Studi, Perbandingan Metode Steganografi, dan Metode Steganalisis pada Berkas HTML

Daniel Widya Suryanata / 13509083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
danielsuryanata@gmail.com

***Abstract*—Makalah ini membahas tentang teknik penyembunyian pesan (steganografi) pada berkas HTML. Ide dasar dari steganografi pada berkas HTML adalah dengan mengubah urutan dari atribut pada suatu *tag* HTML sehingga dapat menghasilkan berbagai kombinasi yang selanjutnya akan digunakan untuk menyembunyikan pesan. Makalah ini akan fokus pada cara penyembunyian pesan serta studi steganalisis pada berkas HTML.**

***Index Terms*—HTML, Steganografi, Bit, Atribut, *Tag*, *Steganalisis*, *Kunci*.**

I. PENDAHULUAN

Di tengah derasnya arus informasi saat ini, teknik penyembunyian pesan atau yang dikenal dengan istilah steganografi menjadi sangat penting. Steganografi menjadi salah satu alat penting yang digunakan untuk merahasiakan pesan.

Steganografi dapat digolongkan ke dalam kriptografi, tetapi ada perbedaan mendasar antara keduanya. Kriptografi bertujuan untuk menyandikan pesan sehingga pihak yang tidak berwenang untuk membacanya kehilangan makna dari pesan tersebut. Tetapi steganografi bertujuan untuk menghindari adanya kecurigaan dari pihak yang tidak berwenang bahwa ada pesan yang tersembunyi di balik media steganografi.

Kata steganografi sendiri berasal dari bahasa Yunani *Steganos* yang berarti *secret writing* atau tulisan rahasia. Secara formal, steganografi dapat diartikan sebagai ilmu dan seni menyembunyikan informasi dengan cara menyisipkan pesan rahasia di dalam pesan lain.

II. STEGANOGRAFI DAN BERKAS HTML

A. Properti dalam Steganografi

Berikut ini adalah ka nada atau ka na-unsur yang diperlukan dalam melakukan steganografi :

- *Embedded Message* (*hiddentext*)
  Embedded Message adalah pesan yang disembunyikan. Pada kasus ini, *embedded message*-nya adalah teks.
- *Cover-object* (*covertext*)
  Cover-object adalah pesan yang digunakan untuk menyembunyikan *embedded message*. Pada kasus ini, *cover-object*-nya adalah berkas HTML.
- *Stego-object* (*stegotext*)
  Stego-object adalah pesan yang sudah terisi oleh *embedded message*. Pada kasus ini *stego-object*-nya adalah berkas HTML yang telah disisipi pesan rahasia.
- *Stego-key*
  Stego-key adalah kunci yang digunakan untuk menyisipkan dan mengekstraksi pesan dari *stego-text*.

B. Steganalisis

Steganalisis adalah ilmu dan seni untuk mendeteksi ada-tidaknya pesan tersembunyi dalam suatu objek.

C. Struktur Berkas HTML

HTML (Hyper Text Markup Language) adalah sebuah bahasa standar yang digunakan oleh *browser* internet untuk membuat halaman dan dokumen pada sebuah web yang kemudian dapat diakses dan dibaca layaknya sebuah artikel.

Berkas HTML diawali dengan <!DOCTYPE> yang berisi keterangan tipe dokumen dalam berkas HTML tersebut.

Sebuah berkas HTML terdiri dari banyak *tag*. *Tag* menyimpan keterangan elemen dari berkas HTML tersebut. Penulisan pada kebanyakan *tag* harus ditutup dengan penutup *tag* tersebut. Tetapi ada juga *tag* yang tidak perlu ditutup.

Tag dari sebuah berkas HTML diawali dengan *tag* <html> dan pada akhir dokumen diakhiri dengan </html>.

Tag <head> dan <body> adalah elemen utama lainnya di dalam berkas HTML. Ada banyak sekali *tag* yang dimiliki oleh HTML, contohnya adalah , <img>, dan <p>.

Setiap *tag* memiliki atribut. Atribut adalah keterangan yang khusus menjelaskan isi dari *tag* tersebut. Misalnya *tag* <img> memiliki atribut src yang merupakan tautan kepada sumber dari gambar (*image*) tersebut, selain itu *tag* <img> juga memiliki atribut alt yang berfungsi untuk

memberikan keterangan pada gambar. Setiap atribut harus diisikan dengan nilai khusus. Misalnya atribut alt hendak diisi dengan nilai “mobil baru”, maka penulisannya menjadi <img alt=”mobil baru”>.

III. STEGANOGRAFI PADA BERKAS HTML

Ada beberapa metode dalam melakukan steganografi dalam berkas HTML. Berikut ini akan dibahas beberapa metode.

Pada prinsipnya steganografi pada berkas HTML dapat dilakukan dengan cara mengkombinasikan susunan atribut dari suatu *tag* apabila suatu *tag* memiliki atribut lebih dari satu. Hal ini mungkin dilakukan karena hasil yang ditampilkan pada *browser* tidak membedakan apabila atribut yang berbeda ditukar letaknya.

