BAB I
PENDAHULUAN

1.1 Latar Belakang

Steganografi merupakan ilmu yang mempelajari, meneliti, dan mengembangkan seni menyembunyikan informasi. Steganografi digolongkan bagian dari ilmu komunikasi. Steganografi seni yang sudah lama digunakan di dunia ini. kata Steganografi berasal dari bahasa Yunani yang berarti "tulisan tersembunyi". Dahulu sering dilakukan untuk menyampaikan pesan rahasia. Misalnya, menato pesan rahasia di kulit kepala para pembawa pesan (Yudi Prayudi, 2005).

Steganografi di era digital merupakan teknik dan seni menyembunyikan informasi di balik informasi digital lain, sehingga informasi digital sesungguhnya tidak terdeteksi. Tujuan utamanya untuk menjaga kerahasiaan informasi sesungguhnya yang disampaikan.

Hampir semua kegiatan manusia modern menggunakan informasi digital. Mulai dari belajar di sekolah, bermain, menyelesaikan pekerjaan kantor, menyimpan arsip penting, termasuk berkomunikasi sehari – hari menggunakan komunikasi digital. Sedemikian luasnya penggunaan media digital dalam komunikasi, teknologi Steganografi menjadi luas implementasinya. Banyak yang dapat Anda lakukan memanfaatkan teknologi ini untuk tujuan baik misal kepentingan bisnis seperti *Copyright rotection,* yang mencegah terjadinya pembajakan dari produk digital, digital *watermarking* untuk menandai kepemilikan dan perizinan. Penggunaan dengan tujuan netral dilakukan seseorang yang menyukai Steganografi atau amatiran sekadar untuk kesenangan atau kepentingan lain yang tidak mengganggu. Peneliti juga menggunakan teknik Steganografi untuk melindungi hasil penelitiannya, berkomunikasi dengan peneliti lain tentang hasil penelitian rahasia. Yang ditakutkan penggunaan Steganografi untuk tujuan jahat. Untuk mengetahui dan mencegah tujuan jahat, sangat sulit jika disertai dengan penggunaan Steganografi di dalamnya sehingga kejahatan aman terlindung. *Firewall* jenis apapun, konfigurasi *filtering* secanggih apapun, sistem *Intrusion Detection* sehebat apapun, tidak akan dapat mengetahui dan mencegah informasi jahat keluar masuk. Untuk itu, sebaiknya

hindarilah penggunaan teknologi ini untuk kepentingan yang mencelakakan orang lain (Ermadi Satriya Wijaya, 2009).

File media merupakan komponen penting pada proses penyembunyian informasi. Dengan *file* yang terlihat sama sekali tidak mencurigakan, data Anda yang sebenarnya tidak terdeteksi dengan mata telanjang. Secara teori, semua *file* umum yang ada di dalam komputer dapat digunakan sebagai media, seperti *file* gambar ber*format* JPG, GIF, BMP, musik MP3, dan film dengan *format* WAV atau AVI. Semua bisa dijadikan tempat bersembunyi, asalkan *file* tersebut memiliki *bit-bit* data redundan yang dimodifikasi. *Bit-bit* data redundan adalah *bit-bit* data ganda yang jika dimodifikasi, maka kualitas dan tampilan *file* yang sesungguhnya tidak terganggu fungsinya dan kualitasnya tidak jauh berbeda dengan aslinya (Temmy Maradilla, 2009).

Secara garis besar, teknik penyembunyian data dengan Steganografi adalah dengan cara menyisipkan sepotong demi sepotong informasi asli pada media, sehingga informasi tampak kalah dominan dengan media pelindungnya. Teknik Steganografi beserta aplikasinya dapat di temui dengan mudah untuk berkomunikasi. Komunikasi aman mungkin istilah yang cukup mewakili hasil dari teknologi Steganografi (Temmy Maradilla, 2009).

Semisal ada karya yang dipermasalahkan kepemilikannya, maka yang mengetahui keaslian karya tersebut adalah pembuatnya sendiri. Karena pembuat karya tersebut memiliki tanda unik untuk membedakan hasil karyanya dengan yang lain. Tetapi akan menjadi permasalahan ketika pembuat karya meninggal dunia. Maka, akan banyak orang yang mengakui karya tersebut. Dan di sini aplikasi Steganografi akan membantu untuk mengamankan dari orang orang yang tidak bertanggung jawab.

Teknologi digital memberikan kontribusi besar pada penerapan teknologi Steganografi, karena banyak *format file* digital dapat dijadikan media menyembunyiakan pesan. *Format* yang biasanya digunakan diantaranya: *bit*map (bmp), gif, png, jpeg, *file* teks, html, pdf, mp3, wav atau voc. Contohnya pada *file* gambar, pesan disembunyikan dengan menyisipkan pada *bit* rendah (LSB) di dalam data *pixel* yang menyusun *file* gambar. Metode *Least*

Significant Bit adalah metode Steganografi yang sederhana dan mudah dipahami.

1.2 Rumusan Masalah

Rumusan masalah dari skripsi ini adalah:

1. Bagaimana implementasi Steganografi citra BMP menggunakan metode LSB.
2. Bagaimana tingkat uji kualitas gambar setelah disisipi pesan dengan prosentase yang berbeda-beda terhadap gambar asli.

1.3 Batasan Masalah

Batasan masalah pada skripsi ini antara lain:

1. Menggunakan gambar ber*format* Bmp.
2. Gambar yang di *input*kan minimal berkuran lebar 24 *pixel* dan tinggi minimal 2 *pixel*.
3. Pesan yang di sembunyikan adalah karakter yang disisipkan pada program.
4. Enkripsi *password* pada sistem ini menggunakan *library* Delphi yaitu enkripsi RC 4 (*Rivest Cipher 4*).
5. Gambar yang di *decode* tidak boleh melalui proses perubahan *pixel*.
6. Perhitungan manual proses uji kualitas menggunakan metode PSNR.

1.4 Tujuan

Tujuan dari penelitian ini adalah :

1. Mengimplementasikan Algoritma Steganografi metode LSB pada citra BMP.
2. Menghitung nilai uji kualitas antara gambar yang disisipi pesan dengan prosentase yang berbeda-beda terhadap gambar asli.

1.5 Manfaat

Manfaat yang diperoleh dari penelitian ini adalah sebagai berikut :

1. Menghasilkan sistem aplikasi komputer yang berfungsi untuk menyisipkan pesan kedalam gambar ber*format* BMP menggunakan metode LSB.
2. Memperoleh gambar ber*format* BMP yang telah disisipi pesan.
3. Memperoleh nilai perbandingan uji kualitas antara gambar yang disisipi pesan dengan gambar asli.

1.6 Sistematika Penulisan

Tugas akhir ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN

   Berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi pemecahan masalah, dan sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA

   Menguraikan teori-teori yang berhubungan dengan Steganografi menggunakan metode LSB dan kompresi menggunkan Metode *descrete cosine transform (dct).*
3. BAB III METODOLOGI DAN PERANCANGAN SISTEM

   Pada bab ini dijelaskan mengenai metode-metode yang digunakan dalam pembuatan program Steganografi menggunakan metode LSB dan kompresi menggunakan metode *descrete cosine transform (dct).*
4. BAB IV HASIL DAN PEMBAHASAN

   Pada bab ini dilakukan implementasi program, pengujian dan analisa sistem perangkat lunak yang dibangun, yaitu apakah sesuai dengan metode LSB dan *descrete cosine transform (dct).*
5. BAB V KESIMPULAN DAN SARAN

   Berisi kesimpulan dari seluruh rangkaian penelitian serta saran kemungkinan pengembangannya.

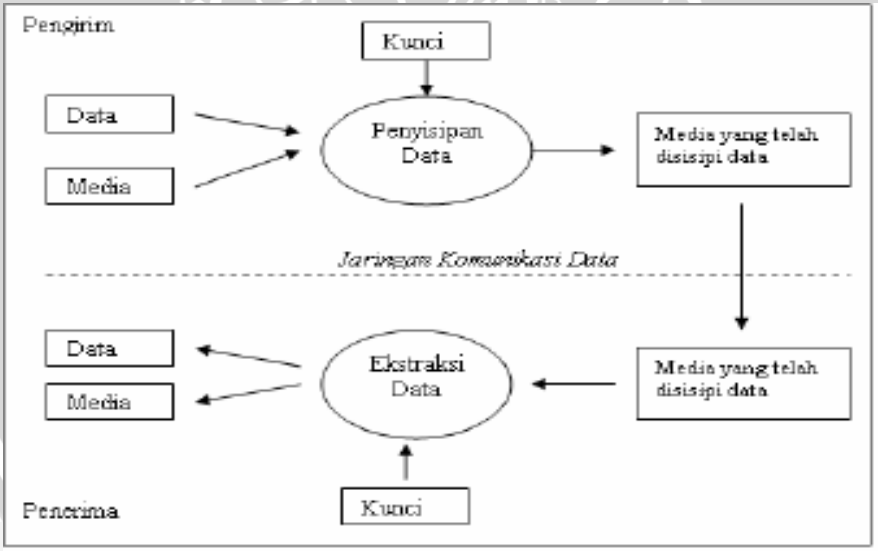
BAB II
TINJAUAN PUSTAKA

2.1 Definisi *Steganography*

Steganography adalah ilmu dan seni menyembunyikan pesan rahasia *(hiding message)* sehingga keberadaan *(eksistensi)* pesan tidak terdeteksi indera manusia. Kata steganorafi berasal dari Bahasa Yunani yang berarti "tulisan tersembunyi" *(covered writing)*. *Steganography* membutuhkan dua properti yaitu wadah penampung dan data rahasia yang akan disembunyikan (Ahmad, Usman, 2005).

Steganography dipandang sebagai kelanjutan kriptografi. Jika pada kriptografi, data yang disandikan *(ciphertext)* tetap tersedia, maka dengan *steganography* cipherteks dapat disembunyikan sehingga pihak ketiga tidak mengetahui keberadaannya. Di negara-negara yang melakukan penyensoran informasi, *steganography* sering digunakan untuk menyembunyikan pesan-pesan melalui gambar *(images)*, video, atau suara *(audio)* (Desiani, Anita, 2005).

Ilustrasi mengenai proses *steganography* dapat digambarkan seperti pada Gambar 2.1.

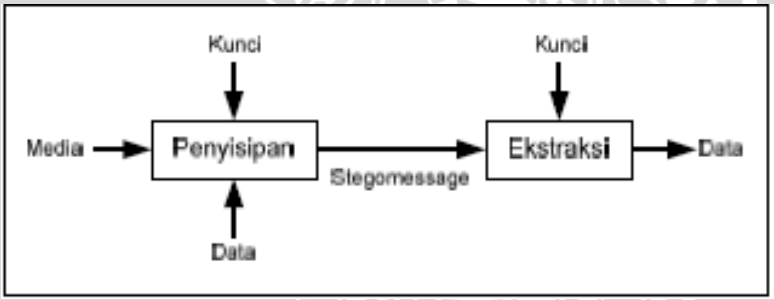


Gambar 2.1 Diagram Sistem *Steganography* (Munir, Rinaldi, 2004).

Steganography memanfaatkan keterbatasan sistem indera manusia seperti mata dan telinga. Dengan keterbatasan inilah, metoda *steganography* diterapkan pada berbagai media digital. Hasil keluaran *steganography* memiliki bentuk persepsi yang sama dengan bentuk aslinya, sebatas kemampuan indera manusia, tetapi tidak oleh komputer atau perangkat pengolah digital lainnya (Desiani, Anita, 2005).

Steganography digital menggunakan media digital sebagai wadah penampung, misalnya citra, suara, teks, dan video. Sedangkan data rahasia yang disembunyikan berupa berkas apapun. Media yang disisipi data disebut stegomessage. Proses penyembunyian data ke dalam media disebut penyisipan (*embedding*), sedangkan proses sebaliknya disebut ekstraksi. Proses tersebut dapat dilihat pada penambahan kunci yang bersifat opsional yang dimaksudkan untuk lebih meningkatkan keamanan (Suharto, Edy, 2004).

Ilustrasi mengenai proses Penyisipan dan Ekstraksi *dalam steganography* dapat digambarkan seperti pada Gambar 2.2.



Gambar 2.2 Proses Penyisipan dan Ekstraksi dalam *Steganography* (Suharto, Edy, 2004)

2.2 Sejarah *Steganography*

Steganography sudah dikenal bangsa Yunani. Herodatus, penguasa Yunani, mengirim pesan rahasia menggunakan kepala budak atau prajurit sebagai media. Rambut budak dibotaki, lalu pesan rahasia ditulis pada kulit kepala budak. Ketika rambut budak tumbuh, budak tersebut diutus membawa pesan rahasia di balik rambutnya.

Bangsa Romawi mengenal *steganography* dengan menggunakan tinta tak-tampak *(invisible ink)* untuk menuliskan

pesan. Tinta tersebut dari campuran sari buah, susu, dan cuka. Jika tinta digunakan menulis maka tulisannya tidak tampak. Tulisan di atas kertas dapat dibaca dengan cara memanaskan kertas tersebut.

2.3 Kriteria *Steganography* yang Bagus

Steganography yang dibahas adalah penyembunyian data di dalam citra digital saja. Meskipun demikian, penyembunyian data dapat juga dilakukan pada wadah berupa suara digital, teks, ataupun video. Penyembunyian data rahasia ke dalam citra digital akan mengubah kualitas citra tersebut (Munir, Rinaldi, 2004).

Kriteria yang harus diperhatikan dalam penyembunyian data adalah:

1. *Fidelity*: Mutu citra penampung tidak jauh berubah. Setelah penambahan data rahasia, citra hasil *steganography* masih terlihat dengan baik. Pengamat tidak mengetahui kalau di dalam citra tersebut terdapat data rahasia.
2. *Robustness*: Data yang disembunyikan harus tahan terhadap manipulasi yang dilakukan pada citra penampung (seperti pengubahan kontras, penajaman, pemampatan, rotasi, perbesaran gambar, pemotongan *(cropping)*, enkripsi, dan sebagainya). Bila pada citra dilakukan operasi pengolahan citra, maka data yang disembunyikan tidak rusak.
3. *Recovery*: Data yang disembunyikan harus dapat diungkapkan kembali *(recovery)*. Karena tujuan *steganography* adalah data hiding, maka sewaktu-waktu data rahasia di dalam citra penampung harus dapat diambil kembali untuk digunakan lebih lanjut.

