STEGANOGRAFI MENGGUNAKAN METODE PERUBAHAN SINONIM PADA DOKUMEN

SKRIPSI

Oleh :
Adam Zulfikar Mumtaz Diva
0510963003-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2012


UNIVERSITAS BRAWIJAYA

STEGANOGRAFI MENGGUNAKAN METODE PERUBAHAN SINONIM PADA DOKUMEN

SKRIPSI

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

Oleh :
Adam Zulfikar Mumtaz Diva
0510963003-96

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

2012

LEMBAR PENGESAHAN SKRIPSI

STEGANOGRAFI MENGGUNAKAN METODE PERUBAHAN SINONIM PADA DOKUMEN

oleh:
ADAM ZULFIKAR MUMTAZ DIVA
0510963003-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 09 Agustus 2012
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	Pembimbing II,
Edy Santoso, SSi., M.Kom	**Nurul Hidayat, SPd., MSc**
NIP. 197404142003121004	**NIP. 196804302002121001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf A., MSc
NIP. 196709071992031001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Adam Zulfikar Mumtaz Diva**
NIM	**: 0510963003-96**
Jurusan	**: Matematika**
Program Studi	**: Ilmu Komputer**
Penulis tugas akhir berjudul	**: Steganografi Menggunakan Metode Perubahan Sinonim pada Dokumen**

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, 09 Agustus 2012
Yang menyatakan,

Adam Zulfikar Mumtaz Diva
NIM. 0510963003

Steganografi Menggunakan Metode Perubahan Sinonim pada Dokumen

ABSTRAK

Kompresi huffman menyederhanakan kode bit yang dimasukkan kedalam *covertext*, dengan semakin sederhana atau semakin pendeknya kode bit yang bisa dimasukkan kedalam covertext maka jumlah pesan rahasia yang dapat disisipkan akan meningkat. Dalam prakteknya, penyisipan kode bit pesan rahasia akan menggunakan metode perubahan sinonim pada dokumen teks. Perubahan sinonim pada teks dapat mengurangi kecurigaan dari pihak luar untuk mengetahui kandungan pesan rahasia dalam *stegotext*.

Pertama-tama pada tahap embedding, pesan rahasia akan dikompresi oleh algoritma huffman menjadi bit string. Selanjutnya sistem akan memilih beberapa kata secara random sesuai kebutuhan bit string. Kelompok kata yang sudah terpilih akan dicari sinonimnya menggunakan kamus Wordnet. Kamus tersebut memiliki sinonim set yang dapat membantu memudahkan proses embedding.

Pada proses pencarian sinonim juga akan dihasilkan bit string yang merupakan kode sinonim yang unik untuk tiap sinonim kata. Bit string dari pesan rahasia akan dicocokkan dengan bit string sinonim kata, hal ini bertujuan untuk memilih sinonim kata yang sesuai dengan bit string pesan rahasia. Bit string dari sinonim yang telah dicocokkan dengan bit string dari pesan rahasia akan digunakan sebagai sinonim pengganti pada *covertext*.

Text Steganography Using Synonym Replacement for Text Document

ABSTRACT

The huffman code help simplify the bit string which is inserted into the covertext, when the bit string is shorter or more simplified, that means more message can be inserted into covertext. Inside the system, bit code insertion will be using the synonym replacement method for text document. Synonym replacement can reduce suspicion from the other people who doesn't need to know what the message is inside the stegotext.

At first, there is embedding step. The secret message will be compressed using huffman compression method to produce huffman bit code. Next, the system will choose random words from the covertext that costumised as the lengt from the compressed bit strinto acomodate the huffman bit code. The Wordnet will choose the synonym set from the selected words.

Inside the synonym searching process, the system will generate the bit code that represent the synonym. The huffman bit code will be matched with the bit code from synonym, then the system will replace words from the covertext by the generated synonym based on the similarity between those bits.

KATA PENGANTAR

Alhamdulillah, dengan mengucapkan puji syukur kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan skripsi ini. Skripsi yang berjudul **"Steganografi Menggunakan Metode Perubahan Sinonim pada Dokumen"** merupakan salah satu syarat memperoleh gelar Sarjana Komputer pada program studi Ilmu Komputer Jurusan Matematika Fakultas MIPA Universitas Brawijaya. Tidak dapat dipungkiri bahwa tidak mungkin penulis dapat menyelesaikan skripsi ini tanpa bantuan dan dukungan dari banyak pihak. Untuk itu, dengan ketulusan dan kerendahan hati penulis menyampaikan ucapan terima kasih yang sebesar-besarnya kepada:

1. Edy Santoso, SSi., M.Kom., selaku dosen pembimbing utama yang telah meluangkan waktu untuk memberikan pengarahan dan masukan bagi penulis.
2. Nurul Hidayat, SPd., MSc., selaku dosen pembimbing kedua yang telah meluangkan waktu untuk memberikan pengarahan dan masukan bagi penulis.
3. Drs. Marji, MT., selaku ketua program studi Ilmu Komputer yang telah banyak memberikan bimbingan serta bantuan.
4. Dr. Abdul Rouf Alghofari, MSc., selaku ketua Jurusan Matematika.
5. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika Fakultas MIPA Universitas Brawijaya.
6. Segenap staf dan karyawan di Jurusan Matematika Fakultas MIPA Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan proposal skripsi ini.
7. Kedua orang tua, bapak ibu, dan adik terima kasih atas semua doa, kasih sayang dan perhatian yang tulus serta dukungan yang telah diberikan.
8. Teman-teman di Program Studi Ilmu Komputer Fakultas MIPA Universitas Brawijaya khususnya Tim Linggis yang

telah banyak memberikan dukungannya demi kelancaran pelaksanaan penyusunan skripsi ini.

9. Dan semua pihak yang telah terlibat baik secara langsung maupun tidak langsung yang tidak dapat penulis sebutkan satu per satu terima kasih atas semua bantuan yang telah diberikan.

Semoga skripsi ini bermanfaat bagi pembaca sekalian. Akhirnya, penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan dan mengandung banyak kekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, 09 Agustus 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR


UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

DAFTAR SOURCE CODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kebutuhan manusia untuk berkomunikasi sekarang semakin kompleks. Banyaknya media untuk berkomunikasi juga menimbulkan banyak celah dalam keamanan dalam berkomunikasi. Bagi sebagian orang, sebuah pesan belum tentu ingin dibagi dengan khalayak umum.Dengan kebutuhan yang meningkat maka manusia membuat metode-metode untuk memenuhi kepentingan mereka, dalam hal ini guna menyampaikan sebuah pesan yang mana khalayak umum tidak sanggup untuk membacanya. Steganografi adalah seni dan ilmu menulis pesan tersembunyi atau menyembunyikan pesan dengan suatu cara sehingga selain si pengirim dan si penerima, tidak ada seorangpun yang mengetahui atau menyadari bahwa ada suatu pesan rahasia. Sebaliknya, kriptografi menyamarkan arti dari suatu pesan, tapi tidak menyembunyikan bahwa ada suatu pesan.

Kata "steganografi" berasal dari bahasa Yunani *steganos*, yang artinya "tersembunyi atau terselubung", dan *graphein*, "menulis".Pada metode steganografi cara ini sangat berguna jika digunakan pada cara steganografi komputer karena banyak format berkas digital yang dapat dijadikan media untuk menyembunyikan pesan.

Kelebihan steganografi jika dibandingkan dengan kriptografi adalah pesan-pesannya tidak menarik perhatian orang lain. Pesan-pesan berkode dalam kriptografi yang tidak disembunyikan, walaupun tidak dapat dipecahkan, akan menimbulkan kecurigaan. Seringkali, steganografi dan kriptografi digunakan secara bersamaan untuk menjamin keamanan pesan rahasianya.Mohammad Shirali-Shahreza mengusulkan metode *text steganography* dengan perubahan ejaan bahasa Inggris US (*United State*)-UK (*United Kingdom*). Ejaan bahasa Inggris US akan mewakili biner '0'sedangkan ejaan bahasa Inggris UK mewakili biner '1'(Mohammad Shirali-Shahreza, 2008). Kata yang tidak memiliki perbedaan ejaan tidak akan diubah. Metode ini memiliki kelemahan yaitu kapasitasnya yang kecil untuk menampung data yang akan disembunyikan, hal ini tergantung dari ukuran *covertext* dan jumlah kata yang memiliki perbedaan ejaan antara Amerika dan Inggris. Cara kerja metode ini mirip dengan *Semantic Method* yang

menggunakan perubahan sinonim. *Word* atau kata yang memiliki sinonim mewakili biner '0' sedangkan *Synonym* atau sinonim dari *Word* mewakili biner '1'(Bender, 1996).Pada tahun 2008 Nanhe. Et al mengusulkan metode steganografi pada file text dimana pada penelitian tersebut menggunakan salah satu *chapter* dari sebuah novel untuk menyisipkan sebuah covertext. *Text steganography* membutuhkan dua elemen yaitu file teks yang akan digunakan sebagai media untuk menyembunyikan pesan yang disebut *covertext* dan pesan yang akan disembunyikan (*secret message*). Saat digabungkan *covertext* dan *secret message* akan berubah menjadi stegotext (file teks hasil steganografi). Keunikan dari penelitian ini adalah penggunaan Huffman code untuk kompresi bit pada secret teks sehingga menghemat jumlah bit yang akan disisipkan kedalam covertext.

Penelitianinimengembangkan steganografi yang kuat dengan penggantian sinonim, yang mengubah pesan menjadi tidak terlalu mencolok secara leksikal. Menjawab kebutuhan steganografi yang terkendala dengan batasan jumlah pesan yang dapat disipkan, penelitian ini menggunakan kompresi terhadap pesan yang akan disisipkan. Dengan latar belakang tersebut maka ditentukan judul skripsi **"Steganografi Metode Perubahan Sinonim pada Dokumen"**

1.2 Rumusan Masalah

Masalah yang akan dibahas pada penelitian ini adalah

1. Bagaimana implementasi sebuah perangkat lunak yang menerapkan metode steganografi berbasis perubahan sinonim bahasa inggris pada pesan yang sudah terkompresi.
2. Seberapa besar *capacity ratio* yang dicapai.
3. Efisiensi metode kompresi huffman yang diterapkan pada pesan tersembunyi.

1.3 Batasan masalah

Batasan masalah dari penelitan ini adalah

1. Media yang digunakan untuk menyimpan pesan adalah file berbahasa inggris dengan format *.txt
2. Kompresi dilakukan pada pesan tersembunyi.
3. Pesan tersembunyi (*secret message*) adalah pesan yang akan disisipkan.

4. Media penyimpanan pesan rahasia mengandung sejumlah kata yang memiliki perbedaan sinonim dalam bahasa inggris.
5. *Stegotext* yang dihasilkan belum tentu benar maknanya secara gramatikal.

1.4 Tujuan penelitian

Tujuan penelitian yang ingin dicapai antara lain :

1. Penerapan steganografi dengan metode gabungan dari metode leksikal perubahan sinonim dan kompresi teks untuk pesan tersembunyi dengan menggunakan Huffman code.
2. Menghitung besar *capacity ratio* pada *stegotext*.
3. Menghitung tingkat efisiensi metode kompresi huffman yang diterapkan pada pesan tersembunyi.

1.5 Manfaat penelitian

Manfaat yang ingin dicapai dari penelitian ini adalah menghasilan perangkat lunak yang mengimplementasikan steganografi untuk mengamankan pesan tersembunyi kedalam file teks untuk membantu menyisipkan pesan tersembunyi bersifat rahasia yang ingin dikirim.

1.6 Metodologi penelitian

Metodologi yang digunakan dalam penulisan skripsi ini antara lain :

1. Studi Literatur
   Mempelajari teori yang berhubungan dengan teks steganografi dari berbagai sumber.
2. Perancangan dan Implementasi Perangkat Lunak
   Membuat rancangan *text steganography* dan diimplementasikan untuk membuat perangkat lunak.
3. Uji Coba dan Analisis Hasil
   Menguji coba perangkat lunak yang dihasilkan dan menganalisa hasil dari steganografi tersebut.

1.7 Sistematika Penulisan

Skripsi ini disusun dengan pembagian bab sebagai berikut :

BAB I : PENDAHULUAN

Bab ini menjelaskan latar belakang, rumusan masalah, batasan masalah, tujuan penelitian,

manfaat penelitian, metodologi penelitian, dan sistematika penulisan skripsi.

BAB II : TINJAUAN PUSTAKA

Bab ini mencantumkan dasar teori yang menjadi acuan dalam proses pembuatan perangkat lunak steganografi.

BAB III : METODOLOGI DAN PERANCANGAN

Bab ini membahas bagaimana rancangan sistem dan u*ser interface* perangkat lunak steganografi yang akan dibuat.

BAB IV : IMPLEMENTASI DAN PEMBAHASAN

Dalam bab ini dijelaskan mengenai implementasi program, ujicoba dan analisis, perangkat lunak yang dibuat.

BAB V : PENUTUP

Bab ini berisi kesimpulan dari pembahasan dan saran yang diharapkan bermanfaat untuk pengembangan skripsi selanjutnya

DAFTAR PUSTAKA

BAB II
TINJAUAN PUSTAKA

2.1 Steganografi

Steganografi merupakan seni penyembunyian pesan ke dalam pesan lainnya sedemikian rupa sehingga orang lain tidak menyadari ada sesuatu di dalam pesan tersebut. Kata steganografi (*steganography*) berasal dari bahasa Yunani yaitu *steganos* yang artinya tersembunyi atau terselubung dan *graphein*, yang artinya menulis, sehingga kurang lebih artinya adalah "menulis tulisan yang tersembunyi atau terselubung" (Sellars, 1996).

Teknik ini meliputi banyak sekali metoda komunikasi untuk menyembunyikan pesan rahasia.Metoda ini termasuk tinta yang tidak tampak, microdots, pengaturan kata, tanda tangan digital, jalur tersembunyi dan komunikasi spektrum lebar. Catatan pertama tentang steganografi ditulis oleh seorang sejarawan Yunani, Herodotus, yaitu ketika Histaeus seorang raja kejam Yunani dipenjarakan oleh Raja Darius di Susa pada abad 5 Sebelum Masehi.Histaeus harus mengirim pesan rahasia kepada anak laki-lakinya, Aristagoras, di Militus. Histaeus menulis pesan dengan cara mentato pesan pada kulit kepala seorang budak dan ketika rambut budak itu mulai tumbuh, Histaeus mengutus budak itu ke Militus untuk mengirim pesan di kulit kepalanya tersebut kepada Aristagoras.

Teknik steganografi yang lain adalah tinta yang tak terlihat. Teknik ini pertama digunakan pada zaman Romawi kuno yaitu dengan menggunakan air sari buah jeruk, urine atau susu sebagai tinta untuk menulis pesan. Cara membacanya adalah dengan dipanaskan di atas nyala lilin, tinta yang sebelumnya tidak terlihat, ketika terkena panas akan berangsur-angsur menjadi gelap, sehingga pesan dapat dibaca. Teknik ini pernah juga digunakan pada Perang Dunia II. Pada abad 20, steganografi benar-benar mengalami perkembangan. Selama berlangsung perang Boer, Lord Boden Powell (pendiri gerakan kepanduan) yang bertugas untuk membuat tanda posisi sasaran dari basis artileri tentara Boer, untuk alasan keamanan, Boden Powell menggambar peta-peta posisi musuh pada sayap kupu-kupu agar gambar-gambar peta sasaran tersebut terkamuflase.