Misalnya pada *tag* <img> yang memiliki atribut src dan alt, dapat dibentuk dua kemungkinan dalam penyusunan atribut, yaitu :

<img src=“C:/Gambar.bmp” alt=“mobil baru”>

dan

<img alt=“mobil baru” src=“C:/Gambar.bmp”>

Bit 0 dapat dikodekan dengan kemungkinan pertama dan bit 1 dapat dikodekan dengan kemungkinan kedua. Apabila kedua kode tersebut dieksekusi oleh *browser*, maka tidak akan ada perbedaan pada halaman yang ditampilkan.

Untuk *tag* yang memiliki tiga atribut, dapat dibentuk enam kemungkinan penyusunan atribut, yaitu :

<a href=”http://www.google.com” title=”cari” target=”_blank”>

dan

<a href=”http://www.google.com” target=”_blank” title=”cari”>

dan

<a title=”cari” href=”http://www.google.com” target=”_blank”>

dan

<a title=”cari” target=”_blank” href=”http://www.google.com”>

dan

<a target=”_blank” href=”http://www.google.com” title=”cari”>

dan

<a target=”_blank” title=”cari” href=”http://www.google.com”>

Dari kedua contoh diatas, dapat disimpulkan bahwa jumlah kemungkinan berbanding lurus dengan jumlah atribut dalam suatu *tag*. Hal ini berarti *tag* yang memiliki atribut lebih banyak dapat menyimpan lebih banyak bit pesan. Banyaknya kemungkinan dapat dihitung dengan rumus :

nk = na!

nk = jumlah kemungkinan
na = jumlah atribut dalam suatu *tag*

A. Steganografi Berbasis Bit

Pada metode ini, penyembunyian data dilakukan dengan cara menggunakan 2 atribut untuk menyembunyikan 1 bit.

Misalnya pada *tag* dengan 4 atribut sebagai berikut, dapat disembunyikan maksimal 2 bit :

<img src=“C:/Gambar.bmp” alt=“mobil baru” width=”180” height=”120”>

Bit pertama dapat disembunyikan di dalam atribut src dan alt, sedangkan bit kedua dapat disembunyikan dalam atribut width dan height. Cara penyembunyian dapat menggunakan urutan alfabet dari atribut tersebut. Misal, bila atribut alt muncul lebih dahulu daripada src, maka bit pertama bernilai 0, bila src muncul lebih dahulu, maka bit pertama bernilai 1. Begitu juga dengan bit kedua yang menggunakan kombinasi *width* dan *height*.

Dengan cara ini penyembunyian dan ekstraksi data dilakukan dapat dilakukan dengan lebih cepat karena apabila ada *tag* yang memiliki banyak atribut, seluruh atribut tidak perlu diurutkan sesuai dengan kemungkinan untuk mendapatkan ekstraksi pesan. Cukup melihat per dua atribut saja, maka bisa didapatkan rangkaian bit.

Tetapi metode ini hanya dapat menyembunyikan sedikit pesan. Pada contoh diatas, sebuah *tag* yang memiliki 4 atribut seharusnya dapat menampung sampai 4 bit bila dihitung dengan rumus jumlah kemungkinan diatas, tetapi pada metode ini hanya dapat disembunyikan 2 bit saja pada *tag* tersebut.

Kelemahan lainnya adalah apabila ditemui *tag* dengan jumlah atribut ganjil, maka akan ada 1 atribut yang tidak terpakai karena penyembunyian 1 bit (ukuran terkecil pesan) dilakukan dengan 2 atribut.

B. Steganografi Berbasis Atribut

Berdasarkan teori kompleksitas algoritma, yaitu dengan

menggunakan notasi O-besar, dapat diketahui :

$$O(2^n) < O(n!)$$

Dimana 2^n adalah banyaknya kemungkinan pada pesan n bit yang hendak disisipkan, sementara n! adalah banyaknya kemungkinan penyembunyian bit pada suatu *tag* dengan n atribut, dimana n > 3. Berarti 4 atribut dapat menyembunyikan 4 bit pesan (karena 24 > 16), 5 atribut dapat menyembunyikan 5 bit pesan (karena 120 > 32), dan seterusnya. Tetapi bila n < 6, 2^{n+1} > n!. Hal ini berarti 5 atribut tidak dapat menyembunyikan 6 bit pesan walaupun masih ada 88 (120 – 32) kemungkinan pada penyembunyian 5 bit pesan.

Dibandingkan dengan metode pertama, metode berbasis atribut ini akan jauh lebih efisien karena bit yang disembunyikan akan jauh lebih banyak dibandingkan metode berbasis bit. *Tag* yang memiliki jumlah atribut ganjil juga tidak menjadi masalah karena seluruh atribut akan ikut dikombinasikan untuk menyembunyikan pesan.

Tetapi metode ini pun memiliki beberapa kelemahan, salah satu kelemahannya adalah sedikitnya bit yang dapat ditampung apabila jumlah atribut dalam suatu *tag* kurang dari 6. Apabila jumlah atribut dari suatu *tag* hanya 5, maka yang dapat ditampung dalam *tag* tersebut hanya 5 bit, tetapi masih ada cukup banyak sisa, yaitu 88 kemungkinan.

