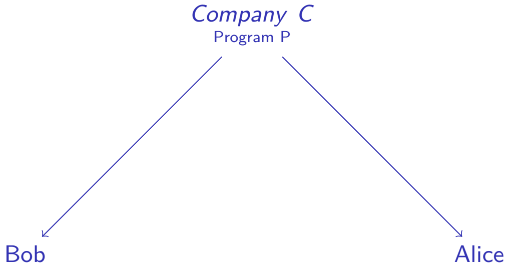
Cryptographic Program Watermarking

Julien Braine, *ENS Lyon*
2015 Internship with
Ron Steinfeld, *Faculty of IT*

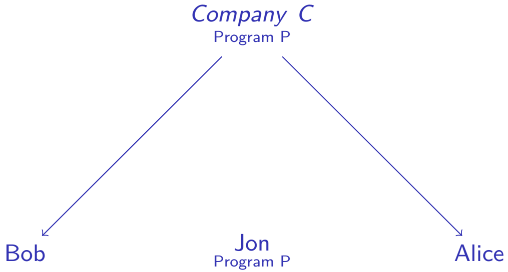
17 August 2015

What is watermarking ?



Protecting Authorship

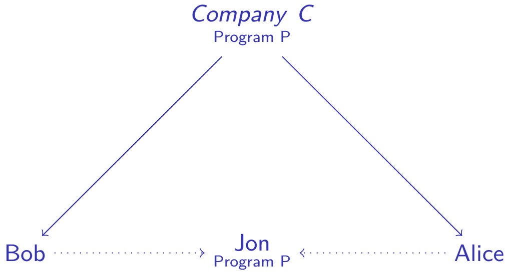
What is watermarking ?



Protecting Authorship

Prove that Jon's P was produced by C

What is watermarking ?



Protecting Authorship

Prove that Jon's P was produced by C

Bonus : who gave P to Jon ?

Embedding a mark

To protect authorship, the company C will add a mark, to P indicating that it is their program.

Figure: Image watermarking [PMA11]

Mark properties

- Must not alter the program's functionality
- Must be hard to remove
- The presence of a mark is deliberate

Program Watermarking

By changing the implementation [IN10].

Simple examples :

1. Change order creation of variables
2. Change instructions to equivalent ones
   $x = x + 1 \Leftrightarrow x = x - (-1)$

Issues

- Hard to do : decompilers
- No formal security proof framework
- There might be a general attack [GGH$^+$13]

Indistinguishability Obfuscation

Indistinguishability Definition

We say that distributions D_1 and D_2 are indistinguishable if for all polynomial time adversary $A, |Pr(A(D_1) = 1) - Pr(A(D_2) = 1)| \simeq 0$.

Indistinguishability Obfuscation Definition [BGI+01]

iO is an indistinguishability obfuscator* if $\forall C, C'$ $(\forall x, C(x) = C'(x)) \Rightarrow iO(C)$ indistinguishable of $iO(C')$

Consequence

- Implementation changes only can not work
- Functionality changes needed, but must be invisible

A brief History of IO

IO definition proposal.
First impossibility result on software watermarking
[BGI+01] 2001

Nothing happens.
Implementation Watermarking continues.
[IN10] 2010

First IO construction proposal.
[GGH+13] 2013

An IO era of cryptography.
2013 - Now

IO for watermarking starts.
[CHV15] [NW15]
2015 - Now

Watermarking definition [CHV15]

Let $\mathbb{C}$ be set of circuits we want to watermark or hide watermarked circuits in. $Scheme = (Setup, Mark, Verify)$

- **Functionality preservation**: $Pr(Mark(C) \equiv C | C \hookleftarrow U(\mathbb{C})) \simeq 1.$
- **Correctness**: $Pr(Verify(Mark(C)) = 1 | C \hookleftarrow U(\mathbb{C})) \simeq 1.$
- **Unremovability**: $\forall A, Pr(A(C') \equiv C \text{ and } Verify(A(C')) = 0 | C' = Mark(C), C \hookleftarrow U(\mathbb{C})) \simeq 0.$
- **Meaningfulness**: most circuits are unmarked or marked circuits can not be forged

Remarks

1. $\mathbb{C}$ is given $\Rightarrow$ we are hiding the circuit within $\mathbb{C}$
2. Properties satisfied for an average $C \neq$ for all C

