Security Through Obscurity

Christian Collberg

University of Arizona

collberg@cs.arizona.edu

Intellectual property attacks on software

Software watermarking

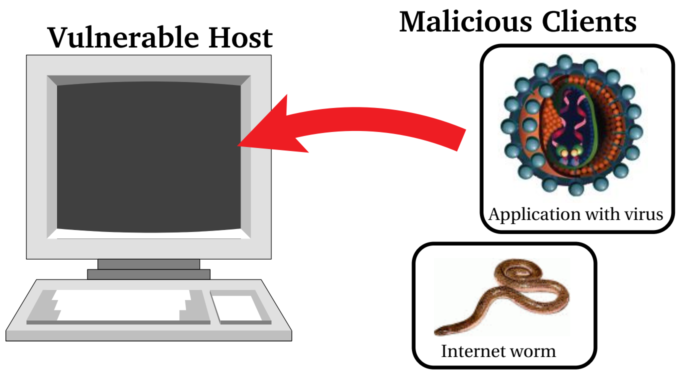
Code obfuscation

Code tamperproofing

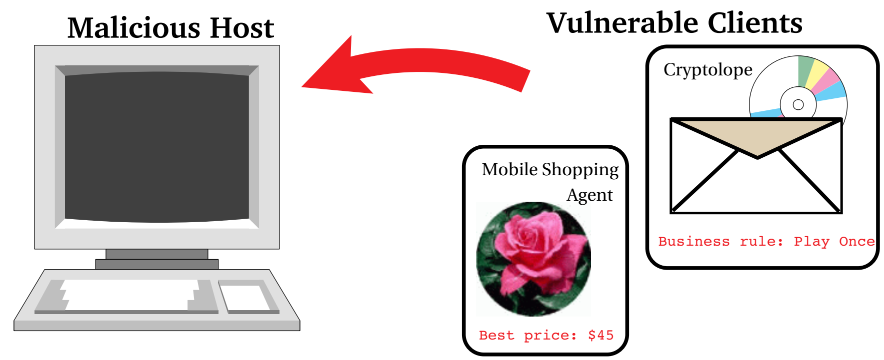
SandMark

Supported by grants from the NSF and the AFRL.

Malicious Clients vs. Malicious Hosts

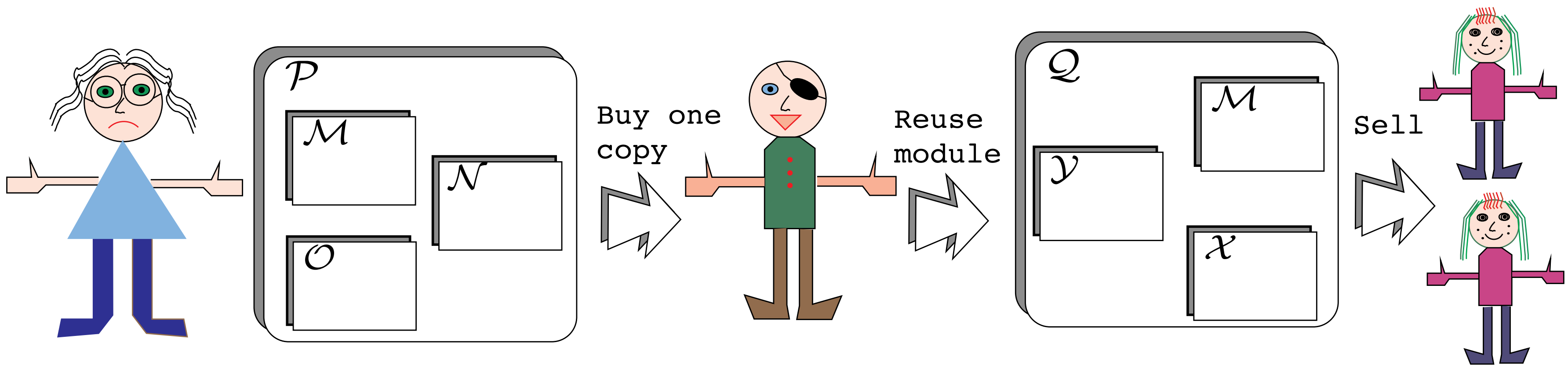


- The client destroys data on the host.
- Focus of most current security research.
- Typical idea: run the client in a sandbox.



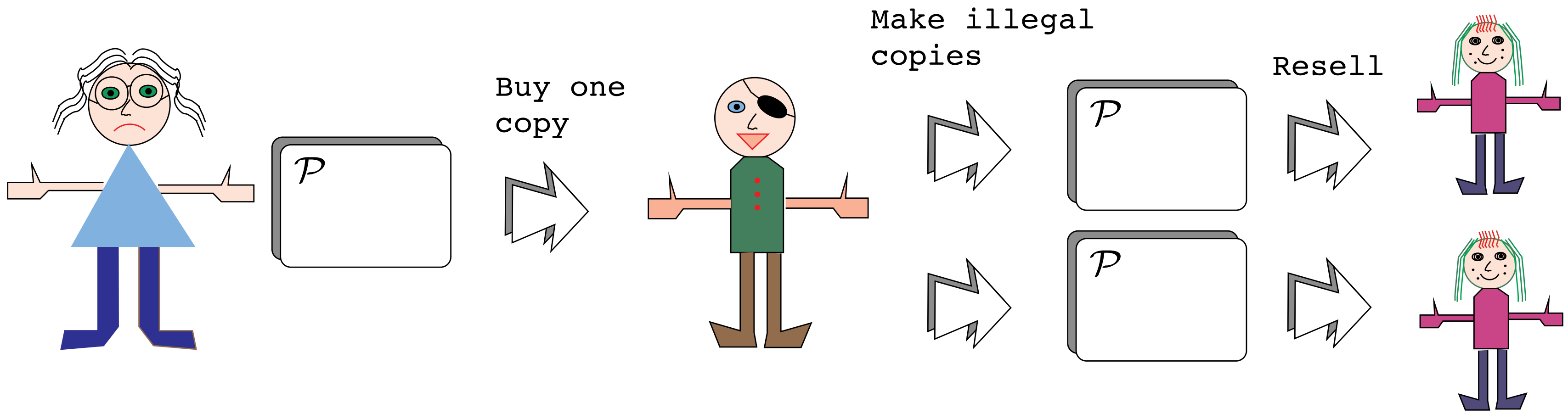
- The client must protect some *Intellectual Property* from the host
- Focus of our research.
- Typical ideas: Obfuscate, watermark, tamperproof the client.

Malicious Reverse Engineering



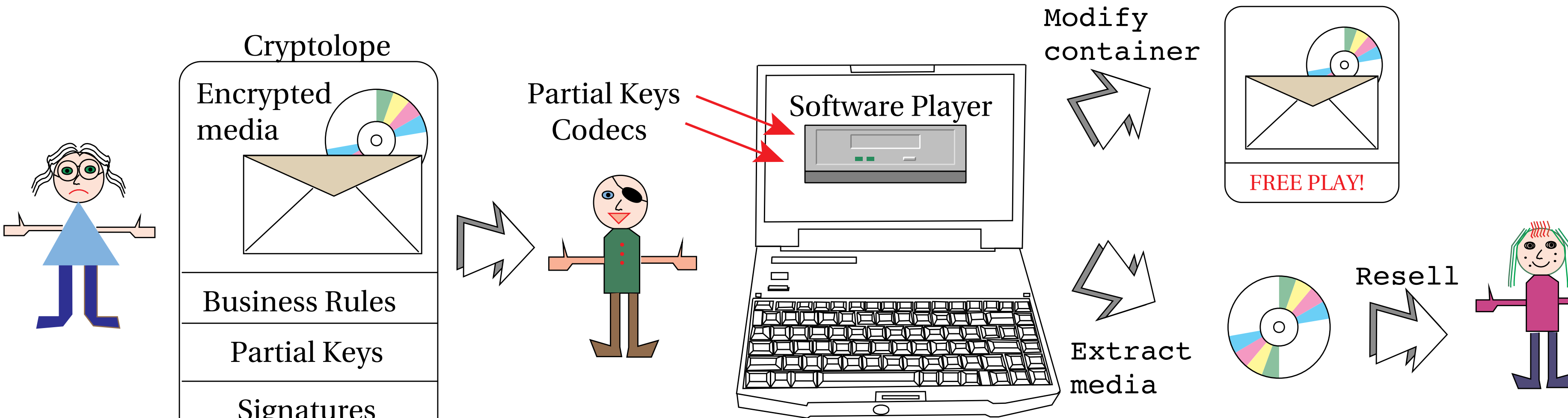
- Alice and Bob are competing software developers.
- Module $\mathcal{M}$ contains Alice's algorithmic trade secrets.
- Bob reverse engineers $\mathcal{M}$ and includes it in his own program.
- Worse with easily decompilable distribution formats such as Java bytecode, .NET, ANDF.

Software Piracy



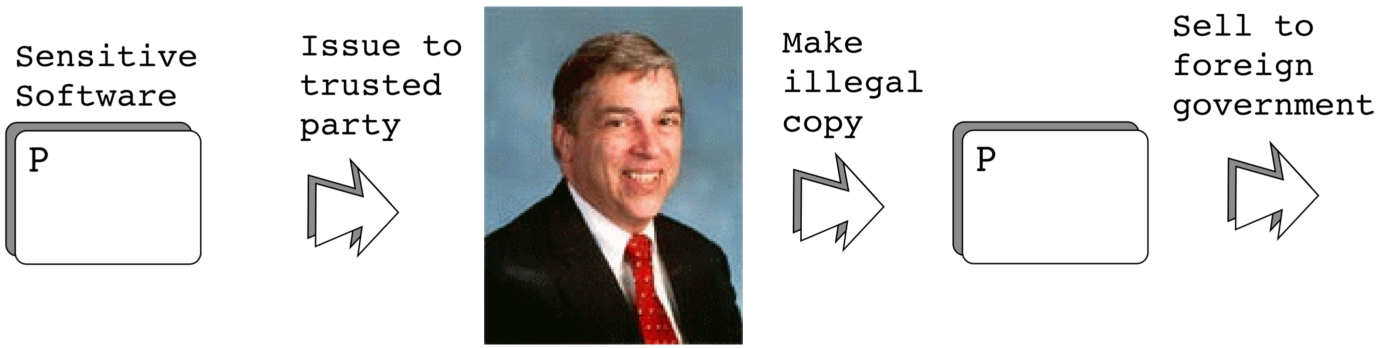
- Alice is a software developer.
- Bob buys one copy of Alice's application.
- Bob makes illegal copies and sells them to a third party.
- Software piracy is a 15 billion-dollar a year industry.

Tampering



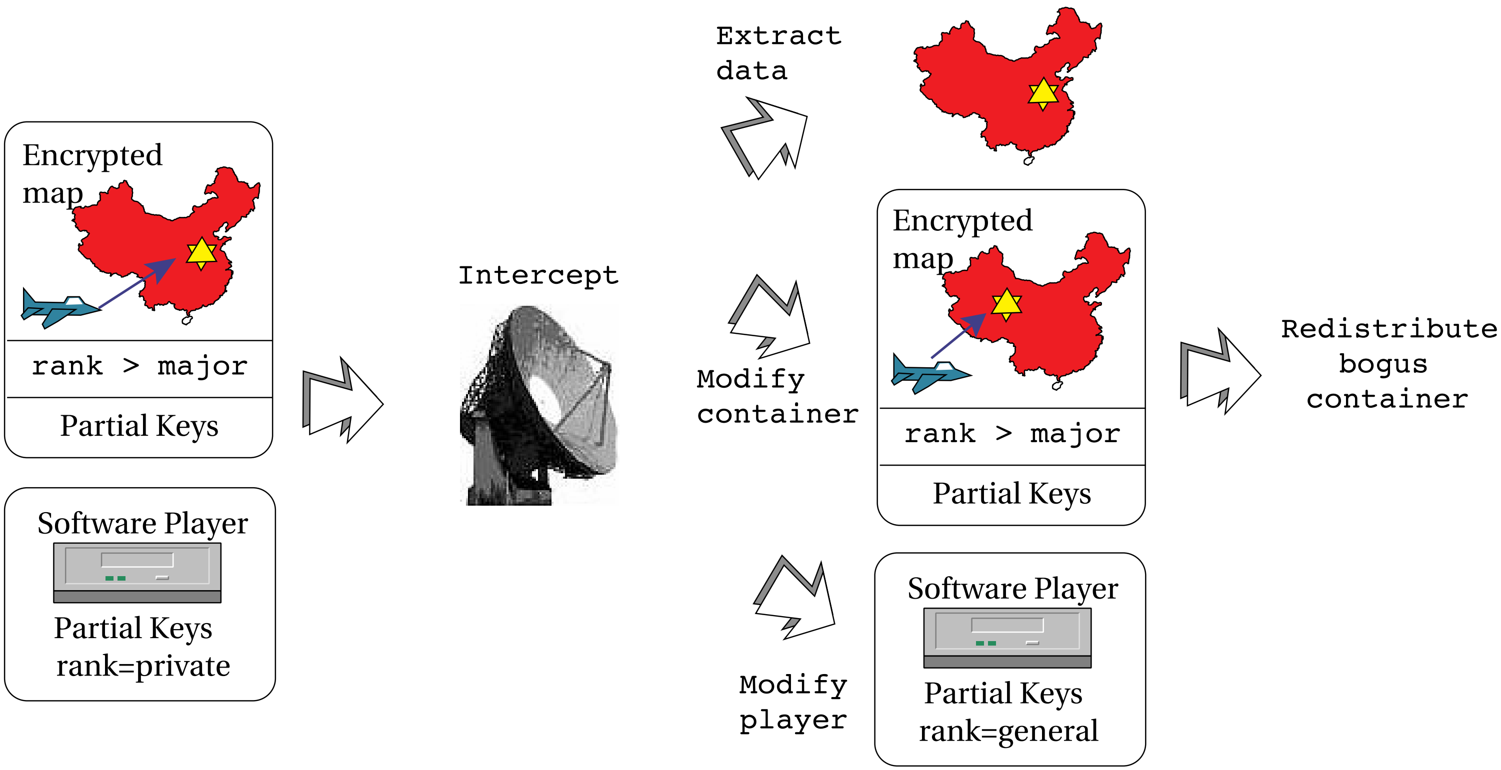
- Alice is a media (images,audio,video) publisher. She packages her media into a *cryptolope*.
- Bob tampers with the software player to extract keys or decrypted media or to tamper with the business rules.
- InterTrust, Intel, IBM, Xerox, Microsoft,. . . .

