STUDI TENTANG *SOFTWARE WATERMARKING*

Febrian Aris Rosadi – NIM : 13503041

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13041@students.if.itb.ac.id

Abstrak

Digital watermarking telah menjadi bagian dari teknik yang digunakan untuk melindungi berbagai macam media dalam bentuk digital dari masalah-masalah penggandaan ilegal, pemalsuan dan bukti kepemilikan. Bentuk media yang dilindungi oleh *digital watermarking* tersebut antara lain citra, audio, video, teks, dan dokumen. Media lain yang tidak boleh dilupakan sebagai suatu bentuk karya intelektual digital yang perlu dilindungi adalah perangkat lunak atau *software*. Penggunaan teknik *digital watermarking* pada media inang perangkat lunak atau *software* acap kali disebut sebagai *software watermarking*.

Makalah ini membahas bagaimana *software watermarking* membantu mengatasi masalah-masalah yang berkaitan dengan perlindungan perangkat lunak. Juga membahas model formal dari *software watermarking* dan kemudian memperkenalkan teknik-teknik yang digunakan dalam *software watermarking*. Beberapa hal yang menyangkut *software watermarking* juga diikut sertakan dalam makalah ini antara lain aspek-aspek apa saja yang diharapkan dari sebuah *software watermarking*, peluang serangan pada *software watermarking*, aplikasi dari *sofware watermarking* dan kemudian bagaimana evaluasi dilakukan terhadap *watermarking* yang berbeda.

Kata kunci*: Software watermarking, Subtractive Attack, Additive Attack, Distortive Attack, ,Colussive Attack, Dynamic Watermarking, Static Watermarking*

1 Pendahuluan

Selama perkembangan kehidupan manusia, seringkali timbul permasalahan untuk melindungi suatu kekayaan intelektual. Apalagi ketika memasuki jaman digital beberapa dekade terakhir, masalah berkembang menjadi semakin luas dengan banyaknya karya intelektual yang direpresentasikan dalam bentuk digital seperti teks, citra, audio maupun video. Karya-karya tersebut akan mengikuti karakteristik dari format digital diantaranya :

1. Mudahnya penggandaan (*copy*) dengan hasil tepat sama dngan aslinya.
2. Mudahnya penyimpanan dan pendistribusian melalui media penyimpanan seperti *hard disk* dan *compact disk.* Apalagi dengan perkembangan internet yang sangat pesat dalam satu dekade terakhir, maka penyebaran data digital semakin pesat.
3. Perubahan yang tidak signifikan pada data dapat tidak dikenali.

Masalah muncul ketika timbul kebutuhan untuk melindung kekayaan intelektual dalam bentuk digital. Antara lain :

1. Masalah kepemilikan. Suatu pihak dapat menyalin karya asli dan mengaku karya tersebut sebagai miliknya. Pemilik asli tidak dapat memberikan bantahan karena tidak ada bukti yang otentik menandakan kepemilikan.
2. Pelanggaran *copyright*. Merupakan penyalinan yang tidak berijin sehingga mengakibatkan kerugian dari pencipta karya karena tidak diperolehnya royalti apapun dari penggandaan tersebut.
3. Masalah keaslian. Karya digital dapat dengan mudah diubah, sehingga dapat menyebabkan hilangnya atau berubahnya data-data yang penting dari sebuah karya.

Maka diperkukan suatu teknik yang mapu melindungi karya cipta seseorang atau suatu pihak dalam bentuk digital, dari bentuk-bentuk perubahan yang merusak, penggandaan secara ilegal dan pemalsuan. Teknik yang selama ini

banyak digunakan adalah dengan menggunakan *digital watermarking*.

Salah satu bentuk kekayaan intelektual dalam bentuk digital yang tidak boleh kita lupakan adalah *software* atau perangkat lunak (kedua istilah ini akan dipakai bergantian di makalah ini untuk mengantisipasi pergantian istilah menjadi cukup asing dan tidak familiar). Perangkat lunak adalah mutlak merupakan hasil kerja keras intelektual dari pihak pengembang yang patut dilindungi. Sayangnya perkembangan akhir-akhir ini mengarah pada semakin berkurangnya penghargaan pada kekayaan intelektual berupa perangkat lunak ini. Semakin berkembangnya kejahatan-kejahatan baik berupa penggandaan program perangkat lunak tanpa membayar, usaha merusak sebuah perangkat lunak, *malicious reverse engineering*, dan lain – lain.

Diajukanlah beberapa teknik *software protection* yang diantaranya adalah berupa menerapkan teknik *digital watermarking* dalam media *software* yang dinamakan *software watermarking*.

2 *Digital Watermarking*

Digital watermarking dapat diarktikan sebagai teknik untuk menyisipkan informasi tertentu, yang disebut *watermark*, ke dalam data digital. *Watermark* dapat berupa teks seperti informasi copyright, gambar, maupum logo, data audio dan rangkaian bit yang pada dasarnya tidak bermakna untuk pihak yang tidak berkepentingan.

Watermark ini disisipkan ke dalam media inang yang bisa berupa data audio, video, citra dan teks, sehingga *watermark* ini tidak bisa diketahui oleh indra manusia. *Watermark* ini harus dapat diekstraksi dengan sebuah kunci. *Watermark* ini harus dapat tangguh, tidak mudah diubah, sehingga ketika media inangnya disalin dan didistribusikan, *watermark* yang disisipkan di dalamnya juga tidak mengalami perubahan sehingga akhirnya dapat diekstraksi kembali.

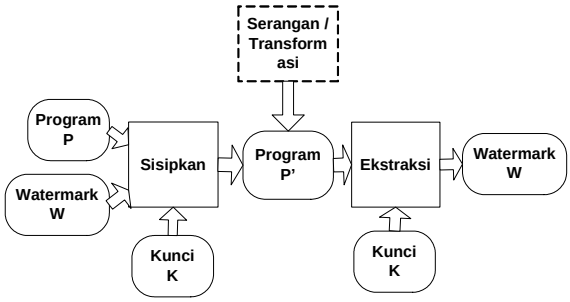
Yang akan dibahas disini adalah *digital watermarking* pada media inang berupa perangkat lunak yang umum dinamakan sebagai *software watermarking*.

3 *Software watermarking*

Software watermarking dapat dideskripsikan sebagai teknik untuk menyisipkan sebuah struktur W pada program P dengan kunci K tertentu sehingga suatu ketika, struktur W tersebut dapat diekstraksi kembali dari program yang telah disisipi *watermark* tersebut.

```
sisipkan(P,W,K) → P'
ekstraksi(P',K) → W
```

demikian sehingga W memiliki kekenyalan (*resilient*) dalam menerima serangan ataupun transformasi program, W memiliki data rate yang tinggi, dan W dapat tersembunyi dan tidak diketahui keberadaannya dari luar (*stealthy*). Yang tidak kalah pentingnya W harus tidak menyebabkan penurunan kinerja dari perangkat lunak yang menjadi inang.