Masalah selanjutnya adalah banyaknya kemungkinan yang tidak terpakai dalam suatu *tag*. Bila mengambil contoh sebelumnya, ada 88 kemungkinan yang tidak terpakai pada suatu *tag*, sedangkan yang terpakai hanya 32 kemungkinan. Apakah 88 kemungkinan tersebut harus diikut sertakan dalam *tag*? Bila harus, bagaimana cara mengikutsertakan mereka semua? Apakah mereka menjadi tidak berarti ataukah ada beberapa kombinasi dari 32 kemungkinan yang terpakai sehingga 88 kemungkinan yang tidak terpakai menjadi terpakai di dalam *tag*?

Selain permasalahan tersebut, ada satu lagi permasalahan, yaitu mengenai banyaknya kemungkinan yang harus dibangkitkan oleh program. Hal ini perlu untuk dilakukan mengingat program akan memasukkan nilai dari suatu bit berdasarkan urutan dari atribut *tag*. Hal ini tentu menyebabkan waktu eksekusi yang lebih besar dibandingkan dengan metode berbasis bit. Hal yang sama dialami saat pesan ingin diekstraksi dari berkas HTML. Program harus memeriksa urutan dari suatu *tag*, kemudian mengurutkannya untuk mendapatkan embedded message.

C. Steganografi Berbasis Atribut dan Huruf Kapital

Steganografi berbasis atribut dan huruf kapital muncul untuk mengatasi salah satu kelemahan dari steganografi berbasis atribut, yaitu banyaknya kemungkinan yang tidak terpakai di dalam suatu *tag*.

Ide dasar dari metode ini adalah sifat *case-insensitive* dari berkas HTML. Secara lebih spesifik, *browser* akan menampilkan hasil yang sama apabila nama atribut ataupun nilai dari atribut diganti dengan huruf kapital. Contohnya :

<font size=“3” color=”red”>

<font size=“3” color=”Red”>

<font size=“3” color=”RED”>

<font SIZE=“3” Color=”red”>

Ketiga contoh tersebut tidak akan berbeda apabila dijalankan pada *browser*. Karena *tag* tersebut memiliki 2 atribut, maka ada 1 bit yang dapat disembunyikan pada *tag* tersebut. Kalau dikombinasikan dengan menggunakan huruf kapital seperti contoh diatas, maka akan tercipta banyak sekali kemungkinan yang mungkin dihasilkan dari suatu *tag*, sekalipun *tag* tersebut hanya memiliki sedikit atribut.

Tetapi apabila banyaknya kemungkinan ini digunakan secara berlebihan, akan menimbulkan kecurigaan pada pembaca *source code* dari berkas HTML tersebut.

<font sIzE=“3” CoLOr=”red”>

Tentunya gaya penulisan seperti contoh diatas akan sangat mencurigakan karena tidak wajar (jarang sekali ada atribut yang diawali dengan huruf kecil tetapi di tengah-tengah atribut tersebut ada huruf kapital pada suatu *source code*). Pembaca akan dapat dengan mudah menebak bahwa ada pesan yang tersembunyi di dalam *tag* tersebut.

Berikut ini adalah beberapa variasi yang akan meminimalkan kecurigaan pembaca *source code* :

<font color=“red”>

<font Color=“red”>

<font COLOR=“red”>

<font color=“Red”>

<font color=“RED”>

<font Color=“Red”>

<font Color=“RED”>

<font COLOR=“Red”>

<font COLOR=“RED”>

Dari satu atribut saja dapat dihasilkan 9 kemungkinan yang tidak akan terlalu memicu kecurigaan pengguna. Apabila dalam satu *tag* ada dua atribut, maka akan ada 81 (9 x 9) kemungkinan yang dapat dihasilkan dari perbedaan huruf kapital ini saja. Dan apabila urutan atribut turut diperhitungkan, maka 2 atribut dapat

menyimpan hingga 162 (81 x 2) kemungkinan, sehingga dapat menyimpan pesan hingga 7 bit panjangnya.

Untuk steganografi dengan metode berbasis atribut dan huruf kapital ini, jumlah kemungkinan yang dapat dihasilkan dapat dihitung dengan rumus :

$$nk = 81 \times na!$$

Dimana nk adalah jumlah kemungkinan, sedangkan na adalah jumlah atribut yang dimiliki oleh suatu *tag*.

Dari 3 atribut dapat dihasilkan 486 kemungkinan atau setara dengan 8 bit. Dari 4 atribut dapat dihasilkan 1944 kemungkinan atau setara dengan 10 bit. Jumlah ini merupakan jumlah yang luar biasa besar.

Kekurangan utama dari metode ini adalah besarnya waktu yang digunakan oleh program untuk mengurutkan dan memberi nilai pada semua kemungkinan yang ada.