Military Piracy



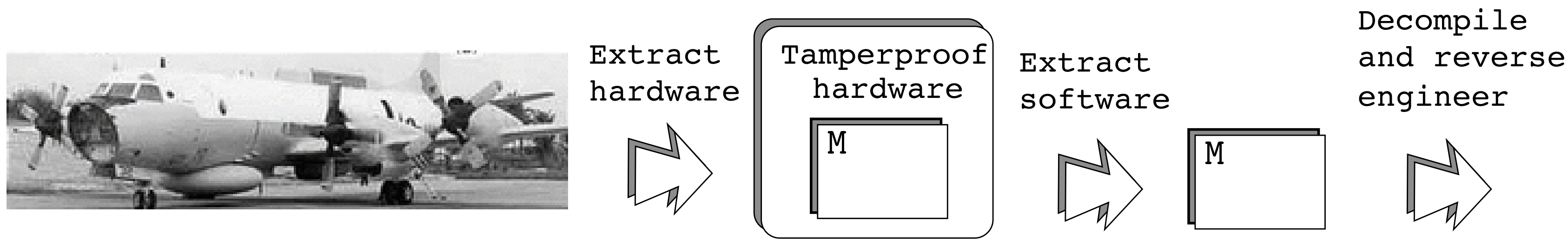
- The military and intelligence communities are also worried about illegal redistribution of software.
- At the very least, they would like to be able to track the whereabouts of classified software.

Military Tampering



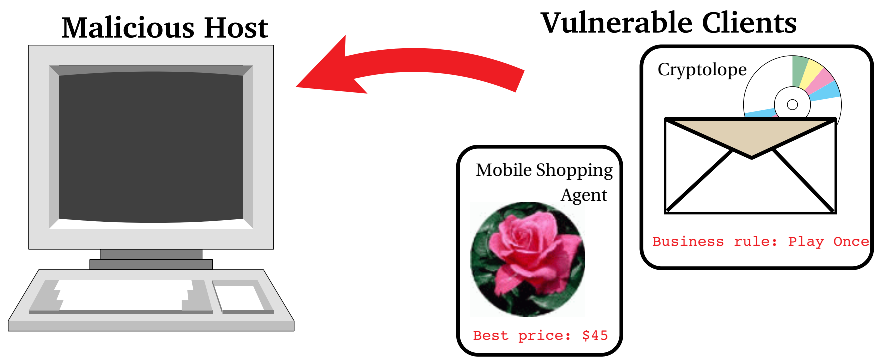
- Cryptolopes can be used for military data.
- To avoid *class attacks*, players (with new keys/privileges) may have to be redistributed in the field.

Military Reverse Engineering



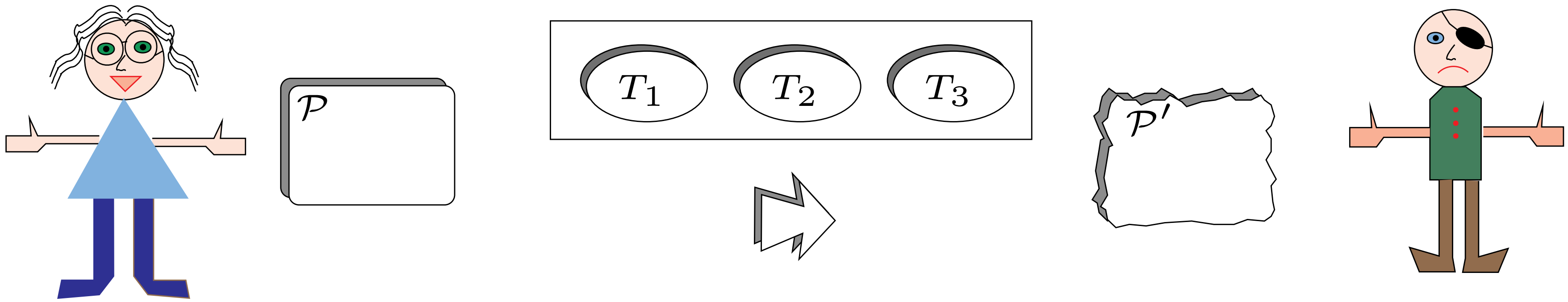
- In 1944, the Soviets recovered three B-29 bombers. 105,000 parts were reverse engineered. The B-29 became the Tu-4 in just two years.
- In 1976, a MiG-25 pilot defected to Japan. The plane was sent back (disassembled) in boxes.
- In 2001, an EP-3 spy/reconnaissance plane landed in China after a collision. The crew was unable to destroy all equipment
- Much AFRL anti-tamper funding in coming years.

Threat Models



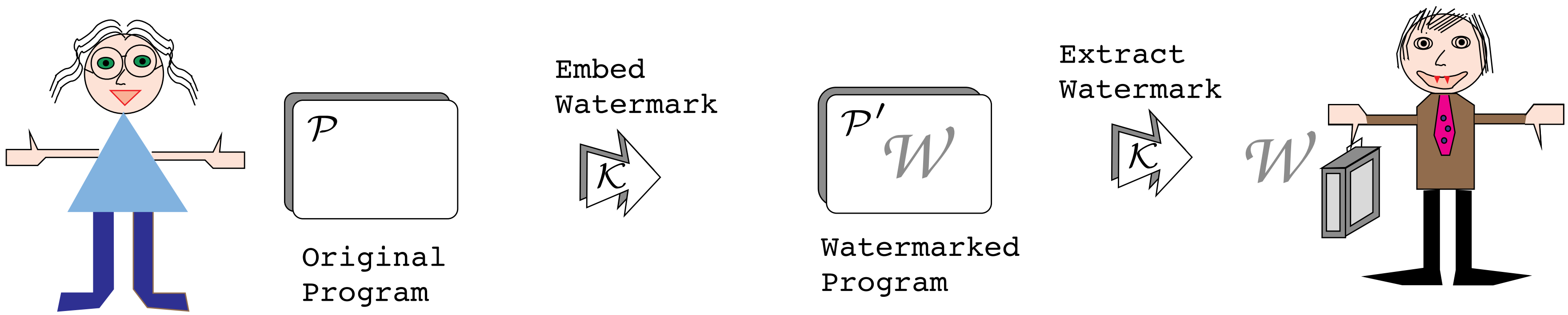
- The malicious host is a
  - human reverse engineer, or a
  - tool that automatically analyses the client, or a
  - human aided by automatic tools.
- The tools could do static analysis , dynamic analysis (debugging, tracing), statistical analysis, . . .

Code Obfuscation



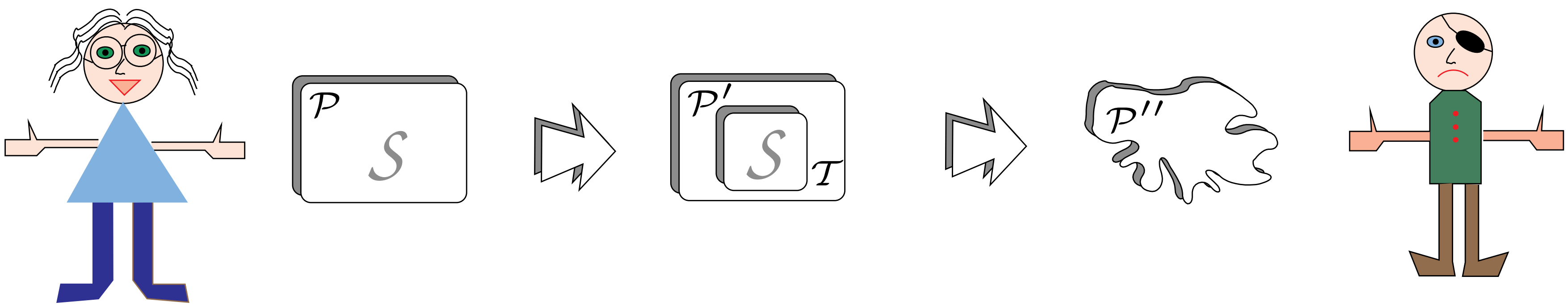
- Code obfuscation is a software-only approach to hamper malicious reverse engineering.
- The idea is to slow down the reverse engineering process by making software harder to understand.
- Complete protection is not expected.

Software Watermarking



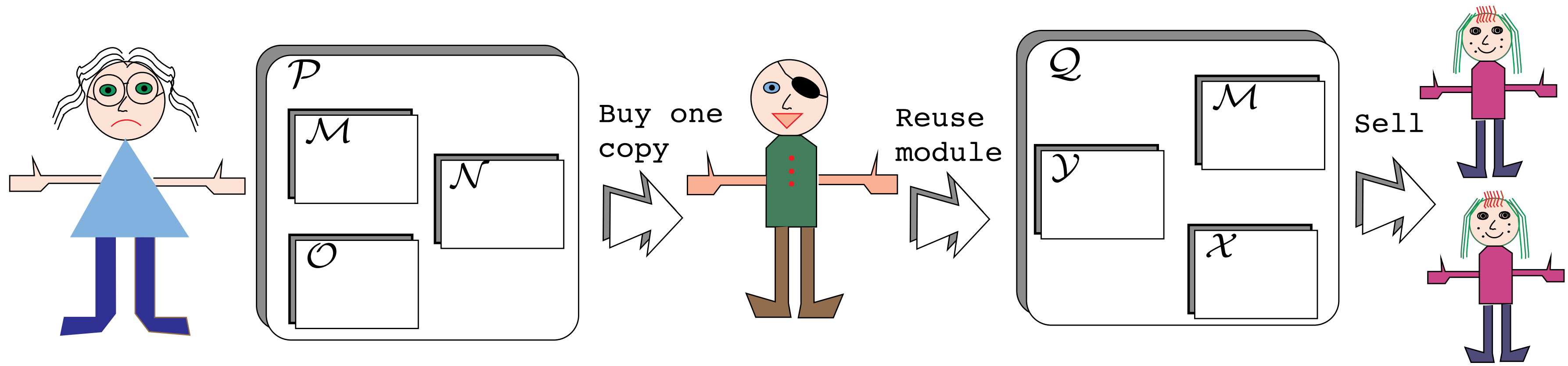
- To assert our IP rights we add an invisible *copyright* notice (a *watermark*) to our code.
- To trace software pirates we add an invisible *fingerprint* (a *customer identification number*) to the code.

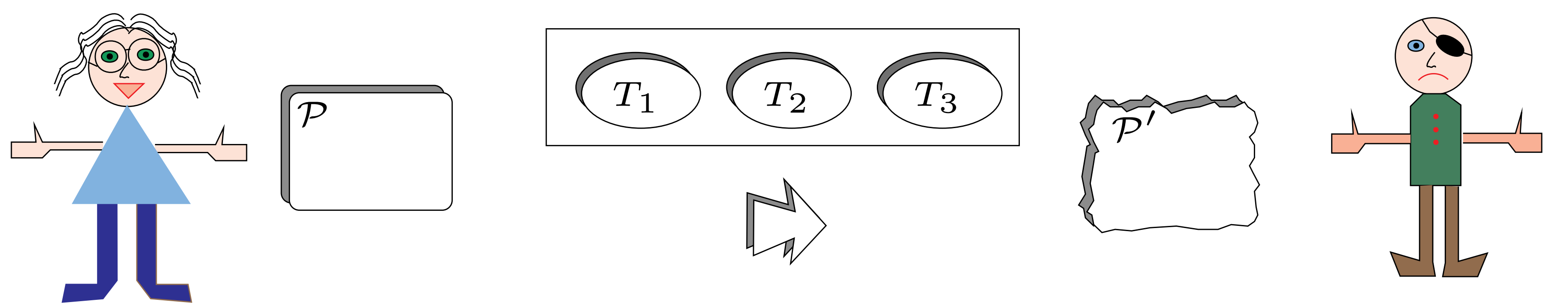
Tamper Proofing



- We add code to our program that
  1. detects if the program has been tampered with, and
  2. either fails or repairs itself .

Code Obfuscation





Obfuscating Transformation

Let $P \xrightarrow{\mathcal{T}} P'$ be a transformation of a source program P into a target program P'. $P \xrightarrow{\mathcal{T}} P'$ is an obfuscating transformation , if P and P' have the same observable behavior .

