat lunak, yang meliputi proses penyisipan pesan/*embedding*, proses menguraikan kembali pesan yang telah disisipkan/*extraction* dan untuk selanjutnya dilakukan evaluasi terhadap kualitas citra serta evaluasi uji ketahanan *stego image* dengan menambahkan *image distortion*.

## 4.1 Lingkungan Implementasi

Lingkungan implementasi meliputi lingkungan perangkat keras serta perangkat lunak.

### 4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem pembuatan steganografi dan uji coba adalah :

1. Genuine Intel (R) CPU T2050 1.60GHz.
2. RAM 1024MB.
3. Harddisk dengan kapasitas 80GB.
4. Monitor.
5. Keyboard.
6. Mouse.

### 4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem pembuatan steganografi dan uji coba adalah :

1. Sistem Operasi Microsoft Windows XP Professional.
2. Pemrograman Java dengan Installer J2SDK 1.5 dan editor Jcreator Pro v3.10.008.
3. Microsoft Visual C# 2005 Express Edition.
4. *ACD See 9 Photo Manajer*.

### 4.2 Implementasi Program

Pada bab ini akan dibahas mengenai implementasi dari sistem pembuatan steganografi.

#### 4.2.1 Input Citra

Proses input yang dilakukan user berupa citra digital dengan format bmp (*.bmp) dan jpg (*.jpg). Kemudian hasil citra steganografi atau *stego image* akan disimpan dengan format jpg (*.jpg).

```
// Author: James R. Weeks and BioElectroMech, 1998.

if (!haveInputImage)
{
   if (!args[i].endsWith(".jpg") &&
       !args[i].endsWith(".bmp"))
      StandardUsage();
      inFileName = args[i];
       outFileName = args[i].substring(0,
                     args[i].lastIndexOf(".")) + ".jpg";
       haveInputImage=true; }
else {
  outFileName = args[i];
  if (outFileName.endsWith(".bmp"))
      outFileName = outFileName.substring(0,
                    outFileName.lastIndexOf("."));
  if (!outFileName.endsWith(".jpg"))
      outFileName = outFileName.concat(".jpg"); }
```

***Sourcecode* 4.1** *Sourcecode* membuka file citra

#### 4.2.2 Perhitungan *Discrete Cosine Transform* (DCT)

Proses perhitungan DCT ini pertama-tama bagi citra menjadi blok-blok 8x8 *pixel*.

```
// Copyright (C)1998, James R. Weeks and BioElectroMech.

// set minimum blok panjang dan minimum blok lebar
MinBlockWidth = ((imageWidth%8 != 0) ?
                (int) (Math.floor((double)
                imageWidth/8.0) + 1) * 8 : imageWidth);
MinBlockHeight = ((imageHeight%8 != 0) ?
                 (int) (Math.floor((double)
                  imageHeight/8.0) + 1)*8: imageHeight);
```

***Sourcecode* 4.2** *Sourcecode* set minimum blok panjang dan blok lebar

Kemudian dilakukan perhitungan DCT untuk mendapatkan koefisien DC sesuai dengan Persamaan 2.2.

```
// Copyright (C)1998, James R. Weeks and BioElectroMech.

for (v = 0; v < 8; v++)
{
   for (u = 0; u < 8; u++)
   {
      for (x = 0; x < 8; x++)
      {
         for (y = 0; y < 8; y++) {
            output[v][u]+=((double)input[x][y])
                          *Math.cos
                         (((double)(2*x+1) *
                         (double)u*Math.PI)/
                         (double)16)*Math.cos
                         (((double)(2*y + 1)
                         (double)v *Math.PI) /
                         (double)16); } }
      output[v][u] *= (double)(0.25)*((u == 0) ?
                      ((double)1.0/Math.sqrt(2)) :
                      (double) 1.0) * ((v==0) ?
                      ((double)1.0/Math.sqrt(2)) :
                      (double) 1.0);  }  }
```

***Sourcecode* 4.3** *Sourcecode* perhitungan DCT

### 4.2.3 Perhitungan *Quantization*

*Quantization* dilakukan untuk mendapatkan blok-blok koefisien DC dengan bilangan bulat atau integer yang terdekat.

```
// Copyright (C)1998, James R. Weeks and BioElectroMech.

index = 0;
for (i = 0; i < 8; i++) {
   for (j = 0; j < 8; j++) {
     // untuk membulatkan blok-blok koefisien DC
        pada integer yang terdekat
     outputData[index]=(int)
                       (Math.round(inputData[i][j] /
                       (double)(((int[])
                       (quantum[code]))[index])));
     index++;  }
}
```

***Sourcecode* 4.4** *Sourcecode* perhitungan *Quantization*

### 4.2.4 Permutasi Koefisien DCT dengan PRNG

Setelah didapatkan koefisien DCT dan melakukan *quantization* maka akan dilakukan permutasi koefisien DCT dengan PRNG.

```
// Author: James R. Weeks and BioElectroMech, 1998.

// untuk membuat shuffled sequence, inisialisasi array
bertipe integer dari 0 ... (size-1).
for (i = 0; i < size; i++) // inisialisasi  size
  shuffled[i] = i;
  int maxRandom = size; // set angka masukan untuk
                        shuffle
  for (i = 0; i < size; i++)
  { // shuffle entries
    randomIndex = random.getNextValue(maxRandom--);
    tmp = shuffled[randomIndex];
    shuffled[randomIndex] = shuffled[maxRandom];
    shuffled[maxRandom] = tmp; }
```

***Sourcecode* 4.5** *Sourcecode* Permutasi Koefisien DCT dengan PRNG

### 4.2.5 *Embedding*

Proses *embedding* yaitu proses untuk menyisipkan pesan kedalam citra yang telah diinputkan. Pada proses ini pertama-tama inisialisasi PRNG dengan *shared secret* atau *stego key*. Kemudian permutasi koefisien DCT dengan PRNG. Kemudian kurangi nilai absolut koefisien DCT (tabel kuantisasi) dengan bit data pesan.

```
// Copyright (C)1998, James R. Weeks and BioElectroMech.
// embed status word first
for(i=0; i<coeffCount; i++) {
   shuffledIndex = permutation.getShuffled(i);
   if (shuffledIndex%64 == 0) continue; // skip DC
                                        coefficients
   if (coeff[shuffledIndex] == 0) continue; // skip
                                            zeroes
   // rev. 12: skip every second 1 or -1
   if (Math.abs(coeff[shuffledIndex]) == 1)
      if (((++one)&1) == 0) continue;
      was2 = (Math.abs(coeff[shuffledIndex]) == 2);

   if (coeff[shuffledIndex] > 0)  {
     if ((coeff[shuffledIndex]&1) != nextBitToEmbed) {
       coeff[shuffledIndex]--; // decrease absolute
                               value
       _changed++;  }
   } else {
       if ((coeff[shuffledIndex]&1) == nextBitToEmbed) {
```

```
          coeff[shuffledIndex]++; // decrease absolute
                                     value
          _changed++; }
        }
        // rev. 12: skip every second 1 or -1
        if (was2 && (Math.abs(coeff[shuffledIndex]) == 1))
          if (((++one)&1) == 0) continue;
          if (coeff[shuffledIndex] != 0) {
          // The coefficient is still nonzero.
          // successfully embedded "nextBitToEmbed".
          // read a new bit to embed now.
          if (availableBitsToEmbed==0)
             break;     // statusword embedded.
             nextBitToEmbed = byteToEmbed & 1;
             byteToEmbed >>= 1;
             availableBitsToEmbed--;
            _embedded++;
          } else {
             _thrown++;
             one--;
           }
    }

embeddingLoop:
do {
    kBitsToEmbed = 0;
    // get k bits to embed
    for (i=0; i<k; i++) {
      if (availableBitsToEmbed==0)
      // If the byte of embedded text is  empty, get a
         new one.
      try {
        if (embeddedData.available()==0) {
          isLastByte = true;
          break;
        }
        byteToEmbed = embeddedData.read();
        byteToEmbed ^= random.getNextByte();
      } catch (Exception e) {
        e.printStackTrace();
        break;
        }
      availableBitsToEmbed=8;
     }
     nextBitToEmbed = byteToEmbed & 1;
     byteToEmbed >>= 1;
     availableBitsToEmbed--;
     kBitsToEmbed |= nextBitToEmbed << i;
```

```
      _embedded++;
      }
      // embed k bits
      do {
        j = startOfN;
        one = startOne;
        // fill codeWord[] with the indices of the
        // next n non-zero coefficients in coeff[]
        for (i=0; i<n; j++) {
          if (j>=coeffCount) {
          // in rare cases the estimated capacity is too
              small
            System.out.println("Capacity exhausted.");
            break embeddingLoop;
          }
          shuffledIndex = permutation.getShuffled(j);
          if (shuffledIndex%64 == 0) continue; //skip DC
                                                coefficients
          if (coeff[shuffledIndex] == 0) continue;
          // skip zeroes


        // rev. 12: skip every second 1 or -1
        if (Math.abs(coeff[shuffledIndex]) == 1) {
           if ((++one&1) == 0) continue;
        }

        codeWord[i++]=shuffledIndex;
      }
      endOfN = j;
      hash = 0;
      for (i=0; i<n; i++) {
      if (coeff[codeWord[i]] > 0) {
        extractedBit = coeff[codeWord[i]]&1;
      } else {
        extractedBit = 1-(coeff[codeWord[i]]&1);
        }
      if (extractedBit == 1) {
        hash ^= i+1;
      }
    }
    i = hash ^ kBitsToEmbed;
    if (i==0) break; // embedded without change
    i--;
    if (coeff[codeWord[i]]>0) {
       coeff[codeWord[i]]--;
    } else {
      coeff[codeWord[i]]++;
```

```
      }
      _changed++;
      if (coeff[codeWord[i]]==0) {
        _thrown++;
      }
    } while (true);
    startOfN = endOfN;
    startOne = one;
} while (!isLastByte);
```

```
else { // default code
       // The main embedding loop follows. It works on
          the shuffled stream of coefficients.
for(i=0; i<coeffCount; i++) {
  shuffledIndex = permutation.getShuffled(i);
  if (shuffledIndex%64 == 0) continue; // skip DC
                                       coefficients
  if (coeff[shuffledIndex] == 0) continue; // skip
                                           zeroes

  // rev. 12: skip every second 1 or -1
  if (Math.abs(coeff[shuffledIndex]) == 1)
    if ((++one&1) == 0) continue;
  was2 = (Math.abs(coeff[shuffledIndex]) == 2);
  _examined++;
  if (coeff[shuffledIndex] > 0) {
    if ((coeff[shuffledIndex]&1) != nextBitToEmbed) {
    coeff[shuffledIndex]--; // decrease absolute value
    _changed++;
  }
} else {
if ((coeff[shuffledIndex]&1) == nextBitToEmbed) {
   coeff[shuffledIndex]++; // decrease absolute value
   _changed++; }
}

// rev. 12: skip every second 1 or -1
if (was2 && (Math.abs(coeff[shuffledIndex]) == 1))
   if (((++one)&1) == 0) continue;

if (coeff[shuffledIndex] != 0) {
// The coefficient is still nonzero.
// successfully embedded "nextBitToEmbed".
// read a new bit to embed now.
  if (availableBitsToEmbed==0) {
  // If the byte of embedded text is empty, get a new
     one.
     try {
```

```
        if (embeddedData.available()==0) break;
        byteToEmbed=embeddedData.read();
        byteToEmbed ^= random.getNextByte();
        } catch (Exception e) {
          e.printStackTrace();
          break; }
        availableBitsToEmbed=8;
      }
      nextBitToEmbed = byteToEmbed & 1;
      byteToEmbed >>= 1;
      availableBitsToEmbed--;
      _embedded++;
    } else {
        _thrown++;
        one--;  } // shrinkage occurred
}
```

