Penyembunyian Pesan di dalam Berkas Citra dengan Metode BPCS (*Bit-Plane Complexity Segmentation*)

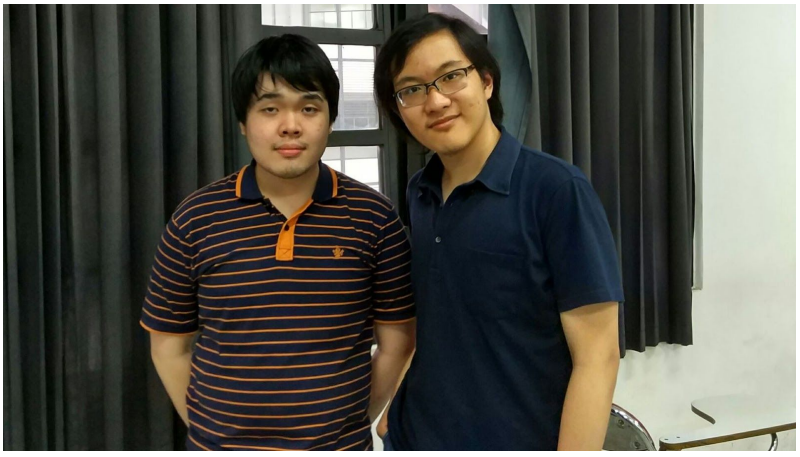
LAPORAN TUGAS BESAR 1

Diajukan untuk memenuhi tugas besar 1 mata kuliah IF4020 Kriptografi pada Semester II tahun Akademik 2017-2018

oleh

Edwin Rachman **NIM 13515042**

Felix Limanta **NIM 13515065**



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2018

1. Teori Dasar

1.1. Steganografi

Steganografi adalah ilmu dan seni menyembunyikan pesan rahasia dengan suatu cara sedemikian sehingga tidak seorang pun yang mencurigai keberadaan pesan tersebut. Steganografi digital adalah penyimpanan pesan digital di dalam dokumen digital lainnya. Dokumen yang digunakan sebagai media penyimpanan disebut sebagai *cover-object*, yang dapat berupa citra, teks, audio, video, dll. Pesan yang disimpan disebut sebagai *embedded message*, yang dapat berupa citra, teks, audio, video, dll. Dokumen yang sudah berisi pesan disebut sebagai *stego-object*. Kunci yang digunakan untuk menyimpan dan mengekstrak pesan disebut sebagai *stego-key*.

1.2. Metode BPCS (*Bit-Plane Complexity Segmentation*)

Metode BPCS, *Bit-Plane Complexity Segmentation*, merupakan metode steganografi berkapasitas besar, lebih besar daripada menggunakan metode modifikasi LSB (*Leasr Significant Bit*). Jika metode modifikasi LSB hanya dapat menyisipkan pada satu atau beberapa bit LSB, maka metode BPCS menyisipkan pada satu atau beberapa *bit-plane*. *Bit-plane* adalah citra biner yang berisi bit ke-i dari pixel-pixel dalam suatu citra. Pada citra *grayscale* terdapat 8 *bit-plane*, melainkan pada citra *true color* terdapat 24 *bit-plane*.

Pada metode BPCS, citra terbagi menjadi blok-blok berukuran 8x8 pixel. Setiap blok pixel-pixel dicek kompleksitasnya untuk menentukan apakah blok tersebut merupakan *noise-like region* (kompleksitas tinggi) atau *informative region* (kompleksitas rendah). Penyisipan pesan (lebih baik terlebih dahulu diubah dari *Pure Binary Coding* menjadi *Canonical Gray Coding* supaya terlihat lebih kompleks) kemudian dilakukan pada setiap *noise-like region*, sehingga bit yang disisipkan jauh lebih banyak dibandingkan pada metode modifikasi LSB (bisa sampai 50% dari ukuran *cover-image*). Untuk meningkatkan lebih lanjut kapasitas dari *cover-image*, dapat juga dilakukan konjugasi terhadap blok-blok *informative region* untuk meningkatkan tingkat kompleksitasnya.