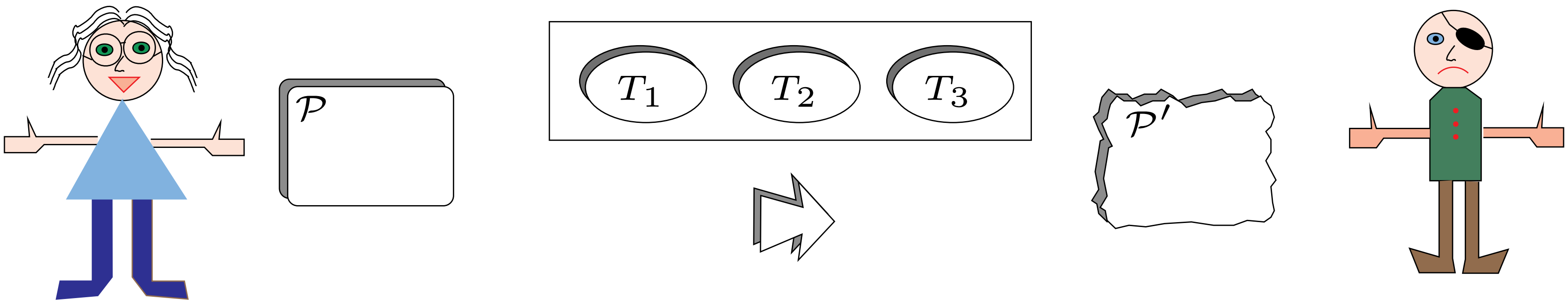
1. If P fails to terminate or terminates with an error condition, then P' may or may not terminate.
2. Otherwise, P' must terminate and produce the same output as P.

- P' may have side-effects that P does not, as long as these side effects are not experienced by the user.
- P and P' don't have to be equally efficient.

Protection By Obfuscation

- The level of security from reverse engineering that an obfuscator adds to an application depends on
  1. the sophistication of the obfuscating transformations,
  2. the power of the deobfuscator,
  3. the amount of resources available to the deobfuscator.
- Ideally, we would like to mimic the situation in cryptography, where there is a dramatic difference in the cost of encryption and decryption.
- There are obfuscating transformations that can be applied in polynomial time but which require worst-case exponential time to deobfuscate.

Principles of Code Obfuscation



Maximize obscurity Understanding $\mathcal{P}'$ is harder than understanding $\mathcal{P}$.

Maximize resilience Automatic de-obfuscation tools are hard to construct or expensive to run.

Maximize stealth $\mathcal{P}$ and $\mathcal{P}'$ have similar statistical properties.

Minimize cost $\mathcal{P}$ and $\mathcal{P}'$ have similar execution times.

Software Metrics

Halstead: $E(P)$ increases with the # of operators+operands in P.

McCabe: $E(P)$ increases with the # of predicates in P.

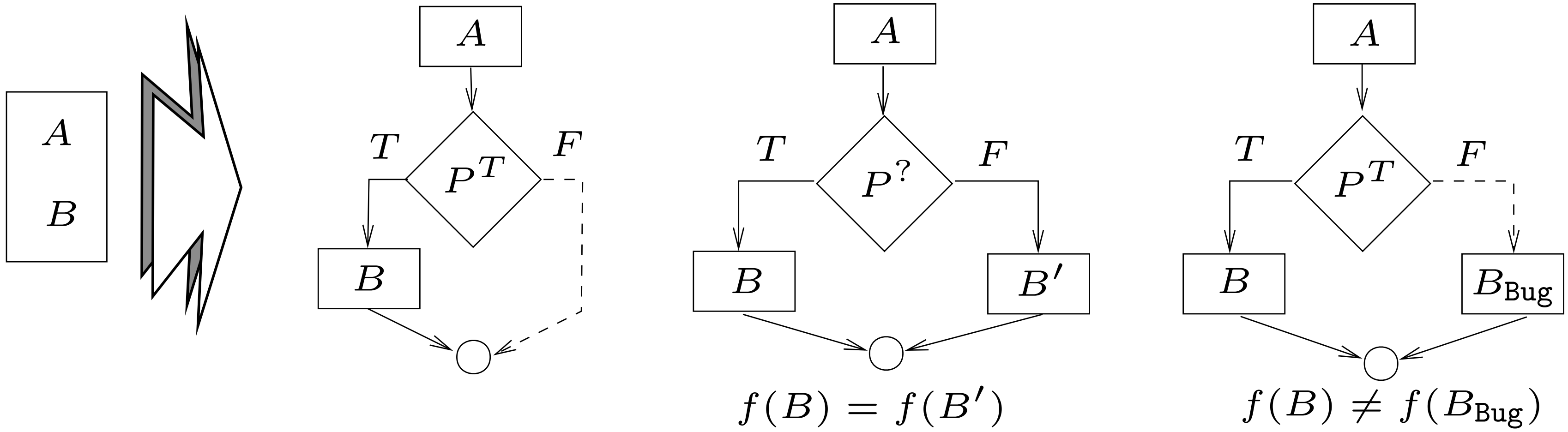
Harrison: $E(P)$ increases with the nesting level of conditionals in P.

Munson: $E(P)$ increases with the complexity of the static data structures (arrays, records) declared in P.

Chidamber: $E(C)$ increases with

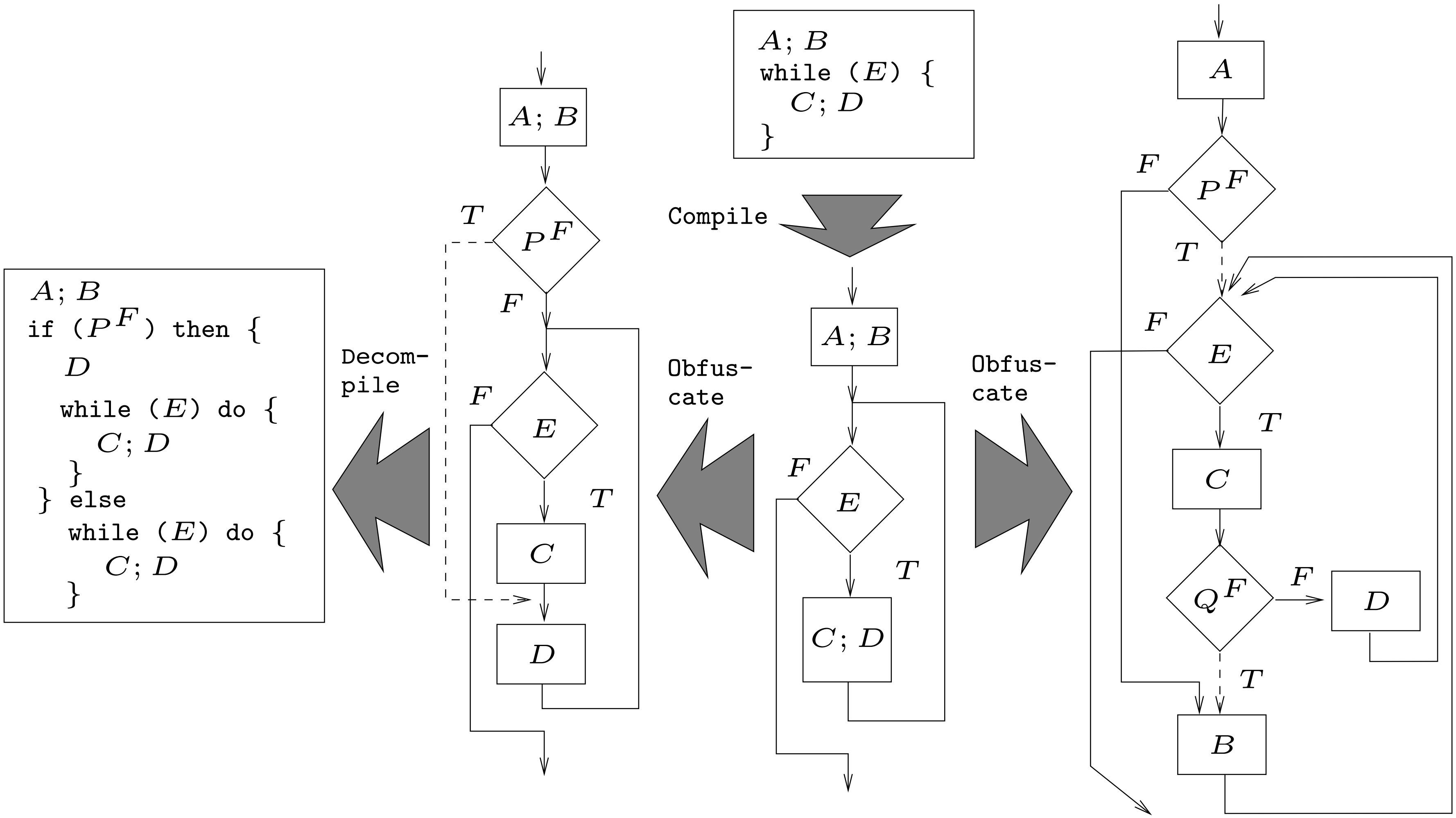
- the number of methods in the class C,
- the depth of C in the inheritance tree,
- the number of direct subclasses of C,
- the number of other classes to which C is coupled,
- the number of methods that can be executed in response to a message sent to an object of C.

Obfuscating Control Transformations



Opaque predicates: $P^T \equiv \texttt{TRUE}$, $P^F \equiv \texttt{FALSE}$, $P^? \equiv \begin{cases} \texttt{TRUE} \\ \texttt{FALSE} \end{cases}$

Irreducible Flow Graphs

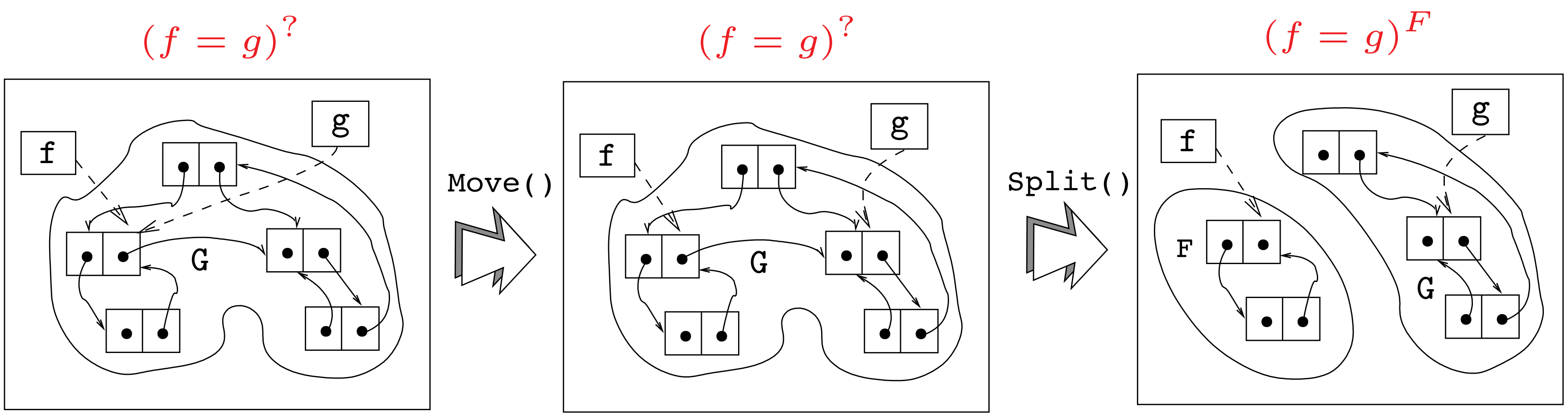


Elementary Opaque Predicates

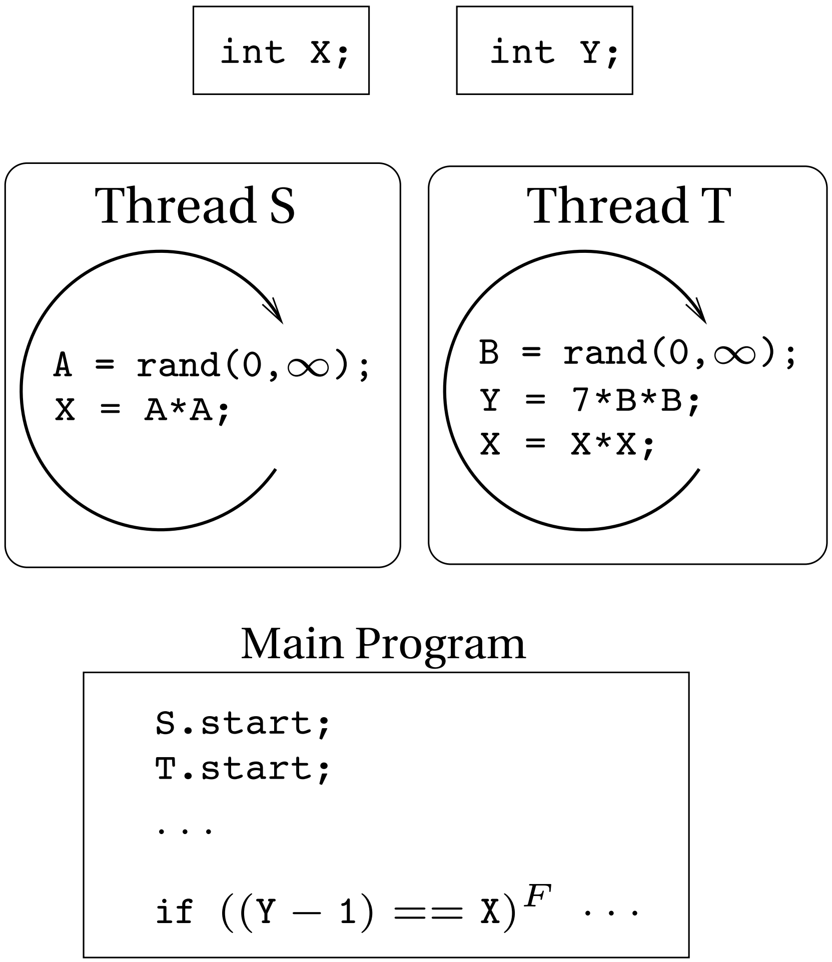
FACT	COMMENTS
$\forall x, y \in \mathcal{I}, 7y^2 - 1 \neq x^2$	
$\forall x \in \mathcal{I}, 2 \mid (x + x^2)$	
$\forall x \in \mathcal{I}, 3 \mid (x^3 - x)$	
$\forall x \in \mathcal{I}, \sum_{i=1, 2 \nmid i}^{2x-1} i = x^2$	The sum of the odd integers is a perfect square.
$\forall x \in \mathcal{I}^+, 8 \mid (7^{2x+1} + 17^x)$	
$\forall x \in \mathcal{I}^+, 2 \mid \lfloor \frac{x^2}{2} \rfloor$	The second bit of a squared number is always 0.