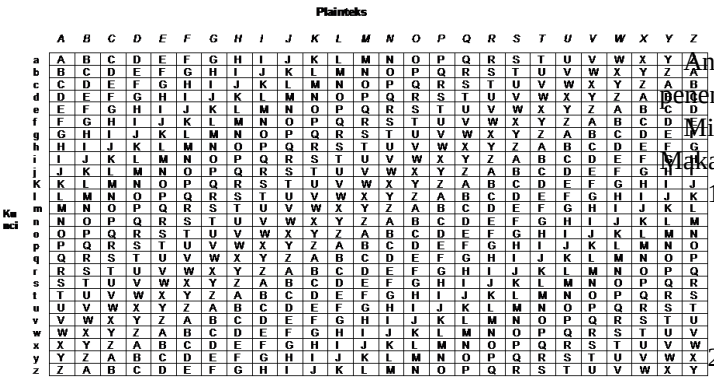
Kekurangan lainnya mirip dengan kekurangan dari steganografi dengan metode berbasis atribut, yaitu mungkin adanya kemungkinan-kemungkinan yang tidak terpakai di dalam suatu *tag*, sehingga harus dipikirkan bagaimana mengelola kemungkinan-kemungkinan tersebut.

D. Penggunaan Kunci dalam Steganografi

Bila suatu pesan ingin disisipkan ke dalam berkas HTML, maka penyisipan tersebut dapat dilakukan secara berurutan. Bit pertama pada *tag* pertama, kemudian bit kedua disisipkan setelahnya, dan begitu seterusnya hingga semua bit berhasil disisipkan.

Cara tersebut sangat sederhana, tetapi sangat tidak aman karena pembaca dapat langsung mengetahui posisi bit yang benar dan hanya perlu menebak cara pengkodean bit di dalam atribut HTML.

Akan jauh lebih baik apabila pesan tersebut disisipkan ke dalam berkas HTML dalam bentuk terenkripsi, misalnya dengan mengenkripsi pesan terlebih dahulu menggunakan Vigenere *cipher*.



Gambar 1. Bujur Sangkar Vigenere

Vigenere *cipher* adalah *cipher* yang menggunakan bujur sangkar Vigenere dalam proses enkripsinya. Contoh bujur sangkar Vigenere dapat dilihat pada Gambar 1 diatas. Apabila panjang kunci lebih pendek daripada panjang *plainteks*, maka kunci diulang. Berikut adalah rumus untuk enkripsi dan dekripsinya :

$$C_i = (P_i + K_i) \bmod 256$$
$$P_i = (C_i - K_i) \bmod 256$$

Dimana C_i adalah *cipherteks* pada posisi i, P_i adalah *plainteks* pada posisi i, dan K_i adalah kunci pada posisi i. Pada kunci, apabila i sudah melebihi panjang kunci, maka i kembali ke posisi awal. Hal ini dilakukan sampai semua *plainteks* terenkripsi atau sampai semua *cipherteks* terdekripsi.

Tetapi hal tersebut masih berbahaya karena pembaca *source code* masih mengetahui posisi bit yang benar dan hanya perlu menebak cara pengkodean bit serta melakukan kriptanalisis Vigenere *cipher* menggunakan metode Kasiski dan analisis frekuensi.

Karena alasan itulah, pada dunia steganografi, ada satu lagi kegunaan kunci, yaitu untuk membangkitkan bilangan *pseudo-random* sebagai posisi dari bit. Pada steganografi ini, bilangan *random* tersebut dibangkitkan setiap kali bit ingin disisipkan ke dalam atribut. Algoritma dari pembangkit bilangan *random* tersebut akan dijelaskan pada subbab berikut.

E. Algoritma Pembangkit Bilangan Random

Pertama, jumlah bit yang ingin disisipkan harus dihitung terlebih dahulu. Hal ini penting dilakukan, karena di dalam algoritma ini, panjang pesan harus disisipkan juga ke dalam berkas HTML.

Kunci dapat dipilih bebas oleh pengguna. Panjang kunci bisa dibatasi maupun tidak dibatasi. Kunci dapat berupa karakter apapun.

Setelah kunci dipilih, maka bisa didapatkan angka ASCII dari tiap-tiap karakter pada kunci. Misalnya apabila kuncinya adalah ITB, maka angka ASCII-nya adalah :

$$I = 73, T = 84, B = 66$$

Angka ASCII inilah yang selanjutnya akan menjadi penentu letak dari bit-bit pesan.

Misalnya bit pesan yang ingin disisipi adalah 01101110. Maka cara penyisipannya adalah sebagai berikut :

1. Panjang bit pesan dihitung dan dibandingkan dengan kapasitas penampungan dalam satu berkas HTML. Apabila memenuhi, maka pesan siap disisipkan dengan langkah-langkah berikutnya.
2. Jumlah byte *plainteks* disisipkan pertama kali, yaitu pada posisi pertama. Maka dari seluruh berkas HTML, dicari *tag* yang jumlah atributnya lebih dari 1 dan memiliki urutan ke-1.
3. Kapasistas penampungan bit pada *tag* pertama dilihat. Apabila kapasistas tersebut mampu menampung seluruh panjang pesan (dalam byte)

yang sudah diubah ke dalam bentuk bit, maka sisipkan informasi panjang pesan tersebut ke dalam bit pertama.

