Penggunaan *Parent Pointer Graph* untuk *Software Watermarking* Berbasis Graf Dinamis

Shofi Nur Fathiya
Institut Teknologi Bandung
if18084@students.if.itb.ac.id

Rinaldi Munir
Institut Teknologi Bandung
rinaldi-m@stei.itb.ac.id

ABSTRAK

Software watermarking adalah teknik untuk melindungi *software* dengan menyisipkan pesan berisi informasi kepemilikan. 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, *Permutation Graph*, dan *Reducible 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.*

Solusi yang dibuat terdiri dari dua bagian, yaitu sebuah penyisip *watermark* dan pengekstrak *watermark*. Dari hasil pengujian, dapat disimpulkan bahwa PPG dapat digunakan sebagai enumerasi pada metode DGW namun memiliki tingkat ketahanan yang rendah.

Kata Kunci

Dynamic Graph Watermarking, Parent Pointer Graph, software watermarking.

1. PENDAHULUAN

Software sebagai benda digital memudahkan siapa pun dalam menduplikasi, mengubah, menambahkan, dan mengurangi isinya serta menyebarkannya kembali. Tindakan ini dinilai 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* adalah teknik untuk melindungi *software* dengan menyisipkan pesan berisi informasi kepemilikan (Dacinic, 2011).

Salah satu metode yang paling kuat terhadap serangan pada *software watermarking* adalah *Dynamic Graph Watermarking* (DGW) (Zu, 2007). Metode ini memasukkan *watermark* dalam sebuah representasi graf saat proses penyisipan (He, 2002). Aplikasi DGW yang telah dibuat hingga saat ini telah mengaplikasikan beberapa jenis enumerasi graf, yaitu *Planted Plane Cubic Tree* (PPCT), Radix-k, *Permutation Graph*, dan *Reducible Permutation Graph* (Collberg, 2011).

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. Untuk itu pada makalah ini akan dibuat sebuah *software watermarking* yang yang mengaplikasikan metode DGW namun dengan enumerasi PPG.

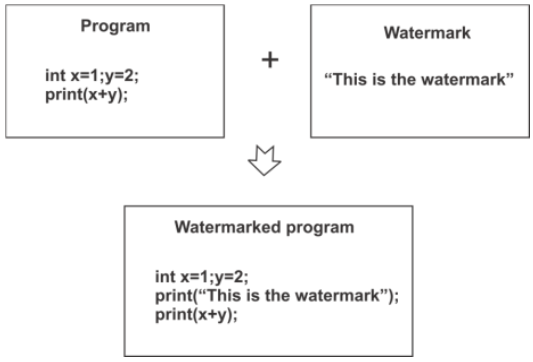
Makalah ini terdiri dari beberapa bab. Pada bab 2, diberikan penjelasan teori yang terkait dengan makalah ini. Kemudian pada bab 3, dijelaskan rancangan metode untuk solusi yang diajukan. Pada bab 4, solusi yang diajukan kemudian diaplikasikan dan diujikan. Bab terakhir, yaitu bab 5, berisi kesimpulan dari makalah ini.

2. KAJIAN TERKAIT

Pada bab 2 ini dibahas beberapa teori terkait, yaitu *software watermark*ing, PPG, DGW, serta penelitian sebelumnya yang berhubungan dengan makalah ini.

2.1 Konsep Dasar Software Watermarking

Contoh sederhana *software watermarking* dapat dilihat pada Gambar 1.



Gambar 1. Contoh *software watermarking (*Zhu, 2007)

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

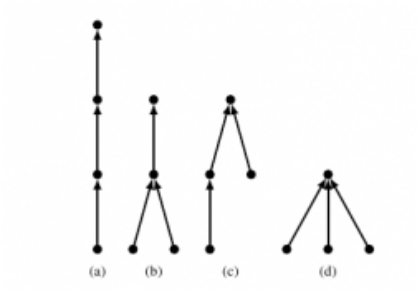
Collberg, dkk. (2001) menyatakan bahwa terdapat beberapa hal yang dapat diukur untuk mengetahui apakah *software watermark* yang disisipkan sudah efektif atau belum, yaitu :

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*.

