On Low-End Obfuscation and Learning

Gal Yehuda

Joint work with:
Elette Boyle, Yuval Ishai, Pierre Meyer and Robert Robere

ITCS 2023

Program Obfuscation

Intuitive goal: hide the implementation of a program (while preserving functionality).

Program Obfuscation

Intuitive goal: hide the implementation of a program (while preserving functionality).

```
int main(){int o_8085b117358aff981ff7326b8ed8d898=(0x0000000000000002 +
    0x0000000000000201 + 0x0000000000000801 - 0x0000000000000A03
    ),o_dfe2821c09b6ce01a45ec650a4e9269e=(0x0000000000000002 + 0x0000000000000201
    + 0x0000000000000801 - 0x0000000000000A03);int
    o_18bfd09edb071abc956dcd4282d09cd1=(0x000000000000005C + 0x000000000000022E +
    0x000000000000082E - 0x0000000000000A8A);for (int
    o_9d8e9c181edb72210dfc090ec6a36c92=(0x0000000000000006 + 0x0000000000000203 +
    0x0000000000000803 - 0x0000000000000A09);(o_9d8e9c181edb72210dfc090ec6a36c92
    <= o_18bfd09edb071abc956dcd4282d09cd1) & !!
    (o_9d8e9c181edb72210dfc090ec6a36c92 <= o_18bfd09edb071abc956dcd4282d09cd1
    );++o_9d8e9c181edb72210dfc090ec6a36c92){int
    o_9dafc46e75becd94b4bc6cd3d5f4a237=o_8085b117358aff981ff7326b8ed8d898 +
    o_dfe2821c09b6ce01a45ec650a4e9269e;std::cout<<"\x46""("
    <<o_9d8e9c181edb72210dfc090ec6a36c92<<"\x29"" \075 "
    <<o_9dafc46e75becd94b4bc6cd3d5f4a237<<std::endl
    ;o_8085b117358aff981ff7326b8ed8d898 = o_dfe2821c09b6ce01a45ec650a4e9269e
    ;o_dfe2821c09b6ce01a45ec650a4e9269e = o_9dafc46e75becd94b4bc6cd3d5f4a237;}
    ;return (0x0000000000000000 + 0x0000000000000200 + 0x0000000000000800 -
    0x0000000000000A00);};
```

Program Obfuscation

Intuitive goal: hide the implementation of a program (while preserving functionality).

```
int main()
{
    int a = 1, b = 1;
    int target = 46;
    for(int n = 3; n <= target; ++n)
    {
        int fib = a + b;
        std::cout << "F("<< n << ") = " << fib << std::endl;
        a = b;
        b = fib;
    }

    return 0;
}
```

Program Obfuscation

And a less successful attempt:

YG

Can you obfuscate the following C++ code: #include <iostream>

```
int main()
{
    unsigned int a = 1, b = 1;
    unsigned int target = 46;
    for(unsigned int n = 3; n <= target; ++n)
    {
        unsigned int fib = a + b;
        std::cout << "F("<< n << ") = " << fib << std::endl;
        a = b;
        b = fib;
    }

    return 0;
}
```

Program Obfuscation

And a less successful attempt:

Certainly! Here's an obfuscated version of the code:

c++ Copy code

```c++
#include <iostream>

int main()
{
    int a[] = {2,6,17,58,197,676,2315,7936,27131,92750,316229};
    int b[] = {4,12,34,116,396,1364,4682,15972,54622,186056,632452};
    int target = 46;
    for(int n = 0; n <= target-3; ++n)
    {
        std::cout << "F("<< n+3 << ") = " << b[n] << std::endl;
    }

    return 0;
}
```

Program Obfuscation

Ideally: the obfuscated code doesn’t give more information than a black box access (“Virtual Black Box”).

Program Obfuscation

Ideally: the obfuscated code doesn’t give more information than a black box access (“Virtual Black Box”).

Unfortunately, VBB is impossible [BGIRSVY01]