1.3. Vigenere Cipher

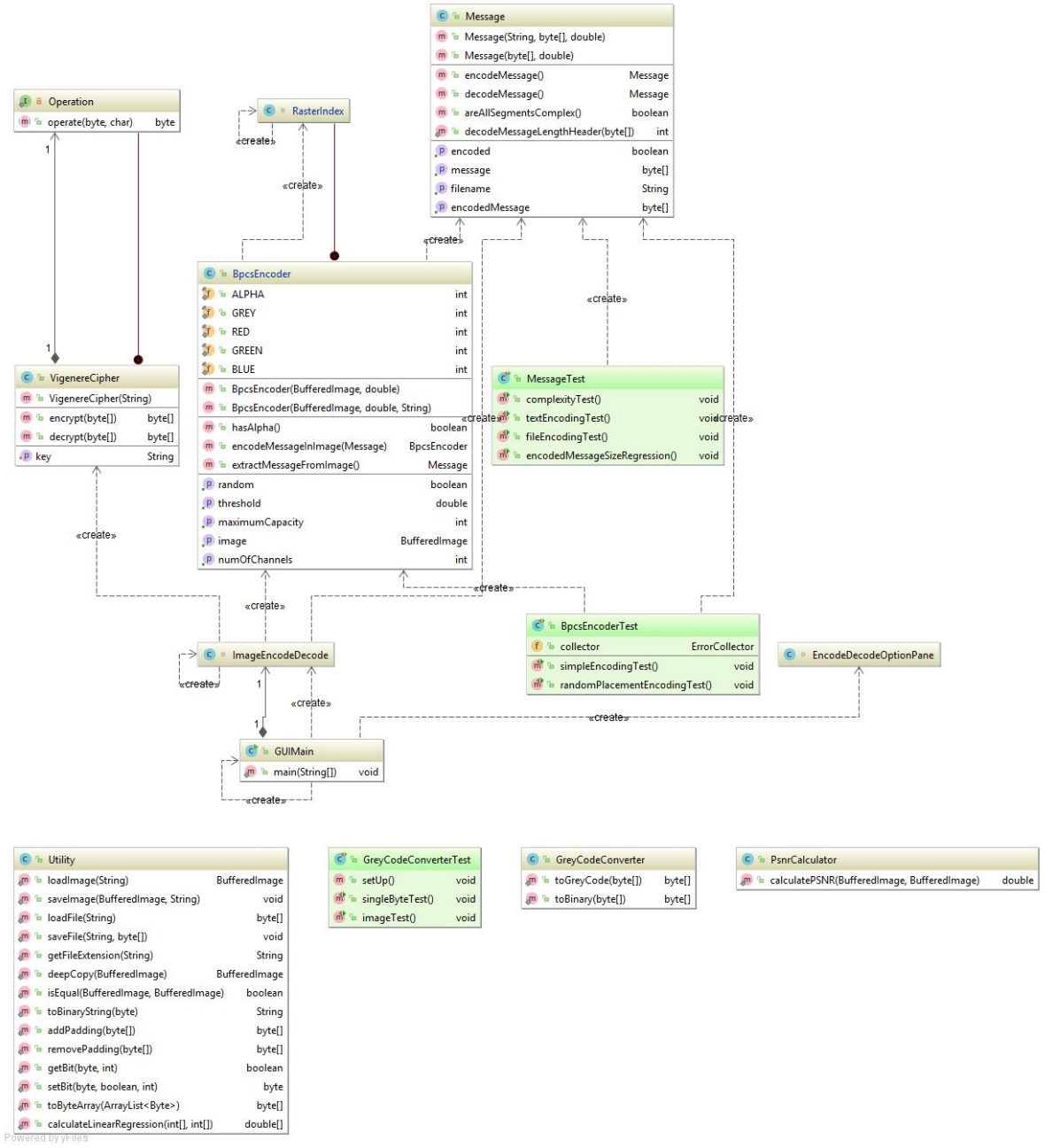
Vigenere cipher adalah sebuah metode enkripsi teks yang menggunakan prinsip substitusi polialfabetik. Pada dasarnya Vigenere cipher mengenkripsi teks dengan menggunakan Caeser cipher yang terjalin berdasarkan huruf-huruf pada keywordnya. Oleh karena itu, suatu huruf pada plainteks tidak selalu terenkripsi menjadi huruf yang sama pada cipherteks yang dihasilkan. Vigenere cipher dan cipher-cipher lain

sering digunakan untuk mengenkripsi pesan sebelum disisipkan menggunakan steganografi.

1.4. Citra bitmap

Citra bitmap merupakan salah satu cara untuk menyimpan suatu citra secara digital, yaitu dalam bentuk array pixel-pixel dimana setiap pixel menyimpan nilai warna citra pada posisi yang sama dengan pixel tersebut. Hal ini berbeda dengan citra vector dimana yang disimpan adalah garis-garis, lengkungan, dll yang mendeskripsikan citranya. Tipe file citra yang menggunakan bitmap adalah tipe file BMP (Bitmap) dan PNG (Portable Network Graphics). File BMP bersifat lossless dan tidak terkompresi, melainkan file PNG bersifat lossless juga, tapi terkompresi.

2. Perancangan dan Implementasi



Implementasi algoritma sebagian besar dibagi dalam 2 kelas: Message dan PsnrEncoder.

