Steganografi dalam Penurunan dan Pengembalian Kualitas Citra konversi 8 bit dan 24 bit

David Soendoro

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Alamat: Jalan Ganeca No. 10 Bandung
e-mail: david.soendoro@gmail.com

ABSTRAK

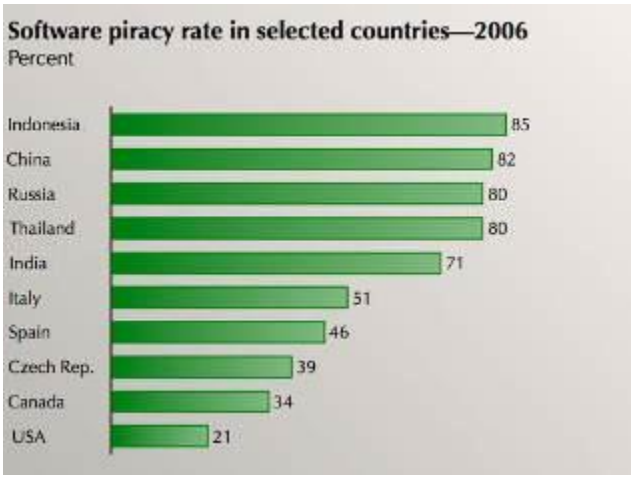
Steganografi, definisi singkatnya adalah sebuah gabungan seni dan ilmu dalam menyembunyikan pesan dalam pesan lainnya. Pesan tersebut dapat berupa tulisan, gambar, *audio*, ataupun *video*. Tujuan utama dari steganografi adalah mengirimkan pesan tersebut tanpa ketahuan pada proses transmisi sehingga kriptografer harus membuat cara agar pesan dapat dienkripsi dan didekripsi dengan aman. Pokok bahasan pada makalah ini adalah konversi citra 24 bit menjadi 8 bit dan mengembalikannya. Tujuan dari steganografi ini adalah mencegah pengambilan karya berupa citra tanpa izin namun tetap membuat citra dapat dipakai sehingga menguntungkan dari aspek promosi. Citra asli adalah sebuah citra 24 bit yang beberapa bitnya akan dioper ke tempat lain untuk membentuk citra 8 bit, yang lebih miskin warna namun tetap memiliki bentuk yang sama. Setelah memperoleh izin dari sang pembuat gambar maka citra harus juga dapat didekripsi kembali menjadi citra 24 bit.

Kata kunci: Steganografi, enkripsi citra, dekripsi citra, citra 24 bit, citra 8 bit.

BAB I - PENDAHULUAN

Latar Belakang

Fenomena pembajakan kekayaan intelektual di Indonesia merupakan salah satu yang sangat populer, hal ini disebabkan kurang mampunya masyarakat untuk membayar dalam memperoleh suatu hasil karya, terutama digital yang nilainya terkadang kurang dihargai. Namun fenomena ini di lain pihak juga memberikan keuntungan dalam bidang promosi bagi para artis. Semakin banyak dibajak maka akan semakin terkenal, begitulah kira-kira fenomena pembajakan di Indonesia. Ilmu steganografi sesungguhnya dapat menjadi jawaban bagi permasalahan tersebut, mulai dari bentuk yang paling sederhana adalah citra gambar. Dengan mengenkripsi sebuah gambar menjadi kualitas yang lebih buruk namun tetap dapat dinikmati oleh pengguna yang tidak ingin membayar akan menjadi sebuah promosi yang bagus, pihak yang membayar pun tidak merasa rugi karena akan mendapatkan gambar yang memiliki kualitas lebih tinggi daripada yang tidak membayar.