4. Apabila kapasistas *tag* tidak dapat menampung jumlah tersebut, maka akan dicari *tag* kedua untuk menampung sisa dari informasi panjang pesan, dan seperti halnya pada *tag* pertama, apabila *tag* kedua juga tidak memenuhi untuk menampung sisa dari informasi panjang, maka dicari *tag* ketiga, keempat, dan seterusnya hingga seluruh informasi panjang pesan dapat tertampung.
5. Informasi mengenai *tag* mana saja yang sudah pernah dipakai untuk menyisipkan pesan harus disimpan. Misalnya dalam penyisipan informasi panjang pesan saja dibutuhkan 4 *tag*, maka simpan *tag* ke-1, 2, 3, dan 4 sebagai *tag* yang sudah pernah disisipi.
6. Setelah informasi mengenai panjang pesan berhasil disisipkan ke dalam beberapa *tag* pertama, selanjutnya tinggal menentukan karakter pembatas yang akan digunakan. Misalnya karakter pembatas yang dipilih adalah "#", maka karakter tersebut dienkripsi dengan Vigenere *cipher*, diubah ke dalam bit, kemudian disisipkan dengan cara seperti menyisipkan kunci. Penyisipan karakter khusus ini dilakukan pada *tag* yang berurutan setelah *tag* terakhir dari informasi panjang pesan.
7. Angka ASCII dari karakter pertama adalah 73, maka dari seluruh berkas HTML dicari *tag* yang memiliki urutan ke-73. Apabila jumlah *tag* pada berkas HTML kurang dari 73, maka penghitungan *tag* tetap dilanjutkan, namun dimulai dari awal lagi. Misalnya berkas HTML tersebut memiliki 63 *tag*, maka *tag* yang dipilih adalah *tag* ke-10 (karena dari awal sampai akhir ada 63 *tag*, maka sisa 10 *tag* dihitung dari awal).
8. Selanjutnya dilihat berapa bit yang dapat disisipkan ke dalam *tag* tersebut (tentu akan berbeda tergantung metode mana yang dipilih). Misalnya pada *tag* ke-73 dapat disisipkan 2 bit, maka ambil 2 bit pertama dari pesan, yaitu 01 untuk disisipi ke dalam *tag* tersebut.
9. Harus dipastikan agar *tag* yang hendak dipilih untuk disisipkan belum pernah disisipi oleh pesan lain. Ini adalah kegunaan dari pencatatan informasi *tag* mana saja yang sudah pernah disisipi.
10. Apabila *tag* yang hendak dipilih sudah pernah disisipi, tambahkan satu pada *tag* yang hendak dipilih, misalnya : *tag* ke-10 sudah pernah disisipi, maka pilih *tag* ke-11 untuk disisipi. Hal ini terus dilakukan sampai mendapatkan *tag* yang belum pernah disisipi.
11. Kemudian kunci pada posisi kedua dilihat, angka ASCII-nya adalah 84. Maka dicari *tag* dengan posisi ke-84. Kemudian dilakukan pemeriksaan dengan informasi *tag* yang sudah pernah disisipi.
12. Langkah tersebut diulangi hingga bit pesan seluruhnya berhasil disisipi ke dalam berkas HTML. Setiap iterasi, posisi kunci selalu ditambah dengan 1.
13. Apabila semua karakter pada kunci sudah digunakan tetapi pesan masih belum selesai disisipkan, ulangi penyisipan dengan karakter pertama kunci dan seterusnya hingga penyisipan pesan selesai.
14. Apabila setelah seluruh pesan disisipkan tetapi masih ada *tag* yang belum menampung informasi apapun, urutan atribut dalam *tag* tersebut diacak untuk menyulitkan proses steganalisis.

Ekstraksi pesan dari berkas HTML yang telah disisipi juga menggunakan cara yang serupa. Pengguna hanya perlu untuk mengetahui kunci yang dipakai untuk penyisipan saja.

Karakter khusus yang diletakkan setelah informasi panjang pesan berguna untuk mengetahui secara jelas *tag* mana saja yang digunakan untuk menyimpan informasi panjang pesan tersebut. Apabila panjang pesan sudah berhasil diketahui, maka iterasi dilakukan sejumlah panjang pesan tersebut untuk mengekstraksi bit-bit yang disisipkan di dalam berkas HTML tersebut.

Langkah ke-14 dalam algoritma tersebut aman untuk dilakukan karena *tag* yang di-*random* tersebut tidak akan dipakai baik oleh proses penyisipan maupun oleh proses ekstraksi.

Algoritma penyisipan ini merupakan algoritma kunci simetri karena untuk mengenkripsi maupun mendekripsi pesan membutuhkan kunci yang sama.

IV. STEGANALISIS PADA BERKAS HTML

Steganalisis adalah ilmu dan seni untuk mendeteksi ada-tidaknya pesan tersembunyi dalam suatu objek.

Steganalisis pada berkas HTML tergolong sulit karena tidak seperti steganalisis pada berkas citra yang dapat dilakukan dengan metode enhanced LSB yang dapat mengacaukan media steganografi, ataupun pada berkas suara yang dapat dilakukan dengan mendengar derau-derau yang muncul apabila pesan yang disisipkan terlalu banyak. Pada berkas HTML akan sulit sekali untuk menentukan apakah berkas tersebut disisipi pesan atau tidak karena perbedaan tidak dapat dilihat pada *browser*, dan bila dilihat dari *source code* berkas HTML, algoritma untuk mengacak urutan atribut membuatnya semakin sulit.