- Kelas Message bertanggung jawab untuk mengenkode dan mendekode pesan yang akan disisipkan menjadi bentuk yang dapat disisipkan pada citra. Operasi yang termasuk pada kelas ini adalah penambahan dan *parsing header* (ukuran, peta konjugasi, dan nama berkas), perhitungan kompleksitas pesan, serta konjugasi pesan yang kurang kompleks.
- Kelas PsnrEncoder bertanggung jawab untuk menyisipkan dan mengekstrak pesan yang telah dienkode sebelumnya pada citra. Operasi yang termasuk pada kelas ini adalah perhitungan kompleksitas tiap blok citra 8x8 piksel, perhitungan kapasitas maksimum citra untuk

threshold tertentu, dan penyisipan pesan (dalam blok terurut dari kiri atas atau blok semi-acak sesuai kunci yang dimasukkan).

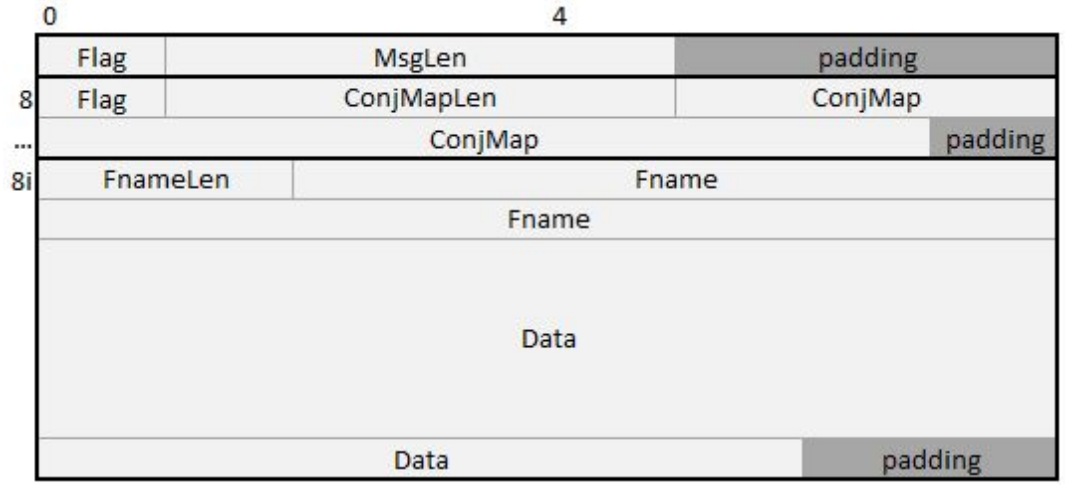
Selain dua kelas utama tersebut, terdapat pula kelas-kelas

- Kelas VigenereCipher bertanggung jawab untuk enkripsi dan dekripsi data.
- Kelas PsnrCalculator bertanggung jawab menghitung nilai PSNR antara dua citra.
- Kelas GreyCodeConverter bertanggung jawab mengkonversi data biner ke *grey code* dan sebaliknya.
- Kelas GUIForm bertanggung jawab atas GUI (Swing) dari program utama.
- Kelas EncodeDecodeOptionPane bertanggung jawab atas GUI option dialog saat akan melakukan encoding atau decoding.
- Kelas ImageEncodeDecode menjembatani GUI dengan logika program.
- Kelas Utility mengandung sejumlah fungsi lain-lain yang sering dipakai.

Kedua kelas utama, beserta kelas GreyCodeConverter, juga dibuat kelas pengujiannya.

Perlu diperhatikan bahwa citra .png dan .bmp tidak memiliki penanganan yang khusus. Kedua format citra diakses dengan *library* ImageIO milik Java agar program lebih fokus pada implementasi BPCS ketimbang akses format citra. Sebagai konsekuensi, program dapat membuka citra dengan format selain yang dispesifikasikan, seperti .jpg.

Message



Message dienkode dalam blok 8 byte, sehingga bagian yang panjangnya kurang dari 8 byte akan diberi *padding*.

Message dibagi dalam 3 bagian:

1. Blok pertama berisi panjang pesan. Bagian ini dipisah untuk memfasilitasi ekstraksi pesan dari citra.
    - Byte pertama dibiarkan kosong sebagai *flag* konjugasi.
    - 4 byte berikutnya berisi panjang pesan.
    - 3 byte sisanya kosong.
2. Beberapa blok berikutnya berisi peta konjugasi. Bagian ini dipisah untuk memfasilitasi pemulihan pesan setelah konjugasi.
    - Byte pertama dibiarkan kosong sebagai *flag* konjugasi.

- 4 byte berikutnya berisi jumlah blok pada bagian selanjutnya.
- Sisa dari bagian ini berisi peta konjugasi. Setiap blok 8 byte berisi 63 bit peta konjugasi.
- Ditambah *padding* jika panjang bagian kurang dari kelipatan 8 byte.

