Pembuatan Java API untuk *Dynamic Graph Software Watermarking* dengan Enumerasi *Parent Pointer Graph*

Shofi Nur Fathiya[#1], Rinaldi Munir[#2]

[#]*Teknik Informatika, Institut Teknologi Bandung*
Jalan Ganeca No. 10 Bandung
[1]if18084@students.if.itb.ac.id
[2]rinaldi-m@stei.itb.ac.id

Abstract— ***Software watermarking* merupakan teknik menyisipkan informasi kepemilikan atau *watermark* ke dalam program. Teknik ini menerima program dan *watermark* sebagai masukan dan menghasilkan program ber-*watermark*. Tujuan dari *software watermarking* ini adalah untuk membuktikan kepemilikan dari sebuah program.**

Dynamic Graph Watermarking* (DGW) merupakan metode yang dianggap paling kuat dalam *software watermarking* karena daya tahannya yang tinggi terhadap serangan. Dalam menyisipkan *watermark*, metode ini menggunakan struktur graf yang dibuat berdasarkan enumerasi graf. Enumerasi graf yang sering digunakan dalam metode DGW adalah *Planted Plane Cubic Tree* (PPCT), Radix-k, dan *Permutation Graph.

Dalam makalah ini diajukan sebuah aplikasi DGW menggunakan enumerasi *Parent Pointer Graph* (PPG). PPG dipilih karena graf yang dihasilkan lebih sederhana dibandingkan dengan enumerasi lainnya dimana setiap simpul pada graf ini hanya memiliki satu *pointer*, sedangkan pada enumerasi lainnya setiap simpul memiliki dua *pointer*.

Program yang akan diberi *watermark* pada makalah ini adalah program berbahasa Java. Solusi yang akan dibuat terdiri dari dua bagian, yaitu sebuah Java API untuk menyisipkan *watermark* dan sebuah aplikasi Java untuk mengekstrak *watermark*. Dari hasil pengujian, dapat disimpulkan bahwa PPG dapat digunakan sebagai enumerasi pada metode DGW namun memiliki tingkat ketahanan yang rendah.

Keywords— ***Dynamic Graph Watermarking, Parent Pointer Graph, software watermarking*.**

I. PENDAHULUAN

Software sebagai benda digital memudahkan siapa pun dalam mengubah isinya serta menyebarkannya kembali. Tindakan ini melanggar hak cipta ketika dilakukan tanpa ada izin dari pencipta *software* tersebut. Tentunya hal ini akan sangat merugikan pencipta *software* terutama ketika ia tidak dapat membuktikan bahwa *software* itu adalah miliknya.

Agar informasi kepemilikan dapat tersimpan dengan baik di dalam *software*, dapat digunakan *software watermarking*. *Software watermarking* merupakan teknik menyisipkan pesan berisi informasi kepemilikan ke dalam sebuah program.

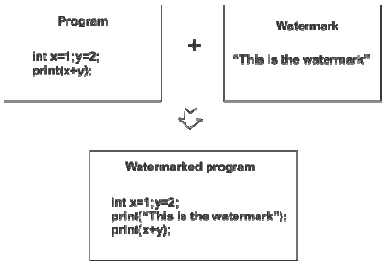
Metode yang paling kuat pada *software watermarking* adalah *Dynamic Graph Watermarking* (DGW). Metode ini memasukkan *watermark* dalam sebuah representasi graf saat proses penyisipan [6]. Aplikasi DGW yang telah dibuat hingga saat ini telah mengaplikasikan beberapa jenis enumerasi graf, yaitu *Planted Plane Cubic Tree*, Radix-k, dan *Permutation Graph.* Selain keempat teknik enumerasi ini, terdapat teknik lainnya yang belum diimplementasikan yaitu *Parent Pointer Graph* atau PPG. PPG merupakan sebuah teknik enumerasi dimana setiap simpul memiliki *pointer* yang mengarah ke orangtuanya.

Akan merepotkan bila setiap kali dilakukan perubahan pada program, program harus dimasukkan kembali ke *software* pemberi *watermark* untuk disisipkan *watermark* di dalamnya. Untuk itu pada makalah ini akan dibuat sebuah *software watermarking* yang diimplementasikan dalam bentuk API. API ini mengaplikasikan metode DGW. Berbeda dari sistem yang telah ada, teknik enumerasi graf yang digunakan pada makalah ini adalah *Parent Pointer Graph* (PPG).