Gambar 1 Skema *Software watermarking*

Properti yang harus dimiliki :

1. *Watermark* yang susah untuk dihancurkan
2. Susah untuk dikenali
3. Memiliki bit rate yang tinggi
4. Memberikan gangguan akan performansi yang paling rendah
5. Satu *watermark* untuk perlindungan semua program

Namun pada dasarnya ada 3 aspek yang perlu diperhatikan dalam *software watermarking* :

1. *Resilient* atau kekenyalan

   Adalah sejauh mana teknik *software watermarking* yang digunakan, mampu bertahan dari serangan yang dilakukan oleh pihak tidak berkepentingan yang bertujuan untuk membuat *watermark* yang disisipkan menjadi tidak berguna. Semakin tinggi *resilient* suatu teknik *software watermarking*, semakin besar kemungkinan

dia bertahan dari adanya serangan-serangan merusak.

2. *Stealthy* atau ketidakkentaraan

   Keberadaan *watermark* pada *software watermarking* harus tidak dikenali oleh penyerang. Hal ini akan mempersulit penyerang untuk mengetahui lokasi dari *watermark* yang disisipkan dan melakukan serangan lebih lanjut. Semakin tidak kentara adanya *watermark* pada perangkat lunak yang disisipi adalah semakin baik teknik tersebut.

3. Data rate

   Adalah seberapa besar data yang dapat ditampung oleh *watermark* yang disisipkan *software watermarking* pada perangkat lunak. Semakin besar data yang dapat ditampung adalah semakin baik teknik *software watermarking* yang digunakan.

Hal yang seringkali menjadi aspek tambahan yang harus diperhatikan, adalah dengan adanya *watermark* yang disisipkan pada perangkat lunak, tidak boleh terjadi penurunan kinerja perangkat lunak tersebut .

4 Mengapa menggunakan *Software watermarking*

Terdapat beberapa peluang kejahatan dalam pengembangan perangkat lunak dimana *software watermarking* dapat membantu untuk mengatasi hal tersebut. Kejahatan tersebut beserta ilustrasinya antara lain sebagai berikut :

1. *Malicious Reverse Engineering*

Alice adalah seorang *software* developer. Alice memproduksi sebuah *software* dengan beberapa modul didalamnya. Ternyata Bob sebagai pesaing Alice membeli produk yang dihasilkan oleh Alice untuk kemudian melakukan reverse engineering, sehingga diperoleh modul-modul yang dibutuhkan oleh Bob dan dimasukkan ke program yang dibuat sendiri oleh Bob, dan kemudian menjualnya sebagai bagian dari program yang diproduksi oleh Bob. Alice dapat menggunakan *software watermarking* untuk meengetahui apakah didalam program Bob telah disisipkan modul yang sebenarnya adalah kekayaan intelektual yang dimilikinya.

2. Pembajakan Perangkat Lunak

Alice adalah seorang *software developer*. Ia menjual produknya ke banyak pihak termasuk Bob. Bob termasuk adalah orang yang membeli produk secara legal dari Alice. Namun Bob secara ilegal membuat salinan dari program tersebut dan menjualnya kembali ke orang lain. Dengan *software watermarking*, Alice dapat memberikan identifikasi berbeda-beda untuk tiap distibusi *software*nya, dan ketika dia menemukan adanya salinan ilegal dari *software* yang dihasilkannya, Alice dapat membuktikan bahwa *software* tersebut miliknya dan melacak dari *watermark* unik yang ada di dalamnya untuk mencari tahu siapakah dari daftar penerima distribusi yang telah membuat salinan ilegal dari *software* tersebut. Akhirnya Alice dapat menghentikan distribusi *software* ke Bob karena *watermark* secara unik menunjuk Bob sebagai orang yang telah membuat salinan ilegal.

Dari berbagai permasalahan yang timbul diatas dapat dijelakan, tujuan diterapkannya softeware *watermarking* antara lain :

1. Untuk mendorong tidak terjadinya penggandaan ilegal dan pendistribusian hasil penggandaan ilegal tersebut.
2. Sebauh catatan *copyright* yang dapat digunakan untuk menyediakan bukti kepemilikan
3. Sebuh fingerprint dapat digunakan untuk melacak sumber dari pendistribusian ilegal

Namum sayangnya, *software watermarking* tidak mencegah dilakukannya penggandaan ilegal dan pendistribusian hasil penggandaan ilegal tersebut. Pihak lain dapat melakukan penggandaan dan tetap menggunakan program sebagaimana mestinya, karena *watermarking* yang disisipkan dalam perangkat lunak tersebut tidak akan mengakibatkan perubahan fungsionalitas untuk memenuhi aspek ketidakkentaraan, kecuali *software watermarking* tadi dikombinasikan dengan teknik perlindungan lain yang akan menonaktifkan semua fitur ketika otentikasi *watermark* dari perangkat lunak yang disediakan tidak sesuai dengan jumlah daftar pemegang lisensi yang seharusnya ada.

Pada banyak literatur, berdasarkan tujuan yang ada, maka *software watermarking* menjadi dua topik yang lebih khusus yaitu *watermarking* dan *fingerprinting*.

Watermarking

- Mendorong tidak dilakukannya pencurian kekayan intelektual
- Sebagai bukti terjadinya pencurian

Fingerprinting

- Sebagai alat Bantu untuk melacak terjadinya pencurian

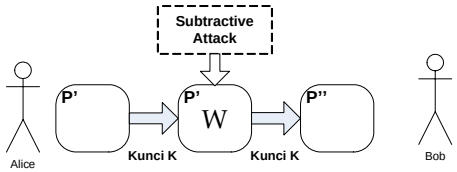
5 Peluang Serangan pada *Software watermarking*

Alice talah menyisipkan *watermark* W pada objek *software* O yang menjadi inang, sebelum menjual O untuk digunakan hanya oleh Bob. Agar Bob dapat menjual O pada orang lain, maka Bob harus dapat membuat *watermark* W menjadi tidak berguna. Jika tidak, maka Alice dapat mengetahui dari ekstraksi W, bahwa Bob adalah orang yang telah melanggar kekayaan intelektual yang dimiliki Alice.