***Sourcecode* 4.6** *Sourcecode Embedding*

### 4.2.6 *Huffman Coding*

Setelah citra disisipkan pesan maka citra harus dikompresi menggunakan kompresi *Huffman coder* sehingga citra stego atau *stego image* yang dihasilkan akan berformat jpg (*.jpg).

```
// Copyright (C)1998, James R. Weeks and BioElectroMech.
/** HuffmanBlockEncoder run length encodes and Huffman
encodes the quantized data. **/
public void HuffmanBlockEncoder(BufferedOutputStream
       outStream, int zigzag[], int prec, int DCcode,
       int ACcode)
{
   int temp, temp2, nbits, k, r, i;
   NumOfDCTables = 2;
   NumOfACTables = 2;
   // The DC portion
   temp = temp2 = zigzag[0] - prec;
   if(temp < 0)
   {
     temp = -temp;
     temp2--;
   }
   nbits = 0;
   while (temp != 0)
   {
      nbits++;
      temp >>= 1;
   }
   bufferIt(outStream,
```

```
            ((int[][])DC_matrix[DCcode])[nbits][0],
            ((int[][])DC_matrix[DCcode])[nbits][1]);
   // The arguments in bufferIt are code and size.
   if (nbits != 0)
   {
      bufferIt(outStream, temp2, nbits);
   }
   // The AC portion
   r = 0;
   for (k = 1; k < 64; k++)
   {
      if ((temp = zigzag[jpegNaturalOrder[k]]) == 0)
      {
         r++; }
      else
      {
         while (r > 15)
         {
            bufferIt(outStream,
                  ((int[][])AC_matrix[ACcode])[0xF0][0],
                 ((int[][])AC_matrix[ACcode])[0xF0][1]);
            r -= 16;
         }
         temp2 = temp;
         if (temp < 0)
         {
            temp = -temp;
            temp2--;
         }
         nbits = 1;
         while ((temp >>= 1) != 0)
         {
            nbits++;      }
         i = (r << 4) + nbits;
         bufferIt(outStream,
                ((int[][])AC_matrix[ACcode])[i][0],
                ((int[][])AC_matrix[ACcode])[i][1]);
         bufferIt(outStream, temp2, nbits);
         r = 0;
      }
   }
   if (r > 0)
   {
      bufferIt(outStream,
             ((int[][])AC_matrix[ACcode])[0][0],
             ((int[][])AC_matrix[ACcode])[0][1]);
   } }
```

***Sourcecode* 4.7** *Sourcecode Huffman Coding*

### 4.2.7 Proses *Extraction*

Pada proses *extraction* atau penguraian pesan diimplementasikan dalam sebuah kelas (extract.java). Data yang diperlukan dalam proses penguraian pesan adalah citra stego (*stego image*) yang sudah dibuat dengan aplikasi ini dan sebuah *key* untuk otentikasi.

Dalam proses penguraian pesan dalam citra, pertama kali yang dilakukan adalah melakukan dekompresi terhadap stego image tersebut dengan menggunakan *huffman decoder*.

```
/* Andreas Westfeld 1999 */
/* HuffmanDecode.class:
/*  This object performs entopy decoding on      */
/*  a JPEG image file. This object instanciates  */
/*  HuffTable.class which extracts the Huffman   */
/* Tables from the file header information.     */
/*  Methods:                                     */
/*  HuffDecode(), returns array of 8x8      */
/*                blocks of image data      */
/*  getX(), returns horizontal image size   */
/*  getY(), returns vertical image size     */
/*  getPrec(), returns sample precision     */
/*  getComp(), returns number of components */
/*  rawDecode(), returns quantized coefficients */

// Return image data
public void HuffDecode(int[][][] buffer) {
    int x, y, tmp, sz = X * Y, scan=0;
    int[][] Block   = new int[8][8];
    int Cs, Ta, Td, blocks;
    long t;
    double time;

   // Read in Scan Header information
    Ls = getInt();  Ns = getByte();
    Cs = getByte();
    Td = getByte();
    Ta = Td & 0x0f;
    Td >>= 4;
    Ss = getByte(); Se = getByte();
    Ah = getByte(); Al = Ah & 0x0f;
    Ah >>= 4;

    // Calculate the Number of blocks encoded
    //blocks = X * Y / 64;
    blocks = getBlockCount()/6;
    // decode image data and return image data in array
```

```
    for(cnt=0;cnt<blocks;cnt++) {
    // Get DC coefficient
       if(Td == 0)
         hftbl = 0;
       else
         hftbl = 2;
       tmp = DECODE();
       DIFF = RECEIVE(tmp);
       ZZ[0] = PRED + EXTEND(DIFF, tmp);
       PRED = ZZ[0];
       // Get AC coefficients
       if(Ta == 0)
         hftbl = 1;
       else
         hftbl = 3;
       Decode_AC_coefficients();

       // dezigzag and dequantize block
       for(lp=0;lp<64;lp++)
          Block[deZZ[lp][0]][deZZ[lp][1]] = ZZ[lp] *
                                            QNT[0][lp];

       // store blocks in buffer
       for(x=0;x<8;x++)
          for(y=0;y<8;y++)
             buffer[cnt][x][y]=Block[x][y];
   }
   closeStream();
}

// Public get methods
public int getX() { return X; }
public int getY() { return Y; }
public int getPrec() { return P; }
public int getComp() { return Nf; }

// Return quantized coefficients
public void rawDecode(int[][][] buffer) {
   int x, y, tmp;
   int[][] Block   = new int[8][8];
   int Cs, Ta, Td, blocks;
   long t;
   double time;
   // Read in Scan Header information
   Ls = getInt();  Ns = getByte();
   Cs = getByte();
   Td = getByte();
   Ta = Td & 0x0f;
```

```
    Td >>= 4;

    Ss = getByte(); Se = getByte();
    Ah = getByte(); Al = Ah & 0x0f;
    Ah >>= 4;

    // Calculate the Number of blocks encoded
    blocks = getBlockCount()/6;

    // decode image data and return image data in array
    for(cnt=0;cnt<blocks;cnt++) {
       // Get DC coefficient
       if(Td == 0)
          hftbl = 0;
       else
          hftbl = 2;
       tmp = DECODE();
       DIFF = RECEIVE(tmp);
       ZZ[0] = PRED + EXTEND(DIFF, tmp);
       PRED = ZZ[0];

       // Get AC coefficients
       if(Ta == 0)
          hftbl = 1;
       else
          hftbl = 3;
       Decode_AC_coefficients();

       // dezigzag
       for(lp=0;lp<64;lp++)
           Block[deZZ[lp][0]][deZZ[lp][1]] = ZZ[lp];

       // store blocks in buffer
       System.out.print(cnt+" ");
       for(x=0;x<8;x++) {
          for(y=0;y<8;y++) {
              buffer[cnt][x][y]=Block[x][y];
              }
           }
    }
    closeStream();
}
```