II. STUDI LITERATUR

A. Konsep Dasar Software Watermarking

Software watermarking merupakan sebuah penanda yang disisipkan ke dalam *software* untuk mengkodekan beberapa informasi pada *software* tersebut [1]. Contoh sederhana dari *software watermarking* dapat dilihat pada Gambar 1.



Gambar 1 Contoh *software watermarking*

Software watermarking terdiri dari dua proses utama, yaitu proses penyisipan dan proses ekstraksi. Proses penyisipan merupakan proses memasukkan *watermark* ke dalam program sedangkan proses ekstraksi merupakan proses mengeluarkan kembali *watermark* dari dalam program.

Referensi [10] menyatakan bahwa keefektifan dari *software watermark* ditunjukkan oleh:

1. *Robustness*, menyatakan seberapa kuat *watermark* dapat bertahan untuk tetap berada di dalam program.
2. *Efficiency*, menyatakan apakah teknik yang digunakan efisien atau tidak.
3. *Perceptibility*, menunjukkan tingkat mudah terlihat atau tidaknya *watermark*.
4. *Fidelity*, menghitung kesalahan yang muncul pada program akibat dari penambahan *watermark.*

B. *Dynamic Graph Watermarking*

Dynamic Graph Watermarking atau DGW merupakan metode yang dibuat oleh Collberg dan Thomborson dengan memanfaatkan topologi graf untuk menyisipkan *watermark* di dalam program [3]. Metode ini termasuk ke dalam metode dinamik sehingga graf dibuat dan disisipkan saat *runtime* program. Metode ini dinilai lebih kuat daripada metode statik dan dinamik lainnya. Ada 4 langkah yang dilakukan dalam proses penyisipan DGW:

- Memilih angka *watermark*, yaitu *n.*
- Membuat representasi *graf G* dari *n.*
- Membuat kode *watermark* yang dapat membangun struktur *graf G* saat *runtime*.
- Menyisipkan kode *watermark* ke dalam program yang ingin diberi *watermark* sehingga pada saat program dijalankan, kode *watermark* pun ikut dieksekusi.

Keuntungan terbesar dari metode DGW adalah pada metode ini terdapat *pointer* sehingga *watermark* akan sulit untuk dianalisis dan dikeluarkan dari program. Dan karena DGW dibangun secara dinamis, informasi saat *runtime* harus dikumpulkan untuk dapat menganalisis struktur *watermark.*

Pada metode DGW terdapat beberapa serangan yang dapat merusak atau bahkan menghapus *watermark* dari dalam program. Serangan-serangan tersebut adalah:

- *Additive attack,* yaitu menyisipkan *watermark* baru ke dalam program yang sudah memiliki *watermark*.
- *Distortive attack*, yaitu mengubah sebagian dari *watermark* sehingga *watermark* tidak dikenali kembali. *Distortive attack* dalam metode DGW ditunjukkan pada Gambar 2.
- *Subtractive attack*, yaitu menghilangkan seluruh bagian *watermark* di program.

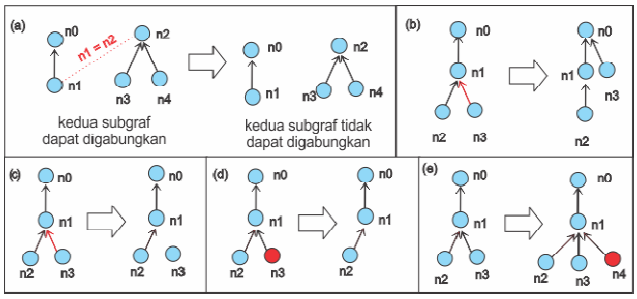
C. *Parent Pointer Graph*

Parent Pointer Graph (PPG) merupakan teknik enumerasi graf dimana setiap simpul hanya memiliki satu *pointer*, yaitu ke arah orang tuanya saja. Teknik ini cukup sederhana karena jumlah *pointer* yang tidak terlalu banyak. Gambar 3 menunjukkan beberapa graf PPG yang dapat dibangun dengan empat buah simpul.