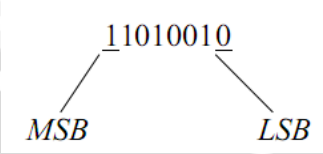
2.4 Metode dalam *Steganography*

Kebanyakan algoritma *steganography* menggunakan kombinasi dari bidang jenis teknik untuk melakukan tugas penyelubungan pesan rahasia. Program *steganography* dibutuhkan untuk melakukannya, baik secara implisit melalui perkiraan maupun eksplisit melalui perhitungan, menemukan kelebihan *bit* dalam selubung *file* yang digunakan menyelubungi pesan rahasia didalamnya (Ahmad, Usman, 2005).

2.4.1 Least Significant *Bit* (LSB)

Penyembunyian data dilakukan dengan mengganti *bit-bit* data di dalam segmen citra dengan *bit-bit* data rahasia. Metode yang paling sederhana adalah metode modifikasi LSB *(Least Significant*

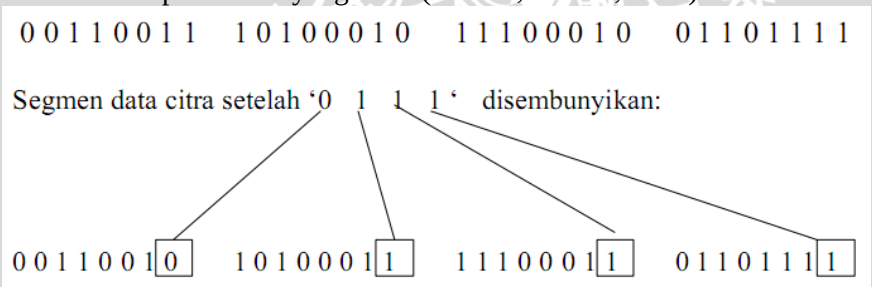
Bit Modification). Pada susunan *bit* di dalam sebuah *byte* (1 *byte* = 8 *bit*), ada *bit* yang paling berarti (*most significant bit* atau MSB) dan *bit* yang paling kurang berarti (*Least Significant Bit* atau LSB*)*.



Gambar 2.3 Susunan *Bit* pada Sebuah *Byte*
(Munir, Rinaldi, 2004).

Keterangan :
LSB = *Least Significant Bit*
MSB = *Most Siginificant Bit*

Bit yang cocok untuk diganti adalah *bit* LSB, sebab perubahan tersebut hanya mengubah nilai *byte* satu lebih tinggi atau satu lebih rendah dari nilai sebelumnya. Misalkan *byte* tersebut menyatakan warna merah, maka perubahan satu *bit* LSB tidak mengubah warna merah tersebut secara berarti. Lagi pula, mata manusia tidak dapat membedakan perubahan yang kecil (Ahmad, Usman, 2005).



Gambar 2.4 Segmen Data Citra Sebelum Perubahan
(Munir, Rinaldi, 2004).

Untuk memperkuat teknik penyembunyian data, *bit-bit* data rahasia tidak digunakan mengganti *byte-byte* yang berurutan, namun dipilih susunan *byte* secara acak. Misalnya jika terdapat 50 *byte* dan 6 *bit* data yang disembunyikan, maka maka *byte* yang diganti *bit* LSB-nya dipilih secara acak, misalkan *byte* nomor 36, 5, 21, 10, 18, 49.

Bilangan acak dapat dibangkitkan dengan program *pseudo-random-number-generator* (PRNG). PRNG menggunakan kunci rahasia untuk membangkitkan posisi *pixel* yang digunakan untuk

menyembunyikan *bit-bit*. PRNG dibangun dalam sejumlah cara, salah satunya dengan menggunakan algoritma kriptografi berbasis blok *(block cipher)*. Tujuan enkripsi adalah menghasilkan sekumpulan bilangan acak yang sama untuk setiap kunci enkripsi yang sama. Bilangan acak dihasilkan dengan memilih *bit-bit* sebuah blok data hasil enkripsi. Dan metode modifikasi LSB kurang bagus untuk *steganography*, karena robustness-nya rendah. Selain itu, dapat terjadi kasus penurunan jumlah warna *(fidelity rendah)* (Munir, Rinaldi, 2004).

2.4.1.1 Gambar 24-*bit*

Untuk menyembunyikan gambar dalam LSB pada setiap *byte* dari gambar 24-*bit*, dapat disimpan 3 *byte* dalam setiap *pixel*. Gambar 1,024 x 768 mempunyai potensi untuk disembunyikan seluruhnya dari 2,359,296 *bit* (294,912 *byte*) pada informasi. Jika pesan tersebut dikompres untuk disembunyikan sebelum ditempelkan, dapat menyembunyikan sejumlah besar dari informasi. Pada pandangan mata manusia, hasil *stego-image* akan terlihat sama dengan gambar cover (Robi'in, Bambang, 2007).

Untuk contoh huruf A dapat disembunyikan dalam tiga *pixel* (asumsikan tidak ada kompresi). *Raster* data asli untuk 3 *pixel* (9 *byte*) menjadi

(00100111 11101001 11001000)
(00100111 11001000 11101001)
(11001000 00100111 11101001)

Nilai biner untuk A adalah 10000011. Sisipan nilai biner untuk A dalam tiga *pixel* akan menghasilkan

(00100111 <u>11101000</u> 11001000)
(<u>00100110</u> 11001000 <u>11101000</u>)
(11001000 00100111 11101001)

Bit-bit yang digaris bawahi hanya tiga perubahan secara aktual dalam 8 *byte* yang digunakan. Secara rata-rata, LSB membutuhkan hanya setengah *bit* dalam perubahan gambar. Menyembunyikan data dalam least dan second *Least Significant Bit* manusia masih belum dapat membedakannya.

Ukuran data yang akan disembunyikan bergantung pada ukuran citra penampung. Pada citra 24-*bit* yang berukuran 256 *pixel* terdapat 65536 *pixel*, setiap *pixel* berukuran 3 *byte* (komponen RGB), berarti seluruhnya ada 65536 ´ 3 = 196608 *byte*. Karena setiap

byte hanya bisa menyembunyikan satu *bit* di LSB-nya, maka ukuran data yang akan disembunyikan di dalam citra maksimum. 196608/8 = 24576 *byte*

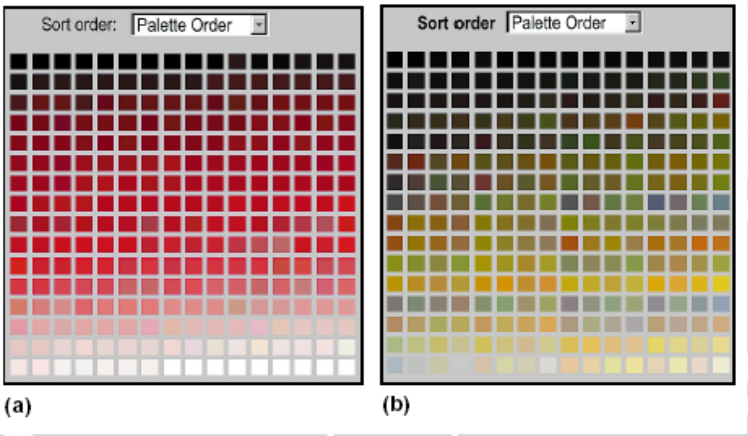
Ukuran data ini harus dikurangi dengan panjang nama berkas, karena penyembunyian data rahasia tidak hanya menyembunyikan isi data tersebut, tetapi juga nama berkasnya. Semakin besar data disembunyikan, semakin besar pula kemungkinan data tersebut rusak akibat manipulasi citra penampung (Lestari, Desi, 2003).

2.4.1.2 Gambar 8-*bit*

Gambar 8-*bit* tidak diberikan untuk manipulasi LSB karena keterbatasan warnanya. Gambar cover harus hati-hati diseleksi sehingga *stego-image* tidak akan mem-*broadcast* keberadaannya pada pesan yang ditempelkan. Ketika informasi disisipkan ke dalam LSB dari *raster data*, penunjuk kemasukan warna dalam *palette* yang diubah. Dalam suatu contoh, suatu *palette* sederhana empat warna dari putih, merah, biru dan hijau mempunyai posisi masukan *palette* yang sesuai secara berturut-turut dari 0 (00), 1 (01), 2 (10), dan 3 (11). Nilai *raster* dari empat *pixel* yang bersebelahan dari putih, putih, biru dan biru adalah 00 00 10 10. Penyembunyian nilai biner 1010 untuk perubahan bilangan 10 *raster data* ke 01 00 11 10, adalah merah, putih, hijau dan biru (Woods, R.E., 2005).

2.4.1.3 Indeks Warna (Pallet)

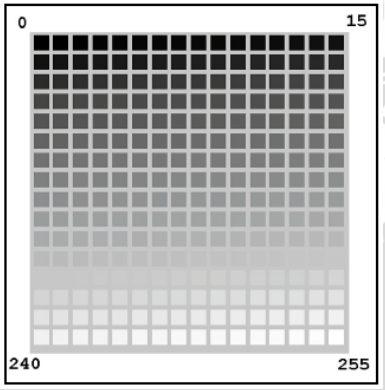
Kebanyakan *software steganography* tidak mendukung menggunakan gambar JPEG, sebagai gantinya direkomendasikan menggunakan gambar *lossless* 24-*bit* seperti BMP. Alternatif terbaik berikutnya untuk gambar 24-*bit* adalah 256 warna atau gambar *gray scale*. Dalam gambar 8-*bit* warna seperti *file* GIF, setiap *pixel* direpresentasikan sebagai *byte* tunggal, dan setiap *pixel* selalu menunjuk ke tabel indek warna (*palette*) dengan 256-kemungkinan warna. Nilai *pixel* adalah diantara 0 dan 255. *Software* secara sederhana menggambarkan indikasi warna pada *palette* merah, menggambarkan perubahan yang sulit dipisahkan dalam variasi warna, perbedaan visualisasi diantara banyak warna yang sulit. Perubahan warna yang sulit dipisahkan dengan baik dapat digambarkan seperti pada Gambar 2.5.



Gambar 2.5 Representasi Warna Pallette
(Neil F. Johnson, Sushil Jajodia, 2003)

Banyak pakar *steganography* merekomendasikan penggunaan gambar meliputi 256 *shade gray*. Gambar gray-scale disukai karena perubahan keteduhan sangat gradual dari *byte* ke *byte*, lebih sedikit perubahan nilai diantara masukan *palette*, dan dapat menyembunyikan informasi lebih baik.

Suatu *palette gray-scale* dari 256 *shade* dapat digambarkan seperti pada Gambar 2.6.



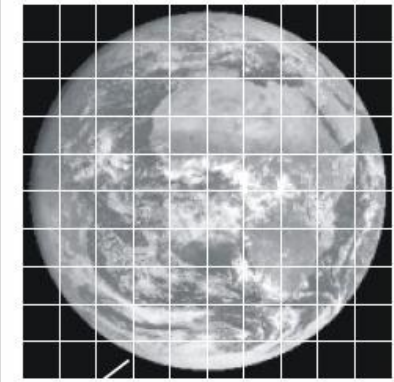
Gambar 2.6 Representasi Gray-Scale *Palette* dari 256 Shade
(Neil F. Johnson, Sushil Jajodia, 2003)

2.5 Citra Digital

Citra adalah gambar dua dimensi yang dihasilkan dari gambar analog dua dimensi yang kontinu menjadi gambar diskrit melalui proses *sampling.* Gambar analog dibagi menjadi N baris dan M kolom sehingga menjadi gambar diskrit. Dimana setiap pasangan indeks baris dan kolom menyatakan suatu titik pada citra. Nilai matriksnya menyatakan nilai kecerahan titik tersebut. Titik-titik tersebut dinamakan sebagai elemen citra, atau *pixel* (*picture element*). Dalam kamus komputer, gambar atau foto diistilahkan sebagai citra digital yang mempunyai representasi matematis berupa matriks $Cm \times n = (cij)$.

Gonzales *and* Woods (1992) mendefinisikan citra digital sebagai fungsi intensitas cahaya dua-dimensi f(x,y) dimana x dan y menunjukkan koordinat spasial, dan nilai f pada suatu titik (x,y) sebanding dengan *brightness* (*gray level*) dari citra di titik tersebut.

Citra digital dapat digambarkan seperti pada Gambar 2.7.



Gambar 2.7 Representasi citra digital
(Munir, Rinaldi, 2004).

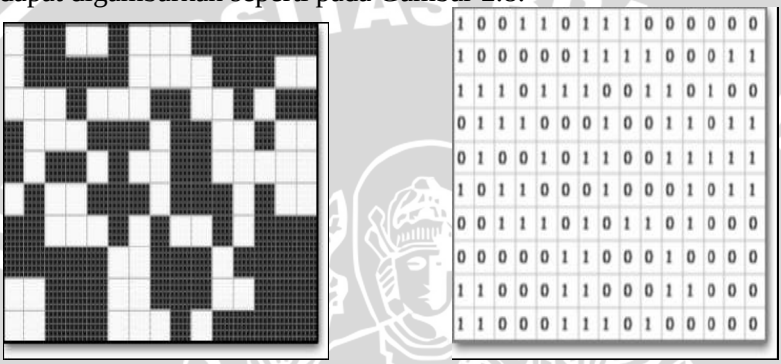
2.5.1 Matriks *bitmap*

Citra *bitmap* adalah susunan *bit-bit* warna untuk tiap *pixel* yang membentuk pola tertentu. Pola-pola warna ini menyajikan informasi yang dapat dipahami sesuai dengan persepsi indera penglihatan manusia. *Format file* ini merupakan *format* grafis yang fleksibel untuk *platform Windows* sehingga dapat dibaca oleh

program grafis manapun. *Format* ini mampu menyimpan informasi dengan kualitas tingkat 1 *bit* samapi 24 *bit*. (Agus, Prijono, 2007).

Citra *bit*map didefinisikan sebagai fungsi f ─(x,y) dengan x dan y adalah koordinat bidang. Besaran f untuk tiap koordinat (x,y) disebut intensitas atau derajat keabuan citra pada titik tersebut.

Pada gambar 2.14 ditunjukkan gambar *bitmap* beserta nilai dapat digambarkan seperti pada Gambar 2.8.



(a) *bitmap* 15×10 *pixel* (b) Matriks *bitmap*

Gambar 2.8 *Bitmap* dengan nilai matriksnya (Munir, Rinaldi, 2004).

