Software Watermarking

Watermarking

- Embed a unique identifier into the executable of a program.

Watermarking

- Embed a unique identifier into the executable of a program.
- A watermark is much like a *copyright notice*.

Watermarking

- Embed a unique identifier into the executable of a program.
- A watermark is much like a *copyright notice*.
- Won't prevent an attacker from reverse engineering or pirating it the program.

Watermarking

- Embed a unique identifier into the executable of a program.
- A watermark is much like a *copyright notice*.
- Won't prevent an attacker from reverse engineering or pirating it the program.
- Allows us to show that the program the attacker claims to be his, is actually ours.

Watermarking

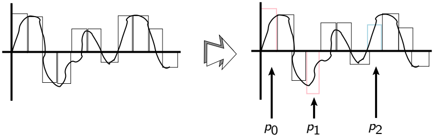
- Embed a unique identifier into the executable of a program.
- A watermark is much like a *copyright notice*.
- Won't prevent an attacker from reverse engineering or pirating it the program.
- Allows us to show that the program the attacker claims to be his, is actually ours.
- *Software fingerprinting*: every copy you sell will have a different unique mark in it

Watermarking

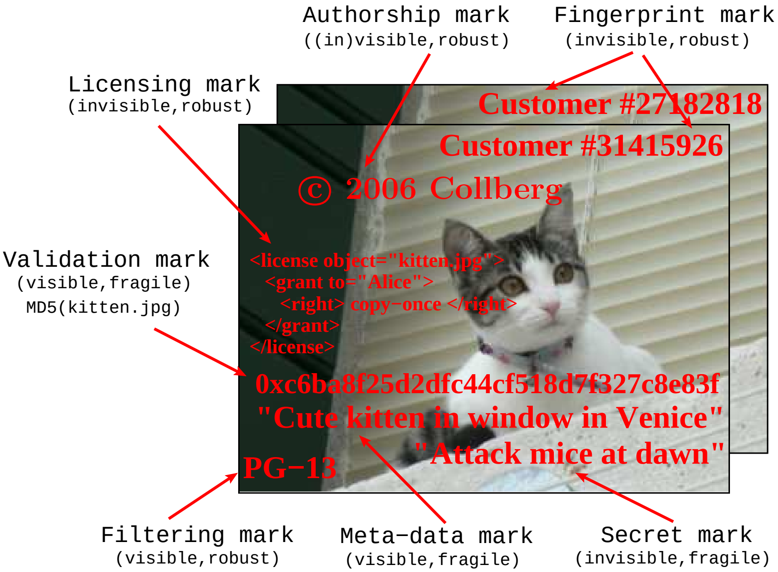
- Embed a unique identifier into the executable of a program.
- A watermark is much like a *copyright notice*.
- Won't prevent an attacker from reverse engineering or pirating it the program.
- Allows us to show that the program the attacker claims to be his, is actually ours.
- *Software fingerprinting*: every copy you sell will have a different unique mark in it
- Trace the copy back to the original owner, and take legal action.

History and Applications

p. 468



History and Applications


Authorship mark
((in)visible,robust)
Fingerprint mark
(invisible,robust)
Licensing mark
(invisible,robust)
Customer #27182818
Customer #31415926
© 2006 Collberg
Validation mark
(visible,fragile)
MD5(kitten.jpg)
<license object="kitten.jpg">
<grant to="Alice">
<right> copy−once </right>
</grant>
</license>
0xc6ba8f25d2dfc44cf518d7f327c8e83f
"Cute kitten in window in Venice"
"Attack mice at dawn"
PG−13
Filtering mark
(visible,robust)
Meta−data mark
(visible,fragile)
Secret mark
(invisible,fragile)

Visible vs. invisible marks

- A visible mark acts as a deterrent against misuse.

Visible vs. invisible marks

- A visible mark acts as a deterrent against misuse.
- An invisible mark, can only be extracted using a secret not available to the end user.

Robust vs. fragile marks

- A robust mark is difficult to modify (accidentally or deliberately).

Robust vs. fragile marks

- A robust mark is difficult to modify (accidentally or deliberately).
- A fragile mark could (and sometimes *should*) be easily destroyed by transformations to the cover object.

Robust vs. fragile marks

- A robust mark is difficult to modify (accidentally or deliberately).
- A fragile mark could (and sometimes *should*) be easily destroyed by transformations to the cover object.
- Marks should survive lossy compression schemes, shrinking, cropping, xeroxing, PAL-to-NTSC,...

Authorship marks

- Embed an identification of the copyright owner in the cover object.

Authorship marks

- Embed an identification of the copyright owner in the cover object.
- Visible marks act as a deterrent and invisible ones allow a web-spider to search for images on the web.

Authorship marks

- Embed an identification of the copyright owner in the cover object.
- Visible marks act as a deterrent and invisible ones allow a web-spider to search for images on the web.
- Example: Playboy's use of Digimarc.

Fingerprint marks

- *Serialize* the cover object, i.e. embed a different mark in every distributed copy.

Fingerprint marks

- *Serialize* the cover object, i.e. embed a different mark in every distributed copy.
- Example: actor Carmine Caridi gave away copies of Academy Award screening tapes,

Fingerprint marks

- *Serialize* the cover object, i.e. embed a different mark in every distributed copy.
- Example: actor Carmine Caridi gave away copies of Academy Award screening tapes,
- Example: Beta copies of software.

Licensing marks

- A *licensing mark* encodes, invisibly and robustly, the way the cover object can be used by the end user.

Licensing marks

- A *licensing mark* encodes, invisibly and robustly, the way the cover object can be used by the end user.
- Integral part of any DRM system.

Licensing marks

- A *licensing mark* encodes, invisibly and robustly, the way the cover object can be used by the end user.
- Integral part of any DRM system.
- Usage rules could be stored in file headers, but using watermarking ensures that the data remains even after transformations.

Meta-data mark

- *Meta-data marks* are visible and (possibly) fragile marks that embed useful data.

Meta-data mark

- *Meta-data marks* are visible and (possibly) fragile marks that embed useful data.
- Example: captions.

Validation marks

- Used by the end user to verify that the marked object is authentic and hasn't been altered.