3. Blok selebihnya mengandung informasi pesan.
    - 2 byte pertama berisi panjang nama berkas.
    - Sampai 256 byte berikutnya berisi nama berkas.
    - Data sebenarnya dimulai setelah nama berkas.
    - Ditambah *padding* jika panjang bagian kurang dari kelipatan 8 byte.

Peta Konjugasi

Peta konjugasi disimpan dalam bentuk *array of boolean*, di mana anggota ke-i bernilai benar jika pada bagian pesan yang disisipkan, blok ke-i dikonjugasi. Ini diperlukan karena tidak ada ruang untuk meletakkan *flag* konjugasi pada data pesan sebenarnya.

Peta tersebut kemudian dienkode dalam bentuk biner pada bagian kedua pesan, setelah panjang peta konjugasi dari byte ke-5. Setiap blok 8 byte berisi 63 bit peta konjugasi. Ini disebabkan bit ke-0 pada byte pertama setiap blok dibiarkan kosong (bernilai 0) sebagai *flag* konjugasi. Jika blok peta konjugasi juga ikut dikonjugasi, *flag* tersebut akan berubah menjadi 1, sehingga saat pesan didekode diketahui blok mana dari *header* yang dikonjugasi.

BpcsEncoder

Segmentasi Citra

Pada saat konstruksi objek, langkah pertama yang dilakukan adalah mencatat kompleksitas setiap *bitplane* dari setiap *channel* dari setiap blok 8x8 pixel pada gambar.

Perhitungan bekerja langsung pada data pixel pada citra. Karena data pixel pada objek BufferedImage berupa *array of byte* linear, parameter-parameter posisi X, posisi Y, *channel* yang diakses, serta *bitplane* yang diakses perlu dikonversi ke bentuk linear.

Untuk selanjutnya, semua operasi akan langsung dilakukan pada data pixel penuh dari citra.

Perhitungan Kapasitas Citra

Setelah selesai menghitung kompleksitas, kapasitas citra dapat diketahui dari jumlah *byte* dalam *bitplane* yang kompleksitasnya berada di atas *threshold*. Dari kapasitas tersebut, ukuran berkas maksimum yang dapat disisipkan adalah sebagai berikut.

$$max.filesize = 0.984375 \times no.of\ bytes - 278.076955$$

Penyisipan Pesan

Jika pesan yang disisipkan berukuran lebih besar dari ukuran maksimum atau pesan memiliki blok yang kompleksitasnya di bawah *threshold*, pesan tidak akan disisipkan.

Penyisipan dilakukan blok pesan per blok pesan. Jika memilih menyisipkan secara sekuensial, maka semua bitplane akan ditelusuri dari awal. Jika bitplane tersebut memiliki kompleksitas yang cukup, semua data pada bitplane tersebut akan diganti dengan data dari pesan.

Jika memilih menyisipkan secara acak, bitplane untuk disisipkan berikutnya didapat dari pembangkit bilangan acak semu dengan *seed* nilai *hash* dari kunci yang diberi. Nilai yang sudah dibangkitkan dikonversi menjadi parameter posisi X, posisi Y, *channel*, dan *bitplane*. Agar tidak terjadi pengulangan, nilai yang sudah dibangkitkan disimpan dalam *set*, di mana bilangan yang sudah ada dalam *set* tidak boleh dipakai lagi.

Ekstraksi Pesan

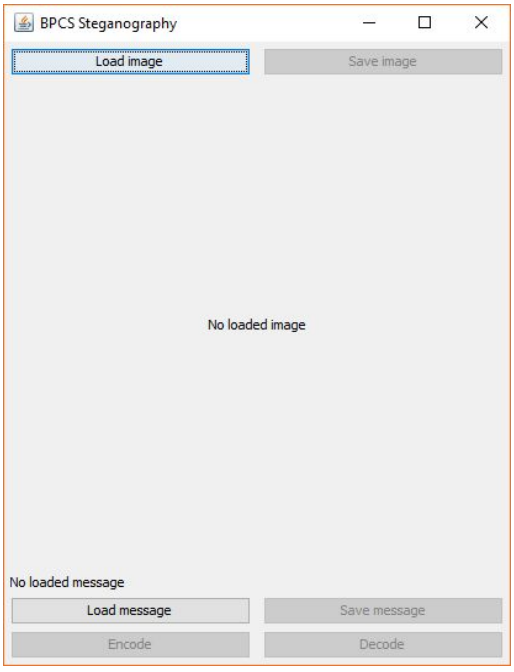
Bitplane pertama yang disisipkan pesan pada tahap sebelumnya berisi informasi mengenai panjang pesan. Informasi tersebut diekstrak, kemudian didekode agar panjang pesan yang akan diekstrak diketahui.