***Sourcecode* 4.8** *Sourcecode Huffman Decode*

Setelah itu akan dilakukan ekstraksi yang akan diimplementasikan pada kelas extract.java berikut ini :

```
//Author: Andreas Westfeld

private static File f;         // carrier file
private static byte[] carrier;  // carrier data
private static int[] coeff;     // dct values
private static FileOutputStream fos;   // embedded file
                                       (output file)
private static String embFileName; // output file
                                   name
private static String password;
HuffmanDecode hd = new HuffmanDecode(carrier);
System.out.println("Huffman decoding starts");
coeff=hd.decode();
System.out.println("Permutation starts");
F5Random random = new F5Random(password.getBytes());
Permutation permutation = new Permutation(coeff.length,
                          random);

// extract length information
for (i=0; availableExtractedBits<32; i++) {
  shuffledIndex = permutation.getShuffled(i);
  if (shuffledIndex%64 == 0)
      continue; // skip DC coefficients
  shuffledIndex = shuffledIndex-
         (shuffledIndex%64)+deZigZag[shuffledIndex%64];
  if (coeff[shuffledIndex] == 0)
      continue; // skip zeroes

  // rev. 12: skip every second 1 or -1
  if (Math.abs(coeff[shuffledIndex]) == 1)
     if (((++one)&1) == 0)
        continue;

     if (coeff[shuffledIndex] > 0)
        extractedBit=coeff[shuffledIndex]&1;
     else
        extractedBit=1-(coeff[shuffledIndex]&1);
     extractedFileLength |= extractedBit <<
                            availableExtractedBits++;
}
// remove pseudo random pad
extractedFileLength ^= random.getNextByte();
extractedFileLength ^= random.getNextByte()<<8;
extractedFileLength ^= random.getNextByte()<<16;
extractedFileLength ^= random.getNextByte()<<24;
int k = extractedFileLength >> 24;
```

```
k %= 32;
int n = (1 << k)-1;
extractedFileLength &= 0x007fffff;
System.out.println("Ukuran File Pesan yang diembed: "
                   +extractedFileLength+ " bytes");
availableExtractedBits = 0;
if (n>0) {
   int startOfN = i;
   int hash;
    System.out.println("Kode yang digunakan (1, "+n+",
                       "+k+") ");
extractingLoop:
   do
   {
   // 1. read n places, and calculate k bits
   hash = 0;
   int code = 1;
   for (i=0; code<=n; i++)
   {
     // check for pending end of coeff
     if (startOfN+i>=coeff.length)
        break extractingLoop;
     shuffledIndex = permutation.getShuffled
                     (startOfN+i);
     if (shuffledIndex%64 == 0)
        continue; // skip DC coefficients
     shuffledIndex = shuffledIndex-(shuffledIndex%64)+
                     deZigZag[shuffledIndex%64];
     if (coeff[shuffledIndex] == 0)
        continue; // skip zeroes

     // rev. 12: skip every second 1 or -1
     if (Math.abs(coeff[shuffledIndex]) == 1)
        if ((++one&1) == 0)
           continue;

     if (coeff[shuffledIndex] > 0)
        extractedBit=coeff[shuffledIndex]&1;
     else
        extractedBit=1-(coeff[shuffledIndex]&1);
        if (extractedBit==1)
        {
           hash ^= code;
        }
     code++;   }
     startOfN += i;
     // 2. write k bits bytewise
     for (i=0; i<k; i++)
```

```
      {
         extractedByte |= ((hash>>i)&1) <<
                          availableExtractedBits++;
         if (availableExtractedBits == 8)
         {
            // remove pseudo random pad
            extractedByte ^= random.getNextByte();
            fos.write((byte) extractedByte);
            extractedByte=0;
            availableExtractedBits=0;
             nBytesExtracted++;
          // check for pending end of embedded data
         if (nBytesExtracted==extractedFileLength)
            break extractingLoop;
         } }
} while (true);
} else {
  System.out.println("Default code used");
  for (; i<coeff.length; i++) {
      shuffledIndex = permutation.getShuffled(i);
      if (shuffledIndex%64 == 0) continue; // skip DC
                                           coefficients
      shuffledIndex = shuffledIndex-
                      (shuffledIndex%64)+
                      deZigZag[shuffledIndex%64];
      if (coeff[shuffledIndex] == 0) continue; // skip
                                               zeroes

      // rev. 12: skip every second 1 or -1
      if (Math.abs(coeff[shuffledIndex]) == 1)
         if ((++one&1) == 0) continue;

      if (coeff[shuffledIndex] > 0)
         extractedBit=coeff[shuffledIndex]&1;
      else
         extractedBit=1-(coeff[shuffledIndex]&1);
      extractedByte |= extractedBit <<
                       availableExtractedBits++;
      if (availableExtractedBits == 8) {
      // remove pseudo random pad
      extractedByte ^= random.getNextByte();
      fos.write((byte) extractedByte);
      extractedByte=0;
      availableExtractedBits=0;
      nBytesExtracted++;
      if (nBytesExtracted==extractedFileLength)
         break;
      }   }
```

```
}
if (nBytesExtracted<extractedFileLength) {
  System.out.println("Incomplete    file:    only    "+
nBytesExtracted+"  of  "+extractedFileLength+  "  bytes
extracted");
}
} catch(Exception e) {
          e.printStackTrace(); }
```

***Sourcecode* 4.9** *Sourcecode Extraction*

### 4.2.8 *Root Mean Square (RMS)*

Perhitungan *RMS* (Persamaan 3.2) digunakan untuk melihat apakah pengaruh penyisipan pesan terlihat jelas pada kualitas citra. Hasil yang dikeluarkan dalam bentuk persen. Perhitungan RMS ini menggunakan aplikasi *Microsoft Visual C# 2005 Express Edition*. Pada perhitungan RMS ini, pertama-tama baik citra asli maupun citra *stego* akan di *gray scale* kemudian akan dihitung selisih nilai piksel *gray scale* citra asli dan nilai piksel citra stego. Setelah itu hasil selisihnya akan dibagi dengan panjang *x* lebar citra yang diinputkan.

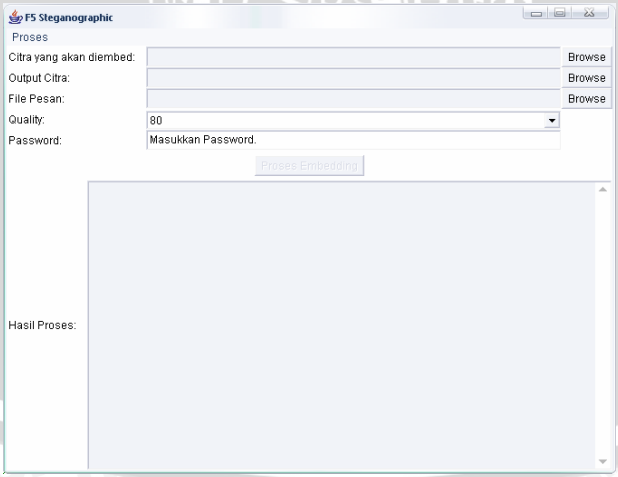
```
if (Form1.Imageasli.Width == Form1.Imagestego.Width &&
Form1.Imageasli.Height==Form1.Imagestego.Height)
{
   int selisih = 0;
   BitmapData mGb1 = Form1.Imageasli.LockBits(new
       Rectangle(0, 0, Form1.Imageasli.Width,
       Form1.Imageasli.Height), ImageLockMode.ReadWrite,
       PixelFormat.Format24bppRgb);
   BitmapData mGb2 = Form1.Imagestego.LockBits(new
       Rectangle(0, 0, Form1.Imagestego.Width,
      Form1.Imagestego.Height),ImageLockMode.ReadWrite,
      PixelFormat.Format24bppRgb);

      unsafe
      {
        byte* ptr1 = (byte*)mGb1.Scan0;
        byte* ptr2 = (byte*)mGb2.Scan0;

        for (int i = 0; i < mGb1.Height; i++)
        {
           for (int j = 0; j < mGb1.Width; j++)
           {
             //mencari gray dari image pada pixel
              tertentu : (R+G+B)/3
              mean1 = ptr1[0] + ptr1[1] + ptr1[2];
```

```
                mean1 /= 3;
                mean2 = ptr2[0] + ptr2[1] + ptr2[2];
                mean2 /= 3;
                selisih += Convert.ToInt32
                           (Math.Pow(Convert.ToDouble
                           (mean1 - mean2), 2));

                ptr1 += 3;
                ptr2 += 3;
            }
        }
        drms = (Math.Sqrt(Convert.ToDouble(selisih)) /
               (mGb1.Height * mGb1.Width));
        percent_drms = drms * 100;
        this.textBox1.Text = "NILAI RMS =" +
                             percent_drms.ToString()
                             + "%";     }

    Form1.Imageasli.UnlockBits(mGb1);
    Form1.Imagestego.UnlockBits(mGb2);
}
```

***Sourcecode* 4.10** *Sourcecode Root Mean Square* (RMS)

### 4.3 Pembahasan

Pada bagian pembahasan ini akan diuji program aplikasi steganografi yang telah dibuat. Berikut ini adalah tampilan program ketika pertama kali dijalankan.:



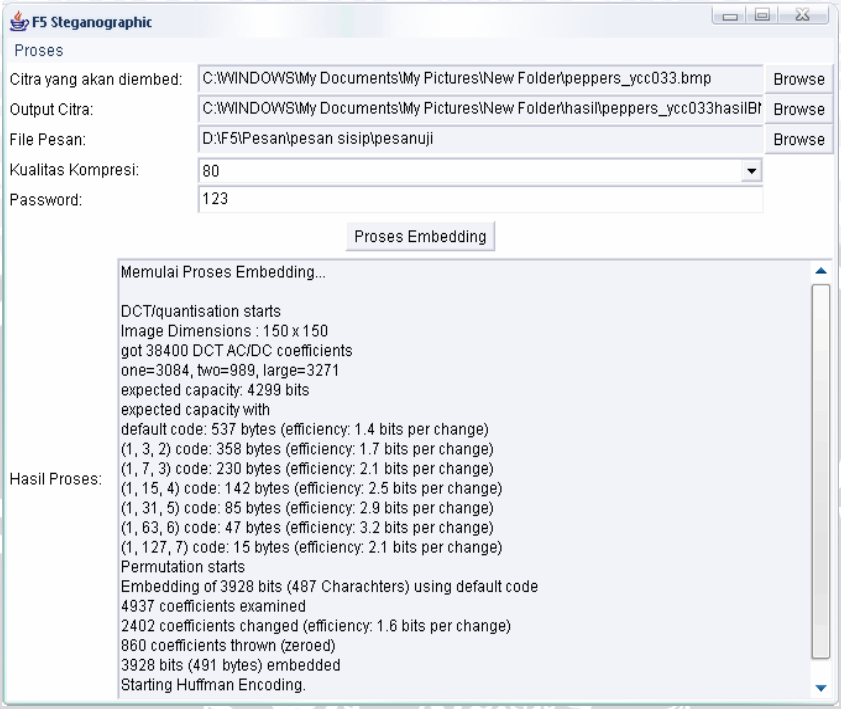

**Gambar 4.1** Tampilan *running* program

### 4.3.1 Proses *Embedding*

Pada proses *embedding* ini digunakan media gambar dengan format BMP dengan kedalaman warna gambar sebesar 24 bpp, resolusi 150 x 150 dan ukuran citra 67 KB. Sedangkan teks yang disisipkan dalam gambar diambil dari file teks dengan kapasitas sebesar 487 Bytes, dan menggunakan Kualitas kompresi sebesar 80.

Langkah pertama memilih gambar dengan menekan tombol Browse yang terletak di atas kotak gambar, kemudian akan muncul open dialog untuk memilih gambar yang diinginkan. Lalu setelah menekan tombol OK, nama direktori serta nama gambar akan muncul pada teks box. Kemudian simpan output citra dengan cara menekan tombol Browse sehingga akan muncul save dialog, tekan tombol save maka akan muncul nama direktori dan nama citra setelah dilakukan *embedding*, hal ini dilakukan untuk menyimpan citra stego dengan format jpg (*.jpg). Sedangkan untuk membuka file teks yang ada, dengan menekan tombol Browse pada File Pesan, lalu akan muncul open dialog untuk memilih file teks yang akan dibuka. Setelah menekan ok, maka akan muncul nama direktori dan nama file teks. Isi Kualitas Kompresi, dimana kualitas ini sebagai nilai acuan dalam mengkompresi citra stego (sebagai *default*-nya adalah 80), Setelah mengisi Password, klik tombol Proses Embedding maka proses penyisipan dilakukan.