Validation marks

- Used by the end user to verify that the marked object is authentic and hasn't been altered.
- Example: compute an MD5 sum of an object and embed it as a watermark.

Validation marks

- Used by the end user to verify that the marked object is authentic and hasn't been altered.
- Example: compute an MD5 sum of an object and embed it as a watermark.
- Example: validate that a crime scene photograph hasn't been changed (by moving, say, a gun from one person's hand to another).

Validation marks

- Used by the end user to verify that the marked object is authentic and hasn't been altered.
- Example: compute an MD5 sum of an object and embed it as a watermark.
- Example: validate that a crime scene photograph hasn't been changed (by moving, say, a gun from one person's hand to another).
- Validation marks need to be fragile,

Filtering marks

- A *filtering* or *classification mark* carries classification codes to allow media players to filter out any inappropriate material.

Filtering marks

- A *filtering* or *classification mark* carries classification codes to allow media players to filter out any inappropriate material.
- The mark needs to be robust and visible.

Secret marks

- A *secret mark* is used for covert communication.

Secret marks

- A *secret mark* is used for covert communication.
- *steganography*.

Secret marks

- A *secret mark* is used for covert communication.
- *steganography*.
- Robustness matters not at all.

Secret marks

- A *secret mark* is used for covert communication.
- *steganography*.
- Robustness matters not at all.
- Invisibility is vitally important.

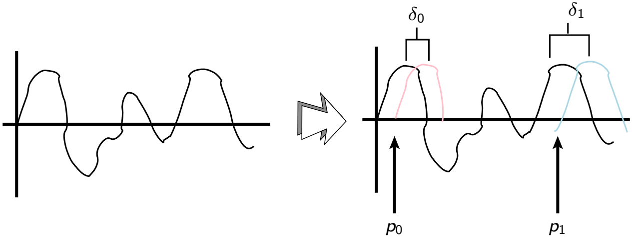
Secret marks

- A *secret mark* is used for covert communication.
- *steganography*.
- Robustness matters not at all.
- Invisibility is vitally important.
- Example:

> *Hidden in the X-rated pictures on several pornographic Web sites and the posted comments on sports chat rooms may lie the encrypted blueprints of the next terrorist attack against the United States or its allies.*

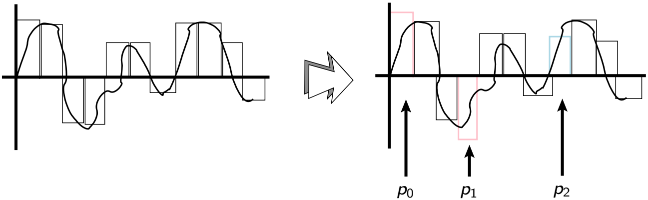
Audio marking: Echo hiding

- Embed echoes that are short enough to be imperceptible to the human ear:



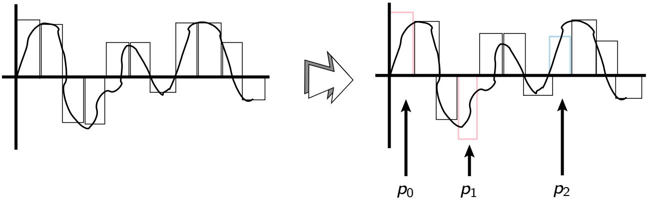
Audio: Least Significant Bit

- LSB of an audio sample is the one that contributes least to your perception,



Audio: Least Significant Bit

- LSB of an audio sample is the one that contributes least to your perception,
- Alter without adversely affecting quality!



Audio: Least Significant Bit

- LSB of an audio sample is the one that contributes least to your perception,
- Alter without adversely affecting quality!
- *Attack*: randomly replace the least significant bit of every sample!

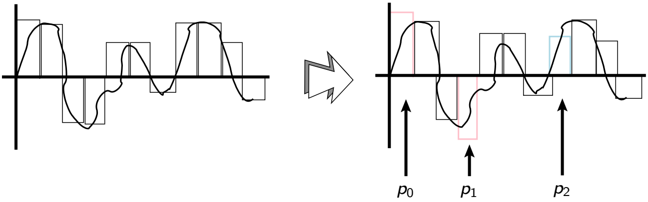


Image: Patchwork

- Embed a single bit by manipulating the brightness of pixels.

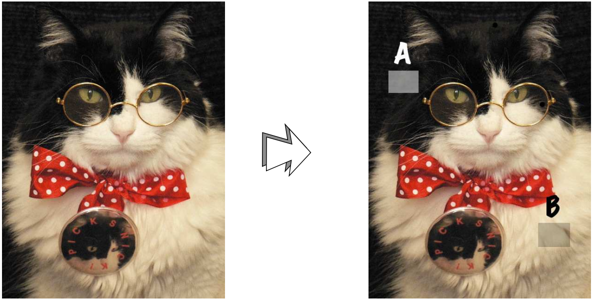


Image: Patchwork

- Embed a single bit by manipulating the brightness of pixels.
- Use a pseudo-random number sequence to trace out pairs (A, B) of pixels

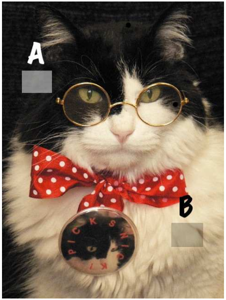
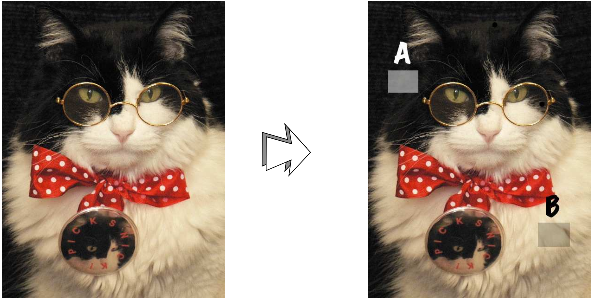


Image: Patchwork

- Embed a single bit by manipulating the brightness of pixels.
- Use a pseudo-random number sequence to trace out pairs (A, B) of pixels
- During embedding adjust the brightness of A up by a small amount, and B down by the same small amount:



Patchwork: Embedding algorithm

EMBED(P, key):

1. `Init_RND(key);` $\delta \leftarrow 5$
2. $i \leftarrow$ `RND()`; $j \leftarrow$ `RND()`
3. Adjust the brightness of pixels a_i and b_i: $a_i \leftarrow a_i + \delta$; $b_j \leftarrow b_j - \delta$
4. repeat from 2 ≈ 10000 times □

Patchwork: Recognition algorithm

RECOGNIZE(P, *key*):

1. `Init_RND(`*key*`);` $S \leftarrow 0$
2. $i \leftarrow$ `RND()`; $j \leftarrow$ `RND()`
3. $S \leftarrow S + (a_i - b_j)$
4. repeat from 2 ≈ 10000 times
5. if $S \gg 0 \Rightarrow 0$ output `"marked!"` □

Blind vs. Informed

- Watermarking recognizers are either *blind* or *informed*.