Pada umumnya, pesan steganografi muncul dengan rupa lain seperti gambar, artikel, daftar belanjaan, atau pesan-pesan lainnya. Pesan yang tertulis ini merupakan tulisan yang menyelubungi atau menutupi. Contohnya, suatu pesan bisa disembunyikan dengan menggunakan tinta yang tidak terlihat di antara garis-garis yang kelihatan.

Sebuah pesan steganografi (*plaintext*), biasanya pertama-tama dienkripsikan dengan beberapa arti tradisional, yang menghasilkan *ciphertext*. Kemudian, *covertext* dimodifikasi dalam beberapa cara sehingga berisi *ciphertext*, yang menghasilkan *stegotext*. Contohnya, ukuran huruf, ukuran spasi, jenis huruf, atau karakteristik *covertext* lainnya dapat dimanipulasi untuk membawa pesan tersembunyi; hanya penerima (yang harus mengetahui teknik yang digunakan) dapat membuka pesan dan mendekripsikannya.

Tujuan dari steganografi adalah merahasiakan atau menyembunyikan keberadaan dari sebuah pesan tersembunyi atau sebuah informasi. Dalam prakteknya, kebanyakan pesan disembunyikan dengan membuat perubahan tipis terhadap data digital lain yang isinya tidak akan menarik perhatian dari penyerang potensial, sebagai contoh sebuah gambar yang terlihat tidak berbahaya. Perubahan ini bergantung pada kunci (sama pada kriptografi) dan pesan untuk disembunyikan. Orang yang menerima gambar kemudian dapat menyimpulkan informasi terselubung dengan cara mengganti kunci yang benar ke dalam algoritma yang digunakan.

2.1.1 Kategori dalam Steganografi

Secara umum steganografi dikelompokkan kedalam 2(dua) kategori, yaitu :

a. Steganografi *linguistic*.

Teknik ini mengambil keuntungan dari struktur bahasa natural untuk menyisipkan informasi tersembunyi.Dalam hal ini bisa memanfaatkan struktur sebuah bahasa yang dikenal manusia untuk menyisipkan informasi tersembunyi, misalnya saja dengan ambuguitas makna dari sebuah kata.

b. Steganografi teknis.

Teknik ini menjangkau lebih luas daripada sekedar permainan makna kata dalam bahasa natural manusia.Teknik ini tidak dibatasi oleh kata-kata yang tertulis, sebuah kalimat

bahkan sebuah paragraf, namun mengikutsertakan beberapa macam alat dan metodologi untuk menyisipkan informasi tersembunyi kedalam sebuah file dimana informasi tersebut tidak akan berubah walaupun file yang digunakan mengalami perubahan bentuk maupun kompresi. (Khan Rafat, 2010)

2.1.*2 Syntactical Steganography*

Pendekatan *syntactical steganography* memanfaatkan struktur dari sebuah bahasa.Pendekatan ini memakai *Context Free Grammar* (CFG) untuk membangun sebuah stegotext yang benar ejaannya.Adapun algoritma yang terkenal dikemukakan oleh Chapman adalah NICETEXT, algortima ini juga berbasis CFG. NICETEXT menggunakan cover text sebagai sumber dari pola sintaksis. NICETEXT akan memproses cover text sehingga menghasilkan sebuat bagan kalimat. Misalnya [*(noun) (verb) (prep) (det) (noun)*] untuk "*I sat in the tree*". Ia juga menghimpun kosa kata yang ditemukan dalam cover text, kemudian diasosiasikan secara berubah-ubah dengan angka biner 1 dan 0. Meskipun NICETEXT menghasilkan *covertext* yang benar secara sintaksis tetapi dalam beberapa kasus ia gagal secara *semantic*. *Output text* cenderung tidak sesuai dengan tata bahasa dan seringkali mengalami anomaly *semantic*.

Tahun 2008 Mohammad Shirali-Shahreza mengusulkan metode *text steganography* dengan perubahaan ejaan bahasa Inggris US dan UK. Metode ini digunakan untuk menyisipkan pesan ke dalam teks bahasa Inggris. Teknik penyisipan ini membutuhkan daftar kata yang memiliki perbedaan ejaan antara bahasa Inggris Amerika dan bahasa Inggris asli. Pesan yang akan disisipkan akan diubah ke dalam format biner 0 dan 1. Saat akan menyembunyikan pesan program akan mencari kata-kata di dalam teks yang sesuai dengan daftar kata yang telah dimiliki. Kemudian program akan mengganti kata tersebut dengan ejaan US untuk mewakili bit 0 dan mengganti ejaan UK untuk mewakili bit 1. Untuk proses meng-ekstrak pesan dari *covertext* program akan mengidentifikasi tipe kata di dalam teks dengan mencocokkan daftar kata yang memiliki perbedaaan ejaan US dan UK. Kemudian susunan biner 0 atau 1 yang telah diwakili oleh kata tersebut akan disimpan dan dikembalikan ke format pesan

awal sehingga dapat dibaca oleh *user*. Metode ini menghasilkan rata-rata capacity ratio sebesar 12,99 bit/kB.

Umumnya, teknik *Syntatical steganography* menghasilkan teks yang secara sintaks berbentuk baik tanpa menghasilkan susunan kata semantic yang baik. (Nanhe et al, 2008)

2.1.3 *Lexical Steganography*

Dalam *lexicalsteganography*, unit leksikal dari teks bahasa natural seperti kata-kata digunakan untuk menyembunyikan bit-bit rahasia. Sebuah kata bisa diganti dengan sinonimnya apabila dalam daftar sinonim kata ditemukan pola bit yang sama dengan bit rahasia yang akan di *embed* kedalam kata tersebut. Sebagai contohnya adalah kalimat berikut :

"Pune is a nice little city"

Jika dicari sinonim dari kata "*nice*" maka akan diperoleh daftar kata {*nice, wonderful, great,* dan *decent*}. Setiap sinonim akan direpresentasikan kedalam dua bit seperti yang ditunjukkan dalam tabel 2.1 :

Tabel 2.1 Tabel sinonim kata "nice"

Word	Code
Nice	00
Wonderful	01
Great	10
Decent	11

Bergantung dari bit rahasia(*secret bits)*yang ada pada *ciphertext* maka sinonim untuk kata "*nice*" akan dipiluh dan dimasukkan kedalam *stegotext*. Jadi, kemungkinan *stegotext* akan seperti :

a) Pune is a nice little city.
b) Pune is a wonderful little city.
c) Pune is a decent little city.
d) Pune is a great little city.

Tehnik leksikal menghasilkan kualitas yang lebih baik dari tehnik *syntactical*. Sangat susah untuk mencari keberadaan pesan rahasia jika digunakan serangan secara statistic. Penggantian kata adalah hal yang paling kritis dari tehnik ini.Contohnya saja di atas menggunakan metode penggantian sinonim, beberapa kata dapat memberikan makna yang berbeda.Kata "bank" bisa bermakna sebuah timbunan atau sebuah institusi tempat menyimpan, meminjam uang dan segala jenis transaksi finansial. Jika tidak digunakan sinonim yang masuk akal kedalam sebuah *covertext* maka hasilnya akan terlihat sangat mencurigakan.

1. Bring those instruments.
2. Bring those tool.

Tantangan lain dari pergantian sinonim pada kata adalah infleksi kelas, yaitu kombinasi kata yang diperbolehkan dan yang tidak diperbolehkan. Pada dua kalimat di atas penggantian kata "instrument" menjadi "tool" akan membuat kalimat tersebut salah secara tata bahasa.

2.1.4 *Ontological technique*

Dari semua teknik yang diulas, teknik *ontology* adalah satu dari semua pendekatan model semantik yang rumit. Metode ini tidak mengikuti metode semantik biasa yang menjaga model semantiknya tetap utuh terjaga, namun metode ini lebih cenderung kearah "makna" dari sebuah kalimat.Sebuah model untuk mengevaluasi persamaan makna dalam konteks sebuah kalimat.

Tehnik *ontology* juga mempunyai masalah.Hasil transformasi kadang mempengaruhi hubungan *semantic* dari teks.Teori yang baru dalam bahasa masih memperdebatkan ketidakcocokan pada *level semantic* dan *sintaksis*, dimana menurutnya pola sintatik itu sendiri sebenarnya sudah cukup. Secara statistik, berbagai macam struktur sintaks tidak sama kedudukan pendistribusian yang tidak sama. Jenis teks yang berbeda bisa memberikan struktur sintaks yang berbeda dan jika dilakukan perubahan pada struktur tersebut maka akan menjadikan teks tersebut "rusak" secara statistika, sebuah celah keamanan pada program.(Nanhe et al, 2008)

2.2 Kompresi

Kompresi adalah proses pengubahan sekumpulan datamenjadi bentuk kode dengan tujuan untuk menghemat kebutuhan tempat penyimpanan data dan waktu untuk transmisi data (Alvin,2009). Jika data tersebut hendak dipergunakan kembali, maka harus dilakukan dekompresi, yaitu pengubahan kode-kode menjadi data awal.

Saat ini, telah banyak tersedia algoritma untuk melakukan kompresi, antara lain*: Huffman, LIFO, LZHUF, LZ77 dan variannya (LZ78, LZW, GZIP), Dynamic Markov Compression (DMC), Block-Sorting Lossless, Run-Length, Shannon-Fano, Arithmetic, PPM (Prediction by Partial Matching), Burrows-Wheeler Block Sorting, dan Half Byte*. Namun, ada beberapa faktor yang dijadikan pertimbangan dalam memilih algoritma yang akan digunakan dalam kompresi data, yaitu :

1. Sumber daya yang dibutuhkan (Memori, kecepatan PC)
2. Kecepatan kompresi
3. Ukuran hasil kompresi
4. Besarnya redudansi
5. Ketepatan hasil dekompresi
6. Kompleksitas algoritma

Berdasarkan output hasil kompresi, metode kompresi dapat dikelompokkan menjadi dua, yaitu :

1. Kompresi *loseless*

   Hasil dekompresi dari data hasil kompresi akan tepat sama persis dengan data sebelum kompresi. Contoh aplikasi : ZIP, RAR, GZIP, 7-ZIP

   Adapun kelemahan dari metode ini adalah rasio kompresi yang rendah.Rasio dapat dihitung dengan persamaan 2.1.

   $$\text{Rasio kompresi} = \frac{\quad}{\quad} \qquad (2.1)$$

   Teknik ini digunakan juga dibutuhkan data dimana setelah dikomresi harus dapat didekompresi lagi dan menghasilkan data yang tepat sama dengan data asli. Contoh pada data teks, data program/biner dan beberapa image random seperti GIF dan PNG.

2. Kompresi *lossy*

   Hasil dekompresi dari data hasil kompresi tidak tepat sama persis, tetapi persepsi terhadap data semantik tetap sama. Contoh : MP3, streaming media, JPEG, MPEG, dan WMA.

   Kelebihan dari metode ini adalah ukuran file lebih kecil dibanding *loseless* namun masih tetap memenuhi syarat yang digunakan. Biasanya teknik ini membuang bagian data yang sebenarnya tidak begitu berguna, tidak begitu dirasakan, tidak begitu dilihat sehingga manusia masih beranggapan bahwa data tersebut masih bisa digunakan walaupun sudah dikompresi.

2.2.1 Pengkodean Huffman

Dalam ilmu komputer, pengkodean Huffman adalah sebuah *entropy encoding algorithm* yang digunakan untuk kompresi data yang bersifat loseless.*Huffman Coding* menggunakan struktur pohon dalampemrosesannya. Pohon adalah graf tak berarah yangtidak mengandung sirkuit(Munir, 2003). Di dalam struktur pohondikenal terminologi *parent* (orang tua) dan *child*(anak). *Parent* (orang tua) yaitu sebuah simpul yangmemilki lintasan ke simpul lain dengan tingkatan(*level*) di bawahnya. *Child* (anak) yaitu sebuah simpulyang memiliki lintasan ke simpul lain dengnatingkatan (*level*) di atasnya.

Berdasarkan jumlah anak, pohon dapat dikategorikan sebagai pohon *n – ary*. Pohon dengan orang tua yang hanya memiliki satu anak, disebut pohon uner. Pohon dengan orang tua yang memiliki dua anak, disebutpohon biner, dan seterusnya.

Pohon Huffman menggunakan strukrut pohon biner,yaitu struktur pohon dengan setiap simpul orang tuayang hanya memiliki maksimal 2 simpul anak.Penyusunan data dalam struktur pohon Huffman menggunakan kode awalan (*prefix code*). Kode awalan adalah himpunan kode (biner) dimana anggota kumpulan yang satu bukan merupakan anggota kumpulan yang lain.

Kompresi Huffman mempunyai rata-rata pengurangan jumlah bit string sebesar 33% dari bit aslinya(Nanhe et al,2008) .

2.2.1.1 Algoritma Huffman

Biasanya sebuah karakter dikodekan dalam kodeASCII (8 bit) atau kode Unicode (16 bit) yang telahmemiliki panjang tetap

(*fixed* - *length*). Berbedadengan cara pengkodean ASCII atau Unicode,pengkodean dengan *Huffman Coding* menggunakan panjang bit yang bervariasi dalam mengkodekan sebuah karakter.Karakter yang memiliki frekuensi kemunculan lebih besar memiliki panjang bit yang lebih pendek, sebaliknya karakter yang memiliki frekuensi kemunculan yang lebih kecil memilki panjang bit yang lebih panjang. Hal ini menyebabkan kode untuk sebuah karakter dengan menggunakan cara pengkodean *Huffman Coding* tidak tetap seperti dalamASCII *code* atau Unicode.

Seperti dijelaskan sebelumnya pengkodean dengancara *Huffman Coding* menggunakan kode awalan(*prefix code*) yang direpresentasikan dalam struktur pohon biner. Cara penyusunan ke dalam pohon biner adalah dengan memberikan label '0' untuk cabang kiri dan label '1' untuk cabang kanan. Hal ini dilakukan secara berulang hingga akhirnya terbentuk rangkaian bit yang merupakan kode awalan (*prefix code*).

Langkah – langkah pembentukan pohon Huffmandengan menggunakan Algoritma Huffman :

Algorithm Huffman(X)
// **input**: String X of length n with d distinct characters
// **output**: coding tree for X

Compute the frequency function f ofcharacters in X
Initialize empty priority queue Q of trees
// Fill Q
for each character, c, in X **do**
Create a single-node binary tree T storing c
Insert T into Q with key f(c)

while Q.size() > 1 **do**
f1 = Q.minKey()
T1 = Q.removeMin()
f2 = Q.minKey()
T2 = Q.removeMin()
Create new binary tree T with left
subtree T1 and right subtree T2
Insert T into Q with key f1 + f2
return tree Q. removeMin()

2.2.1.2 Pengkodean dengan *Huffman coding*

Berikut ini adalah sebuah contoh cara pengkodeansebuah string. Misalkan akan mengkodekansebuah string "AABCABC".