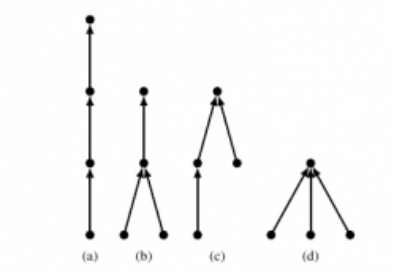
D. *Watermark* pada Java

1) *Bytecode pada Java*

Pada Java, program yang telah selesai dibuat direpresentasikan dengan *bytecode*. *Bytecode* dapat dijalankan di atas Java *Virtual Machine* (JVM), sebuah platform untuk



Gambar 2 *Distortive attack* pada metode DGW dengan cara (a) menghapus penghubung antar subgraf, (b) memindahkan *pointer*, (c) menghapus *pointer*, (d) menghapus simpul, dan (e) menambahkan *pointer* baru



Gambar 3 Graf yang dapat dibangun dengan empat simpul

Java yang berada di antara bahasa tingkat tinggi dan komputer sebenarnya [4]. *Bytecode* mengandung banyak informasi mengenai kode aslinya. Ini membuatnya mudah didekompilasi menjadi kode kembali untuk mendapatkan algoritma penting di dalamnya [9] sehingga membuat proses penyisipan dan ekstraksi menjadi lebih mudah. Hal ini mendasari penggunaan bahasa Java dalam makalah ini.

2) *Java API*

API atau *Application Programming Interface* merupakan seperangkat fungsi standar yang telah disiapkan oleh suatu sistem operasi dan bahasa tertentu. Pada Java, fungsi pada API disimpan pada *package* yang sesuai dengan fungsi tersebut. Di dalam masing-masing *package* tersebut terdapat *classes* dan *interfaces* serta *method, field*, dan *constructor* di dalamnya [7].

File yang akan dijadikan API dapat berupa file .jar. Untuk mendapatkan file .jar ini, program cukup di-*build* sehingga muncul file .jar di folder dist pada *project.*

E. *Penelitian Terkait*

Saat ini terdapat beberapa penelitian yang juga terkait dengan *software watermarking*. Salah satunya adalah tesis yang dibuat oleh Yong He dari *University of Auckland* dengan judul "*Tamperproofing a Software Watermark by Encoding Constants*" [6]. Isi dari tesis ini adalah penambahan metode *encoding constant* pada DGW dengan enumerasi PPCT dan menghasilkan sebuah program pemberi *watermark* bernama JSafeMark. Penelitian lainnya adalah sistem Sandmark yang dibuat oleh Collberg dan Townsend dari *University of Arizona* [2]. Beragam teknik *watermarking* diaplikasikan pada sistem

ini, namun pada sistem ini belum diterapkan metode DGW dengan enumerasi PPG.

Penelitian terkait selanjutnya yaitu sistem DashO yang merupakan sebuah aplikasi komersial dari PreEmptive *Solution* [8]. DashO menggunakan teknik *watermarking* statik dengan cara mengubah nama dari setiap file kelas dan menambahkan beberapa kode di dalam kelas-kelas tersebut. Pada penelitian terakhir, sistem Allatori, diterapkan *watermarking* teknik statik dengan menggunakan serangkaian operasi *push* dan *pop*. Rangkaian ini terdiri dari 4 instruksi *push integer* dan diikuti oleh 4 instruksi *pop* [5].

III. SOLUSI YANG DIAJUKAN

Pada bagian ini dirancang dua komponen utama dari *software watermarking*, yaitu penyisip dan pengekstrak. Penyisip dibuat dalam bentuk Java API dan untuk selanjutnya disebut sebagai API Penyisip *Watermark* sedangkan pengekstrak dibangun sebagai aplikasi biasa dan untuk selanjutnya disebut sebagai Aplikasi Pengekstrak *Watermark.*

Ada beberapa notasi yang digunakan untuk mempermudah pembahasan, yaitu:

- Program *P* : program yang akan diberi *watermark.*
- *Watermark W* : *watermark* yang ingin disisipkan.
- Program *P'* : program yang memiliki *watermark.*
- Penanda *mark* : *method* pada API Penyisip *Watermark* yang berfungsi menandai lokasi penyisipan *watermark* di dalam program *P.*
- *Method M* : *method* pada API Penyisip *Watermark* yang berfungsi menerima masukan *watermark W* dan memanggil perintah untuk memulai penyisipan.
- Angka *k* : jumlah subgraf yang dibentuk dari *G.*
- Graf *G* : graf yang dibangun dari *W.*