Gambar 4.2 menyatakan gambar tampilan ketika proses selesai dilakukan:




**Gambar 4.2** Tampilan proses penyisipan pesan dengan citra input format BMP

Pada Gambar 4.2 menunjukkan bahwa ukuran citra inputan adalah 150x150 *pixel*. Kemudian nilai hasil perhitungan DCT yaitu sebesar 38400, dengan koefisien pertama = 3084, kedua = 989 dan yang terbesar = 3271 dan sebesar 3928 bits atau 487 karakter berhasil di-*embed* atau disisipkan menggunakan *default code* (dengan $k$ = 1, sehingga kode 1,1,1) yaitu sebesar 537 bytes dengan efisiensi sebesar 1,4 bits per *change*. Sebanyak 2402 koefisien diubah atau disisipi dengan bit data pesan dan sebanyak 860 koefisien yang di-*skip* atau di-*shrinkage*.

Setelah semua karakter berhasil di-*embed* maka langkah selanjutnya adalah melakukan *Huffman Coding* untuk mengompres citra *stego*.

Dari proses penyisipan pesan diatas, dihasilkan sebuah gambar stego dengan format jpg (*.jpg) yang ukurannya sama persis dengan gambar aslinya sebelum disisipkan pesan. Ukuran resolusi dan

kedalaman warnanya sama persis, keduanya mempunyai resolusi 150x150 tetapi kapasitas citranya berkurang sebesar 58 KB, dengan kapasitas citra asli atau citra sebelum disisipkan adalah 67 KB sedangkan citra hasil stego yang telah dikompresi menjadi 9 KB.

a. media citra asli (peppers_ycc0333.bmp, 67KB)

b. Citra stego (peppers_ycc033hasilBMP.jpg, 9 KB)

**Gambar 4.3** Media Citra asli format BMP dan gambar stego format JPEG

Setelah dilakukan penyisipan pesan maka citra dengan format BMP tersebut tidak terlalu banyak mengalami perubahan hal ini terlihat dari gambar stego peppers_ycc033hasilBMP.jpg yang telah mengalami penyisipan pesan. Tidak terlihat perbedaan secara nyata dengan media gambar aslinya yaitu peppers_ycc0333.bmp.

Untuk mengetahui apakah terjadi perubahan *pixel*, berikut ini nilai RGB (*Red, Green, Blue*) sebelum citra disisipkan pesan dan RGB sesudah citra disisipkan pesan (sesuai pada Tabel 4.1). Hampir seluruh nilai *pixel* mengalami perubahan yang cukup besar. Ada nilai

*pixel* yang mengalami penambahan dan ada pula nilai *pixel* yang mengalami pengurangan.

**Tabel 4.1** Contoh nilai pixel citra asli format BMP dan citra stego yang berubah

| Koordinat | | Sebelum peppers_ycc0333.bmp | | | Sesudah peppers_ycc033hasilBMP.jpg | | |
|---|---|---|---|---|---|---|---|
| X | Y | R | G | B | R_S | G_S | B_S |
| 0 | 0 | 145 | 40 | 12 | 139 | 41 | 6 |
| 1 | 0 | 182 | 77 | 49 | 179 | 81 | 46 |
| 2 | 0 | 178 | 73 | 46 | 180 | 82 | 47 |
| 3 | 0 | 171 | 66 | 39 | 169 | 71 | 36 |
| 4 | 0 | 173 | 67 | 41 | 169 | 68 | 38 |
| … | … | … | … | … | … | … | … |
| 145 | 149 | 208 | 119 | 199 | 208 | 112 | 198 |
| 146 | 149 | 207 | 118 | 197 | 207 | 116 | 198 |
| 147 | 149 | 209 | 123 | 197 | 208 | 123 | 198 |
| 148 | 149 | 210 | 125 | 199 | 208 | 128 | 200 |
| 149 | 149 | 210 | 126 | 201 | 209 | 131 | 201 |

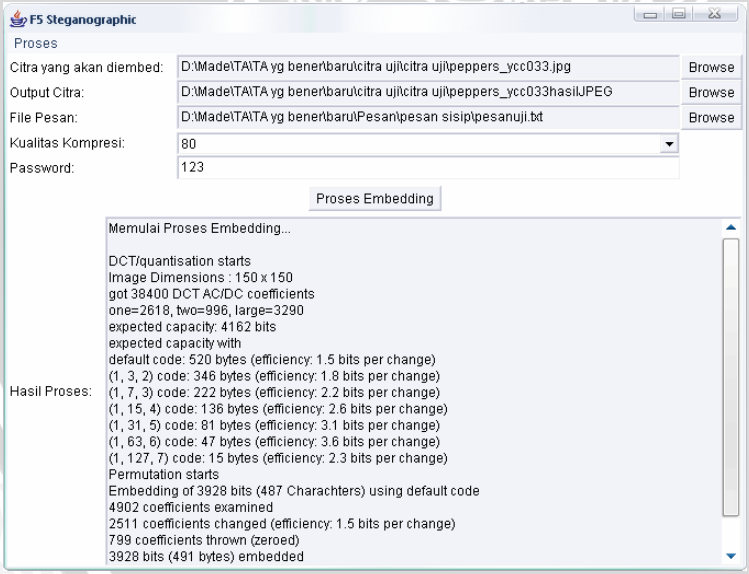
Setelah itu akan diuji kualitas dari citra asli format BMP dan citra stego dengan menggunakan RMS (*Root Mean Square*). Berikut ini proses perhitungan *RMS* dari citra asli format BMP :

**Tabel 4.2** Nilai pixel *Gray scale* dan selisih citra asli format BMP dan citra *stego*

| X | Y | GrayScale_Citra Asli (BMP) | GrayScale_Citra Stego (JPEG) | Selisih |
|---|---|---|---|---|
| 0 | 0 | 65 | 62 | 0 |
| 1 | 0 | 102 | 102 | 9 |
| 2 | 0 | 99 | 103 | 9 |
| 3 | 0 | 92 | 92 | 25 |
| 4 | 0 | 93 | 91 | 25 |
| … | … | … | … | … |
| 145 | 149 | 175 | 172 | 320281 |
| 146 | 149 | 174 | 173 | 320290 |
| 147 | 149 | 176 | 176 | 320291 |
| 148 | 149 | 178 | 178 | 320291 |
| 149 | 149 | 179 | 180 | 320291 |

Untuk menghitung RMS, nilai selisih (nilai selisih didapatkan dari hasil pengurangan antara nilai piksel *grayscale* citra *stego* dengan *grayscale* citra aslinya, kemudian hasilnya akan ditambahkan terus sesuai dengan panjang dan lebar citra) seperti pada Tabel 4.2 antara nilai piksel grayscale citra asli dan citra stego kemudian dibagi dengan panjang *x* lebar citra. Sehingga nilai *RMS* citra asli peppers_ycc0333.bmp dan citra stego peppers_ycc033hasilBMP.jpg adalah 2,515%, ini berarti bahwa citra *stego* mengalami penurunan kualitas tetapi penurunan kualitasnya tidak terlalu besar.

Dan untuk mengetahui apakah citra *stego* mengalami penurunan kualitas bukan karena proses kompresi (menggunakan kompresi *Huffman*), maka akan dilakukan proses *embedding* dengan citra input format JPEG. Pada proses *embedding* dengan citra input JPEG ini, media citra asli format BMP (Gambar 4.3.a) dengan ukuran atau dimensi 150x150 pixel akan dikompresi menjadi format JPEG. Sehingga kapasitas atau *size*-nya menjadi 8,83 KB. Kemudian akan dilakukan proses *embedding* atau penyisipan data teks dengan ukuran 487 Bytes.




**Gambar 4.4** Tampilan proses penyisipan pesan dengan citra input format JPEG

Dari Gambar 4.4 menunjukkan terjadi perbedaan perhitungan koefisien DCT dari citra input format BMP dan citra input format JPEG. Untuk menyisipkan pesan sebesar 487 *bytes* pada citra input JPEG, sebanyak 2511 koefisien yang diubah dengan efisiensi 1.5 bit per *change* dan sebanyak 799 koefisien yang di-*skip* atau di-*shrinkage*.

Dari proses penyisipan pesan diatas, dihasilkan sebuah gambar stego dengan format jpg (*.jpg) yang ukurannya sama persis dengan gambar aslinya sebelum disisipkan pesan.

a. media citra asli (peppers_ycc033.jpg, 9 KB)

b. Citra stego (peppers_ycc033hasilJPEG.jpg, 6 KB)

**Gambar 4.5** Media Citra asli format JPEG dan gambar stego format JPEG

Dan Tabel 4.3 menunjukkan nilai RGB (*Red, Green, Blue*) sebelum citra disisipkan pesan dan RGB sesudah citra disisipkan pesan. Seperti pada citra input format BMP, citra input format JPEG juga mengalami perubahan nilai piksel. Ada nilai *pixel* yang mengalami penambahan dan ada pula nilai *pixel* yang mengalami pengurangan.