Blind vs. Informed

- Watermarking recognizers are either *blind* or *informed*.
- To extract a blind mark you need the marked object and the secret key.

Blind vs. Informed

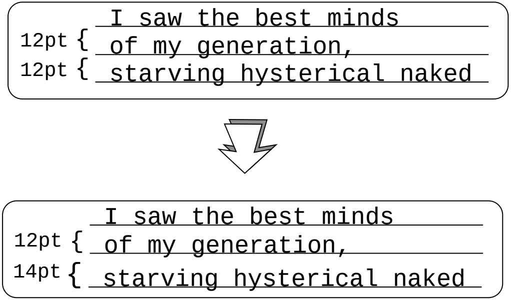
- Watermarking recognizers are either *blind* or *informed*.
- To extract a blind mark you need the marked object and the secret key.
- To extract an informed mark you need extra information, such as original, unwatermarked, object.

Watermarking text

- Cover object types:
    - the text itself with formatting (ASCII text); or
    - free-flowing text;
    - an *image* of the text (PostScript or PDF).

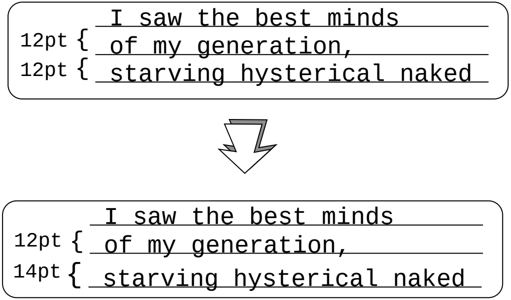
Watermarking Text: PDF

- Similar to marking images.



Watermarking Text: PDF

- Similar to marking images.
- Example: encode 0-bit or a 1-bit by hanging word/line spacing.



Watermarking Text: formatted ASCII

- Encode the mark in white-space: 1 space = 0-bit, 2 spaces = 1-bit:

```
I␣saw␣the␣best␣minds
of␣my␣generation,
starving␣hysterical␣naked
```

```
I␣␣␣saw␣␣the␣best␣␣␣minds
of␣␣␣␣␣␣␣my␣␣␣generation,
starving␣hysterical␣naked
```

Watermarking Text: Synonym replacement

- Replace words with synonyms.
- Insert spelling or punctuation errors.

```
I saw the best minds
of my generation,
starving hysterical naked
```

```
I observed the choice intellects
of my generation,
famished hysterical nude
```

Watermarking Text: Syntax

- Encode a mark in the syntactic structure of an English text:
    1. Devise an extract function which computes a bit from a sentence,
    2. Modify the sentence until it embeds the *right* bit.

```
I saw the best minds
of my generation,
starving hysterical naked
```

```
It was the best minds
of my generation that I saw,
starving hysterical naked
```

Watermarking Text: Atallah et al.

1. Chunk up the watermark, embed one piece per sentence.
2. A function computes one bit per syntax tree node.
3. Modify sentence until these bits embed a watermark chunk.
4. A *marker* sentence precedes every watermark-bearing sentence.

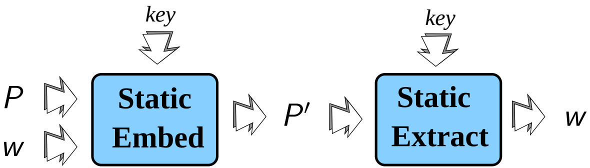
```
I saw the best minds
of my generation,
starving hysterical naked
```

```
I saw the best minds of my
generation. They were starving
hysterical naked. None, baby,
none were smarter than them. Nor
more lacking in supply of essential
nutrients or in more need of
adequate clothing. Baby.
```

Watermarking Software

p. 478

Static watermarks



You care about

- Encoding bitrate
- Stealth
- Resilience to attack

Ideas for Software Watermark Algorithms

Encode the watermark

- in a permutation of a language structure

Ideas for Software Watermark Algorithms

Encode the watermark

- in a permutation of a language structure
- in an embedded media object

Ideas for Software Watermark Algorithms

Encode the watermark

- in a permutation of a language structure
- in an embedded media object
- in a statistical property of the program

Ideas for Software Watermark Algorithms

Encode the watermark

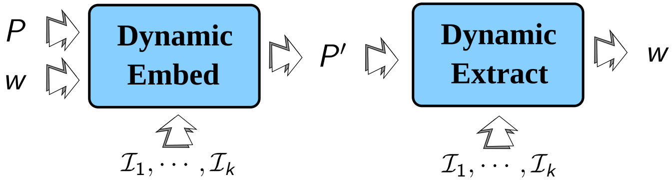
- in a permutation of a language structure
- in an embedded media object
- in a statistical property of the program
- as a solution to a static analysis problem

Ideas for Software Watermark Algorithms

Encode the watermark

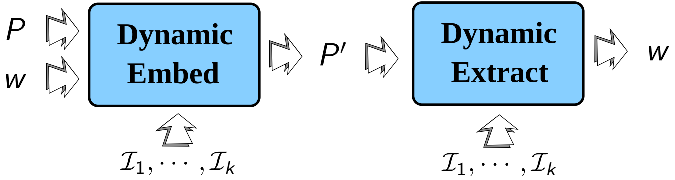
- in a permutation of a language structure
- in an embedded media object
- in a statistical property of the program
- as a solution to a static analysis problem
- in the topology of a CFG

