University of Arizona

CSc 620

Security Through Obscurity

Christian Collberg

February 5, 2002

Software Watermarking

University of Arizona

CSc 620

Security Through Obscurity

Overview

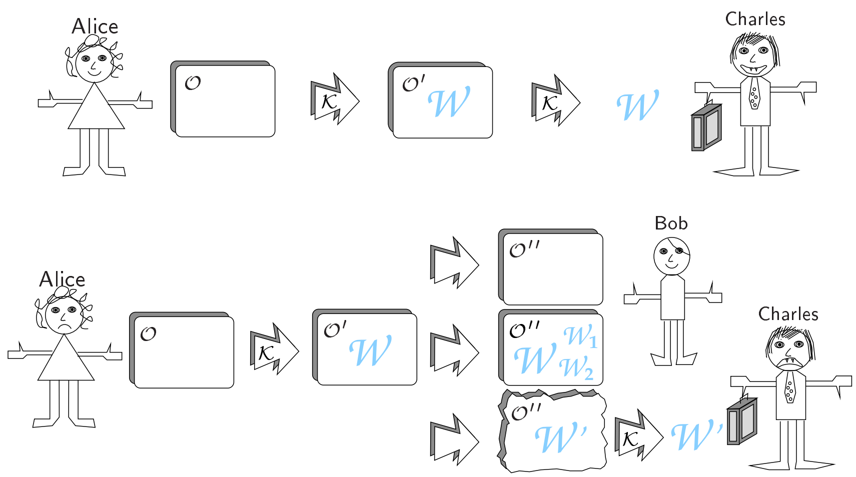
A–Overview

Watermarking & Fingerprinting

Watermark: a secret message embedded into a cover message.

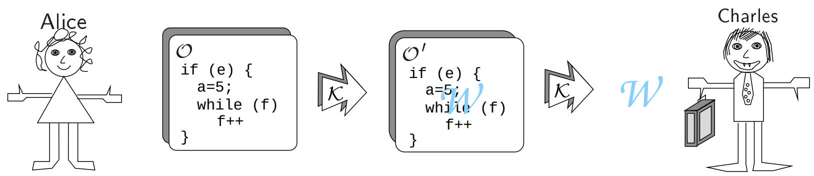
- Image, audio, video, text,...
- Visible or invisible marks.
- Watermarking
  1. discourages theft,
  2. allows us to prove theft.
- Fingerprinting
  3. allows us to trace violators.

Watermarking Overview I



Slide 12A–Overview–5

Software Watermarking



data rate: $\leq$ 1000 bits?

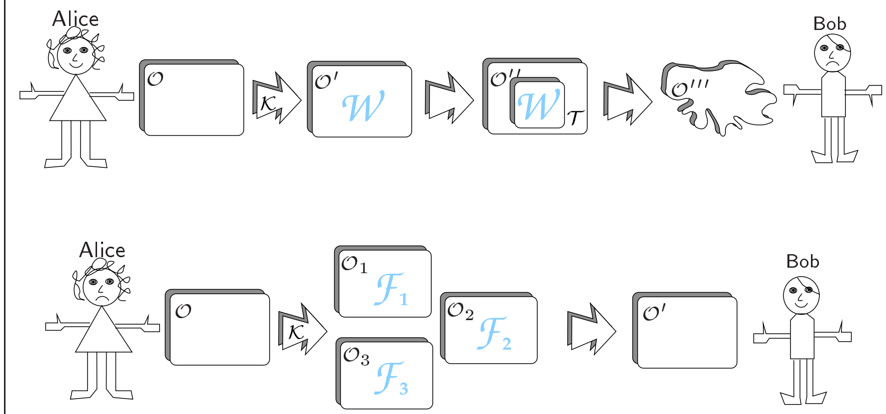
cover program: source code/object code? typed/untyped? architecture-neutral/native binary?

threat-model: semantics-preserving transformations (translation, optimization, obfuscation)?

logistics: generation, distribution, bug-reports?