Dari definisi di atas yang diperjelas oleh gambar 1.1, *bitmap* dimodelkan dalam bentuk matriks. Nilai *pixel* atau entri-entri dari matriks ini mewakili warna yang ditampilkan dimana ordo matriks merupakan dimensi panjang dan lebar dari *bitmap*.

Nilai-nilai warna ditentukan berdasarkan intensitas cahaya yang masuk. Dalam komputer, derajat intensitas cahaya diwakili oleh bilangan cacah. Nilai 0 menerangkan tidak adanya cahaya sedangkan nilai yang lain menerangkan adanya cahaya dengan intensitas tertentu. Nilai-nilai ini bisa didapatkan melalui fungsi-fungsi yang disediakan oleh bahasa pemrograman berdasarkan *input* berupa lokasi entri-entri matriks yang hendak dicari (Lestari, Desi, 2003).

2.5.2 *Pixel*

Pixel (*Picture Elements*) adalah nilai tiap-tiap entri matriks pada *bitmap*. Rentang nilai-nilai *pixel* ini dipengaruhi oleh banyaknya warna yang dapat ditampilkan. Jika suatu *bitmap* dapat

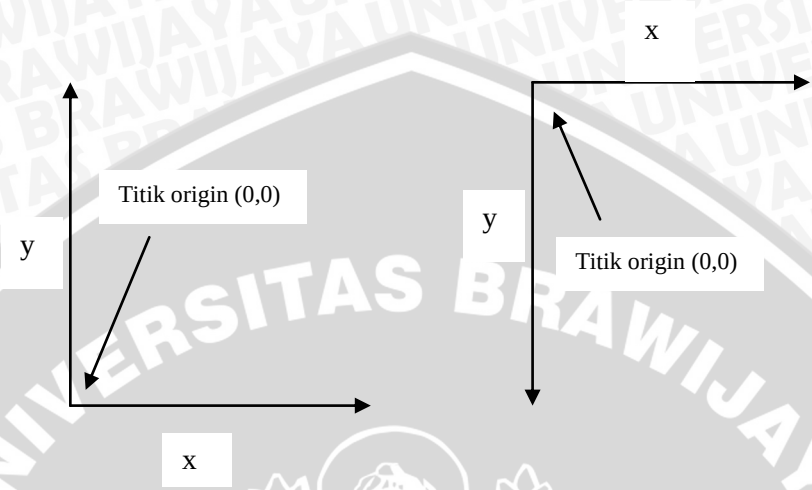
menampilkan 256 warna maka nilai-nilai *pixel*nya dibatasi dari 0 hingga 255. Suatu *bitmap* dianggap mempunyai ketepatan yang tinggi jika dapat menampilkan lebih banyak warna (Robi'in, Bambang, 2004).

Prinsip ini dapat dilihat dengan memberikan contoh, dua buah *bitmap* dapat memiliki perbedaan dalam menangani transisi warna putih ke warna hitam yang dapat digambarkan seperti pada Gambar 2.9.

Gambar 2.9 Perbedaan ketepatan warna *bitmap* (Munir, Rinaldi, 2004).

Perbedaan ketepatan warna *bitmap* pada gambar 2 menjelaskan bahwa *bitmap* sebelah atas memberikan nilai untuk warna lebih sedikit daripada *bitmap* dibawahnya. Untuk *bitmap* dengan pola yang lebih kompleks dan dimensi yang lebih besar, perbedaan keakuratan dalam memberikan nilai warna akan terlihat lebih jelas.

Menurut Usman Ahmad (2005) sebuah *pixel* adalah sampel dari pemandangan yang mengandung intensitas citra yang dinyatakan dalam bilangan bulat. Sebuah citra adalah kumpulan *pixel-pixel* yang disusun dalm larik dua dimensi. Indeks baris dan kolom (x,y) dari sebuah *pixel* dinyatakan dalam bilangan bulat. *Pixel* (0,0) terletak pada sudut kiri atas pada citra, indeks x begerak ke kanan dan indeks y bergerak ke bawah. Konvensi ini dipakai merujuk pada cara penulisan larik yang digunakan dalam pemrograman komputer. Letak titik origin pada koordinat grafik citra dan koordinat pada grafik matematika terdapat perbedaan. Hal yang berlawanan untuk arah vertikal berlaku pada kenyataan dan juga pada sistem grafik dalam matematika yang sudah lebih dulu dikenal. perbedaan kedua sistem ini dapat digambarkan seperti pada Gambar 2.10.



(a) koordinat pada grafik matematika (b) koordinat pada citra

Gambar 2.10 Perbedaan letak titik origin pada koordinat grafik dan pada citra
(Munir, Rinaldi, 2004).

2.5.3 Dimensi dan Resolusi

Dimensi *bitmap* adalah ukuran *bitmap* yang dinotasikan dengan menulis lebar × tinggi *bitmap*. Satuan ukur dimensi *bitmap* dapat berupa satuan ukur metris maupun *pixel*. Dimensi yang digunakan oleh *bitmap* mewakili ordo matriks citra itu sendiri. Contoh pada gambar 2.1 menunjukkan sebuah *bitmap* berdimensi 15×10 *pixel* yang diwakili oleh matriks C10×15. Model matriks untuk *bitmap* dipengaruhi oleh kerapatan *pixel* atau resolusi. Kerapatan *pixel* ini digunakan *bitmap* dalam mendekati kekontinyuan. Semakin besar resolusi suatu *bitmap*, obyek yang ditampilkan citra tersebut semakin akurat (Robi'in, Bambang, 2004).

Kerapatan titik-titik pada citra dinamakan resolusi, yang menunjukkan seberapa tajam gambar ini ditampilkan yang ditunjukkan dengan jumlah baris dan kolom. Resolusi merupakan ukuran kuantitas bukan kualitas. *Pixel* merupakan satuan ukuran terhadap jumlah area *photo-receptor* pada sensor gambar kamera, yang menentukan seberapa banyak data yang dapat ditangkap.

Resolusi digunakan untuk pendataan (*sampling*) citra dari sensor. Sensor mengubah citra dari fungsi kontinu ke fungsi diskrit

sehingga semakin besar resolusi citra maka informasi yang dihasilkan akan semakin baik, sebab data yang diperoleh menjadi lebih banyak.

2.5.4 Citra *Grayscale*

Grayscale (skala keabuan) merupakan suatu istilah untuk menyebutkan satu citra yang memiliki warna putih, abu-abu dan hitam. *Format* citra ini disebut skala keabuan karena pada umumnya warna yang dipakai adalah antara hitam sebagai warna minimal dan warna putih sebagai warna maksimalnya, sehingga warna antaranya adalah abu-abu (Robi'in, Bambang, 2004).

Pada citra digital banyaknya kemungkinan nilai dan nilai maksimumnya bergantung pada jumlah *bit* yang digunakan. Misalnya pada citra skala keabuan 4 *bit*, maka jumlah kemungkinan nilainya adalah $2^4 = 16$ dan nilai maksimumnya adalah $2^4 - 1 = 15$. Sedangkan untuk skala keabuan 8 *bit*, maka jumlah kemungkinan nilainya adalah $2^8 = 256$, dan nilai maksimumnya adalah $2^8 - 1 = 255$. Sehingga Makin besar angka *grayscale*, citra yang terbentuk makin mendekati kenyataan. (Balza, Kartika, 2005)

2.6 Kriptografi RC4

RC4 *adalah stream cipher* yang diciptakan oleh Ron Rivest pada tahun 1987. RC sendiri merupakan singkatan dari *Rivest Cipher* (kadang juga disebut Ron's *Code*). Pada awalnya algoritma RC4 ini dirahasiakan sampai dikirimkan ke milis Cypherpunks secara anonim pada tahun 1994. Setelah itu algoritma RC4 mulai diketahui banyak orang dan menyebar luas melalui internet. RC4 pun menjadi populer dan banyak dipakai untuk berbagai kegunaan, terutama dalam koneksi internet seperti Wi-Fi (Munir, Rinaldi, 2006).

Faktor-faktor dalam kesuksesan RC4 antara lain adalah kecepatan, efisien untuk diimplementasikan baik dalam *hardware* maupun *software*, dan mudah untuk dikembangkan. Serupa dengan skema umum stream cipher, RC4 membangkitkan *keystream* dengan *keystream* generator lalu dilakukan XOR antara key tersebut dengan *plainteks.*

Algoritma enkripsi RC4 beroperasi dalam *byte*, berarti XOR dilakukan setiap satu *byte plainteks* dengan satu *byte key* dari keystream. RC4 menggunakan fungsi dekripsi dan enkripsi yang sama karena operasi yang dilakukan hanyalah XOR antara *keystream* yang dibangkitkan dengan *plainteks*.

Keystream generator pada RC4 menggunakan Initial State yang berupa S-box yang berukuran 16x16. PertamaS-box diinisialisasi secara linear, berarti S[i]=i. Lalu dilakukan permutasi isi S-box dengan memanfaatkan kunci yang dimiliki. Algoritma dari RC4 secara umum adalah sebagai berikut (Rogaway Phillip) :

1. Inisialisasi S-box dengan linear.
2. Lakukan padding pada kunci apabila panjang kunci kurang dari 256.
3. Permutasi isi dari S-box dengan mengacaknya menggunakan kunci.
4. Bangkitkan *keystream* lalu di-XOR-kan dengan *plainteks.*

Permutasi pada S-box dilakukan seperti pada *pseudocode* berikut (S adalah S-box dan K adalah kunci yang telah di-padding jika perlu):

for i = 0 to 255:
j = (j + S[i] + K[i]) mod 256
swap S[i] and S[j]

Sementara itu dalam membangkitkan *keystream* algoritma yang dilakukan adalah sebagai berikut:

i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap S[i] and S[j]
t = (S[i] + S[j]) mod 256
K = S[t]

Setelah itu akan dilakukan operasi XOR antara K yang diperoleh dan satu *byte plainteks*, P[i]. Operasi permutasi dan pembangkitan keystream yang dilakukan sebelumnya sudah cukup baik untuk membuat K sebuah *byte* yang acak. Bahkan dengan permutasi tersebut bisa terdapat sebanyak 21700 (256 × 2562) kemungkinan S-box.

2.7 Peak Signal to Noise Ratio (PSNR)

Informasi yang hilang akibat steganografi seharusnya seminimal mungkin sehingga kualitas hasil steganografi bagus. Tetapi biasanya kualitas steganografi bagus bila proses steganografi menghasilkan kemiripan terhadap gambar asli (Munir, Rinaldi, 2004).

Rumus umum untuk mengukur kwalitas hasil steganografi dengan PSNR (*peak signal – to – noise ratio*) ditunjukkan dalam persamaan 2.1 :

$$PSNR = 20\ x\ \log_{10}\left(\frac{b}{rms}\right)$$

$$rms = \sqrt{\frac{1}{tinggixlebar}\sum_{i=1}^{N}\sum_{j=1}^{M}(f_{ij} - f'_{ij})^2} \qquad \textbf{(2.1)}$$

b = sinyal terbesar (pada citra hitam putih, b = 255)
rms = akar pangkat dua dari selisih antara citra semula dengan citra hasil kompresi
f = nilai piksel citra semula
f’ = nilai piksel citra kompresi

PSNR memiliki satuan *decibel* (dB). Semakin besar nilai PSNR, semakin bagus kualitas steganografi.

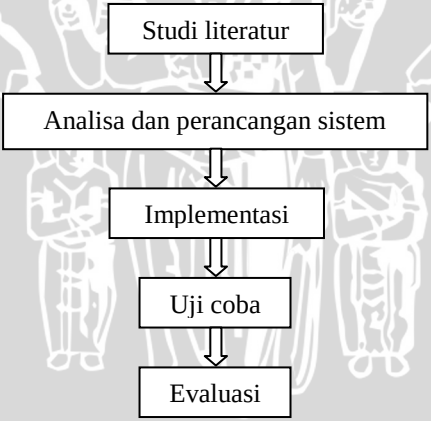
BAB III
METODOLOGI DAN PERANCANGAN

Pembahasan pada bab ini meliputi tentang desain, implementasi desain dan metode *output* dalam steganografi dan kemudian dilakukan uji kualitas. Desain dan implementasi ini meliputi deskripsi sistem, desain data, desain proses dan implementai desain.

Langkah – langkah yang dilakukan dalam penelitian ini meliputi :

1. Melakukan studi literatur mengenai steganografi dan uji kualitas.
2. Menganalisa dan melakukan perancangan sistem.
3. Mengimplementasikan rancangan yang dilakukan pada tahap sebelumnya menjadi sebuah perangkat lunak.
4. Melakukan uji coba terhadap perangkat lunak.
5. Mengevaluasi hasil analisa yang dilakukan oleh sistem.

Langkah – langkah penelitian ini dapat digambarkan seperti pada Gambar 3.1.

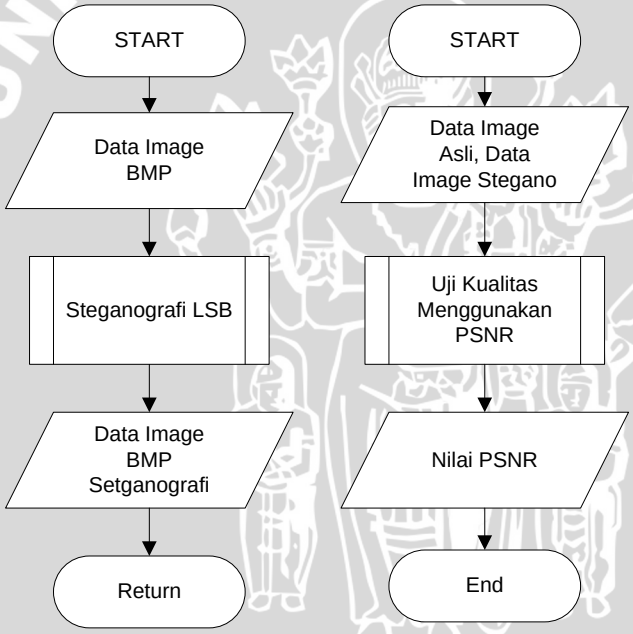


Gambar 3.1 Langkah – langkah Penelitian

3.1 Deskripsi Sistem

Tujuan pembuatan sistem ini yaitu untuk menyembunyikan pesan rahasaia ke dalam *file* gambar ber*format* BMP, pada proses steganografi pengguna memasukkan *input* data berupa citra ber*format* BMP, karena sistem hanya dibatasi untuk memproses citra tersebut, dan di dalam proses steganografi terdapat proses *encoding* (sisipan) dan *decoding* (ekstraksi). Dan pada tahap akhir gambar asli dan gambar yang sudah disisipkan pesan (berbeda-beda prosentase karakter yang disisipkan) di uji kualitas untuk mengetahui perbandingan nilai PSNR-nya.

