Implementasi *Direct Sequence Spread Spectrum* Steganography pada Data *Audio*

Rizky Maulana Nugraha – 13508083[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]r_maulana_n@students.itb.ac.id

***Abstrak*— Steganografi pada data citra sudah banyak berkembang. Algoritma-algoritma yang dikembangkan juga bermacam-macam. Sementara, ketertarikan menggunakan data *audio* sebagai sarana steganografi bisa dibilang terlambat munculnya dibanding data citra. Makalah ini membahas implementasi steganografi pada data *audio* menggunakan metode *Direct sequence spread spectrum*. Metode *Spread Spectrum* sering digunakan untuk mengirimkan pesan tersembunyi melalui gelombang radio. Pesan ini ditransmisikan melalui gelombang yang menyerupai derau. Metode yang sama bisa diaplikasikan untuk menyisipkan pesan pada data *audio*. Pesan yang disisipkan pada data *audio* ini akan terdengar seperti sebuah derau. Metode *Spread Spectrum* yang digunakan adalah *Direct sequence spread spectrum*. Sebuah kunci dibutuhkan untuk menyisipkan pesan pada derau, kunci ini digunakan untuk membangkitkan gelombang derau semu. Informasi yang akan disisipkan harus terlebih dahulu dimodulasi menggunakan derau semu tersebut. Makalah ini membahas implementasi metode tersebut pada data *audio* untuk menyembunyikan pesan teks. Metode *Spread Spectrum* terkenal sangat *robust,* namun sebagai konsekuensinya, *cost* penggunaannya cukup besar dan implementasinya relatif kompleks. Permasalahan tersebut juga akan dibahas dalam makalah ini.**

Kata Kunci—Steganography, Direct sequence spread spectrum.

I. PENDAHULUAN

Steganografi adalah seni atau studi menyembunyikan informasi dengan cara menyisipkan pesan rahasia dalam pesan lain. Medium tempat informasi disisipkan bisa berupa apa saja. Medium ini disebut *cover object*. Steganografi yang diterapkan untuk menyembunyikan informasi pada *cover object* digital disebut Steganografi digital. *Cover Object* yang digunakan pada steganografi digital bisa bermacam-macam, misalnya pada arsip citra. Algoritma steganografi pada arsip citra sudah banyak dibuat dan dikembangkan. Sedangkan, algoritma steganografi pada arsip suara relatif lebih sedikit. Makalah ini membahas penerapan steganografi digital pada arsip suara (*audio*) menggunakan metode *Direct-sequence Spread Spectrum*.

II. DASAR TEORI

Steganografi pada arsip suara tidak semudah pada arsip citra. Tidak seperti pada arsip citra mentah, arsip suara mentah biasanya lebih besar. Sebagai perbandingan, arsip citra mentah dengan jenis warna 24bit dan resolusi 1280x800 (resolusi standar layar desktop) memiliki ukuran data sekitar 3 MB. Sedangkan arsip suara mentah dengan frekuensi *sampling* 44,1 kHz 16 bit stereo dengan durasi 4 menit (durasi standar sebuah lagu) memiliki ukuran data sekitar 40 MB. Perbedaan data ini cukup besar, sehingga mengakibatkan implementasi steganografi pada data *audio* menjadi lebih sulit. Sebagai ilustrasi, andaikata kita menggunakan *Discrete Fourier Transform* untuk mengubah domain penyisipan data, maka arsip suara jelas membutuhkan *cost* yang cukup besar karena jumlah sampel yang harus ditransformasi jauh lebih banyak. Selain itu, andaikan kita menggunakan metode LSB, *noise* yang dihasilkan pada arsip suara lebih besar. Ini disebabkan karena range sinyal suara lebih rendah daripada sinyal *pixel*. Sinyal *pixel* dikodekan sebesar 24 bit, sedangkan sinyal suara dikodekan sebesar 15 bit (karena ada suara positif dan negatif). Selain itu, penggunaan arsip mentah suara (WAV) lebih jarang daripada arsip mentah citra (BMP), karena ukurannya yang besar. Pada subbab berikutnya, penulis akan menjelaskan beberapa teori dasar yang perlu diketahui terlebih dahulu.

A. Representasi Digital dari Suara

Suara, seperti halnya citra, merupakan representasi gelombang. Citra adalah representasi intensitas gelombang cahaya pada bidang datar 2D. Sedangkan arsip suara adalah representasi amplitudo gelombang suara pada domain waktu.

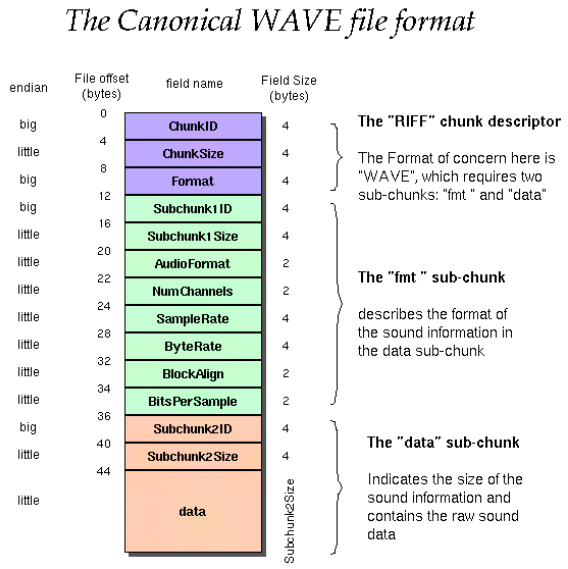
Suara diubah menjadi digital menggunakan Analog to Digital Converter (ADC). ADC mengkonversi suara pada titik waktu tertentu. Proses konversi pada titik ini atau saat dimana amplitudo gelombang ditangkap adalah proses *sampling*. Proses *sampling* diulang sebanyak frekuensi *sampling* yang dibutuhkan. Berdasarkan teorema Nyquist, frekuensi *sampling* harus minimal dua kali dari frekuensi suara yang ingin didengar. Sebagai contoh, frekuensi terbesar yang biasa diperdengarkan pada arsip suara