Dalam ASCII string tersebut akan dikodekan sebagai01000001010000010100000100100001101000001010000100 1000011 dengan perincian kode setiap karakter adalah :

- Karakter A dalam kode ASCII 01000001
- Karakter B dalam kode ASCII 01000010
- Karakter C dalam kode ASCII 01000011

Pengkodean string ini ke dalam kode ASCII akanmembutuhkan memori sebesar 7 x 8 bit atau samadengan 7 byte.

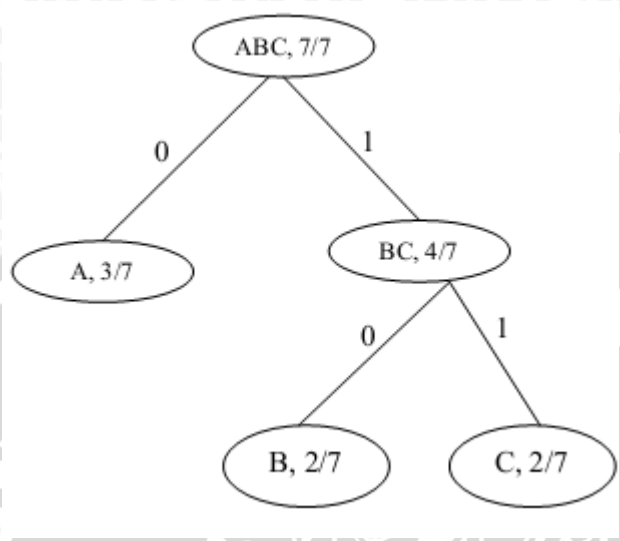
Berikut adalah cara pengkodean string yang samadengan menggunakan *Huffman Coding*. Berdasarkan algoritma yang telah disebutkan sebelumnya, dapat dihitung frekuensi kemunculan dari setiapkarakter dalam string X.

Tabel yang menunjukkan frekuensi kemunculan setiapkarakter tersebut ditunjukkan pada tabel2.2.

Tabel2.2 Tabel Frekuensi dari kode Huffman untukstring "AABCABC"

Huruf	Frekuensi	Peluang
A	3	3/7
B	2	2/7
C	2	2/7

Berdasarkan tabel yang ditunjukkan oleh tabel2.2dapat disusun pohon Huffman seperti yangditunjukkan oleh gambar 2.1 dengan kode awalan(*prefix code*).



Gambar 2.1 Pohon Huffman untuk string "AABCABC"

Berdasarkan pohon Huffman yang ditunjukkan padatabel 2.1, dapat ditentukan Kode Huffman untuk setiap simbol yang ada dalam string"AABCABC". Kode Huffman untuk setiap simbol(karakter) dalam string "AABCABC" ditunjukkan dalam tabel2.3.

Tabel 2.3 Kode Huffman untuk string "AABCABC"

Simbol	Kode Huffman
A	0
B	10
C	11

Dengan menggunakan Kode Huffman yangditunjukkan pada tabel 2.3, maka pengkodean string"AABCABC" dalam kode Huffman adalah : 00101101011 dimana representasi ini hanyamembutuhkan memori sebesar 11 bit (1,375 byte).

Bandingkan jumlah memori yang dibutuhkan apabila menggunakan kode ASCII dan kode Huffman. KodeHuffman hanya membutuhkan memori sebesar19,642% dari kode ASCII.

2.2.1.3 Penguraian Kode Huffman (*Decoding*)

Penguraian kode (*Decoding*) adalah sebuah proses untuk menyusun kembali data yang telah dikodekan sebelumnya sehingga

informasi yang diterima dapat dibaca dan diolah. Penguraian kode (*Decoding*) iniadalah lawan dari pengkodean (*Encoding*).Ada 2 cara penguraian kode Huffman. Cara yang pertama adalah menggunakan pohon Huffman sedangkan cara yang kedua adalah menggunakan table kode Huffman.

Berikut adalah penjelasan untuk cara pertama, yaitumenggunakan pohon Huffman. Seperti yangdijelaskan sebelumnya, pohon Huffman adalah pohon biner dengan menggunakan kode awalan (*prefix code*).Hal ini memudahkan proses penguraian kode. Langkah – langkah yang dilakukan dalam penguraiankode (*decoding*) menggunakan pohon Huffman adalahsebagai berikut:

1. Baca bit pertama dari string biner masukan
2. Lakukan traversal pada pohon Huffman mulai dari akar sesuai dengan bit yang dibaca. Jika bit yang dibaca adalah 0 maka baca anak kiri, tetapi jika bit yang dibaca adalah 1 maka baca anakkanan.
3. Jika anak dari pohon bukan daun (simpul tanpa anak) maka baca bit berikutnya dari string binermasukan.
4. Hal ini diulang (traversal) hingga ditemukan daun.
5. Pada daun tersebut simbol ditemukan dan proses penguraian kode selesai. Proses penguraian kode ini dilakukan hingga keseluruhan string biner masukan diproses.

Contoh cara penguraian kode menggunakan pohonHuffman. Dengan menggunakan kode hasil enkripsiyang telah ditunjukkan dalam proses pengkodean(*Encoding*) sebelumnya, akan ditunjukkan cara penguraian kode (*decoding*) menggunakan pohon Huffman.

Hasil pengkodean string "AABCABC" kedalam string biner adalah 00101101011. Bit pertama dari string biner tersebut adalah 0. dengan menggunakanpohon Huffman yang ditunjukkan dalam gambar 2.1, ditemukan bahwa anak kiri nya adalah daun yang menyimpan simbol A. dengan melakukan hal yang sama pada bit kedua (angka 0), ditemukan symbol berikutnya adalah A. pada bit ketiga (angka 1),ditemukan bahwa anak kanannya bukanlah sebuah daun. Oleh karena itu harus diambil bit berikutnyayaitu bit keempat (angka 0). Karena bit keempatadalah 0 maka harus diambil anak kiri. Pada anak kiritersebut ditemukan sebuah daun yang menyimpan simbol B. Dengan melakukan hal ini secara berulang hingga bit

terakhir, maka akan ditemukan bahwa string biner 00101101011 adalah hasil pengkodean(enkripsi) dari string "AABCABC".

Cara kedua untuk menguraikan kode Huffman adalah dengan menggunakan tabel kode Huffman.Oleh karena kode Huffman disusun menggunakan kode awalan(*prefix code*) maka dapat dipastikan bahwa sebuah kode untuk sebuah simbol / karakter yang satu tidak boleh menjadi awalan dari kode / simbol yang lain. Oleh karena itu pastilah string biner yang berisi hasil enkripsi dapat dipisahkan dengan mudah berdasarkan setiap rangkaian bitnya untuk diuraikan menjadi informasi semula. Yang perlu dilakukanhanyalah melihat setiap rangkaian bit yang ditemukan dalam string biner hasil enkripsi di dalam tabel kode Huffman.

Berikut ini akan diberikan contoh cara penguraian kode dengan menggunakan Tabel kode Huffman.Misalkan akan dilakukan penguraian kode (*decoding*)berdasarkan string biner yang dihasilkan dari proses pengkodean (*encoding*) sebelumnya. String biner yangdihasilkan sebelumnya adalah 00101101011 dengan tabel kode Huffman yang ditunjukkan pada tabel 2.3.

String biner tersebut dapat dipisahkan menjadirangkaian bit : 0 0 10 11 0 10 11 . Hal ini dapat dilakukan dengan mudah karena penyusunan rangkaian bit tersebut menggunakan kode awalan(*prefix code*). Setelah string biner tersebut dipisah menjadi rangkaian bitnya, hanya perlu dilihat rangkaian bit yang bersesuaian dengan tabel kode Huffman yang ditunjukkan pada tabel 2.3. Maka akan dihasilkan string hasil penguraian kode (*dekripsi*),yaitu "AABCABC".

2.2.2 Pengkodean Canonical Huffman

Canonical huffman adalah salah satu tipe dari huffman *code* dengan properti unik yang membuatnya dapat dideskripsikan dalam secara singkat. Pada awalnya sebuah data inputan akan dikodekan kedalam kode huffman kemudian kode tersebut akan disusun menjadi kode *canonical* huffman sebelum digunakan. Adapun aturan-aturan yang diterapkan dalam membentuk canonical huffman adalah :

- nilai kode dengan panjang bit terpendek nilainya lebih besar daripada nilai kode dengan panjang bit yang lebih panjang.
- kode yang memiliki panjang yang sama, nilai numeriknya akan naik seiring dengan nilai alfabet.

Aturan-aturan tersebut akan membuat proses decoding lebih

efisien(campos, 1999)

2.2.2.1 Algoritma Canonical Huffman

Pertama-tama yang dibutuhkan adalah kumpulan simbol dan panjang kode yang dihitung menggunakan algoritma huffman, kemudian kumpulan simbol tersebut akan dirubah menjadi bentuk canonical dengan pseudo code sebagai berikut :

```
compute huffman code:
 input:   message ensemble (set
of(message,probability)).base D.
 output:  code ensemble (set of (message, code)).
 algorithm:
   1. sort the message ensemble by decreasing
      probability.
   2. N is the cardinal of the message ensemble
      (number of different messages).
   3. compute the integer n_0 such as 2<=n_0<=D and
      (N-n_0)/(D-1) is integer.
   4. select the n_0 least probable messages, and
      assign them each a digit code.
   5. substitute the selected messages by a
      composite message summing their probability,
      and re-order it.
   6. while there remains more than one message, do
      steps thru 8.
      7. select D least probable messages, and
         assign them each a digit code.
      8. substitute the selected messages by a
         composite message summing their
         probability, and re-order it.
   9. the code of each message is given by the
      concatenation of thecode digits of the
      aggregate they've been put in.
```

2.2.2.2 Pengkodean Algoritma Canonical Huffman

Penggunaan algoritma huffman secara normal akan menghasilkan panjang code yang variatif untuk tiap alfabet yang ada. Penggunaan simbol alfabet yang banyak akan memperpendek panjang kode yang dihasilkan. Untuk lebih jelasnya dicontohkan dengan alfabet A, B, C dan D. Sebagai contoh, pada saat pengkodean huffman normal, simbol-simbol tersebut mendapat nilai seperti pada

gambar 2.2.

A	11
B	0
C	101
D	100

Gambar 2.2 contoh simbol Huffman

Pada gambar di atas huruf A memiliki panjang 2 bit, huruf B memiliki panjang 1 bit, huruf C dan D memiliki panjang 3 bit. Untuk membentuk kode *canonical* huffman maka kode-kode tersebut akan ditata ulang dengan diurutkan berdasarkan panjang kode. Gambar 2.3 menampilkan ilustrasi perubahan kode yang dihasilkan :

B	0	Menjadi	B	0
A	11		A	10
C	101		C	110
D	100		D	111

Gambar 2.3 ilustrasi perubahan kode huffman normal menjadi kode canonical huffman

2.3 Dokumen

Menurut kamus WordNet 3.1 dokumen dapat diartikan menjadi 4 hal, yaitu :

1. Dokumen tertulis yaitu sebuah tulisan yang memberikan informasi khususnya yang berhubungan dengan hal resmi.
2. Dokumen, apapun yang tersaji sebagai representasi dari pikiran manusia melalui rangkaian simbol-simbol.
3. Dokumen sebagai catatan dari kepemilikan sebuah benda.
4. Text file, dalam ilmu komputer dokumen dinyatakan sebagai sebuah file komputer yang didalamnya terkandung teks (kemungkinan mengandung instruksi pengaturan) menggunakan 7 bit karakter ASCII.

2.4 Analisis Capacity Ratio

Analisis *capacity ratio* dilakukan pada file yang telah mengalami perubahan. Analisis tersebut digunakan untuk mengetahui kapasitas *embedding* dari suatu *covertext.* Nilai ratio data

yang disembunyikan dapat dinyatakan pada persamaan 2.2 sebagai berikut (Shirali-Shahreza et al., 2008)

$$Capacity\ Ratio = \left(\frac{TextCapacity(bit)}{TextSize(Kilobyte)} \right) \qquad (2.2)$$

BAB III
METODE DAN PERANCANGAN

Bab ini akan memberikan penjelasan mengenai metode dan langkah-langkah perancangan yang dilakukan dalam penelitian yaitu untuk membuat sistem *text steganography* menggunakan metode perubahan sinonim pada pesan terenkripsi. Langkah-langkah yang dilakukan dalam penelitian adalah sebagai berikut :

1. Melakukan studi literatur mengenai algoritma *huffman* dan steganografi sebagai suatu algoritma untuk mengecilkan pesan teks dan menyelesaikan masalah steganografi. Studi literatur yang dilakukan baik dari buku, makalah, jurnal dan artikel-artikel terkait.
2. Menganalisis dan melakukan perancangan sistem steganografi teks meliputi proses input *covertext*, input *secret message*, pemrosesan input *secret message* menjadi format biner yang terkompresi, proses *embedding*, proses *extraction* dan perancangan *interface*.
3. Mengimplementasikan hasil analisis dan rancangan yang dilakukan sebelumnya menjadi sebuah sistem steganografi pada pesan terkompresi..
4. Melakukan uji coba terhadap perangkat lunak menggunakan dokumen tes. Hasil yang diperoleh dokumen berupa text yang di uji coba untuk proses *embedding* dan *extraction*.
5. Melakukan analisis yang berkaitan dengan manipulasi hasil steganografi, dan *ratio embedding.*

3.1 Analisis perangkat lunak

Pada subbab ini akan dibahas deskripsi dan batasan untuk perangkat lunak.

3.1.1 Deskripsi umum

Perangkat lunak ini berfungsi untuk melakukan proses *embedding* dan extraksi sebuah pesan rahasia yang terkompresi. Adapaun pesan rahasia yang dimasukkan kedalam sebuah *file text* merupakan pesan rahasia yang terkompresi, sehingga memaksimalkan jumlah pesan rahasia yang dapat dimasukkan kedalam *file text*. Metode perubahan sinonim diberlakukan kepada

kata yang dikenai oleh proses *embedding* untuk menyamarkan keberadaan steganografi. *Secret message* yang dimasukkan oleh user akan dirubah kedalam format biner. Format biner yang dihasilkan adalah hasil dari algoritma huffman dan kemudian di *embbed* kedalam *covertext*. Hal ini bertujuan untuk meningkatkan kapasitas *secret text* pada *covertext*. Kata-kata yang terpilih menjadi pembawa pesan rahasia pada *covertext* akan dicari sinonimnya dan kemudian disubtitusi pada *stegotext*. Hasil dari perangkat lunak ini berupa *stegotext* berupa file **.txt*. Secara umum, proses yang akan dilakukan *user* saat menggunakan perangkat lunak ini adalah sebagai berikut :

1. User memasukkan *covertext* yang akan digunakan sebagai media penyimpanan *secret message*. User akan memasukkan pesan tersembunyi lewat sebuat *textbox* yang sudah disediakan.
2. User memasukkan *secret message* yang akan disisipkan. Apabila *secret message* yang dihasilkan ternyata melebihi kapasitas yang bisa ditampung oleh *covertext* maka user akan mengulangi proses input dengan mempersingkat pesan rahasia yang akan dimasukkan atau mengganti*covertext* yang akan digunakan.
3. Untuk proses extraksi, user akan me-load *stegotext* dan *covertext* kemudian hasil extraksi akan ditampilkan di *textbox*.