Flowchart implementasi dari proses keseluruhan dapat digambarkan seperti pada Gambar 3.2.



Gambar 3.2 *Flowchart* sistem secara keseluruhan

3.1.1 Proses Steganografi LSB

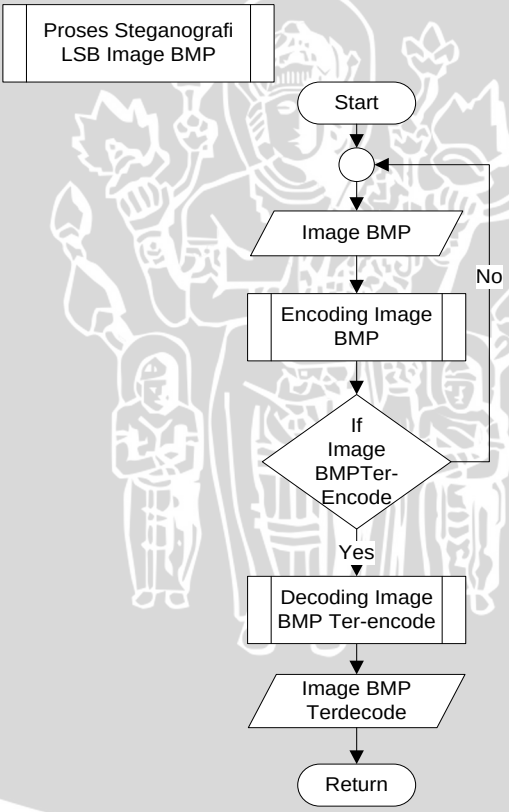
Proses selanjutnya sistem melakukan proses steganografi, awalnya pengguna memasukkan *input* data berupa citra ber*format*

BMP. Selanjutnya dilakukan proses steganografi menggunakanan metode LSB *(least significant bit).*

Setiap *pixel* gambar nilainya dikonversi ke bilangan *biner*, selanjutnya disisipi pesan yang sudah dikonversi ke bilangan desimal. Dimana urutan prosesnya dapat diuraikan sebagai berikut :

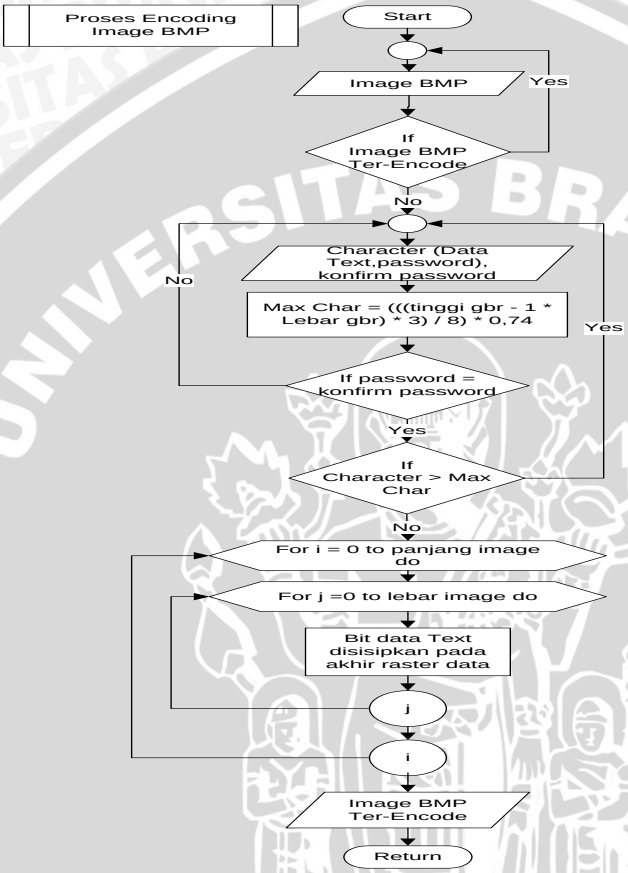
1. Masukkan *bit-bit* dan gambar asli
2. Dilakukan proses *encoding* (sisipan), dimana 1 *pixel* gambar memuat 1 *bit*
3. Proses *output* yang dihasilkan yaitu gambar *stego* (gambar yang berisi pesan di dalamnya)

Implementasi dari proses steganografi LSB *(least significant bit)* dapat digambarkan seperti pada Gambar 3.3.



Gambar 3.3 *Flowchart* sistem steganografi LSB *image* BMP

Flowchart implementasi dari proses *encoding* dapat digambarkan seperti pada Gambar 3.4.

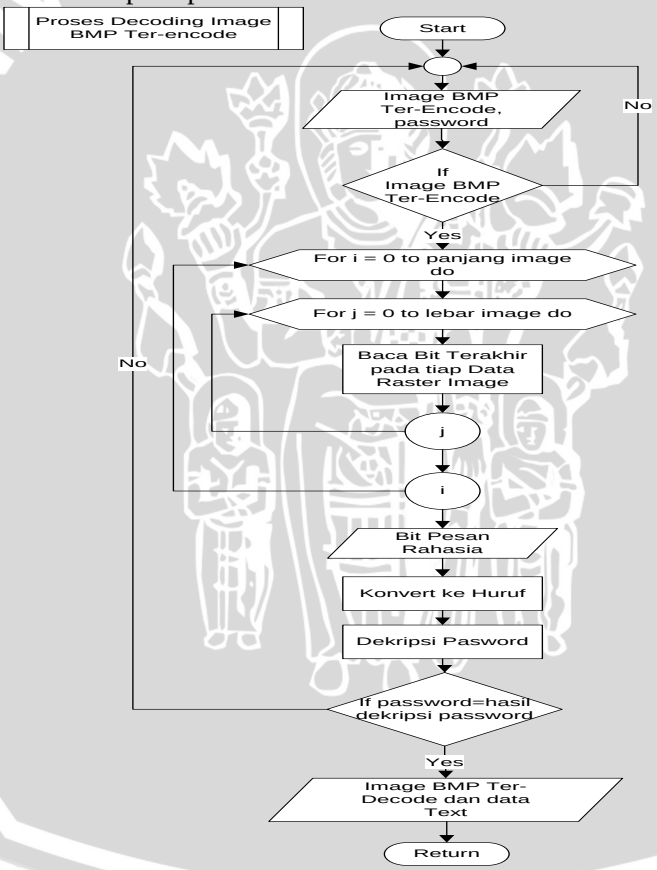


Gambar 3.4 *Flowchart* sistem *encoding image* BMP

Untuk proses *encoding* (sisipan) pengguna memasukkan *input* data berupa citra yang ber*format* BMP**.** karena sistem hanya dibatasi untuk memproses citra tersebut selanjutnya dilakukan cek pesan dan maksimal karakter yang dapat disisipkan terhadap citra *input*an, jika tidak ada pesan maka dilakukan penyisipan pesan tehadapan citra tersebut, sebaliknya jika sudah terdapat pesan di dalam citra tersebut maka sistem kembali ke awal untuk melakukan *input* citra yang belum berisi pesan, *output* yang dihasilkan adalah citra yang berisi

pesan (citra *stego*). Dan untuk menghitung banyaknya maksimal karakter ialah lebar dari gambar dikali tinggi dari gambar dikurangi 1 (karena baris pertama digunakan untuk menyimpan informasi bahwa gambar berisi pesan atau tidak, kemudian dikali 3, karena tiap pixel gambar terdiri dari 3 warna dasar yaitu R, G dan B. Kemudian hasilnya akan dibagi 8 karena tiap karakter akan disisipkan pada 8 baris bit. Dan terkahir dari uji coba *try and error* hasil diatas akan dikalikan dengan 74%, didapatkannya nilai 74% karena di system ini menggunakan password yang dialokasikan 26% ketika di enkripsi menggunakan RC4.

Flowchart implementasi dari proses *decoding* dapat digambarkan seperti pada Gambar 3.5.



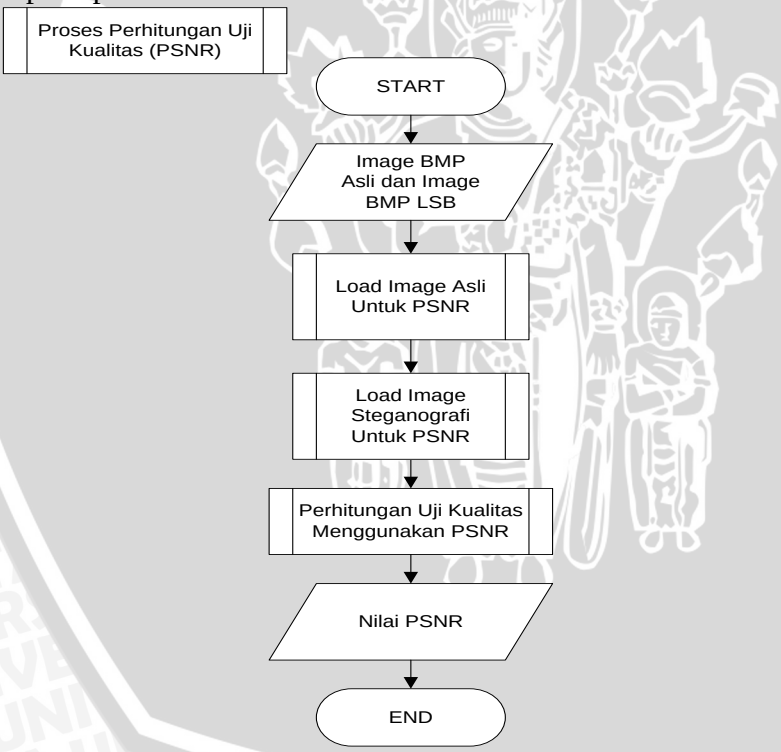
Gambar 3.5 *Flowchart* sistem *decoding image* BMP

Untuk proses *decoding* (ekstraksi) pengguna memasukkan *input* data berupa citra yang dihasilkan dari proses *encoding* (sisipan), karena sistem hanya dibatasi untuk memproses citra tersebut, *output* yang dihasilkan adalah citra dan pesan yang telah disisipkan.

3.1.2 Proses Perhitungan Uji Kualitas (PSNR)

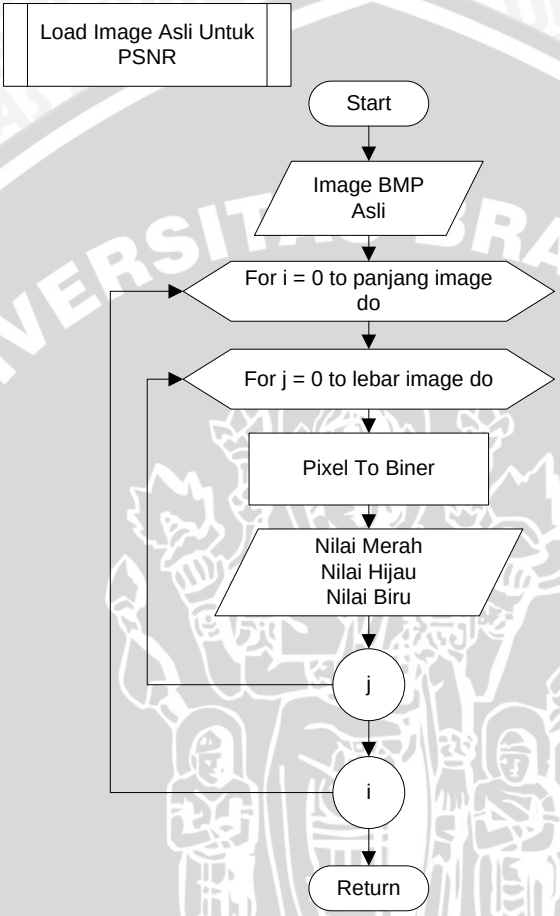
Proses selanjutnya sistem melakukan proses perhitungan uji kualitas menggunakan metode PSNR, pengguna memasukkan *input* berupa *image* BMP asli dan *image* BM yang sudah disisipi pesan dengan prosentase yang berbeda untuk kemudian dibandingkan nilai *pixel*nya dan didapatkan nilai PSNR-nya.

Dimana Implementasi dari proses perhitungan uji kualitas menggunakan metode PSNR dapat digambarkan seperti pada Gambar 3.6.



Gambar 3.6 *Flowchart* sistem perhitungan uji kualitas (PSNR)

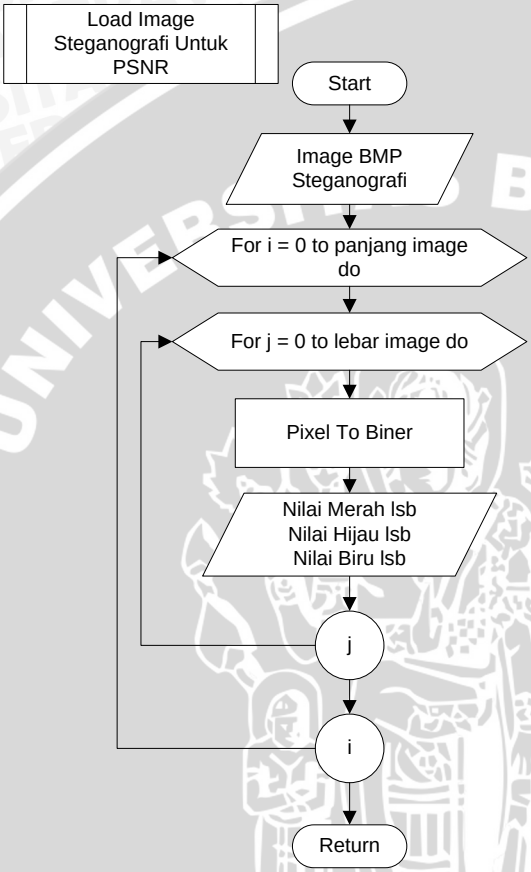
Flowchart implementasi dari proses *load image* asli untuk PSNR dapat digambarkan seperti pada Gambar 3.4.