Serangan-serangan yang dapat dilakukan pada *Software watermarking* dapat dikategorikan sebagai berikut :

1 Substractive Attack

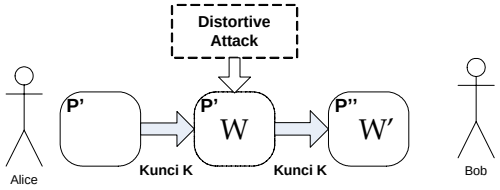
Serangan ini dilakukan untuk mengambil struktur W dari objek O. Untuk itu penyerang harus mengetahui keberadaan dan lokasi dari W.



Gambar 2 *Subtractive Attack*

2 Distortive Attack

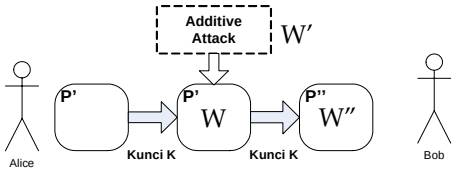
Jika keberadaan dan lokasi dari W tidak dapat dikenali, maka penyerang dapat menerapkan sehimpunan transformasi atau perubahan pada obyek O, dengan harapan W akan ikut mengalami tranformasi menjadi W' dan tidak dapat dikenali lagi.



Gambar 3 *Distortive Attack*

3 Additive Attack

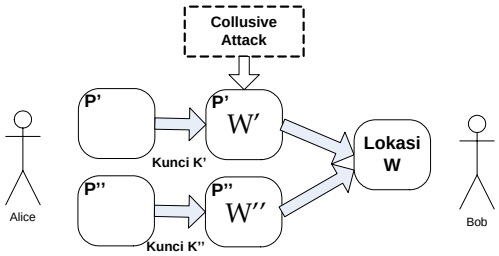
Penyerang akan menambahkan W' miliknya sendiri pada objek O sehingga dapat membuat W awal tidak dapat atau susah untuk diekstraksi.



Gambar 4 *Additive Attack*

4 Collusive Attack

Penyerang membandingkan dua objek O berbeda, yang telah disisipkan *watermark* yang berbeda, agar dapat menentukan lokasi dari *watermark* yang ada.



Gambar 5 *Colussive Attack*

6 Model Formal dari *Software watermarking*

Agar dapat secara legal mengklaim kepemilikan dari sebuah *watermark*, kita harus dapat menunjukkan bahwa pengenalan dari *watermark* tersebut adalah bukan sebuah even kebetulan.

Definisi 1 : *Software watermarking*

Jika **W** adalah sebuah struktur matematis dan p adalah sebuah predikat, maka $\forall w \in W : p(w)$. Kita harus memilih *p* dan *w* yang memiliki kemungkinan kecil terjadinya *p(x)* untuk sebuah ***x*** yang dipilih secara acak dan $x \notin W$.

Seperti nanti akan kita lihat, *watermark* dapat disisipkan baik di dalam teks program maupun di state program ketika dijalankan dengan himpunan input tertentu. Sehingga serangan dapat dilakukan juga baik pada teks program maupun pada state program.

Definisi 2 : Program

Jika **P** adalah sebuah himpunan program, P_W adalah berarti menyisipkan *watermark* $w \in W$ pada program $P \in \mathbf{P}$. dom(*P*) adalah sebuah himpunan sekuens dari masukan/input yang diterima oleh program *P* dan out(*P,I*) adalah keluaran/output dari program P jika menerima masukan *I*. *S(P,I)* adalah state internal dari program *P* setelah memproses masukan *I*, |*S(P,I)* | adalah ukuran dari state

Definisi 3 : Transformasi Program

Kekenyalan (*Resilient*) sebuah *watermark* w dari Program P_W ddidefinisikan dari serangan de-*watermarking* yang dapat dilakukan pada P_W Serangan adalah berupa transformasi program yang dapat dikategorikan sebagai *semantic preserving* (jika serangan tetap menjaga tingkah laku dari masukan-keluaran), *state preserving* (jika serangan tetap menjaga tingkah laku dari state), dan *cropping* (jika bukan merupakan *semantic preserving*).

Definisi 4 : Transformasi Program

Jika T adalah himpunan transformasi dari program ke program. $T_{sem} \subset T$ adalah Transformasi semantic preserving. $T_{state} \subset T$ adalah himpunan transformasi state preserving dan. $T_{crop} \subset T$ adalah himpunan transformasi yang bukan *semantic preserving*.

$$T_{sem} = \{ t : T \mid P \in P, I \in \mathrm{dom}(P), \mathrm{dom}(P) = \mathrm{dom}(t(P)), \mathrm{out}(P,I) = \mathrm{out}(t(P),I) \}$$

$$T_{state} = \{ t : T \mid P \in P, I \in \mathrm{dom}(P), S(P) = S(t(P),I) \}$$

$$T_{crop} = \{ t : T \mid \exists P \in P, \exists I \in \mathrm{dom}(P), (I \notin \mathrm{dom}(t(P)) \vee \mathrm{out}(P) \neq \mathrm{out}(t(P),I)) \}$$

Jika sebuah transfomasi adalah *state preserving* maka dia juga *semantic preserving*. Namun tidak sebaliknya, banyak transformasi *semantic preserving* yang tidak *state preserving*(seperti transformasi optimasi kode).

Definisi 4 : Pengenalan *Watermark*

Sebuah *watermark* $w \in W$ di dalam program $P_w \in \boldsymbol{P}$ dikatakan dapat dikenali (recognizable) setelah melalui sehimpunan transformasi $T \in \mathbf{T}$ jika terdapat sebuah pengenal R
$R_T : (P \times S) \rightarrow W$
Dan sebuah masukan I sehingga

$$\forall t \in T \, (p(R_t(t(P_w), S(t(P_w), I)))) = p(w)$$

Dari definisi di atas, dapat dibedakan beberapa sub kelas dari pengenal (recognizer) :

- $\mathbf{R_{\emptyset}}$ (P_w,$S(P_w,I)$) adalah sebuah *trivial recognizer* yang tidak menjamin dapat mengenali w jika sbuah transformasi dilakukan pada P_W.
- $\mathbf{R_{Tsem}}$ (P_w,$S(P_w,I)$) adalah sebuah *strong recognizer* yang memiliki daya kenyal terhadap transformasi *semantic preserving*
- $\mathbf{R_T}$(P_w,$S(P_w,I)$) adalah sebuah *ideal recognizer* yang kenyal terhadap segala jenis transformasi
- **R**(P_w, Ø) adalah sebuah *static recognizer* yang hanya dapat memeriksa teks dari P_w bukan state eksekusinya
- **R**(Ø,$S(P_w,I)$) adalah pure *dynamic recognizer* yang hanya dapat meemeriksa state eksekusi dari Pw bukan teks yang dimiliki nya.