**Tabel 4.3** Contoh nilai pixel citra asli format JPEG dan citra stego yang berubah

| Koordinat | | Sebelum peppers_ycc0333.jpg | | | Sesudah peppers_ycc033hasilJPEG.jpg | | |
|---|---|---|---|---|---|---|---|
| X | Y | R | G | B | R_S | G_S | B_S |
| 0 | 0 | 146 | 41 | 11 | 150 | 50 | 18 |
| 1 | 0 | 180 | 75 | 45 | 174 | 74 | 42 |
| 2 | 0 | 180 | 75 | 45 | 180 | 76 | 47 |
| 3 | 0 | 172 | 67 | 37 | 174 | 70 | 41 |
| 4 | 0 | 173 | 68 | 39 | 167 | 63 | 36 |
| … | … | … | … | … | … | … | … |
| 145 | 149 | 207 | 118 | 199 | 207 | 119 | 202 |
| 146 | 149 | 207 | 120 | 197 | 207 | 119 | 201 |
| 147 | 149 | 210 | 123 | 197 | 208 | 123 | 201 |
| 148 | 149 | 210 | 124 | 200 | 209 | 124 | 200 |
| 149 | 149 | 210 | 124 | 200 | 210 | 125 | 201 |

Setelah itu akan diuji kualitas dari citra asli format JPEG dan citra stego dengan menggunakan RMS (*Root Mean Square*). Pada Tabel 4.4 menunjukkan proses perhitungan *RMS* dari citra asli format JPEG :

**Tabel 4.4** Nilai pixel *Gray scale* dan selisih citra asli format JPEG dan citra *stego*

| X | Y | GrayScale_Citra Asli | GrayScale_Citra Stego | Selisih |
|---|---|---|---|---|
| 0 | 0 | 66 | 72 | 0 |
| 1 | 0 | 100 | 96 | 36 |
| 2 | 0 | 100 | 101 | 52 |
| 3 | 0 | 92 | 95 | 53 |
| 4 | 0 | 93 | 88 | 62 |
| … | … | … | … | … |
| 145 | 149 | 174 | 176 | 353482 |
| 146 | 149 | 174 | 175 | 353486 |
| 147 | 149 | 176 | 177 | 353487 |
| 148 | 149 | 178 | 177 | 353488 |
| 149 | 149 | 178 | 178 | 353489 |

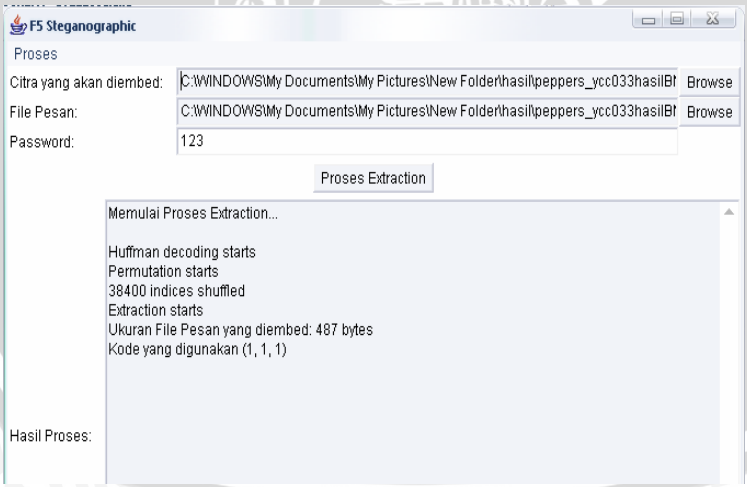
Dari hasil Tabel 4.4, hasil perhitungan *RMS* citra asli format JPEG dan citra stego format JPEG mengalami penurunan kualitas sebesar 2,642%. Hasil *RMS* ini tidak terlalu jauh berbeda dengan citra input atau citra asli format BMP sehingga penurunan kualitas ini tidak disebabkan oleh proses kompresi, melainkan proses *embedding* pada citra tersebut.

Sehingga dapat disimpulkan bahwa citra stego dari hasil citra input format BMP memiliki kualitas yang lebih baik daripada citra stego dari hasil citra input format JPEG. Tetapi citra input format JPEG (karena telah mengalami kompresi) memiliki ukuran kapasitas yang jauh lebih kecil daripada citra input format BMP.

### 4.3.2 Proses *Extraction*

Kemudian setelah proses *embedding*, proses selanjutnya adalah proses *extraction*. Untuk melakukan penguraian pesan atau *extraction* dalam gambar, dibutuhkan citra *stego* yang telah dibuat dengan aplikasi ini. Untuk memulai proses *extraction* pertama pilih citra stego dengan menekan tombol Browse, kemudian pilih citra stego yang akan diekstraksi, setelah itu tekan tombol Browse untuk menyimpan file teks yang telah diekstrak dan masukkan Password yang benar dan tekan tombol Proses Extraction, maka teks yang tersembunyi akan dapat diurai.

Gambar 4.6 menunjukkan tampilan proses penguraian pesan.




**Gambar 4.6** Tampilan proses *extraction*

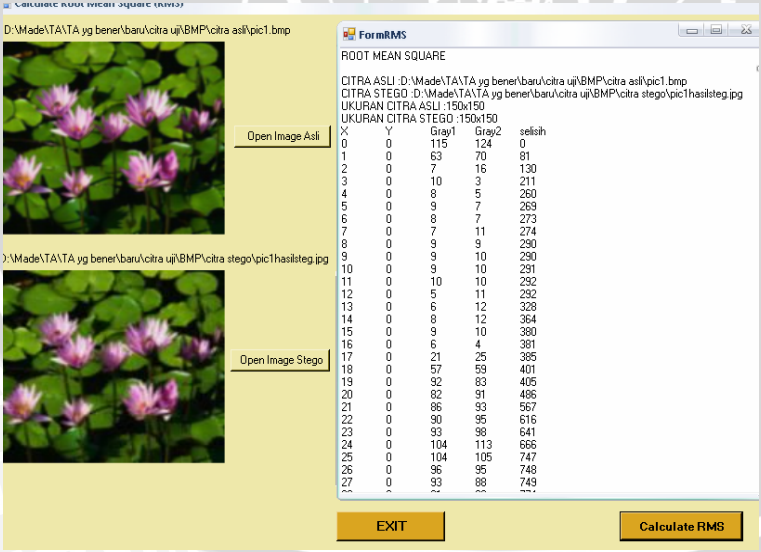
Pada proses *extraction* ini, pertama-tama yaitu melakukan *Huffman decoding*, setelah itu lakukan permutasi sehingga didapatkan 38400 indeks-indeks yang telah diubah. Setelah proses permutasi selesai maka langkah selanjutnya adalah proses ekstraksi Pada proses ekstraksi ini menggunakan kode 1,1,1 (*default code*) sehingga didapatkan file data pesan yang telah disisipkan sebesar 487 bytes.

## 4.4 Implementasi Uji Coba

Pada subbab ini akan dilakukan implementasi uji coba mengenai pengujian kualitas dan ketahanan yang telah dilakukan pada sistem dan hasil evaluasi dari hasil yang dikeluarkan sistem.

### 4.4.1 Evaluasi Kualitas (*Fidelity*) Citra

Kualitas citra akan dihitung dengan *RMS* (sesuai dengan Persamaan 3.1) untuk melihat pengaruh penyisipan pesan dalam citra. Dalam perhitungan RMS ini dibutuhkan citra asli dengan format BMP (*.bmp) dan JPEG (*.jpg) dan citra stego dengan format JPEG (*.jpg). Pada proses perhitungan *RMS* ini, pertama-tama masukkan citra asli dan citra stego kemudian tekan button Calculate RMS. Gambar 4.6 menunjukkan tampilan proses perhitungan *RMS* :




**Gambar 4.7** Contoh tampilan proses perhitungan RMS

Pada Tabel 4.5 menunjukkan ini nilai *pixel Gray scale* dan selisih citra asli format BMP (pic1.bmp) dan citra *stego* Pic1hasilsteg.jpg) :

**Tabel 4.5** Nilai pixel *Gray scale* dan selisih citra asli format BMP (pic1.bmp) dan citra *stego* Pic1hasilsteg.jpg)

| X | Y | GrayScale_Citra Asli | GrayScale_Citra Stego | Selisih |
|---|---|---|---|---|
| 0 | 0 | 115 | 124 | 0 |
| 1 | 0 | 63 | 70 | 81 |
| 2 | 0 | 7 | 16 | 130 |
| 3 | 0 | 10 | 3 | 211 |
| 4 | 0 | 8 | 5 | 260 |
| 5 | 0 | 9 | 7 | 269 |
| 6 | 0 | 8 | 7 | 273 |
| 7 | 0 | 7 | 11 | 274 |
| 8 | 0 | 9 | 9 | 290 |
| 9 | 0 | 9 | 10 | 290 |
| 10 | 0 | 9 | 10 | 291 |
| … | … | … | … | … |
| 140 | 149 | 7 | 7 | 484644 |
| 141 | 149 | 8 | 6 | 484644 |
| 142 | 149 | 8 | 6 | 484648 |
| 143 | 149 | 7 | 7 | 484652 |
| 144 | 149 | 6 | 6 | 484652 |
| 145 | 149 | 5 | 6 | 484652 |
| 146 | 149 | 5 | 4 | 484653 |
| 147 | 149 | 13 | 8 | 484654 |
| 148 | 149 | 13 | 6 | 484679 |
| 149 | 149 | 8 | 5 | 484728 |

Dari hasil perhitungan selisih sesuai dengan Tabel 4.5 maka hasil *Root Mean Square* (RMS) untuk citra asli format BMP (pic1.bmp) dan citra stego (pic1hasilsteg.jpg) yang berdimensi 150 *x* 150 piksel adalah 3,09%. Dari hasil RMS ini menunjukkan bahwa terjadi penurunan kualitas citra.

Tabel 4.6 menunjukkan hasil perhitungan RMS 30 citra BMP dan 30 citra JPEG yang diuji