Dynamic watermarks



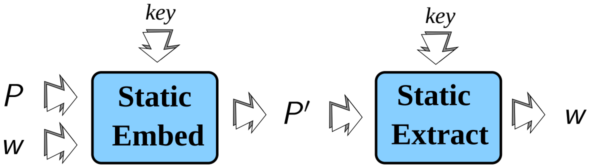
- Encode the watermark in the runtime state of the program

Dynamic watermarks



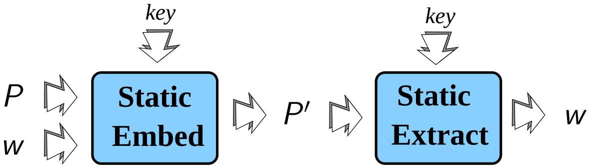
- Encode the watermark in the runtime state of the program
- Dynamic marks appear more robust, but are more cumbersome to use

Attacks against software watermarks



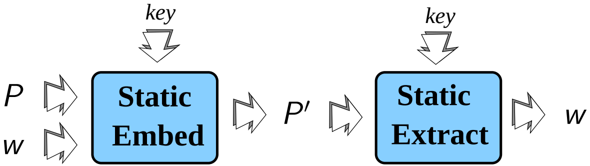
- The adversary knows the algorithm

Attacks against software watermarks



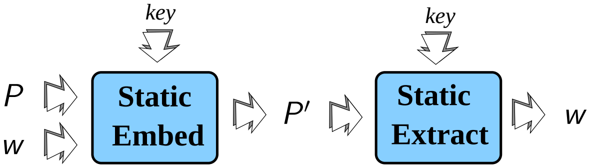
- The adversary knows the algorithm
- The adversary has complete access to the program

Attacks against software watermarks



- The adversary knows the algorithm
- The adversary has complete access to the program
- The adversary doesn't know the key

Attacks against software watermarks



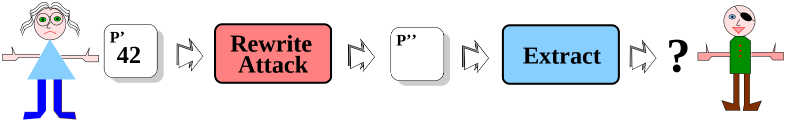
- The adversary knows the algorithm
- The adversary has complete access to the program
- The adversary doesn't know the key
- The adversary doesn't know the embedding location (it's key dependent)

Attacks — Rewrite attack

- Alice has to assume that Bob will try to destroy her marks before trying to resell the program!
- One attack will always succeed...

Attacks — Rewrite attack

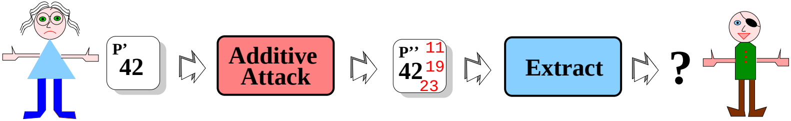
- Alice has to assume that Bob will try to destroy her marks before trying to resell the program!
- One attack will always succeed. . .



- Ideally, this is the *only* effective attack.

Attacks — Additive attack

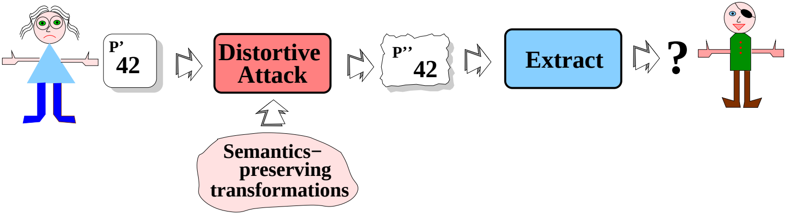
- Bob can also add his own watermarks to the program:



- An additive attack can help Bob to cast doubt in court as to whose watermark is the original one.

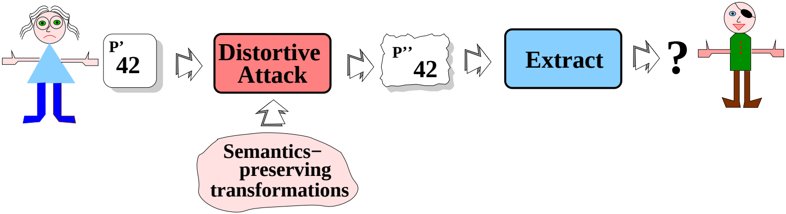
Attacks — Distortive attack

- A distortive attack applies semantics-preserving transformations to try to disturb Alice's recognizer:



Attacks — Distortive attack

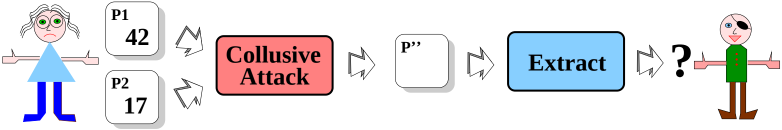
- A distortive attack applies semantics-preserving transformations to try to disturb Alice's recognizer:



- Transformations: code optimizations, obfuscations,...

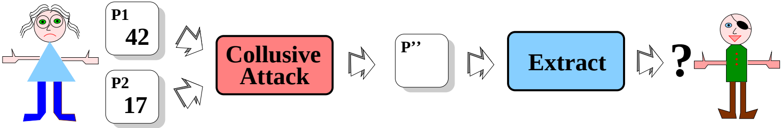
Attacks — Collusive attack

- Bob buys two differently marked copies and comparing them to discover the location of the fingerprint:



Attacks — Collusive attack

- Bob buys two differently marked copies and comparing them to discover the location of the fingerprint:

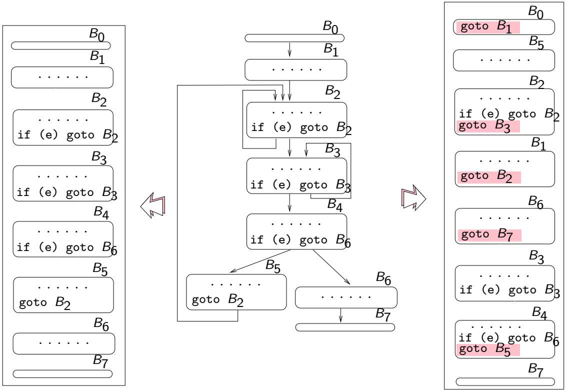


- Alice should apply a different set of obfuscations to each distributed copy, so that comparing two copies of the same program will yield little information.