Apabila pesan tidak dienkripsi dan diacak posisinya sebelum ditempatkan ke dalam berkas HTML, maka proses steganalisis akan menjadi semakin mudah. Steganalis hanya perlu menebak pola penyusunan atribut pada *tag* HTML. Misalnya, apakah penyembunyian pesan dilakukan dengan mengurutkan atribut mulai dari alfabet

terdahulu atau mengurutkan atribut mulai dari alfabet terakhir? Ataukah pengurutan atribut dilakukan dengan pola-pola lainnya? Hal ini dapat dilakukan secara *brute force*.

Bila pesan dienkripsi terlebih dahulu dan posisi dari penempatan bit dibangkitkan *random* oleh kunci, maka pekerjaan steganalis menjadi sulit karena steganalis harus menebak *tag* mana yang disisipi pesan.

V. CONTOH KASUS STEGANOGRAFI

Pada bab ini akan dibahas beberapa contoh kasus untuk steganografi pada berkas HTML.

Contoh 1 :

Pesan yang ingin disisipkan : "A"

Metode : berbasis bit, tanpa kunci.

Cover object :

```
<html>
 <head>
 </head>

 <body>
  <img src="C;/gambar1.jpg" alt="Kriptografi"> </img>
  <p> <font size="4" color="red"> Ini adalah salah satu contoh kriptografi </font> </p>
  <a href="http://www.kripto.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

  <img src="C;/gambar2.jpg" alt="Steganografi"> </img>
  <p> <font size="4" color="blue"> Ini adalah salah satu contoh steganografi </font> </p>
  <a href="http://www.stegano.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

  <img src="C;/gambar3.jpg" alt="Contoh Cipher"> </img>
  <p> <font size="4" color="blue"> Ini adalah salah satu contoh Vigenere Cipher </font> </p>
  <a href="http://www.vigenere.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>
 </body>
</html>
```

Pembahasan :

Angka ASCII dari "A" adalah 65, sehingga apabila 65 diubah ke dalam bentuk bit adalah 01000001.

Pada berkas HTML tersebut ada 9 buah *tag*, tetapi yang dibutuhkan hanya 8. Maka *tag* terakhir (Silakan klik tautan berikut
) tidak akan dipakai.

Pada contoh kali ini panjang plainteks juga tidak disisipkan.

Stego object :

```
<html>
 <head>
 </head>

 <body>
  <img alt="Kriptografi" src="C;/gambar1.jpg"> </img>
  <p> <font size="4" color="red"> Ini adalah salah satu contoh kriptografi </font> </p>
  <a href="http://www.kripto.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

  <img alt="Steganografi" src="C;/gambar2.jpg"> </img>
  <p> <font color="blue" size="4"> Ini adalah salah satu contoh steganografi </font> </p>
  <a href="http://www.stegano.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

  <img alt="Contoh Cipher" src="C;/gambar3.jpg"> </img>
  <p> <font size="4" color="blue"> Ini adalah salah satu contoh Vigenere Cipher </font> </p>
  <a href="http://www.vigenere.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>
 </body>
</html>
```

Untuk melakukan dekripsi pada berkas HTML tersebut, cukup membaca delapan *tag* pertama.

Contoh 2:

Pesan yang ingin disisipkan : "ITB"

Metode : berbasis atribut, kunci = "IF". Tetapi kunci hanya digunakan untuk membangkitkan bilangan *random*, enkripsi *plainteks* menggunakan kunci tidak dilakukan.

Cover object :

```
<html>
 <head>
 </head>
 <body>
  <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba1.css" />
  <img alt="Kriptografi" src="C:/gambar1.jpg"> </img>
  <p> <font size="4" color="red"> Ini adalah salah satu contoh kriptografi </font> </p>
  <a href="http://www.kripto.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

  <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba2.css" />
  <img alt="Steganografi" src="C:/gambar2.jpg"> </img>
  <p> <font color="blue" size="4"> Ini adalah salah satu contoh steganografi </font> </p>
  <a href="http://www.stegano.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

  <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba3.css" />
  <img alt="Contoh Cipher" src="C:/Vigenere.jpg"> </img>
  <p> <font size="4" color="blue"> Ini adalah salah satu contoh Vigenere Cipher </font> </p>
  <a href="http://www.vigenere.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>

  <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba4.css" />
  <img alt="Contoh Cipher" src="C:/gambar1.jpg"> </img>
  <p> <font size="4" color="red"> Ini adalah salah satu contoh Caesar Cipher </font> </p>
  <a href="http://www.caesar.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

  <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba5.css" />
  <img alt="Contoh Cipher" src="C:/gambar2.jpg"> </img>
  <p> <font color="blue" size="4"> Ini adalah salah satu contoh Engima </font> </p>
```

```
    <a href="http://www.enigma.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

    <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba6.css" />
    <img alt="Contoh Cipher" src="C:/gambar3.jpg"> </img>
    <p> <font size="4" color="blue"> Ini adalah salah satu contoh Playfair Cipher </font> </p>
    <a href="http://www.playfair.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>

    <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba7.css" />
    <img alt="Contoh Cipher" src="C:/gambar1.jpg"> </img>
    <p> <font size="4" color="red"> Ini adalah salah satu contoh DES </font> </p>
    <a href="http://www.des.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

    <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba8.css" />
    <img alt="Contoh Cipher" src="C:/gambar2.jpg"> </img>
    <p> <font color="blue" size="4"> Ini adalah salah satu contoh AES </font> </p>
    <a href="http://www.aes.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

    <link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba9.css" />
    <img alt="Contoh Cipher" src="C:/gambar3.jpg"> </img>
    <p> <font size="4" color="blue"> Ini adalah salah satu contoh SHA-1 </font> </p>
    <a href="http://www.sha.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>
  </body>
</html>
```