F. API Penyisip Watermark

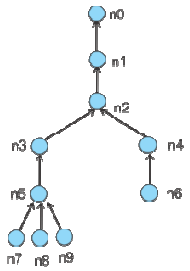
Analisis dan perancangan solusi untuk API Penyisip *Watermark* terdiri dari dua hal, yaitu membuat representasi graf dari masukan *string watermark* dan menyisipkan graf tersebut ke dalam program masukan. API ini memiliki *method* penanda *mark* dan *method M.*

1) Pembuatan Graf dari String Watermark

Langkah-langkah yang dilakukan dalam membuat graf ini adalah sebagai berikut.

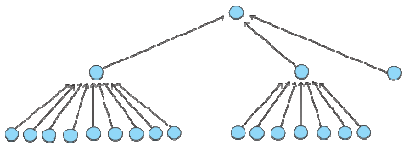
1. Seluruh *watermark* dianggap sebagai teks.
2. Diambil 3 digit nilai ASCII dari setiap karakter pada W. Nilai-nilai ASCII ini kemudian digabungkan menjadi satu barisan angka. Misalkan untuk *W* bernilai "pq", karena nilai ASCII dari '*P'* adalah 112 dan nilai ASCII dari 'q' adalah 113, maka barisan angka yang terbentuk adalah 112113.
3. Dibuat sebuah graf dimana setiap digit angka pada barisan ini menunjukkan jumlah anak dari setiap simpul. Untuk barisan angka 112113, graf yang dihasilkan ditunjukkan pada Gambar 4.

   Pada langkah 3 ini terdapat hal yang perlu diperhatikan, yaitu ketika barisan angka tidak dapat membentuk graf.



Gambar 4 Graf untuk *watermark* bernilai "pq"

Contohnya adalah string bernilai "a". Nilai ASCII dari 'a' adalah 097 sehingga anak simpul pertama adalah 0. Jika simpul pertama tidak memiliki anak, maka simpul-simpul lainnya pun tidak dapat terbentuk. Untuk itu, pada kasus seperti ini nilai dari 'a' ditambahkan 300 sehingga nilai yang semula merupakan 097 berubah menjadi 397. Graf yang merepresentasikan nilai 'a' adalah seperti pada Gambar 5.



Gambar 5 Graf untuk *watermark* "a"

2) Penyisipan Watermark pada Program

Cara yang dilakukan untuk menyisipkan *watermark* ke dalam program adalah sebagai berikut.

1. API dimasukkan sebagai *library* program *P* lalu di-*import* di dalam program *P.*
2. Di dalam program *P* dituliskan beberapa *mark* di tempat yang diinginkan dan *method M* yang menerima masukan *watermark W* seperti pada Gambar 6.

```
public static void main(String[] args) {
    // TODO code application logic here
    String sapa = "Hello World";
    int a = 0;
    int b = 10;
    Tanda.beriPenanda();

    for(int i=a; i<b; i++) {
        System.out.println(i);
    }

    Tanda.beriPenanda();
    System.out.println(sapa);
    Tanda.beriPenanda();

    Sisip.sisipWatermark("pq");

}
```

penanda *mark*

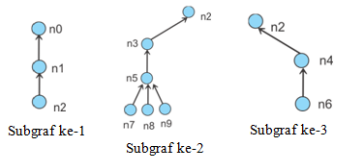
method M

Gambar 6 Program *P* dengan penanda *mark* dan *method M*

3. Program P kemudian di-build dan dijalankan. API akan menghitung jumlah *mark* yang telah diletakkan pada program *P*. Tidak seluruh *mark* dihitung, hanya *mark* yang berada pada jalur eksekusi program saja yang

diambil. Penanda *mark* yang tidak dieksekusi tidak akan dihitung dan tidak akan digantikan dengan *method* pembangun *watermark*.