Gambar 1 - pembajakan di Indonesia

Rumusan Masalah

Citra 24 bit terdiri dari 24 bit setiap pixelnya, hal ini terdiri dari RRGGBB di mana R adalah Red atau merah, G adalah Green atau hijau, dan B adalah Blue untuk biru. Setiap karakter di atas mewakili sebuah heksadesimal atau bernilai 4 bit dan tiap warna memilki 256 kemungkinan ($2^4 * 2^4$). Sedangkan pada citra 8 bit sebuah pixel terdiri dari 256 warna dasar atau dapat diubah sesuai keinginan (Palette Encryption). Sehingga kita bisa memotong beberapa bit dari 24 bit menjadi 8 bit dan menentukan palet yang sesuai dengan gambar tersebut. Tentunya karena sebuah gambar tetap menyimpan informasi 24 bit gambar tersebut, ukuran gambar tidak berubah dan gambar dapat dikembalikan ke warna semula apabila diinginkan. Selama ini pengubahan format citra 24 bit ke 8 bit dan kembali ke 24 bit telah dapat dilakukan, namun menghadapi kendala di mana citra 24 bit yang dikembalikan tidak memiliki kualitas yang sama dengan citra awalnya. Berikut adalah contoh nyata pengkonversian citra dengan menggunakan salah satu kakas untuk mengkonversi citra 24 bit ke 8 bit dan kembali ke 24 bit.

Gambar 2 - lena 24 bit

Gambar 3 - lena 8 bit hasil konversi

Gambar 4 - konversi kembali ke 24 bit

Batasan Masalah

Sebenarnya ilmu steganografi ini dapat diterapkan pada media lainnya seperti suara ataupun video. Namun pada makalah ini penulis membatasi masalah pada masalah citra. Hal ini dikarenakan tingkat kesulitan steganografi video dan audio yang sangat tinggi yang menurut penulis adalah ilmu yang harus dipelajari mulai dari melakukan steganografi pada sebuah citra. Format yang akan dibahas adalah format gambar standard, yakni BMP. Alasan penulis membatasinya hanya BMP adalah hanya BMP yang memiliki format penggambaran 8 bit dan 24 bit, selain itu BMP dapat dengan mudah dikonversi ke bentuk lainnya seperti GIF maupun JPG/JPEG.

BAB II – METODE

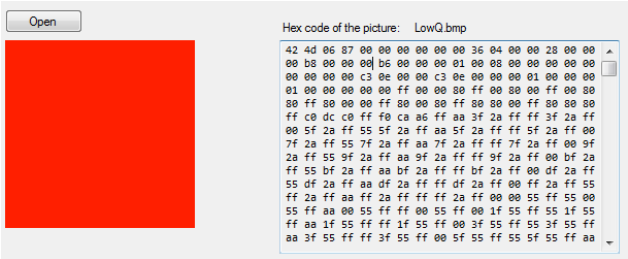
Kerangka Citra BMP

Setiap format gambar memiliki *header* untuk membedakan file tersebut dengan ekstensi yang lainnya. Berikut adalah format *header* ekstensi .bmp

Offset	Size	Hex Value	Value	Description
0h	2	42 4D	"BM"	Magic Number (unsigned integer 66, 77)
2h	4	46 00 00 00	70 Bytes	Size of the BMP file
6h	2	00 00	Unused	Application Specific
8h	2	00 00	Unused	Application Specific
Ah	4	36 00 00 00	54 bytes	The offset where the bitmap data (pixels) can be found.
Eh	4	28 00 00 00	40 bytes	The number of bytes in the header (from this point).
12h	4	02 00 00 00	2 pixels	The width of the bitmap in pixels
16h	4	02 00 00 00	2 pixels	The height of the bitmap in pixels
1Ah	2	01 00	1 plane	Number of color planes being used.
1Ch	2	18 00	24 bits	The number of bits/pixel.
1Eh	4	00 00 00 00	0	BI_RGB, No compression used
22h	4	10 00 00 00	16 bytes	The size of the raw BMP data (after this header)
26h	4	13 0B 00 00	2,835 pixels/meter	The horizontal resolution of the image
2Ah	4	13 0B 00 00	2,835 pixels/meter	The vertical resolution of the image
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	Means all colors are important
Start of Bitmap Data				
36h	3	00 00 FF	0 0 255	Red, Pixel (1,0)
39h	3	FF FF FF	255 255 255	White, Pixel (1,1)
3Ch	2	00 00	0 0	Padding for 4 byte alignment (Could be a value other than zero)
3Eh	3	FF 00 00	255 0 0	Blue, Pixel (0,0)
41h	3	00 FF 00	0 255 0	Green, Pixel (0,1)
44h	2	00 00	0 0	Padding for 4 byte alignment (Could be a value other than zero)