adalah 22050 Hz, sehingga frekuensi *sampling* minimalnya adalah 44100 Hz, yang berarti ada 44100 sample tiap detiknya. Misalkan proses *sampling*nya menggunakan *Pulse Code Modulation* (PCM), maka data *sampling* yang disimpan adalah amplitudonya.

Arsip mentah suara yang umum berformat Microsoft WAV 16bit PCM. 16bit PCM berarti untuk tiap data amplitudo dikodekan menjadi 16 bit signed integer. Jadi nilai amplitudo maksimal adalah 32768 dan nilai amplitudo minimal adalah -32768. Struktur data seperti ini sebenarnya masih memungkinkan penerapan metode LSB. Hanya saja metode LSB terlalu sederhana, selain itu jika *file* WAV dikonversi menjadi format lain, misalnya MP3, maka metode ini tidak efektif. Padahal pada praktiknya, *file* WAV sering dikonversi menjadi format lain untuk memperkecil ukuran *file*. Inilah salah satu alasan penggunaan metode *Spread Spectrum* pada arsip suara.

Arsip suara biasanya memiliki lebih dari satu *channel*, misalnya suara stereo memiliki *channel* kiri dan kanan. Maksud dari dua *channel* disini berarti ada dua gelombang suara yang disimpan, gelombang suara kiri dan kanan. Jika arsip suara stereo memiliki durasi 2 menit. Berarti arsip berisi suara *channel* kiri berdurasi 2 menit dan arsip suara *channel* kanan berdurasi 2 menit.

Aplikasi akan menyisipkan pesan pada *file* berformat WAV. Berikut adalah spesifikasi format *file* WAV:



B. Metode Spread Spectrum

Spread spectrum adalah teknik pembangkitan sinyal (elektrik, elektromagnetik atau akustik) yang dengan sengaja disebar pada rentang *bandwidth* yang lebih lebar dari yang seharusnya. Teknik ini dilakukan dengan berbagai alasan, diantaranya untuk jaringan komunikasi yang aman, memperkuat gelombang yang dikirim terhadap interferensi atau *jamming* dan menghindari pendeteksian.

Spread spectrum pada awalnya adalah teknik yang digunakan untuk komunikasi gelombang radio untuk alasan keamanan dan menghindari *jamming*. Sinyal radio yang dikirimkan sengaja disebar pada rentang frekuensi yang lebih lebar. Hasil sinyal radio yang ditangkap hanya terlihat sebagai *noise* biasa (*static noise*) dan tidak dapat diinterpretasi dengan cara biasa.

Spread spectrum memiliki kelebihan yang sangat penting, yaitu ketahanannya terhadap *jamming* dan interferensi. Andaikata sinyal yang dibuat mengalami kerusakan ditengah jalan, informasi yang disampaikan masih dapat dipersepsi. Sifat ini cocok digunakan untuk steganografi *audio* yang format *file*nya mungkin mengalami kompresi, terutama kompresi *lossy* seperti MP3.

Steganografi *spread spectrum* pada arsip *audio* ini akan diimplementasikan dengan skema sebagai berikut:

1. Mengubah data *audio cover-object* di *time*-domain ke *frequency*-domain
2. Menambahkan sinyal informasi dengan metode *spread-spectrum* ke *cover-object frequency*-domain
3. Mengubah lagi data *audio cover-object frequency*-domain ke *time*-domain

Dengan skema tersebut, data *audio cover-object* diharapkan lebih tahan dari proses kompresi, maupun manipulasi, karena informasi ditambahkan pada *frequency*-domain.

Metode *spread spectrum* memiliki beberapa jenis, sedangkan makalah ini akan membahas pengaplikasian metode *Direct-Sequence Spread Spectrum* (DSSS). *Direct-Sequence Spread Spectrum* adalah teknik *spread spectrum* yang menggunakan modulasi *Pseudo Noise Sequence* (PN *Sequence*). Teknik modulasi yang dimaksud memiliki skema sebagai berikut:

1. Siapkan sinyal informasi yang akan disebar.
2. Siapkan *Pseudo Noise Sequence* yang digunakan untuk memodulasi sinyal informasi
3. Modulasikan sinyal informasi dengan PN *Sequence*. Namun, frekuensi PN *Sequence* harus lebih besar (lebih cepat) dari frekuensi sinyal informasi.
4. Kirimkan hasil sinyal hasil modulasi
5. Penerima harus memiliki PN *Sequence* yang sama agar dapat mengerti pesan yang dikirimkan.
6. Modulasikan ulang pesan yang dikirimkan menggunakan PN *Sequence* yang sama.
7. Hasil modulasi adalah sinyal informasi yang dimaksud

Teknik modulasi ini akan dilakukan untuk menyembunyikan informasi pada *cover-object*. Namun, karena sinyal informasi pada representasi digital adalah bit *sequence* maka domain sinyal informasi kita ubah dari domain real ke domain bit, misalnya 1 dan 0 atau 1 dan -1. Domain bit yang dipilih adalah 1 dan -1. Alasan pemilihan ini akan dijelaskan kemudian.