Setelah semua dijalankan maka akan didapat sebuah *filestegotext* yang berisi pesan tersembunyi.

3.1.2 Batasan Perangkat Lunak

1. Media yang digunakan untuk penyimpanan pesan (covertext) adalah file teks berbahasa inggris dengan format *.txt. Pada penelitian ini digunakan penggalan *chapter* dari sebuah novel berbahasa inggris.
2. Pesan rahasia yang akan disisipkan berupa teks dan berfokus pada *ASCII Text*.
3. Baik pengirim pesan ataupun penerima pesan harus memiliki file wadah (*covertext*) yang asli, karena proses ekstraksi sangat bergantung pada *covertext*.
4. Untuk pesan rahasia yang terlalu singkat atau tidak layak dilakukan proses kompresi maka akan dimasukkan *string bit* apa adanya tanpa melalui proses kompresi. Hal ini digunakan untuk menjaga efisiensi.

3.2 Perancangan Perangkat Lunak

Setalah melakukan analisis dari deskripsi dan batasan sistem, yang dilakukan selanjutnya adalah melakukan perancangan terhadap sistem berdasarkan analisis yang telah dilakukan sebelumnya. Bagian ini menjelaskan secara rinci mengenai proses yang dilakukan sistem pengkategori secara berururtan dan sistematis. Macam-macam proses diantaranya : proses input *covertext*, input *secret message*, pemrosesan input *secret message* menjadi format biner menggunakan algoritma huffman, proses *embedding* , proses ekstraksi dan kemudian konversi dari kode biner menjadi bentuk *text*.

3.2.1 Proses Part-of-speech Tagging

Kebutuhan dasar dari algoritma ini adalah sebuah *covertext* yang harus dibagi antara pengirim dan penerima. Pemrosesan bahasa natural dilakukan pada *covertext* untuk menentukan bagian mana dari kata-kata yang ada yang mengandung *secret message*. Subtitusi sinonim akan dilakukan pada setiap *noun*(kata benda) dan *verb*(kata kerja). Dari semua *noun* dan *verb* yang ada pada *covertext*, sistem akan memeriksa berapa besar kapasitas bit yang sanggup ditampung tiap kata.Sistem akan menandai semua *noun* dan *verb* yang dimungkinkan untuk disisipkan *bit secret message* kedalamnya, adapun untuk mencegah agar tidak terjadi penumpukan *bit secret message* pada satu area *covertext*, sistem akan meng-*generate* penempatan kata yang akan disubtitusi agar merata.

3.2.2 Proses Input

Ada 2(dua) bagian input dalam perangkat lunak ini, yaitu input *covertext* dan *secret message*. Inputan *covertext* menggunakan file berisi artikel berbahasa inggris dengan ekstensi*.*txt* yang telah disediakan oleh user sebelumnya. Selanjutnya proses *checking* akan diberlakukan kepada *covertext* guna mengukur seberapa banyak bit yang bisa disimpan oleh *covertext*. User hanya bisa memasukkan pesan rahasia yang jumlahnya lebih sedikit atau sama dengan kapasistas bit yang bisa tampung oleh *covertext*. Apabila ternyata pesan yang dimasukkan terlalu panjang maka akan ada 2 opsi untuk *user*, yaitu mengganti pesan rahasia menjadi lebih pendek atau mengganti sumber *covertext* dengan kapasistas yang lebih besar.*Secret message* di-*input*-kan oleh *user* melalui *textbox* yang disediakan.

3.2.3 Proses Kompresi Pesan Rahasia

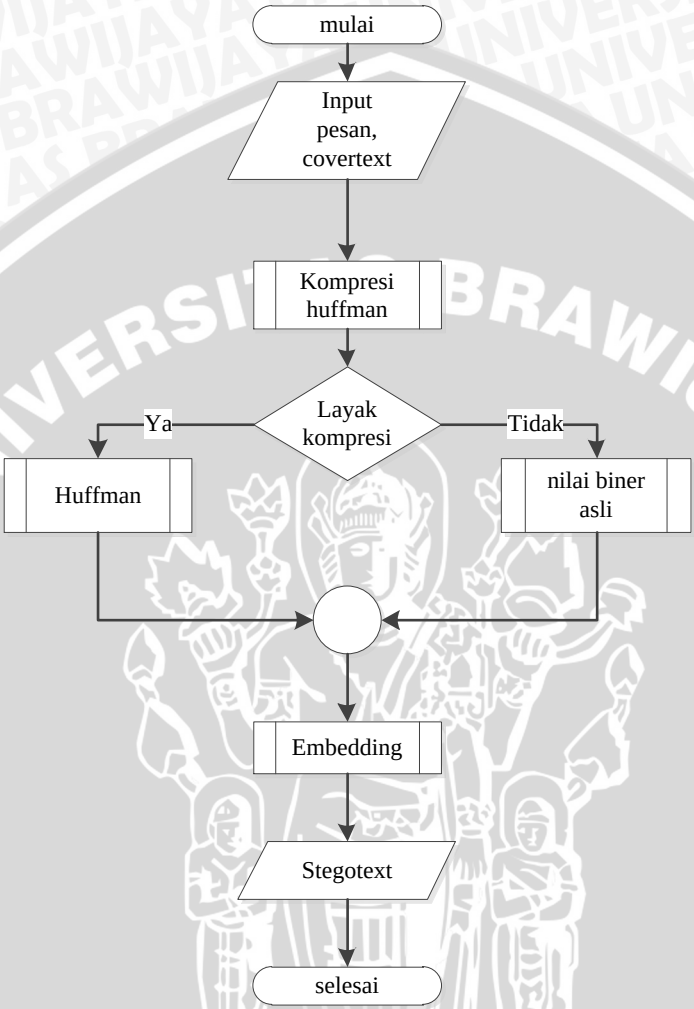
Pesan rahasia *(secret message)* yang dimasukkan user kedalam sistem akan mengalami proses kompresi, hal ini untuk menghemat jumlah bit yang akan dimasukkan kedalam *covertext*. Dalam bahasa inggris, banyak ditemukankata yang memiliki beberapa huruf yang sama dalam satu kata. Hal ini potensial untuk dimanfaatkan, dimana metode huffman sangat membantu untuk mengurangi jumlah bit yang akan dimasukkan. Adapun struktur data dari pesan yang dimasukkan kedalam *covertext* seperti pada gambar 3.1.

X	secret message	dictionary

Gambar 3.1 Struktur Data yang Disisipkan

"x" mewakili penanda awal pesan yang terdiri dari 1 bit yaitu sebuah *variable* yang menyatakan apakah pesan ini menerima proses kompresi huffman atau tidak, dimana bit 0 menyatakan tidak dan bit 1 menyatakan bahwa pesan ini menerima kompresi huffman. *Secret message* berisi rangkaian angka biner hasil dari kompresi huffman terhadap pesan rahasia yang dimasukkan oleh *user* ataupun hanya rangkaian angka biner hasil konversi dari pesan rahasia yang dinilai oleh sistem tidak layak untuk dilakukan kompresi. *Dictionary* adalah "kamus" yang digunakan oleh sistem untuk menyimpan kode-kode huffman yang dihasilkan, dalam proses dekompresi hanya kode-kode tersebut yang bisa diandalkan untuk memecah angka-angka biner menjadi sebuah *secret message*.

Selanjutnya pada gambar 3.2 akan digambarkan keseluruhan proses yang terjadi secara umum. Pesan yang dimasukkan oleh user akan diproses apakah layak untuk dikompresi mengingat pesan yang terlalu singkat tidak efisien untuk dikompresi. Pesan panjang akan dipendekkan dengan menggunakan metode huffman, sedangkan pesan yang terlalu pendek akan dikonversi langsung kedalam bentuk biner dan kemudian siap untuk di-*embed* kedalam cover text.



Gambar 3.2 *Flowchart* Proses Kompresi

3.2.4 Proses pergantian sinonim

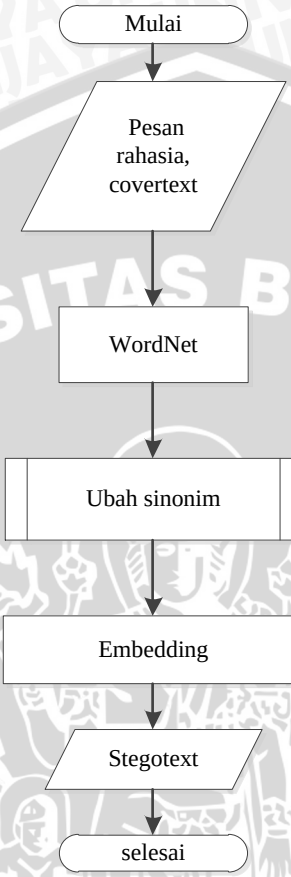
Bagian ini merupakan inti dari semua algoritma yang digunakan dalam perancangan perangkat lunak berupa steganografi pada *text*. Pada tahap ini inputan adalah bit string yang terkompresi dan *covertext* yang sudah diberi penanda oleh pengirim *stegotext* sedangkan pada pihak penerima pesan memerlukan *stegotext* dan *covertext*. Disinilah letak sebuah kamus sinonim diperlukan. Pada

penelitian ini digunakan kamus WordNet. Wordnet merupakan kamus *open source* yang berisi hampir semua kata bahasa inggris. Sinonim bisa didapat dari kata yang dipakai guna menyembunyikan *secrettext* dari kamus ini. WordNet menyediakan "*synsets*" atau *synonym sets* yang bisa dipilih berdasarkan kata mana yang paling "masuk akal" untuk merubah sebuah *covertext*. WordNet juga menyediakan informasi mengenai frekuensi penggunaan sebuah kata yang lazim digunakan dalam bahasa inggris, hal ini sangat membantu untuk memilih sinonim yang layak digunakan dalam stegotext.

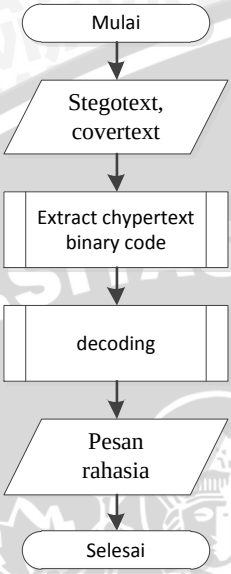
Pada gambar 3.3 ditunjukkan mekanisme yang dikerjakan pada pihak pengirim stegotext. Seperti yang bisa dilihat pada gambar, setiap *noun* pada cover text dipilih secara random. Kata-kata yang sudah terpilih tadi merupakan wadah dari string bit *secretmessage* yang akan di-*embbed* kedalam *covertext*. Setiap kata yang terpilih akan dicari sinonimnya melalui WordNet. Pada *synset* juga terdapat informasi mengenai frekuensi kemunculan kata yang lazim dalam struktur bahasa inggris. Dengan memperoleh frekuensi, bisa didapat kadar ambigu sebuah kata. Dengan memanfaatkan WordNet *dictionary*, dapat diperoleh sinonim yang cocok dan tidak jauh berbeda dengan makna yang sama.

Setelah membuat tabel encoding, kami menggunakan bit string dari secret message yang dijelaskan pada subbab 3.2.3 untuk memilih sinonim dari tabel. Isi dari tabel encoding merupakan kamus sinonim dari kata terpilih yang diterjemahkan kedalam kode biner. Panjang kode biner tergantung dari jumlah variasi sinonim yang dihasilkan. Apabila sebuah kata memiliki 4 sinonim maka kode biner yang dihasilkan hanya sepanjang 2 bit. Kode biner dari pesan rahasia yang telah dimasukkan oleh user akan digunakan sebagai pemilih sinonim apa yang akan digunakan.

Untuk proses ekstraksi pesan rahasia akan ditunjukkan pada gambar 3.4. Perangkat lunak memerlukan *covertext* dan *stegotext* untuk mencari kata mana saja yang menyimpan pesan rahasia dan kemudian akan memecah kode yang telah dimasukkan dalam *covertext* menjadi pesan rahasia yang ditujukan kepada penerima dari pengirim pesan.



Gambar 3.3 Flowchart Proses Embedding

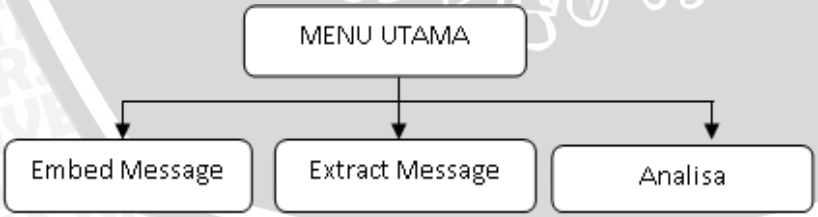


Gambar 3.4 Flowchart Proses Decoding

3.3 Perancangan Interface

Dalam penyajian informasi untuk user, program akan dibagi menjadi 3 bagian yaitu *embedding*, *extraction* dan analisa. Embedding berfungsi untuk penyisipan pesan rahasia (*secret text*) kedalam *covertext.* Bagian extraction berfungsi untuk melihat pesan dari *stegotext*. Sedangkan untuk bagian analisa berisi tentang informasi mengenai parameter pengujian yang dijelaskan pada subbab 3.4.

Pada gambar 3.5 digambarkan dalam bentuk diagram susunan desain interface aplikasi yang akan dihasilkan.

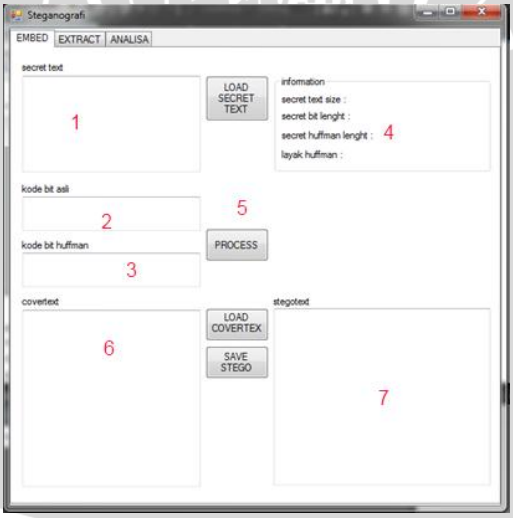


Gambar 3.5 Diagram menu system

User interface dalam program ini terbagi menjadi 3 tab yang mewakili 3 bagian yang telah disebutkan, yaitu :

1. Tab Embedding
   Pada bagian ini *user* bisa memasukkan sendiri pesan rahasia melalui *textbox* yang tersedia atau memuat file pesan yang telah disiapkan. Pada tab ini *user* juga bisa melihat metode apa yang dikenakan terhadap pesan rahasia yang akan di-*embed* kedalam *covertext* beserta output *stegotext* yang telah dihasilkan selama proses.
2. Tab Extraction
   Pada bagian ini *user* akan memuat file *stegotext* dan mengambil pesan rahasia dari file *stegotext*. Selain itu user juga dapat melihat kata-kata mana saja yang mengandung kode bit pesan rahasia beserta kode bit yang terkandung di dalamnya.
3. Tab Analisa
   Pada bagian ini *user* akan memuat file *stegotext* dan melakukan analisis untuk mengetahui nilai parameter-parameter yang di uji.