Gambar 5 - kerangka citra BMP

Contoh:



Gambar 6 - contoh Bitmap (8 bit)

Pengkodean Bitmap adalah *low-endian*

`42 4d` = Format BMP ('Magic Number')
`06 87 00 00` = Size (34566 bytes)
`36 04 00 00` = Mulainya pixel (byte ke-1078)
28 00 00 00 = Panjang header
b8 00 00 00 = Lebar gambar
b6 00 00 00 = Tinggi gambar
08 00 = Bit / pixel

Terdapat juga beberapa informasi lainnya seperti resolusi pixel/meter untuk print dan palet gambar tersebut (akan dibahas pada subbab berikutnya).

Perlu juga diperhatikan akan perbedaan kerangka bitmap 24 bit dan bitmap 8 bit, perbedaan dari kedua kerangka ini antara lain:

1. Pada header 1ch, tempat menyimpan data berapa bit per pixel tentunya apabila 24 bit akan tersimpan nilai heksa 18h atau 24 desimal sedangkan pada header bmp 8 bit akan tersimpan nilai heksa 8h atau 8 desimal.
2. Pada header citra 8 bit terdapat *palette* yang digunakan oleh citra tersebut di mana nantinya citra hanya akan mereferensikan ke *palette* ini, *palette* pada header disimpan dalam 4 *byte* yang berisi RGB dan 1 *byte* kosong atau 00h. Hal ini mengkibatkan header citra 8 bit bmp jauh lebih besar ketimbang header citra 24 bit bmp, karena pada 24 bit bmp seluruh informasi pixel langsung disimpan pada tempatnya yang dimulai pada informasi yang diberikan oleh header pada posisi Ah.

Selain kerangka, isi bmp 24 bit dan bmp 8 bit juga memiliki sedikit perbedaan, di mana pada bmp 24 bit, informasi tiap pixel disimpan pada file sedangkan pad bmp 8 bit hanya pendekatan dengan menggunakan palette. Sedangkan informasi pada pixel data sebenarnya hanyalah referensi terhadap palette yang telah didefinisikan di atas, hal ini menyebabkan mungkinnya lebih banyak warna pada 8 bit yang sesungguhnya sangat terbatas pada 256 macam warna, hal ini juga berakibat baik di mana pada citra yang lebih banyak menggunakan suatu warna saja akan lebih sedikit mengalami penurunan kualitas. Hal inilah yang menyebabkan pewarnaan 8 bit disebut indexed color atau warna yang telah ditentukan sebelumnya.

Palleting dari 24 bit ke 8 bit

Tidak seperti citra 24 bit, citra 8 bit tidak menggunakan kombinasi antara 3 warna dasar (R, G, dan B) melainkan menggunakan *palette* atau seperti seorang pelukis, memiliki 256 macam pilihan warna. Dengan palet inilah sesungguhnya kita bisa menyimpan gambar 24 bit ke dalam gambar 8 bit tanpa membuat ukurannya membengkak. Prinsip dasarnya adalah, mengambil warna yang sering dipakai pada suatu gambar saja, memang akan terjadi penurunan kualitas gambar namun tentu gambarnya tidak akan menjadi terlalu buruk untuk dinikmati oleh pengguna. Serta, setelah menggabungkannya dengan fungsi pengembalian citra dari 8 bit ke 24 bit, kita telah berhasil melakukan penyembunyian citra 24 bit ke dalam citra 8 bit.

Algoritma *Paletting*:

1. Buat sebuah standard palette
   Cara membuat palette ini beragam tergantung pada kebutuhan pengkonversian citra, sebuah standard *palette* untuk web akan dibahas pada bab implementasi, *palette* ini dapat juga dikustomisasi untuk memaksimalkan hasil gambar 8 bit.