Gambar 3.7 *Flowchart* sistem *load image* asli untuk PSNR

Untuk proses *load image* asli untuk PSNR dimasukkan *input* data berupa *image* yang ber*format* BMP dan masih asli belum disisipi pesan, karena berfungsi sebagai pembanding nilai antara *pixel-pixel*nya dengan *image* BMP yang sudah disisipi pesan dengan prosentase yang berbeda–beda pada proses perhitungan uji kualitas PSNR.

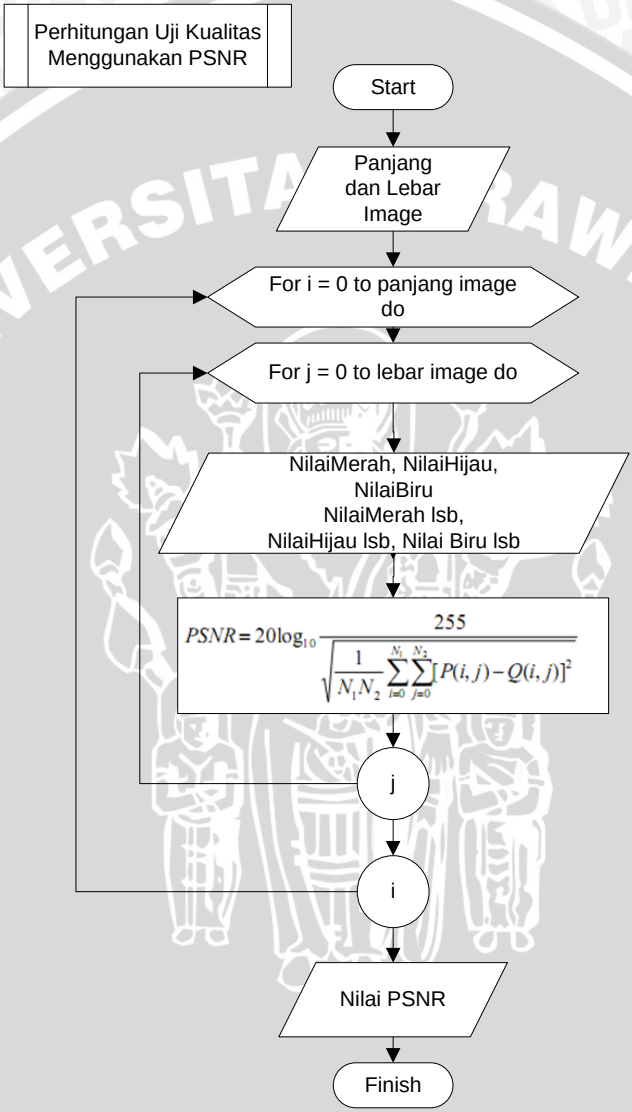
Flowchart implementasi dari proses *load image* steganografi untuk PSNR dapat digambarkan seperti pada Gambar 3.4.



Gambar 3.8 *Flowchart* sistem *load image* steganografi untuk PSNR

Untuk proses *load image* steganografi untuk PSNR dimasukkan *input* data berupa *image* BMP yang sudah disisipi pesan dengan prosentase yang berbeda–beda, karena berfungsi untuk membandingkan nilai antara *pixel-pixel*nya dengan *image* BMP asli yang belum disisipi pesan pada proses perhitungan uji kualitas PSNR.

Flowchart implementasi dari proses perhitungan uji kualitas menggunakan PSNR dapat digambarkan seperti pada Gambar 3.4.



Gambar 3.9 *Flowchart* sistem perhitungan uji kualitas menggunakan PSNR

Dan untuk proses perhitungan uji kualitas menggunakan PSNR, *input*an data berupa *image* yang ber*format* BMP yang belum disisipi pesan dan data *image* BMP yang sudah disisipi pesan dengan prosentase yang berbeda–beda dibandingkan nilai antara *pixel-pixel*nya dengan metode uji kualitas PSNR untuk diketahui nilai PSNR yang menandakan semakin mirip atau tidaknya *image* yang sudah disisipi pesan dengan *image* asli.

3.2 Desain Sistem

Pada bab ini dijelaskan mengenai desain aplikasi sistem untuk implementasi metode *output*. Desain aplikasi ini meliputi desain data, algoritma yang digunakan dalam sistem yang digambarkan dengan *flowchart* dan desain proses. Desain data berisikan penjelasan data yang diperlukan untuk dapat menerapkan proses steganografi dan uji kualitas gambar menggunakan metode PSNR. Desain data meliputi data masukan, data selama proses dan data keluaran. Desain proses steganografi antara lain menjelaskan tentang proses penyisipan pesan, proses *konversi* bilangan *biner*, proses *konversi* pesan rahasia, proses penggantian *bit*, proses penghitungan karakter maksimal dan proses penghitungan karakter sisa, dan desain uji kualitas gambar meliputi antara gambar yang sudah disisipi pesan terhadap gambar aslinya.

3.2.1 Desain Data

Data yang digunakan untuk implementasi perangkat lunak ini dibagi menjadi tiga bagian utama, yaitu data masukan, data yang digunakan selama proses steganografi dan uji kualitas, dan data keluaran.

3.2.1.1 Data Masukan

Data masukan pertama dari pengguna adalah arsip citra yang dipilih oleh pengguna. Pada sistem ini citra yang dimasukkan berupa arsip citra *bitmap* (bmp) dengan ukuran *file* dan ukuran panjang * lebar citra yang berbeda-beda. Daftar data masukan (daftar citra yang digunakan) pada sistem ini dapat dilihat pada tabel 3.1.

Tabel 3.1 Data citra *input*an system

No	Nama *File*	Ukuran Gambar (P*L) *Pixel*	Ukuran *File*
1	Alam.Bmp	732 * 490	1,02 MB
2	Burung.Bmp	565 * 750	1,21 MB
3	Elang.Bmp	800 * 600	1,37 MB
4	Kucing.Bmp	662 * 811	1,53 MB
5	Monyet.Bmp	727 * 488	1,01 MB

3.2.1.2 Data Selama Proses

Pada tahap proses steganografi, ketika citra dibaca berbentuk nilai *pixel* matriks (baris dan kolom) dilakukan *encode* pesan terhadap citra tersebut, dimana satu *pixel* citra memuat satu *bit* untuk disisipkan, penggantian *bit* menggunakan metode least signifikant *bit* (LSB), selanjutnya dilakukan *decode* pesan terhadap citra tersebut yaitu, *bit* rendah setiap *pixel* dikumpulkan hingga terbentuk *bit* stream, arah bacanya adalah kiri ke kanan dan atas ke bawah. Setiap tujuh *bit* stream mempresentasikan *output* sebuah karakter.

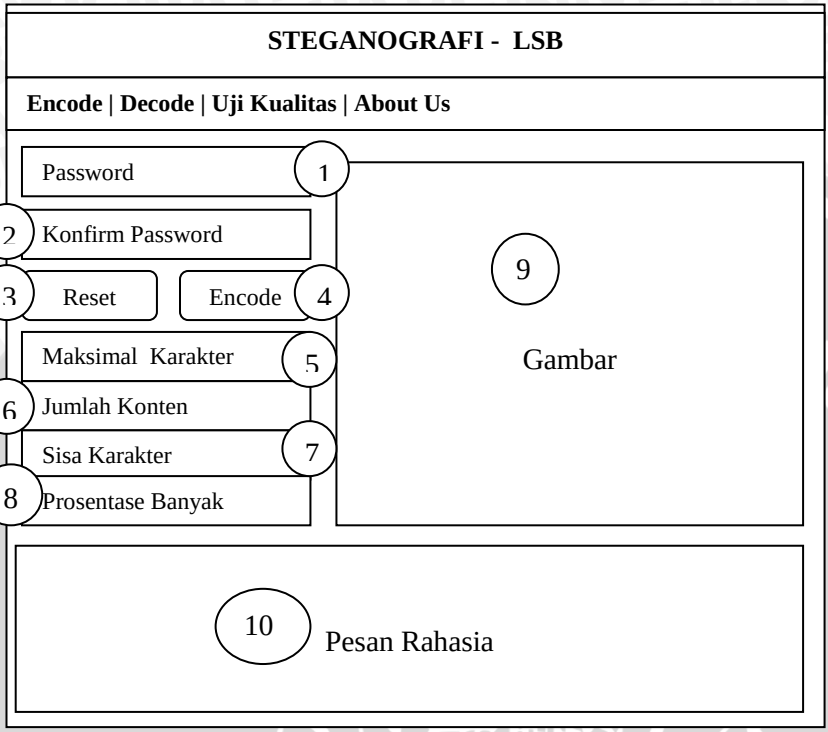
Pada tahap proses uji kualitas, citra *image* asli dibandingkan dengan citra *image* yang telah disisipi pesan yang prosentase jumlah pesannya berbeda-beda. Kemudian dihitung perbandingan tingkat uji kualitas gambar keduanya menggunakan rumus PSNR.

3.2.1.3 Data Keluaran

Data keluaran yang dihasilkan sistem pada proses steganografi adalah citra yang disipi pesan, pesan rahasia yang di*decode*, karakter maksimal, karakter yang digunakan dan karakter sisa pada citra tersebut. Dan data keluaran yang dihasilkan sistem pada proses uji kualitas adalah nilai PSNR dengan satuan *decibel* (dB).

3.3 Rancangan Antar Muka Sistem

Antar muka tulis pesan yang dibangun ditunjukkan pada gambar 3.10



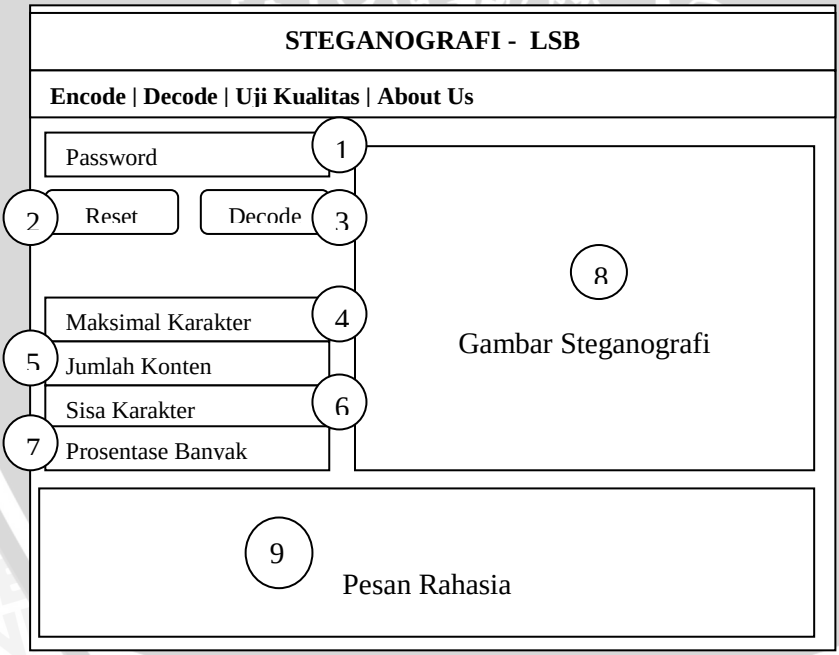
Gambar 3.10 Rancangan antar muka tulis pesan

Berdasarkan karakteristik himpunan dengan model biagram keterangan bagian-bagian dalam rancangan antar muka menu kompresi DCT adalah sebagai berikut:

1. *Textbox password* berfungsi untuk meng*input*kan *password* yang digunakan kedalam *image* BMP yang dipilih.
2. *Textbox* konfirm *password* berfungsi untuk meng*input*kan kembali *password* yang digunakan kedalam *image* BMP yang dipilih.
3. *Button* reset berfungsi untuk membatalkan *input*an *password*, pesan rahasia dan pemilihan *image* yang telah di*input*kan beserta semua informasi yang ditampilkan.
4. *Button encode* berfungsi untuk memproses pesan yang telah ditulis untuk disisipkan kedalam gambar BMP.
5. *Textbox* maksimal karakter berisi keterangan berapa jumlah maksimal karakter yang bisa disisipkan kedalam gambar.

6. *Textbox* karakter maksimal berisi keterangan berapa jumlah maksimal karakter yang bisa tersisipkan kedalam gambar BMP yang dipilih.
7. *Textbox* sisa karakter berisi keterangan berapa jumlah sisa karakter yang bisa disisipkan kedalam gambar.
8. *Textbox* prosentase berfungsi untuk menampilkan prosentase dari jumlah pesan rahasia yang disisipkan kedalam gambar BMP yang telah dipilih.
9. *Frame* gambar berfungsi untuk memanggil gambar ber*format* BMP dan berfungsi untuk menampilan gambar yang telah dipilih untuk kemudian disisipkan pesan.
10. *Memo* pesan rahasia berisi keterangan karakter apa yang telah ditulis dan untuk kemudian disisipkan kedalam gambar.