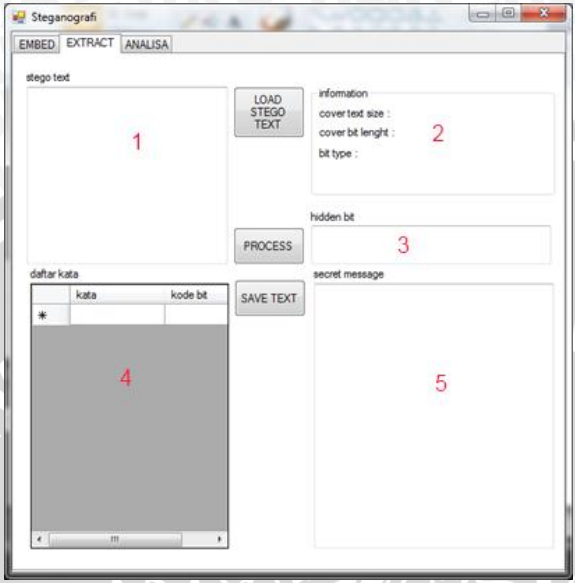
Rancangan *user interface* dapat dilihat pada gambar 3.6, 3.7, dan 3.8.



Gambar 3.6 *Interface* untuk embedding pesan

Rancangan *interface* untuk fungsi *hiden message* memiliki beberapa bagian yang akan dijelaskan sesuai urutan nomer yang tertera pada gambar 3.6. Dibawah ini adalah keterangan angka-angka pada gambar 3.6

1. *RichTextBox* yang berisi pesan rahasia. *User* dapat mengetikkan langsung pesan rahasia atau memuat sebuah *file* yang berisi pesan rahasia melalui *button* "*load secrettext*"
2. *RichTextBox* berisi kode bit asli dari pesan rahasia yang sudah di inputkan. *RichTextBox* akan terisi ketika *user* telah mengisi *RichTextBox* pesan rahasia dan menekan tombol "*proces*"
3. *RichTextBox* berisi kode bit huffman dari pesan rahasia yang sudah di inputkan. *Textbox* akan terisi ketika user telah mengisi *textbox* pesan rahasia dan menekan tombol "*proces*"
4. *Groupbox* berisi informasi mengenai besar(*byte*) pesan rahasia, panjang bit pesan rahasia, panjang bit huffman pesan rahasia dan apakah pesan rahasia yang telah dimasukkan layak untuk dikenakan kompresi huffman. *Groupbox* akan terisi ketika user telah mengisi *textbox* pesan rahasia dan menekan tombol "*proces*"
5. Tombol "*proces*" yang akan menginisiasi proses kompresi dan *embedding*. Tombol proses akan berfungsi ketika user telah memasukkan *secrettext* dan *covertext*.
6. Berisi *covertext* yang dimuat setelah user menekan tombol "*load covertext*"
7. *RichTextBox stegotext* akan terisi setelah tombol "*proces*" ditekan. User dapat menyimpan *stegotext* yang dihasilkan melalui tombol "*savestego*".

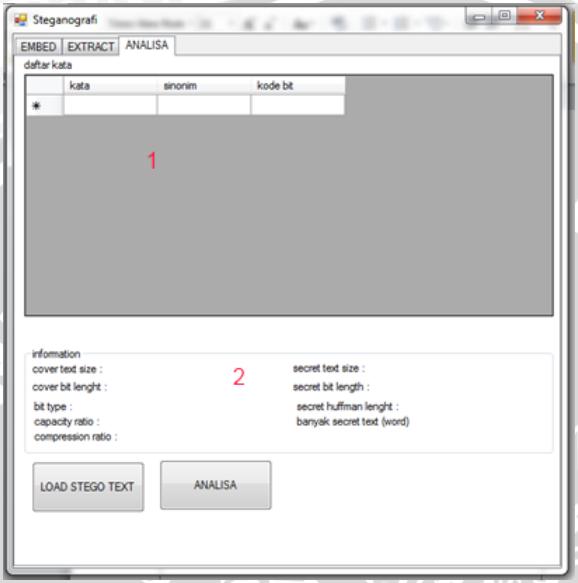


Gambar 3.7 *Interface* untuk extraksi pesan

Rancangan *interface* untuk fungsi *extract message* memiliki beberapa bagian yang akan dijelaskan sesuai urutan nomer yang tertera pada gambar 3.7. Dibawah ini adalah keterangan angka-angka pada gambar 3.7

1. *RichTextBox* yang berisi *stegotext* yang membawa pesan rahasia. *User* memuat sebuah *file stegotext* yang berisi pesan rahasia melalui *button* "*load stegotext*"
2. *Groupbox* berisi informasi mengenai besar *covertext*(byte), panjang bit pesan rahasia dan jenis pesan bit pesan rahasia. *Groupbox* akan terisi ketika *user* telah memuat *stegotext* dan menekan tombol "*proces*"
3. *RichTextBox* berisi kode bit dari pesan rahasia yang sudah di *extract*. *RichTextBox* akan terisi ketika user telah memuat*stegotext* dan menekan tombol "*proces*"
4. Tabel yang berisi daftar kata yang memuat kode bit rahasia beserta informasi bit rahasia yang ada didalamnya. Tabel akan terisi ketika user telah memuat *stegotext* dan menekan tombol "*proces*"
5. *RichTextBox* pesan rahasia berisi pesan rahasia hasil ekstraksi dari stegotext.*Textbox* akan terisi ketika user telah memuat

stegotext dan menekan tombol "*proces*". *User* juga bisa menyimpan pesan rahasia hasil ekstraksi melalui tombol "*savetext*"



Gambar 3.8 *Interface* untuk informasi hasil analisa

Rancangan *interface* untuk fungsi *analisa* memiliki beberapa bagian yang akan dijelaskan sesuai urutan nomer yang tertera pada gambar 3.8. Dibawah ini adalah keterangan angka-angka pada gambar 3.8

1. Tabel yang berisi kata asli, kata sinonim dan kode bit yang terkandung di dalam sinonim kata. *User* memuat sebuah *file stegotext* yang berisi pesan rahasia melalui *button* "*load stegotext*"
2. *Groupbox* berisi informasi mengenai informasi dari proses *embedding*, ekstraksi serta menampilkan informasi mengenai hasil analisa seperti *capacityratio*, *compressionratio* dan berapa banyak kata yang berhasil disisipkan kedalam *covertext*. *Groupbox* akan terisi ketika *user* telah memuat *stegotext* dan menekan tombol "analisa"

3.4 Perhitungan Manual proses Embedding dan Ekstraksi

Pada bagian ini akan diberikan contoh mengenai proses yang

terjadi di dalam sistem perangkat lunak yang dihasilkan oleh rancangan yang telah disebutkan sebelumnya.

3.4.1 Perhitungan Kompresi Pesan Rahasia (*secret text*)

Contoh *secret message* yang akan dimasukkan adalah kata "lulus". Kata tersebut akan dirubah menjadi kode huffman yang sudah dijelaskan pada subbab 2.2.1 dan menghasilkan bit string "**00000000-01101100-01110101-01101100-01110101-01110011**"
Panjang *cipherthext* dan *dictionary* adalah 48 bit, sedangkan panjang bit teks asli adalah 40bit. Dari perbandingan antara panjang text asli dan panjang *chipertext* maka dapat disimpulkan bahwa kata "lulus" tidak dapat dikatakan efisien dalam algoritma huffman yang diterapkan. Maka bit string yang digunakan sebagai inputan *covertext* adalah hasil konversi biner dari kata "lulus" itu sendiri, yaitu "01101100011101010110110001110101011100 11".

3.4.2 Perhitungan Embedding Pesan Rahasia

Proses ini diawali dengan mengolah *covertext* yang akan disisipi oleh pesan rahasia. Sistem akan mengolah *covertext* dan mendata sejumlah noun untuk dicari sinonim dan jumlah bit yang bisa ditampung. Contoh *covertext* :

EVER since I have been scrutinizing political **events**, I have taken a tremendous interest in propagandist **activity**. I saw that the Socialist-Marxist organizations**mastered** and **applied** this **instrument**with astounding **skill**. And I **soonrealized** that the **correct** use of propaganda is a **true** art which has **remainedpractically** unknown to the **bourgeois** parties. **Only** the Christian-Social movement, **especially** in Lueger's time, **achieved** a **certain**virtuosity on this instrument, to which it owed **many** of its **successes**.
But it was not until the War that it became **evident** what immense **results** could be **obtained** by a **correct** application of propaganda. **Hereagain**, unfortunately, all our **studying** had to be **done** on the enemy side, for the **activity** on our side was modest, to say the least. The total **miscarriage** of the German'**enlightenment** ' service stared every soldier in the face, and this spurred me to **take** up the **question** of propaganda even more deeply than before. There was often more than enough time for thinking, and the **enemy** offered practical **instruction** which, to our **sorrow**, was only

too good.

Kata-kata yang diberi cetak tebal pada *covertext* merupakan kata-kata potensial yang akan disisipi oleh pesan rahasia. Pemilihan letak penyisipan akan dilakukan secara *random* agar pesan tersebar merata dan tidak menumpuk pada awal artikel saja. Selanjutnya kata-kata yang sudah terpilih akan dicari sinonimnya. Dijabarkan pada tabel 3.1 adalah contoh tabel sinonim yang akan dihasilkan oleh aplikasi untuk digunakan kedalam proses selanjutnya.

Tabel 3.1 Contoh Tabel Sinonim

kata	**sinonim**	**Kode biner**
Events	Adventure	000
	Affair	001
	Cases	010
	Conclusion	100
	Action	101
	Activeness	110
	Diligence	111
Mastered	Winner	000
	Authority	001
	Chief	010
	Subdued	100
	Overthrow	101
	Ace	110
	Vanquish	111
applied	Used	00
	Utilized	01
	Employed	10
	Practiced	11
Instrument	Compose	00
	Equip	01
	Tool	10
	Address	11
skill	Ability	011
	Accomplishment	001
	Competence	010

	Craft	100
	Expertise	101
	Proficiency	110
Soon	Before long	00
	Shortly	01
	In a minute	10
	In the near future	11
realized	Actualized	11
	Appreciate	01
	Bring about	10
Correct	Earn	000
	recognized	001
	Realise	010
	Fix	100
	Right	101
	Proper	110
	Accurate	111
True	Absolute	000
	Actual	001
	Authentic	010
	Sincere	100
	Exact	101
	Correct	110
Remained	Abide	000
	Continue	001
	Actually	010
	Stayed	100
	Last	011
	Stick	110
Practically	Actually	00
	As good as	01
	In effect	10
	In reality	11
only	Alone	00
	Barely	01
	Lone	10
	Entirely	11

Especially	Specially	000
	Notably	001
	Personally	010
	Particulary	100
	Peculiarly	101
Achieved	Predessor	1
	Master	0
Certain	Positive	00
	Sure	01
	Well known	10
	Sureness	11
Many	Several	00
	Numerous	01
	A lot	10
	Large amount	11
Successes	Truimph	000
	Victory	001
	Virtue	110
	Winner	100
	Achievement	101
Evident	Unclear	11
	Obvious	01
	Noticeable	10
	Undeniable	00
Result	Consequence	00
	Lead	01
	Equaled	10
	Outcome	11
Obtained	Acquired	1
	Procured	1
Correct	Earn	000
	Realize	001
	recognized	010
	Fix	100
	Right	101
	Proper	110
	Accurate	111

Here	Over here	0
	This way	1
Again	Anew	00
	Once again	01
	Once more	10
	Over again	11
Studying	Considering	1
Done	Finish	00
	Complete	01
	Finished	10
	Performed	11
Activity	Life	00
	Action	01
	Diligence	10
	Activeness	11
Misscarriage	Stillbirth	1

Dari tabel 3.1 diperoleh padanan kata(sinonim) dari kata-kata potensial yang sudah di *generate* secara random oleh sistem. Selanjutnya sinonim tersebut akan diganti menjadi format biner yang kemudian dikombinasikan dengan *bit secrettext* untuk di-*embbed* kedalam *covertext*. Pada proses sebelumnya sudah diperoleh *secrettext* dan setelah diproses lebih lanjut akan menjadi bit string yang berisi 53 karakter"0-00110011-00110111-01101100-01110101-01101100-01110101-01110011". Proses *embbed* bit disesuaikan dengan kode bit pada tabel daftar sinonim.

Stegotext :

EVER since I have been scrutinizing political **adventures**, I have taken a tremendous interest in propagandist activity. I saw that the Socialist-Marxist organizations **ace**and **utilized**this **tool**with astounding **ability**. And I **shortlyactualized**that the **recognized**use of propaganda is a **right**art which has **stayedas good as**unknown to the bourgeois parties. **Entirely** the Christian-Social movement, **personally**in Lueger's time, **achieved**a **sure**virtuosity on this instrument, to which it owed **a lot**of its **virtue**.

But it was not until the War that it became **undeniable**what immense **outcome**could be **acquired**by a **recognized**application

of propaganda. **This wayonceagain**, unfortunately, all our **consideing**had to be **finished**on the enemy side, for the **action**on our side was modest, to say the least. The total **still birth**of the German 'enlightenment ' service stared every soldier in the face, and this spurred me to take up the question of propaganda even more deeply than before.

There was often more than enough time for thinking, and the enemy offered practical instruction which, to our sorrow, was only too good.

3.4.3 Perhitungan Ekstraksi Pesan Rahasia

Stegotext :

EVER since I have been scrutinizing political **adventures**, I have taken a tremendous interest in propagandist activity. I saw that the Socialist-Marxist organizations **ace**and **utilized**this **tool**with astounding **ability**. And I **shortlyactualized**that the **recognized**use of propaganda is a **right**art which has **stayedas good as**unknown to the bourgeois parties. **Entirely** the Christian-Social movement, **personally**in Lueger's time, **achieved**a **sure**virtuosity on this instrument, to which it owed **a lot**of its **virtue**.

But it was not until the War that it became **undeniable**what immense **outcome**could be **acquired**by a **recognized**application of propaganda. **This wayonceagain**, unfortunately, all our **consideing**had to be **finished**on the enemy side, for the **action**on our side was modest, to say the least. The total **still birth**of the German 'enlightenment ' service stared every soldier in the face, and this spurred me to take up the question of propaganda even more deeply than before.

There was often more than enough time for thinking, and the enemy offered practical instruction which, to our sorrow, was only too good.

Covertext :

EVER since I have been scrutinizing political **events**, I have taken a tremendous interest in propagandist **activity**. I saw that the Socialist-Marxist organizations **mastered** and **applied** this **instrument** with astounding **skill**. And I **soonrealized** that the **correct** use of propaganda is a **true** art which has

remainedpractically unknown to the **bourgeois** parties. **Only** the Christian-Social movement, **especially** in Lueger's time, **achieved** a **certain**virtuosity on this instrument, to which it owed **many** of its **successes**.
But it was not until the War that it became **evident** what immense **results** could be **obtained** by a **correct** application of propaganda. **Hereagain**, unfortunately, all our **studying** had to be **done** on the enemy side, for the **activity** on our side was modest,
to say the least. The total **miscarriage** of the German **'enlightenment** ' service stared every soldier in the face, and this spurred me to **take** up the **question** of propaganda even more deeply than before.
There was often more than enough time for thinking, and the **enemy** offered practical **instruction** which, to our **sorrow**, was only too good.