2.2 Parent Pointer Graph

Parent Pointer Graph (PPG) merupakan teknik enumerasi graf dimana setiap simpul hanya memiliki satu *pointer*, yaitu ke arah orang tuanya saja (He, 2002). Teknik ini cukup sederhana karena jumlah *pointer* yang sedikit pada setiap grafnya, dibandingkan dengan jumlah *pointer* pada enumerasi graf lainnya. Graf PPG dibangun berdasarkan masukan sebuah angka tertentu. Angka tersebut dapat digunakan sebagai suatu representasi dalam graf, misalkan angka yang menunjukkan jumlah simpul. Gambar 2 menunjukkan beberapa graf PPG yang dapat dibangun dengan empat buah simpul.



Gambar 2. Graf yang dapat dibangun dengan empat simpul

2.3 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 (Collberg dkk, 1999). Metode ini termasuk ke dalam metode dinamik sehingga graf dibuat dan disisipkan saat eksekusi program. Ada 4 langkah yang dilakukan dalam proses penyisipan DGW (He, 2002):

1. Memilih angka *watermark*, yaitu *n.*
2. Membuat representasi graf *G* dari *n*.
3. Membuat kode *watermark* yang dapat membangun struktur graf *G* saat eksekusi program.
4. 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 penggunaan graf yang memiliki banyak *pointer* sehingga *watermark* akan sulit untuk dianalisis dan dikeluarkan dari program. Dan karena DGW dibangun secara dinamis, informasi saat eksekusi harus dikumpulkan untuk dapat menganalisis struktur *watermark.*

He (2002) menyatakan bahwa terdapat eberapa serangan pada metode DGW yang dapat merusak atau bahkan menghapus *watermark* dari dalam program, yaitu :

- *Additive attack* : menyisipkan *watermark* baru ke dalam program yang sudah memiliki *watermark.*
- *Distortive attack : m*engubah sebagian *watermark* sehingga tidak dikenali kembali.
- *Subtractive attack : m*enghilangkan seluruh bagian *watermark* di program.

2.4 Penelitian Terkait

Terdapat beberapa penelitian yang terkait dengan makalah ini. Penelitian pertama adalah tesis yang dibuat oleh Yong He dari *University of Auckland* dengan judul "*Tamperproofing a Software Watermark by Encoding Constants*" (He, 2002). Isi dari tesis ini adalah penambahan metode *encoding constant* pada DGW dengan enumerasi PPCT. Penelitian lainnya yaitu sistem Sandmark yang dibuat oleh Collberg dan Townsend dari *University of Arizona* (Collberg, 2011). Beragam teknik *watermarking* diaplikasikan pada sistem ini, namun belum diterapkan metode DGW dengan enumerasi PPG.

Penelitian terkait selanjutnya yaitu sistem DashO (Patki, 2011). 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 yang terdiri dari 4 instruksi *push integer* dan diikuti oleh 4 instruksi *pop* (Dacinic dkk, 2011).

3. METODE PENYELESAIAN

Dengan menganalisis kelebihan yang dimiliki oleh enumerasi PPG dibandingkan dengan enumerasi lainnya serta menganalisis metode DGW sebagai metode yang memanfaatkan struktur graf dalam menyisipkan *watermark*, pada makalah ini diajukan solusi untuk membuat *software watermarking* metode DGW yang menggunakan PPG sebagai enumerasinya. PPG yang merupakan enumerasi sederhana diharapkan dapat membuat proses penyisipan *watermark* menjadi lebih sederhana namun tetap kuat terhadap serangan.

Solusi yang dibuat ini terbatas pada program Java, yang artinya program yang dapat menggunakan solusi ini untuk menyisipkan *watermark* di dalamnya hanyalah program Java saja. Bagian *software watermarking* untuk proses ekstraksi dibentuk sebagai sebuah API (*Application Programming Interface*) sehingga pembuat program hanya perlu menambahkan API ini di programnya untuk menyisipkan *watermark* sehingga pembuat program dapat menyisipkan *watermark* lebih mudah digunakan dibandingkan aplikasi biasa.

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.*

Untuk memudahkan penjelasan solusi, digunakan beberapa notasi dalam pembahasan metode solusi yang diajukan. Daftar notasi yang digunakan dapat dilihat pada Tabel 1.