6.1 *Resilient* atau Kekenyalan dari *Watermark*

Secara umum, kita lebih tertarik melakukan evaluasi kekuatan dari sebuah *watermark* yang ditulis secara dinamis di state dari program atau

dynamic watermarking (akan dibahas pada bab 7). *Watermark* bertipe tersebut dapat diserang oleh penyerang dengan menulis informasi lain ke dalam state, Jika *watermark* hanya dapat dilenyapkan oleh serangan yang meningkatkan ukuran dari state dinamis milik program dengan faktor r maka dapat kita katakan *watermark* tersebut adalah r-space *resilient* atau kekenyalan r-space.

Definisi 5 : *Watermark* dengan r-space *resilient*

Sebuah *watermark* w dalam program P_w adalah r-spcace *resilient* melalui sehimpunan transformasi $T \subset T$ jika terdapat recognizer R_T dan masukan I sehingga

$$\forall t \in T: (p(R_T(t(P_w), S(t(P_w), I))) \neq p(w)) \rightarrow$$

$$\frac{|S(t(Pw), I))|}{|S(Pw, I))|} \geq r$$

Sebuah 1-space *resilient* adalah *watermark* yang memiliki pengenal strong recognizer. Untuk r>1 maka r akan menjadi parameter yang akan digunakan untuk mengukur kelemahan dari sistem *watermarking*.

Untuk *static watermarking*, beberapa *watermark* yang ditulis di dalam teks program atau data statis adalah rentan terhadap serangan yang manambahkan ukuran statis dari kode

Definisi 6 : *Watermark* dengan r-size *Watermark resilient*

Sebuah *watermark* w dalam Program P_w adalah r-size *resilient* melalui sehimpunan transformasi $T \subset T$ jika terdapat recognizer R_T dan masukan I sehingga

$$\forall t \in T: (p(R_T(t(P_w), S(t(P_w), I))) \neq p(w)) \rightarrow$$

$$\frac{|t(Pw)|}{|Pw|} \geq r$$

Dan juga, banyak serangan yang dilakukan pada program yang terdapat *watermark* P*w* akan meningkatkan waktu runtime.

Definisi 7 : *Watermark* dengan Daya Kenyal r-runtime

Sebuah *watermark* w dalam Program P_w adalah r-runtime *resilient* melalui sehimpunan transformasi $T \subset T$ jika terdapat recognizer R_T dan masukan I sehingga

$$\forall t \in T: (p(R_T(t(P_w), S(t(P_w), I))) \neq p(w)) \rightarrow$$

$$\exists i \in dom(P) \frac{|\text{Time}(t(Pw), i)|}{|\text{Time}(Pw)|} \geq r$$

6.2 *Stealthy* atau Ketidakkentaraan dari *Watermark*

Beberapa tipe dari *watermark* rentan terhadap serangan dengan menggunakan analisa statistik. Jika pencampuran instruksi *static* atau dinamic dari P_w secara radikal berbeda dari yang dapat diharapkan dari program setipe dengan Pw, kita dapat menyangka *watermark* telah disembunyikan di instruksi yang terjadi lebih sering.

Definisi 8 : *Stealthy* dari *Watermark*

Watermark w adalah secara statistik tersembunyi untuk program P melalui pengukuran statistik M jika $M(P) - M(P_W)$ adalah tidak signifikan.

Watermark w adalah secara dinamis tersembunyi jika $M(S(P,I)) - M(S(P_W,I))$ adalah tidak signifikan.

6.3 Data rate dari *Watermark*

Hal yang perlu diperhatikan, data yang ditambahkan dalam *watermark* seharusnya mampu menyimpan informasi sebanyak mungkin tanpa menambah ukuran dari teks program atau working set dari program yang sedang dieksekusi.

Sebagai catatan, data rate adalah banyaknya bit tersembunyi per ekstra kata yang ditambahkan.

Definisi 9 : Efisiensi pengkodean *Watermark*

$H(w) = \log_2 \mathbf{W}$ adalah entropi dari w dalam bit jika w diambil secara seragam probabilitasnya dari **W**.

Jika |P| P∈ P adalah ukuran dari P dalam kata, dan |S(P)| = maxI I∈ dom(P) $S(P,I)$ | adalah menjadi batas atas paling kecil untuk ukuran internal state P.

Penyisipan w pada program Pw memiliki data rate statistik tinggi jika

$$\frac{H(W)}{\max(1,|Pw|-|P|)} \geq 1.$$

Penyisipan w pada program Pw memiliki data rate dinamik tinggi jika

$$\frac{H(W)}{\max(1,|S(Pw)|-|S(P)|)} \geq 1.$$

7 Teknik *Software watermarking*

Teknik-teknik dalam *Software watermarking* dapat dibagi menjadi dua jenis yaitu *static watermarking* dan *dynamic watermarking* berdasarkan bagaimana tiap-tiap teknik menyimpan *watermark.* Namun sebelumnya, teknik tersebut dapat dibedakan lagi menjadi *informed watermarking* dan *blind watermarking.*

7.1.1 *Informed Watermarking*

Pada *watermark* yang bersifat informed, pihak yang melakukan ekstraksi juga harus memiliki akses kepada program yang belum mendapatkan sisipan *watermark*. Ekstraksi dilakukan dengan membandingkan kedua program yang didapat untuk menemukan *watermark* yang telah disisipkan. Hal ini lebih sulit dilakukan karena untuk mengirimkan juga program asli yang belum mendapatkan sisipan *watermark*, butuh *resource* yang lebih. Sebagai alternatif, digunakan *blind watermark*

7.1.2 *Blind Watermarking*

Di dalam alogoritma *watermark*ing yang bersifat blind *watermark*, pihak yang akan melakukan ekstraksi hanya program yang telah disisipi *watermark*, dan hanya masukan key sebagai input. Blind *Watermark*ing tampak lebih beralasan sehingga lebih sering digunakan.

7.2 Static *Watermark*ing

Pada *static watermarking*, *watermark* disimpan secara langsung pada data ataupun code section aplikasi atau kelas asli itu sendiri. Misalnya pada Unix, *watermark* disimpan dalam aplikasi pada initialized *data section* (dimana data string yang bersifat *static* disimpan), *text section* (di lokasi kode yang akan dieksekusi), ataupun *symbol section* (tempat disimpannya informasi debugging).