Antar muka baca pesan yang dibangun ditunjukkan pada gambar 3.11

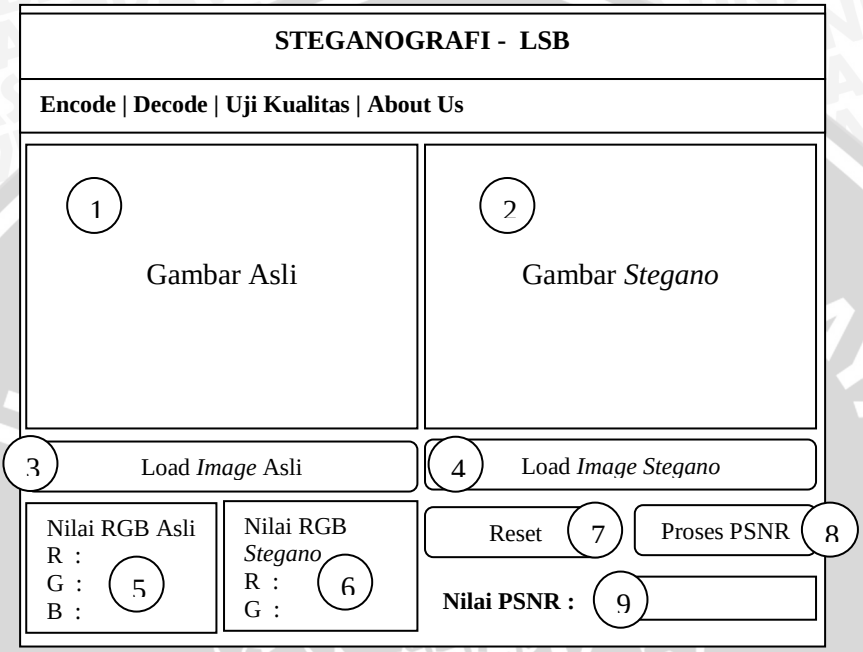


Gambar 3.11 Rancangan antar muka tulis pesan

Berdasarkan karakteristik himpunan dengan model biagram keterangan bagian-bagian dalam rancangan antar muka utama adalah sebagai berikut:

1. *Textbox password* berfungsi untuk menyamakan *password* yang di*input*kan dengan *password* yang ada didalam *image* BMP yang dipilih.
2. *Button* reset berfungsi untuk membatalkan *input*an *password* dan pemilihan *image* yang telah di*input*kan beserta semua informasi yang ditampilkan.
3. *Button decode* berfungsi untuk melakukan proses baca pesan yang ada didalam gambar BMP yang dipilih.
4. *Textbox* maksimal karakter berisi keterangan berapa jumlah maksimal karakter yang bisa tersisipkan kedalam gambar JPG yang dipilih.
5. *Textbox* karakter dipakai berisi keterangan berapa jumlah karakter yang digunakan pada pesan rahasia didalam gambar BMP yang dipilih dan berisi pesan.
6. *Textbox* sisa karakter berisi keterangan berapa jumlah sisa karakter yang tersisa pada gambar BMP yang berisi pesan rahasia.
7. *Textbox* prosentase berfungsi untuk menampilkan prosentase dari jumlah pesan rahasia yang telah disisipkan kedalam gambar BMP yang telah dipilih.
8. *Frame* gambar steganografi berisi tampilan dari gambar ber*format* BMP yang telah dipilih untuk kemudian dibaca pesan.
9. *Memo* pesan rahasia berisi keterangan karakter apa yang telah disisipkan kedalam gambar yang telah dipilih.

Antar muka uji kualitas PSNR yang dibangun ditunjukkan pada gambar 3.12



Gambar 3.12 Rancangan antar muka uji kualitas PSNR

Berdasarkan karakteristik himpunan dengan model biagram keterangan bagian-bagian dalam rancangan antar muka utama adalah sebagai berikut:

1. *Frame* gambar asli berfungsi untuk menampilan gambar yang belum disisipi pesan untuk kemudian dibandingkan *pixel-pixel*nya dengan gambar *stegano* untuk didapatkan nilai PSNR.
2. *Frame* gambar steganografi berfungsi untuk menampilan gambar yang sudah disisipi pesan untuk kemudian dibandingkan *pixel-pixel*nya dengan gambar aslinya untuk didapatkan nilai PSNR
3. *Button load image* asli berfungsi untuk memanggil gambar ber*format* BMP yang belum disisipi pesan.

4. *Button load image stegano* berfungsi untuk memanggil gambar ber*format* BMP yang sudah disispi pesan.
5. *Textbox* nilai RGB asli berfungsi untuk menampilkan informasi dari jumlah nilai *pixel* gambar asli yang terdiri dari 3 bagian utama , yaitu nilai R untuk nila red, B untuk nilai blue dan G untuk nilai green.
6. *Textbox* nilai RGB stegano berfungsi untuk menampilkan informasi dari jumlah nilai *pixel* gambar *stegano* yang terdiri dari 3 bagian utama , yaitu nilai R untuk nila red, B untuk nilai blue dan G untuk nilai green.
7. *Button* reset berfungsi untuk membatalkan pemilihan semua *image* yang telah diipilih beserta semua informasi yang ditampilkan dan diproses.
8. *Button* proses PSNR berfungsi untuk menghitung nilai uji kualitas dari gambar asli dan *stegano* yang telah dipilih.
9. *Textbox* Nilai PSNR berfungsi untuk menampilkan nilai dari hasil uji kualitas PSNR dari gambar asli dengan gambar *stegano* yang telah dipilih.

3.4 Rancangan Uji Coba dan Evaluasi Hasil

File hasil pengujian Steganografi LSB di gunakan untuk evaluasi kinerja sistem. Tujuan dari hasil uji coba ini adalah untuk mengetahui tingkat keberhasilan dari penerapan metode steganografi LSB yang mengacu terhadap kualitas gambar menggunakan uji kualitas PSNR.

3.4.1 Rancangan Uji Coba

Pengujian terhadap sistem dilakukan dengan cara meng*input*kan *file* gambar ber eksentensi BMP dengan berbagai ukuran *file* dan jumlah pesan yang berbeda-beda yang disisipkan untuk mengetahui pengaruh steganografi LSB terhadap jumlah pesan yang disisipkan kedalam gambar BMP menggunakan uji kualitas PSNR.

3.4.2 Analisa dan Evaluasi Hasil

Dari hasil steganografi di evaluasi dengan memperhatikan pengaruhnya terhadap kualitas gambar menggunakan uji kualitas PSNR dengan cara melihat perbedaan nilai PSNR pada tiap-tiap jumlah pesan yang disipkan . Untuk *output image* yang di dapatkan

pada steganografi kualitas gambar di evaluasi dengan membandingkan dengan *file image* BMP asli dan *image* BMP yang sudah disisipkan pesan dengan jumlah berbeda-beda.

Rancangan tabel untuk hasil uji coba yang di lakukan ditampikan pada tabel 3.2.

Tabel 3.2 Data hasil uji coba

No	Banyak Karakter *Input*	Prosentase Karakter *Input*	Ukuran *File*	Nilai PSNR

3.5 Contoh Perhitungan Manual

3.5.1 Contoh Perhitungan Steganografi LSB

3.5.1.1 Tulis Pesan

Pada citra *grayscale* 4x4 piksel seperti contoh perhitungan kompresi kemudian disisipkan pesan yang berbunyi "UB".

Pesan → Kode ASCII

85 66

Kode ASCII → kode-kode biner

1010101 1000010

Matrik derajat keabuan citra sbb:

35	17	19	16
7	3	6	4
7	1	4	1
9	3	3	4

Derajat keabuan citra → biner

00100011	00010001	00010011	00010000
00000111	00000011	00000110	00000100
00000111	00000001	00000100	00000001
00001001	00000011	00000011	00000100

LSB diganti dengan *text* pesan

00100011	00010001	00010010	00010001
00100010	00010000	00010010	00010000
00100011	00010001	00010010	00010000
00100010	00010001	00010010	00010000

Diubah menjadi desimal (derajat keabuan citra baru)

35	17	18	17
34	16	18	16
35	17	18	16
34	17	18	16

3.5.1.2 Baca Pesan

Dari nilai citra *grayscale* 4x4 yang telah disisipi pesan rahasia diatas, terlebih dahulu di konversi ke biner sehingga di dapat hasil seperti berikut:

00100011	00010001	00010010	00010001
00100010	00010000	00010010	00010000
00100011	00010001	00010010	00010000
00100010	00010001	00010010	00010000

Dari setiap nilai *byte* per piksel diambil nilai yang paling akhir sehingga di dapat nilai biner seperti berikut :

1010101 1000010

Kemudian di konversi ke bilangan desimal

1010101 1000010 → 85 66

Kemudian diterjemahkan kedalam bentuk huruf sesuai ASCII

85 66 → U B

Maka didapat hasil pesan rahasia “UB”.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini dibahas implementasi sistem dan uji coba sistem serta evaluasi hasil. Implementasi sistem merupakan implementasi rancangan yang sudah dilakukan pada bab 3. Sedangkan pembahasan adalah penjelasan dari masing-masing implementasi tersebut.

Uji coba dilakukan pada *image* BMP dengan menerapkan algoritma yang didasarkan pada metode LSB (*Least Significant Bit*). Hasil dari uji coba ini digunakan untuk mengevaluasi tingkat perbedaan uji kualitas antara *image* asli dan *image* yang sudah disisipkan pesan rahasia yang sudah dilakukan oleh sistem yang telah dibuat.

Perangkat lunak yang digunakan dalam pengembangan sistem steganografi ini adalah :

1. Sistem operasi yang digunakan adalah Microsoft Windows XP Professional.
2. *Software* yang digunakan untuk mengembangkan sistem adalah Borland Delphi 7.

Perangkat keras yang digunakan untuk mengembangkan sistem pengsteganografian citra BMP menggunakan metode LSB (*Least Significant Bit*) adalah :

1. Prosesor Intel Dual Core 2.16 Ghz
2. Memori 3 GB
3. Harddisk dengan kapasitas 250 GB

4.1 Implementasi Program

Berdasarkan analisa dan rancangan sistem pada Bab 3, maka pada sub bab ini dijelaskan implementasi proses-proses tersebut.

4.2 Implementasi Tulis Pesan (*Encode*)

Berdasarkan pada rancangan antar muka pada bab 3 maka dihasilkan antar muka yang merupakan tampilan proses awal dari *system* steganogarafi, yaitu antar muka proses tulis pesan (*Encode*) ditunjukkan pada gambar 4.1.

Gambar 4.1 Antar muka proses tulis pesan (*Encode*)

Pada gambar 4.1 terdapat tombol "*Encode*" yang berfungsi untuk memproses penyisipin karakter pada gambar berekstensi *.Bmp yang dipilih. Dimana Proses penyisipin karakter menggunakan prosedur *encode* ditunjukkan pada *source code* 4.1.

```
procedure TfrmMain.Encode;
var
  i, j, k, x, LSBCount: Integer;
  Red, Green, Blue: Integer;
  strMark, bitMark, bitContent, tempContent: String;
  bitRed, bitGreen, bitBlue: String;
  bitBaris, bitKolom: String;
begin
  try
    Bmp.Assign(imgEncode.Picture.Bitmap);
    Bmp.PixelFormat := pf24bit;
    if CarrierCheck(Bmp) then begin
      MessageDlg('File Gambar sudah disisipi data. Gunakan file
yang lain',mtWarning,[mbOK],0);
      Exit;
    end;

    LSBCount := (Bmp.Width * Bmp.Height) * 3
    LSBCount := LSBCount - (Bmp.Width * 3);
    tempContent := edEncodePass1.Text + '#' +
moEncode.Lines.Text;
```

```
    tempContent := doEncrypt(edEncodePass1.Text, tempContent);
    for i := 1 to Length(tempContent) do begin
      bitContent := bitContent + getByte(Ord(tempContent[i]));

    if Length(bitContent) > LSBCount then begin
      MessageDlg('Content yang akan diisikan terlalu besar.'+
                 #13#10+'Kurangi content atau pilih file
carrier lain yang lebih besar.',mtWarning,[mbOK],0);
      Exit;
    end;

    Self.Cursor := crHourGlass;
    strMark := '@d@'; // string penanda
    bitMark := getByte(Ord(strMark[1])) +
               getByte(Ord(strMark[2])) +
               getByte(Ord(strMark[3]));
```

***Source Code* 4.1** Proses tulis pesan (*Encode*)

proses selanjutnya ialah perhitungan maksimal karakter yang dapat disisipkan terhadap citra *input*an. Untuk menghitung banyaknya maksimal karakter ialah lebar dari gambar dikali tinggi dari gambar dikurangi 1 (karena baris pertama digunakan untuk menyimpan informasi bahwa gambar berisi pesan atau tidak, kemudian dikali 3, karena tiap pixel gambar terdiri dari 3 warna dasar yaitu R, G dan B. Kemudian hasilnya akan dibagi 8 karena tiap karakter akan disisipkan pada 8 baris bit. Dan terkahir dari uji coba *try and error* hasil diatas akan dikalikan dengan 74%, didapatkannya nilai 74% karena di system ini menggunakan password yang dialokasikan 26% ketika di enkripsi menggunakan RC4. Proses perhitungan maksimal karakter ditunjukkan pada *source code* 4.2.

```
procedure TfrmMain.panjang;
var
pjg : integer;
begin
  try
      Bmp.Assign(imgEncode.Picture.Bitmap);
      Bmp.PixelFormat := pf24bit;
      pjg := (Bmp.Width * (Bmp.Height-1)) * 3;
      Edit1.Text := FormatFloat(  '#',(pjg/8));
      Edit1.Text := FormatFloat(  '#',strtofloat(Edit1.Text)*
0.74);

  except
    on E: Exception do begin
      MessageDlg(E.Message,mtError,[mbOK],0);
    end;
```

```
  end;
end;
```

***Source Code* 4.2** Proses tulis pesan (*encode*) untuk perhitungan maksimal karakter

Proses selanjutnya ialah penyisipan *bit content*, dimana langkah awal ialah *bit content* mulai disisipkan pada baris ke dua dari struktur *pixel*. Kemudian langkah kedua mengambil nilai setiap warna pada data *pixel* pada setiap baris. Kemudian langkah ketiga mengkonversi nilai setiap warna ke *format binary*. Kemudian langkah keempat menghapus LSB pada setiap *byte* warna untuk nantinya diganti dengan *bit content*. Kemudian mengganti *bit* LSB yang sebelumnya dihapus dengan menambahkan *bit content* berturut-turut pada akhir *bit* setiap warna dan langkah terkahir ialah mengkonversi *bit binary* setiap warna ke bentuk *integer* kembali. Proses penyisipan *bit content* ditunjukkan pada *source code* 4.3.

```
k := 1;
    for i := 1 to Bmp.Height - 1 do begin
      for j := 0 to Bmp.Width - 1 do begin
        Red := GetRValue(GetPixel(Bmp.Canvas.Handle,j,i));
        Green := GetGValue(GetPixel(Bmp.Canvas.Handle,j,i));
        Blue := GetBValue(GetPixel(Bmp.Canvas.Handle,j,i));

        bitRed := getByte(Red);
        bitGreen := getByte(Green);
        bitBlue := getByte(Blue);
        Delete(bitRed,8,1);
        Delete(bitGreen,8,1);
        Delete(bitBlue,8,1);
        bitBlue := bitBlue + bitContent[k];
        bitGreen := bitGreen + bitContent[k+1];
        bitRed := bitRed + bitContent[k+2];
        k := k + 3;

        Red := getDecimal(bitRed);
        Green := getDecimal(bitGreen);
        Blue := getDecimal(bitBlue);
        Bmp.Canvas.Pixels[j,i] := RGB(Red,Green,Blue);
        if k > length(bitContent) then Break;
      end;
     if k > length(bitContent) then Break;
    end;
```

***Source Code* 4.3** Proses tulis pesan (*encode*) untuk penyisipan *bit content*

Proses selanjutnya ialah untuk menyimpan informasi batas baris yang digunakan, dimana langkah awal ialah mendapatkan batas indeks baris yang didapatkan dari nilai *variable* i dari hasi prosess penyisipan *content* di atas. Kemudian langkah kedua penyisipan informasi baris dimulai dari *pixel* ke 8 sampai 15, karena 8 *pixel* (*pixel* 0 sampai 7) pertama telah digunakan untuk menyimpan *bit* penanda, maka untuk *bit* baris dimulai dari indeks *pixel* ke 8.