Indistinguishability Obfuscation

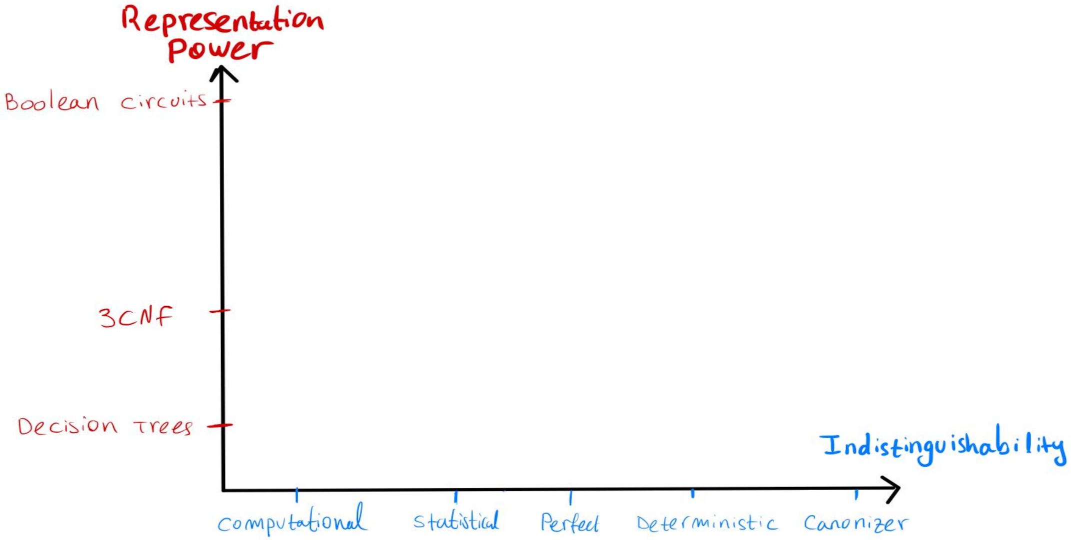
Definition (informal):

A probabilistic polynomial time algorithm iO is an *indistinguishability obfuscator* , if for all pairs of "programs" C_1, C_2 satisfying $C_1 \equiv C_2$ and $|C_1| = |C_2|$ we have that $\mathrm{iO}(C_1) \sim \mathrm{iO}(C_2)$.

Indistinguishability Obfuscation

Definition (informal):

A probabilistic polynomial time algorithm iO is an *indistinguishability obfuscator* , if for all pairs of "programs" C_1, C_2 satisfying $C_1 \equiv C_2$ and $|C_1| = |C_2|$ we have that $\mathrm{iO}(C_1) \sim \mathrm{iO}(C_2)$.



Indistinguishability Obfuscation

Definition (informal):

A probabilistic polynomial time algorithm iO is an *indistinguishability obfuscator* , if for all pairs of "programs" C_1, C_2 satisfying $C_1 \equiv C_2$ and $|C_1| = |C_2|$ we have that $\mathrm{iO}(C_1) \sim \mathrm{iO}(C_2)$.

Indistinguishability Obfuscation

Indistinguishability:

Two distribution ensembles $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are (T,ϵ)-indistinguishable, for $T\colon \mathbb{N}\to\mathbb{N}$ and $\epsilon\colon \mathbb{N}\to[0,1]$, if for every circuit family A_λ of size $T(\lambda)$ and all sufficiently large λ it holds that:

$$\left| \Pr_{x \xleftarrow{\$} X_\lambda}[A_\lambda(x)=1] - \Pr_{y \xleftarrow{\$} Y_\lambda}[A_\lambda(y)=1] \right| \le \epsilon(\lambda).$$

We say that X and Y are:

- *computationally indistinguishable* if they are $(T,1/T)$-indistinguishable for every polynomial T;
- *statistically indistinguishable* if they are $(T',1/T)$-indistinguishable for every polynomial T and arbitrary T';
- *perfectly indistinguishable* if they are identically distributed, namely $X_\lambda \equiv Y_\lambda$ for all λ.

Applications

Deniable encryption [SW13]

Optimal MPC [GP15]

Hardness of finding a Nash equilibrium [BPR15]

Does (computational) iO exist?

Algorithmica

P = NP

✓

Idea: pick the lexicographically first equivalent program

Does (computational) iO exist?

Algorithmica	Heuristica
P = NP	NP is worst case hard, but easy on average
✓	✗ [KMNPRY 14]

Idea: iO + NP $\not\subset$ i.o BPP => OWF

Does (computational) iO exist?

Algorithmica	Heuristica	Pessiland
P = NP	NP is worst case hard, but easy on average	NP is hard on average, but NO OWF
✓	✗	✗
	[KMNPRY 14]	[KMNPRY 14]

Idea: iO + NP $\not\subset$ i.o BPP => OWF

Does (computational) iO exist?