Tabel 1. Notasi yang Digunakan

Notasi	Deskripsi
P	program yang akan diberi *watermark*
W	*watermark* yang ingin disisipkan bertipe *string*
P'	program yang telah memiliki *watermark*
mark	metode pada API Penyisip *Watermark* yang berfungsi menandai lokasi penyisipan *watermark* di dalam program *P*
metode *M*	metode pada API Penyisip *Watermark* yang berfungsi menerima masukan *watermark W* dan memanggil perintah untuk memulai penyisipan
k	jumlah subgraf yang dibentuk dari *G*
G	graf yang dibangun dari *W*

3.1 API Penyisip Watermark

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 metode *mark* dan metode *M.*

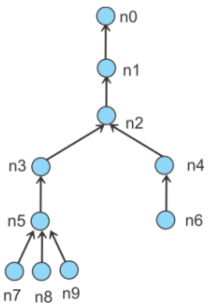
3.1.1 Pembuatan Graf dari String Watermark

Pembuatan graf PPG dari sebuah angka dapat dilakukan dengan cara bermacam-macam. Gambar 2 menunjukkan graf PPG yang dapat dibangun dengan menganggap angka masukan sebagai jumlah simpul. Namun pada solusi yang diajukan ini, angka yang dimasukkan dianggap sebagai jumlah anak setiap simpul pada graf. Langkah-langkah yang dilakukan dalam membuat graf ini adalah:

1. Diambil 3 digit nilai ASCII dari setiap karakter pada *W*.
2. Nilai-nilai ASCII ini kemudian digabungkan menjadi satu barisan angka.
3. Dibuat graf *G* dimana setiap digit angka pada barisan ini menunjukkan jumlah anak setiap simpul pada *G*.

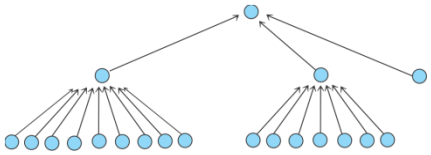
Contoh kasus dalam pembuatan graf dari string *watermark* ini adalah sebagai berikut.

1. Misalkan nilai *W* adalah "pq".
2. 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. Untuk barisan angka 112113, graf yang dihasilkan ditunjukkan pada Gambar 3.



Gambar 3. Graf untuk *watermark* bernilai "pq"

Pada langkah 3 ini terdapat hal yang perlu diperhatikan, yaitu ketika barisan angka tidak dapat membentuk graf. 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 4.



Gambar 4. Graf untuk *watermark* bernilai "a"

3.1.2 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*. Di dalam program *P* dituliskan beberapa *mark* di tempat yang diinginkan dan metode *M* yang menerima masukan *watermark W*.
2. Program P kemudian dibangun (*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. *mark* yang tidak dieksekusi tidak akan dihitung dan tidak akan digantikan dengan metode pembangun *watermark.*
3. Graf *G* dari *watermark W* dibentuk seperti yang telah dijelaskan pada subbab 3.1.1.
4. Nilai *k* dicari untuk nantinya membagi graf *G* menjadi sejumlah *k*. Nilai *k* diambil dari jumlah metode *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.
5. Graf *G* dibagi menjadi *k* buah subgraf.
6. API lalu membuat kelas *watermark* yang merupakan representasi 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 metode pada kelas Hasil yang akan membangun subgraf tersebut.
7. Penanda di dalam program *P* diganti dengan metode dari kelas Hasil dan metode *M* dihapuskan. Hasil pengubahan program P ini merupakan program *P'*. Penyisipan *watermark* pun selesai dilakukan.

Dalam subbab 3.1.2 ini, dibuat pula contoh kasus untuk mempermudah penjelasan. Contoh kasus ini dikaitkan dengan contoh kasus pada bagian 3.1.1., yaitu pembuatan graf dari *watermark.*

1. Program *P* yang telah diberi metode *mark* dan metode *M* dapat dilihat pada Gambar 5.

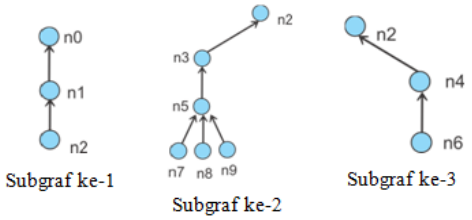
```
public static void main(String[] args) {
    // TODO code application logic here
    String sapa = "Hello World";
    int a = 0;
    int b = 10;
    Tanda.beriPenanda();

    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 5. Program *P* dengan *mark* dan metode *M*

2. API menemukan 3 buah *mark* pada program *P*.
3. Dari *watermark* bernilai "pq" seperti pada contoh kasus subbab 3.1.1., graf *G* yang dihasilkan tampak seperti Gambar 3.
4. Nilai *k* yang diambil adalah jumlah *mark*, yaitu 3.
5. Hasil pembagian subgraf *G* terdapat pada Gambar 6.



Gambar 6. Hasil pembagian graf *G*

6. Kelas HashSimpul dan kelas Hasil dapat dilihat pada Gambar 8 dan Gambar 9.
7. Setelah metode *mark* dan metode *M* dihapuskan, dihasilkan program *P'* seperti pada Gambar 7.