Dalam perangkat lunak ini, baik pihak pengirim dan penerima pesan harus memiliki dokumen asli yang belum terkena proses steganografi (*covertext*), hal ini digunakan untuk membandingkan perbedaan text yang terdapat pada kedua dokumen. Kedua dokumen akan dibandingkan dan diketahui letak kata-kata yang berbeda. Setelah ditemukan kata-kata yang berbeda tadi, kata-kata yang berbeda disusun dan dicari biner bit yang sudah disimpan didalamnya melalui tabel yang sama seperti proses *embedding*. Sistem penerima pesan akan meng-generate tabelnya sendiri melalui perbedaan yang ditemukan dalam kedua tabel. Setelah dilakukan proses decoding akan dihasilkan bit string "000110011001101110110110001110101011011000111010101110 011". Rangkaian bit ini akan di *decoding* lagi sesuai dengan struktur data yan dimasukkan seperti pada tabel 3.1. Setelah format struktur data terbaca maka proses perubahan bentuk biner menjadi text akan menghasilkan pesan "lulus" sebagai pesan rahasia yang dihasilkan.

3.5 Perancangan Uji Kinerja *TextSteganography*

Pengujian yang dilakukan pada sistem ini ditinjau dari seberapa banyak kapasistas yang bisa ditampung sebuah *covertext*(*capacity ratio*), seberapa banyak jumlah kata yang bisa dimasukkan dalam sebuah *filetext*, dan seberapa efisien penggunaan kompresi huffman untuk memperbanyak jumlah kata yang akan dimuat *stegotext*.

3.5.1 Uji Capacity Ratio

Pengujian selanjutnya adalah dengan *capacity ratio*. Dengan pengujian *capacity ratio* ini bisa diketahui sejauh mana suatu *covertext* dapat menampung sejumlah *secret message*. Semakin besar nilai *capacity ratio* maka semakin tinggi pula kapasitas penyimpanan *secret message* di dalam *covertext*. Pengujian akan dilakukan dengan atau tanpa metode huffman untuk mengetahui seberapa besar pengaruh dari kompresi pesan. Tabel pengujian *capacity ratio* dapat dilihat pada tabel 3.2.

Tabel 3.2 Tabel Uji *Capacity Ratio*

Covertext	Ukuran *File*(Kb)	*Capacity Ratio*(bit/Kb)

3.5.2 Uji efisiensi kompresi huffman

Pengujian selanjutnya adalah pengujian untuk mengetahui seberapa efisien algoritma huffman untuk mempersingkat panjang pesan yang akan disisipkan. Dengan semakin pendek pesan yang disisipkan maka semakin banyak pesan yang dapat dimasukkan kedalam covertext.Tabel pengujian efisiensi kompresi huffman dapat dilihat pada tabel 3.3.

Tabel 3.3 Tabel Uji Efisiensi Kompresi Huffman

Besar pesan rahasia (byte)	Panjang bit asli	Panjang bit huffman (termasuk tabel)	Compression ratio (termasuk tabel)	Banyak kata

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini akan dijelaskan mengenai implementasi sistam dan pembahasan terhadap hasil uji coba. Setelah dilakukan uji coba terhadap sistem selanjutnya dapat dilakukan evaluasi dan analisa terhadap sistem yang telah dibuat. Implementasi sistem adalah penerapan dari rancangan sistem yang telah dibahas pada bab sebelumnya.

Adapun perangkat lunak yang digunakan dalam pengembangan sistem steganografi ini adalah sebagai berikut :

1. Sistem operasi Microsoft Windows 7
2. Microsoft Visual Studio 2011 Beta

Perangkat keras yang digunakan dalam pengembangan sistem ini adalah laptop ASUS seri A43SJ dengan hardware sebagai berikut:

1. Proseor Intel i3 2.2Ghz
2. Memori 8GB
3. Harddisk 500GB

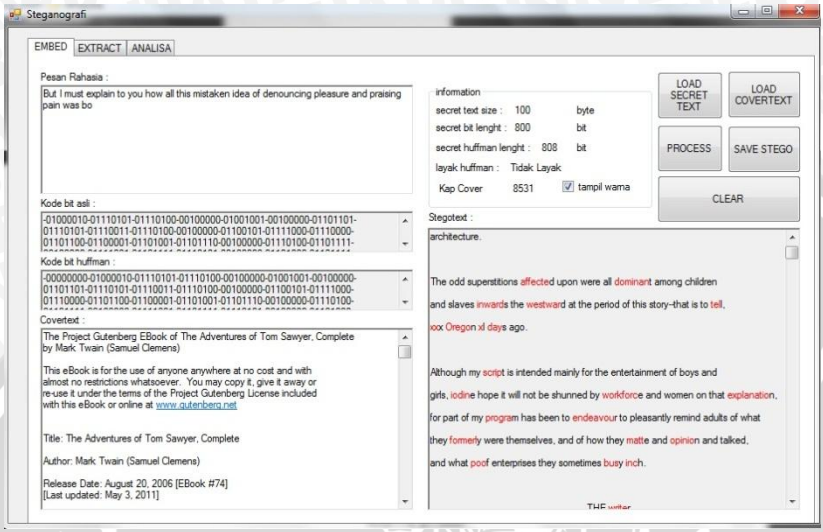
4.1 Implementasi Program

Sesuai dengan pembahasan pada BAB III, proses pada sistem ini terbagi menjadi dua bagian utama yaitu proses *embedding* dan *decoding*. Tujuan utama dari proses ini adalah memasukkan *secrettext* kedalam *covertext* yan sudah disiapkan sebelumnya proses ini diawali dengan me-load file *.txt yang merupakan *covertext*. Selanjutnya user diminta untuk memasukkan *secrettext*, berupa file *.txt.

Proses selanjutnya adalah melakukan proses kode huffman terhadap *secrettext* yang sudah dimasukkan. Pada proses Encode() akan dilakukan pengecekan apakah *secrettext* apakah *secrettext* tersebut layak dikenakan kompresi huffman. Jika diketahui bahwa pesan rahasia ternyata tidak layak dikenakan kompresi huffman maka bit asli dari pesan rahasia akan disisipkan kedalam *covertext*/

4.1.1 Implementasi Interface

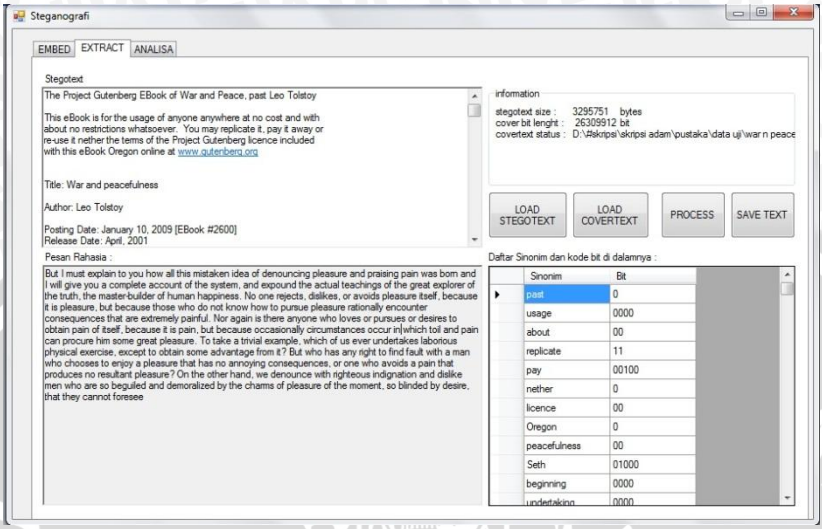
Berdasarkan perancangan *interface* yang telah dijelaskan pada subbab 3.3 maka dihasilkan suatu perangkat lunak yang mempunyai 3 tab yaitu melakukan *embedding*, ekstraksi dan analisa. Implementasi bisa dilhat pada gambar 4.1, 4.2, dan 4.3



Gambar 4.1 Form untuk embedding pesan

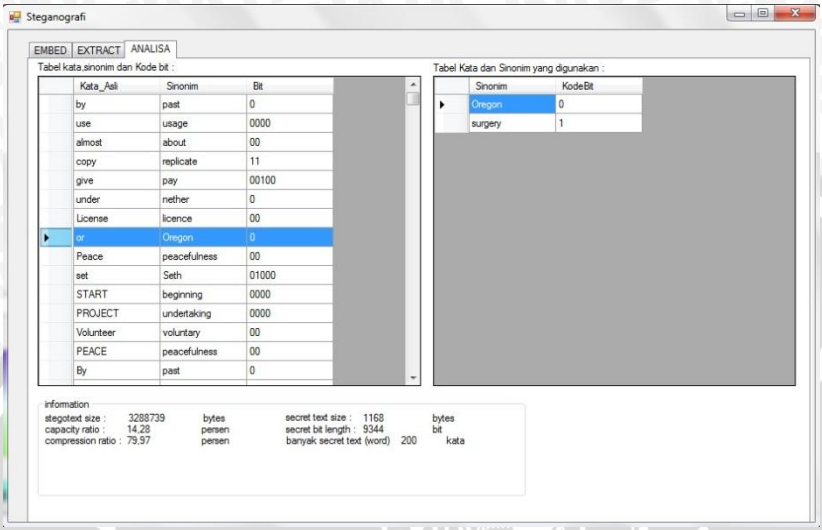
Pada gambar 4.1 merupakan *interface* untuk embedding pesan rahasia kedalam *covertext*. Dalam form tersebut terdapat 5 tombol yaitu load *secrettext*, *loadcovertext*, *process*, *savestego* dan *clear*. User dapat memasukkan secret text secara langsung pada richtextbox yang terdapat pada pojok kiri atas atau mengambil sebuah *filetext* lewat tombol load *secrettext*. Untuk mengambil *covertext* user dapat mengambilnya melalui tombol load covertex. Pada saat tombol process ditekan maka proses *embedding* akan terjadi dan menghasilkan tampilan *stegotext*. Pada panel *information* akan disebutkan data mengenai ukuran *secrettext*, panjang bit *secrettext*, panjang bit kode huffman dan keterangan *apakah* secret *text* layak atau tidak jika dikenakan kode huffman. Tombol *save stego* berfungsi untuk menyimpan *stegotext* kedalam file *.txt. sedangkan tombol *clear* berfungsi untuk menghilangkan semua isi *richtextbox* yang ada.

Richtextbox kode bit asli, kode bit huffman dan *stegotext* akan terisi secara otomatis ketika tombol process ditekan. *Richtexbox* tersebut tidak akan bisa di edit oleh pengguna.



Gambar 4.2 Form untuk ekstraksi pesan

Interface extrak pesan (gambar4.2) memiliki 4 tombol yaitu *open stegotext, load covertext, process dan save text*. Tombol *opens tegotext* berfungsi untuk memuat *stegotext* yang akan diekstrak pesannya. Tombol *load covertext* untuk memuat file *covertext*. Untuk mengekstrak pesan rahasia diperlukan *stegotext* dan *covertext*. *Covertext* yang digunakan juga harus sama persis seperti yang digunakan untuk menyisipkan pesan rahasia kedalam *stegotext*. Tombol *process* berguna untuk memulai proses ekstrak data yang mana akan menghasilkan pesan rahasia yang disisipkan kedalam stegotext. Tombol *process* juga akan menampilkan informasi mengenai pesan rahasia pada panel *information*. Informasi yang ditampilkan meliputi besar file *stegotext*, panjang kode bit, tipe kode bit apakah kode bit merupakan kode bit huffman atau tidak serta sumber *covertext* untuk melihat apakah *covertext* sudah dimuat. Tombol *savetext* berguna untuk menyimpan pesan rahasia yang sudah di ekstrak kedalam file teks. Dalam form ini ditampilkan *stegotext*, pesan rahasia dan bit rahasia yang ada pada *stegotext*.



Gambar 4.3 Form untuk menetahui informasi

Tombol analisa berfungsi untuk menghasilkan analisa tentang steganografi yang dipakai. Proses analisa menampilkan kata-kata yang dipakai, sinonim kata beserta kode bit yang terkandung di dalamnya. Pada panel *information* ditampilkan informasi mengenai *compression ratio* dan *capacityratio*. Proses analisa akan bekerja bila pada tab *extract* terisi *stegotext* dan *covertext*. Di panel informasi mengenai juga ditampilkan informasi *stegotext* dan *covertext* yang sebelumnya juga terdapat pada tab *embed* dan *extract*.

4.1.2 Proses Input File

Proses *input file* terdiri dari tiga bagian, proses *input file* untuk*secret text*, proses *input file covertext* dan*input file stegotext.* Proses *inputsecrettext* dapat dilakukan dengan dua cara yaitu dengan mengetikkan langsung pada saat program dijalankan atau dengan menekan tombol *load secret text*untukme-*load file* yang berekstensi *.txt.*Load file* untuk*secret text* dilakukan dengan menggunakan *openFileDialog1*.Pada saat load file juga terdapat fungsi yang menampilkan besar file dalam bytes pada panel information. Fungsi input *secret text* dapat dilihat pada *sourcecode* 4.1.

```
//load secret text dari file
        private void button1_Click(object sender, EventArgs e)
        {
```

```
            if (openFileDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
            {
                string text =
File.ReadAllText(openFileDialog1.FileName);
                richTextBox1.Text = text;
            }
            //mengetahui ukuran file secret text
            FileInfo f = new
FileInfo(openFileDialog1.FileName);
            long s1 = f.Length;
            labelInfoTextSize.Text = Convert.ToString(s1);
        }
```

Sourcecode 4.1 fungsi *open secret text*

Load file untuk *covertext* dilakukan dengan menggunakan *openFileDialog1*.Fungsi input *covertext* dapat dilihat pada *sourcecode* 4.2.

```
private void button2_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
            {
                string text =
File.ReadAllText(openFileDialog1.FileName);
                richTextBox2.Text = text;
            }
        }
```

Sourcecode 4.2 fungsi *open covertet*

Load file untuk *stegotext* dilakukan dengan menggunakan *openFileDialog1*.Pada saat load file juga terdapat fungsi yang menampilkan besar file dalam bytes pada panel information. Fungsi input *stegotext* dapat dilihat pada *sourcecode* 4.3.

```
//load isi stegotext
        private void button8_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
            {
                string text =
File.ReadAllText(openFileDialog1.FileName);
                richTextBox6.Text = text;
            }
            //mengetahui ukuran file secret text
            FileInfo f = new
FileInfo(openFileDialog1.FileName);
            long s2 = f.Length;
```

```
            labelInfoStegoLength.Text = Convert.ToString(s2);
        }
```

Sourcecode 4.3 fungsi *open stegotext*

4.1.3 Proses Konversi dari String ke Biner

Proses ini digunakan untuk melakukan konversi dari string ke dalam bentuk binernya masing-masing. Fungsi konversi dari string ke biner bisa dilihat pada sourcecode 4.4.

```
string str = richTextBox1.Text;
byte[] arr = System.Text.Encoding.ASCII.GetBytes(str);
richTextBox3.Text = arr.KeTeks();

 //mengetahui panjang bit secret text
int pjgArr = arr.Length;
labelInfoBitLenght.Text = Convert.ToString(pjgArr);
```