Langkah ketiga mengambil nilai setiap warna pada data *pixel* pada baris pertama serta mengkonversi nilai setiap warna ke *format binary*. Kemudian langkah keempat ialah menghapus LSB pada setiap *byte* warna untuk nantinya diganti dengan *bit* baris dan mengganti *bit* LSB yang sebelumnya dihapus dengan menambahkan *bit* baris berturut-turut pada akhir *bit* warna.

Dan langkah terkahir ialah menentukan indeks awal *bit* baris pada sesi looping selanjutnya serta mengkonversi *bit binary* setiap warna ke bentuk *integer* kembali. Proses untuk menyimpan informasi batas baris yang digunakan ditunjukkan pada *source code* 4.4.

```
k := 1;
  bitBaris := getWord(i);

  for x := 8 to 15 do begin
    Red := GetRValue(GetPixel(Bmp.Canvas.Handle,x,0));
    Green := GetGValue(GetPixel(Bmp.Canvas.Handle,x,0));
    Blue := GetBValue(GetPixel(Bmp.Canvas.Handle,x,0));


    bitRed := getByte(Red);
    bitGreen := getByte(Green);

    Delete(bitRed,8,1);
    Delete(bitGreen,8,1);

    bitGreen := bitGreen + bitBaris[k];
    bitRed := bitRed + bitBaris[k+1];

    k := k + 2;

    Red := getDecimal(bitRed);
    Green := getDecimal(bitGreen);
    Bmp.Canvas.Pixels[x,0] := RGB(Red,Green,Blue);
  end;
```

***Source Code* 4.4** Proses tulis pesan (*Encode*) untuk menyimpan informasi batas baris yang digunakan

Proses selanjutnya ialah untuk menyimpan informasi batas kolom yang digunakan, dimana langkah awal ialah mendapatkan batas indeks kolom yang didapatkan dari nilai *variable* j dari hasi prosess penyisipan *content* di atas.Kemudian langkah kedua penyisipan penyisipan informasi kolom dimulai dari *pixel* ke 16 sampai 2, Karena 8 *pixel* (*pixel* 8 sampai 15) kedua telah digunakan untuk menyimpan *bit* baris, maka untuk *bit* baris dimulai dari indeks *pixel* ke 16.

Langkah ketiga mengambil nilai setiap warna pada data *pixel* pada baris pertama serta mengkonversi nilai setiap warna ke *format binary*. Kemudian langkah keempat ialah menghapus LSB pada setiap *byte* warna untuk nantinya diganti dengan *bit* kolom dan mengganti *bit* LSB yang sebelumnya dihapus dengan menambahkan *bit* baris berturut-turut pada akhir *bit* warna.

Dan langkah terkahir ialah menentukan indeks awal *bit* kolom pada sesi looping selanjutnya serta mengkonversi *bit binary* setiap warna ke bentuk *integer* kembali. Proses untuk menyimpan informasi batas kolom yang digunakan ditunjukkan pada *source code* 4.5.

```
  k := 1;
    bitKolom := getWord(j);

    for x := 16 to 23 do begin
      Red := GetRValue(GetPixel(Bmp.Canvas.Handle,i,0));
      Green := GetGValue(GetPixel(Bmp.Canvas.Handle,i,0));
      Blue := GetBValue(GetPixel(Bmp.Canvas.Handle,i,0));

      bitRed := getByte(Red);
      bitGreen := getByte(Green);

      Delete(bitRed,8,1);
      Delete(bitGreen,8,1);

      bitGreen := bitGreen + bitKolom[k];
      bitRed := bitRed + bitKolom[k+1];


      k := k + 2;

      Red := getDecimal(bitRed);
      Green := getDecimal(bitGreen);
      Bmp.Canvas.Pixels[x,0] := RGB(Red,Green,Blue);
    end;

    Self.Cursor := crDefault;
```

```
    if MessageDlg('Encoding telah selesai. Silahkan pilih Yes
jika akan menyimpan hasil
encoding.',mtConfirmation,[mbYes,mbNo],0) = ID_YES then begin
      if SavePictureDialog1.Execute then begin
        SavePictureDialog1.Default text := '*.bmp';
        Bmp.SaveTo File(SavePictureDialog1.File Name+'.bmp');
      end;
    end;
  except
    on E: Exception do begin
      MessageDlg(E.Message,mtError,[mbOK],0);
    end;
  end;
end;
```

***Source Code* 4.5** Proses tulis pesan (*Encode*) untuk menyimpan informasi batas kolom yang digunakan

4.3 Implementasi Baca Pesan (*Decode*)

Pada rancangan antar muka selanjutnya dihasilkan antar muka yang merupakan tampilan proses selanjutnya dari *system* steganografi proses *encode*, yaitu antar muka proses baca pesan (*Decode*) ditunjukkan pada gambar 4.2.

Gambar 4.2 Antar muka proses baca pesan (*Decode*)

Pada gambar 4.2 terdapat tombol "*Decode*" yang berfungsi untuk memproses pembacaan karakter tersembunyi yang disispkan pada gambar berekstensi *.Bmp yang dipilih. Dimana Proses pembacaan karakter tersembunyi menggunakan prosedur *decode* ditunjukkan pada *source code* 4.6.

```
 procedure TfrmMain.Decode;
var i, j, k: Integer;
    Red, Green, Blue: Integer;
    bitContent, tempContent: String;
    bitRed, bitGreen, bitBlue: String;
    bitBaris, bitKolom: String;
    Baris, Kolom: Integer;
begin
  try
    Bmp.Assign(imgDecode.Picture.Bitmap);
    Bmp.PixelFormat := pf24bit;
    if not CarrierCheck(Bmp) then begin
      MessageDlg('File gambar tidak mengandung
pesan.',mtWarning,[mbOK],0);
      Exit;
    end;

    bitBaris := '';
    for k := 8 to 15 do begin
      Red := GetRValue(GetPixel(Bmp.Canvas.Handle,k,0));
      Green := GetGValue(GetPixel(Bmp.Canvas.Handle,k,0));

      bitBaris := bitBaris + getByte(Green)[8] +
getByte(Red)[8];
    end;
    Baris := getDeimal(bitBaris);

    bitKolom := '';
    for k := 16 to 23 do begin
      Red := GetRValue(GetPixel(Bmp.Canvas.Handle,k,0));
      Green := GetGValue(GetPixel(Bmp.Canvas.Handle,k,0));
      bitKolom := bitKolom + getByte(Green)[8] +
getByte(Red)[8];
    end;
  Kolom := getDecimal(bitKolom);
```

***Source Code* 4.6** Proses baca pesan (*Decode*)

Proses selanjutnya ialah untuk pengambilan *bit content*, dimana langkah awal ialah *bit content* mulai diambil pada baris ke dua dari struktur *pixel* serta mengambil nilai setiap warna data *pixel* pada setiap baris. Kemudian langkah terkahir ialah mengkonversi nilai setiap warna ke *format binary* dan mengambil LSB pada setiap

byte warna dan menampungnya di *variable* temp*Content*. Proses untuk pengambilan *bit content* ditunjukkan pada *source code* 4.7.

```
    tempContent := '';
    for i := 1 to Bmp.Height - 1 do begin
      for j := 0 to Bmp.Width - 1 do begin
        Red := GetRValue(GetPixel(Bmp.Canvas.Handle,j,i));
        Green := GetGValue(GetPixel(Bmp.Canvas.Handle,j,i));
        Blue := GetBValue(GetPixel(Bmp.Canvas.Handle,j,i));

        bitRed := getByte(Red);
        bitGreen := getByte(Green);
        bitBlue := getByte(Blue);

        tempContent := tempContent + Copy(bitBlue,8,1) +
Copy(bitGreen,8,1) + Copy(bitRed,8,1);


        if (Baris = i) and (Kolom = j) then Break;
      end;
      if (Baris = i) and (Kolom = j) then Break;
    end;
```

***Source Code* 4.7** Proses baca pesan (*Decode*) untuk pengambilan *bit content*

Proses selanjutnya ialah untuk memilah setiap *byte content* dan menerjemahkannya ke dalam bentuk karakter yang bisa dibaca, dimana langkah awalnya ialah *content* harus didecript terlebih dahulu karena karakter dienkripsi pada saat encoding. Dan langkah selanjutnya ialah mengeliminasi karakter *password* + # dari *content*, hingga yang tersisa karakter *content* untuk ditampilkan di memo. Proses untuk memilah setiap *byte content* dan menerjemahkannya ke dalam bentuk karakter yang bisa dibaca ditunjukkan pada *source code* 4.8.

```
        bitContent := '';
    k := 1;
    for i := 1 to Round(Length(tempContent)/8) do begin
      bitContent := bitContent +
Chr(getDecimal(Copy(tempContent,k,8)));
      k := k + 8;
    end;

    bitContent := doDecrypt(edDecodePass1.Text,bitContent);
    if edDecodePass1.Text =
Copy(bitContent,1,Length(edDecodePass1.Text)) then
```

```
      moDecode.Lines.Text :=
StringReplace(bitContent,edDecodePass1.Text+'#','',[rfReplaceAl
l])
    else MessageDlg('Password yang anda masukkan
salah.',mtWarning,[mbOK],0);
  except
    on E: Exception do begin
      MessageDlg(E.Message,mtError,[mbOK],0);
    end;
  end;
end;
```

***Source Code* 4.8** Proses baca pesan (*Decode*) untuk memilah setiap *byte content* dan menerjemahkannya ke dalam bentuk karakter yang bisa dibaca

4.4 Implementasi Uji Kualitas Gambar PSNR

Dan pada rancangan antar muka selanjutnya dihasilkan antar muka yang merupakan tampilan proses uji kualitas gambar PSNR ditunjukkan pada gambar 4.3.

Gambar 4.3 Antar muka proses uji kualitas gambar PSNR

Pada gambar 4.3 terdapat nilai PSNR yang berfungsi untuk menampilkan nilai hasil uji kualitas dari gambar asli dan gambar

yang telah disisipi karakter rahasia. Proses penghitungan nilai uji kualitas PSNR ditunjukkan pada *source code* 4.9.

```
    rms;
    rms1;

    rms_r:=exp(0.5*ln((((StrToFloat(Edit7.Text)-
StrToFloat(Edit10.Text)) )/(bmp.Width * bmp.Height))));
    rms_g:=exp(0.5*ln((((StrToFloat(Edit8.Text)-
StrToFloat(Edit11.Text)) )/(bmp.Width * bmp.Height))));
    rms_b:=exp(0.5*ln((((StrToFloat(Edit9.Text)-
StrToFloat(Edit12.Text)) )/(bmp.Width * bmp.Height))));

    rmse:=(rms_r+rms_g+rms_b)/3;
    panel10.Caption:=FormatFloat(  '#,##0.00',rmse);
    psnr := (20*ln(255/sqr(rmse))/ln(10));
    Panel7.Caption := FormatFloat(  '#,##0.00',psnr)+'  db';
```

***Source Code* 4.9** Proses perhitungan PSNR

4.5 Uji Coba dan Evaluasi Hasil

4.5.1 Rancangan Evaluasi

Pengujian dilakukan terhadap 5 *file* gambar berekstensi *.Bmp yang mempunyai ukuran gambar dan ukuran *file* yang berbeda-beda. Masing-masing gambar disisipi karakter dengan prosentase yang berbeda sebesar 10%, 20% sampai dengan 100% dari jumlah maksimal karakter yang bisa disisipkan pada gambar. Sehingga tiap gambar menghasilkan 10 gambar yang memiliki jumlah karakter yang telah disisipkan berbeda-beda. Data *input* yang digunakan dalam uji coba ditunjukkan pada tabel 4.1.

Tabel 4.1 Data *input* untuk proses steganografi LSB

No	Nama *File*	Ukuran Gambar (P*L)	Ukuran *File*
1	Buah.Bmp	176 * 145	74,8 KB
2	Keren.Bmp	640 * 480	301 KB
3	Monyet.Bmp	727 * 488	1,01 MB
4	Planet.Bmp	962 * 869	2,39 MB
5	Wow.Bmp	1.920 * 1.200	6,59 MB

4.5.2 Hasil Uji Coba

Uji coba pertama dari penelitian ini adalah proses *encode* pada gambar buah.Bmp yang berukuran *file* 74,8 KB dan dapat menampung maksimal karakter untuk disisipkan sebesar 7.033 karakter. Hasil uji coba penyisipan karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar buah.Bmp ditunjukkan pada tabel 4.2.

Tabel 4.2 Data *encode* karakter pada gambar buah.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *input*	Ukuran *File*
1	703	10%	74,8 KB
2	1.406	20%	74,8 KB
3	2.110	30%	74,8 KB
4	2.813	40%	74,8 KB
5	3.516	50%	74,8 KB
6	4.220	60%	74,8 KB
7	4.923	70%	74,8 KB
8	5.626	80%	74,8 KB
9	6.330	90%	74,8 KB
10	7.033	100%	74,8 KB

Uji coba kedua dari penelitian ini adalah proses *encode* pada gambar keren.Bmp yang berukuran *file* 301 KB dan dapat menampung maksimal karakter untuk disisipkan sebesar 85.070 karakter. Hasil uji coba penyisipan karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar keren.Bmp ditunjukkan pada tabel 4.3.