Setelah panjang pesan diketahui, proses ekstraksi dilakukan dengan menelusuri semua *bitplane* yang disisipi pesan, lalu mengekstrak pesan dari *bitplane* tersebut. Karena *bitplane* yang tidak kompleks tidak akan disisipi pesan, penelusuran sekuensial dijamin akan menelusuri jalur yang sama seperti saat penyisipan. Penggunaan *seed* pada pembangkit bilangan acak semu menjamin penelusuran acak menghasilkan jalur yang sama.

Program tidak mengetahui apakah gambar yang akan diekstraksi benar-benar mengandung pesan, sehingga ekstraksi pesan dari gambar yang tidak disisipi pesan akan gagal. Selain itu, jika parameter ekstraksi salah (penelusuran sekuensial/acak, kunci, dll.), ekstraksi juga tidak mungkin dilakukan.

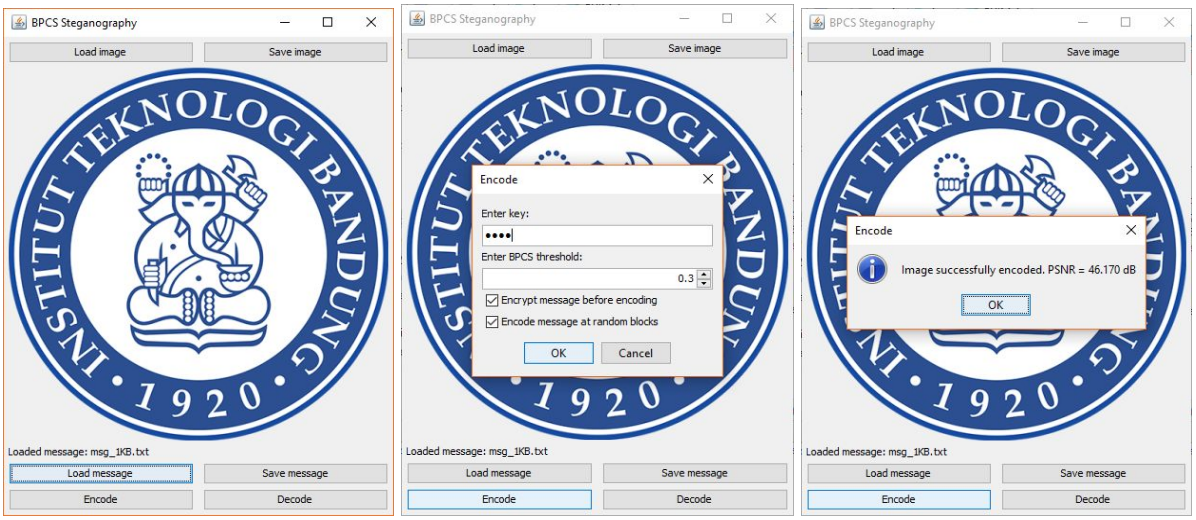
GUI Program

Tampilan awal program



Tampilan awal program. Tombol ‘Load image’ digunakan untuk membuka sebuah citra (dengan tipe file BMP atau PNG) untuk diproses. Tombol ‘Load message’ digunakan untuk membuka pesan (tipe file apapun) yang akan disisipkan.

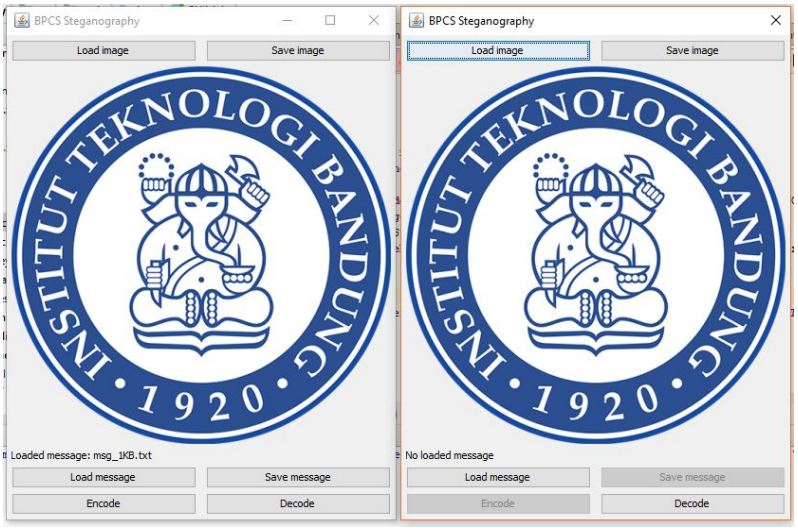
Tampilan proses *encoding* citra



Gambar di kiri adalah tampilan program setelah cover-image dan pesan diload. Di tengah ditampilkan cover-image, dan dibawahnya ditampilkan filename dari pesan yang akan disisipkan. Tombol ‘Encode’ digunakan untuk melakukan proses *encoding* steganografi pesan yang diload ke dalam cover-image.