Watermarking Algorithms

Watermarking by Permutation

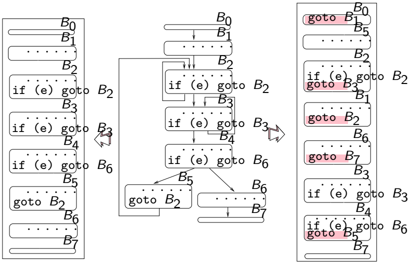
p. 486



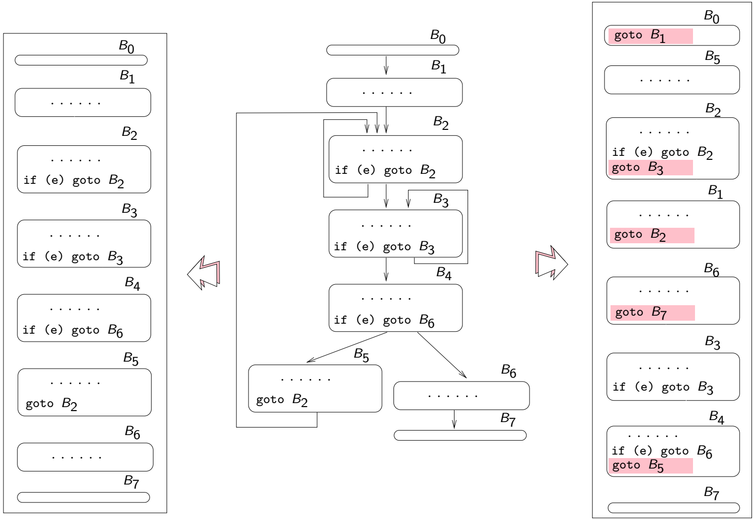
Algorithm WMDM

p. 488

Reordering Basic Blocks



Algorithm WMDM: Reordering Basic Blocks



Algorithm WMDM: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.

Algorithm WMDM: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
- Negligible slowdown for a set of Java benchmarks.

Algorithm WMDM: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
- Negligible slowdown for a set of Java benchmarks.
- If you have m items to reorder you can encode

$$\log_2(m!) \approx \log_2(\sqrt{2\pi m}(m/e)^m) = \mathcal{O}(m \log m)$$

watermarking bits.

Algorithm WMDM: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
- Negligible slowdown for a set of Java benchmarks.
- If you have m items to reorder you can encode

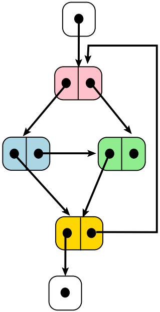
$$\log_2(m!) \approx \log_2(\sqrt{2\pi m}(m/e)^m) = \mathcal{O}(m \log m)$$

  watermarking bits.
- What about stealth?

Algorithm WMVVS

p. 506

Watermarks in CFGs



Algorithm WMVVS: Watermarks in CFGs

- Basic idea:
  1. Embed the watermark in the CFG of a function.

Algorithm WMVVS: Watermarks in CFGs

- Basic idea:
  1. Embed the watermark in the CFG of a function.
  2. Tie the CFG tightly to the rest of the program.

Algorithm WMVVS: Watermarks in CFGs

- Basic idea:
  1. Embed the watermark in the CFG of a function.
  2. Tie the CFG tightly to the rest of the program.
- Issues:
  1. How do you encode a number in a CFG?
  2. How do you find the watermark CFG?
  3. How do you attach the watermark CFG to the rest of the program?

Algorithm WMVVS: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two

Algorithm WMVVS: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible

Algorithm WMVVS: Embedding

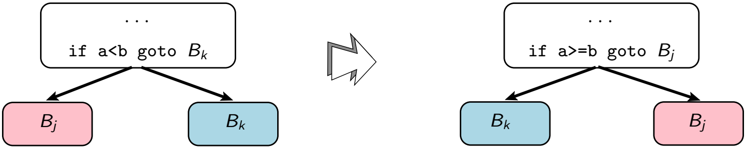
- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn't deeply nested)

Algorithm wmVVS: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn't deeply nested)
  4. it is small (real functions aren't big)

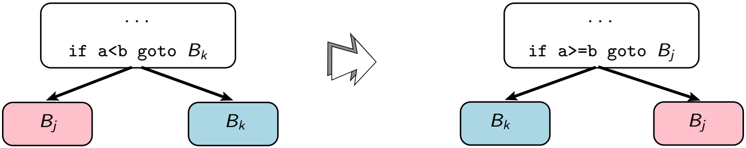
Algorithm WMVVS: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn't deeply nested)
  4. it is small (real functions aren't big)
  5. it is resilient to edge-flips:

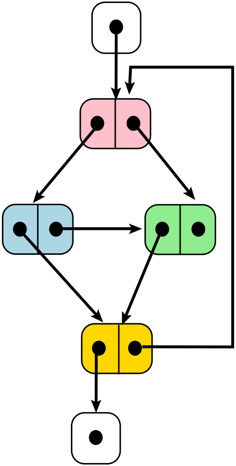


Algorithm wmVVS: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn't deeply nested)
  4. it is small (real functions aren't big)
  5. it is resilient to edge-flips:



- Reducible Permutation Graphs (RPGs)



```
public static int bogus;
public static int m4(int i) {
    i = i & 0x7BFF;
    bogus+=2; i-=i>>2;
    do {
        i = i >> 3;
        label: {
            if (++bogus <= 0) {
                i = i | 0x1000;
                if ((bogus += 6) == 0)
                    break label;
            }
            ++bogus;
            i = i * 88 >>> 1;
        }
        i = i | 0x4;
    } while((bogus += 6)<0);
    bogus+=2; return i;
}
```

```
public void P(boolean S) {
   if (S)
      System.out.println("YES");
   else
      System.out.println("NO");
}

public void main (String args[]) {
   for (int i=1; i<args.length; i++) {
      if (args[0].equals(args[i])) {
         P(true);
         if (m4(3)<0)
         P(false);
         return;
      }
   }
   m3(-1);
   P(false);
}
```

```
public int bogus;
public int m4(int i) {
   i = i & 0x7BFF;
   bogus += 2;
   i -= i >> 2;
   do {
      if (i<-6)
         P(bogus<i);
      i = i >> 3;
      label: {
         if (++bogus <= 0) {
            i = i | 0x1000;
            m3(0);
            if ((bogus+=6)==0)
               break label;
         }
         ++bogus;
         i = i * 88 >>> 1;
      }
      i = i | 0x4;
   } while (((bogus += 6)<0)
            && (m3(9)>=0) )
   bogus += 2;
   return i;
}
```

```
public int m3(int i) {
   i = i ^ i >> 0x1F;
   i = i / 4 * 3;
   do {
      i -= i >> 3;
      if((bogus += 11) <= 0)
         break;
```

Algorithm WMVVS: Recognition

- So, how do you find the watermark CFG among all the “real” CFGs?