Table of contents

Construction

```
Setup()
  x* ↩ D(I)
  y* ↩ D′(O)
```

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Verify(C) = return C(x*)==y*
```

Properties

- Have : functionality, correctness and meaningfulness.
- Need : Unremovability

Input and Output Distribution Attack

```
Setup()
  x* ↩ D(I)
  y* ↩ D'(O)
```

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Verify(C) = return C(x*)==y*
```

```
UnMark(C) =
function :
  x  → ⊥ when D(x) big
| x → ⊥ when D'(C(x)) ≠ ℂ(I)
| x -> C(x) in other cases
);
```

Conclusion

- Pick x^* uniformly in I $\Rightarrow$ Have Sampler in I
- Pick y^* uniformly in $\mathbb{C}$(I) $\Rightarrow$ Have Sampler in $\mathbb{C}$(I)

The property Attack

Assume circuits in $\mathbb{C}$ have a property.
For example $C(x+1) = xC(x)$.

```
Setup()
  x* ←↩ D(I)
  y* ←↩ D'(O)
```

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Verify(C) = return C(x*)==y*
```

```
UnMark(C) =

function :
  x → ⊥ when C(x+1) ≠ xC(x)
| x → C(x) otherwise
);
```

Conclusion

- Properties $\Rightarrow$ hard to watermark
- Sets of circuits with no properties ?

PPRFs

Definition [BW13]

A set $\mathbb{C} = \{C_k\}$ is a PPRF set iff

- $\mathbb{C}$ is a PRF :
  $\forall PPT\ A, |Pr(A^{C_k}() = 1 | C_k \hookleftarrow \mathbb{C}) - Pr(A^{C}() = 1 | C \hookleftarrow U(O^I))| \simeq 0.$

- $\mathbb{C}$ is puncturable (resistant to property attacks) :
  $\forall C, \forall x^*, \exists$ punctured key k'
  - Given only k' and x we can calculate $C(x)$ for any $x \neq x^*$
  - Given k', $C(x^*)$ is indistinguishable from uniform.

Proof of Unremovability for PPRFs

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

Proof by contradiction [NW15].

1. Remover R for our scheme.

Proof of Unremovability for PPRFs

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

Proof by contradiction [NW15].

1. Remover R for our scheme.
2. Distinguisher for $(Mark(C), x^*)$ and $(Mark(C), U(I))$

Proof of Unremovability for PPRFs

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Mark^(1)(C) = return iO (
function :
  x* → y*
  x → P_x*(C)(x) when x ≠ x*
);
```

Proof by contradiction [NW15].

1. Remover R for our scheme.
2. Distinguisher for $(Mark(C), x^*)$ and $(Mark(C), U(I))$
3. Distinguisher for $(Mark^{(1)}(C), x^*)$ and $(Mark^{(1)}(C), U(I))$

Proof of Unremovability for PPRFs

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Mark^(1)(C) = return iO (
function :
  x* → y*
  x → P_x*(C)(x) when x ≠ x*
);
```

```
Mark^(2)(C) = return iO (
function :
  x* → C(x*)
  x → P_x*(C)(x) when x ≠ x*
);
```

Proof by contradiction [NW15].

1. Remover R for our scheme.
2. Distinguisher for $(Mark(C), x^*)$ and $(Mark(C), U(I))$
3. Distinguisher for $(Mark^{(1)}(C), x^*)$ and $(Mark^{(1)}(C), U(I))$
4. Distinguisher for $(Mark^{(2)}(C), x^*)$ and $(Mark^{(2)}(C), U(I))$

Proof of Unremovability for PPRFs

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Mark^(1)(C) = return iO (
function :
  x* → y*
  x → P_x*(C)(x) when x ≠ x*
);
```

```
Mark^(2)(C) = return iO (
function :
  x* → C(x*)
  x → P_x*(C)(x) when x ≠ x*
);
```

```
Mark^(3)(C) = return iO (C);
```

Proof by contradiction [NW15].