Hal yang perlu ditekankan pada implementasi DSSS adalah frekuensi PN *Sequence*. Frekuensi PN *Sequence* atau biasa disebut *chip rate*, mempengaruhi *payload cover-object*. Data *audio* yang disimpan biasanya memiliki frekuensi *sampling* sebesar 44,1 kHz. Konsekuensinya,

frekuensi PN *Sequence* sebaiknya sama atau lebih dari 44,1 kHz. Hal ini diilustrasikan sebagai berikut, andaikan kita ingin mengirimkan informasi sepanjang 8 bit, artinya kita mengirimkan informasi 44100 *sampling* untuk tiap bitnya. Misalnya bit pertama bernilai 1, maka ada 44100 sample bernilai satu yang akan disisipkan. Penjelasan tadi adalah konsekuensi jika PN *Sequence* memiliki frekuensi 44,1 kHz. Jika PN *Sequence* memiliki frekuensi 44,1 kHz, sama saja artinya dengan kita menyisipkan 1 bit pesan dalam satu detik. *Payload* ini termasuk kurang karena umumnya data *audio* hanya sepanjang 4 menit (240 detik) yang berarti hanya sanggup disisipkan 30 *byte* data total. Pada makalah ini akan dijelaskan hasil analisis penggunaan berbagai frekuensi *chip rate*.

Salah satu cara untuk mengurangi frekuensi *chip rate* adalah memperkuat sinyal informasi yang akan disisipkan. Faktor penguat ini, kita sebut strength factor, akan memperkuat sinyal dengan ratio tertentu yang menyebabkan frekuensi *chip rate* yang digunakan tidak perlu terlalu tinggi. Hal ini bisa dijelaskan sebagai berikut, pada gelombang biasa, jika ketahanan terhadap *jamming* kita nyatakan sebagai rasio energi yang diterima dengan yang ditransmisikan maka makin besar rasionya (makin mendekati 1) maka transmisi tersebut dikatakan makin tahan *jamming*. Karena kita menginginkan energi yang dikirmkan sebisa mungkin masih dapat dipersepsi, energi transmisi harus dinaikan. Energi transmisi berbanding lurus dengan kuadrat frekuensi dan kuadrat amplitudo gelombang.

Jadi kita dapat memperbesar ketahanan *jamming* dengan cara memperbesar *chip rate* atau dengan memperbesar amplitudo. Jika *chip rate* besar, konsekuensinya *payload* pesan akan kecil. Jika amplitudo yang diperbesar, maka *noise* pada *cover object* akan makin besar dan terdengar. Namun, kita dapat menyeimbangkannya dengan memperkecil *chip rate* dan memperbesar amplitudo untuk ketahanan *jamming* yang sama. Prinsip ini digunakan untuk mempertahankan pesan dari proses kompresi data yang merusak data asal (kompresi *lossy*).

C. Discrete Fourier Transform

Pada skema DSSS yang dijelaskan sebelumnya, telah dijelaskan bahwa kita harus mentransformasi *cover-object* ke *frequency*-domain terlebih dahulu dengan tujuan mempertahankan pesan dari proses kompresi *lossy*. Transformasi yang akan digunakan adalah *Discrete Fourier Transform*.

Discrete Fourier Transform (DFT) adalah transformasi yang mengubah domain fungsi dari domain waktu ke domain frekuensi. Andaikan kita memiliki N buah sampel data dan x(n) adalah menunjukan nilai titik tersebut pada waktu ke-n, maka X(k) menunjukan nilai titik tersebut pada frekuensi ke-k. Rumus DFT adalah sebagai berikut:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

Nilai x dan X adalah nilai bilangan kompleks. Variabel i pada persamaan diatas adalah satuan imajiner. Nilai $e^{\frac{2\pi i}{N}}$ adalah *primitive* n-th *root of unity*. Transformasi balikan dari DFT, yaitu Inverse *Discrete Fourier Transform* adalah sebagai berikut:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1.$$

Transformasi balikan ini mengubah fungsi X(k) dalam domain frekuensi ke fungsi x(n) dalam domain waktu.

Perhitungan DFT dan IDFT, seperti yang dapat dilihat dari rumus di atas, memiliki kompleksitas $O(n^2)$. Oleh karena itu, *cost* perhitungan DFT dan IDFT cukup besar. Data *audio* yang akan ditransformasi bisa cukup banyak (10 juta sampel untuk durasi 4 menit satu *channel* 16bit WAV 44,1 kHz). Selain itu, bilangan yang dioperasikan adalah bilangan kompleks. Oleh karena itu, algoritma yang mangkus sangat dibutuhkan untuk menghitung DFT dan IDFT.

D. Fast Fourier Transform

Fast Fourier Transform (FFT) merujuk kepada kelas algoritma yang digunakan khusus untuk menghitung DFT dan IDFT dengan efisien. Algoritma FFT ini penting digunakan untuk implementasi DSSS ini. Jika kita menghitung DFT secara *brute-force*, *cost* yang dibutuhkan sangat besar dengan kompleksitas $O(n^2)$ dan tidak praktis untuk penggunaan ini. Sedangkan, kompleksitas FFT hanya O(n log n).