   Dalam membuat *palette* sebaiknya buat dengan prinsip gradasi sehingga mudah untuk mengambilnya kembali. Sebuah *palette* dapat dibuat dari beberapa warna utama untuk kemudian dibuat menjadi gradasi, berikut adalah algoritma yang dapat dipakai dalam membuat *palette* gradasi.

```
const DWORD STANDARD_PALETTE[] = {00,51,102,153,204,255}
const INT STANDARD_COLOR_SIZE = 6;
const INT STANDARD_PALETTE_VAL_DIF = 51;
// difference between two consecutive standard color Palette.
```

Gambar 7 - menentukan warna dasar

```
DWORD dwColorMapTable[216] = {0};
int  nColorMapIdx = 0;
for (int nBlueIdx = 0; nBlueIdx < STANDARD_COLOR_SIZE; ++nBlueIdx)
{
    for(int nGreenIdx = 0; nGreenIdx < STANDARD_COLOR_SIZE; ++nGreenIdx)
    {
        for(int nRedIdx = 0; nRedIdx < STANDARD_COLOR_SIZE; ++nRedIdx)
        {
            RGBQUAD objColor;
            objColor.rgbRed    = STANDARD_PALETTE[nRedIdx];
            objColor.rgbGreen = STANDARD_PALETTE[nGreenIdx];
            objColor.rgbBlue = STANDARD_PALETTE[nBlueIdx];
            objColor.rgbReserved = 0;
            memcpy(&dwColorMapTable[nColorMapIdx],&objColor,sizeof(RGBQUAD));
            ++nColorMapIdx;
        }
    }
}
```

Gambar 8 - membuat *palette* gradasi

Contoh di atas adalah palette dasar yang terdiri dari warna RGB pada 8 bit dengan nilai R, G, dan B 0 – 5 hingga kemungkinan warna adalah $6^3 = 216$.

Kemungkinan warna RGB itu adalah:
0-0-0
0-0-1
0-0-2
…
5-5-5
Di mana tiap angka mewakili 51 yang merupakan hasil selisih yang ditentukan pada standard palette (hasil dari pendekatan 256 / 5).

Tiap warna disimpan dalam 24 bit yang memiliki format sama dengan gambar 24 bit yakni RRGGBB.

2. Mencocokkan pixel dari citra asli (24 bit) dengan citra hasil penurunan kualitas (8 bit)
Berikut adalah potongan algoritma dalam mencocokkan pixel citra dengan *palette:*

```
UINT GetPixelValue(UINT uPixelValue_i)
{
    UINT uRetValue = 0;
    UINT uPos = uPixelValue_i / PALETTE_VAL_DIF;
    if(0 == uPixelValue_i % PALETTE_VAL_DIF)
    {
        uRetValue = uPixelValue_i/PALETTE_VAL_DIF;
    }
    else
    {
        if(abs(uPixelValue_i - STANDARD_PALETTE [uPos]) >
           abs(uPixelValue_i - STANDARD_PALETTE [uPos+1]))
        {
            uRetValue = uPos+1;
        }
        else
        {
            uRetValue = uPos;
        }
    }
    return uRetValue;
}
```

Gambar 9 - algoritma GetPixelValue()

Algorima di atas akan bekerja dengan baik apabila *palette* dibuat gradasi.

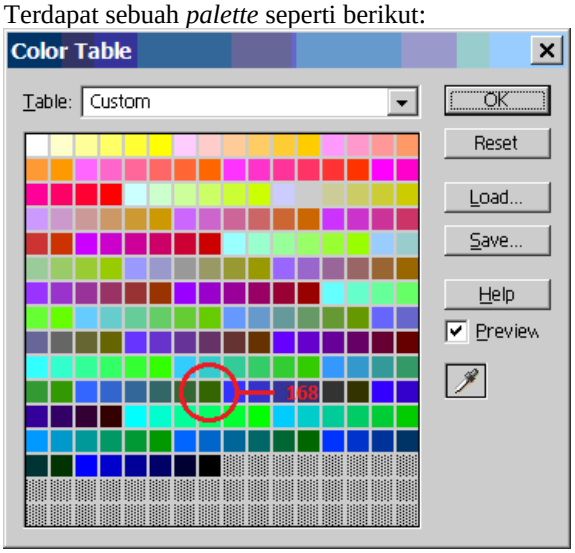
3. Ubah header dari 24 bit menjadi bit, seperti pada subbab sebelumnya untuk disimpan ke file.