Manufacturing Opaque Predicates

- Control transformations require strong opaque predicates.
- Threat-model: Deobfuscators will use static analysis.
- Base opaque predicates on hard static analysis problems, such as alias analysis.



Opaque Predicates by Concurrency

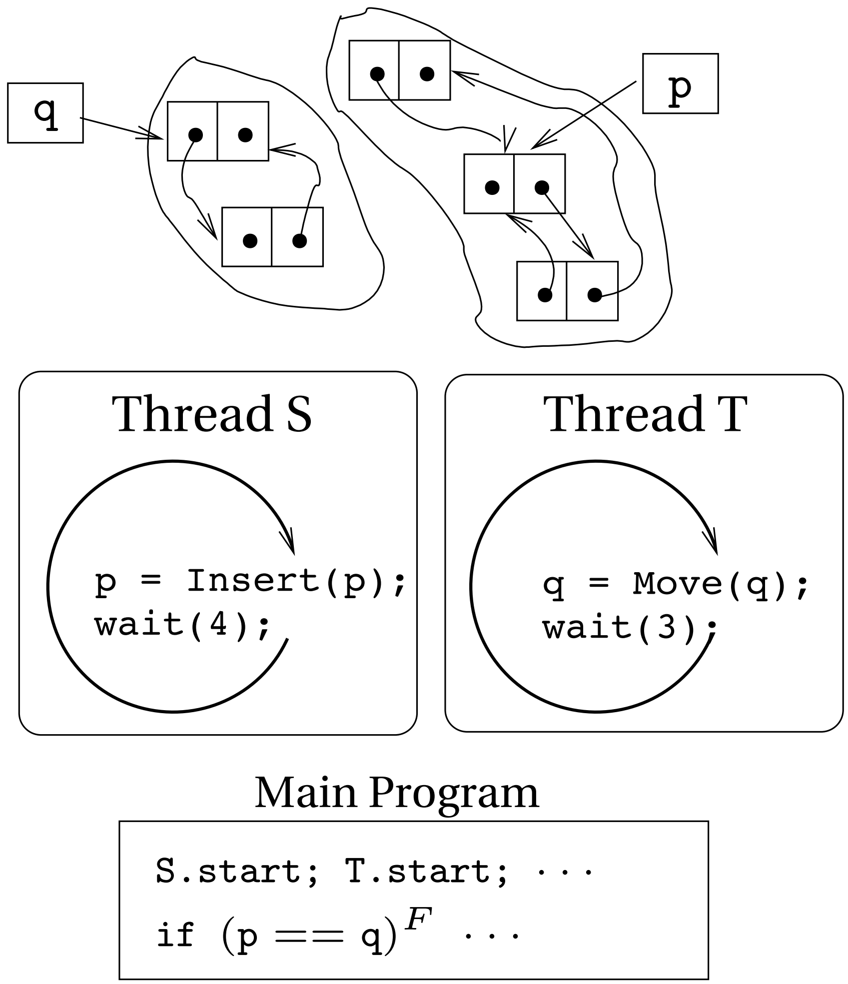


- Parallel programs are hard to analyze statically:

$$\mathbf{PAR}\{S_1; S_2; \cdots; S_n\}$$

  can be executed in $n!$ different ways.
- We create a set of threads that occasionally update a global data structure V.
- V is kept in a state such that opaque queries can be made.

Concurrency & Aliasing



- If we let V be a dynamic data structure, we can combine interleaving and aliasing effects.
- The threads asynchronously move the global pointers `p` and `q` around in their respective components.
- This is quite resilient to de-obfuscation attacks by static analysis.

Obfuscating Data Transformations

p	q	V	$2p+q$
0	0	False	0
0	1	True	1
1	0	True	2
1	1	False	3

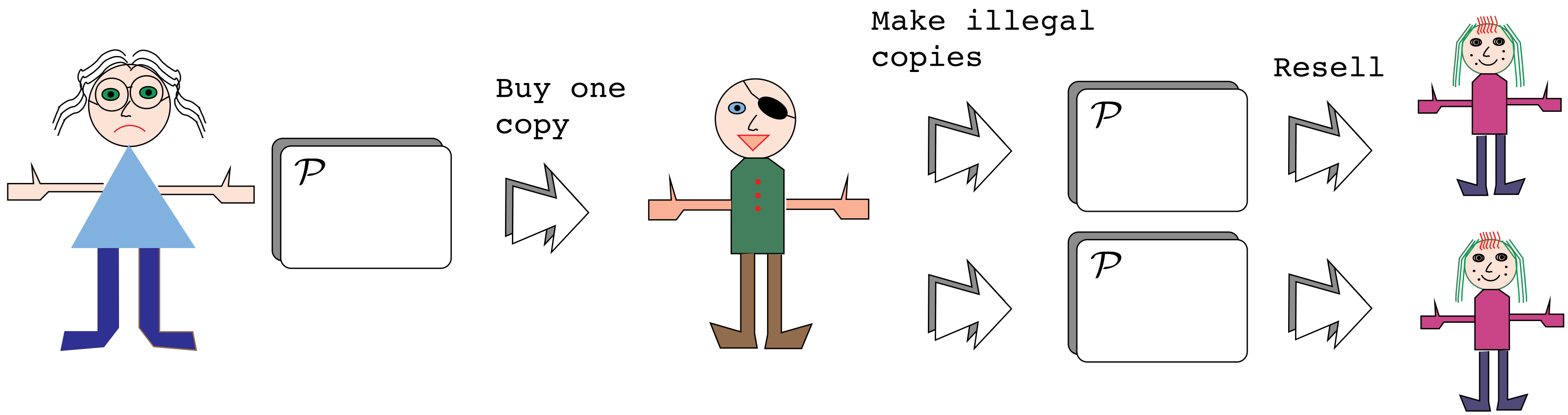
AND	0	1	2	3
0	3	0	0	0
1	3	1	2	3
2	0	2	1	3
3	3	0	0	3

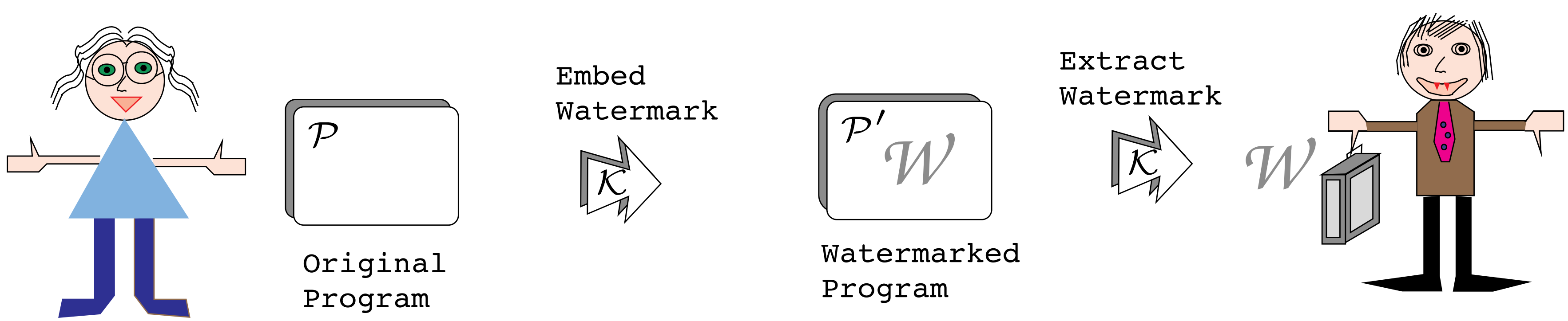
```
bool A,B,C;
B = False;
C = False;
C = A & B;
C = A & B;
if (A) ···;
if (B) ···;
```

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

```
short a1,a2,b1,b2,c1,c2;
b1=0; b2=0;
c1=1; c2=1;
x=AND[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
c1=(a1 ^ a2) & (b1 ^ b2); c2=0;
x=2*a1+a2; if ((x==1) || (x==2)) ···;
if (b1 ^ b2) ···;
```

Software Piracy vs. Watermarking



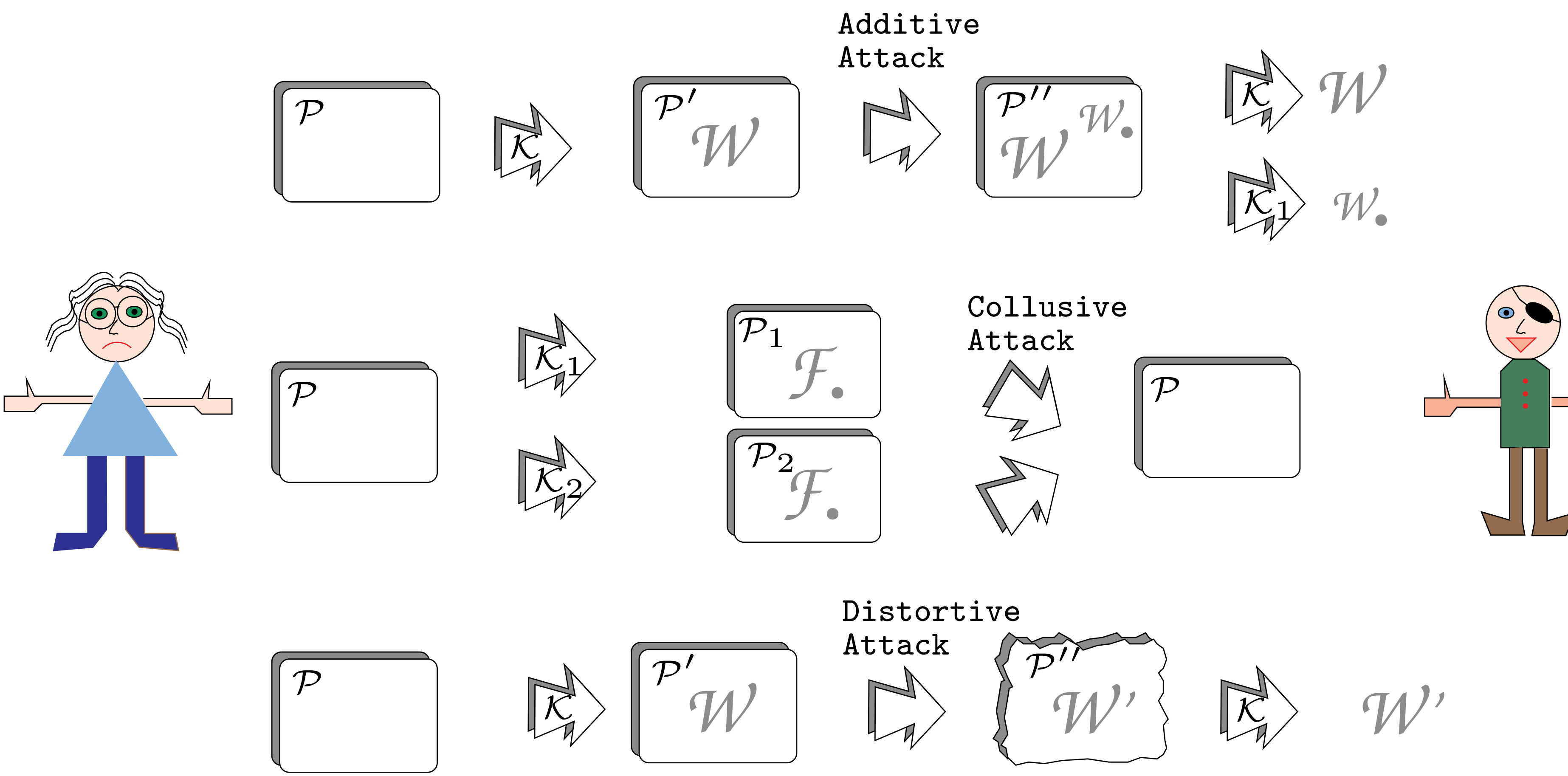


Watermarking & Fingerprinting

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.

Attacks on Software Watermarks



Principles of Software Watermarking

Embed a structure W into a program P such that:

Maximize resilience

- W can be reliably located and extracted from P

Maximize bit-rate

- W is large

Maximize performance

- the embedding does not adversely affect P

Maximize stealth

- the embedding does not change P's statistical properties

Signature property

- W has an "interesting" mathematical property.