4. Membuat graf *G* dari *watermark W* seperti yang telah dijelaskan pada subbab 3.1.1. Dari *watermark* bernilai "pq", graf *G* yang dihasilkan tampak seperti Gambar 4.
5. Setelah graf *G* dibuat, selanjutnya adalah mencari nilai *k* untuk nantinya membagi graf *G* menjadi sejumlah *k*. Nilai *k* diambil dari jumlah penanda *mark* yang ditemukan. Namun jika graf *G* tidak dapat dibagi sejumlah *mark* misalnya karena jumlah simpul kurang dari jumlah *mark*, maka nilai *k* diisi sejumlah graf *G* dapat dibagi.
6. Graf *G* dibagi menjadi *k* buah subgraf. Hasil pembagian subgraf *G* dapat dilihat pada Gambar 7.



Gambar 7 Hasil pembagian graf *G*

7. API lalu membuat sebuah kelas *watermark* yang merupakan representasi dari sebuah simpul, dalam hal ini diberi nama kelas Hasil. Untuk mempermudah proses pembuatan kode graf, dibuat pula kelas bernama HashSimpul yang berisi *hashtable* untuk menyimpan simpul tertentu. Untuk masing-masing subgraf, dibuat *method* pada kelas Hasil yang akan membangun subgraf tersebut. Kelas HashSimpul dan kelas Hasil dapat dilihat pada Gambar 8 dan Gambar 10.

```
public class HashSimpul {
    public static Hashtable tabel = new Hashtable();
}
```

Gambar 8 Kelas HashSimpul

8. Penanda di dalam program *P* diganti dengan *method* dari kelas Hasil dan *method M* dihapuskan. Didapatkan program *P'* seperti pada Gambar 9. Penyisipan *watermark* pun selesai dilakukan.

```
public static void main(String[] args) {
    // TODO code application logic here
    String sapa = "Hello World";
    int a = 0;
    int b = 10;
    Hasil.sub0();       // method penyusun graf watermark ke-1

    for(int i=a; i<b; i++) {
        System.out.println(i);
    }

    Hasil.sub1();       // method penyusun graf watermark ke-2
    System.out.println(sapa);
    Hasil.sub2();       // method penyusun graf watermark ke-3
}
```

Gambar 9 Program *P'*

```
public class Hasil {
    public Hasil parent;

    // kode untuk subgraf 1
    public static void sub0() {
        Hasil n0 = new Hasil();
        Hasil n1 = new Hasil();
        n1.parent = n0;
        Hasil n2 = new Hasil();
        n2.parent = n1;
        HashSimpul.tabel.put(2, n2);
    }

    // kode untuk subgraf 2
    public static void sub1() {
        Hasil n2 = (Hasil)HashSimpul.tabel.get(2);
        Hasil n3 = new Hasil();
        n3.parent = n2;
        Hasil n5 = new Hasil();
        n5.parent = n3;
        Hasil n7 = new Hasil();
        n7.parent = n5;
        Hasil n8 = new Hasil();
        n8.parent = n5;
        Hasil n9 = new Hasil();
        n9.parent = n5;
    }

    // kode untuk subgraf 3
    public static void sub2() {
        Hasil n2 = (Hasil)HashSimpul.tabel.get(2);
        Hasil n4 = new Hasil();
        n4.parent = n2;
        Hasil n6 = new Hasil();
        n6.parent = n4;
    }
}
```

Gambar 10 Kelas Hasil

G. *Aplikasi Pengekstrak Watermark*

Aplikasi ini menerima masukan program *P'* yang merupakan sebuah program .jar dan nama kelas utama dari program *P'* tersebut. Aplikasi ini kemudian akan memunculkan *watermark* yang tersimpan di dalam *P'* serta ilustrasi graf yang tersimpan pada *P'*.

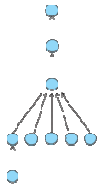
Cara kerja Aplikasi Pengekstrak *Watermark* ini adalah dengan terlebih daulu mengubah program *P'* yang merupakan file .jar menjadi file .class. Selanjutnya, baris-baris yang merupakan tempat kode pembangun graf yang berada di dalam program *P'* pun dipilih. Seluruh bagian dari kode yang telah ditemukan ini lalu disatukan untuk mendapatkan graf *G*. Dari graf *G* ini kemudian didapatkan *watermark W* yang tersimpan.

Ada beberapa langkah yang perlu dilakukan untuk mengubah graf *G* menjadi *watermark W*. Misalkan ditemukan graf *G* seperti pada Gambar 4. Untuk mengubah graf *G* menjadi *watermark W*, dapat dilakukan langkah-langkah berikut.