Pengembalian citra 8 bit ke citra 24 bit
Seperti telah dibahas sebelumnya, tiap warna pada palet sesungguhnya adalah warna 24 bit atau sama dengan format 24 bit yang dipakai dalam gambar dan format 8 bit adalah menghemat penulisan warna menjadi hanya pada palet dan yang lain hanya merujuknya.

Permasalahannya adalah bagaimana menjaga agar warna yang tidak terwakilkan pada palet dapat diambil kembali. Untuk mengatasi hal tersebut kita dapat mencatatnya pada akhir program dengan format tergantung dari offset palet terbesar. Misalnya pada contoh di atas, palet dibuat dengan offset terbesar 256 / 5 = 51 maka untuk tiap pixel akan dicatat 6 * 3 bit offsetnya, yaitu 18 bit. Secara teori cara ini akan membuat ukuran lebih besar daripada gambar biasa 24 bit, karena gambar sendiri adalah 8 bit mengakibatkan 18 + 8 = 26 bit per pixel, namun pada prakteknya kita dapat menekan ukuran dengan membuat *palette* yang lebih kecil offsetnya.

Penggambaran dari sebuah bitmap pada program hanya dilakukan hingga panjang kali lebar yang ada disimpan pada header seperti pada subbab 1. Mengetahui hal tersebut, kita dapat menggunakan metode penulisan di akhir file di atas.

Contoh pengembalian citra:
1. Terdapat sebuah *palette* seperti berikut:



Gambar 10 - contoh *palette* 8-bit

2. Pada file gambar 8 bit terdapat informasi bahwa pixel pada suatu tempat adalah "`a8`" dalam heksa atau "168" dalam decimal, maka kita akn merujuk dari *palette*, warna ke 168, yaitu seperti yang ditunjukkan gambar 6 di atas yaitu warna hijau dengan format warna RRGGBB 336600

3. Kemudian ambil offset yang telah disimpan di akhir file, misalkan pada akhir program terdapat offset sebagai berikut 00111 10101 11001 maka tambahkan data tersebut ke data RGB yang telah diperoleh:
   R = 33h + 00111b = 51 + 7 = 58 = 3Ah
   G = 66h + 10101b = 102 + 21 = 123 = 7Bh
   B = 00h + 11001b = 0 + 25 = 25 = 19h
   Atau format RGB 24 bitnya adalah 3A7B19

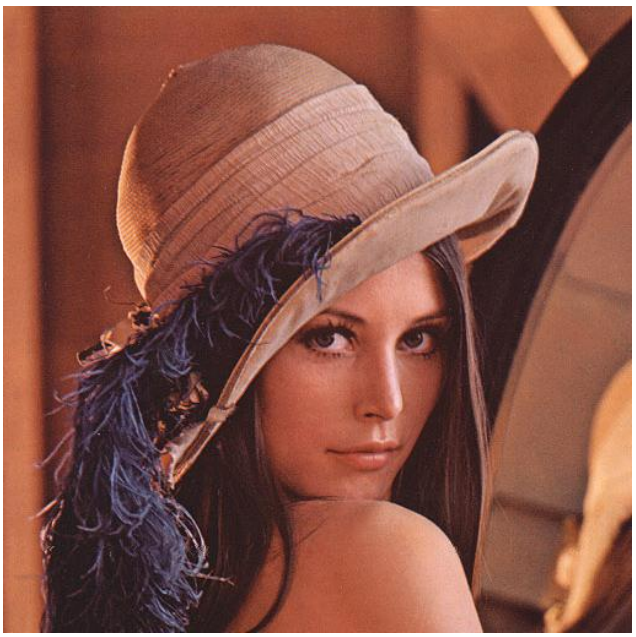
BAB III – IMPLEMENTASI

Pada contoh implementasi, *palette* yang akan dibuat adalah *palette generic* yang mencakup seluruh warna dengan membaginya menjadi 5 bagian (0 – 51, 51 – 102, 102 – 153, 153 – 204, 204 – 255) dengan kombinasi 6 * 6 * 6 RGB.