Slide 12A–Overview–3

Watermarking Overview II



University of Arizona

CSc 620

Security Through Obscurity

Static

B–Static

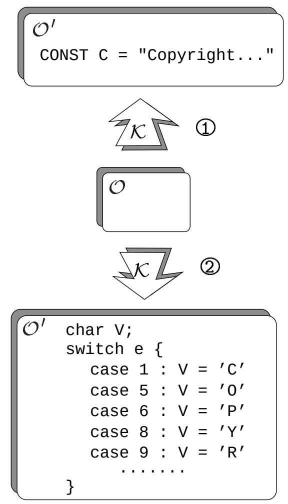
Slide 12A–Overview–4

Attacks on Media Watermarks

- We trade-off between
  1. *stealth* (we want imperceptible marks),
  2. *bit-rate* (we want to embed much data),
  3. *resilience* (we want the mark to withstand attacks).
- Attacks: compression, scaling, cropping, blurring, rotation...

Slide 12B–Static–1

Static Software Watermarks



① **Static Data Watermarks** are embedded in the initialized data (string) section of the program:

```
> strings ···/bin/netscape | \
   grep -i copyright
Copyright (C) 1995, Thomas G. Lane
```

② **Static Code Watermarks** are embedded in the text (code) section of the program.

Slide 12B–Static–2

Static Data Watermarks – DICE Method

```
class Main {
  const Picture C =
```

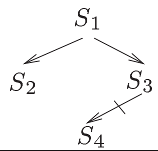

```
    ...
  Code R = Decode(C);
  Execute(R);
}
```

- US Patent 5,745,569, Jan 1996.
- A watermarked media object is embedded in the program's static data segment.
- "Essential" parts of the program are steganographically encoded into the static data.

Slide 12B–Static–3

Static Code Watermarks

Reorder statements S_1 S_2 S_3 S_4 $\Rightarrow$



$\Rightarrow$

S_1	S_1	S_1
S_2	S_3	S_3
S_3	S_2	S_4
S_4	S_4	S_2

Reorder switches

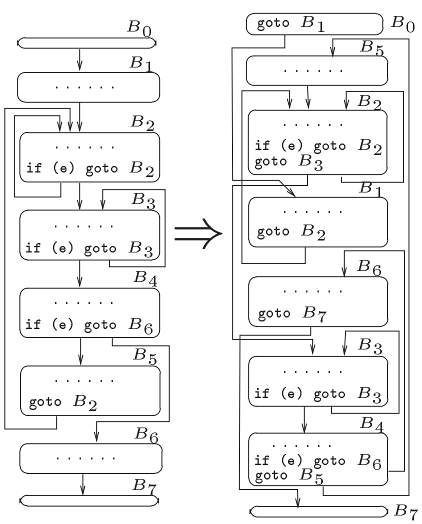
```
case e of {                 case e of {
  case 1 :  S1 |              case 2 :  S2 |
  case 2 :  S2 |     ==>      case 3 :  S3 |
  case 3 :  S3 |              case 1 :  S1 |
}                           }
```

- m-cases $\Rightarrow \mathcal{O}(m \log m)$ watermarking bits.

- Kirowski et.al.: Store watermarks in the register allocation.

Slide 12B–Static–4

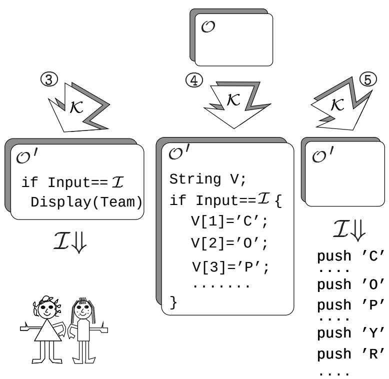
Static Code Watermarks – Microsoft



- Davidson & Myhrvold, US Patent 5,559,884, Microsoft, 1996.
- The watermark is encoded in the basic block sequence $\langle B_5, B_2, B_1, B_6, B_3, B_4 \rangle$.

Slide 12C–Dynamic–1

Dynamic Software Watermarks



The program is run with input $\mathcal{I}{=}\mathcal{I}_1\cdots\mathcal{I}_k$. The watermark is embedded in

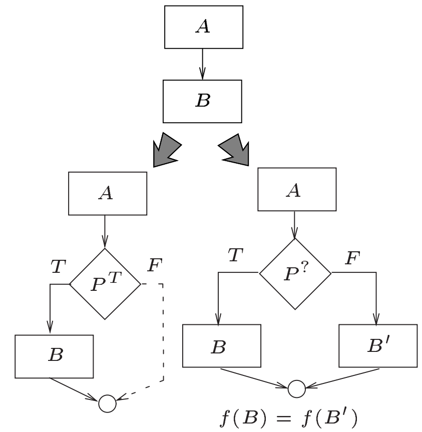
① **Easter Egg**
the unexpected behavior of the program.