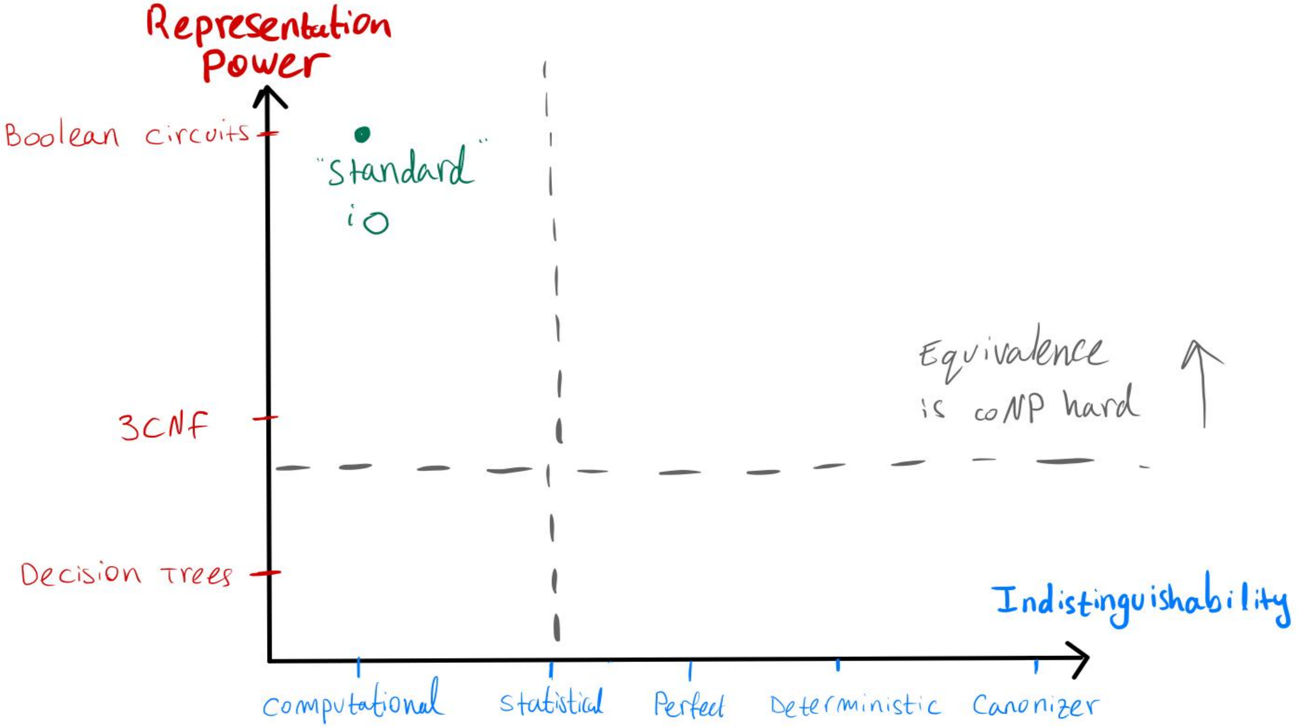
Algorithmica	Heuristica	Pessiland	Minicrypt
P = NP	NP is worst case hard, but easy on average	NP is hard on average, but NO OWF	OWFs exist, but no PKE
✓	✗	✗	✗
	[KMNPRY 14]	[KMNPRY 14]	[SW13]

Idea: iO + OWF => PKE

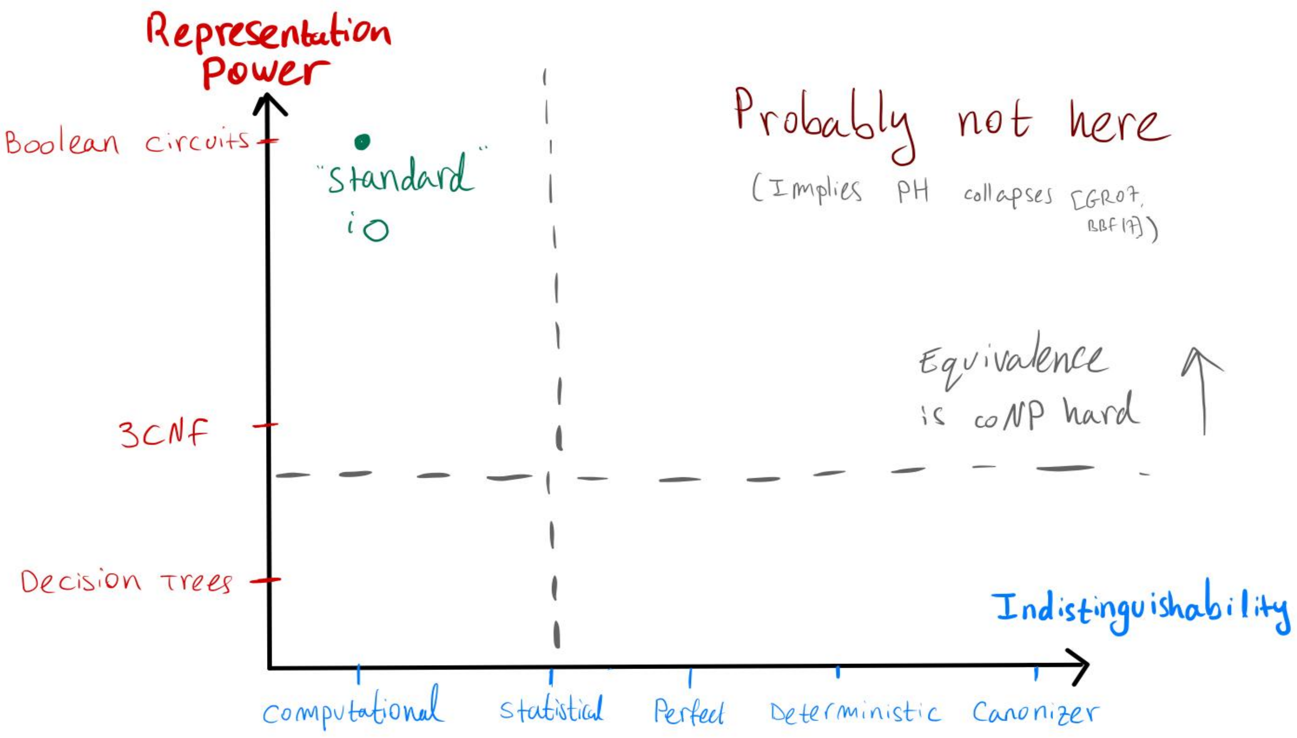
Does (computational) iO exist?