Yang dapat dikategorikan sebagai dasar dan akan dibahas disini adalah adalah *watermark* pada data, *watermark* yang disimpan dalam data section yaitu seperti *header, string section, informasi debugging* dan lain-lainnya, dan yang kedua adalah *watermark* pada kode yaitu *watermark* yang disimpan pada *section* dari sebuah *executable* yang menyimpan instruksi-instruksi.

7.2.1 *Watermark* Data Statis

```
cons c = "Copyright .. "
```

Gambar 6 *Watermark* Data Statis

Watermark pada data adalah sangat umum karena mudah dibuat dan dikenali. Data ini disimpan berupa konstanta string yang bersifat statik dan data-data statis lainya. Nantinya data tersebut akan disimpan secara langsung di dalam program yang dapat dieksekusi, di dalam data section dari program.

Beberapa cara yang cukup unik yang dapat dimasukkan pada *watermark* data statis ini antara lain dengan memasukkan potongan kode data media yanng dimasukkan sebagai data statis. Hal ini diajukan oleh Mowkowitz dan Cooperman [2]. Potongan kode yang harus cukup esensial tersebut disisipkan secara steganografikal pada data media. Data media itu sendiri dimasukkan sebagai variable statis dalam program.

```
cons picture p =
```

```
Code R = decode (p);
execute(R);
```

Gambar 7 Mowkowitz dan Cooperman

Dengan adanya data statis yang menyimpan potongan program, maka ketika terjadi transformasi program yang bertujuan merusak *watermark* tersebut, maka otomatis potongan program tersebut juga tidak dapat diekstraksi dan dieksekusi, sehingga secara keseluruhan program tidak dapat digunakan.

Namun sayangnya *watermark* pada data statis ini secara umum adalah sangat rentan dalam menerima serangan termasuk transformasi prorgram. Serangan untuk membingungkan (*obfuscation*) dapat dengan mudah merusak sistem *watermark* pada data ini. Penyerang dapat merubah string yang disimpan dalam aplikasi untuk menjadi substring-substring yang disebar di seluruh aplikasi, sehingga proses ekstraksi *watermark* yang disisipkan menjadi tidak mungkin untuk dilakukan.

Serangan lain yang lebih canggih adalah mengubah data statis enjadi program yang dapat menghasilkan data.

7.2.2 *Watermark* Kode

Pada media, *watermark* umumnya disisipkan berupa bit yang redundan dimana penyisipan tersebut tidak dapat dikenali oleh indera karena keterbatasan dari persepsi manusia. Pembangunan *watermark* kode dapat dibangun dengan cara yang hampir mirip. Jika tidak ada ketergantungan data atau kontrol antara dua statement yang berurutan S1 dan S2, mereka dapat dibalik urutannya. Bit dari *watermark* dapat dikodekan dalam apakah S1 dan S2 adalah telah terurut secara leksikografik atau tidak.

Dengan membangun urutan dari cabang sebuah *m*-cabang case statement, kita dapan mengkodekan $\log_2(m!) \approx \log_2(\sqrt{2\pi m}\,(m/e)^m) = O(m \log m)$ panjang bit untuk membentuk *watermark*.

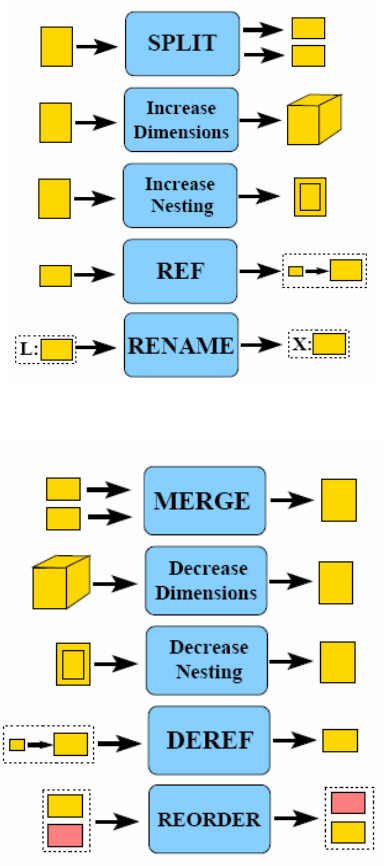
```
char v;
switch e {
      case 1 : v = 'c';
      case 5 : v = 'o';
      case 6 : v = 'p';
      case 8 : v = 'y';
      case 9 : v = 'r';
      . . .
}
```

Gambar 8 Contoh *Watermark* Kode

7.2.3 Serangan dengan Obfuscating Transformation

Transformasi adalah mengubah program yang menjadi inang dari *watermark* yang telah disisipkan. Beberapa serangan transformasi yang sering dilakukan adalah kompilasi-dekompilasi, code optimization, dead-code removal, *obfuscating* dan lain-ain. Transformasi termasuk salah satu serangan distortive attack yang bertujuan agar, dengan adanya transformasi, *watermark* W yang disisipkan didalamnya dapat ikut mengalami transformasi menjadi W'. Hal ini akan mempersulit proses pengenalan *watermark* dalam proses ekstraksi.

Pada serangan obfuscating atau serangkain transformasi yang bersifat mengaburkan, bentuk-bentuk Transformasi yang dapat dilakukan antara lain memecah, menambah ukuran atau dimensi, menambah nesting atau persarangan, menambahkan referensi, menggati nama, atau juga sebaliknya dengan menggabung, mengurangi dimensi, mengurangi persarangan, menarik referensi, termasuk juga mengganti urutan statement. Jadi cara-cara yang digunakan untuk memperoleh dalam mengkodekan *watermark* dalam program dapat digunakan kembali untuk menyerang dan mengganggu karena pada dasarnya perubahan sedikit saja pada data statis akan mengakibatkan kesulitan dalam melakukan ekstraksi *watermark*.



Gambar 9 Serangan dengan Obfuscating Transformation

Distortive attack ini cukup mudah dilakukan dibanding dengan serangan-serangan lain yang memerlukan modal awal sebelum melakukan serangan seperti pengetahuan tentang lokasi *watermark* pada *subtractive attack* , membuat *watermark* sendiri pada *additive attack*, atau mencari program dari author yang sama dengan *watermark* yang berbeda untuk dilakukan serangan *colussive attack*.

Teknik *static watermarking* ini sangat populer karena menarik dalam penggunaannya. Namun teknik *static watermarking* adalah sangat rentan dengan serangan-serangan obfuscating di sini. Dengan mudah dan murah, yaitu tanpa memerlukan *cost* yang terlalu besar, penyerang dapat melakukan serangan dan membuat *watermark* yang disipkan menjadi tidak berguna lagi.