```
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 7. 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 8. Kelas Hasil

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

Gambar 9. Kelas HashSimpul

3.2 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 dahulu mengubah program *P'* yang merupakan file .jar menjadi file .class. Selanjutnya, baris-baris yang mungkin menjadi 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. Untuk mengubah graf *G* menjadi *watermark* W, dapat dilakukan langkah-langkah berikut.

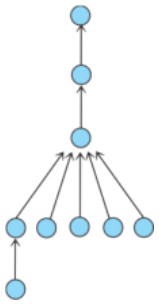
1. Jumlah anak pada masing-masing simpul dihitung.
2. Deretan angka 0 terakhir dihapuskan karena kemungkinan besar bukan barisan angka *watermark.*
3. Setiap 3 digit angka dikonversi ke karakter menurut nilai ASCII dari karakter yang dicari.

Sebagai contoh, ditemukan graf *G* seperti pada Gambar 3. Untuk mendapatkan nilai *watermark W* dilakukan 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. Setelah deretan angka 0 terakhir dihapuskan, angka yang tersisa adalah 1, 1, 2, 1, 1, dan 3 sehingga barisan yang terbentuk adalah 112113.
3. Tiga digit pertama yaitu 112 diubah menjadi '*P'* dan 3 digit selanjutnya diubah menjadi 'q' sehingga didapat *watermark* bernilai "pq".

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 menjadi kelipatan 3.



Gambar 10. Graf *G1*

Pada Gambar 10, jumlah anak setiap simpul di *G1* 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.

4. HASIL DAN PEMBAHASAN

Pada bab 4 ini diberikan hasil pengujian yang dilakukan setelah mengaplikasikan metode yang telah dirancang pada bab 3. Selain itu, diberikan pula analisis dari hasil pengujian tersebut.

4.1 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 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 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 2.

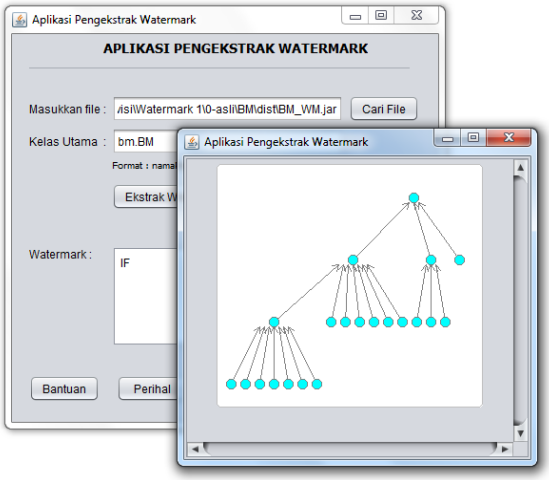
Tabel 2. Hasil Pengujian API Penyisip *Watermark*

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

Hal yang diujikan pada Aplikasi Pengekstrak *Watermark* 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 3. Hasil pengujian *watermark*, "IF" dapat dilihat pada Gambar 11.

Tabel 3. 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 11. 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*). *Additive attack* ditunjukkan pada ID U-T-01 sementara *distortive attack* ditunjukkan oleh ID U-T-02 hingga U-T-06. *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 4.

Sementara pada Tabel 5, diberikan hasil pengujian pada aspek keefektifan lainnya, yaitu *efficiency* (U-E-01 sampai U-E-04), *perceptibility* (U-E-05), dan *fidelity* (U-E-06).

Tabel 4. Hasil Pengujian Ketahanan

ID	Serangan	*Watermark*		
		1	2	3
U-T-01	Menambahkan *watermark* baru	O	O	O
U-T-02	Menghapus penghubung 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

Tabel 5. Hasil Pengujian Keefektifan

ID	Serangan	*Watermark*		
		1	2	3

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

4.2 Analisis Hasil Pengujian

Bagian ini berisi analisis dari hasil pengujian yang didapat pada subbab 4.1.