② **Data Structure**
a variable V.

③ **Execution Trace**
the program's instruction or address trace.

Slide 12B–Static–5

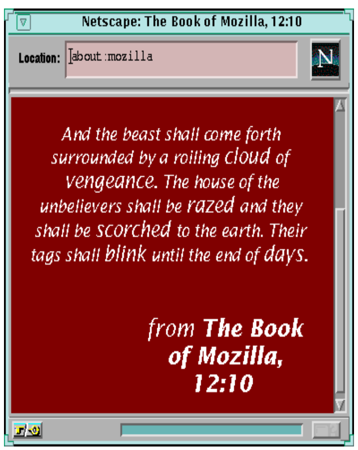
Attacks on Static Watermarks



- Effective threat models:
  1. locality-improving optimizations,
  2. code/data obfuscations,
  3. decompile + recompile,
  4. binary translation,
  5. …

Slide 12C–Dynamic–2

Easter Egg Watermarks



- The watermark performs an action that is immediately perceptible.

  $\Downarrow$

  Extraction is trivial.

- Effects must not be too subtle.
- `www.eeggs.com/lr.html`.

University of Arizona

CSc 620
Security Through Obscurity

Dynamic

C–Dynamic

University of Arizona

CSc 620

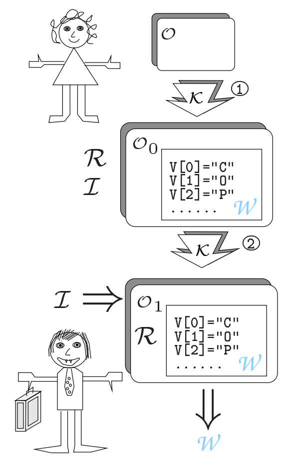
Security Through Obscurity

Graph

D–Graph

Slide 12C–Dynamic–3

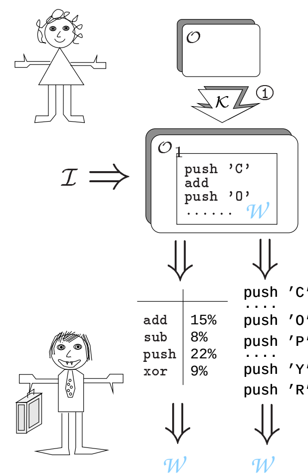
Dynamic Data Structure Watermark



- The watermark is embedded within the state (globals, heap, stack) of the program.
- A recognizer $\mathcal{R}$ extracts the watermark by examining the state after input $\mathcal{I}$.
- No "special" output is produced.
- $\mathcal{R}$ is not shipped.

Slide 12C–Dynamic–4

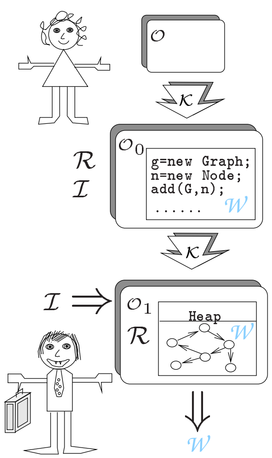
Dynamic Execution Trace Watermark



- The watermark is embedded within the instruction or address trace.
- Watermark extraction:
  - the actual trace,
  - some statistical property of the trace.
- Threat model:
  - optimizations,
  - obfuscations, $\cdots$