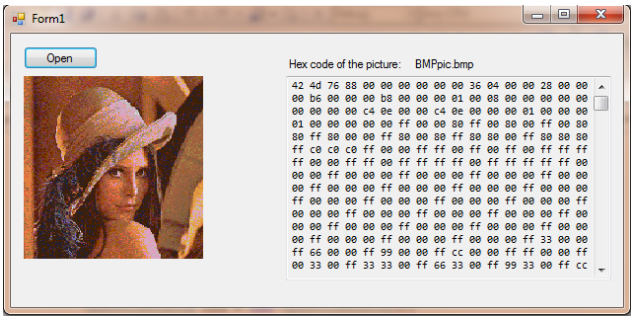
Berikut adalah tampilan antar muka program konversi yang dibuat

Gambar 11 - tampilan antar muka

Dengan tombol “open” adalah untuk membuka file 32 bit atau 8 bit yang hasilnya akan ditampilkan di bawahnya seperti di bawah ini.

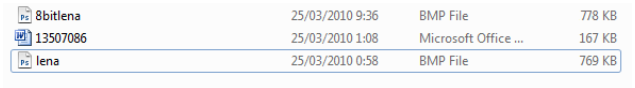


Gambar 12 - gambar asli 24 bit



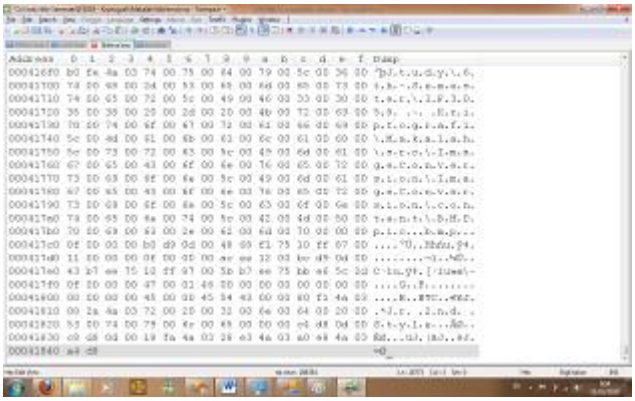
Gambar 13 - lena.bmp dalam 8 bit

Terlihat gambar lena.bmp mengalami penurunan kualitas. Gambar 8 bit lena.bmp akan memiliki size yang tidak jauh dengan 24 bit lena.bmp karena harus menyimpan informasi offset.



Gambar 14 - besar file lena.bmp

Berikut adalah isi footer 8bitlena.bmp yang berisi offset dari lena.bmp yang tidak terwakili oleh palette.



Gambar 15 - footer 8bitlena.bmp

Penyembunyian Informasi

Proses penyembunyian informasi yang dilakukan oleh program konversi merupakan penerapan dari prinsip yang ada pada bab sebelumnya yaitu *paletting* dan mengestimasi kecocokkan pixel dengan warna pada palette dan sisa offsetnya akan disimpan di akhir file secara sekuens. Berikut adalah potongan dari program ImageConversion.cs untuk melakukan

1. Paletting

```
byte[] STANDARD_PALLETE = { 00, 51, 102, 153, 204, 255 };
const int STANDARD_COLOR_SIZE = 6;
const int STANDARD_PALLETE_VAL_DIF = 51;
int[] dwColorMapTable = new int[216];
```

```
private void makePallete()
{
    int  nColorMapIdx = 0;
    for (int nBlueIdx = 0; nBlueIdx < STANDARD_COLOR_SIZE; ++nBlueIdx)
    {
        for(int nGreenIdx = 0; nGreenIdx < STANDARD_COLOR_SIZE; ++nGreenIdx)
        {
            for(int nRedIdx = 0; nRedIdx < STANDARD_COLOR_SIZE; ++nRedIdx)
            {
                byte[] objColor = { STANDARD_PALLETE[nRedIdx],
                                    STANDARD_PALLETE[nGreenIdx],
                                    STANDARD_PALLETE[nBlueIdx],
                                    00 };
                dwColorMapTable[nColorMapIdx] = BitConverter.ToInt32(objColor, 0);
                ++nColorMapIdx;
            }
        }
    }
}
```