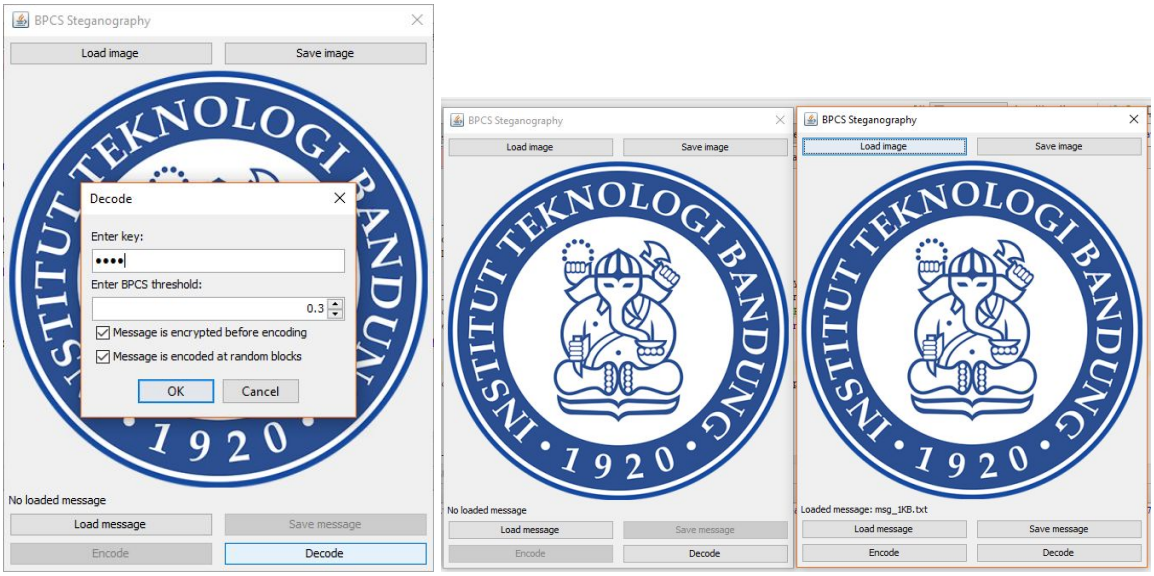
Gambar di tengah adalah tampilan dialog Encode dimana pengguna dapat menspesifikasikan kunci dan threshold yang akan digunakan. Pengguna dapat memilih ‘Encrypt message before encoding’ jika pengguna ingin pesan dienkripsi terlebih dahulu dengan Vigenere cipher menggunakan key yang dimasukkan. Pengguna juga dapat memilih ‘Encode message at random blocks’ jika pengguna ingin penyisipan pesan pada cover-image dilakukan secara random tergantung key yang dimasukkan.

Gambar di tengah adalah tampilan message box yang menyatakan gambar berhasil diencoding dengan nilai PSNR tertentu.



Tampilan citra asli dan citra stegano dalam dua jendela berbeda. Pada jendela citra stegano terdapat tombol ‘Decode’ yang digunakan untuk melakukan decode steganografi terhadap citra tersebut.

Tampilan proses *decoding* citra



Gambar di kiri adalah tampilan dialog Decode dimana pengguna harus memasukkan setiap parameter sama dengan saat encoding supaya dapat didecode kembali ke citra asli.

Gambar di kanan adalah tampilan citra stegano dan citra asli hasil decoding dalam dua jendela berbeda.

3. Pengujian Program dan Analisis Hasil

3.1. Citra homogen

Citra homogen *grayscale* dengan *alpha*

Ukuran citra: 512 x 512 px

Ekstensi: .png

Jumlah *channel*: 2

Ukuran	***Threshold***	**Kapasitas (B)**	**PSNR (dB)**	**Waktu (ms)**
128B	0.1	113775	68.698128	704
	0.2	61060	62.71539	797
	0.3	28260	65.942404	609
	0.4	10683	70.873435	560
	0.5	1517	71.611183	719
256B	0.1	113775	53.887291	717
	0.2	61060	60.840026	714
	0.3	28260	64.037581	548
	0.4	10683	66.877375	646
	0.5	1517	69.363851	698
1KB	0.1	113775	44.979525	900
	0.2	61060	56.272379	674
	0.3	28260	59.236157	510
	0.4	10683	61.370861	654