Tabel 4.3 Data *encode* karakter pada gambar keren.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *input*	Ukuran *File*
1	8.511	10%	301 KB
2	17.018	20%	301 KB
3	25.525	30%	301 KB
4	34.032	40%	301 KB
5	42.539	50%	301 KB
6	51.046	60%	301 KB
7	59.553	70%	301 KB
8	68.060	80%	301 KB
9	76.567	90%	301 KB
10	85.070	100%	301 KB

Uji coba ketiga dari penelitian ini adalah proses *encode* pada gambar monyet.Bmp yang berukuran *file* 1,01 MB dan dapat menampung maksimal karakter untuk disisipkan sebesar 98.248 karakter. Hasil uji coba penyisipan karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar monyet.Bmp ditunjukkan pada tabel 4.4

Tabel 4.4 Data *encode* karakter pada gambar monyet.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *input*	Ukuran *File*
1	9.829	10%	1,01 MB
2	19.654	20%	1,01 MB
3	29.479	30%	1,01 MB

4	39.304	40%	1,01 MB
5	49.128	50%	1,01 MB
6	59.953	60%	1,01 MB
7	68.778	70%	1,01 MB
8	78.603	80%	1,01 MB
9	88.428	90%	1,01 MB
10	98.248	100%	1,01 MB

Uji coba keempat dari penelitian ini adalah proses *encode* pada gambar planet.Bmp yang berukuran *file* 2,39 MB dan dapat menampung maksimal karakter untuk disisipkan sebesar 231.717 karakter. Hasil uji coba penyisipan karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar planet.Bmp ditunjukkan pada tabel 4.5

Tabel 4.5 Data *encode* karakter pada gambar planet.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *input*	Ukuran *File*
1	23.183	10%	2,39 MB
2	46.354	20%	2,39 MB
3	69.526	30%	2,39 MB
4	92.698	40%	2,39 MB
5	115.870	50%	2,39 MB
6	139.041	60%	2,39 MB
7	162.213	70%	2,39 MB
8	185.385	80%	2,39 MB
9	208.556	90%	2,39 MB
10	231.717	100%	2,39 MB

Uji coba kelima dari penelitian ini adalah proses *encode* pada gambar wow.Bmp yang berukuran *file* 6,59 MB dan dapat menampung maksimal karakter untuk disisipkan sebesar 638.827 karakter. Hasil uji coba penyisipan karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar wow.Bmp ditunjukkan pada tabel 4.6

Tabel 4.6 Data *encode* karakter pada gambar wow.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *input*	Ukuran *File*
1	63.914	10%	6,59 MB
2	127.797	20%	6,59 MB
3	191.680	30%	6,59 MB
4	255.562	40%	6,59 MB
5	319.445	50%	6,59 MB
6	383.328	60%	6,59 MB
7	447.210	70%	6,59 MB
8	511.093	80%	6,59 MB
9	574.976	90%	6,59 MB
10	638.827	100%	6,59 MB

Uji kualitas gambar buah.Bmp, dengan jumlah maksimal karakter yang dapat ditampung sebesar 7.033 karakter dan berukuran *file* 74,8 KB sebagai informasi pada gambar aslinya. Dan dibandingkan dengan gambar buah.Bmp yang sudah disisipi karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar buah.Bmp. Hasil uji coba untuk proses uji kualitas gambar buah.Bmp ditunjukkan pada tabel 4.7.

Tabel 4.7 Uji kualitas gambar buah.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *Input*	Ukuran *File*	Nilai PSNR
1	703	10%	74,8 KB	75,61 db
2	1.406	20%	74,8 KB	70,45 db
3	2.110	30%	74,8 KB	67,63 db
4	2.813	40%	74,8 KB	65,98 db
5	3.516	50%	74,8 KB	64,57 db
6	4.220	60%	74,8 KB	64,24 db
7	4.923	70%	74,8 KB	64,09 db
8	5.626	80%	74,8 KB	63,95 db
9	6.330	90%	74,8 KB	63,51 db
10	7.033	100%	74,8 KB	62,51 db

Uji kualitas gambar keren.Bmp, dengan jumlah maksimal karakter yang dapat ditampung sebesar 85.070 karakter dan berukuran *file* 301 KB sebagai informasi pada gambar aslinya. Dan dibandingkan dengan gambar keren.Bmp yang sudah disisipi karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar keren.Bmp. Hasil uji coba untuk proses uji kualitas gambar keren.Bmp ditunjukkan pada tabel 4.8.

Tabel 4.8 Uji kualitas gambar keren.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *Input*	Ukuran *File*	Nilai PSNR
1	8.511	10%	301 KB	87,61 db
2	17.018	20%	301 KB	82,27 db

3	25.525	30%	301 KB	80,68 db
4	34.032	40%	301 KB	77,74 db
5	42.539	50%	301 KB	78,74 db
6	51.046	60%	301 KB	76,98 db
7	59.553	70%	301 KB	75,57 db
8	68.060	80%	301 KB	74,75 db
9	76.567	90%	301 KB	74,13 db
10	85.070	100%	301 KB	76,01 db

Uji kualitas gambar monyet.Bmp, dengan jumlah maksimal karakter yang dapat ditampung sebesar 98.248 karakter dan berukuran *file* 1,01 MB sebagai informasi pada gambar aslinya. Dan dibandingkan dengan gambar monyet.Bmp yang sudah disisipi karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar monyet.Bmp. Hasil uji coba untuk proses uji kualitas gambar monyet.Bmp ditunjukkan pada tabel 4.9.

Tabel 4.9 Uji kualitas pada gambar monyet.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *Input*	Ukuran *File*	Nilai PSNR
1	9.829	10%	1,01 MB	100,13 db
2	19.654	20%	1,01 MB	94,46 db
3	29.479	30%	1,01 MB	90,32 db
4	39.304	40%	1,01 MB	87,70 db
5	49.128	50%	1,01 MB	85,82 db
6	59.953	60%	1,01 MB	84,54 db
7	68.778	70%	1,01 MB	83,42 db

8	78.603	80%	1,01 MB	82,19 db
9	88.428	90%	1,01 MB	81,10 db
10	98.248	100%	1,01 MB	80,23 db

Uji kualitas gambar planet.Bmp, dengan jumlah maksimal karakter yang dapat ditampung sebesar 231.717 karakter dan berukuran *file* 2,39 MB sebagai informasi pada gambar aslinya. Dan dibandingkan dengan gambar planet.Bmp yang sudah disisipi karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar planet.Bmp Hasil uji coba untuk proses uji kualitas gambar planet.Bmp ditunjukkan pada tabel 4.10.

Tabel 4.10 Uji kualitas gambar planet.Bmp

No	Banyak Karakter *Input*	Prosentase Karakter *Input*	Ukuran *File*	Nilai PSNR
1	23.183	10%	2,39 MB	74,78 db
2	46.354	20%	2,39 MB	68,78 db
3	69.526	30%	2,39 MB	65,27 db
4	92.698	40%	2,39 MB	63,08 db
5	115.870	50%	2,39 MB	61,28 db
6	139.041	60%	2,39 MB	59,74 db
7	162.213	70%	2,39 MB	58,60 db
8	185.385	80%	2,39 MB	57,58 db
9	208.556	90%	2,39 MB	56,48 db
10	231.717	100%	2,39 MB	55,48 db

Uji kualitas pada gambar wow.Bmp, dengan jumlah maksimal karakter yang dapat ditampung sebesar 638.827 karakter dan berukuran *file* 6,59 MB sebagai informasi pada gambar aslinya.

Dan dibandingkan dengan gambar wow.Bmp yang sudah disisipi karakter sebesar 10%, 20% sampai dengan 100% dari total maksimal karakter yang bisa disisipkan pada gambar wow.Bmp. Hasil uji coba untuk proses uji kualitas gambar wow.Bmp ditunjukkan pada tabel 4.11.

Tabel 4.11 Uji kualitas gambar wow.Bmp

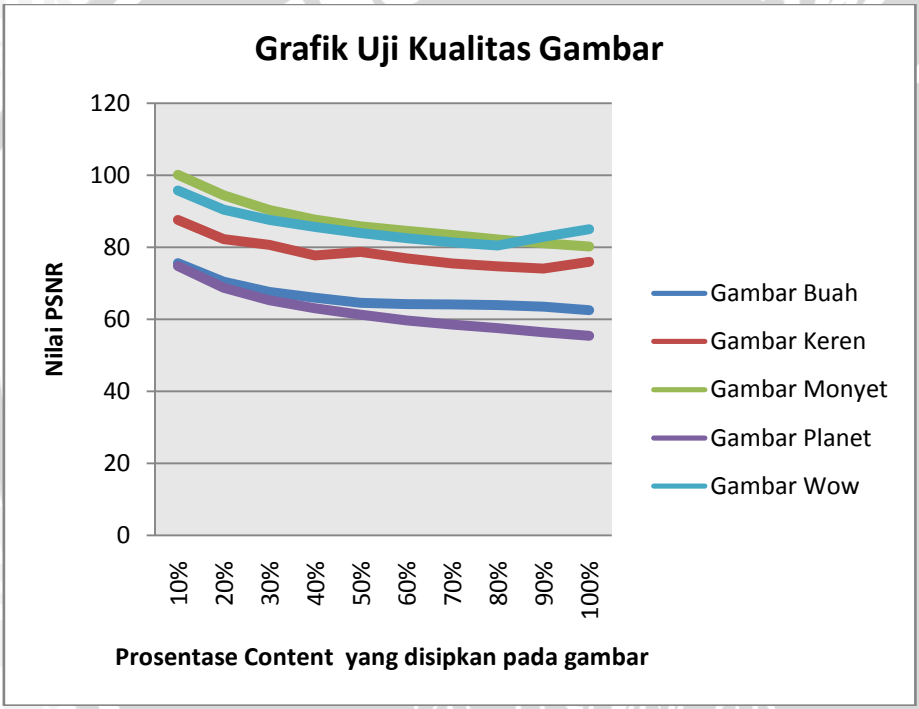
No	Banyak Karakter *Input*	Prosentase Karakter *Input*	Ukuran *File*	Nilai PSNR
1	63.914	10%	6,59 MB	95,77 db
2	127.797	20%	6,59 MB	90,47 db
3	191.680	30%	6,59 MB	87,58 db
4	255.562	40%	6,59 MB	85,62 db
5	319.445	50%	6,59 MB	83,97 db
6	383.328	60%	6,59 MB	82,56 db
7	447.210	70%	6,59 MB	81,38 db
8	511.093	80%	6,59 MB	80,50 db
9	574.976	90%	6,59 MB	82,86 db
10	638.827	100%	6,59 MB	85,05 db

4.5.3 Analisa dan Evaluasi Hasil

Berdasarkan Gambar 4.4 dapat diketahui bahwa nilai uji kualitas gambar yang telah disisipi pesan dengan jumlah *content* yang berbeda-beda dan dibandingkan dengan masing-masing gambar aslinya menghasilkan nilai PSNR yang memiliki grafik semakin menurun seiring dengan makin banyaknya *content* yang disisipkan, hal ini berlaku terhadap *file* yang berukuran kecil maupun *file* yang berukuran besar.

Namun terdapat perbedaan alur grafik pada gambar 4.4 pada 4 nilai uji kualiatas, masing-masing pada gambar keren di nilai persentase 50% dan 100% dan gambar wow di nilai persentase 90%

dan 100%. Pada keempat gambar yang telah berisikan pesan diatas terjadi perubahan nilai PSNR yang grafiknya lebih tinggi daripada nilai PSNR persentase sebelumnya, dimana hal ini bertentangan dengan nilai-nilai uji kualitas PSNR persentase sebelumnya dan lainnya dimana semakin banyak *content* yang disisipkan maka semakin kecil nilai PSNR yang didapatkan. Ini terjadi karena adanya kemungkinan *content* yang disisipkan mempunyai nilai *byte* yang sama dengan nilai *byte* terakhir per-*pixel* dari gambar aslinya. Analisa uji kualitas gambar diatas ditunjukkan pada gambar 4.4



Gambar 4.4 Grafik uji kualitas gambar

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pembahasan yang telah disampaikan dapat disimpulkan sebagai berikut :

1. Algoritma steganografi menggunakan metode LSB pada citra BMP telah menghasilkan gambar yang berisikan pesan dengan kualitas gambar yang tetap terjaga ketika dilihat secara kasat mata dan memiliki ukuran *size* yang tetap sama seperti gambar aslinya. Serta pada uji coba kali ini telah menghasilkan 10 gambar berbeda yang telah disisipkan pesan dengan jumlah prosentase yang berbeda-beda dimulai dari 10%, 20% sampai dengan 100% dari jumlah maksimal karakter yang bisa disisipkan ke dalam gambar penampung atau asli.
2. Gambar hasil steganografi dengan menggunakan metode LSB pada citra BMP menghasilkan nilai PSNR, dimana nilai PSNR akan semakin rendah pada saat pesan yang disisipkan semakin banyak tanpa melihat perbedaan besarnya ukuran file. Nilai uji kualitas PSNR dari steganografi metode LSB selain semakin rendah seiring dengan semakin banyaknya *content* yang disisipkan, tetapi juga dimungkinkan untuk memiliki nilai yang lebih tinggi walaupun *content* yang disisipkan lebih banyak daripada gambar lainnya yang memiliki nilai uji kualitas PSNR lebih kecil dengan jumlah *content* yang kecil pula. Ini terjadi karena adanya kemungkinan *content* yang disisipkan mempunyai nilai *byte* yang sama dengan nilai *byte* per-*pixel* dari gambar aslinya

5.2 Saran

Perlu di lakukan penelitian lebih lanjut mengenai steganografi menggunakan metode LSB (Least Significant *Bit*) agar karakter yang disisipkan dapat lebih banyak jumlahnya dari jumlah sebenarnya yang mampu disisipkan ke media penampung.


UNIVERSITAS BRAWIJAYA

DAFTAR PUSTAKA

Ahmad, Usman. 2005. *Pengolahan Citra Digital dan Teknik Pemrogramannya* : Graha Ilmu. Yogyakarta.

Arhami, Muhammad dan Desiani, Anita. 2005. *Pemrograman Matlab* : Andi. Yogyakarta.

Lestari, Desi. 2003. *Implementasi Teknik Watermarking Digital Pada Domain Dct Untuk Citra Berwarna.* Yogyakarta.

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

R.C. Gonzalez, R.E. Woods, 1992, *Digital Image Processing* : Addison-Wesley Publishing Company. New York

Robi'in, Bambang. 2004. *Pemrograman Grafis Multimedia Menggunakan Delphi* : Andi. Yogyakarta.

Wijaya, Marvin.ch dan Agus, Prijono. 2007. *Pengolahan Citra Digital Menggunakan Matlab* : Informatika. Bandung.

Sugiharto, Aris. 2006. *Pemrograman Gui dengan Matlab* : Andi. Yogyakarta.

Paulus, Erick dan Nataliani, Yessica. 2007. *Cepat Mahir Gui Matlab* : Andi. Yogyakarta.

Munir. Rinaldi, 2006, *Diktat Kuliah IF3058 Kriptografi* : Informatika. Bandung.

Phillip Rogaway & Don Coppersmith, *A Software-Opt imized Encrypt ion Algorithm.*