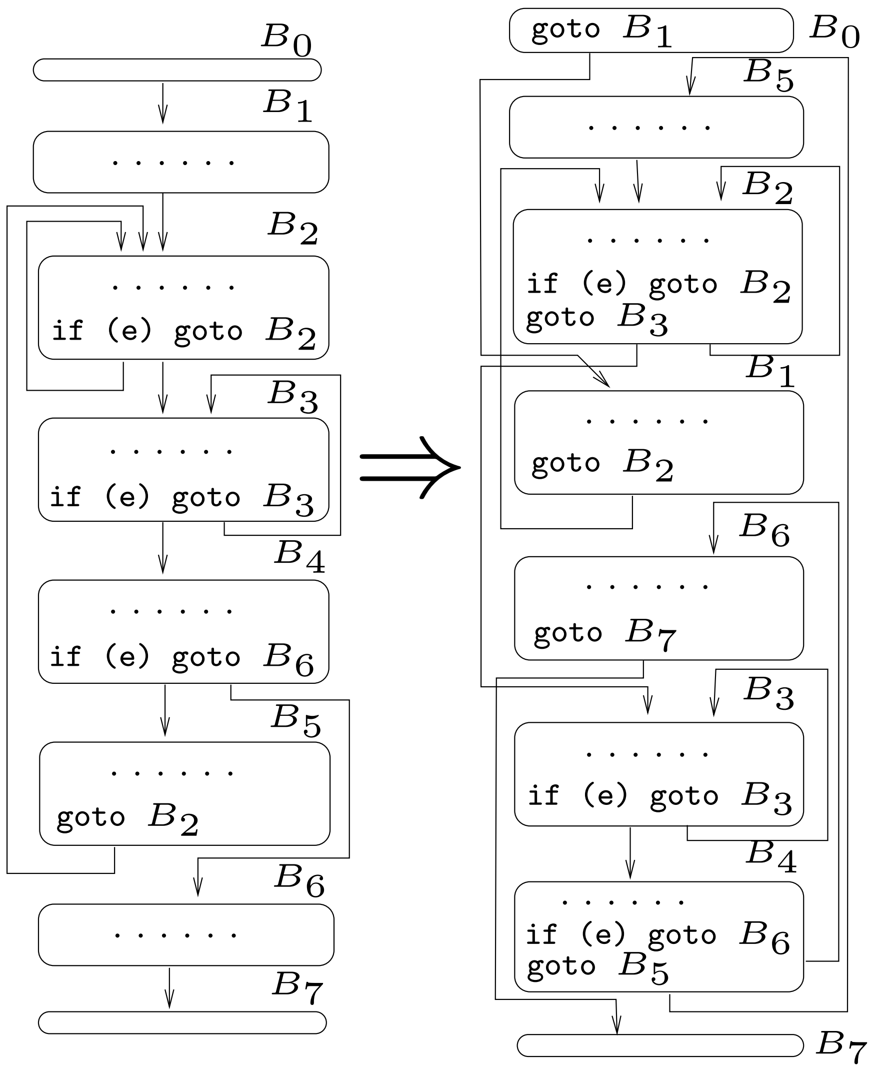
Static Data Watermarks – DICE Method

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

```
         ...
}
```

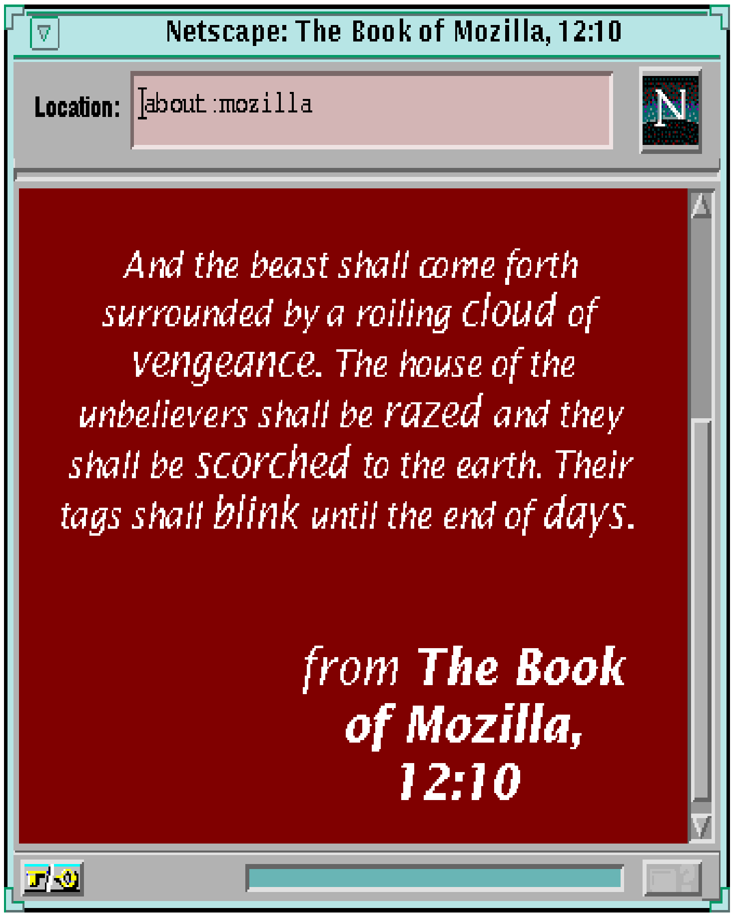
- US Patent 5,745,569, Jan 1996.
- A watermarked media object is embedded in the program's static data segment.

Static Code Watermarks – Microsoft



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

Dynamic Watermarks — Easter Eggs



- The watermark performs an action that is immediately perceptible.

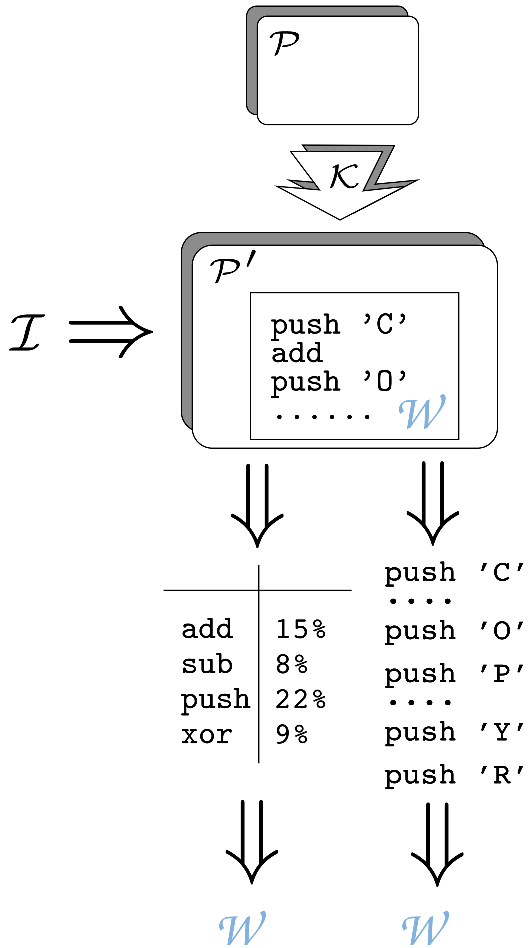
  $\Downarrow$

  Extraction is trivial.

- Effects must not be too subtle.

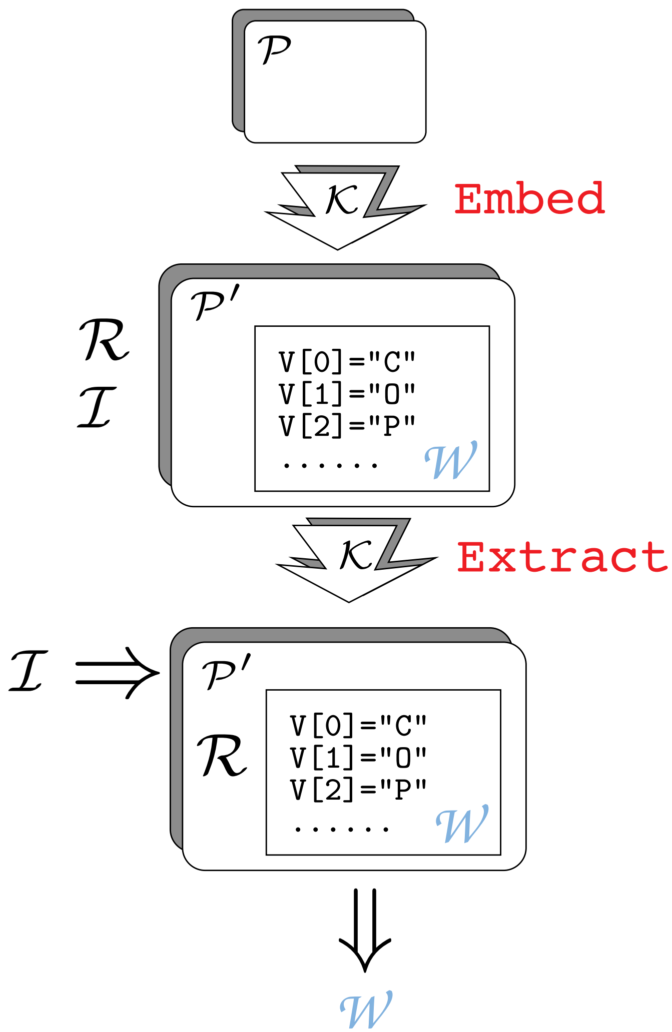
- `www.eeggs.com/lr.html`.

Dynamic Watermarks — Execution Trace



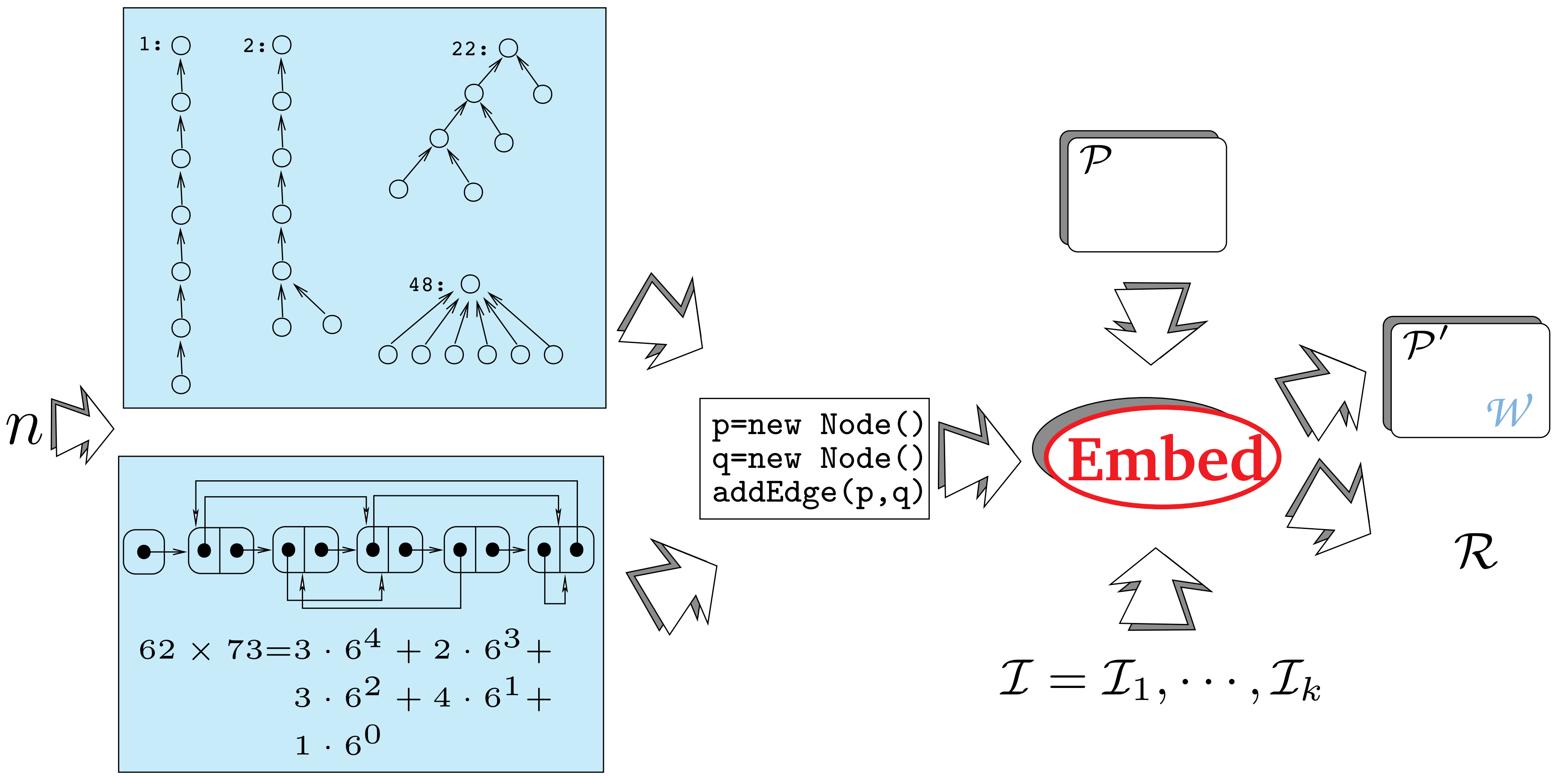
- Dynamic watermarks are constructed at run-time in response to a secret input sequence $\mathcal{I} = \mathcal{I}_1, \cdots, \mathcal{I}_k$.
- Execution trace watermarks are embedded within the instruction or address trace.
- The watermark is extracted from
  - the actual trace, or
  - from some statistical property of the trace.