Algorithmica	Heuristica	Pessiland	Minicrypt	Cryptomania++
P = NP	NP is worst case hard, but easy on average	NP is hard on average, but NO OWF	OWFs exist, but no PKE	PKE Obfuscation
✓	✗	✗	✗	✓
	[KMNPRY14]	[KMNPRY14]	[SW13]	[JLS21,22]

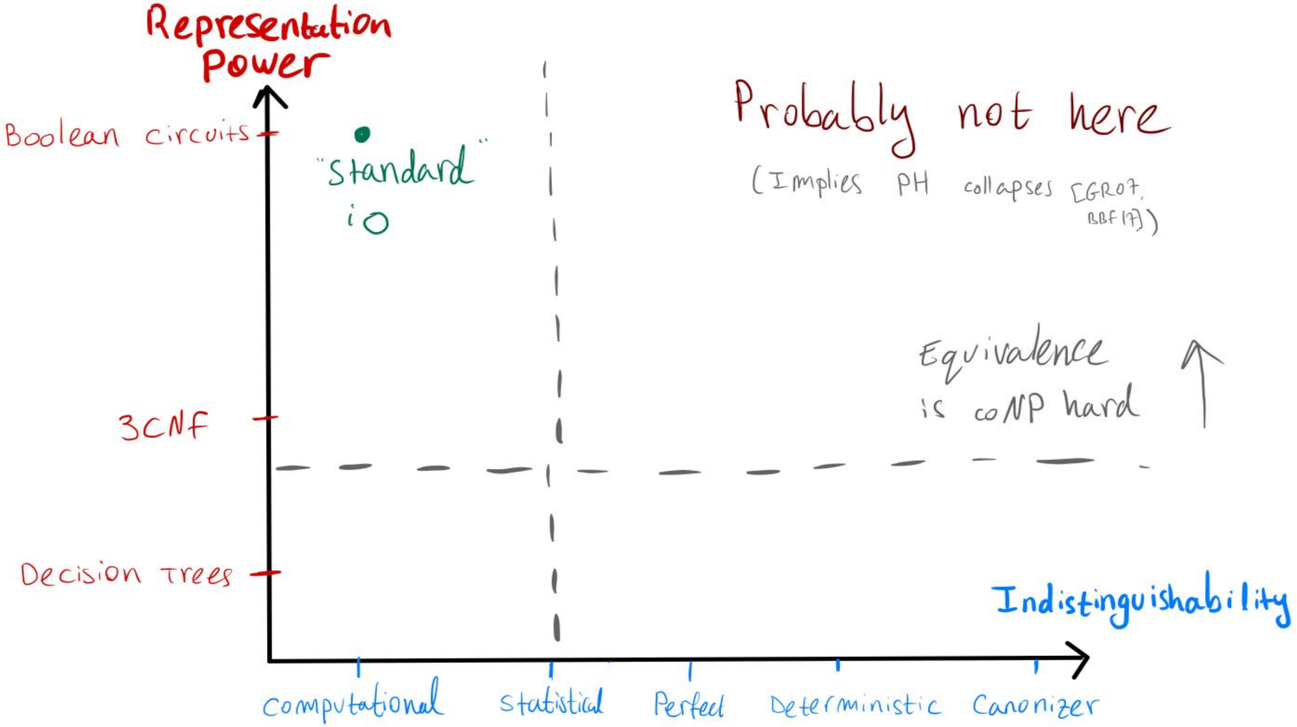
Does statistical iO exist?



Does statistical iO exist?