1. Hitung jumlah anak pada masing-masing simpul mulai dari n0 hingga n9, yaitu 1, 1, 2, 1, 1, 3, 0, 0, 0, dan 0.
2. Deretan angka 0 terakhir dihapuskan dari barisan karena kemungkinan besar bukan merupakan barisan angka *watermark* sehingga angka yang tersisa adalah 1, 1, 2, 1, 1, dan 3 serta barisan yang terbentuk adalah 112113.
3. Dilakukan konversi setiap 3 digit angka ke karakter dimana 3 digit angka tersebut merupakan nilai ASCII dari karakter yang dicari. Tiga digit pertama yaitu 112

diubah menjadi ‘p’ dan 3 digit selanjutnya diubah menjadi ‘q’ sehingga didapat *watermark* bernilai “pq”.

4. Ada beberapa hal yang perlu diperhatikan, yaitu:
   - Jika banyaknya angka pada barisan angka bukan merupakan kelipatan 3, maka ditambahkan angka 0 di belakangnya sehingga banyaknya angka pada barisan menjadi kelipatan 3. Sebagai contoh, terdapat graf *G*1 seperti pada Gambar 11.



Gambar 11 Graf *G1*

Jumlah anak pada masing-masing simpul adalah 1, 1, 5, 1, 0, 0, 0, 0, dan 0. Jika seluruh deretan angka 0 di bagian akhir dihilangkan, maka barisan angka akan menjadi 1151. Perlu ditambahkan angka 0 sehingga barisan angka berubah menjadi 115100. Dari barisan angka ini, didapatkan *watermark* “sd”.

   - Jika 3 digit angka yang akan dikonversi bernilai lebih dari 300, maka nilai tersebut dikurangi 300.

IV. PENGUJIAN SOLUSI

A. *Hasil Pengujian*

Untuk pengujian pada API Penyisip *Watermark* diujikan dua hal, yaitu memilih baris penanda di dalam program *watermark* dan menyisipkan *watermark* ke dalam program. Ada 3 *project* program yang akan diberikan *watermark* menggunakan API ini dan masing-masing dari *project* ini diberikan API Penyisip *Watermark*, diberikan penanda di dalamnya, serta disisipkan *watermark* 3 kali dengan 3 *watermark* yang berbeda. *Watermark* yang akan disisipkan adalah “IF” , “Teknik Informatika”, dan “Program ini dimiliki oleh PT. DGW PPG.”. Program yang telah diberi *watermark* harus dapat berjalan seperti sebelum diberi *watermark*. Hasil pengujian ini dapat dilihat pada Tabel 1.

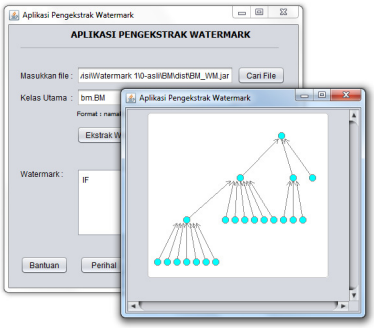
TABEL 1
HASIL PENGUJIAN API PENYISIP *WATERMARK*

ID	Deskripsi	Hasil
U-A-01	Memberikan baris penanda	Berhasil
U-A-02	Memasukkan *watermark* dan menyisipkannya ke dalam program	Berhasil

Untuk pengujian pada Aplikasi Pengekstrak *Watermark*, hal yang diujikan adalah mengekstrak *watermark* dari dalam program dan menampilkan graf yang terbentuk dari *watermark* tersebut. Masing-masing program yang telah diberi *watermark* sebelumnya diujikan pada bagian ini. Hasil dari pengujian ini dapat dilihat pada Tabel 2. Untuk hasil pengujian watermark “IF” dapat dilihat pada Gambar 12.

TABEL 2
HASIL PENGUJIAN APLIKASI PENGEKSTRAK *WATERMARK*

ID	Deskripsi	Hasil
U-B-01	Mengekstrak *watermark* dari program	Berhasil
U-B-02	Menampilkan graf yang terbentuk	Berhasil



Gambar 12 Hasil ekstrak watermark untuk watermark “IF”

Program yang telah diberi *watermark* lalu diberikan berbagai macam serangan seperti pada subbab 2.2 sebagai pengujian ketahanan (*robustness*). *Subtractive attack* tidak diujicobakan karena serangan ini mengambil seluruh isi *watermark* sehingga *watermark* sudah pasti tidak dapat diambil kembali. Hasil pengujian ketahanan dapat dilihat pada Tabel 3.