FFT biasanya diimplementasikan dengan cara memanfaatkan faktorisasi dari N (jumlah sampel yang ditransformasi) dan juga bahwa $e^{\frac{2\pi i}{N}}$ adalah *primitive* n-th *root of unity*. Algoritma yang terkenal salah satunya adalah algoritma FFT Cooley-Tukey. Algoritma ini menggunakan skema Divide and Conquer yang memecah N secara rekursif menjadi dua buah DFT.

Algoritma FFT Cooley-Tukey yang paling sederhana adalah *radix*-2 *Decimation In Time*. *Radix*-2 yang dimaksud adalah pembagian N menjadi dua buah DFT yang sama panjang (dibagi 2), sehingga dinamakan *radix*-2. Secara umum, N bisa difaktorisasi menjadi $N_1 x N_2$. Jika nilai N_1 yang menjadi *radix*, maka algoritmanya dinamakan *Decimation In Time* (DIT). Sedangkan jika N_2 yang dijadikan *radix*, maka algoritmanya dinamakan *Decimation In Frequency* (DIF). Selain itu, algoritma FFT ini juga dibedakan antara algoritma *In-place* dan *Out-of-place*. Algoritma *In-place* adalah algoritma yang perhitungannya menggunakan permutasi atau bit *reversal* dengan kontainer data yang sama, sedangkan algoritma *Out-of-place* menggunakan kontainer lain untuk

menyimpan hasil perhitungan. Selain menggunakan perhitungan seperti ini, kita juga dapat mengubah DFT menjadi transformasi matriks. Pengubahan ini terutama sangat berguna jika prosesor penghitungnya adalah prosesor grafik, yang memang dirancang untuk menghitung transformasi matriks bilangan *floating-point*.

Algoritma FFT ini sangat penting untuk diimplementasikan dengan baik. Karena jika proses transformasi memakan waktu yang lama, aplikasi steganografi pada data *audio* menggunakan DSSS menjadi tidak praktis dan *feasible*. Penulis mencoba beberapa implementasi algoritma FFT dan DFT naif.

III. IMPLEMENTASI DAN PENGUJIAN

A. Implementasi FFT

Implementasi algoritma FFT akan sangat menentukan kepraktisan penerapan DSSS untuk steganografi *audio*. Algoritma FFT harus dapat diimplementasikan seefisien mungkin. Semakin lama proses FFT dilakukan, steganografi *audio* menggunakan DSSS semakin tidak *feasible*.

Implementasi algoritma FFT mengharuskan kita mengimplementasikan tipe data bilangan kompleks. Penulis mengimplementasikan kelas bilangan kompleksnya sendiri pada bahasa C#. Hal ini dikarenakan, kelas tersebut tidak ada pada librari .NET dan harus diimplementasi sendiri. Kelas bilangan kompleks yang dibuat memiliki dua properti, bagian real dan bagian imajiner yang keduanya memiliki tipe data float. Untuk menghindari perhitungan yang tidak perlu, jumlah konversi dan operasi tipe data float diperkecil sesedikit mungkin untuk operasi perkalian dan pembagian bilangan kompleks.

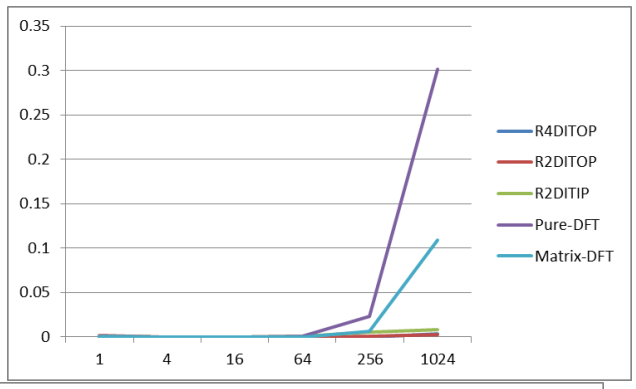
Awalnya, penulis mengimplementasikan DFT naif (*brute-force*). Namun, ternyata waktu proses sangat lama. Kemudian penulis mencoba mengimplementasi algoritma DFT menggunakan perkalian matriks. Hasilnya, waktu pemrosesan lebih cepat, namun masih belum cukup efisien. Kemudian penulis mengimplementasikan *Radix*-2 DIT *In-Place*, dan dihasilkan pertambahan kecepatan proses yang signifikan. Namun, setelah sedikit modifikasi, penulis mengimplementasikan *Radix*-2 DIT *Out-of-Place* yang ternyata memiliki performa jauh lebih bagus daripada daripada versi *In-Place*, dengan rasio hampir 28 kali lebih cepat untuk 16384 sampel. Tapi, jumlah sampel itu masih terlalu kecil dibandingkan jumlah sampel arsip suara rata-rata. Akhirnya, dengan mengorbankan memori untuk menambah kecepatan proses, penulis mengimplementasi sendiri *Radix*-4 DIT *Out-of-Place* hasil modifikasi dari *Radix*-2 DIT *Out-of-Place* dan memberikan performa yang terbaik dari algoritma yang sudah penulis implementasi sebelumnya dengan rasio hampir 2 kali lebih cepat dari *Radix*-2 DIT *Out-of-Place*. Penulis juga mempertimbangkan membuat versi *Radix*-16 DIT *Out-of-Place*, namun memori yang dibutuhkan untuk pemrosesan menjadi cukup besar, sementara jumlah operasi aritmatikanya terlalu banyak dan sulit disederhanakan. Sebagai ilustrasi, *Radix*-2 membutuhkan 4 buah suku bilangan kompleks yang akan dioperasikan untuk tiap n. *Radix*-4 membutuhkan 16 buah suku bilangan kompleks yang akan dioperasikan untuk tiap n (setelah disederhanakan akhirnya hanya 8 buah suku). Sedangkan *Radix*-16 membutuhkan 256 buah suku bilangan kompleks berbeda. Sehingga keuntungan yang didapat termasuk kecil. Akhirnya penulis memutuskan untuk tidak mengimplementasi *Radix*-16 DIT.