Untuk itulah digunakan teknik lain yang diharapkan, walau tidak dapat mencegah adanya serangan, namun akan mempersulit dan membuat *cost* dalam melakukan serangan menjadi jauh lebih sulit. Teknik tersebut adalah *dynamic watermarking*.

7.3 *Dynamic Watermark*ing

Static *Watermark*ing adalah rentan dalam menerima serangan semantic-preserving transformations. Maka dicarilah jalan menghilangkan kerentanan terhadap serangan tersebut.

Dynamic *watermarking* adalah berbeda dengan *static watermarking*. *Watermark* disimpan dalam state eksekusi program daripada disimpan didalam kode program itu sendiri. Ada 3 jenis *dynamic watermarking*, yang akan didekripsikan sebagai object aplikasi hosting O yang akan menerima sekuens masukan $I = i_1 \ldots i_k$. Sekuens masukan ini akan membuat aplikasi memasuki state yang akan merepresentasikan *watermark*.

Tipe-tipe dari *dynamic watermarking* ini berbeda pada di bagian mana dari program state *watermark* tersebut, dan bagaimana *watermark* tersebut diekstraksi. Ketiga tipe tersebut antara lain *watermark Easter Egg*, struktur data dinamis, dan trace eksekusi dinamis. Yang paling terkenal dan sering dipakai adalah Ester Egg.

7.3.1 *Watermark Easter Egg*

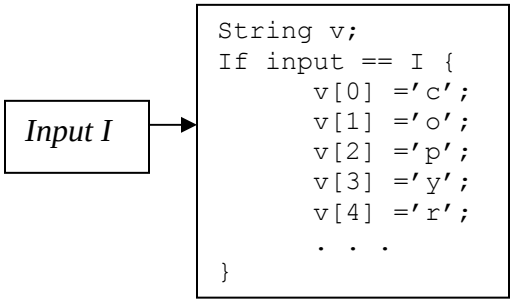
Watermark easter egg adalah *watermark* yang akan diaktivkan hanya ketika aplikasi menerima input yang sangat tidak biasa. Contohnya ketika kita mengetikkan url `about:mozilla` pada aplikasi internet browser Firefox versi 1.5, kita dapatkan kutipan sebagai berikut :

"And so at last the beast fell and the unbelievers rejoiced.But all was not lost, for from the ash rose a great bird.The bird gazed down upon the unbelievers and cast fire and thunder upon them. For the beast had been reborn with its strength renewed, and the followers of Mammon cowered in horror.
from The Book of Mozilla, 7:15 "

Masalah utama dari Easter Egg adalah *watermark* yang digunakan akan mudah untuk ditemukan. Akan sangat sulit untuk membedakan apakah mereka benar dimaksudkan sebagai *watermark* dan bukan karena bisa saja terjadi

sebagai konsekuensi adanya *bugs* atau dari programmer, yang akan mudah ditentukan ketika *watermark* telah ditemukan. Dengan adanya input yang benar untuk memunculkan *Easter Egg*, teknik debugging standard akan memudahkan penyerang untuk melacak keberadaan *watermark* di dalam file aplikasi, dan kemudian dengan mudah dihilangkan atau dinonaktifkan sepenuhnya oleh penyerang.

7.3.2 *Watermark* Struktur Data Dinamis



Gambar 10 *Watermark* Struktur Data Dinamis

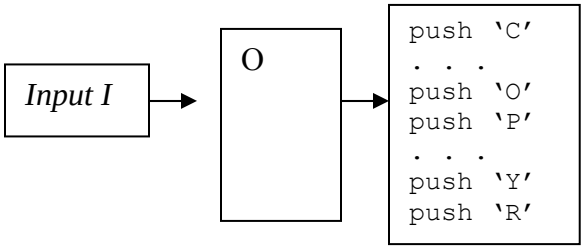
Watermark pada struktur data dinamis ini hampir mirip dengan easter egg, hanya saja, setelah menerima masukan, *watermark* akan disimpan pada struktur data yang dinamis. Proses ekstraksi dilakukan dengan memeriksa nilai dari variable milik hosting O saat itu juga setelah selesai menerima suatu urutan masukan. Hal ini bisa dilakukan dengan routine ekstraksi *watermark* yang telah dihubungkan dengan program eksekusi atau dengan menjalankan program dalam sebuah proses debug.

Watermark struktur data dinamis ini memiliki beberapa keunggulan. *Watermark* struktur data dinamis ini tidak menghasilkan output, sehingga penyerang tidak akan sadar terdapat *watermark* walaupun sekuens input khusus telah dimasukkan. Inilah yang membedakan dari *Easter Egg*, walaupun pada skema tertentu *Watermark* struktur data dinamis dapat menggenerate sekuens input secara acak dan menunggu dihasilkannya output yang tidak diharapkan.

Routine untuk ekstraksi *watermark* juga tidak dimasukkan bersama aplikasi, melainkan akan di-*link* kan dengan program selama kegiatan ekstraksi berlangsung. Hal ini mampu membuat informasi keberadaan *watermark* tidak terlalu banyak tersedia di program eksekusi, sehingga dengan program eksekusi itu sendiri akan sulit untuk menentukan keberadaan *watermark*.

7.3.3 *Watermark Trace* Eksekusi Dinamis

Jika digambarkan, *watermark* trace eksekusi dinamis dapat didekskripsikan sebagai berikut :



Gambar 11 *Watermark* Trace Eksekusi Dinamis

Watermark disisipkan di dalam trace eksekusi (baik instruksi maupun alamat, atau keduanya) dari program yang dijalankan dengan input I. Ekstraksi *watermark* dilakukan dengan memonitor (kemungkinan secara statistikal) beberapa property dari alamat trace dan atau urutan dari operator yang dieksekusi.

8 Teknik –teknik Penyisipan *Watermark*

Beberapa pendekatan yang dapat dilakukan dalam teknik penyisipan *watermark* ini antara lain dengan menggunakan *Naive Approaches*. Cara-cara yang digunakan secara tipikal dalam *Naive Approaches* ini antara lain :

1. Mengurutkan kembali (Basic Block, Statement)
2. Menamakan kembali (Register, Methods)

Pendekatan lebih lanjut yang dapat dilakukan adalah dengan

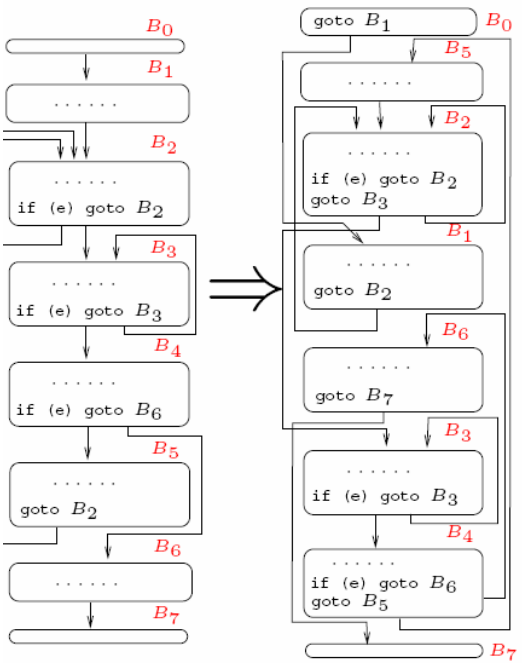
1. Menambahkan semantic pada program
2. Mengubah statistik dari program.