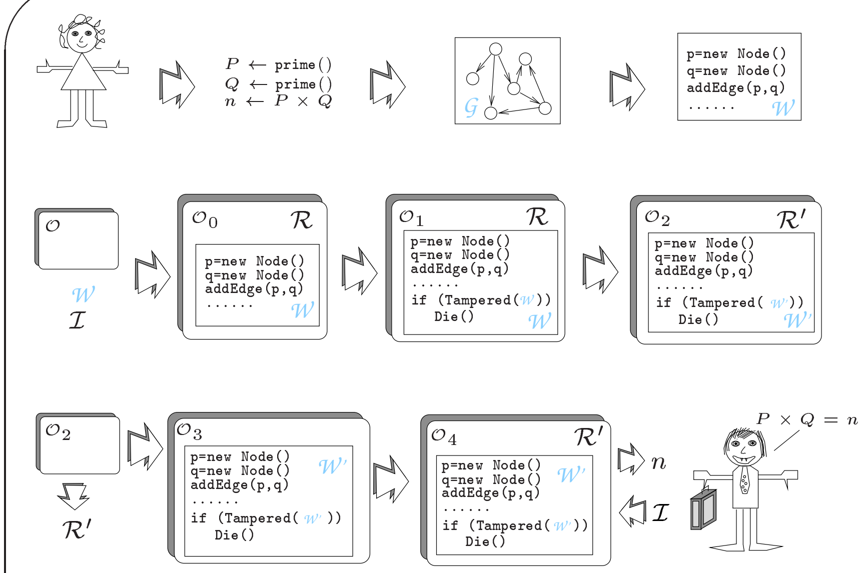
Slide 12D–Graph–1

Dynamic Graph Watermarks



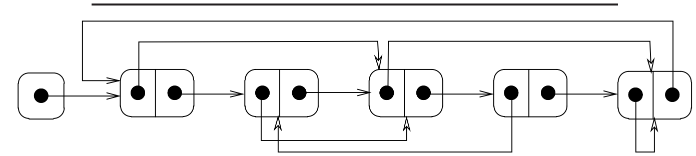
- We embed watermarks in the topology of a graph:
  1. Hard for Bob to analyze (aliasing),
  2. Easier for Alice to tamper-proof.
- A recognizer $\mathcal{R}$ extracts the watermark by examining heap-objects after input $\mathcal{I}$.

Slide 12D–Graph–2



Slide 12D–Graph–3

Graph Embedding: Radix-k



$3 \cdot 6^4 \; + \; 2 \cdot 6^3 \; + \; 3 \cdot 6^2 \; + \; 4 \cdot 6^1 \; + \; 1 \cdot 6^0 = 4453 = 61 * 73$

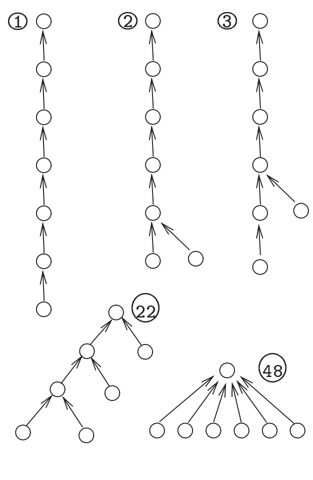
- G is a circular linked list.
- An extra pointer field encodes a base-k digit:

null-pointer	$\Rightarrow$	0
self-pointer	$\Rightarrow$	1
next node pointer	$\Rightarrow$	$2 \cdots$

- A 255-node list hides 2040 bits $\Rightarrow$ bit-rate is 4 bits per word.

Slide 12D–Graph–4

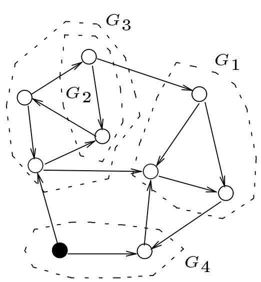
Graph Embedding: Enumeration



- n is represented by the *index* of the the graph G in some enumeration.
- We must, efficiently, be able to
  1. given n, generate the n:th graph,
  2. given G, find Gs index n.
- Oriented parent-pointer trees $\Rightarrow$
  1. 655 nodes $\Rightarrow$ 1024-bit integer,
  2. bit-rate: 1.56 bits per word.

Slide 12D–Graph–5

Graph Recognition

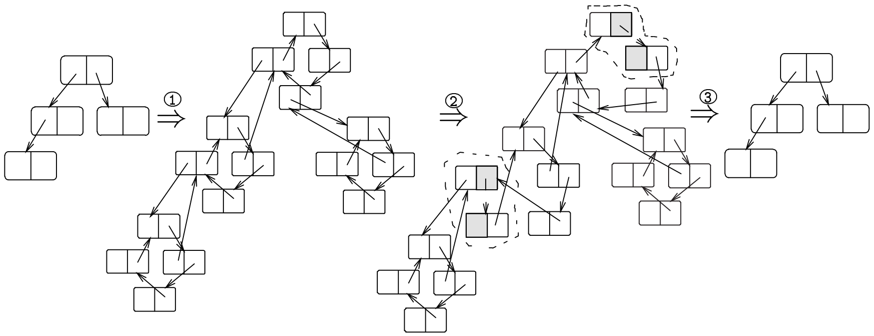


if (input = $\mathcal{I}_1$) $G_1 = \cdots$;
if (input = $\mathcal{I}_2$) $G_2 = \cdots$;
if (input = $\mathcal{I}_3$) $G_3 = G_2 \oplus G_3$;
......
if (input = $\mathcal{I}_k$) $G = G_1 \oplus G_3 \oplus \cdots$;

- For each input $\mathcal{I}_i$, one graph component is built.
- After input $\mathcal{I}_k$ the entire graph has been assembled.
- To find the root of G we need only consider nodes created during the processing of $\mathcal{I}_k$.