TABEL 3
HASIL PENGUJIAN KETAHANAN

ID	Serangan	Hasil		
		W1	W2	W3
U-T-01	Menambahkan *watermark* baru	O	O	O
U-T-02	Menghapus penghubung antar subgraf	O	O	O
U-T-03	Mengubah arah *pointer* simpul	X	X	X
U-T-04	Menghapus *pointer* simpul	O	X	O
U-T-05	Menghapus simpul	X	X	X
U-T-06	Menambahkan simpul baru	X	X	X

Keterangan : O = *watermark* masih dapat diambil secara utuh
X = *watermark* tidak dapat diambil secara utuh

Selain ketahanan, dilakukan pula pengujian pada aspek keefektifan lain, yaitu *efficiency, perceptibility*, dan *fidelity*. Hasil pengujian ini dapat dilihat pada Tabel 4.

TABEL 4
HASIL PENGUJIAN KEEFEKTIFAN

ID	Elemen	Hasil		
		W1	W2	W3
U-E-01	Jumlah karakter	2	18	38
U-E-02	Jumlah simpul	21	135	273
U-E-03	Waktu penyisipan (ms)	381	428	446
U-E-04	Waktu ekstraksi (ms)	59	134	178
U-E-05	Ukuran kelas Hasil (KB)	1	2.72	4.99
U-E-06	Perbedaan dengan program asli	-	-	-
U-E-07	Berjalan seperti seharusnya	Ya	Ya	Ya

B. Analisis Hasil Pengujian

Seluruh hasil dari pengujian pada API Penyisip *Watermark,* yaitu U-A-01 dan U-A-02, serta pengujian pada Aplikasi Pengekstrak *Watermark,* yaitu U-B-01 dan U-B-02, adalah berhasil. Ini menunjukkan perangkat lunak yang dibuat sudah memenuhi kebutuhan yang diperlukan dalam *software watermarking*.

Dalam pengujian ketahanan, tidak seluruh *watermark* hasil ekstraksi bernilai benar setelah diberi serangan. Pada pengujian U-T-01, *watermark* hasil ekstraksi bernilai benar karena telah diberikan penanganan pada API Penyisip *Watermark*, dimana bila sudah terdapat *watermark* yang dibangun dengan menggunakan API ini, maka tidak akan dibangun kembali *watermark* di dalamnya. Untuk pengujian U-T-02, ketiga *watermark* masih dapat diekstraksi dengan benar walaupun terdapat penghubung antar subgraf yang dihapuskan. Hal ini dikarenakan kerusakan yang terjadi akibat penghapusan penghubung ini tidak terlalu besar sehingga saat ekstraksi dilakukan, subgraf yang tidak terhubung masih dapat disambungkan kembali.

Di pengujian U-T-03, perubahan arah pada *pointer* membuat perubahan jumlah anak dari simpul yang ditunjuk oleh *pointer* tersebut. Perubahan jumlah ini mempengaruhi barisan angka yang didapat pada proses ekstraksi sehingga nilai *string* yang dihasilkan pun berubah. Sementara pada pengujian U-T-04, terdapat *watermark* yang masih dapat diekstrak dengan benar dan terdapat pula yang tidak. ini terjadi karena penghapusan *pointer* membuat graf menjadi tidak sempurna seperti yang dapat dilihat pada Gambar 3(c). Terdapat sebuah simpul yang terpisah dari graf. Saat ekstraksi dilakukan, dibuat *pointer* baru untuk simpul ini agar simpul dapat terhubung dengan graf kemudian *pointer* tersebut dicoba untuk dihubungkan ke simpul lain yang mungkin merupakan orang tuanya. Namun cara ini tidak selalu berhasil, terbukti dari hasil pengujian yang dilakukan.

Pada pengujian U-T-05, walaupun terdapat simpul yang dihapuskan pada graf, namun graf masih sesuai dengan aturan sehingga ekstraksi tetap dilakukan dan menghasilkan *watermark* yang salah. Begitu pula dengan penambahan simpul pada U-T-06, penambahan ini tidak membuat graf menjadi menyalahi aturan sehingga ekstraksi pun tetap dilakukan dan *watermark* hasil ekstraksi menjadi salah.