Davidson dan Myhrvold [2] mengajukan pengkodean *watermark* dengan menggunakan pendekatan pengurutan atau reordering blok. di dalam *basic block sequence* <*B5,B2,B1,B6,B3,B4*>. Seperti tampak pada

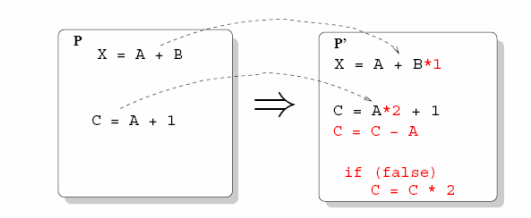
gambar 12. Hasil pengurutan inilah yang akan digunakan untuk mengkodekan *watermark.*

Beberapa teknik lain yang digunakan antara lain menyisipkan *watermark* dengan menambahkan frekuensi dari pola instruksi (Stern) [2], yang menggunakan pendekatan pengubahan statistik program. Seperti tampak pada gambar 10. Hal pertama yang dilakukan adalah mengganti group instruksi dengan penggantinya yang secara semantic adalah ekuivalen. Kemudian menambahkan instruksi yang redundan. Dari penambahan frekuensi inilah yang akan mengubah hasil statistik program, untuk digunakan dalam mengkodekan *watermark*.

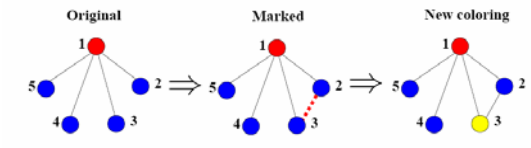
Selain itu masih terdapat cara lain, yaitu dengan menyisipkan mark dengan menambahkan constrain (sisi tambahan) dalam graph register interference graph. Qu Potkonjak [3] menyisipkan *watermark* di dalam register allocation dari program dengan teknik yang dinamakan edge-adding. Teknik ini menggunakan interference graph dan permasalahan pewarnaan graph untuk menyisipkan sebuah *watermark* di dalam register allocation. Seperti ditunjukkan dalam gambar 11.



Gambar 12 Davidson dan Myhrvold



Gambar 13 Stern



Gambar 14 Qu Pontkojak

Gambar 15 berikut menunjukkan pendekatan dengan menambahkan semantik tertentu yang digunakan oleh Cousot [2] dalam mengembed *watermark* didalam hasil dari *static analysis problem*. Seperti kita lihat akan muncul kosntanta konstanta yang tidak wajar, yang hasil perhitungannya akan menjadi *watermark* bagi program yang memuat potongan kode tersebut.

```
int n = ...
int a = 0, b = 1;
for (int i=1;i<n;i++) {
        int c = a+b;
        a = b;
        b = c;
}

// menjadi
-----------------------------
int n = ...
int a = 0, b = 1;
int d = 1, e 35538, f = 1, g = -111353;
e = d* e; d = e+ 11445; g = f*g;
f = g - 47305;
for (int i=1;i< n;i++) {
        int c = a+b; e = d*658;f=f*4;
        a = b;
        f = g + 1566;e = e + 971;
        g = g * f; e = e * d;
        b = c;
        d = e + 4623; f = g + 21494;
}
```

Gambar 15 Cousot dengan Hasil Static Analysis Problem sebagai *Watermark*

9 Aplikasi dari *Software watermarking*

Kita akan melihat penggunaan *software watermarking* dilihat dari berbagai sudut padang antara lain author, distributor dan user.

9.1 *Author*

Hal yang paling umum dalam penggunaan *software watermarking* adalah untuk mengindentifikasi pengarang / pembuat program / author sehingga dapat melindungi kekayaan intelektual yang dimilikinya. Berdasarkan objektif ini pembuat program menyisipkan *watermark* pada objek O untuk mencegah orang lain melakukan klaim sebagai author dari program tersebut. Salah satu variasinya adalah mark dapat diberikan oleh beberapa orang yang memang terlibat sehingga pengenal dapat mengenali seluruh nama yang mencantumkan *watermark*nya di dalam objek O. Mark yang digunakan dalam hal ini didefinisikan sebagai *Authorship Mark (AM)*.

Authorship Mark (AM) adalah *watermark* yang disisipkan di dalam *software* yang meberikan informasi untuk mengindentifikasikan siapa pembuat program tersebut.

Authorship Mark dapat mengindentifikasi seorang author yang dinamakan *Single Authorship Mark*, atau bisa mengindentifikasikan banyak orang yang telah menyisipkan mark ke dalam *software*, yang disebut *Multiple Authorship Mark*, sebagai bukti nama-nama yang diidentifikasikan didalamnya ikut berkontribusi dalam pembuatan *software*.

9.2 Distributor

Watermark yang disisipkan dapat digunakan untuk mengindentifikasi distribusi channel, yang digunakan. Hal ini dapat berguna ketika ditemukan sebuah salinan illegal dari *software*, dengan pengidentifikasian distribusi channel, maka dapat diketahui siapakah yang telah membuat salinan illegal dan mendistribusikannya. Hal ini didefinisikan sebagai *Fingerprint Mark (FM)*.

Fingerprint Mark (FM) adalah sebuah *watermark* yang menyisipkan informasi di dalam perangkat lunak untuk mengidentifikasikan nomor serial dari pembeli *software*.

Jika pada Authorship Mark, *author* menyisipkan *watermark* yang sama pada semua perangkat lunak, *Fingerprint Mark* harus mampu dibedakan berdasarkan skema tertentu untuk tiap-tiap perangkat lunak yang dijual. Hal ini akan bermanfaat untuk melacak jalur distribusi berdasarkan skema yang telah dibuat.