	0.5	1517	63.105855	760
16KB	0.1	113775	27.281521	725
	0.2	61060	36.898311	741
	0.3	28260	38.462841	768
	0.4	10683		
	0.5	1517		

Citra homogen RGB dengan *alpha*

Ukuran citra: 512 x 512 px

Ekstensi: .png

Jumlah *channel*: 4

Ukuran	***Threshold***	**Kapasitas (B)**	**PSNR (dB)**	**Waktu (ms)**
128B	0.1	333047	60.567697	502
	0.2	186327	58.575178	694
	0.3	62068	58.827698	583
	0.4	12329	61.048872	627
	0.5	265	57.593995	641
256B	0.1	333047	58.368801	901
	0.2	186327	54.274273	563
	0.3	62068	56.503551	628
	0.4	12329	61.102875	644

	0.5	265		
1KB	0.1	333047	45.780889	595
	0.2	186327	50.865067	747
	0.3	62068	53.370823	495
	0.4	12329	54.655798	652
	0.5	265		
16KB	0.1	333047	33.084231	755
	0.2	186327	37.074647	647
	0.3	62068	36.553335	706
	0.4	12329		
	0.5	265		

Citra homogen RGB tanpa *alpha*

Ukuran citra: 1024 x 1024 px

Ekstensi: .bmp

Jumlah *channel*: 3

Ukuran	***Threshold***	**Kapasitas (B)**	**PSNR (dB)**	**Waktu (ms)**
128B	0.1	583999	62.275174	1592
	0.2	360767	62.689838	1456
	0.3	225443	64.576716	1423
	0.4	137699	70.502338	1443
	0.5	42365	82.65371	1571

256B	0.1	583999	61.582823	1426
	0.2	360767	61.18916	1411
	0.3	225443	64.940128	1339
	0.4	137699	68.77943	1429
	0.5	42365	79.549349	1557
1KB	0.1	583999	55.965564	1438
	0.2	360767	57.859788	1198
	0.3	225443	59.840126	1209
	0.4	137699	67.126843	1549
	0.5	42365	67.026902	1382
16KB	0.1	583999	45.477295	1457
	0.2	360767	47.749304	1556
	0.3	225443	50.399573	1657
	0.4	137699	51.209231	1543
	0.5	42365	51.673465	1655

3.2. Citra heterogen

Citra *greyscale* tanpa *alpha*

Ukuran citra: 512 x 512 px

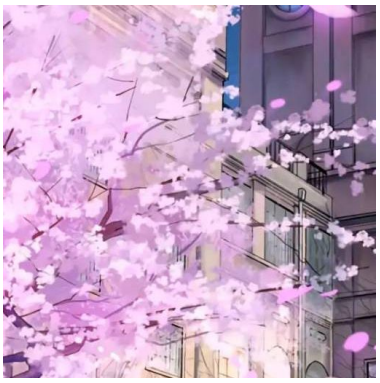
Ekstensi: .png

Jumlah *channel*: 1

Ukuran	***Threshold***	**Kapasitas (B)**	**PSNR (dB)**	**Waktu (ms)**

128B	0.1	181524	53.333327	248
	0.2	149386	54.947361	182
	0.3	124099	59.36145	164
	0.4	96316	64.406015	204
	0.5	33773	63.551788	189
256B	0.1	181524	47.00736	183
	0.2	149386	52.112737	186
	0.3	124099	58.365937	178
	0.4	96316	57.697114	187
	0.5	33773	56.576889	195
1KB	0.1	181524	40.291548	180
	0.2	149386	44.61515	197
	0.3	124099	47.950748	180
	0.4	96316	48.850143	188
	0.5	33773	52.651149	181
16KB	0.1	181524	28.000053	182
	0.2	149386	32.317273	203
	0.3	124099	35.96756	200
	0.4	96316	38.776479	186
	0.5	33773	39.853151	201

Citra RGB tanpa *alpha*



Ukuran citra: 512 x 512 px

Ekstensi: .png

Jumlah *channel*: 3

Ukuran	*Threshold*	Kapasitas (B)	PSNR (dB)	Waktu (ms)
128B	0.1	537316	66.574132	472
	0.2	443997	67.832334	492
	0.3	366720	68.51863	588
	0.4	281316	68.842551	566
	0.5	97490	71.064052	461
256B	0.1	537316	59.368948	406
	0.2	443997	63.944538	607
	0.3	366720	60.088497	517
	0.4	281316	58.947146	481
	0.5	97490	68.202693	549
1KB	0.1	537316	44.636352	537
	0.2	443997	50.117983	391
	0.3	366720	50.706938	682
	0.4	281316	55.192031	486
	0.5	97490	58.208936	494
16KB	0.1	537316	31.20949	1227
	0.2	443997	35.673192	525
	0.3	366720	39.14485	666
	0.4	281316	42.70296	511
	0.5	97490	46.926966	515