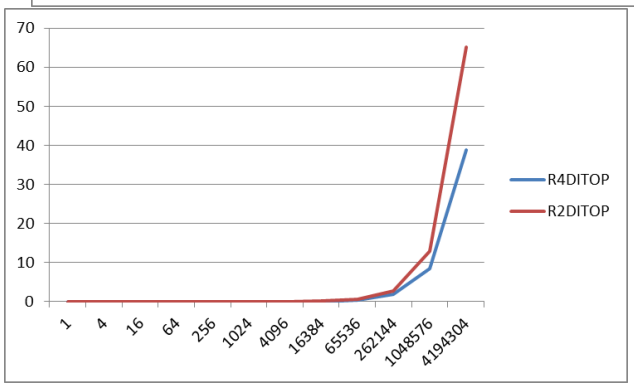
Penulis mencoba implementasi algoritma tersebut di lingkungan dengan prosesor Intel Core 2 Duo 2.0 GHz menggunakan bahasa C#. Berikut adalah tabel dan grafik percobaan:

N (Jumlah sampel) vs Algoritma	*Radix*-4 DIT *Out-of-Place*	*Radix*-2 DIT *Out-of-Place*	*Radix*-2 DIT *In-Place*	*Matrix*-DFT	*Pure*-DFT
1	0	0.002	0.001	0.001	0.001
4	0	0	0	0	0
16	0	0	0	0	0
64	0	0	0.001	0.001	0
256	0	0.001	0.005	0.023	0.006
1024	0.004	0.003	0.008	0.302	0.109
4096	0.015	0.024	0.054	4.648	1.909
16384	0.072	0.118	0.648	75.403	30.509
65536	0.362	0.541	14.986	-	-
262144	1.91	2.654	-	-	-
1048576	8.433	12.962	-	-	-
4194034	38.847	65.115	-	-	-

Tabel diatas menunjukkan hubungan antara jumlah sampel yang ditransformasi dan waktu (detik) yang dibutuhkan oleh algoritma yang digunakan. Dari tabel, dapat diketahui bahwa *Radix*-4 DIT *Out-of-Place* memiliki performa yang cukup bagus. Andaikata waktu rata-rata untuk mentransformasi 4 juta sampel (95 detik durasi suara) adalah 40 detik. Maka untuk arsip suara stereo, kira-kira dibutuhkan waktu 160 detik untuk menyisipkan pesan dan 80 detik untuk mengambil pesan. Lama waktu penyisipan tersebut dianggap masih praktis dan *feasible*.

Berikut adalah tabel performa implementasi algoritma tersebut:





Dari tabel dan grafik yang dihasilkan, kita dapat mengambil kesimpulan bahwa implementasi *Radix*-4 DIT *Out-of-Place* yang akan digunakan untuk steganografi ini.

B. Skema Steganografi Audio Menggunakan DSSS

Pada subbab ini, skema steganografi yang digunakan akan diterangkan. *Cover-object* yang digunakan adalah *file audio* mentah seperti *file* WAV. Andaikan kita memiliki informasi berupa *byte-sequence* yang akan disipkan pada *cover-object*. Informasi *byte-sequence* tersebut kita ubah menjadi informasi bit-*sequence*. Kemudian kita representasikan bit tersebut menjadi sinyal sedemikian sehinga jika nilai bit tersebut 1 maka amplitudo sinyalnya 1, sedangkan jika nilai bit tersebut 0 maka amplitudo sinyalnya -1. Seperti yang ditunjukkan sebagai berikut:

$A = \{a_i \mid a_i \in \{-1, 1\}\}$.

Selanjutnya, buka *cover-object file* WAV dan ambil data amplitudo sinyalnya. Amplitudo ini direpresentasikan sebagai nilai 16 bit signed integer dengan rentang 2^{15}-1 hingga -2^{15}+1. Jadi, bagi amplitudo ini dengan nilai 2^{15}-1 agar rentang nilai yang didapat antara 1 hingga -1.

Transformasikan data amplitudo ke domain frequensi menggunakan FFT.

Buat PN *Sequence* yang panjang dan acak dengan nilai amplitudo 1 atau -1. Jika PN *Sequence* memiliki *chip rate* cr, dan jika sinyal informasi memiliki jumlah sebanyak n sinyal, maka PN *Sequence* yang harus dibangkitkan sebanyak *cr* x *n*. Kita sebut PN Sequnce tersebut P, maka:

$P = \{p_i \mid p_i \in \{-1, 1\}\}$

Modulasikan tiap satu sinyal informasi dengan PN *Sequence* sebanyak cr kali dengan cara mengalikan amplitudo masing-masing. Maka akan dihasilkan sinyal B yang merupakan sinyal persebaran dari A dan tentunya panjangnya *cr* kali panjang semula. Awalnya, sebar dulu informasi A menjadi B seperti berikut:

$B = \{b_i \mid b_i = a_j, j \cdot cr \le i < (j+1) \cdot cr\}$

Selanjutnya modulasikan B dan P dan kalikan dengan faktor penguat α. Maka pesan inilah yang akan kita sisipkan pada *cover-object*. Misalkan w adalah pesan yang akan disisipkan, v adalah *cover-object* dan v' adalah *cover-object* berisi pesan. Maka proses tadi bisa dirumuskan sebagai berikut:

$w_i = \alpha \cdot b_i \cdot p_i$

$v'_i = v_i + w_i$

Skema pembuatan w akan menghasilkan *noise* yang sulit dideteksi maknanya. Jika faktor penguat dipilih terlalu besar, suara *noise* yang dihasilkan juga besar dan mungkin dapat merusak *cover-object*. Jadi pemilihan besar faktor penguat dan *chip-rate* sangat penting.

Skema ekstraksi akan dijelaskan selanjutnya. Karena efek dari PN *Sequence* yang dibangkitkan sebelumnya, sinyal yang ditambahkan pada data menjadi sangat acak. Agar informasi dapat diambil kembali, penerima pesan harus membangkitkan PN *Sequence* yang sama.

Kalikan sinyal PN *Sequence* yang bersesuaian untuk tiap sinyal *cover-object* hasil penyisipan, yang hubungannya ditunjukkan sebagai berikut:

$$\sum_{i=j.cr}^{(j+1)cr-1} p_i.v'_i = \sum_{i=j.cr}^{(j+1)cr-1} v_i.p_i + \sum_{i=j.cr}^{(j+1)cr-1} \alpha b_i.p_i^2$$

Jika kita perhatikan suku berikut:

$$\sum_{i=j.cr}^{(j+1)cr-1} v_i.p_i$$

Nilai dari suku tersebut akan mendekati 0 untuk jumlah sampel yang besar (*chip rate* yang besar). Hal ini dikarenakan PN *Sequence* yang acak menyebabkan penjumlahan sinyal menghasilkan bilangan yang mendekati nol atau nilai ambang tertentu.

Sedangkan suku kedua yaitu:

$$\sum_{i=j.cr}^{(j+1)cr-1} \alpha b_i.p_i^2$$

Memiliki sifat yang unik. Karena PN *Sequence* bernilai 1 atau -1, maka hasil dari kuadrat sinyal p_i adalah 1. Suku tadi bisa disederhanakan menjadi:

$$\sum_{i=j.cr}^{(j+1)cr-1} \alpha b_i$$

Jika dijumlahkan untuk satu periode *chip*, maka akan didapat sinyal αb_i yang merupakan informasi yang ditanam dan diperkuat oleh faktor penguat. Karena kita sudah menentukan b_i bernilai 1 atau -1 maka jika αb_i melebihi nilai nol, anggap informasi yang diambil adalah 1 dan jika nilai αb_i dibawah nilai nol, anggap informasi yang diambil adalah 0. Inilah alasan pemilihan rentang nilai B dan P.

Dari penjelasan sebelumnya, bisa kita simpulkan bahwa nilai αb_i harus melebihi nilai ambang yang dihasilkan *cover-object* agar didapatkan informasi yang jelas. Inilah yang menyebabkan pemilihan nilai faktor penguat harus tepat, tidak boleh terlalu kecil dan tidak boleh terlalu besar.

C. Hasil Pengujian

Penulis mengimplementasikan skema penyisipan dan ekstraksi di atas menggunakan *file* berformat WAV Stereo 16 bit PCM berdurasi 15 detik dengan jumlah sampel sekitar 1 juta sampel. Data yang disisipkan berupa teks dengan panjang karakter 8 karakter atau 8 *byte*. Penulis melakukan beberapa pengujian dan menyimpulkan faktor kekuatan maksimal adalah 10 dan *chip rate* adalah 10000 (hampir seperempat *sampling rate*). Faktor penguat dibawah 10 menghasilkan *static noise* yang terdengar seperti suara radio yang statik (tidak mendapat sinyal siaran), namun dibandingkan dengan suara lagu *cover-object noise* ini hanya terdengar seperti *noise* rekaman. Sedangkan faktor penguat sebesar 100 sudah menghilangkan suara *cover-object* dan suara yang terdengar hanya suara *noise* yang besar (seperti bunyi hujan).

Pemilihan yang *chip rate* yang cocok juga cukup sulit. Penyisipan dengan *chip rate* hampir seperempat *sampling rate* hanya menghasilkan *payload* yang kecil. Sama saja dengan menganggap satu detik digunakan untuk menyisipkan 4 bit yang berarti 2 detik digunakan untuk menyisipkan 1 *byte*. Durasi lagu total dari *channel* kiri dan kanan adalah 30 detik, sehingga kira-kira 15 *byte* data dapat disisipkan pada *cover-object* tersebut. *Payload* tersebut masih terbilang sangat kecil dibandingkan menggunakan metode LSB. Namun, jika *chip rate* diperkecil lagi, data yang disisipkan bisa rusak. *Chip rate* 1000 masih belum cukup untuk menyembunyikan data.