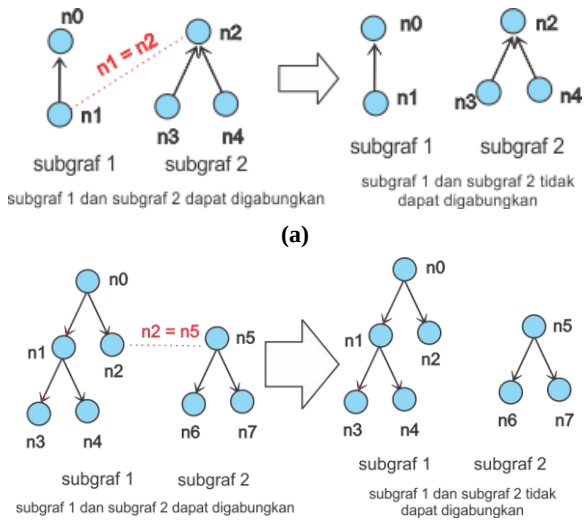
4.2.1 Analisis Hasil Pengujian Kebutuhan

Seluruh hasil dari pengujian pada API Penyisip *Watermark* dan Aplikasi Pengekstrak *Watermark* yang ditunjukkan pada Tabel 2 dan Tabel 3 adalah berhasil. Ini menunjukkan perangkat lunak yang dibuat sudah memenuhi kebutuhan yang diperlukan dalam *software watermarking*.

4.2.2 Analisis Hasil Pengujian Ketahanan

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 API ini, maka tidak akan dibangun kembali *watermark* di dalamnya. Ini berarti program yang dibuat tahan terhadap *additive attack*.

Untuk analisis pengujian terhadap *distortive attack*, diberikan contoh kasus dengan gambar apabila masing-masing *distortive attack* ini diberikan kepada enumerasi graf lain sebagai bentuk perbandingan. Enumerasi yang dipilih adalah enumerasi PPCT. Ilustrasi kasus penghapusan penghubung subgraf (U-T-02) dapat dilihat pada Gambar 12.

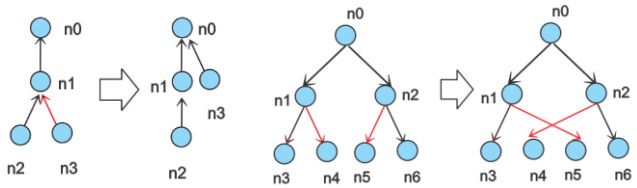


Gambar 12. Penghapusan penghubung subgraf pada (a) PPG dan (b) PPCT

Dalam pengujian U-T-02 ini, ketiga *watermark* masih dapat diekstraksi dengan benar walaupun terdapat penghubung antar subgraf yang dihapuskan. Hal ini dikarenakan kerusakan akibat penghapusan penghubung subgraf yang ditunjukkan pada Gambar 12 (a) untuk enumerasi PPG tidak terlalu besar sehingga saat ekstraksi dilakukan, subgraf yang tidak terhubung masih dapat disambungkan kembali. Begitu pula dengan enumerasi PPCT, yang ditunjukkan pada Gambar 12 (b).

Di pengujian U-T-03 seperti yang dapat dilihat pada Gambar 13 (a), simpul n3 yang semula mengarah ke simpul n1 dibuat menjadi mengarah ke simpul n0. Saat proses ekstraksi dilakukan, graf hasil pengubahan ini tetap terbaca karena tetap sesuai aturan, dimana setiap simpul memiliki satu *pointer* ke simpul lainnya. Dan karena graf yang dibaca merupakan graf yang salah, *watermark* yang dihasilkan pun menjadi salah.

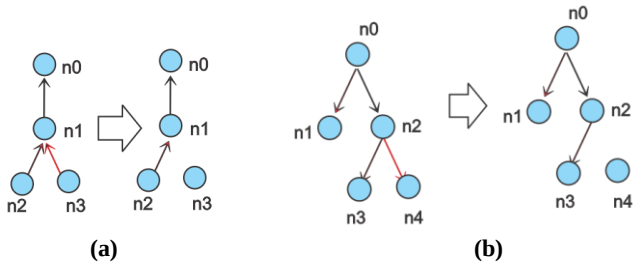
Untuk kasus pada graf PPCT seperti pada Gambar 13 (b), dapat dilihat bahwa graf hasil pengubahan pun tetap memenuhi aturan, dimana setiap simpul memiliki dua *pointer* ke simpul lainnya. Hal ini menyebabkan graf hasil pengubahan pun tetap akan dibaca dan menghasilkan *watermark* yang salah, seperti halnya pada graf PPG.