2. Pencocokkan informasi

```
int GetPixelValue(int uPixelValue_i)
{
    int uRetValue = 0;
    int uPos = uPixelValue_i / STANDARD_PALLETE_VAL_DIF;
    if (0 == uPixelValue_i % STANDARD_PALLETE_VAL_DIF)
    {
        uRetValue = uPixelValue_i / STANDARD_PALLETE_VAL_DIF;
    }
    else
    {
        if (Math.Abs(uPixelValue_i - STANDARD_PALLETE[uPos]) >
           Math.Abs(uPixelValue_i - STANDARD_PALLETE[uPos + 1]))
        {
            uRetValue = uPos + 1;
        }
        else
        {
            uRetValue = uPos;
        }
    }
    return uRetValue;
}
```

3. Penulisan ke file

```
int iter = 0;
for (int i = index; i < lastpixel; i += 3)
{
    int uBlueValue = GetPixelValue(imageData[i]);
    int uGreenValue = GetPixelValue(imageData[i + 1]);
    int uRedValue = GetPixelValue(imageData[i + 2]);

    //Console.WriteLine(uRedValue + "-" + uGreenValue + "-" + uBlueValue);

    int uPalletePos = uBlueValue * 36 + uGreenValue * 6 + uRedValue;
    pixels[iter] = (byte)uPalletePos;
    iter++;
}

Bitmap bmp = new Bitmap(width, height, PixelFormat.Format8bppIndexed);
BitmapData bmpData = bmp.LockBits(
                     new Rectangle(0, 0, bmp.Width, bmp.Height),
                     ImageLockMode.WriteOnly, bmp.PixelFormat);
//Copy the data from the byte array into BitmapData.Scan0
Marshal.Copy(pixels, 0, bmpData.Scan0, pixels.Length);
//Unlock the pixels
bmp.UnlockBits(bmpData);
imageData = imageToByteArray(bmp, ImageFormat.Bmp);
```

Pengembalian Informasi

Setelah gambar 24 bit berhasil disembunyikan pada gambar 8 bit, langkah yang harus bisa dilakukan berikutnya adalah mengambil kembali gambar 24 bit dengan kualitas yang sama dengan sebelum dimasukkan ke dalam gambar 8 bitnya. Untuk melakukan hal ini ada beberapa tahapan yang harus dilakukan, yaitu:

1. Membaca header

   Saat membaca header, ada beberapa informasi yang harus disimpan untuk kepentingan mengkonversi balik gambar 24 bit yang telah disimpan, yaitu awal index pixel, tinggi gambar, lebar gambar, dan *palette* yang digunakan oleh gambar. Adapula informasi yang diambil berada pada:

   `0Ah` = awal index pixel

   `12h` = lebar bitmap

   `16h` = tinggi bitmap

   `36h` = palette

2. Mengkonversi per pixel

```
private int convert8bitTo32bit(byte i8bit)
{
    //mengambil warna dari palette
    int color = dwColorMapTable[i8bit];

    //mengambil offset
    //offsetIndex merupakan variable global yang berisi index awal offset
    int offset = imageData[offsetIndex];
    byte iRedColor = (byte)(getRed(color) + getRed(offset));
    byte iGreenColor = (byte)(getGreen(color) + getGreen(offset));
    byte iBlueColor = (byte)(getBlue(color) + getBlue(offset));

    byte[] i32bit = { iRedColor, iGreenColor, iBlueColor };
    int o32bit = BitConverter.ToInt32(i32bit, 0);

    return o32bit;
}
```

Gambar 16 - fungsi konversi pixel dari 8 bit ke 24 bit

Konversi akan dilakukan terhadap semua pixel pada gambar dan akan disimpan pad suatu array of integer untuk kemudian dituliskan kembali ke sebuah file eksternal bmp 24 bit.