Citra RGB berukuran besar tanpa *alpha*

Ukuran citra: 1920 x 1080 px

Ekstensi: .png

Jumlah *channel*: 3

Ukuran	*Threshold*	Kapasitas (B)	PSNR (dB)	Waktu (ms)
128B	0.1	3737039	59.46754	4840
	0.2	2987756	59.46831	4663
	0.3	2402604	64.65841	4603
	0.4	1812720	71.24882	4247
	0.5	596576	74.75862	4241
256B	0.1	3737039	57.86588	4784
	0.2	2987756	59.52498	5157
	0.3	2402604	64.03178	4899
	0.4	1812720	68.6692	4640
	0.5	596576	74.67296	4623
1KB	0.1	3737039	52.55125	5241
	0.2	2987756	55.2118	5219
	0.3	2402604	59.96909	4662
	0.4	1812720	65.56806	4926
	0.5	596576	72.21502	4976
16KB	0.1	3737039	45.62057	4903

	0.2	2987756	49.11947	4562
	0.3	2402604	52.52695	4751
	0.4	1812720	57.19516	4834
	0.5	596576	61.2292	4601
725KB	0.1	3737039	29.30362	5262
	0.2	2987756	33.94001	5068
	0.3	2402604	37.77539	5361
	0.4	1812720	40.39809	5201
	0.5	596576		

Analisis

Nilai PSNR meningkat dengan meningkatnya nilai *threshold*, tetapi kapasitas menurun drastis dengan peningkatan nilai *threshold*. Citra yang disisipkan dengan *threshold* 0.5 hanya dapat menanggung, rata-rata, kurang dari 1% jumlah *pixel* yang ada pada citra.

Waktu eksekusi program tidak terlalu dipengaruhi *threshold* jika penelusuran berjalan sekuensial, tetapi jika penelusuran berjalan acak waktu eksekusi dapat meningkat drastis. Ini disebabkan *bitplane* pada bilangan acak yang dibangkitkan belum tentu memenuhi syarat penyisipan, sehingga proses pencarian memakan waktu yang lebih lama. Selain itu, waktu eksekusi juga cenderung meningkat seiring ukuran pesan, ukuran citra, dan jumlah *channel* meningkat.

Sebagian kasus uji dengan ukuran pesan dan *threshold* rendah, tetapi memiliki nilai PSNR yang tinggi. Ini disebabkan nilai PSNR dihitung dari seluruh gambar, sedangkan yang diubah hanya sebagian. Meskipun demikian, jika dilihat dengan mata, artefak dari penyisipan dengan *threshold* rendah sangat mencolok, sehingga PSNR tidak dapat terlalu dijadikan sebagai tolak ukur.

Penyisipan berkas 128 byte dengan *threshold* 0.1

Terdapat pula kasus di mana peningkatan *threshold* justru mengakibatkan penurunan nilai PSNR. Ini disebabkan secara kompleksitas *bitplane* memenuhi syarat, tetapi nilai *bitplane* awal dan pesan yang disisipkan, jika direpresentasikan sebagai bilangan, berbeda lebih jauh, sehingga derau yang dihasilkan lebih mencolok.

4. Kesimpulan Hasil Implementasi

Menurut hasil implementasi metode BPCS Steganography yang kami telah buat, dapat disimpulkan bahwa memang kapasitas penyisipan pesan jauh lebih besar (hingga lebih dari 50%) dibandingkan metode modifikasi LSB. Akan tetapi algoritmanya lebih kompleks sehingga lebih sulit untuk diimplementasikan dibandingkan metode modifikasi LSB yang relatif lebih sederhana.

Selain itu, terdapat kemungkinan besar terjadi kerusakan atau kehilangan informasi yang disisipkan pada citra stego yang dihasilkan jika implementasi yang digunakan kurang benar. Jika hal ini terjadi, maka citra stego tersebut tidak dapat dikembalikan ke citra aslinya.

5. Pembagian Tugas

Edwin Rachman (NIM 13515042)

- Desain dan Implementasi kelas EncodeDecodeOptionPane
- Desain dan Implementasi kelas ImageEncodeDecode
- Desain dan Implementasi kelas GUIMain
- Integrasi GUI program dengan algoritma
- Laporan

Felix Limanta (NIM 13515065)

- Desain, Implementasi, dan Pengujian kelas BpcsEncoder
- Desain, Implementasi, dan Pengujian kelas GreyCodeConverter
- Desain, Implementasi, dan Pengujian kelas Message
- Desain dan Implementasi kelas PsnrCalculator
- Desain dan Implementasi kelas Utility
- Desain dan Implementasi kelas VigenereCipher
- Laporan