Sourcecode 4.4 fungsi konversi dari string ke biner

4.1.4 Proses Huffman

Proses ini diawali dengan fungsi pengecekan apakah secret text yang telah dimasukkan layak untuk dikenakan kompresi huffman. Pada awal proses pengecekan, secret text yang dimasukkan akan langsung dikompresi dengan metode huffman. Apabila ternyata dalam proses ini diketahui bahwa *secrettext* tidak layak untuk dikompresi maka biner asli dari *secrettext* akan dimasukkan kedalam *covertext*. Fungsi pengecekan *secrettext* bisa dilihat pada sourcecode 4.5 dan fungsi huffman bisa dilihat pada sourcode 4.6

```
public static Tuple<string, bool>[] Cover(this string
coverText, byte[] data)
        {
            var datalama = data.ToArray();
            data = data.Encode();
            if (!data.Decode().SequenceEqual(datalama))
            {
            }
            var pisahan = Regex.Split( Regex.Split(coverText,
pattern).KeTeks(),pattern);
            var pisahanBertanda = pisahan.Select(x => new
Tuple<string, bool>(x, false)).ToArray();
            var jumlahBit = pisahan.Select(x =>
x.Bit()).ToArray();
            var max = jumlahBit.Sum();
            var daftarKapasitas = pisahan.Select(x =>
x.Bit()).ToArray();
            var daftarBit = data.ToBoolArray();
            var bitSudahMasuk = 0;
            var bisaLompat = max > data.Length * 24;
```

```
            for (var i = 0; i <
pisahan.Length;i+=bisaLompat?acak.Next(1,3):1)
            {
                var yangBisaMasuk = jumlahBit[i];
                if (yangBisaMasuk+bitSudahMasuk >
data.Length*8)
                {
                    continue;
                }
                if (yangBisaMasuk == 0)
                {
                    continue;
                }
                var yangDiganti = pisahan[i];
                var yangDimasukkan = new bool[yangBisaMasuk];
                daftarBit.CopyTo(bitSudahMasuk, yangDimasukkan,
0, yangBisaMasuk);
                // misal data sisa cuma 110
                //kalau kapastias bisa masuk 4, yang terambil
tetap 110
                //terjemahannya berarti 0110
                var indeksSinonim =
yangDimasukkan.ToByteArray()[0];
                pisahan[i] =
pisahan[i].Syn().Skip(indeksSinonim).First();
                pisahanBertanda[i] = new Tuple<string,
bool>(pisahan[i], true);
                bitSudahMasuk += yangBisaMasuk;
                if (bitSudahMasuk == data.Length*8)
                {
                    break;
                }
            }
            return pisahanBertanda;
        }
```

Sourcecode 4.4 fungsi pengecekan *secrettext*

```
namespace StegoSyno
{

    public static class HuffmanHandler
    {
        public static byte[] Encode(this byte[] data)
        {
            var hasil = new Huffman(data).Kompresi();
            if (hasil.Length<data.Length)
            {
                return new[] { (byte)1
}.Concat(hasil).ToArray();
            }
            else
            {
                return new[] { (byte)0
```

```
}.Concat(data).ToArray();
            }

        }

        public static byte[] Decode(this byte[] data)
        {
            if (data[0]==0)
            {
                return data.Skip(1).ToArray();
            }
            return new
Huffman().Dekompresi(data.Skip(1).ToArray());
        }
    }

        /// <summary>
        /// Lakukan kompresi pada data yang masuk
        /// </summary>
        /// <returns>Array byte yang terkompresi</returns>
        public byte[] Kompresi()
        {

            var path = Path.GetTempFileName();

            var cacheOutput = new FileStream(path,
FileMode.Create);

            var TreeOptimal = TulisTree(cacheOutput);

            TulisDataTerkompresi(ref cacheOutput, TreeOptimal);

            cacheOutput.Flush(true);

            cacheOutput.Close();
            cacheOutput.Dispose();
            var hasil = File.ReadAllBytes(path);

            File.Delete(path);

            return hasil;
        }

        private void TulisDataTerkompresi(ref FileStream
cacheOutput, List<Cabang> TreeOptimal)
        {
            //ukuran data sebelum didekompresi
            var ukuran = (int)cacheInput.Length;
            var data = new byte[ukuran];
            //posisi dalam stream input ebelum mulai menulis
            var sebelum = cacheInput.Position;

            cacheInput.Position = 0;
            cacheInput.Read(data, 0, ukuran);
            cacheInput.Position = sebelum;
```

```
            //hitung sisa data yang tidak memenuhi sebuah blok
            var sisa = data.Length % UkuranBlok;
            //hitung jumlah blok yang diperlukan
            var jumlahBlok = data.Length / UkuranBlok + (sisa >
0 ? 1 : 0);

            //membuat array temporer, yang disusun dengan
memanggil secara paralel
            //kompresi untuk tiap blok
            var arrayTemporer = new MemoryStream[jumlahBlok];
            Parallel.For(0, jumlahBlok, new ParallelOptions() {
MaxDegreeOfParallelism = jumlahParalel }, blokKe =>
            {
                var posisiBlok = blokKe * UkuranBlok;
                arrayTemporer[blokKe] = TulisBlok(TreeOptimal,
data, UkuranBlok, posisiBlok);
            });
            //tulis data dalam arrayTemporer ke stream output
            foreach (var temporer in arrayTemporer)
            {
                temporer.Position = 0;
                temporer.CopyTo(cacheOutput);
                //temporer.WriteTo(cacheOutput);
                //cacheOutput.Write(temporer.ToArray(), 0,
(int)temporer.Length);
            }

            //kompresi secara serial

        private List<Cabang> TulisTree(FileStream
msEncodedOutput)
        {

            BangunPohon();


            SusunTreeDariTiapCabang();

            //buang cabang pertengahan yang sudah tidak
diperlukan
            var TreeOptimal = pohonHuffman.Where(cabang =>
!cabang.Pertengahan).ToList();


            long buffer;
            var jumlahByteTertulis = 0;
            MemoryStream awalanTree = new MemoryStream();
            foreach (Cabang l in TreeOptimal)
            {

                if (!l.Pertengahan & l.Frekuensi > 0)
                {
                    buffer = l.NilaiByte;
                    buffer <<= 8;
```

```
                    buffer ^= l.JumlahBit;
                    buffer <<= 48;
                    buffer ^= l.NilaiBit;

awalanTree.Write(BitConverter.GetBytes(buffer), 0, 8);
                    jumlahByteTertulis++;
                    buffer = 0;
                }
            }

            //tulis ukuran tree

msEncodedOutput.Write(BitConverter.GetBytes(cacheInput.Length),
0, 8);

            //tulis ukuran data
            msEncodedOutput.WriteByte((byte)(jumlahByteTertulis
- 1));

            //tulis data
            msEncodedOutput.Write(awalanTree.ToArray(), 0,
Convert.ToInt16(awalanTree.Length));

            //tambahi dengan 3 null untuk memberi ruang bagi
data kompresi

            msEncodedOutput.Write(new byte[] { 66, 67, 68 }, 0,
3);
            return TreeOptimal;
        }



        public byte[] Dekompresi(byte[] bInput)
        {

            List<Cabang> Tree = new List<Cabang>(255);
            long buffer = 0;
            long panjangStream = 0;
            int jumlahEntryTree = 0;
            int posisiAkhirDataTree = 0;

            //baca tree
            byte[] bDecodedOutput = BacaTree(bInput, Tree, ref
buffer, ref panjangStream, ref jumlahEntryTree, ref
posisiAkhirDataTree);

            //buang cabang dengan nilai identik
            var key = Tree.ToDictionary(x => x, x => x.NilaiBit
* 100000 + x.JumlahBit);
            Tree = key.Values.Distinct().Select(x =>
key.First(y => y.Value == x).Key).ToList();


            var DTree = Tree.ToDictionary(x => x.NilaiBit *
```

```
100000 + x.JumlahBit, x => x.NilaiByte);

            IndexBitValue = Tree.Select(x =>
x.NilaiBit).ToList();


            buffer = 0;

            var offsets = CariOffset(bInput,
posisiAkhirDataTree).ToList();

            //panggil perintah dekompresi secara paralel
            if (JumlahParalel > 1)
            {
                Parallel.For(0, offsets.Count, new
ParallelOptions() { MaxDegreeOfParallelism = JumlahParalel }, i
=>
                {
                    DekompresiBlok((int)offsets[i], bInput,
DTree, posisiAkhirDataTree, UkuranBlok * i, bDecodedOutput);
                });
            }
            //panggil perintah dekompresi secara singgle
            else
            {
                for (var i = 0; i < offsets.Count; i++)
                {
                    DekompresiBlok((int)offsets[i], bInput,
DTree, posisiAkhirDataTree, UkuranBlok * i, bDecodedOutput);
                }
            }

            return bDecodedOutput;
        }

        /// <summary>
        /// Baca tree dari data yang terkompresi
        /// </summary>
        /// <param name="bufferInput"></param>
        /// <param name="Tree"></param>
        /// <param name="buffer"></param>
        /// <param name="panjangData"></param>
        /// <param name="jumlahTree"></param>
        /// <param name="posisiAkhirDataTree"></param>
        /// <returns></returns>
        private static byte[] BacaTree(byte[] bufferInput,
List<Cabang> Tree, ref long buffer, ref long panjangData, ref
int jumlahTree, ref int posisiAkhirDataTree)
        {
            //buat tree dengan 256 cabang
            for (int i = 0; i < 256; i++)
            {
                Tree.Add(new Cabang());
            }
```

```
            panjangData = BitConverter.ToInt64(bufferInput, 0);

            byte[] bDecodedOutput = new byte[panjangData];

            jumlahTree = bufferInput[8];

            posisiAkhirDataTree = (((jumlahTree + 1) * 8) + 8);

            //mulai menerjemahkan tree
            for (int i = 9; i <= posisiAkhirDataTree; i += 8)
            {
                buffer = BitConverter.ToInt64(bufferInput, i);

                var cabang = new Cabang();

                cabang.NilaiByte = (byte)(buffer >> 56);
                if (cabang.NilaiByte != 0) buffer ^=
(((Int64)cabang.NilaiByte) << 56);

                cabang.JumlahBit = (int)(buffer >> 48);
                buffer ^= (((Int64)cabang.JumlahBit) << 48);

                cabang.NilaiBit = (int)(buffer);

                Tree[cabang.NilaiByte] = cabang;
            }
            return bDecodedOutput;
        }
```

Sourcecode 4.5 kode huffman

4.1.5 Proses Embedding

Proses ini merupakan proses inti dari sistem. Pada saat dilakukan proses embedding, hal pertama yang dilakukan adalah memetakan kata-kata yang berpotensi untuk diganti sinonimnya.. *Sourcecode tagging word* bisa dilihat para sourcecode 4.6

```
for (var i = 0; i <
pisahan.Length;i+=bisaLompat?acak.Next(1,3):1)
            {
                var yangBisaMasuk = jumlahBit[i];
                if (yangBisaMasuk+bitSudahMasuk >
data.Length*8)
                {
                    continue;
```

```
                }
                if (yangBisaMasuk == 0)
                {
                    continue;
                }
                var yangDiganti = pisahan[i];
                var yangDimasukkan = new bool[yangBisaMasuk];

                daftarBit.CopyTo(bitSudahMasuk, yangDimasukkan,
0, yangBisaMasuk);
                // misal data sisa cuma 110
                //kalau kapastias bisa masuk 4, yang terambil
tetap 110
                //terjemahannya berarti 0110
                var indeksSinonim =
yangDimasukkan.ToByteArray()[0];
                pisahan[i] =
pisahan[i].Syn().Skip(indeksSinonim).First();
                pisahanBertanda[i] = new Tuple<string,
bool>(pisahan[i], true);
                bitSudahMasuk += yangBisaMasuk;
                if (bitSudahMasuk == data.Length*8)
                {
                    break;
                }
            }
```

Sourcecode 4.6 tagging word

4.1.6 Proses penyimpanan

Fungsi ini digunakan untuk menyimpan hasil *embedding* dan ekstraksi kedalam file dengan format *.txt. berikut adalah *sourcecode* dari salah satu fungsi simpan file yang ada. *Sourcecode* penyimpanan *stegotext* bisa dilihat pada *sourcecode* 4.7.

```
private void button4_Click(object sender, EventArgs e)
        {
            // Create a SaveFileDialog to request a path and
file name to save to.
            SaveFileDialog saveFile1 = new SaveFileDialog();

            // Initialize the SaveFileDialog to specify the RTF
extension for the file.
            saveFile1.DefaultExt = "*.txt";
            saveFile1.Filter = "Text Files|*.txt";
            // Determine if the user selected a file name from
the saveFileDialog.
            if (saveFile1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK &&
               saveFile1.FileName.Length > 0)
            {
                // Save the contents of the RichTextBox into
the file.
                File.WriteAllText(saveFile1.FileName,
```

```
textStego);
            }
        }
```

Sourcecode 4.7 fungsi *simpan pesan*

4.1.7 Proses ekstraksi

Untuk poroses ekstraksi user akan membuka file *stegotext* dan *covertext* dari suatu file tertentu, kemudian kata-kata dalam *stegotext* dicari apakah ada yang berbeda dari *covertext*. Kata-kata yang sudah terpilih karena berbeda akan dicari kode bit yang terkandung didalamnya dan kemudian kode bit tersebut akan diterjemahkan menjadi karakter ASCII. *Sourcecode* untuk mencari perbedaan kata akan ditampilkan pada *sourcecode* 4.8 dan *sourcecode* untuk mengubah bit menjadi ASCII akan ditampilkan pada *sourcecode* 4.9

```
public static string Decover(this string stego, string
teksCover)
        {
            var pisahanStego = Regex.Split(stego, pattern);
            var pisahanCover =
Regex.Split(Regex.Split(teksCover, pattern).KeTeks(),pattern);

            var yangBerbeda = Enumerable.Range(0,
pisahanStego.Length).Where(x => pisahanStego[x].ToLower() !=
pisahanCover[x].ToLower());

            var arrayBit = new List<bool>();


            foreach (var posisi in yangBerbeda)
            {
                var indeksSinonim =
pisahanCover[posisi].Syn().ToList().IndexOf(pisahanStego[posisi
]);
                var biner = Convert.ToString(indeksSinonim,
2).PadLeft(8,'0').Skip(8-
pisahanCover[posisi].Bit()).Select(x=>x=='1').ToList();
                arrayBit.AddRange(biner.Reverse<bool>());
            }

            var totalBitTersedia = pisahanCover.Sum(x =>
x.Bit());

            var arrayByte = arrayBit.ToArray().ToByteArray();

            var dekompres = arrayByte.Decode();

            var banding =
ASCIIEncoding.ASCII.GetBytes(ASCIIEncoding.ASCII.GetString(deko
mpres));
```

```
            return ASCIIEncoding.ASCII.GetString(dekompres);
        }
```

Source code 4.8 Fungsi mencari perbedaan kata

```
public static string KeTeks(this byte[] arr)
        {
            return arr.Aggregate("", (ak, b) => ak + "-" +
Convert.ToString(b, 2).PadLeft(8, '0'));

        }
```

Sourcecode 4.9 Fungsi untuk mengubah bit menjadi karakter ASCII

4.2 Hasil Uji

Pada tabel 4.1, 4.2 dan 4.3 diperlihatkan hasil pengujian terhadap *capacity ratio* dan *compression ratio*. File yang digunakan dalam percobaan ini berupa file dengan ekstensi *.txt. File yang digunakan sebagai *covertext* adalah sebuah novel berjudul "The adventures of Tom Sawyer" dan "War and Peace".