9.3 User

Pada level end user, *watermark* dapat digunakan untuk melakukan identifikasi dua hal : bahwa perangkat lunak yang digunakan oleh user sama sekali tidak mengalami perubahan semenjak penyisipan *watermark*. Kedua, agar author dapat menjamin pengguna tidak melanggar persetujuan lisensi, bahwa benar pengguna tersebut merupakan pengguna yang telah membayar untuk *software* tersebut, bukan pengguna salinan illegal dari perangkat lunak tersebut. Hal-hal tersebut dapat didefinisikan dalam *Validity Mark (VM)* dan *Licesing Mark (LM)*.

Validity Mark (VM) adalah *watermark* yang disisipkan di dalam perangkat lunak untuk melakukan verifikasi bahwa *software* yang digunakan secara esensial tidak mengalami perubahan semenjak diterbitkan

Licensing Mark (LM) adalah *watermark* yang disisipkan dalam perangkat lunak yang informasinya digunakan untuk mengkontrol bagaimana perangkat lunak tersebut digunakan.

10 Mengevaluasi *Software watermarking*

Hal yang umum digunakan untuk membandingkan tingkat kefektifan dari tiap *watermarking* yang berbeda adalah dengan menggunan metrik yang berisi kan tingkat efisiensi, *robustness*, *fidelity* dan *visibility*

10.1 Efisiensi

Ada dua aspek dari efisiensi *watermarking* yang dapat dievaluasi. Pertama adalah biaya komputasi yang termasuk dalam pengembangan, penyisipan, dan pengenalan *mark*. Kedua adalah impact atau pengaruh terhadap *running time* dan penggunaan memori dalam penyisipan *watermark* ke dalanm program.

10.2 *Robustness*

Robustness atau ketangguhan dari sebuah *watermark* adalah tentang seberapa banyak komputasi dan waktu yang dibutuhkan untuk seorang penyerang untuk usaha menghapus *watermark* tanpa merusak *source* asli.

10.3 *Fidelity*

Fidelity dijabarkan sebagai seberapa luas *watermark* telah mengubah isi asli dari perangkat lunak. Seringkali suatu *watermark* harus mengubah struktur dari perangkat lunak. Perhatian utama dari fidelity adalah pada seberapa tingkat usabiltas yang tersisa dari perangkat lunak setelah penyisipan *watermark* dilakukan.

10.4 *Visibility*

Beberapa *watermark* harus mampu memperhatikan apakah *watermark* tersebut dapat dilihat oleh pengguna perangkat lunak atau tidak. Disini erat kaitannya untuk menghindarkan penyerang dari dengan mudah menemukan *watermark* dan mencari cara untuk menghapus atau melakukan serangan selanjutnya pada perangkat lunak.

11 Kesimpulan

Beberapa kesimpulan yang dapat diambil dari studi mengenai *Software watermarking* ini antara lain :

1. Perlindungan karya dalalam bentuk digital adalam menjadi suatu topik masalah sendiri dengan teknik *digital watermarking* sebagai teknik diharapkan mampu memberikan perlindungan lebih pada karya digital.
2. Digital *watermarking* yang diterapkan pada media inang perangkat lunak disebut *software watermarking*. *Software watermarking* adalah proses untuk menyisipkan struktu yang besar ke dalam sebuah program, sehingga struktur tersebut dapat diperoleh kembali walaupun telah program telah mengalami transformasi, penyisipan harus tidak dikenali oleh perusak, dan penyisipan yang dilakukan tidak mengakibatkan penurunan kinerja dari program.
3. Tiga aspek penting yang perlu diperhatikan dalam *software watermarking* adalah *resilient* atau kekenyalan terhadap serangan dan transformasi program, *stealthy* atau ketidakkentaraan *watermark* yang disisipkan dan juga data rate, yaitu besaran data yang bisa disimpan oleh *watermark* yang disisipkan dalam program.
4. Jenis serangan yang dapat dilakukan pada *software watermarking* adalah antara lain *subtractive attact, additive attact, distortive attact*, dan *collusive attact*. Masing-masing bertujuan untuk merusak *watermark* yang telah disisipkan pada sebuah program.
5. Teknik-teknik *software watermarking* antara lain dibedakan menjadi Static *Watermark*ing dan *Dynamic Watermark*ing berdasarkan lokasi dimana *watermark* disimpan
6. *Static Watermark*ing adalah teknik yang menyimpan *watermark* secara langsung pada program atau aplikasi itu sendiri. antara lain berupa *watermark* data statis yang disimpan di data section program dan *watermark* kode yang menyimpan *watermark* bersama dengan penyimpanan instruksi-instruksi dari
7. Dynamic *Watermark*ing menyimpan *watermark* pada state eksekusi dari program. Dynamic *Watermark*ing ini sendiri dibagi menjadi 3 jenis yaitu antara lain *watermark Easter Egg*, Struktur Data Dinamis, dan Trace Eksekusi Dinamis.
8. Penggunaan *Software watermarking* antara lain pada *Authors Mark (AM), Fingerprint Mark (FM)*, *Validation Mark (VM)* dan *Licensing Mark (LM).*
9. Untuk mengukur kinerja dari teknik *watermark*ing yang berbeda , digunakan sebuah metrik yang akan mengukur tingkat efisiensi, *robustness, fidelity*, dan *visibility*

12 Acknowledgement

Makalah ini sebagian besar diambil dari bahan yang dituangkan oleh Collberg [1].

DAFTAR PUSTAKA

[1] Collberg, Christian; Thomborson, Clark . *Software watermarking : Model and Dynamic Embedding*. In Principles of Programming Languages, San Antonio, Januari, 1999.

[2] Collberg, Christian. *Software watermarking Handout*. www://cs.arizona.edu. Department of Computer Science,

University of Arizona. Diakses pada tanggal 25 September 2006.

[3] Collberg, Christian; Myles, Ginger. *Software watermarking Through Register Allocation*. www://cs.arizona.edu. Department of Computer Science, University of Arizona. Diakses pada tanggal 4 Oktober 2006.

[4] Colberg, C, et All. *Dynamic Path Based Watermarking*. Department of Computer Science, University of Arizona. 2002

[5] Munir, Rinaldi. *Bahan Kuliah IF5054 Kriptografi*. Departemen Teknik Informatika, Institut Teknologi Bandung. 2004

[6] Nagra, Jasvir; Collberg, Christian; Thomborson, Clark. *Software watermarking : Protective Terminology.* Austrlian Computer Science Conference. Desember, 2001

[7] Razeen, Muhammad, et All. *Software Protection : The Last Defense Against Piracy.* Al Khawarizmi Institute of Computer Science, Lahor, Pakistan. 2002