1. Remover R for our scheme.
2. Distinguisher for $(Mark(C), x^*)$ and $(Mark(C), U(I))$
3. Distinguisher for $(Mark^{(1)}(C), x^*)$ and $(Mark^{(1)}(C), U(I))$
4. Distinguisher for $(Mark^{(2)}(C), x^*)$ and $(Mark^{(2)}(C), U(I))$
5. Distinguisher for $(Mark^{(3)}(C), x^*)$ and $(Mark^{(3)}(C), U(I))$

Proof of Unremovability for PPRFs

```
Mark(C) = return iO (
function :
  x* → y*
  x → C(x) when x ≠ x*
);
```

```
Mark^(1)(C) = return iO (
function :
  x* → y*
  x → P_x*(C)(x) when x ≠ x*
);
```

```
Mark^(2)(C) = return iO (
function :
  x* → C(x*)
  x → P_x*(C)(x) when x ≠ x*
);
```

```
Mark^(3)(C) = return iO (C);
```

Proof by contradiction [NW15].

1. Remover R for our scheme.
2. Distinguisher for $(Mark(C), x^*)$ and $(Mark(C), U(I))$
3. Distinguisher for $(Mark^{(1)}(C), x^*)$ and $(Mark^{(1)}(C), U(I))$
4. Distinguisher for $(Mark^{(2)}(C), x^*)$ and $(Mark^{(2)}(C), U(I))$
5. Distinguisher for $(Mark^{(3)}(C), x^*)$ and $(Mark^{(3)}(C), U(I))$
6. $Mark^{(3)}$ does not depend on x^*. Absurd.

Additional goals

1. Watermarking with a message $\Rightarrow$ [NW15]
2. Protection against chosen Watermark attacks $\Rightarrow$ [CHV15] [NW15]
3. Protection against partial functionality change $\Rightarrow$ [NW15]
4. Public verification $\Rightarrow$ [CHV15] [NW15]
5. Collusion resistant verification $\Rightarrow$ Done
6. Collusion resistant message extraction $\Rightarrow$ Seems achievable [BS96] in $O(users^4)$

Traitor Tracing

Company
Broadcast signal

Bob Alice

Assumptions

- The broadcast signal is encrypted
- The decrypt function is a marked PPRF

Protection

Traitor Tracing

Company
Broadcast signal

Bob · Jon (read signal) · Alice

Assumptions

- The broadcast signal is encrypted
- The decrypt function is a marked PPRF

Protection

Traitor Tracing

Assumptions

- The broadcast signal is encrypted
- The decrypt function is a marked PPRF

Protection

Jon can not read the signal without a marked decrypt function

Traitor Tracing

Assumptions

- The broadcast signal is encrypted
- The decrypt function is a marked PPRF

Protection

Jon can not read the signal without a marked decrypt function

Issue

Already been done more efficiently using IO only [BZ14].

Can we watermark usual circuits ?

Reminder
Properties $\Rightarrow$ hard to watermark.

Intuition
Usual circuits $\Rightarrow$ not watermarkable

Goal
Prove that only cryptographic sets of circuits are watermarkable

Idea of proof

How can we define cryptographic sets

Let $F : (C, E = (x_1, ...x_n)) \rightarrow ((C(x_1), ..., C(x_n)), E)$ for E of polynomial size (so that F is efficiently computable). A set $\mathbb{C}$ is cryptographic iff
$\forall A, \exists E, Pr(F(A(F(C, E))) = F(C, E) | C \hookleftarrow \mathbb{C}) \simeq 0$

Steps of the proof

Non cryptographic $\Rightarrow$ Learner $\Rightarrow$ Remover.

Learner $\Rightarrow$ Remover

Learner definition

L is a learner if $Pr(L^C() \equiv C) > 0$

Proof

Idea: Apply the learner to the mark circuit and we get the original one which is unmarked.

Problem: The learner might not work as it might query modified points.

Case investigation

- If the position of the modified values x^* have enough entropy $\Rightarrow$ probability that the learner queries at those points $\simeq 0$. Therefore, just applying L removes the mark.
- If all the modified values are always at the same spot $\Rightarrow$ find those spots, change them and remove the mark.
- If we have a mix of both $\Rightarrow$ problem as the value with entropy requires us to use the learner and the second might stop the learner from working

  $\Rightarrow$ Additional hypothesis : the learner has enough entropy in the values it queries.

Non cryptographic to learning

A first Idea

Let f^{-1} be a computable inverter of
$F : (C, E = (x_1, ...x_n)) \to ((C(x_1), ..., C(x_n)), E)$.

```
L^C() =
   (x_1, ..., x_n) <-- U(I^n)
   return Circuit(f^{-1}((C(x_1), ..., C(x_n)), (x_1, ...x_n)));
```

Does this really learn ?