Slide 12D–Graph–8

Tamperproofing Against Node-Splitting



① Alice expands each node into a 4-cycle.

② Bob splits two nodes.

③ Alice shrinks the underlying graph's biconnected components.

Slide 12D–Graph–9

Tamperproofing by Reflection

```
class C {public int a; public C car, cdr;}
```

```
Field[] F = C.class.getFields();
if (F.length != 3) die();
if (F[1].getType() != C.class) die();
```

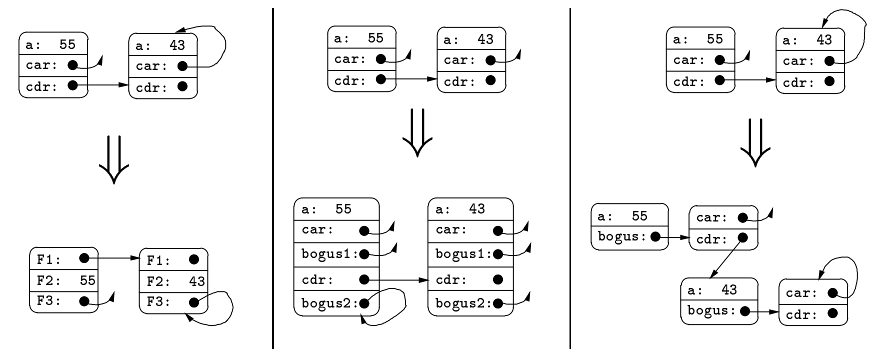
```
{
 O.car=V;
}
```

$\stackrel{\mathcal{T}}{\Longrightarrow}$

```
Field[] F = C.class.getFields();
for(int i=0; i<F.length; i++)
 if (F[i].getType().
          isAssignableFrom(C.class)) {
    F[i].set(O, V); break;
 }
```

Slide 12D–Graph–6

Graph Obfuscation Attacks

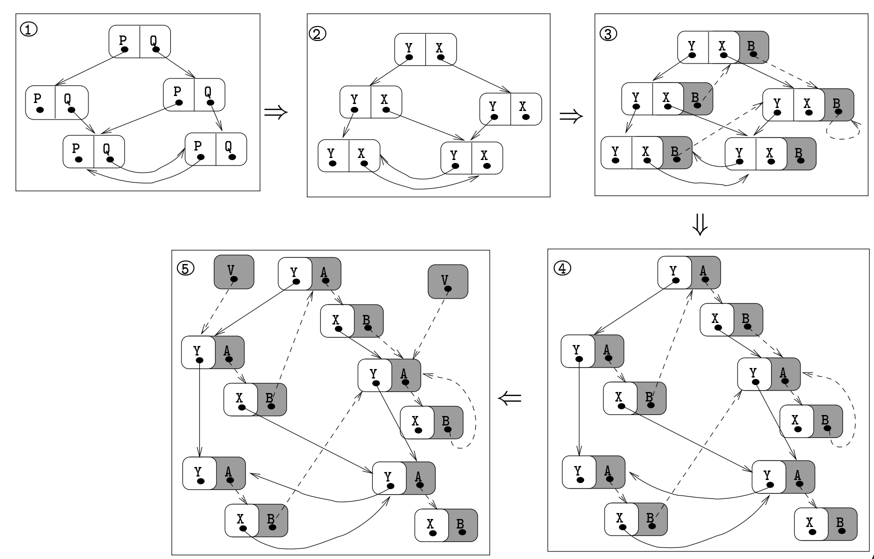


Rename Fields

Add Pointers

Split Nodes

Slide 12D–Graph–7



Slide 12E–Model–1

A Model of Software Watermarking

Definition 1 (Software Watermark) Let $\mathbb{W}$ be a set of mathematical structures, and p a predicate such that $\forall w \in \mathbb{W} : p(w)$. We choose p and $\mathbb{W}$ such that the probability of $p(x)$ for a random $x \notin \mathbb{W}$ is small. $\square$

Slide 12E–Model–2

Software Watermarking: Programs

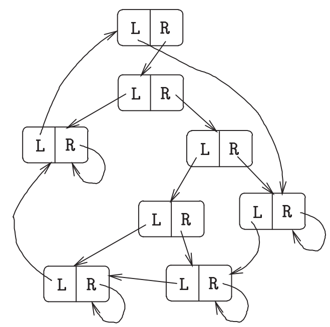
Definition 2 (Programs) Let $\mathbb{P}$ be the set of programs. P_w is an embedding of a watermark $w \in \mathbb{W}$ into $P \in \mathbb{P}$.