Pertimbangan yang bisa dibuat dari kesimpulan ini adalah pemilihan *chip rate* dan faktor penguat. Perkalian dari *chip rate* dan faktor penguat ini, paling tidak harus lebih besar dari 100000 untuk menghasilkan penyisipan pesan yang tahan kompresi. Untuk selanjutnya logaritma dari perkalian *chip rate* dan faktor penguat ini akan saya sebut nilai kualitas penyisipan. Jika nilai kualitas penyisipan ini makin besar, informasi yang disisipkan dijamin tersimpan dengan baik meskipun *file* WAV tersebut dikonversi menjadi format lain. Bahkan dengan kualitas penyisipan 5 (perkalian *chiprate* dan faktor penguat mengasilkan nilai 100000), *file* suara *cover-object* tahan terhadap proses pemotongan *audio*. Sebaliknya, semakin kecil kualitas penyisipan, semakin kecil pula kemungkinan pesan yang disisipkan terbaca jika diekstraksi.

Namun, kualitas penyisipan berbanding terbalik dengan kualitas suara *cover-object*. Makin besar kualitas penyisipan kualitas *cover-object* (kualitas *object* yang disisipkan) makin menurun.

Penulis mencoba berbagai metode untuk menguji ketahanan metode *Direct-Sequence* Steganografi yang sudah diimplementasikan. Metode yang digunakan adalah metode pemotongan data *audio* (*cropping*), inversi data *audio*, penimpaan informasi dan pengubahan format (terjadi kompresi *lossy*).

1. *Audio Cropping*

Sampel hasil steganografi (kita namakan *file* tersebut romanesca.wav) dibuka menggunakan software audacity. *Audio cropping* dilakukan menggunakan software tersebut. Romanesca.wav hanya berdurasi 15 detik, secara intuisi jika data tersebut kita potong maka informasi yang disisipkan dapat hilang secara signifikan (karena informasi disebar pada durasi yang relatif pendek).

Penulis mencoba memotong sinyal Romanesca.wav sebanyak 2 detik di dua tempat berbeda, satu tempat di dekat awal lagu dan satu tempat di dekat akhir lagu. Namun ternyata pesan dapat diekstraksi sempurna, padahal ukuran pesan hampir setengah dari *payload*.

Penulis juga mencoba faktor penguat yang lebih kecil dari 10 dan pesan masih dapat dipersepsi dengan kesalahan ekstraksi hampir 2 *byte*.

2. *Audio Inversion*

Penulis juga mencoba melakukan inversi pada Romanesca.wav. Hasilnya, informasi masih dapat diekstrak dengan cara membalikkan semua bit yang diambil. Hal ini dikarenakan informasi yang kita ambil juga ter-inversi.

3. *Audio Compression*

Tujuan dari pembuatan makalah ini terutama menghasilkan skema steganografi pada *cover-object* arsip *audio* yang tahan perubahan koversi format data. Perubahan konversi format data pada data *audio* biasanya disertai dengan penerapan kompresi tertentu agar ukuran data tidak terlalu besar. Namun, jika kompresi tersebut adalah kompresi *lossy* (ada sinyal-sinyal tidak penting yang dibuang) maka metode LSB jelas tidak cocok pada data

audio. Oleh karena itu, penulis digunakan metode DSSS.

Penulis mencoba mengkonversi Romanesca.wav menjadi *file* OGG dan FLAC. *File* format FLAC memiliki kompresi tipe *lossless*, jadi apabila *file* FLAC diubah lagi menjadi *file* WAV, data *audio*nya tidak berubah, sehingga proses ekstraksi masih menghasilkan informasi yang sama. Namun kompresi yang digunakan OGG adalah kompresi *lossy* (yang metode kompresinya mirip dengan yang digunakan MP3). Setelah diubah menjadi *file* OGG, *file* diubah lagi menjadi *file* WAV. Hasilnya, informasi yang disisipkan masih dapat diambil dengan sempurna dengan kualitas penyisipan 5. Hasil ini sangat penting dan membuktikan bahwa metode DSSS cocok digunakan untuk steganografi dengan *cover-object file audio*.

4. Penimpaan informasi

Penulis juga dengan sengaja menggunakan *cover-object* yang sudah disisipi pesan untuk digunakan menyembunyikan pesan dengan PN*Sequence* yang berbeda. Hasilnya, pesan masih dapat diambil dari dua PN *Sequence* tersebut. Artinya, data yang sudah disisipkan tidak dapat ditimpa untuk batas tertentu.

D. Analisis Kelayakan dan Pengembangan

Berdasarkan hasil pengujian yang dilakukan, metode DSSS terbukti mungkin dilakukan pada data *audio*. Metode ini memiliki keunggulan sebagai berikut:

1. Mampu menyisipkan pesan dengan ketahanan sangat bagus. Pesan masih dapat diekstraksi meskipun *cover-object* mengalami *audio-cropping* ataupun perubahan format.
2. Karena informasi yang disisipkan berbentuk *noise* biasa, informasi tersebut tidak dapat dipersepsi. Sehingga mampu mengecoh orang ketiga yang berusaha menangkap pesan tersebut, tapi tidak memiliki PN *Sequence* yang sama. Orang biasa pun tidak akan curiga apabila mendengar *noise* tersebut.
3. Kemampuan untuk mengekstraksi data sangat ditentukan oleh PN *Sequence*. Perbedaan pembangkitan PN *Sequence* yang sedikit saja, dapat mengacaukan seluruh isi pesan hasil ekstraksi. Tanpa PN *Sequence* yang sama, pesan sangat sulit diekstraksi.