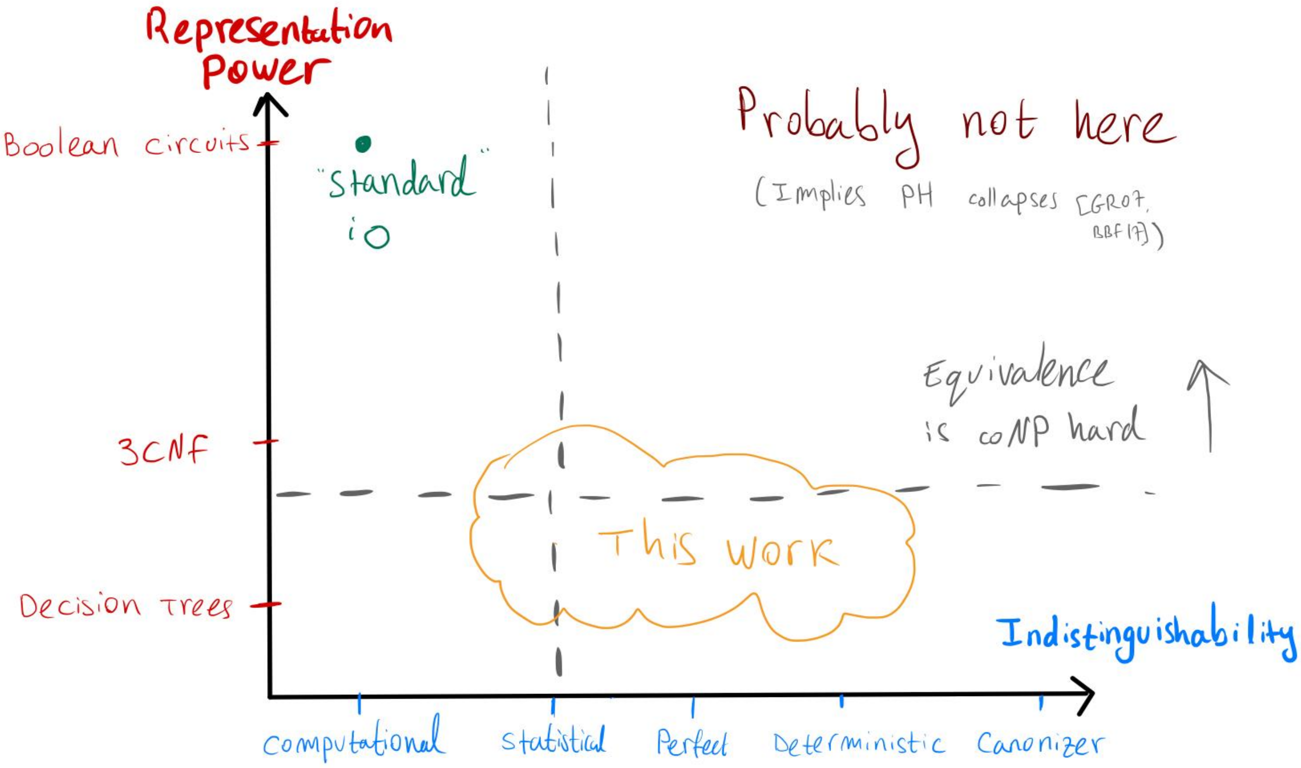


Does statistical iO exist?



Idea: use the randomness of the iO as a short proof for equivalence

This work: “Low-End” Obfuscation



This work: “Low-End” Obfuscation

If we want to achieve *information-theoretic* security, we must settle for weaker models of computation.

Weak classes
Strong guarantees

vs.

Strong classes
weak guarantees

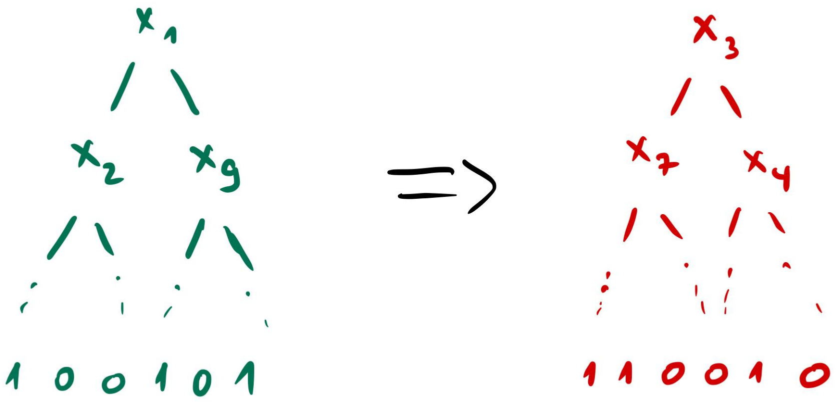
Low end

High end

“Low-End” Obfuscation - why?

If we want to achieve *information-theoretic* security, we must settle for weaker models of computation.

For example: can we obfuscate decision trees?



“Low-End” Obfuscation - why?

If we only want to obfuscate “simple” functions, can we preserve the representation class of the obfuscated program? (*Proper Obfuscation*)

For example: can we obfuscate 3CNFs by 3CNFs?

“Low-End” Obfuscation - why?

We can always apply “high end” obfuscation.

“Low-End” Obfuscation - why?

We can always apply “high end” obfuscation.

The right question: why obfuscating a decision tree by a circuit?

“Low-End” Obfuscation - why?

We can always apply “high end” obfuscation.

The right question: why obfuscating a decision tree by a circuit?

Simple representations are often desired in practice: in the learning literature, the difference between “proper” and “improper” algorithms is very common.

Why not here?

“Low-End” Obfuscation - why?

We can always apply “high end” obfuscation.