Untuk aspek *efficiency*, dapat dilihat bahwa penambahan jumlah karakter pada *watermark* mengakibatkan penambahan jumlah simpul (U-E-02), penambahan waktu penyisipan (U-E-03), penambahan waktu ekstraksi (U-E-04), serta penambahan ukuran kelas Hasil (U-E-05). Ini berarti semakin panjang *watermark* yang diberikan, tingkat *efficiency* dari *watermark* pun semakin berkurang. Untuk aspek *perceptibility*, baik pada *watermark* dengan jumlah karakter banyak atau sedikit, tidak terlihat perubahan pada program. Begitu pula untuk aspek *fidelity*, program masih dapat berjalan seperti seharunya walaupun telah diberikan *watermark* dengan jumlah karakter banyak ataupun sedikit. Dapat ditarik kesimpulan bahwa panjang *watermark* tidak mempengaruhi aspek *perceptibility* dan *fidelity*.

V. KESIMPULAN

Kesimpulan yang didapat adalah sebagai berikut.

1. Tahapan dalam membuat graf PPG pada makalah ini adalah menentukan barisan angka dan membuat graf dengan jumlah anak pada setiap simpul adalah angka pada barisan tersebut.
   Tahapan penyisipan dalam metode DGW pada makalah ini adalah menentukan *watermark* dan lokasi penyisipan, mengubah *watermark* ke dalam barisan angka sesuai nilai ASCII, mengubah angka ke dalam bentuk graf PPG, membuat kode yang membangun graf, serta meletakkan kode ke lokasi yang telah ditentukan
   Sedangkan tahapan ekstraksi pada metode DGW adalah kebalikan dari tahapan penyisipan.
2. Enumerasi PPG dapat digunakan pada metode DGW. Untuk proses penyisipan, aplikasi dapat dibuat dalam bentuk Java API yang lebih memudahkan pembuat program dalam menyisipkan *watermark*.
3. Enumerasi PPG tidak terlalu kuat jika dibandingkan dengan enumerasi lainnya, yaitu PPCT. Panjang *watermark* tidak mempengaruhi kekuatan, *perceptibility*, dan *fidelity* enumerasi ini, namun semakin panjang *watermark*, semakin kecil pula nilai *efficiency* yang dimilikinya.

REFERENCES

[1] E. Carter, C. Collberg, S. Debray, A. Huntwork, C. Linn, and M. Stepp, "*Dynamic Path-Based Software Watermarking,*"*in* ACM SIGPLAN 2004, 2004, p. 1-2.

[2] C.Collberg. (2002) *SandMark : A Tool for the Study of Software Protection Algorithms.* [*Online*]. *Available* : http://sandmark.cs.arizona.edu/index.html

[3] C. Collberg and C. Thomborson, "*Software watermarking: Models and Dynamic Embeddings,*" *in 26th* ACM SIGPLAN-SIGACT, 1999, p.3, 7.

[4] S. Dacinic and J. Hamilton, "*An Evaluation of Static Java Bytecode Watermarking,*" in *International Conference on Computer Science and Applications ICCSA'10 The World Congress on Engineering and Computer Science WCECS'10,* 2010, p. 1, 3.

[5] S. Dacinic and J. Hamilton, "*An Evaluation of the Resilience of Static Java Bytecode Watermarks Against Distortive Attacks,*" in *IAENG International Journal of Computer Science,* 2011, p. 8.

[6] Y. He, "*Tamperproofing a Software Watermark by Encoding Constants,*" University of Auckland, p. 1, 11.

[7] Kossiakof Computer Center Facility. (1995) *API's and Javadoc: What Are They?.* [*Online*]. *Available* : http://apl.jhu.edu/~hall/java/beginner/api.html

[8] T. Patki. (2008) *DashO Java Obfuscator.* [*Online*]. *Available :* http://www.cs.arizona.edu/~collberg/Teaching/620/2008/Assignments/tools/DashO/index.html

[9] W. F. Zhu, "*Concepts and Techniques in Software watermarking and Obfuscation,*" University of Auckland, p. 11

[10] C. Collberg. "*Software Watermarking : Protective Terminology,*" The University of Auckland, p. 5-7.