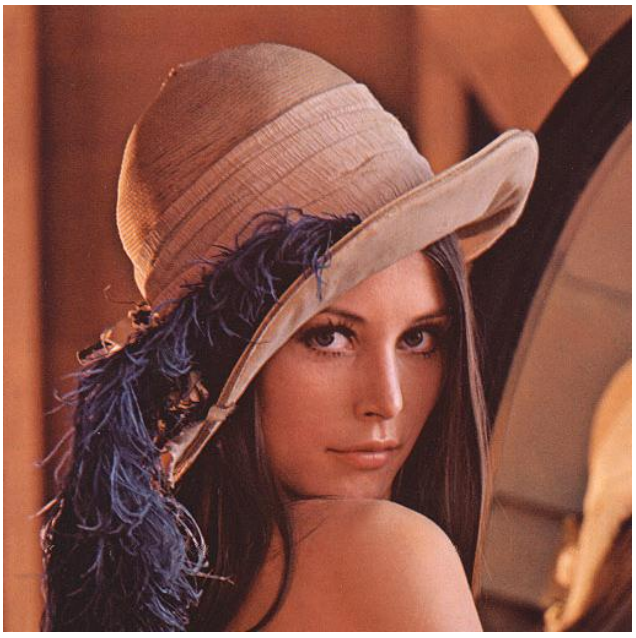
3. Menuliskan kembali file citra ke dalam sebuah citra bmp 24 bit dengan format penulisan sama dengan sebelum disembunyikan. Karena menggunakan penambahan offset maka dapat dipastikan bahwa konversi in tidak akan mengakibatkan kerusakan gambar atau bersifat loseless. Berikut adalah hasil konversi balik dari gambar lena 8bit bmp ke 24bit bmp. Pada program tidak dicantumkan heksadesimalnya karena terlalu panjang dan membuat program lama untuk berjalan hanya untuk menuliskannya.

Gambar 17 - lena24bit hasil steganalis lena8bit

Sekian implementasi dari konversi citra 24 bit menjadi citra 8 bit dan pengembaliannya yang telah penulis coba untuk membuat, hasil citra yang dikembalikan akan memiliki ukuran yang sama dengan citra yang belum didekripsi ini merupakan prinsip dasar steganografi yaitu pesan yang disembunyikan tidak boleh hilang.

Berikut adalah perbandingan citra 24 bit yang dihasilkan program konversi biasa dibandingkan dengan program konversi yang dibuat oleh penulis dengan metode di atas:

Gambar 18 - lena 24 bit hasil konversi konvensional



Gambar 19 - lena 24 bit hasil konversi dengan program yang dibuat

BAB IV – KESIMPULAN

Meyimpan sebuah file yang lebih besar ke dalam file yang lebih kecil adalah sebuah tantangan besar yang patut diperhitungkan dalam dunia Informatika, apalagi apabila pengaturan kualitas itu dapat menekan jumlah pembajakan terhadap kekayaan ilmu pengetahuan. Steganografi atau ilmu menyembunyikan informasi merupakan salah satu jawaban atas pengrahasiaan informasi citra yang menarik, dalam hal ini berformat 24 bit menjadi citra dengan kualitas lebih rendah berformat 8 bit.

Namun, cara yang saya coba terapkan di sini masih sangat sederhana dan tidak terlalu sulit untuk di-*steganalisis* yang berujung pada pembajakan kembali. Hal ini lah yang perlu diperhatikan dan dapat dikembangkan oleh para informatikawan di masa mendatang. Isu selain itu adalah ukuran gambar, dalam penyembunyian informasi 24 bit pada percobaan ini citra 24 bit yang telah disembunyikan dapat dikenali dengan mudah karena mengakibatkan besarnya ukuran citra 8 bit, pada kelanjutannya diharapkan dapat ditemukan sebuah metode untuk mengurangi ukuran citra tersebut sehingga menyerupai citra 8 bit yang tidak diisikan citra 24 bit.

Bersamaan dengan akhir dari tulisan ini, penulis mengucapkan terima kasih sebesarnya terhadap bantuan dari segala pihak baik yang dapat dituliskan pada referensi maupun tidak. Sekiranya ada kekurangan pada tulisan tersebut harap dimaklumi dan dapat dikembangkan, terima kasih.

REFERENSI

[1] H. Kathryn, "A Java Steganography Tool," http://diit.sourceforge.net/files/Proposal.pdf

[2] K.S. Shanu, " Image Bit depth conversion from 32 Bit to 8 Bit," 2010, http://www.codeproject.com/KB/graphics/Image_Bitdepth_Conversion.aspx

[3] Munir, Rinaldi, "Diktat Kuliah Kriptografi," Penerbit ITB, 2005.

[4] N.F. Johnson, J. Suhil, " Exploring Steganography:Seeing the Unseen," Computing practices, 2006, 2006, http://www.jjtc.com/pub/r2026.pdf