Dynamic Watermarks — Data Structure

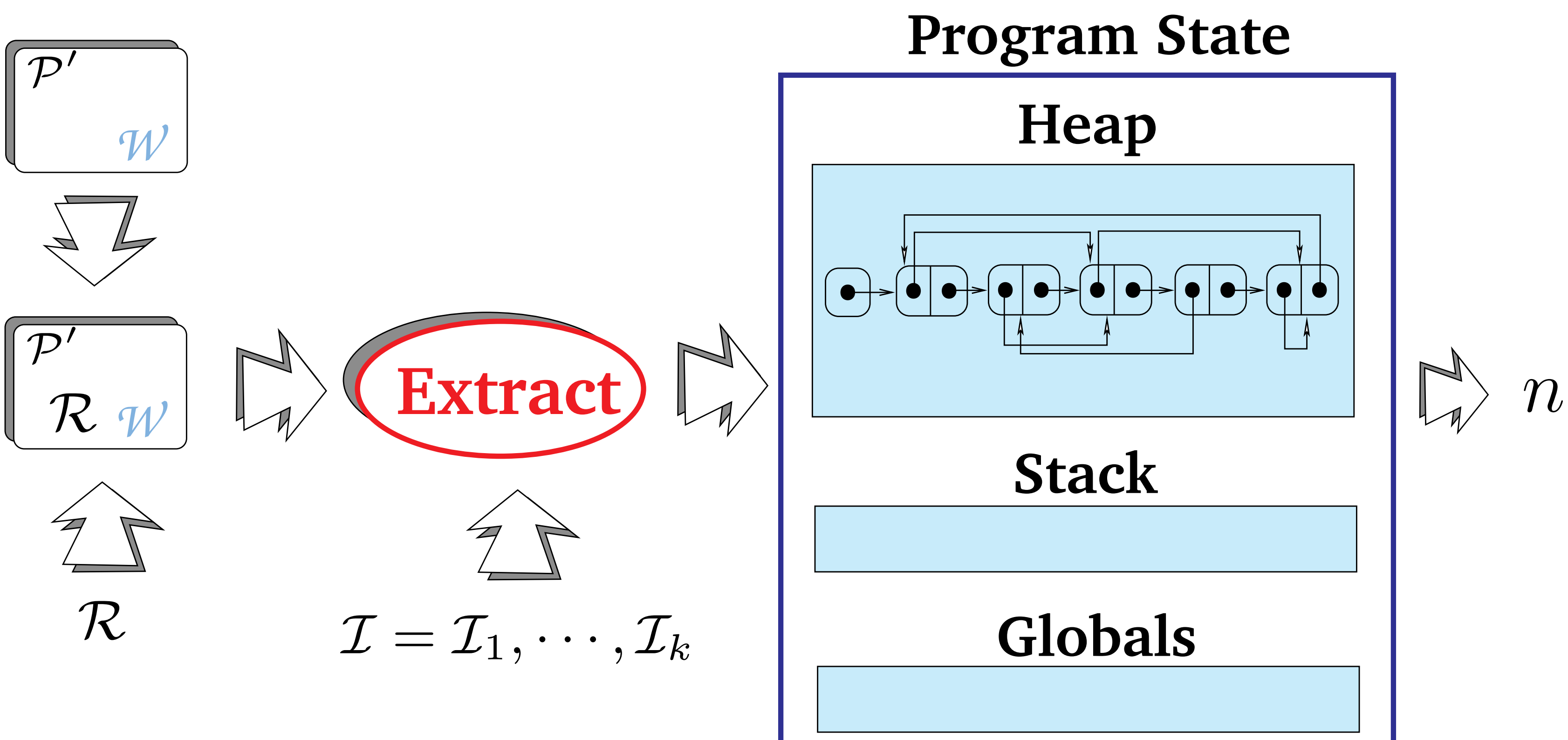


- The watermark is embedded within the state (globals, heap, stack) of the program.
- A recognizer $\mathcal{R}$ extracts the watermark by examining the state after input $\mathcal{I}$.
- No "special" output is produced.
- $\mathcal{R}$ is not shipped.

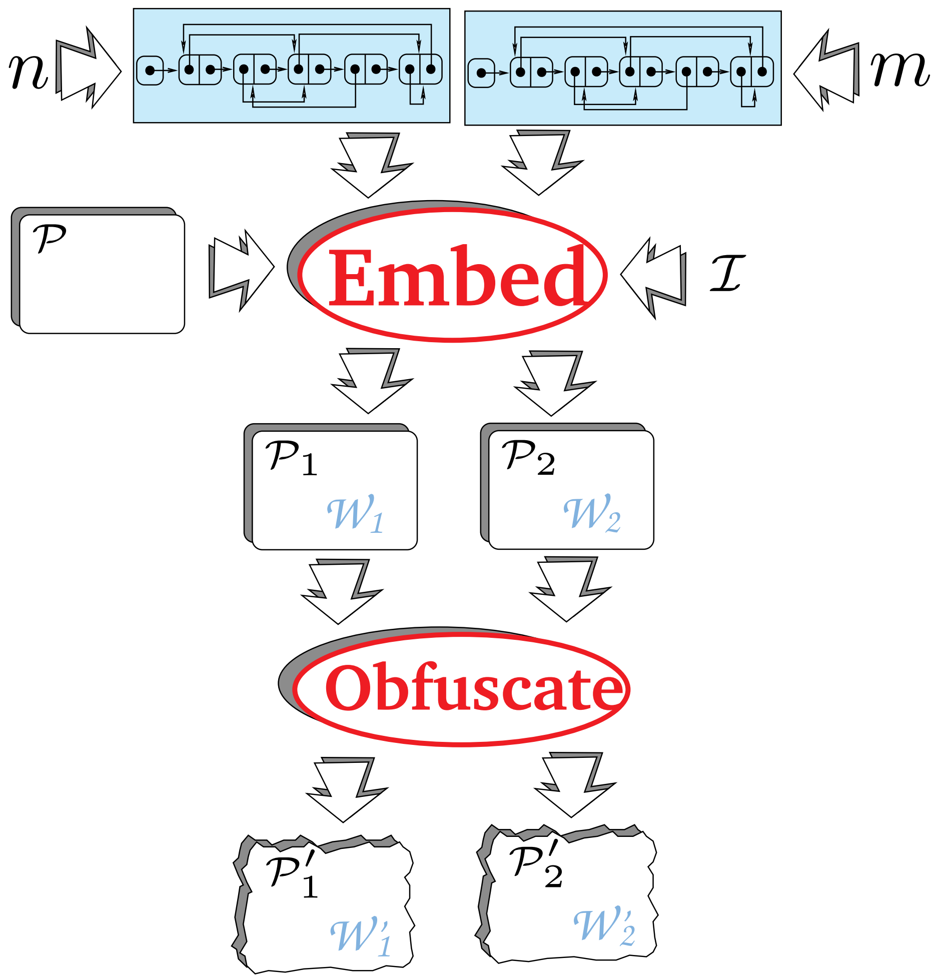
Dynamic Graph Watermark — Embed



Dynamic Graph Watermark — Extract



Dynamic Watermarks — Obfuscate



- Using graph watermarks for fingerprinting leaves us open to collusive attacks.

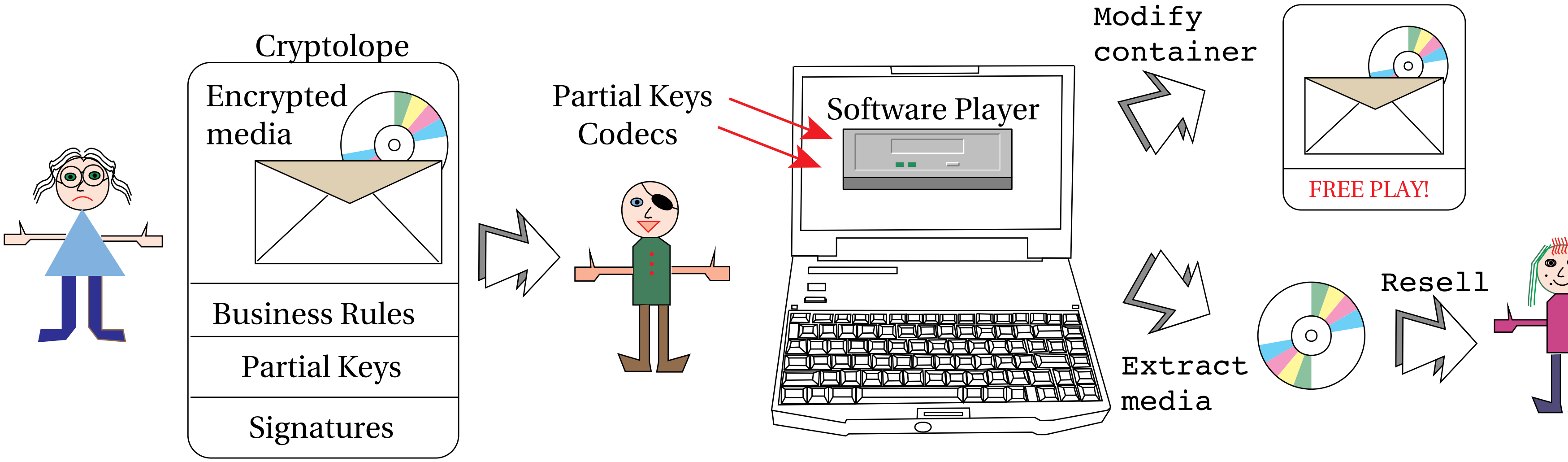
$$\Downarrow$$

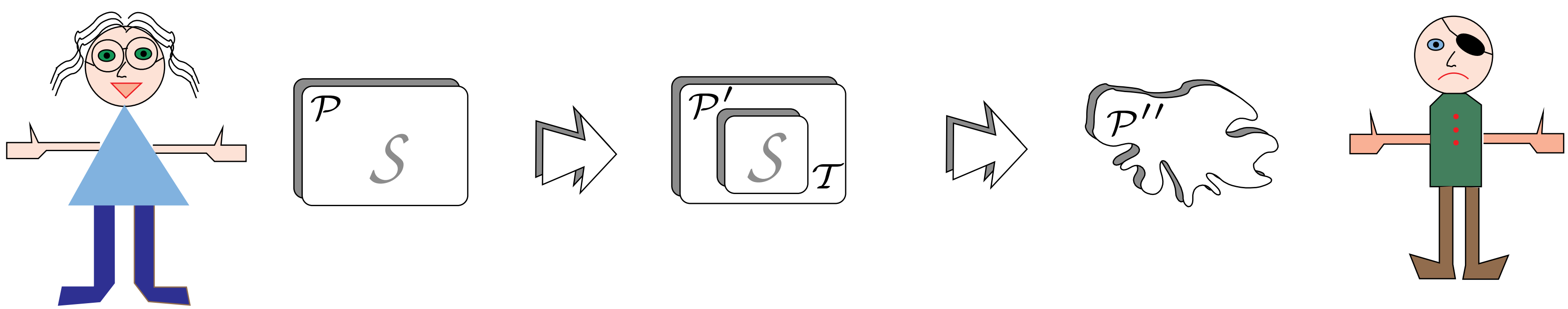
Embed

+

Obfuscate

Tampering vs. Tamperproofing





Tamperproofing

Attacks	Defense
Don't execute P if 1. P's watermark $\mathcal{W}$ has been altered, 2. P has been augmented with a virus, 3. P's security sensitive code has been altered.	Protect P such that 1. we can detect that P has been altered, 2. we can cause P to fail when tampering is detected.

- Detection and failure should be separated by time and space.

Tamperproofing Defenses

Inspect Code	$f\left(\boxed{\mathcal{P}}\right) =?$	Examine the executable program itself.
Inspect State	$f\left(\mathcal{I}, \boxed{\mathcal{P}}\right) =?$	Use *program result checking* to examine intermediate results.
Generate Code	$f(\mathcal{X}) = \boxed{\mathcal{P}}$	Generate the executable on the fly.