Gambar 13. Pengubahan arah *pointer* pada (a) PPG dan (b) PPCT

Pada pengujian U-T-04, *pointer* pada sebuah simpul dihapuskan seperti yang dapat dilihat pada Gambar 14 (a) dan graf PPG gagal mendapatkan kembali *watermark* yang benar pada *watermark* W2. Saat ekstraksi dilakukan, Aplikasi Pengekstrak *Watermark* memberikan *pointer* baru untuk simpul yang tidak memiliki *pointer* dan mencoba memasangkan *pointer* tersebut ke simpul lainnya dengan mengacu pada simpul di sebelahnya. Pemasangan *pointer* baru ini tidak selalu berhasil, seperti yang dapat dilihat pada Tabel 4 dimana untuk W1 dan W3, *watermark* masih dapat diambil kembali, sementara pada W2, *watermark* yang diambil sudah berubah nilai.

Selanjutnya, bila kasus ini diterapkan pada graf PPCT, akan menunjukkan hasil yang berbeda. Pada Gambar 14 (b), *pointer* n2 yang dihapuskan membuat simpul n4 tidak terhubung dengan graf. Untuk menyambungkan simpul n4 kembali, diperlukan sebuah *pointer* dari simpul lain. Jika dilakukan pencarian, maka dapat diketahui bahwa simpul n2 hanya memiliki satu buah *pointer* sehingga pada simpul n2 dapat dibuat *pointer* baru yang terhubung dengan simpul n4. Hal ini menunjukkan bahwa pada kasus penghapusan *pointer*, PPCT lebih kuat dibandingkan dengan PPG.

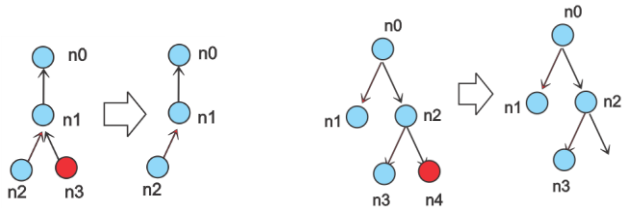


(a) **(b)**

Gambar 14. Penghapusan sebuah *pointer* pada (a) PPG dan (b) PPCT

Pada pengujian U-T-05, sebuah simpul daun, yaitu n3, dihapuskan dari graf seperti yang dapat dilihat pada Gambar 15 (a). Karena simpul n3 dihapus, *pointer* milik simpul n3 pun dihapus Namun karena graf hasil penghapusan simpul tetap memenuhi kriteria graf PPG, maka tidak dideteksi terdapat kesalahan seperti pada U-T-04 dan tetap dilakukan pembacaan terhadap graf hasil penghapusan. Karena graf tersebut salah, maka *watermark* hasil ekstraksi pun salah.

Selanjutnya, dilakukan perbandingan jika kasus ini diterapkan pada graf PPCT. Ilustrasi dari penghapusan simpul pada graf PPCT dapat dilihat pada Gambar 15 (b). Penghapusan simpul n4 membuat *pointer* kanan pada simpul n2 kehilangan simpul tujuan sehingga dapat dideteksi bahwa seharusnya terdapat simpul di bagian yang ditunjuk oleh *pointer* kanan tersebut. Karena itu, simpul baru pun dapat dibuat kembali dan digabungkan dengan simpul n2.

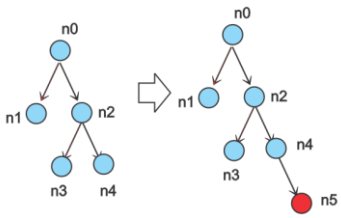


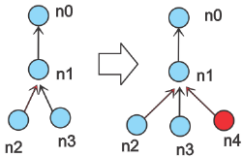
(a) **(b)**

Gambar 15. Penghapusan simpul pada (a) PPG dan (b) PPCT

Pada pengujian terakhir, yaitu pengujian U-T-06, sebuah simpul ditambahkan pada graf PPG, seperti yang dapat dilihat pada Gambar 16 (a). Seperti halnya pada kasus sebelumnya, graf PPG tidak mendeteksi adanya kesalahan pada graf. Graf tersebut pun tetap dibaca dan akhirnya membaca dan menghasilkan *watermark* yang salah.