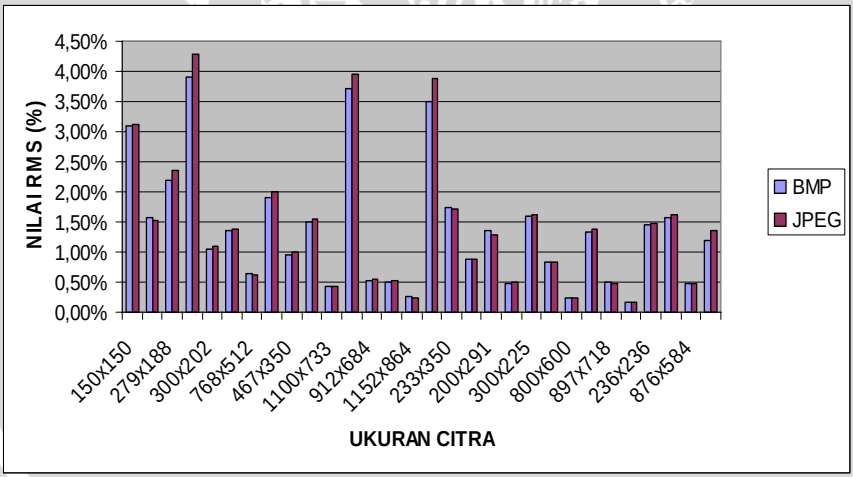
**Tabel 4.6** Tabel Kualitas *Stego Image*

| NO. | Nama citra | dimensi (piksel) | ukuran citra (KB) | | ukuran pesan | RMS | |
|---|---|---|---|---|---|---|---|
| | | | BMP | JPEG | | (*.bmp - *.jpg) | (*.jpg - *.jpg) |
| 1. | pic1 | 150x150 | 67 | 11 | 648 B | 3,09% | 3,13% |
| 2. | Stonehenge | 133x100 | 154 | 14 | 368 B | 1,58% | 1,53% |
| 3. | 032029B | 279x188 | 155 | 18 | 3,79 KB | 2,20% | 2,37% |
| 4. | 036050B | 172x115 | 59 | 8 | 3,79 KB | 3,90% | 4,29% |
| 5. | Borobudur | 300x202 | 178 | 27 | 368 B | 1,04% | 1,09% |
| 6. | 377076 | 338x227 | 226 | 25 | 6,80 KB | 1,36% | 1,39% |
| 7. | 454049 | 768x512 | 1.153 | 161 | 6,80 KB | 0,64% | 0,62% |
| 8. | Butterfly | 295x200 | 17 4 | 16 | 368 B | 1,91% | 1,99% |
| 9. | 2257717410031856995GGVvPm_fs | 467x350 | 480 | 40 | 6,80 KB | 0,96% | 1,00% |
| 10. | 2320910920028702361kEAyhJ_fs | 386x257 | 292 | 32 | 6,80 KB | 1,50% | 1,55% |
| 11. | 2359563510028702361fTnQGZ_fs | 1100x733 | 2.917 | 192 | 8,64 KB | 0,42% | 0,44% |
| 12. | 2613464020101281656wlsmvf_fs | 291x216 | 185 | 37 | 3,79 KB | 3,72% | 3,94% |
| 13. | 2890974650014756417mTPOsj_ph | 912x684 | 1.828 | 224 | 8,64 KB | 0,53% | 0,54% |
| 14. | APACHE | 640x518 | 972 | 97 | 6,80 KB | 0,50% | 0,51% |
| 15. | Firstrays | 1152x864 | 2.917 | 187 | 8,64 KB | 0,25% | 0,24% |
| 16. | Heron | 156x112 | 52 | 6 | 3,79 KB | 3,51% | 3,88% |
| 17. | Kaktus | 233x350 | 240 | 27 | 3,79 KB | 1,73% | 1,71% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 18. | Leaves02 | 499x502 | 736 | 69 | 6,80 KB | 0,88% | 0,89% |
| 19. | lidah_buaya | 200x291 | 171 | 17 | 3,79 KB | 1,35% | 1,29% |
| 20. | MATAHARI | 640x480 | 901 | 25 | 6,80 KB | 0,48% | 0,51% |
| 21. | Melati | 300x225 | 198 | 14 | 6,80 KB | 1,59% | 1,61% |
| 22. | pic16 | 485x320 | 456 | 72 | 6,80 KB | 0,82% | 0,82% |
| 23. | PPLFLR04 | 800x600 | 1.407 | 66 | 8,64 KB | 0,25% | 0,25% |
| 24. | shuttle3-768 | 410x308 | 371 | 43 | 6,80 KB | 1,34% | 1,38% |
| 25. | Worl~190 | 897x718 | 1.888 | 287 | 8,64 KB | 0,51% | 0,48% |
| 26. | pic13 | 900x900 | 2.374 | 100 | 8,64 KB | 0,16% | 0,16% |
| 27. | pic14 | 236x236 | 164 | 12 | 3,79 KB | 1,45% | 1,48% |
| 28. | Kuda | 245x172 | 145 | 16 | 368 B | 1,57% | 1,61% |
| 29. | pic18 | 876x584 | 1.499 | 163 | 8,64 KB | 0,47% | 0,47% |
| 30. | pic19 | 500x398 | 584 | 85 | 6,80 KB | 1,20% | 1,36% |
| | | | | | JUMLAH | 40,91% | 42,53% |
| | | | | | | | |
| | | | | | MEAN | 1,367% | 1,418% |

Pada Tabel 4.6 menunjukkan hasil perhitungan kualitas *stego image* dengan citra input format BMP dan citra input format JPEG. Angka 1-30 menunjukkan 30 citra yang berbeda-beda masing – masing dengan format BMP dan JPEG. Dan ukuran citra atau dimensi citra serta ukuran pesan yang disisipkan yang berbeda. Dari Tabel 4.6 ukuran citra dengan format BMP jauh lebih besar dari pada citra format JPEG. Dari Tabel 4.6 dapat diketahui bahwa citra stego (baik dari citra input format BMP dan citra input format JPEG) mengalami penurunan kualitas. Tetapi dari hasil *Root Mean Square* (*RMS*) citra stego dari citra asli format JPEG memiliki RMS yang cukup besar dibandingkan dari citra input format BMP. Hal ini dapat diketahui dari hasil rata-rata (mean) RMS-nya yaitu dari 30 citra uji hasil rata-rata RMS citra input format BMP adalah sebesar 1,367%, sedangkan hasil rata-rata RMS citra input format JPEG adalah sebesar 1,418%. Sehingga dapat disimpulkan bahwa citra format BMP lebih baik digunakan sebagai citra input dalam sistem steganografi dengan algoritma F5 ini.

Grafik dari hasil perhitungan *RMS* sesuai pada Tabel 4.6 dengan sumbu *x* adalah ukuran citra dan sumbu *y* adalah nilai hasil RMS.




**Gambar 4.8** Grafik hasil perhitungan *RMS* citra format BMP dan JPEG

### 4.4.2 Evaluasi Ketahanan (*Robustness*) Citra *Stego*

Pada subbab ini akan diuji ketahanan (sesuai dengan subbab 3.3.3) citra *stego* yang telah disisipkan pesan kemudian citra *stego* tersebut akan diberi operasi manipulasi pengolahan citra atau *image distortion*.

Dalam subbab ini akan dibahas tentang pengujian mutu berkas citra yang telah disisipi pesan (*stego image*). Apakah informasi masih dapat diperoleh dari citra yang telah diubah-ubah tersebut.

#### 4.4.2.1 Penambahan Efek *Gaussian Blur*

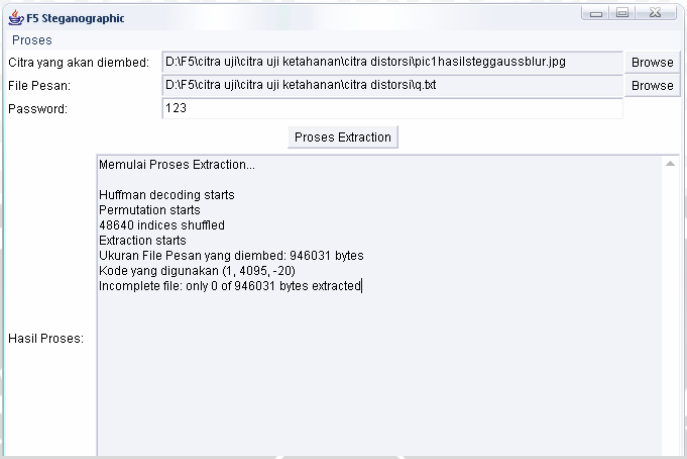
Pengujian kemudian dilanjutkan untuk melihat apakah pesan yang disisipkan masih dapat diekstrak meskipun gambar mengalami beberapa perubahan. Berikut pengujian ketahanan citra stego dengan operasi penambahan efek *Gaussian blur* :

**Pic1hasilsteg.jpg (citra stego)**

**Pic1hasilsteggaussblur.jpg (citra stego gaussblur)**

**Gambar 4.9** Gambar stego tanpa efek dan gambar stego yang diberi efek *Gaussian Blur*

Berikut ini hasil ekstraksi Gambar 4.9 :



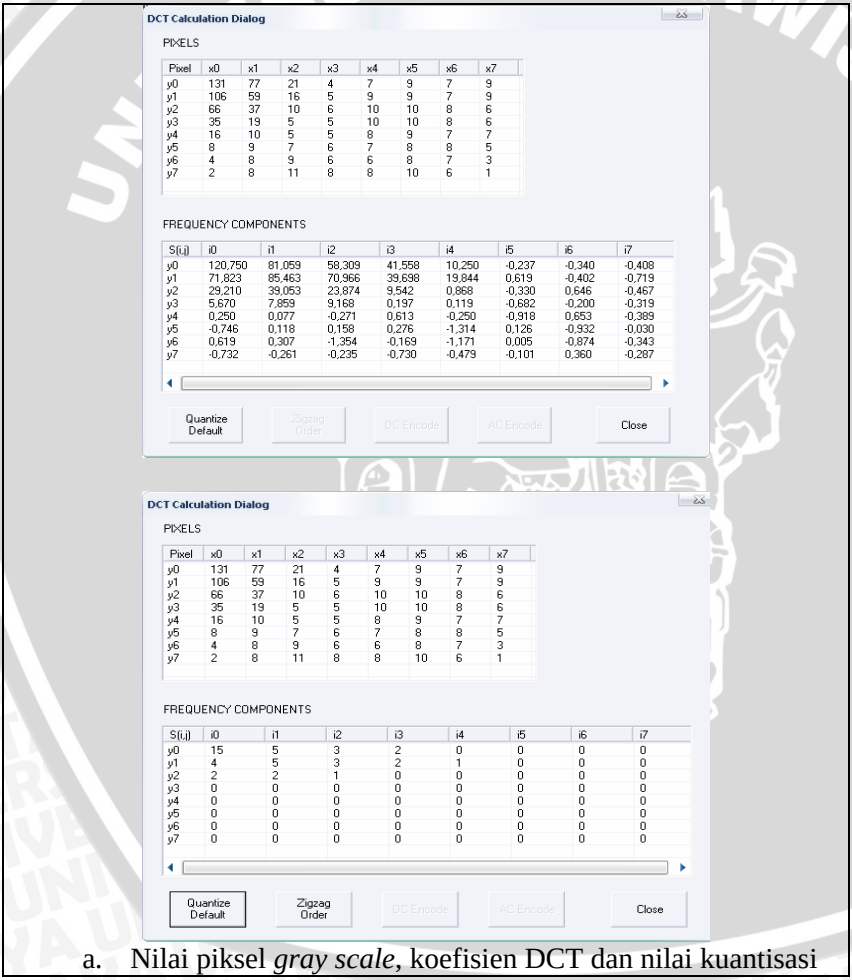

**Gambar 4.10** Hasil ekstraksi Pic1hasilsteggaussblur.jpg

Setelah dilakukan penambahan efek *Gaussian Blur* ternyata pesan tidak dapat diekstrasi lagi. Hal ini disebabkan adanya perubahan warna tiap-tiap piksel. Dan untuk membuktikan bahwa terjadi perubahan warna pada citra stego yang diberi efek *gaussian blur* maka Tabel 4.7 menunjukkan perbandingan nilai RGB dari citra stego tanpa efek dan citra stego dengan efek :