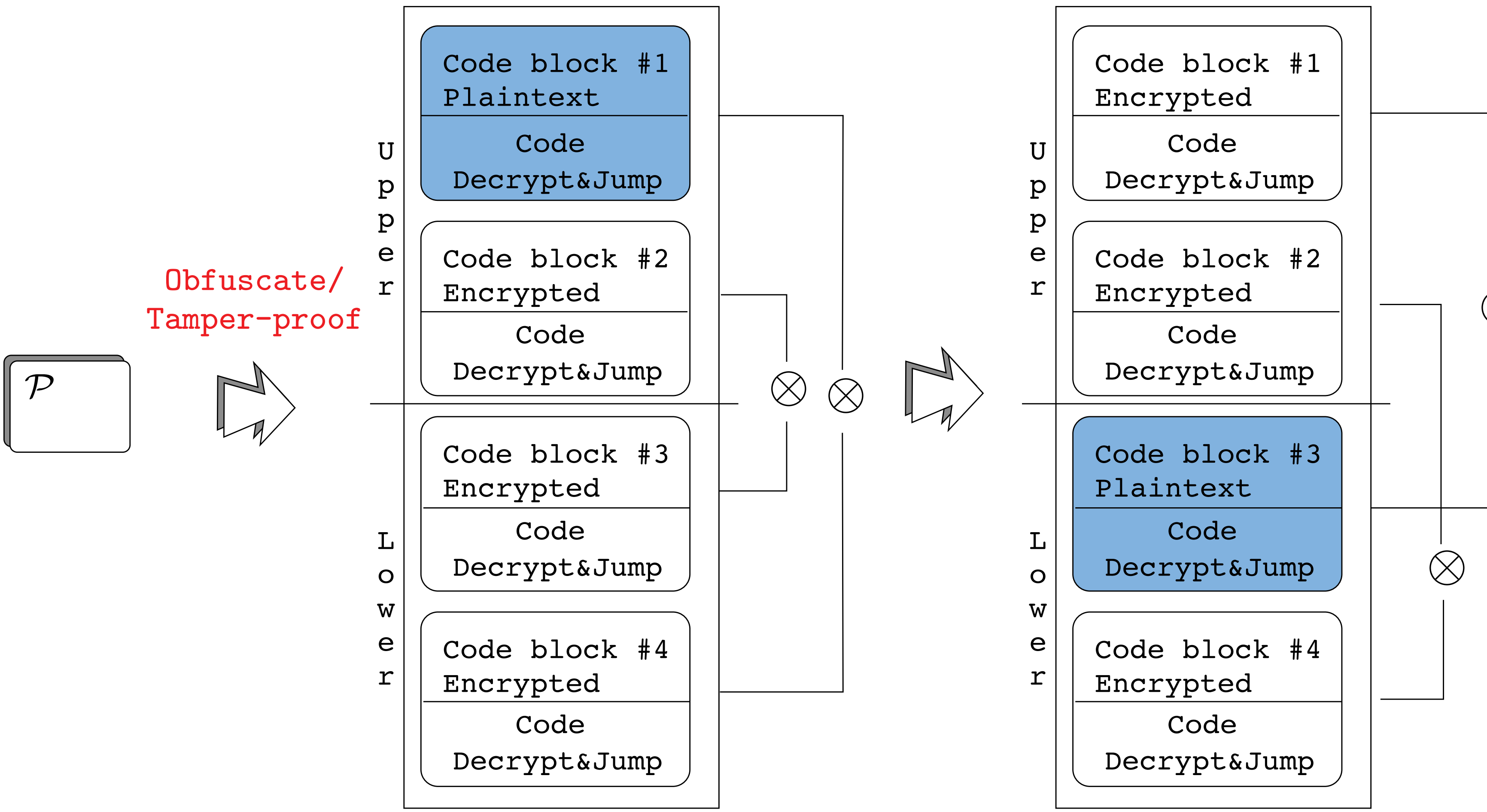
Tamperproofing Watermarks – DICE

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

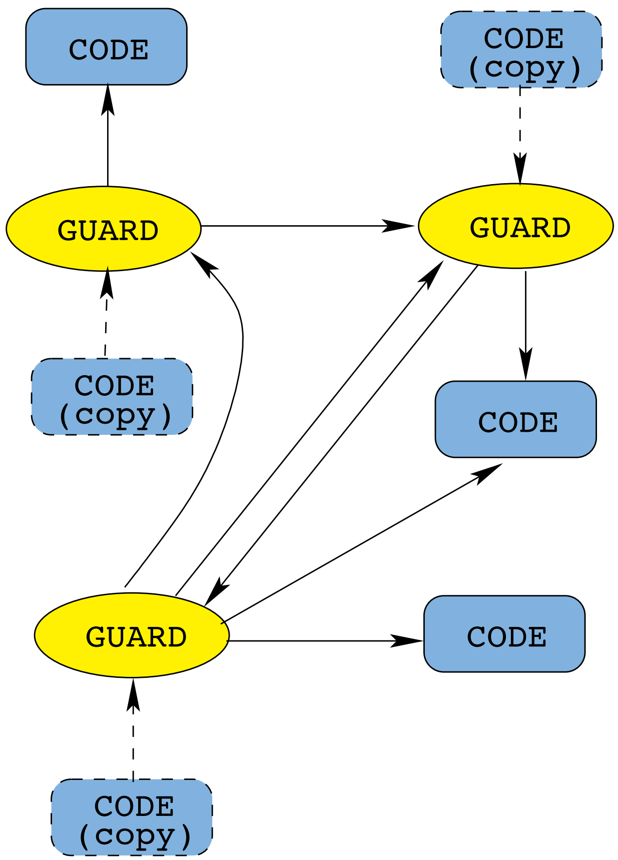

```
      ...
  Code R = Decode(C);
  Execute(R);
}
```

- Generate Code
- “Essential” parts of the program are steganographically encoded into the media.
- If the watermarked image is attacked, the embedded code will crash.

Tamperproofing — Aucsmith/Intel

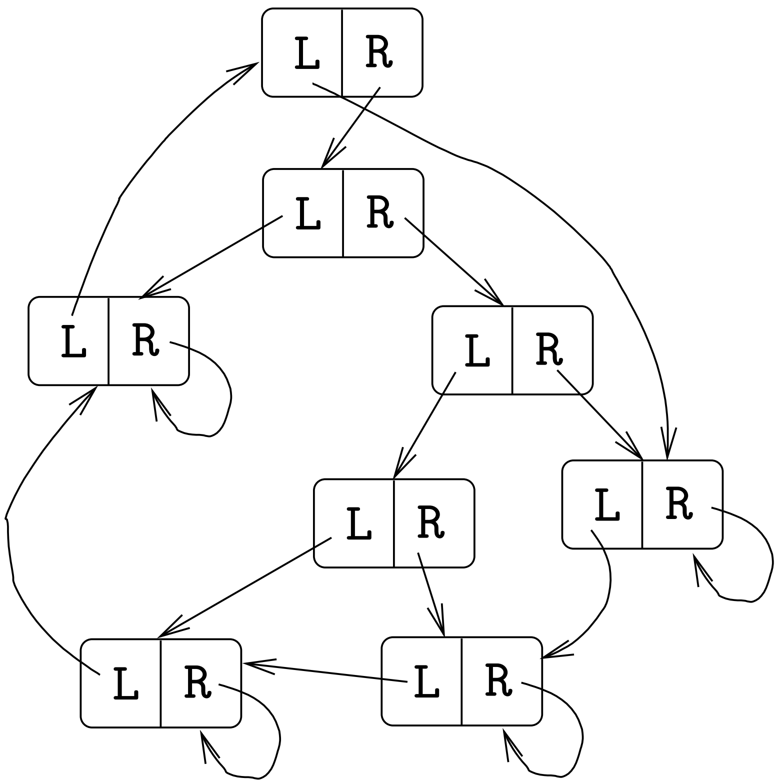


Tamperproofing by Guards



- Chang & Atallah (Purdue).
- Extend the code with *guards* which
  1. checksum the code, and
  2. repair tampered segments.
- Guards form a network, checking and repairing each other.

Tamperproofing Graph Watermarks



- Inspect State
- A planted plane cubic tree.
- Planarity check:

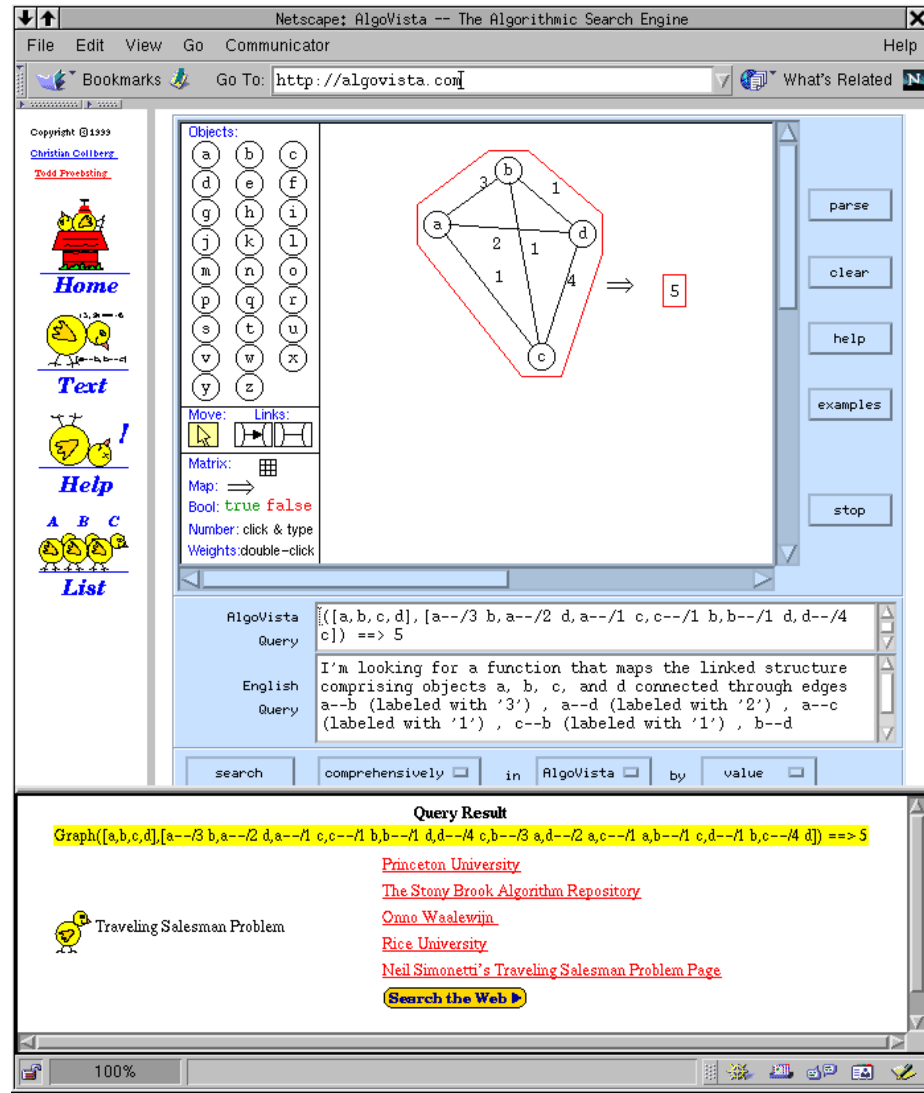
  For each internal node x, the left-most child of x's right subtree is L-linked to the right-most child of x's left subtree.

Discussion

- What's our threat model?
  1. Manual inspection?
  2. Static analysis?
  3. Dynamic analysis?
  4. Class attacks?
- How do we evaluate software protection techniques?
  1. Runtime overhead (time/space)?
  2. Stealth?
  3. Resilience to semantics-preserving transformations?
- What theoretical approach should we take?