Let $\texttt{dom}(P)$ be the set of input sequences accepted by P. Let $\texttt{out}(P, I)$ be the output of P on input I.

Let $S(P, I)$ be the internal state of program P after having processed input I. Let $|S(P, I)|$ be the size of this state, in accessible words. $\square$

Slide 12D–Graph–10

Non-Semantics-Preserving Attacks



- A planted plane cubic tree on 8 nodes.
- Bit-rate is 0.5 bits per word.
- Planarity check:
  - For each internal node x, the left-most child of x's right subtree is L-linked to the right-most child of x's left subtree.

University of Arizona

CSc 620

Security Through Obscurity

Model

E–Model

Software Watermarking: Coding Efficiency

Definition 5 (Watermark Coding Efficiency) $H(w) = \log_2 |\mathbb{W}|$ is the *entropy* of w, in bits, when w is drawn with uniform probability from $\mathbb{W}$.

Let $|P|$, $P \in \mathbb{P}$ be the size (in words) of P as expressed in some encoding.

Let $|S(P)| = \max_{I \in \mathtt{dom}(P)} |S(P,I)|$ be the least upper bound on the size of P.

An embedding of P_w of w in P has a *high static data rate* if $\frac{H(w)}{|P_w|-|P|} \geq 1$. An embedding P_w of w in P has a *high dynamic data rate* if $\frac{H(w)}{|S(P_w)|-|S(P)|} \geq 1$. □

University of Arizona

CSc 620

Security Through Obscurity

Summary

F–Summary

A Model of Watermarking: Transformations

Definition 3 (Program Transformations) Let $\mathbb{T}$ be the set of transformations from programs to programs.

$\mathbb{T}_{\text{sem}} \subset \mathbb{T}$ is the set of *semantics preserving* transformations:

$$\begin{aligned}\mathbb{T}_{\text{sem}} = \{t : \mathbb{T} \quad | \quad & P \in \mathbb{P}, I \in \mathtt{dom}(P),\\ & \mathtt{dom}(P) = \mathtt{dom}(t(P)),\\ & \mathtt{out}(P,I) = \mathtt{out}(t(P),I)\}.\end{aligned}$$

Similarly, $\mathbb{T}_{\text{stat}} \subset \mathbb{T}$ is the set of *state preserving* transformations:

$$\begin{aligned}\mathbb{T}_{\text{stat}} = \{t : \mathbb{T} \quad | \quad & P \in \mathbb{P}, I \in \mathtt{dom}(P),\\ & S(P,I) = S(t(P),I)\}.\end{aligned}$$

□

Software Watermarking: The Recognizer

Definition 4 (Watermark Recognizer) $\mathcal{R}_T(P_w, S(P_w,I))$ is a *recognizer* of $w \in \mathbb{W}$ in $P_w \in \mathbb{P}$ with input I wrt a set of transformations $T \subset \mathbb{T}$, if,

$$\forall t \in T : p(\mathcal{R}(t(P_w), S(P_w,I))) = p(w)$$

□

- $\mathcal{R}_\emptyset(P_w, S(P_w,I))$ is the *trivial* recognizer.
- $\mathcal{R}_T(P_w, \emptyset)$ is a *static* recognizer.
- $\mathcal{R}_T(\emptyset, S(P_w,I))$ is a *pure dynamic* recognizer.
- $\mathcal{R}_{\mathbb{T}_{\text{sem}}}(P_w, S(P_w,I))$ is a *strong* recognizer.

Slide 12F–Summary–1

Dynamic Graph Watermarking – Summary

Embed n in P such that	Threat Resistance:
a) the embedding is imperceptible,	a) Distortive attacks • translation, • optimization, • obfuscation.
b) any successful attack incurs a large performance penalty,	b) Collusive attacks.
c) n is large,	c) Statistical attacks.
d) there is no performance penalty,	d) Cropping attacks.
e) n can be retrieved after any attack on P.	e) Additive attacks.

Slide 12F–Summary–2