Pembahasan :

Angka ASCII dari "ITB" adalah 73, 84, dan 66. Sedangkan ASCII dari "IF" adalah 73, 70. Panjang byte *plainteks* adalah 3. Maka yang akan ditulis pada berkas HTML adalah "3#ITB" atau bila diubah ke rangkaian bit adalah "00110011 00100011 01001001 01010100 01000010".

Bila *plainteks* disisipi ke dalam berkas HTML sesuai kunci, maka berikut adalah list dari *tag* yang sudah pernah disisipi secara berurutan adalah : (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 34, 15, 35, 16, 36, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30).

Berdasarkan sifat dari contoh berkas HTML tersebut, *tag* ke-1, 5, 9, 13, 17, 21, 25, 29, dan 33 dapat menampung dua bit. Sedangkan *tag* lain hanya mampu menampung 1 bit saja. Berikut ini adalah posisi penempatan tiap-tiap bit dalam *tag*.

Tabel 1. Penempatan Bit "3"

Bit	Posisi *Tag*
0	1
0	1
1	2
1	3
0	4
0	5
1	5
1	6

Tabel 2. Penempatan Bit "#"

Bit	Posisi *Tag*
0	7
0	8
1	9
0	9
0	10
0	11
1	12
1	13

Tabel 3. Penempatan Bit "I"

Bit	Posisi *Tag*
0	14
1	34
0	15
0	35
1	16
0	36
0	17
1	17

Tabel 4. Penempatan Bit "T"

Bit	Posisi *Tag*
0	18
1	19
0	20
1	21
0	21
1	22
0	23
0	24

Tabel 5. Penempatan Bit "B"

Bit	Posisi *Tag*
0	25
1	25
0	26
1	27
0	28
1	29
0	29
0	30

Yang perlu diperhatikan adalah saat menempatkan bit "1" pada *tag* ke-13. *Tag* ke-13 hanya menampung 1 bit. Sebenarnya *tag* ke-13 dapat digunakan untuk menampung 2 bit, tetapi karena *tag* ke-13 digunakan untuk menyimpan akhir dari pembatas, maka *plainteks* harus dimulai dari *tag* yang lain. *Tag* ke-13 seharunya memiliki tempat

penyimpanan untuk bit “00”, “01”, “10”, dan “11”, tetapi pada kasus ini, *tag* ke-13 hanya memiliki tempat penyimpanan untuk bit “0” dan “1”, sehingga dari 6 kemungkinan, yang digunakan hanya 2 saja.

Selain itu, pada Tabel 3, setelah bit 0 ditempatkan ke dalam *tag* ke-14, bit berikutnya ditempatkan dalam *tag* ke-34, begitu juga 2 rangkaian bit berikutnya. Hal ini disebabkan karena penggunaan kunci dalam penempatan bit. Bit pertama dari karakter “I” disisipkan berdasarkan karakter pertama kunci, yaitu “I”, sedangkan bit kedua dari karakter “I” disisipkan berdasarkan karakter kedua kunci, yaitu “F”.

Maka hasil dari penyisipan tersebut adalah :