Consider $\mathbb{C} = \{C_\alpha\}$ with $C_\alpha(\alpha) = 1, C_\alpha(x) = 0$. The pre-image is huge so that technique seem to fail, yet, it does not, it just gives us an approximate learner.

Another approach to learning

$$\begin{tikzcd}
 & & & \mathbb{C}, x_a \arrow[dll] \arrow[drr] & & & \\
 & \mathbb{C}_{x_a=0}, x_b \arrow[dl, dotted] \arrow[dr, dotted] & & & & \mathbb{C}_{x_a=1}, x_c \arrow[dl, dotted] \arrow[dr, dotted] & \\
f^{-1}((0,0,\ldots),(x_a,x_b,\ldots)) & & f^{-1}((0,1,\ldots),(x_a,x_b,\ldots)) & & f^{-1}((1,0,\ldots),(x_a,x_c,\ldots)) & & f^{-1}((1,1,\ldots),(x_a,x_c,\ldots))
\end{tikzcd}$$

How to choose the x's

We would like each x to split each set $\mathbb{C}_{x_i=y_i,\ldots,x_j=y_j}$ into two sets of similar size. That would guarantee that the algorithm is efficient.

Existence of such x

$$\begin{tikzcd}
& & & \mathbb{C}, x_a \arrow[dll] \arrow[drr] & & & \\
& \mathbb{C}_{x_a=0}, x_b \arrow[dl, dotted] \arrow[dr, dotted] & & & & \mathbb{C}_{x_a=1}, x_c \arrow[dl, dotted] \arrow[dr, dotted] & \\
f^{-1}((0,0,\ldots),(x_a,x_b,\ldots)) & & f^{-1}((0,1,\ldots),(x_a,x_b,\ldots)) & & f^{-1}((1,0,\ldots),(x_a,x_c,\ldots)) & & f^{-1}((1,1,\ldots),(x_a,x_c,\ldots))
\end{tikzcd}$$

When we can not split a set $\mathbb{C}_{x_i=y_i,\ldots,x_j=y_j}$ into two sets of similar size, we reach a leaf of the tree and return f^{-1}.
For all x, we have an overwhelming set and a negligible set.
Therefore, all the circuits in it are similar to $Perfect(x)$, the circuit which on input x agrees with the majority.

Improving the proof

Problem with the current proof

We need to find those x's !

Solution

Instead of needing the best x, just estimate a good x (we use the standard average estimator). Therefore, we pick x's at random and take the one that splits best. This requires a sampler in each of the sets $\mathbb{C}_{x_i=y_i,\ldots,x_j=y_j}$.

Even better

Instead of estimating the best x, we can split for all those x's and one of them should be good enough. And we are back to our initial solution.

```
L^C() =
    (x_1, ..., x_n) ↩ U(I^n)
    return Circuit(f^{-1}((C(x_1), ..., C(x_n)), (x_1, ...x_n)));
```

$$L^C() = (x_1, ..., x_n) \hookleftarrow U(I^n) \quad \textbf{return } \mathrm{Circuit}(f^{-1}((C(x_1), ..., C(x_n)), (x_1, ...x_n)));$$

Except that now we have proven that it works (and we also have information on the value of n and how each parameter changes in function of one another).

Conclusion

We created a Learner with huge entropy, we thus have a remover and non cryptographic functions are not watermarkable.

Further research on cryptographic program watermarking

The result

Cryptographic program watermarking does not seem all that promising given that:

- We have not found any cryptographic application that is not already done with iO
- We have proven that we can not watermark non cryptographic functions

Research paths

- Relax or change definition
- Find cryptographic uses
- Apply those results to image watermarking

References

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. 2001.

[BS96] Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data. 1996.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. 2014.

[CHV15] Aloni Cohen, Justin Holmgren, and Vinod Vaikuntanathan. Publicly verifiable software watermarking. 2015.

[GGH+13] Shelly Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Anant Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. 2013.

[IN10] A.N. Fionov I.V. Nechta. Digital watermarks for c/c++ programs. 2010.

[NW15] Ryo Nishimaki and Daniel Wichs. Watermarking cryptographic programs against arbitrary removal strategies. 2015.

[PMA11] Ante Poljicak, Lidija Mandic, and Darko Agic. Discrete fourier transform–based watermarking method with an optimal implementation radius. 2011.

Thank you for your attention !

Any Questions ?