Algorithm WMVVS: Recognition

- So, how do you find the watermark CFG among all the "real" CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.

Algorithm WMVVS: Recognition

- So, how do you find the watermark CFG among all the "real" CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.
- Recognition procedure:
  1. compute the mark value for each basic block in the program

Algorithm WMVVS: Recognition

- So, how do you find the watermark CFG among all the “real” CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.
- Recognition procedure:
  1. compute the mark value for each basic block in the program
  2. assume that any function with more than $t\%$ blocks marked is a watermark function

Algorithm WMVVS: Recognition

- So, how do you find the watermark CFG among all the "real" CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.
- Recognition procedure:
  1. compute the mark value for each basic block in the program
  2. assume that any function with more than t% blocks marked is a watermark function
  3. construct CFGs for the watermark functions

Algorithm WMVVS: Recognition

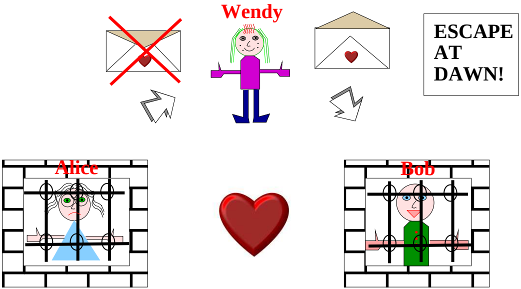
- So, how do you find the watermark CFG among all the "real" CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.
- Recognition procedure:
  1. compute the mark value for each basic block in the program
  2. assume that any function with more than $t\%$ blocks marked is a watermark function
  3. construct CFGs for the watermark functions
  4. decode each one into an integer watermark

Algorithm WMVVS: Recognition

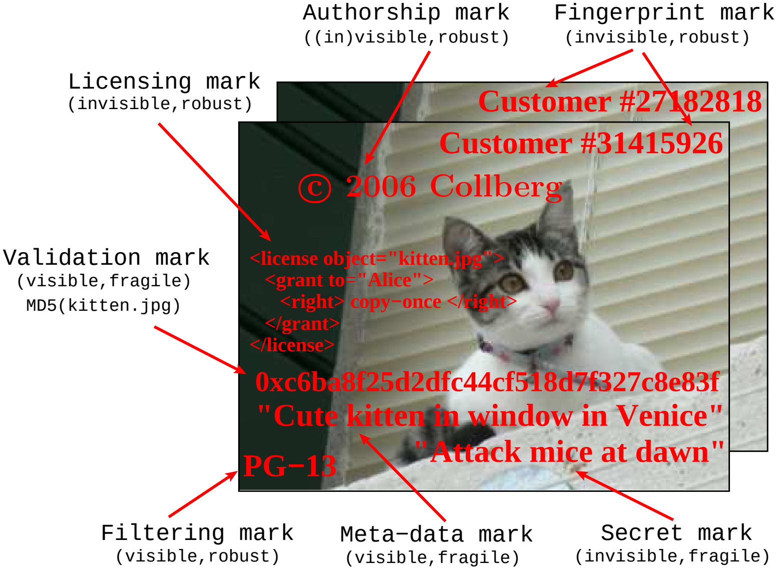
- So, how do you find the watermark CFG among all the "real" CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.
- Recognition procedure:
  1. compute the mark value for each basic block in the program
  2. assume that any function with more than t% blocks marked is a watermark function
  3. construct CFGs for the watermark functions
  4. decode each one into an integer watermark
- The embedder can split the watermarking into pieces, for higher bitrate.

Steganographic Embeddings

p. 522



Steganographic Embeddings



Watermark Embeddings

- Watermarks are
  - short identifiers
  - difficult to locate
  - hard to destroy

Watermark Embeddings

- Watermarks are
  - short identifiers
  - difficult to locate
  - hard to destroy
- The adversary
  - knows that the object is marked
  - knows the algorithm used
  - doesn't know the key
  - is active

Watermark Embeddings

- Watermarks are
  - short identifiers
  - difficult to locate
  - hard to destroy
- The adversary
  - knows that the object is marked
  - knows the algorithm used
  - doesn't know the key
  - is active
- You care about
  - data-rate
  - stealth
  - resilience

Steganographic Embeddings

- Stegomarks are
  - long identifiers
  - difficult to locate

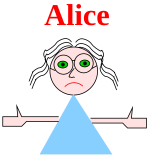
Steganographic Embeddings

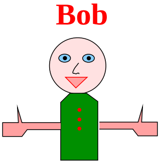
- Stegomarks are
  - long identifiers
  - difficult to locate
- The adversary
  - wants to know if the object is marked
  - knows the algorithm used
  - doesn't know the key
  - is passive

Steganographic Embeddings

- Stegomarks are
  - long identifiers
  - difficult to locate
- The adversary
  - wants to know if the object is marked
  - knows the algorithm used
  - doesn't know the key
  - is passive
- You care about
  - data-rate
  - stealth