Ilustrasi jika kasus ini diterapkan pada graf PPCT dapat dilihat pada Gambar 16 (b). Graf hasil penambahan simpul menunjukkan adanya kesalahan, dimana simpul n4 hanya memiliki satu *pointer*. Namun, hal ini tidak berarti langsung diketahui bahwa simpul n5 merupakan simpul tambahan. Jika semula simpul n4 memiliki dua buah *pointer* dimana *pointer* kiri tersebut menunjuk ke simpul lain, kemudian simpul tersebut beserta *pointer* kiri dari simpul n4 dihapuskan, maka keadaan graf akan menjadi seperti pada Gambar 16 (b). Hal ini menunjukkan bahwa dalam kasus penambahan simpul, seperti halnya graf PPG, graf PPCT belum tentu dapat menyelesaikan kasus tersebut.





(a) **(b)**

Gambar 16. Penghapusan sebuah *pointer* pada (a) PPG dan (b) PPCT

Dari pengujian ketahanan ini, dapat diketahui bahwa penjang *watermark* tidak berpengaruh dalam meningkatkan ketahanan *watermark*, dimana *watermark* dengan jumlah karakter lebih banyak pun dapat mudah rusak. Namun bila dilihat dari hasil ekstraksi, *watermark* yang lebih panjang dapat lebih terbaca atau tersampaikan informasi di dalamnya walaupun rusak dibandingkan dengan *watermark* yang lebih pendek.

4.2.3 Analisis Hasil Pengujian Keefektifan

Untuk aspek *efficiency*, penambahan jumlah karakter pada *watermark* mengakibatkan penambahan jumlah simpul, penambahan waktu penyisipan, penambahan waktu ekstraksi, serta penambahan ukuran kelas Hasil. 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 seharusnya walaupun diberikan *watermark* berjumlah karakter banyak ataupun sedikit. Dapat ditarik kesimpulan bahwa panjang *watermark* tidak mempengaruhi aspek *perceptibility* dan *fidelity*.

5. KESIMPULAN DAN SARAN

Kesimpulan yang didapat adalah sebagai berikut.

1. Langkah membuat graf PPG pada makalah ini adalah menentukan barisan angka dan membuat graf dengan jumlah anak setiap simpul adalah angka 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 adalah kebalikan dari tahapan penyisipan.

2. PPG dapat digunakan pada DGW dengan hasil kode yang dibentuk lebih sederhana namun tidak terlalu kuat terhadap serangan jika dibandingkan dengan PPCT. Panjang *watermark* tidak mempengaruhi kekuatan, *perceptibility*, dan *fidelity* enumerasi ini, namun semakin panjang *watermark*, semakin kecil pula nilai *efficiency* yang dimilikinya.

Untuk penelitian selanjutnya, saran yang diberikan adalah bagian enumerasi PPG pada DGW perlu diperbaiki agar *watermark* yang tersimpan dapat menjadi lebih kuat terhadap serangan namun dengan graf yang tetap sederhana.

6. DAFTAR REFERENSI

Carter, E., Collberg, C., Debray, S., Huntwork, A., Linn, C., Stepp, M. *Dynamic Path-Based Software watermarking*. ACM SIGPLAN 2004. 1-2. 2004.

Collberg, C. *SandMark : A Tool for the Study of Software Protection Algorithms*. http://sandmark.cs.arizona.edu/index.html. Diakses pada 8 November 2011.

Collberg, C., Thomborson, C. *Software watermarking: Models and Dynamic Embeddings.* POPL '99. *26th* ACM SIGPLAN-SIGACT. 3, 7. 1999.

Collberg, C., Nagra, J., Thomborson, C. *Software watermarking: Protective Terminology.* Australian Computer Science Conference, ACSC 2002. 5-7. 2001.

Dacinic, S., Hamilton, J. *An Evaluation of the Resilience of Static Java Bytecode Watermarks Against Distortive Attacks*. *IAENG International Journal of Computer Science*. 1, 8. 2011.

He, Y. *Tamperproofing a Software Watermark by Encoding Constants*. University of Auckland. 1, 11, 16. 2002.

Patki, T. *DashO Java Obfuscator*. http://www.cs.arizona.edu/~collberg/Teaching/620/2008/Assignments/tools/DashO/index.html. Diakses pada 30 Desember 2011.

Zu, W. F. *Concepts and Techniques in Software watermarking and Obfuscation.* University of Auckland. 11. 2007