Dilakukan dua percobaan secara terpisah, hal ini dilakukan untuk menguji efisiensi algoritma huffman untuk memperbanyak jumlah byte yang akan disisipkan kedalam *covertext*.

Percobaan pertama :

- Menggunakan *covertext* dari novel "War and Peace" karya Leo Tolstoy
- Digunakan dummy text dari potongan novel *De finibus bonorum et malorum (On the ends of good and evil)* yang merupakan karya filosof berkebangsaan romawi yaitu Marcus Tullius Cicero.
- Potongan novel dihitung berdasarkan jumlah kata yang ada didalamnya, yaitu 200,500,750,1000 dan 1500 kata.

Percobaan kedua :

- Menggunakan covertext dari novel "The Adventures of Tom Sawyer" karya Mark Twain (Samuel Clemens)
- Digunakan dummy text dari potongan novel *De finibus bonorum et malorum (On the ends of good and evil)* yang merupakan karya filosofis berkebangsaan romawi yaitu Marcus Tullius Cicero.

- Potongan novel dihitung berdasarkan jumlah karakter yang ada didalamnya, yaitu 100, 300, 500, 600, 700, 800, 900, 1000, 1100, 1200, dan 1300 karakter.

File yang digunakan sebagai secret message/pesan rahasia dihasilkan oleh *dummy text generator* yang berisikan potongan cerita dari penulis bernama cicero, file dihasilkan dari sebuah website dengan alamat *www.blindtextgenerator.com/cicero-en*. Tabel 4.1 adalah tabel hasil pengujian *capacity ratio* dan tabel 4.2 dan 4.3 adalah tabel pengujian *compression ratio*.

Tabel 4.1 Tabel Uji *Capacity Ratio*

Covertext	Ukuran *File*(Kb)	*Capacity Ratio*(bit/Kb)
the adventures of Tom Sawyer	407	16.40
War and Peace	3.212	14.28

Pada tabel 4.1 terlihat perbedaan *capacity ratio* antara kedua novel yang digunakan. Semakin besar ukuran *covertext* maka semakin besar pula *kapisitas* yang dapat ditampung. Hasil ini dapat bervariasi tergantung dari pemilihan *covertext* dan jumlah perbendaharaan kata pada kamus. Semakin banyak *covertext* dan jumlah kata pada kamus maka semakin tinggi pula *capacity ratio* yang dimiliki. Algoritma pemilihan kata yang akan diganti sinonimnya juga berpengaruh pada *capacity ratio*.

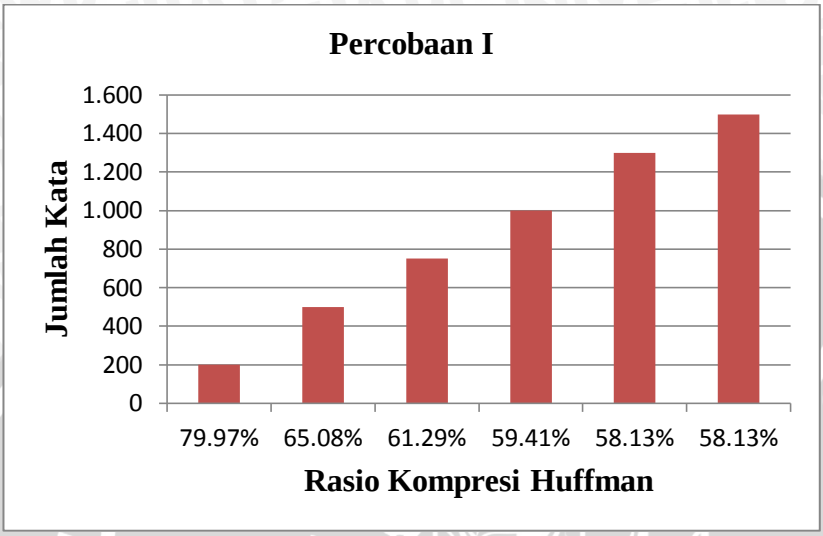
Tabel 4.2 Tabel Uji Efisiensi Huffman untuk percobaan I

Besar pesan rahasia (byte)	Panjang bit asli	Panjang bit huffman (termasuk tabel)	Compression ratio (termasuk tabel)	Banyak kata
1.168	9.344	7.472	79,97%	200

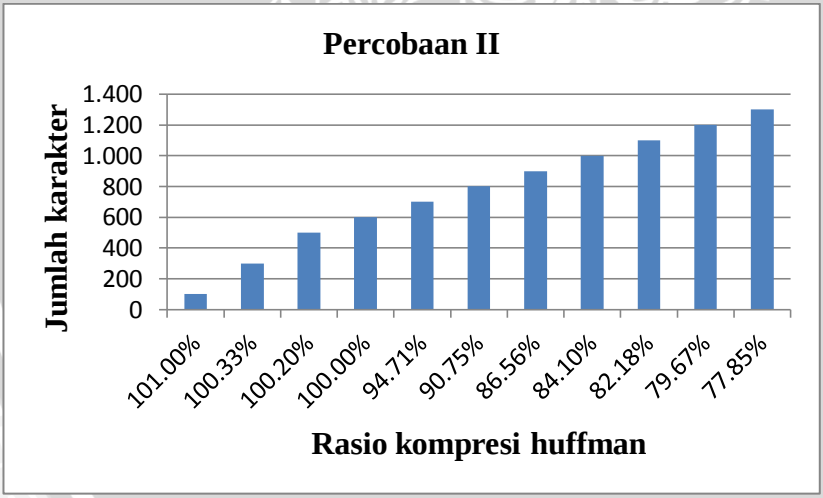
2.901	23.208	15.104	65,08%	500
4.355	34.840	21.352	61,29%	750
5.800	46.400	27.568	59,41%	1.000
7.533	60.264	35.032	58,13%	1.500
7.533	60.264	35.032	58,13%	1.500

Tabel 4.3 Tabel Uji Efisiensi Huffman untuk percobaan II

Besar pesan rahasia (byte)	Panjang bit asli	Panjang bit huffman (termasuk tabel)	Compression ratio (termasuk tabel)	Banyak karakter
100	800	808	101,00%	100
300	2.400	2.408	100,33%	300
499	3.992	4.000	100,20%	500
600	4.800	4.800	100,00%	600
700	5.600	5.304	94,71%	700
800	6.400	5.808	90,75%	800
900	7.200	6.232	86,56%	900
1.000	8.000	6.728	84,10%	1.000
1.100	8.800	7.232	82,18%	1.100
1.200	9.600	7.648	79,67%	1.200
1.300	10.400	8.531	77,85%	1.300



Gambar 4.4 Grafik compression ratio dan banyak kata untuk percobaan I



Gambar 4.5 Grafik compression ratio dan banyak kata untuk percobaan II

Dari grafik 4.4 yang merujuk pada percobaan pertama dapat disimpulkan bahwa semakin besar jumlah kata yang disisipkan akan memperkecil panjang bit yang akan disisipkan. Hal ini dikarenakan algoritma kompresi huffman akan memperpendek jumlah bit yang akan disisipkan. Dalam percobaan yang dilakukan, *covertext* hanya mampu menampung hingga 1300 kata sehingga pada uji untuk memasukkan 1500 kata terdapat 200 kata yang terpotong. Dapat disimpulkan juga bahwa untuk percobaan ini angka maksimum untuk rasio kompresi huffman adalah 58,13% untuk *covertext* berukuran 3.212KB dan *secrettext* 7.533Byte.

Dari grafik 4.5 yang merujuk pada percobaan kedua dapat disimpulkan bahwa semakin besar jumlah karakter yang disisipkan akan memperkecil panjang bit yang akan disisipkan. Hal ini dikarenakan algoritma kompresi huffman akan memperpendek jumlah bit yang akan disisipkan. Dalam percobaan yang dilakukan, jumlah karakter pada *secrettext* yang akan mempengaruhi besarnya rasio kompresi huffman. Mengingat tidak semua *secrettext* layak untuk diberlakukan kompresi huffman. Pada grafik 4.5 dapat dilihat bahwa untuk jumlah karakter antara 100 hingga 600 karakter tidak layak dikenakan kompresi huffman karena jumlah bit huffman yang dihasilkan lebih panjang daripada bit asli dari *secrettext*. Dalam percobaan kedua ini juga dapat disimpulkan bahwa untuk menyisipkan *secrettext* dengan jumlah karakter kurang atau samadengan 600 karakter(kurang atau samadengan 600Byte) tidak layak dikenakan kompresi huffman. Untuk file *secrettext* dengan jumlah karakter(Byte) lebih dari 600 kompresi huffman akan meningkatkan jumlah karakter yang dapat disisipkan kedalam *covertext*.

4.3 Analisis Hasil

Pada subbab ini akan dibahas mengenai hasil analisis dengan data seperti yang disebutkan pada subbab 4.2. Pada uji *capacity ratio* dapat dilihat bahwa ukuran *capacity ratio* untuk *covertext* bervariasi antara 16,40 bit/kB hingga 14,28 bit/kB. Besar kecilnya *capacity ratio* tergantung pada besar file *covertext* dan perbendaharaan kata yang digunakan pada kamus WordNet. Semakin besar *file* dan semakin banyak persamaan kata yang ada pada kamus maka jumlah *capacity ratio* akan meningkat. Dalam percobaan yang dilakukan ternyata pada novel “War and Peace” perbendaharaan kata pada

kamus tidak banyak kemiripan sehingga meskipun novel tersebut memiliki ukuran yang besar, capacity ratio tidak meningkat.

Untuk studi kasus pada percobaan I dan II dengan melihat data pada tabel 4.2 dan 4.3 dapat disimpulkan bahwa dengan *capacity ratio* yang sama, jumlah *secrettext* yang dapat disisipkan dapat ditingkatkan menggunakan algoritma huffman. Pada percobaan I nilai rasio kompresi huffman mencapai titik maksimal di 58,13%. Hal ini berarti adanya kompresi huffman pada percobaan pertama sanggup meningkatkan efisiensi penyisipan secrettext kedalam *covertext* sebesar 41,87% untuk data sebesar 7.533Byte. Pada perobaan kedua juga dapat disimpulkan bahwa agar algoritma huffman dapat bekerja lebih efisien, jumlah karakter pada *secret text* setidaknya lebih dari 600 karakter atau 600Byte.

Dengan ini dapat disimpulkan bahwa besar file pada *covertext* belum tentu meningkatkan *capacity ratio* untuk proses steganografi dan agar algoritma huffman dapat bekerja secara efisien, besar data pada *secrettext* harus melebihi 600 karakter atau 600Byte.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari data pengujian pada subbab 4.2 dapat ditarik beberapa kesimpulan, yaitu :

1. Telah diimplementasikan sebuah perangkat lunak *text steganography* yang merupakan penggabungan dari metode perubahan sinonim dan kompresi huffman.
2. Pengujian *capacity ratio* dapat dilihat bahwa ukuran *capacity ratio* untuk *covertext* bervariasi antara 16,40 bit/kB hingga 14,28 bit/kB. Dapat disimpulkan bahwa nilai rata-rata capacity ratio untuk penelitian ini adalah 15,34 bit/kB. Hasil ini jauh lebih rendah daripada metode *syntatical steganography* yang diusulkan oleh Shirali Shahreza yang sanggup memperoleh nilai *capacity ratio* sebesar 21,99 bit/kB.
3. Kompresi huffman pada percobaan pertama sanggup meningkatkan rasio penyisipan *secrettext* kedalam *covertext* sebesar 41,87% untuk data sebesar 7.533Byte. Pada perobaan kedua juga dapat disimpulkan bahwa agar algoritma huffman dapat bekerja lebih efisien, jumlah karakter pada *secret text* setidaknya lebih dari 600 karakter atau 600Byte. Pada pengujian ini dapat diambil nilai rata-rata efisiensi kompresi huffman sebesar 36,33%. Hasil ini memiliki perbedaan yang relatif sedikit dengan penelitian kompresi huffman oleh Nanhe pada tahun 2008 yang memiliki nilai efiesiensi rata-rata sebesar 33%.

Dari poin-poin di atas dapat ditarik kesimpulan bahwa pada implementasi gabungan antara kompresi huffman dan metode perubahan sinonim pada dokumen dapat meningkatkan jumlah pesan rahasia dalam dokumen dengan daya tampung *covertext* yang relatif kecil.

5.2 Saran

Adapun saran yang dapat menjadi pertimbangan adalah sebagai berikut :

1. Untuk meningkatkan nilai *capacity ratio* dapat digunakan kamus lain yang memiliki perbendaharaan kata lebih luas.

2. Dapat digunakan varian kompresi huffman yang lain untuk meminimalisir jumlah bit yang dimasukkan kedalam *covertext*.

DAFTAR PUSTAKA

Augusta. 2012. Pengertian Steganografi. (Online). (http://infoini.com/2012/pengertian-steganografi.html, diakses 25 januari 2012)

Bender, Gruhl, Morimoto, and Lu. 1996. *Techniques for data hiding*. IBM Systems Journal. Vol. 35. Issues 3&4. pp. 313-336

Campos, Arturo. 1999. *Canonical Huffman*. (Online). (http://www.arturocampos.com/ac_canonical_huffman.html, diakses 30 january 2012)

Irwan Kurniawan. 2010. *Penyandian (Encoding) dan Penguraian Sandi (Decoding) Menggunakan Huffman Coding*. Institut Teknologi Bandung.

Nanhe, Aniket M., et al. 2008. *Improved Synonym Approach to Linguistic Steganography*. B.Tech Computer Science, College of Enginering, Pune.

Sarifuddin Madenda., et al. 2010. *Kompresi Citra Berwarna Menggunakan Metode Pohon Biner Huffman*. Lab. Pengolahan Citra dan Multimedia. Universitas Gunadarma.

Sellars, Duncan. 1996. *An Introduction to Steganography*.(Online). (http://www.totse2.com/totse/en/privacy/encryption/163947.html, diakses 26 januari 2012)

Shirali-Shahreza, M.H and Shirali-Shahreza, M. 2008. *A New Synonym Text Steganography*. International Conference on Intelligent Information Hiding and Multimedia Signal Processing 978-0-7695-3278-3/08. IEEE.

Shirali-Shahreza, M. H and Shirali-Shahreza, M. 2005. *Steganography in Persian and Arabic Unicode Texts Using Pseudo-Space and Pseudo Connection Characters*. JATIT. Iran.

Shirali-Shahreza, M. 2008. *Text Steganography by Changing Words Spelling*. Computer Science Department Sharif University of Technology Tehran. Iran.

Yun Chang, Ching dan Clark, Stephen. 2010. *Practical Linguistic Steganography using Contextual Synonym Subtitution and Vertex Colour Coding*. Cambridge University.