Stego Object :

```
<html>
<head>
</head>
<body>
<link href="http://www.haha.co.id/sites/coba1.css" rel="stylesheet" type="text/css" />
<img src="C:/gambar1.jpg" alt="Kriptografi"> </img>
<p> <font size="4" color="red"> Ini adalah salah satu contoh kriptografi </font> </p>
<a href="http://www.kripto.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

<link href="http://www.haha.co.id/sites/coba2.css" type="text/css" rel="stylesheet" />
<img src="C:/gambar2.jpg" alt="Steganografi"> </img>
<p> <font color="blue" size="4"> Ini adalah salah satu contoh steganografi </font> </p>
<a href="http://www.stegano.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

<link rel="stylesheet" href="http://www.haha.co.id/sites/coba3.css" type="text/css" />
<img alt="Contoh Cipher" src="C:/Vigenere.jpg"> </img>
<p> <font color="blue" size="4"> Ini adalah salah satu contoh Vigenere Cipher </font> </p>
<a title="Lihat Vigenere" href="http://www.vigenere.com"> Silakan klik tautan berikut </a>

<link href="http://www.haha.co.id/sites/coba4.css" type="text/css" rel="stylesheet" />
<img alt="Contoh Cipher" src="C:/gambar1.jpg"> </img>
<p> <font color="red" size="4"> Ini adalah salah satu contoh Caesar Cipher </font> </p>
<a title="Lihat Kriptografi" href="http://www.caesar.com"> Silakan klik tautan berikut </a>

<link href="http://www.haha.co.id/sites/coba5.css" type="text/css" rel="stylesheet" />
<img alt="Contoh Cipher" src="C:/gambar2.jpg"> </img>
<p> <font size="4" color="blue"> Ini adalah salah satu contoh Engima </font> </p>
<a href="http://www.enigma.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

<link rel="stylesheet" href="http://www.haha.co.id/sites/coba6.css" type="text/css" />
<img src="C:/gambar3.jpg" alt="Contoh Cipher"> </img>
<p> <font color="blue" size="4"> Ini adalah salah satu contoh Playfair Cipher </font> </p>
<a href="http://www.playfair.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>

<link href="http://www.haha.co.id/sites/coba7.css" type="text/css" rel="stylesheet" />
<img alt="Contoh Cipher" src="C:/gambar1.jpg"> </img>
<p> <font color="red" size="4"> Ini adalah salah satu contoh DES </font> </p>
<a href="http://www.des.com" title=Lihat Kriptografi"> Silakan klik tautan berikut </a>

<link href="http://www.haha.co.id/sites/coba8.css" type="text/css" rel="stylesheet" />
<img alt="Contoh Cipher" src="C:/gambar2.jpg"> </img>
<p> <font color="blue" size="4"> Ini adalah salah satu contoh AES </font> </p>
<a href="http://www.aes.com" title="Lihat Steganografi"> Silakan klik tautan berikut </a>

<link type="text/css" rel="stylesheet" href="http://www.haha.co.id/sites/coba9.css" />
<img src="C:/gambar3.jpg" alt="Contoh Cipher"> </img>
<p> <font color="blue" size="4"> Ini adalah salah satu contoh SHA-1 </font> </p>
<a href="http://www.sha.com" title="Lihat Vigenere"> Silakan klik tautan berikut </a>
</body>
</html>
```

Untuk melakukan dekripsi, lakukan pembacaan dari *tag* pertama, kedua, ketiga, dan seterusnya secara urut sampai menemukan bit terakhir dari pembatas (karakter “#”) yang berada di *tag* ke-13. Kemudian gunakan karakter pertama dari kunci (“I”), hitung angka ASCII-nya, kemudian sesuaikan dengan jumlah *tag* yang ada pada HTML tersebut. Karena HTML tersebut tidak sampai berjumlah 73, maka baca *tag* ke-14 untuk mendapatkan bit. Kemudian baca bit berikutnya dengan karakter kedua dari kunci, yaitu “F”, karena jumlah *tag* pada HTML tersebut tidak sampai 70, maka baca bit ke-34. Begitu seterusnya hingga terbentuk 3 karakter pesan (3 karakter diketahui dari tag yang berisi panjang pesan).

VI. KESIMPULAN

Steganografi dengan memanfaatkan berkas HTML sebagai media penyembunyian dapat dilakukan dengan memanfaatkan sifat fleksibel serta case-insensitive dari berkas HTML. Salah satu caranya adalah dengan menukar letak dari atribut-atribut dalam suatu *tag*.

Steganografi ini dapat dilakukan dengan 3 metode, yaitu metode berbasis bit, berbasis atribut, dan berbasis atribut serta huruf kapital. Masing-masing memiliki kelebihan dan kelemahannya sendiri.

Pada steganalisis berkas HTML, penentuan apakah dalam suatu berkas HTML terdapat sisipan pesan atau tidak adalah hal yang sulit untuk dilakukan. Tetapi apabila diyakini ada pesan tersembunyi di dalam suatu berkas HTML, maka untuk memecahkannya relatif mudah. Hal tersebut dapat dilakukan dengan cara *brute force* kunci.

Keuntungan menggunakan berkas HTML sebagai media steganografi adalah sulitnya steganalis untuk menentukan apakah ada pesan yang tersembunyi pada berkas.

Kerugian menggunakan berkas HTML sebagai media steganografi adalah kapasitas yang dimiliki oleh berkas HTML relative rendah untuk menampung pesan.

Sangat mungkin untuk menemukan metode baru dalam steganografi pada berkas HTML. Sejauh ini belum banyak pihak yang mengembangkan steganografi jenis ini.

REFERENCES

[1] Munir, Rinaldi. 2009. "Diktat Kuliah IF3058, Kriptografi," Bandung : Program Studi Teknik Informatika ITB.

[2] http://konteseoblog.blogspot.com/2010/04/html-adalah.html. Tanggal akses : 14 Maret 2012, pukul 20.00.

[3] http://www.htmlquick.com. Tanggal akses : 14 Maret 2012, pukul 20.00.

PERNYATAAN

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

Bandung, 16 Maret 2012

Daniel Widya Suryanata
13509083