Selain memiliki keunggulan diatas, DSSS yang diimplementasi ini masih memiliki banyak kekurangan dan keterbatasan, diantaranya:

1. *Cost* implementasi cukup besar dan rumit. Penggunaan transformasi DFT masih terbilang belum efisien dan memakan waktu terlalu lama. Padahal jumlah sampel rata-rata yang harus ditransformasi sebanyak hampir 20 juta (4 menit data *audio* 16 bit PCM Stereo).
2. Memori yang dibutuhkan untuk pemrosesan rekursif yang efisien dari algoritma *Radix*-4 FFT cukup besar. Untuk *file audio* dengan 20 juta sampel, memori yang dibutuhkan di atas 1 GB. Penulis juga mengalami keterbatasan memori RAM (*out of* memory exception) saat mencoba mentransformasi *file audio* berdurasi 4 menit.
3. Nilai kualitas penyisipan yang aman adalah sekitar nilai 5 (penulis melakukan pengujian dengan kualitas penyisipan 5). Nilai kualitas ini sangat dipengaruhi sinyal *audio cover-object* itu sendiri. Penulis melakukan pengujian menggunakan Romanesca.wav yang berisi lagu yang relatif tenang. Jika *cover-object* adalah musik rock atau jazz, kualitas penyisipan yang dibutuhkan mungkin berbeda.
4. Konsekuensi dari kualitas penyisipan aman yang besar adalah kecilnya *payload*. Romanesca.wav dengan kualitas penyisipan 5 dan faktor penguat 10 hanya mampu menyimpan 15 *byte* data.

Penulis masih berusaha mengembangkan metode ini agar proses penyisipan semakin praktis digunakan (tidak terlalu lama). Meskipun begitu, penulis mengambil kesimpulan, bahwa cara ini tetap baik untuk digunakan karena, proses yang lama adalah proses penyisipan, sedangkan proses ekstraksi hanya memakan waktu setengah dari proses penyisipan.

IV. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian, penulis mengambil kesimpulan bahwa penerapan *Direct-Sequence Spread Spectrum* Steganography pada *cover-object* arsip *audio*, mungkin dan praktis digunakan. Setidaknya untuk durasi 15 detik data pertama. Metode ini terbukti sangat tahan manipulasi dan sangat aman dengan *noise* yang dihasilkan cukup kecil. Namun *cost* yang dibutuhkan jauh lebih mahal daripada metode LSB.

Penulis masih merencanakan pengembangan metode ini. Beberapa saran yang bisa digunakan untuk memperbaiki metode ini adalah sebagai berikut:

1. Menggunakan algoritma *Split Radix* FFT yang diklaim lebih cepat daripada *Radix*-4 FFT, ataupun algoritma FFT yang lebih bagus lagi.
2. Mempartisi sampel yang ditransformasi. Sample *audio* dipartisi menjadi potongan *audio* 15 detik (atau lebih kecil) dan penyisipan dilakukan dengan memecah informasi ke tiap partisi. Cara ini akan memakan waktu lebih cepat.
3. Menggunakan skema penyisipan berulang. Jika informasi melebihi *payload*, *cover-object* dapat digunakan untuk disisipi ulang dengan menggunakan PN *Sequence* yang berbeda. Skema ini mengizinkan penghitungan *payload* dan *noise* yang berbeda.

V. REFERENSI

Adriansyah, Y. (2010). *Simple Audio Cryptography.* Bandung: Department of Informatics Engineering, Schools of Electronics and Informatics Engineering, Bandung Institute of Technology.

Alfatwa, D. F. (2007). *DIGITAL AUDIO WATERMARKING MENGGUNAKAN ANALISIS AUDIO CONTENT.* Bandung: Department of Informatics Engineering, Schools of Electronics and Informatics Engineering, Bandung Institute of Technology.

Anonymous. (2011, 3 1). *Direct-sequence Spread Spectrum.* Retrieved 3 1, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Direct-sequence_Spread_Spectrum

Anonymous. (2011, 3 1). *Spread Spectrum.* Retrieved 3 1, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Spread_Spectrum

Bistok D.L, I. S. (2005). *Perancangan Algoritma Audio Watermarking dan Pengukuran Performansinya.* Bandung: Department of Informatics Engineering, Schools of Electronics and Informatics Engineering, Bandung Institute of Technology.

Cvejic, N. (2004). *ALGORITHMS FOR AUDIO WATERMARKING AND STEGANOGRAPHY* (Vols. ISBN 951-42-7384-2). Oulu: Department of Electrical and Information Engineering. Information Processing Laboratory. University of Oulu.

Herianto. (2008). *Novel Digital Audio Watermarking.* Bandung: Department of Informatics Engineering, Schools of Electronics and Informatics Engineering, Bandung Institute of Technology.

Rinaldi Munir, B. R. (2007). *Modifikasi Spread Spectrum Watermarking dari Cox Berbasiskan pada Enkripsi Chaotic.* Bandung: Sekolah Teknik Elekto dan Informatika, ITB.

Rinaldi Munir, B. R. (2007). Secure Spread Spectrum Watermarking Algorithm Based on Chaotic Map for Still Images. *Proceedings of the International Conference on Electrical Engineering and Informatics Institut Teknologi Bandung.* -. Bandung, Indonesia: -.

Wei Qin Cheng, F. H. (2007). *Robust Audio Steganography using Direct-Sequence Spread Spectrum Technology.* -: University of British Columbia.

Zamani, M. a. (2009). *A Genetic-Algorithm-Based Approach for Audio Steganography.* Kuala Lumpur, Malaysia: World Academy of Science, Engineering and Technology.

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, 29 April 2010

Rizky Maulana Nugraha - 13508083