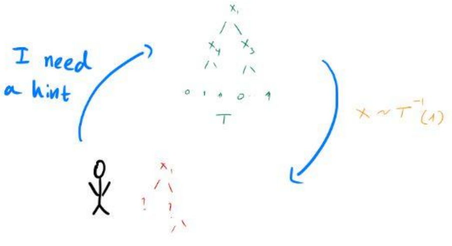
If we restrict to “weak” program classes, we can get stronger guarantees using simpler algorithms.

E.g. OBDDs [GR17]

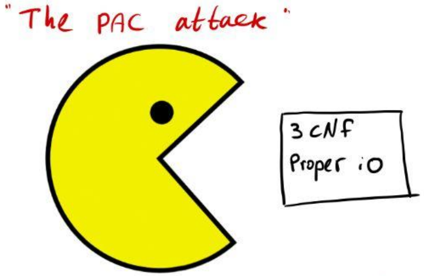
(more on that later)

“Low-End” Obfuscation: our results

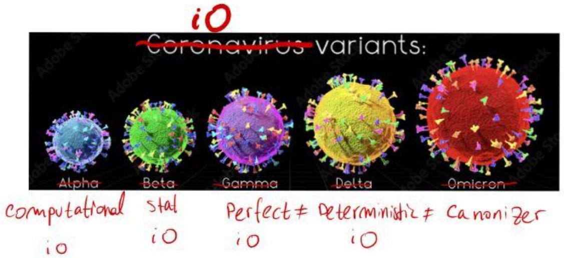
High level summary of our results:



Positive results
"white box" learning



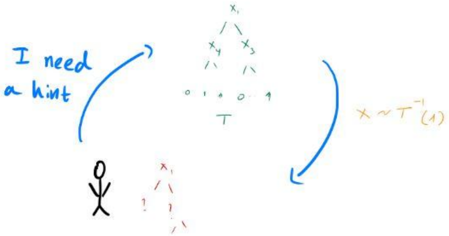
Negative results
PAC-learning



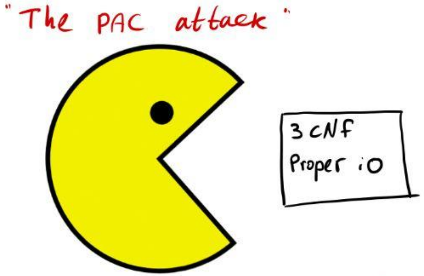
Separating variants
of information
theoretic iO

"Low-End" Obfuscation: our results

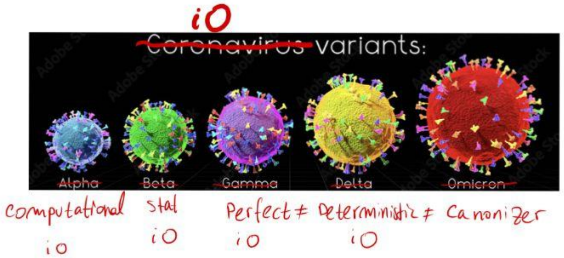
High level summary of our results:



Positive results
"white box" learning



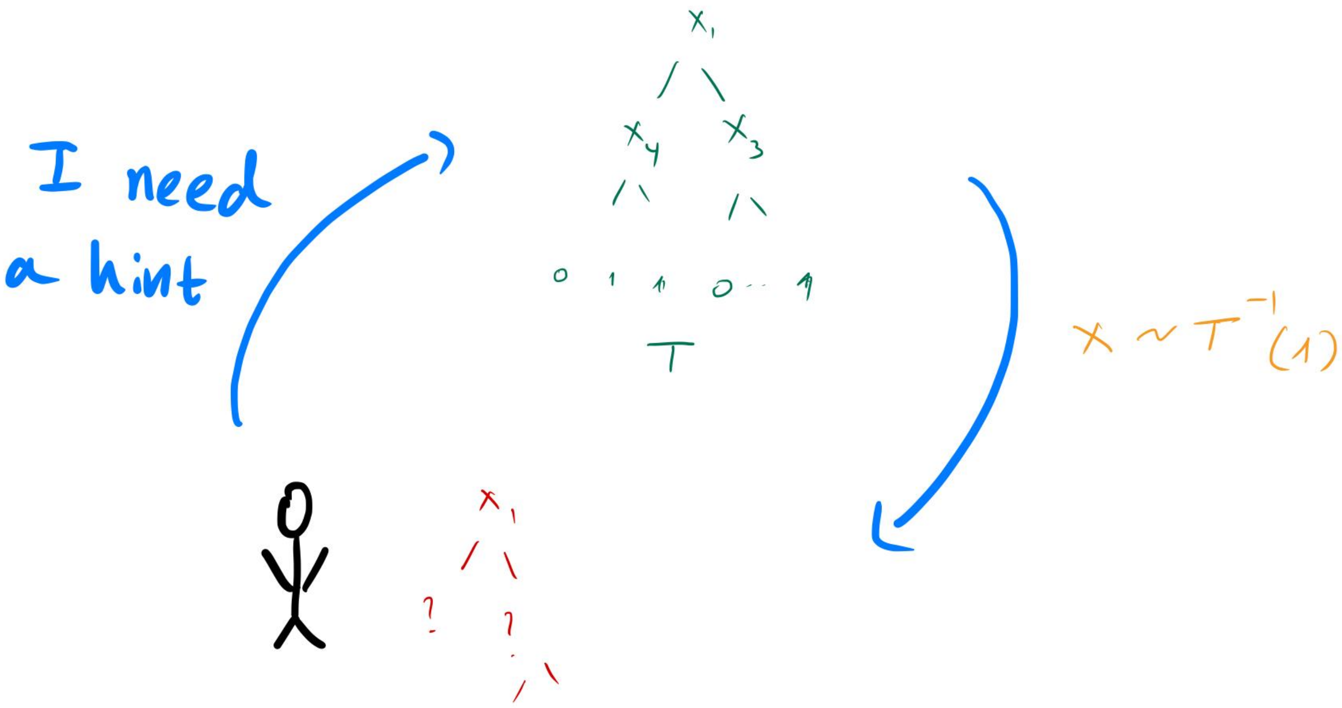
Negative results
PAC-learning



Separating variants
of information
theoretic iO

Learning is used to derive both
negative and positive results!

Result 1: Positive results via “white box” learning

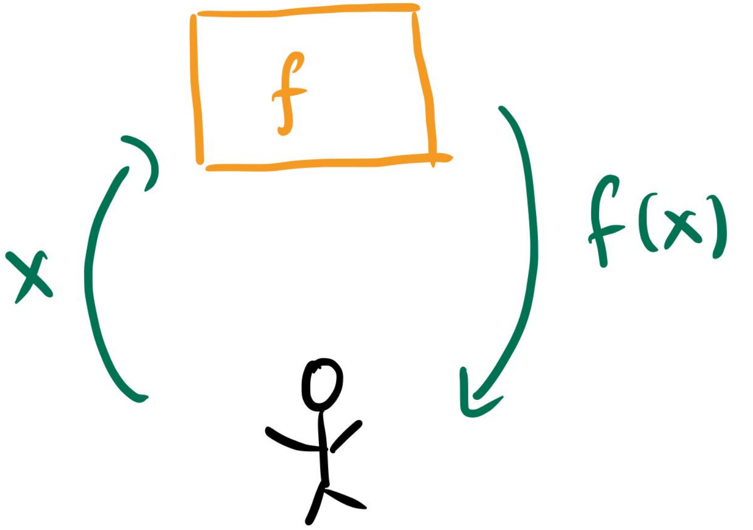


Exact learning implies (information theoretic) iO

Suppose that a class C is exact learnable with membership queries

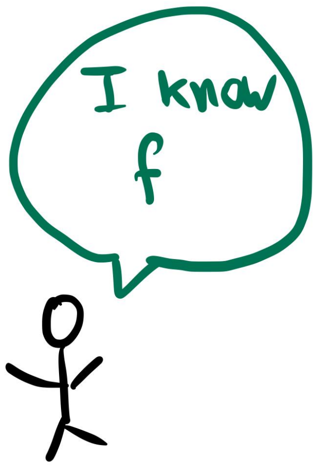
Exact learning implies (information theoretic) iO

Suppose that a class C is exact learnable with membership queries



Exact learning implies (information theoretic) iO

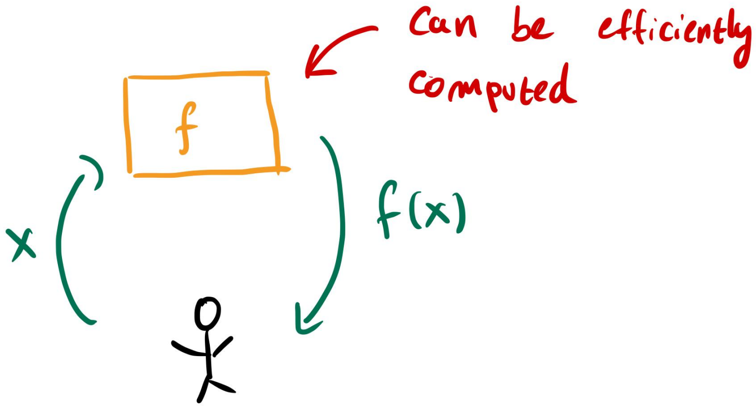
Suppose that a class C is exact learnable with membership queries



Exact learning implies (information theoretic) iO

Suppose that a class C is exact learnable with membership queries.

And that the membership queries can be efficiently computed



Exact learning implies (information theoretic) iO

Then we can obfuscate: simply run the learning algorithm, and simulate the membership queries.

Exact learning implies (information theoretic) iO

Then we can obfuscate: simply run the learning algorithm, and simulate the membership queries.

Since the learning algorithm only “sees” semantic properties, the resulting program does not depend on the initial representation.