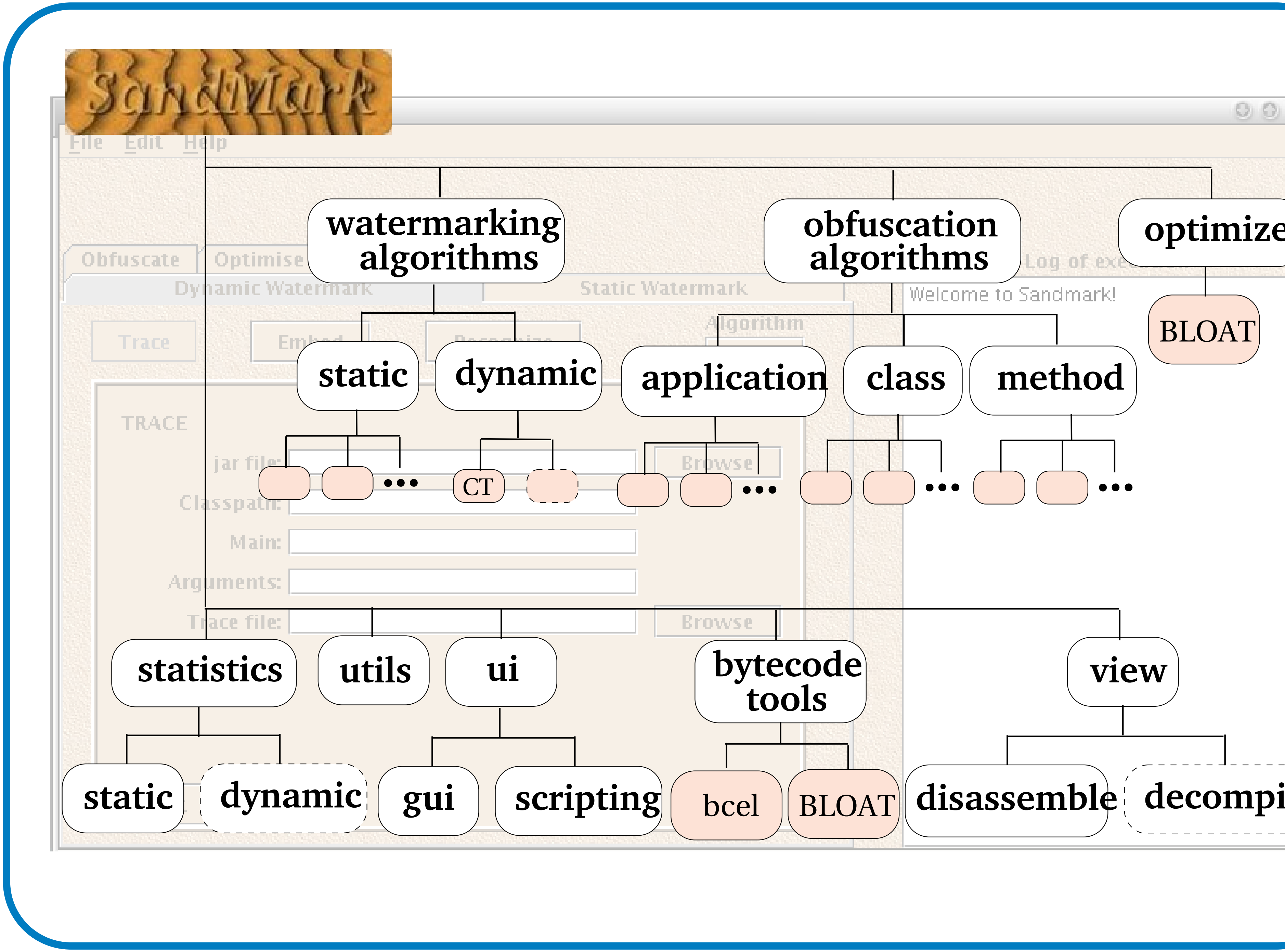
AλgoVista

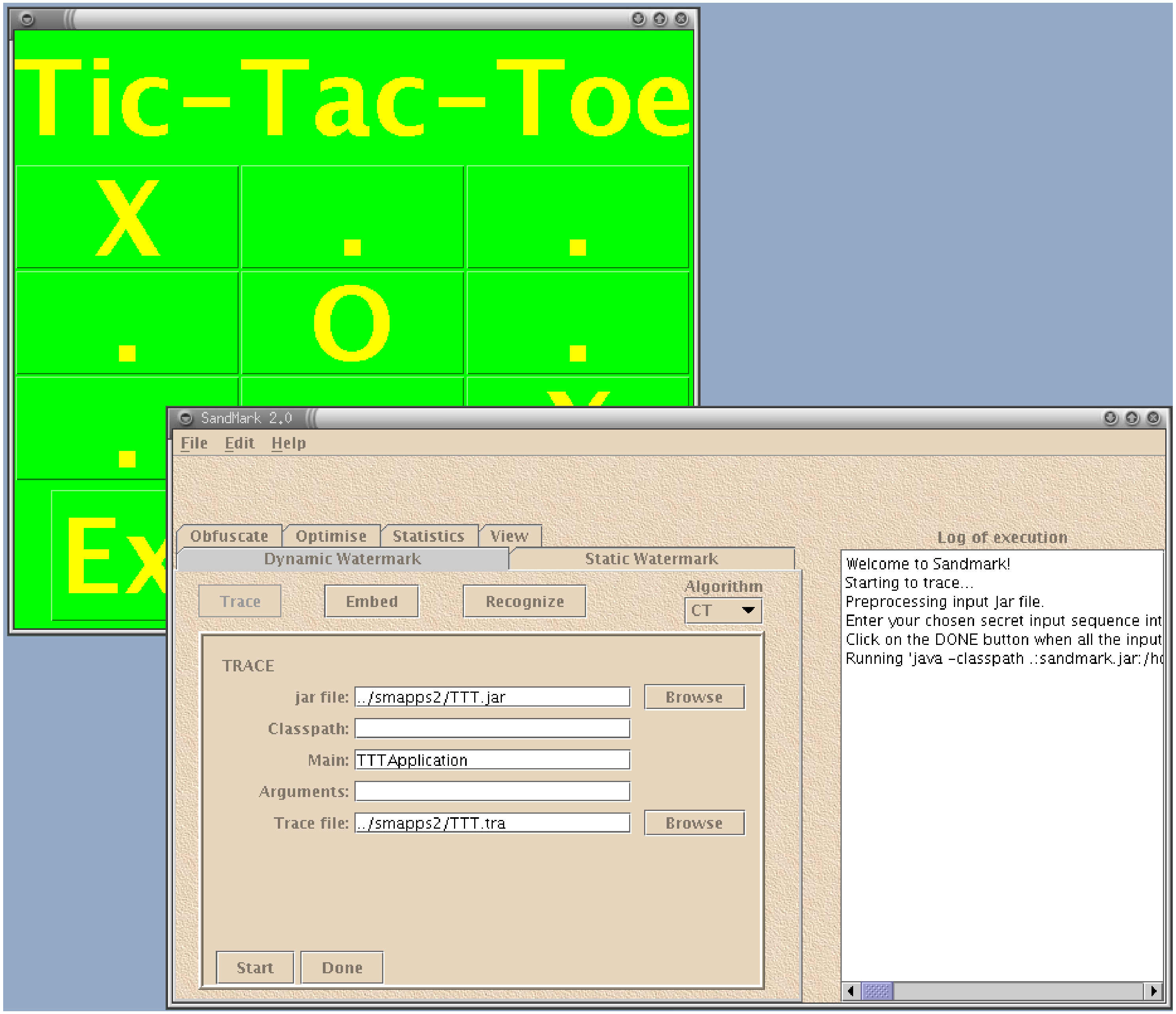
- A search engine for programmers.
- *Query-by-example* not *query-by-keyword*.
- *Draw* a description of the problem you're looking for.
- Joint work with Todd Proebsting.
- `www.algovista.com.`

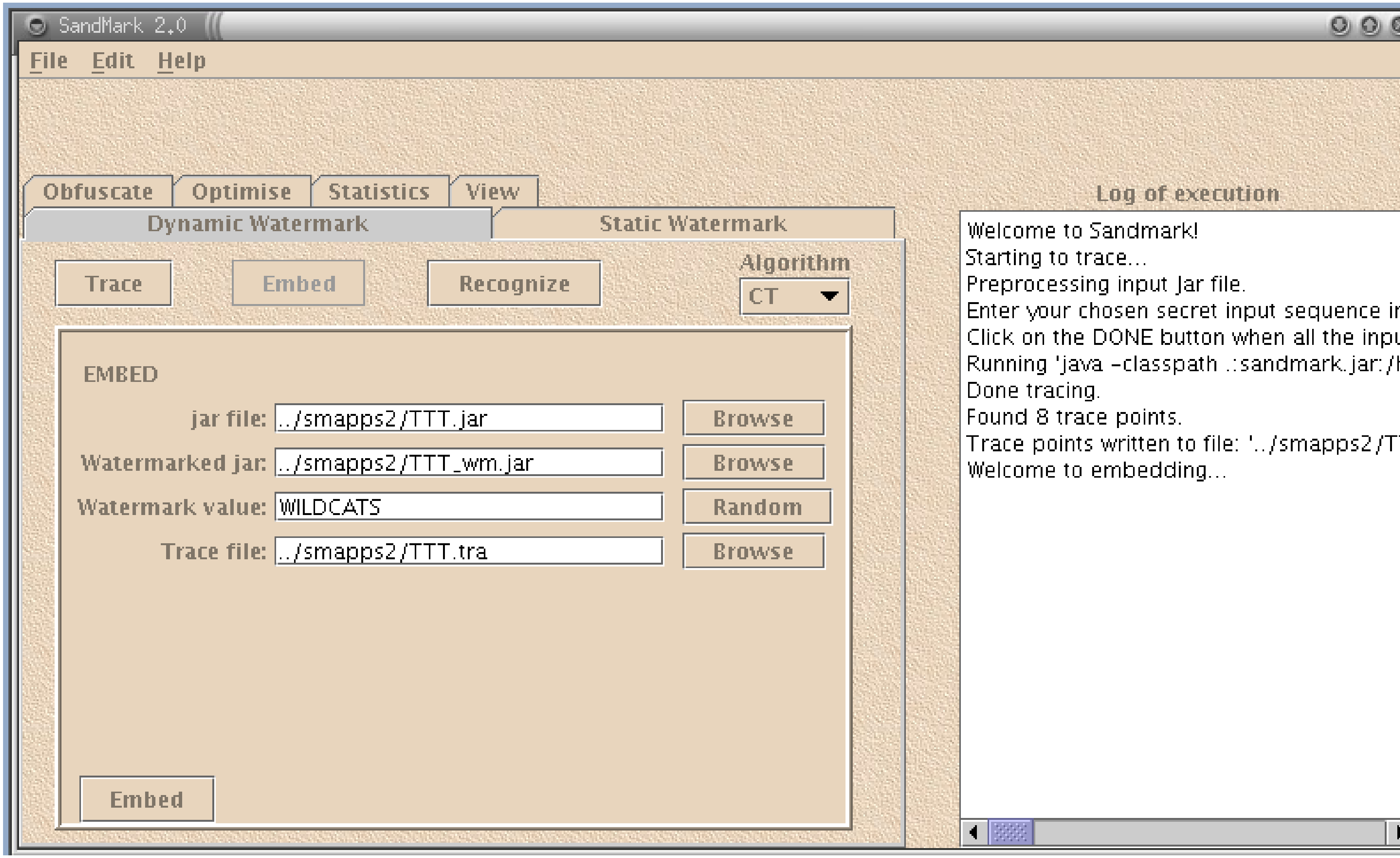


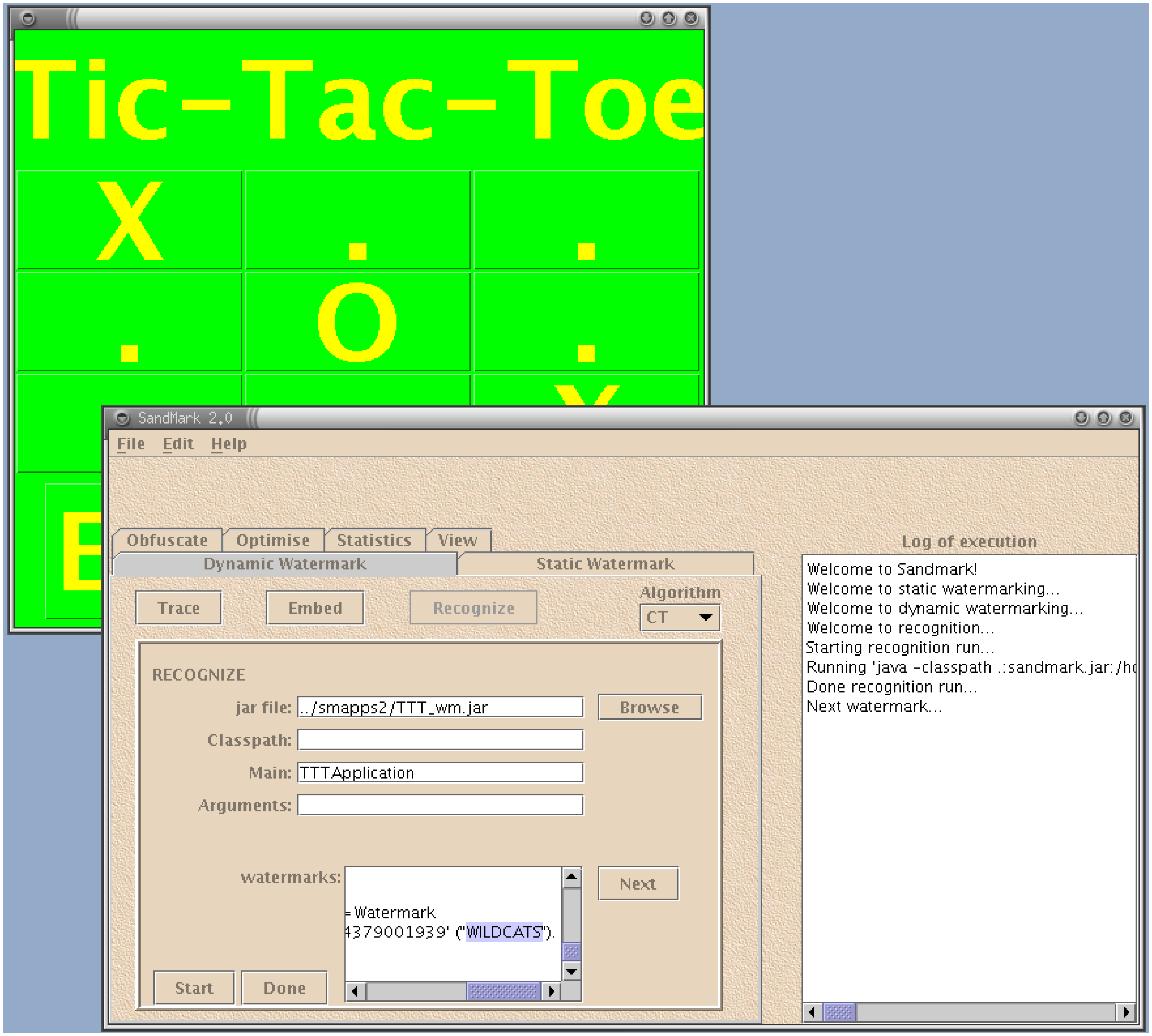
SandMark

- SandMark is our framework for studying the effectivness of software protection techniques.
- Our goal is to implement and evaluate *every* known algorithm.
- SandMark watermarks, obfuscates, and tamper-proofs Java applications.
- SandMark is 40 KLOC of Java.
- SandMark uses a very simple plug-and-play architecture: drop in a new algorithm, type `make`...
- We're still coding; no results yet.


SandMark
watermarking algorithms
obfuscation algorithms
optimize
static
dynamic
CT
application
class
method
BLOAT
statistics
utils
ui
bytecode tools
view
static
dynamic
gui
scripting
bcel
BLOAT
disassemble
File Edit Help
Obfuscate
Optimise
Dynamic Watermark
Static Watermark
Trace
TRACE
jar file:
Classpath:
Main:
Arguments:
Trace file:
Browse
Welcome to Sandmark!


Tic-Tac-Toe
X
O
SandMark 2.0
File
Edit
Help
Obfuscate
Optimise
Statistics
View
Dynamic Watermark
Static Watermark
Trace
Embed
Recognize
Algorithm
CT
TRACE
jar file:
../smapps2/TTT.jar
Browse
Classpath:
Main:
TTTApplication
Arguments:
Trace file:
../smapps2/TTT.tra
Browse
Start
Done
Log of execution
Welcome to Sandmark!
Starting to trace...
Preprocessing input Jar file.
Enter your chosen secret input sequence int
Click on the DONE button when all the input
Running 'java -classpath .:sandmark.jar:/h


SandMark 2.0
File
Edit
Help
Obfuscate
Optimise
Statistics
View
Dynamic Watermark
Static Watermark
Trace
Embed
Recognize
Algorithm
CT
EMBED
jar file:
../smapps2/TTT.jar
Browse
Watermarked jar:
../smapps2/TTT_wm.jar
Browse
Watermark value:
WILDCATS
Random
Trace file:
../smapps2/TTT.tra
Browse
Embed
Log of execution
Welcome to Sandmark!
Starting to trace...
Preprocessing input Jar file.
Done tracing.
Found 8 trace points.
Welcome to embedding...


Tic-Tac-Toe
X
O
SandMark 2.0
File
Edit
Help
Obfuscate
Optimise
Statistics
View
Dynamic Watermark
Static Watermark
Trace
Embed
Recognize
Algorithm
CT
RECOGNIZE
jar file:
../smapps2/TTT_wm.jar
Browse
Classpath:
Main:
TTTApplication
Arguments:
watermarks:
Watermark
4379001939' ("WILDCATS").
Next
Start
Done
Log of execution
Welcome to Sandmark!
Welcome to static watermarking...
Welcome to dynamic watermarking...
Welcome to recognition...
Starting recognition run...
Done recognition run...
Next watermark...