**Tabel 4.7** Nilai *pixel* citra stego tanpa efek *Gaussian Blur* dan citra stego dengan efek *Gaussian Blur*

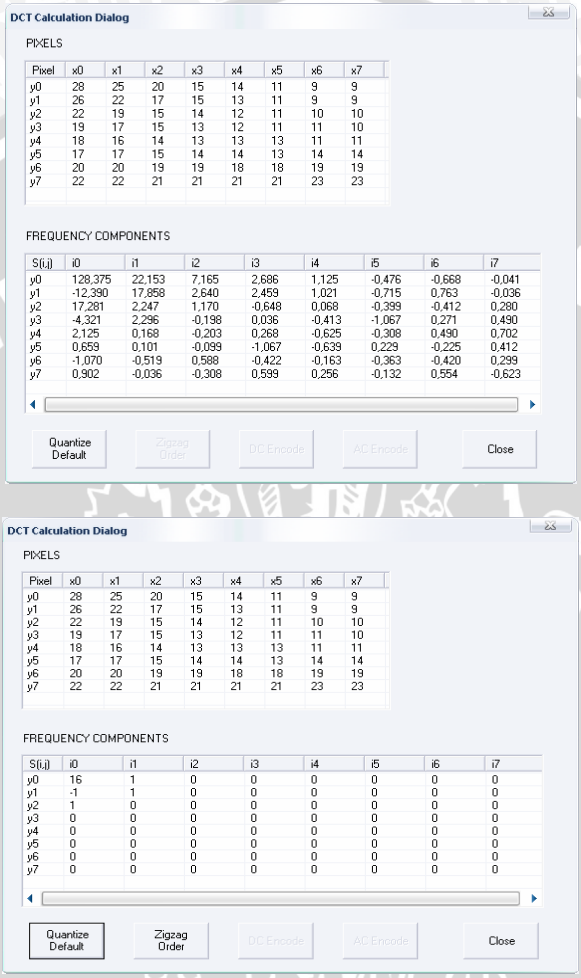
| Koordinat | | Sebelum pic1hasilsteg.jpg | | | Sesudah pic1hasilsteggaussblur.jpg | | |
|---|---|---|---|---|---|---|---|
| X | Y | R | G | B | R_blur | G_ blur | B_ blur |
| 0 | 0 | 135 | 136 | 102 | 26 | 33 | 15 |
| 1 | 0 | 80 | 81 | 49 | 22 | 29 | 13 |
| 2 | 0 | 24 | 24 | 0 | 17 | 24 | 8 |
| 3 | 0 | 6 | 5 | 0 | 13 | 19 | 7 |
| 4 | 0 | 8 | 9 | 0 | 11 | 17 | 7 |
| 10 | 0 | 5 | 7 | 19 | 5 | 11 | 7 |
| … | … | … | … | … | … | … | … |
| 145 | 149 | 12 | 1 | 5 | 29 | 6 | 13 |
| 146 | 149 | 12 | 0 | 2 | 31 | 6 | 15 |
| 147 | 149 | 18 | 8 | 0 | 34 | 6 | 17 |
| 148 | 149 | 12 | 0 | 6 | 37 | 7 | 15 |
| 149 | 149 | 15 | 0 | 0 | 39 | 7 | 18 |

Tabel 4.7 menunjukkan bahwa terjadi perubahan nilai piksel yang cukup signifikan. Sehingga hal ini membuktikan bahwa citra stego yang diberi efek *gaussian blur* (pic1hasilsteggaussblur.jpg) mengalami perubahan warna tiap pikselnya (karena telah mengalami perubahan warna tiap-tiap *pixel* agar membentuk distribusi normal sesuai pada Bab 2 Persamaan 2.5). Dan hal ini juga mempengaruhi nilai koefisien DCT dan kuantisasinya. Berikut perbandingan Nilai piksel *gray scale*, koefisien DCT dan kuantisasi antara citra stego tanpa efek *gaussian blur* dan citra *stego* dengan efek *gaussian blur*.




a. Nilai piksel *gray scale*, koefisien DCT dan nilai kuantisasi

citra stego tanpa efek *gaussian blur* (pic1hasilsteg.jpg)




b. Nilai piksel *gray scale,* koefisien DCT dan nilai kuantisasi citra stego dengan efek *gaussian blur* (pic1hasilsteg.jpg)

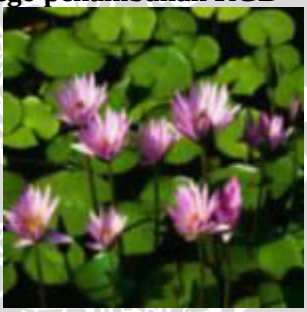
**Gambar 4.11** Nilai piksel *gray scale,* koefisien DCT dan kuantisasi pic1hasilsteg.jpg dan pic1hasilsteggaussblur.jpg.

#### 4.4.2.2 Penambahan Intensitas Warna

Uji ketahanan selanjutnya adalah menambah intensitas warna pada citra stego. Hal ini untuk dimaksudkan untuk mengetahui apakah dengan menambah atau mengubah intensitas warna, pesan dalam citra stego tersebut dapat diekstrak kembali atau tidak (untuk mengetahui apakah dengan mengubah nilai pikselnya, pesan masih dapat diekstrak kembali). Dalam uji coba ini, nilai RGB (*Red*, *Green*, *Blue*) citra stego akan ditambah. Dengan menggunakan *ACD See 9 Photo Manager* maka intensitas RGB citra stego akan ditambah masing-masing sebanyak 5 (*Red* = 5, *Green* = 5, *Blue* = 5).

**Pic1hasilsteg.jpg (citra stego)**

**pic1hasilstegRGB5.jpg (citra stego penambahan RGB = 5)**



**Gambar 4.12** Gambar stego tanpa penambahan intensitas dan gambar stego yang diberi penambahan intensitas warna.

Tabel 4.8 menunjukkan nilai piksel citra *stego* sebelum ditambahkan intensitas dan citra *stego* yang telah ditambahkan intensitas warnanya:

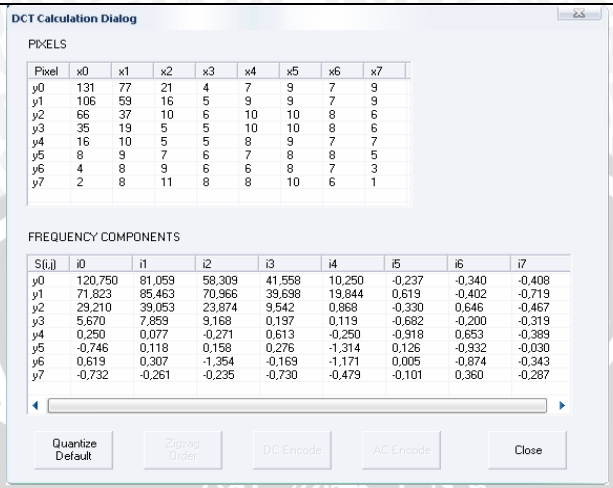
**Tabel 4.8** Nilai *pixel* citra *stego* tanpa penambahan RGB dan citra stego dengan penambahan RGB sebanyak 5.

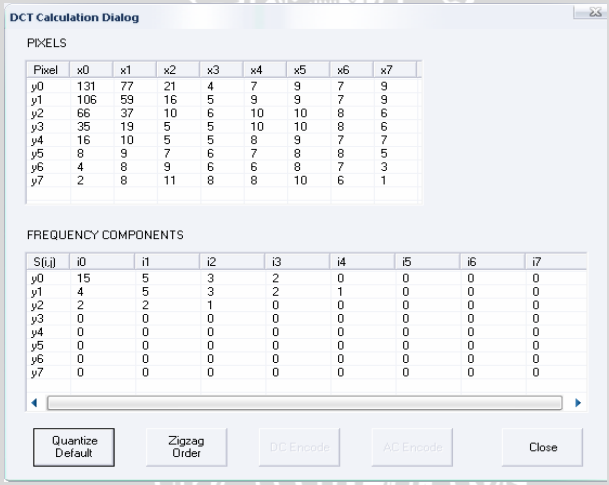
| Koordinat | | Sebelum pic1hasilsteg.jpg | | | Sesudah pic1hasilstegRGB5r.jpg | | |
|---|---|---|---|---|---|---|---|
| X | Y | R | G | B | R_ | G_ | B_ |
| 0 | 0 | 135 | 136 | 102 | 139 | 145 | 109 |
| 1 | 0 | 80 | 81 | 49 | 80 | 85 | 53 |
| 2 | 0 | 24 | 24 | 0 | 22 | 26 | 0 |
| 3 | 0 | 6 | 5 | 0 | 3 | 6 | 0 |
| 4 | 0 | 8 | 9 | 0 | 8 | 10 | 0 |
| … | … | … | … | … | … | … | … |
| 145 | 149 | 12 | 1 | 5 | 16 | 0 | 5 |
| 146 | 149 | 12 | 0 | 2 | 14 | 0 | 3 |
| 147 | 149 | 18 | 8 | 0 | 17 | 4 | 2 |
| 148 | 149 | 12 | 0 | 6 | 12 | 0 | 3 |
| 149 | 149 | 15 | 0 | 0 | 17 | 0 | 0 |

Dari nilai piksel citra stego tanpa penambahan RGB dan citra stego dengan penambahan RGB sebanyak 5 seperti pada Tabel 4.8 menunjukkan nilai piksel berubah bukan hanya karena penyisipan tetapi juga karena adanya penambahan intensitas warna. Kemudian citra stego dengan penambahan intensitas tersebut akan diekstraksi, Gambar 4.13 menunjukkan hasil ekstraksi citra *stego* tersebut :