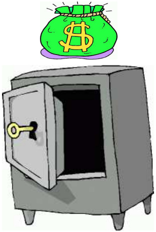
Steganography — Prisoners' Problem

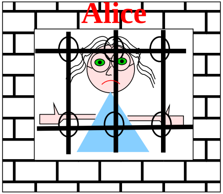


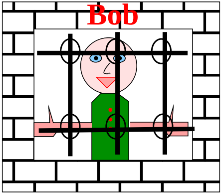


Steganography — Prisoners' Problem

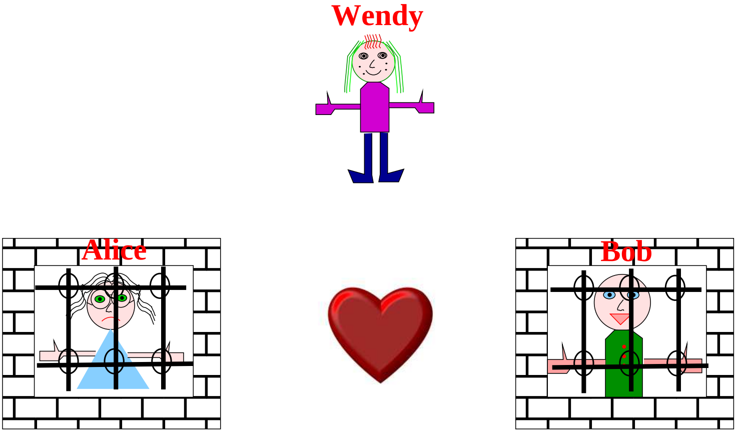
Steganography — Prisoners’ Problem



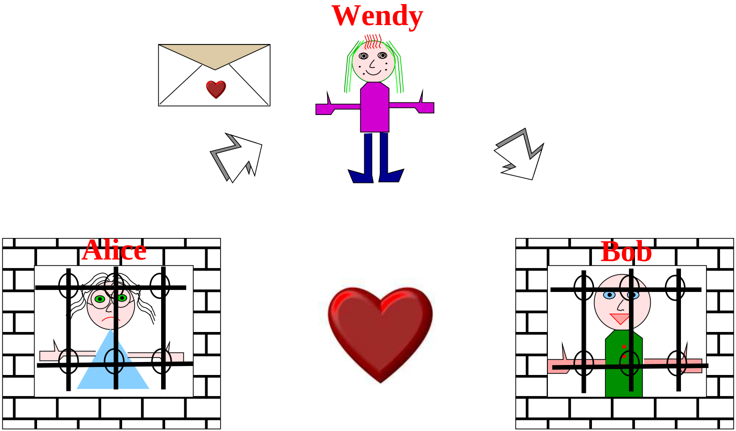




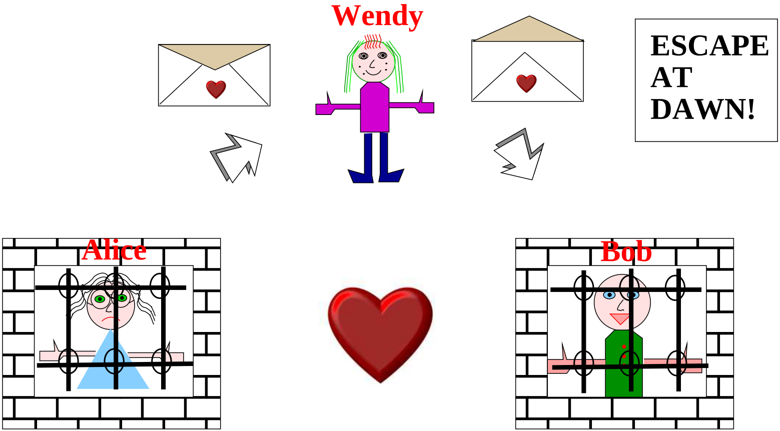
Steganography — Prisoners' Problem



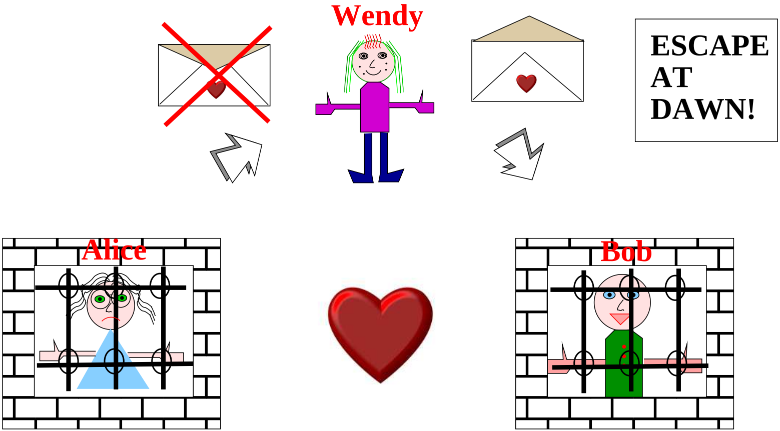
Steganography — Prisoners' Problem



Steganography — Prisoners' Problem



Steganography — Prisoners' Problem



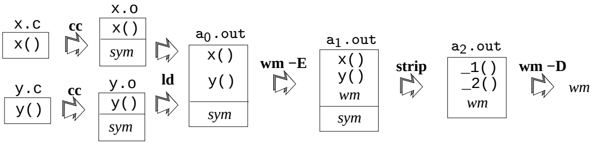
Steganography — Null cipher

```
Easter is soon, dear! So many flowers! Can you
smell them? Are you cold at night? Prison food
stinks! Eat well, still! Are you lonely? The
prison cat is cute! Don't worry! All is well!
Wendy is nice! Need you! ):
```

Algorithm wmASB

p. 523

Hidden Messages in x86 Binaries



WMASB: Hidden Messages in x86 Binaries

- Basic idea: Play compiler!

  whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W*.*

WMASB: Hidden Messages in x86 Binaries

- Basic idea: Play compiler!

  whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W.

- Four sources of ambiguity:
  1. code layout (ordering of chains of basic blocks)

wmASB: Hidden Messages in x86 Binaries

- Basic idea: Play compiler!

  whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W.

- Four sources of ambiguity:
  1. code layout (ordering of chains of basic blocks)
  2. instruction scheduling (instruction order within basic blocks)

wmASB: Hidden Messages in x86 Binaries

- Basic idea: Play compiler!

  whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W.

- Four sources of ambiguity:
  1. code layout (ordering of chains of basic blocks)
  2. instruction scheduling (instruction order within basic blocks)
  3. register allocation

WMASB: Hidden Messages in x86 Binaries

- Basic idea: Play compiler!

  whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W.