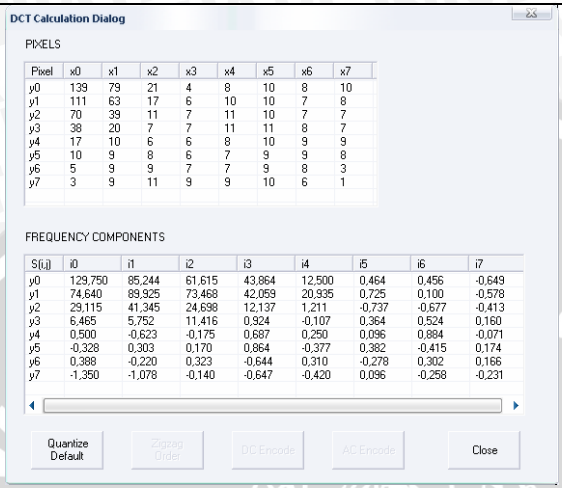

**Gambar 4.13** Hasil ekstraksi citra *stego* dengan penambahan intensitas warna.

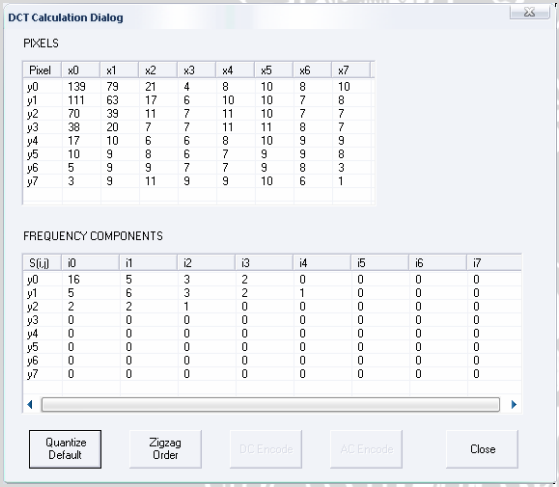
Gambar 4.13 menunjukkan bahwa pesan tidak dapat diekstraksi kembali. Dan Gambar 4.14 menunjukkan perbandingan Nilai piksel *gray scale*, koefisien DCT dan kuantisasi antara citra stego tanpa penambahan intensitas warna dan citra *stego* dengan penambahan intensitas warna (penambahan RGB sebanyak 5) :




a. Nilai piksel *gray scale*, koefisien DCT dan nilai kuantisasi citra stego tanpa penambahan intensitas (pic1hasilsteg.jpg)




b. Nilai piksel *gray scale*, koefisien DCT dan nilai kuantisasi citra stego dengan penambahanRGB =5 (pic1hasilstegRGB5r.jpg)

**Gambar 4.14** Nilai piksel *gray scale*, koefisien DCT dan kuantisasi pic1hasilsteg.jpg dan pic1hasilstegRGB5r.jpg.

Dari kedua uji coba tersebut diperoleh informasi bahwa ternyata pesan yang disisipkan gagal untuk diestraksi apabila media JPEG (media *stego*) yang digunakan dikenai perubahan. Hal ini terjadi karena palet warna atau nilai piksel (RGB) pun berubah ketika sebuah berkas *stego* atau *stego image* diberi penambahan *image distortion*.

Sehingga dapat disimpulkan bahwa algoritma F5 mempunyai kelemahan yaitu tidak tahan atau tidak *robust* terhadap operasi manipulasi pengolahan citra atau *image distortion*, hal ini salah satunya disebabkan oleh adanya perubahan warna tiap –tiap piksel citra seperti pada operasi penambahan efek yaitu efek *Gaussian blur* dan penambahan intensitas warna.

# BAB V
# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Kesimpulan yang didapat dalam pengerjaan Tugas Akhir ini adalah :

1. Steganografi dapat digunakan untuk komunikasi rahasia tanpa mencurigakan karena media penyimpanannya berupa gambar digital yang masih dapat dilihat dengan mata tanpa ada suatu kejanggalan.
2. Algoritma F5 menawarkan kapasitas penyimpanan data yang besar, efisiensi yang besar, *Matrix Encoding* yang lebih aman dan *Permutative Straddling* yang membuat *F5* menjadi sistem yang dapat diunggulkan.
3. Kualitas citra input format BMP lebih baik daripada citra input format JPEG. Hal ini dapat dilihat dari perhitungan Root Mean Square (RMS) yaitu rata-rata RMS dari 30 citra input uji format BMP adalah sebesar 1,367% sedangkan rata-rata RMS dari 30 citra input uji format JPEG adalah sebesar 1,418%.
4. Algoritma F5 ini memiliki kelemahan yaitu citra yang telah disisipi pesan tidak tahan terhadap perubahan. Karena setelah dilakukan beberapa perubahan terhadap citra diperoleh hasil bahwa pesan tidak dapat lagi diekstrak. Hal ini disebabkan adanya perubahan warna tiap-tiap piksel citra seperti pada operasi penambahan efek yaitu efek *Gaussian blur* dan penambahan intensitas warna atau penambahan RGB.

## 5.2 Saran

Beberapa saran untuk pengembangan lebih lanjut yang dapat diberikan oleh Penulis adalah :

1. Format gambar yang dipakai sebaiknya tidak hanya menggunakan format file BMP dan JPEG saja, tetapi bisa format gambar yang lain.
2. Untuk pengembangan selanjutnya diharapkan pesan yang disisipkan dapat diekstrak kembali apabila citra stego tersebut diberi operasi manipulasi pengolahan citra atau *image distortion*.
3. Apabila ingin mengembangkan lebih jauh lagi, steganografi diharapkan dapat membantu menyembunyikan data dalam *video*

atau *audio*, tidak hanya data berupa teks saja yang dapat disembuyikan.

4. Penambahan penggunaan algoritma enkripsi pada pesan yang akan disisipkan akan menambah keamanan data yang ingin disisipkan.
5. Penambahan penggunaan algoritma kompresi pesan akan menambah jumlah pesan yang akan disisipkan.

## DAFTAR PUSTAKA


Achmad, Balza dan Kartika Firdausy. 2005. *Teknik Pengolahan Citra Digital Menggunakan Delphi*. Ardi Publishing. Yogyakarta.

Adythia, Rhesa. 2007. *Studi dan Deteksi Steganografi pada File Bertipe JPEG dengan Tiga Steganographic System*. http://www.informatika.org/~rinaldi/Matdis/2006-2007/ Makalah/Makalah0607-107.pdf. tanggal akses : 01 Juni 2008

Alam, Ibnu. 2008. *Aplikasi Kode Huffman dalam Kompresi Gambar Berformat JPEG*. http://www.informatika.org/~rinaldi/Matdis/2007-2008 /Makalah/MakalahIF2153-0708-075.pdf, tanggal akses : 01 Juni 2008.

Daryanto, Tri. 2007. *Analisa Perbandingan Standar-Standar Kompresi Pada Gambar*. http://jurnal.bl.ac.id/wp-content/uploads/2007/01/BIT-v3=n1-artikel3-sept2007.pdf, tanggal akses : 01 Juni 2008.

Fridrich, J, Miroslav Goljan, Dorin Hogea. 2002. *Steganalysis of JPEG Images : Breaking The F5 Algorithm*. http://www.ws.binghamton.edu/fridrich/Research/f5.pdf., tanggal akses : 18 September 2007.

Henry. 2006. *Video Steganography*. http://www.informatika.org/~rinaldi/Kriptografi/2006-2007/henry_report.pdf. , tanggal akses : 07 Maret 2007.

Irianto. 2004. *Embedding Pesan Rahasia Dalam Gambar*. http://www.cert.or.id/~budi/courses/ec7010/2004-2005/irianto-report.pdf., tanggal akses : 18 September 2007.

Jendricke, Uwe. 2003. *Steganography and Watermaking*. http://www.iig.uni-freiburg.de/n.pdf., tanggal akses : 07 Maret 2007.

Latifawariq. 2007. *Ketahanan Digital Watermarking Pada Citra Terhadap Berbagai Image Distortion Dalam Sebaran Frekuensi Dengan Discrete Cosine Transform*. Skripsi FMIPA Universitas Brawijaya. Malang. Tidak dipublikasikan.

Mohanty, Saruju P. 1999. *Digital Watermarking : A Tutorial Review*. http://www.cs.unt.edu/~smohanty/research/Reports/Mohanty

WatermarkingSurvey1999.pdf., tanggal akses : 07 Maret 2007.

Munir, Rinaldi. 2004. *Pengolahan Citra Digital dengan Pendekatan Algoritmik*. Informatika, Bandung.

Murinto. 2005. *Penyisipan Robust Watermark Dalam Suatu Citra Untuk Perlindungan Hak Cipta*. http://www.murintokusno.com/publikasi_files/petra_2005.pdf., tanggal akses : 07 September 2007.

Penalosa, Ronald.A. 2005. *Steganografi Pada Citra Dengan Format GIF Menggunakan Algoritma GifShuffle*. http://www.informatika.org/~rinaldi/Kriptografi/2006-2007/Makalah1/Makalah1-053.pdf. , tanggal akses : 07 Maret 2007.

Provos, Niels & Honeyman, Peter. Hide and Seek : *An Introduction to Steganopgraphy*. www.eng.auburn.edu/csse/classes/comp7370/lessons/Steganography.pdf., tanggal akses : 18 September 2007.

Scruggs, J. 2003. *Color Theory*. http://www.bway.net/~jscruggs/color2.html., tanggal akses : 18 September 2007.

Setiana. 2006. *Steganografi Pada File Citra Bitmap 24 bit Untuk Pengamanan Data Menggunakan Metode Least Significant Bit (LSB) Insertion*. Skripsi FMIPA Universitas Brawijaya, Malang. Tidak dipublikasikan.

Sulistyanto, Hermawan. 2002. *Implementasi Alihragam Wavelet untuk Pemampatan Data Citra*. http://eprints.ums.ac.id/776/1/Emitor_HNS_ImpelementasiAlihRagamWavelet.pdf., tanggal akses : 18 September 2007.

Supangkat, H.S, Kuspriyanto, dan Juanda. 2000. *Watermaking Sebagai Teknik Penyembunyian Label Hak Cipta Pada Data Digital*. http://digitally1.paume.itb.ac.id/MITE/Paper%206%203%202000%203.pdf. , tanggal akses : 07 Maret 2007.

Westfeld, Andreas. 2001. *F5-A Steganographic Algorithm*. www.os.inf.tu-dresden.de/~westfeld/publikationen/f5.pdf., tanggal akses : 18 September 2007.