- Four sources of ambiguity:
  1. code layout (ordering of chains of basic blocks)
  2. instruction scheduling (instruction order within basic blocks)
  3. register allocation
  4. instruction selection

WMASB: Embedding

1. Construct:
   1. codebook $\mathcal{B}$ of equivalent instruction sequences

`mul` r_i,x,5
`shl` r_i,x,2 `add` r_i,r_i,x
`add` r_i,x,x `add` r_i,r_i,r_i `add` r_i,r_i,x

   2. statistical model $\mathcal{M}$ of real code

WMASB: Embedding

1. Construct:
   1. codebook $\mathcal{B}$ of equivalent instruction sequences

`mul` r_i,`x`,`5`
`shl` r_i,`x`,`2` `add` r_i,r_i,`x`

`add` r_i,`x`,`x`
`add` r_i,r_i,r_i
`add` r_i,r_i,`x`

   2. statistical model $\mathcal{M}$ of real code
2. Encrypt W with *key*.

wmASB: Embedding

1. Construct:
   1. codebook $\mathcal{B}$ of equivalent instruction sequences

`mul` r_i,x,5
`shl` r_i,x,2 `add` r_i,r_i,x
`add` r_i,x,x `add` r_i,r_i,r_i `add` r_i,r_i,x

   2. statistical model $\mathcal{M}$ of real code
2. Encrypt W with *key*.
3. Canonicalize P:
   1. Sort block chains, procedures, modules
   2. Order instructions in each block in standard order
   3. Replace each instruction with the first alternative from $\mathcal{B}$.

WMASB: Embedding

4. Code layout: Embed bits from W by reordering code segments within the executable.

wmASB: Embedding

4. Code layout: Embed bits from W by reordering code segments within the executable.
5. Instruction scheduling:
   1. Build dependency graph
   2. Generate all valid instruction schedules
   3. Embed bits from W by picking a schedule

   Use $\mathcal{M}$ to avoid picking unusual schedules.

WMASB: Embedding

4. Code layout: Embed bits from W by reordering code segments within the executable.
5. Instruction scheduling:
   1. Build dependency graph
   2. Generate all valid instruction schedules
   3. Embed bits from W by picking a schedule

   Use $\mathcal{M}$ to avoid picking unusual schedules.
6. Instruction selection: Use $\mathcal{B}$ to embed bits from W by replacing instructions. Use $\mathcal{M}$ to avoid unusual instruction sequences.

WMASB: Stealth

- Instruction selection:
  - There are 3078 different encodings of three instructions for `EAX=(EAX/2)`!
  - Most don't occur in real code...

wmASB: Stealth

- Instruction selection:
  - There are 3078 different encodings of three instructions for `EAX=(EAX/2)`!
  - Most don't occur in real code...
- Instruction scheduling:
  - Avoid bad schedules: no compiler would generate it!
  - Avoid generating different schedules for two blocks with the same dependency graph!

WMASB: Stealth

- Instruction selection:
  - There are 3078 different encodings of three instructions for `EAX=(EAX/2)`!
  - Most don't occur in real code...
- Instruction scheduling:
  - Avoid bad schedules: no compiler would generate it!
  - Avoid generating different schedules for two blocks with the same dependency graph!
- Code layout:
  - Compilers lay out code for locality: don't deviate too much from that!

wmASB: Stealth

- Encoding rate
  - Unstealthy code: $\frac{1}{27}$
  - Stealthy: $\frac{1}{89}$.

WMASB: Stealth

- Encoding rate
  - Unstealthy code: $\frac{1}{27}$
  - Stealthy: $\frac{1}{89}$.
- Encoding space:
  - 58% from code layout
  - 25% from instruction scheduling
  - 17% from instruction selection

WMASB: Stealth

- Encoding rate
  - Unstealthy code: $\frac{1}{27}$
  - Stealthy: $\frac{1}{89}$.
- Encoding space:
  - 58% from code layout
  - 25% from instruction scheduling
  - 17% from instruction selection
- Real code doesn't use unusual instruction sequences.
- Real code contains many schedules for the same dependency graph

Wanna design a watermarking algorithm?

- Find a language structure into which to encode the mark (CFGs, threads, dynamic control flow. . .)

$\langle$*language structure*, *encoder*/*decoder*, *tracer*/*locator*, *embedder*/*extractor*, *attacker*/*protector*$\rangle$

Wanna design a watermarking algorithm?

- Find a language structure into which to encode the mark (CFGs, threads, dynamic control flow...)
- Construct an encoder/decoder (number$\leftrightarrow$CFG,...)

$\langle$*language structure*, *encoder*/*decoder*, *tracer*/*locator*, *embedder*/*extractor*, *attacker*/*protector*$\rangle$

Wanna design a watermarking algorithm?

- Find a language structure into which to encode the mark (CFGs, threads, dynamic control flow. . .)
- Construct an encoder/decoder (number↔CFG,. . .)
- Construct a tracer/locater to find locations for the mark (using key, every function, . . .)

$\langle$*language structure*, *encoder*/*decoder*, *tracer*/*locator*, *embedder*/*extractor*, *attacker*/*protector*$\rangle$

Wanna design a watermarking algorithm?

- Find a language structure into which to encode the mark (CFGs, threads, dynamic control flow...)
- Construct an encoder/decoder (number$\leftrightarrow$CFG,...)
- Construct a tracer/locater to find locations for the mark (using key, every function, ...)
- Construct a embedder/extractor to tie the mark to surrounding code

$\langle$*language structure*, *encoder/decoder*, *tracer/locator*, *embedder/extractor*, *attacker/protector*$\rangle$

Wanna design a watermarking algorithm?

- Find a language structure into which to encode the mark (CFGs, threads, dynamic control flow. . .)
- Construct an encoder/decoder (number↔CFG,. . .)
- Construct a tracer/locater to find locations for the mark (using key, every function, . . .)
- Construct a embedder/extractor to tie the mark to surrounding code
- Decide on an attack model.

$\langle$*language structure*, *encoder*/*decoder*, *tracer*/*locator*, *embedder*/*extractor*, *attacker*/*protector*$\rangle$