Exact learning implies (information theoretic) iO

This paradigm works with any query that is efficiently computable from a representation, and depends only on the semantic of the function.

Exact learning implies (information theoretic) iO

This paradigm works with any query that is efficiently computable from a representation, and depends only on the semantic of the function.

Hint function for a program class:

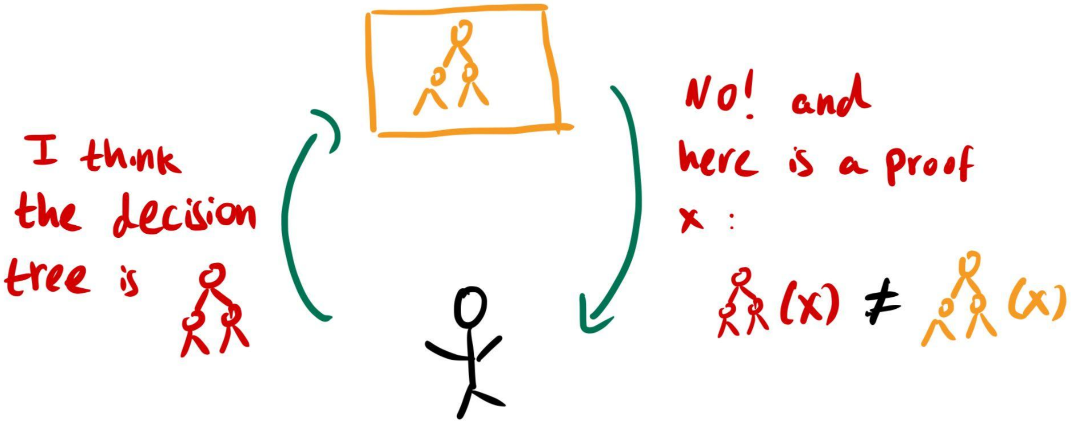
Efficiently computable property of the class that depends only on the underlying function.

Exact learning implies (information theoretic) iO

Hint function for a program class:

Efficiently computable property of P that depends only on the underlying function.

Example: equivalence queries for decision trees.

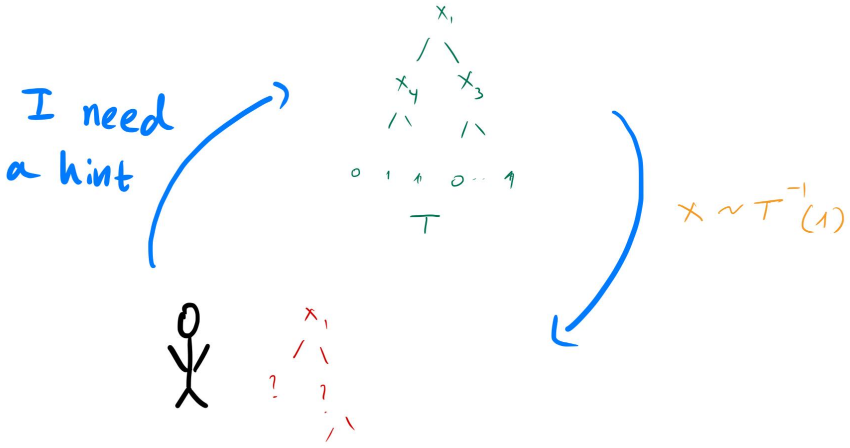


Exact learning implies (information theoretic) iO

Hint function for a program class:

Efficiently computable property of P that depends only on the underlying function.

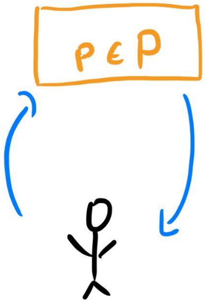
Example: a uniform satisfying assignment for decision trees.



Characterizing Information-Theoretic iO

Theorem 1:

For any class of programs P, P admits a polynomial-time perfect (resp. deterministic, canonical) iO **if and only if** P has a polynomial-time exact learning algorithm with randomized (resp. deterministic, canonical) hint functions.



$$\equiv$$

$$\forall P_1, P_2 \in P : |P_1| = |P_2|$$
$$P_1 \equiv P_2$$
$$iO(P_1) \approx iO(P_2)$$

Characterizing Information-Theoretic iO

Theorem 1:

For any class of programs P, P admits a polynomial-time perfect (resp. deterministic, canonical) iO **if and only if** P has a polynomial-time exact learning algorithm with randomized (resp. deterministic, canonical) hint functions.

Furthermore, the obfuscation algorithm is proper **if and only if** the learning algorithm is proper.

Characterizing Information-Theoretic iO

Theorem 1:

For any class of programs P, P admits a polynomial-time perfect (resp. deterministic, canonical) iO **if and only if** P has a polynomial-time exact learning algorithm with randomized (resp. deterministic, canonical) hint functions.

Proof idea:

=> Run the learning algorithm

<= Use the iO as the hint function

Learning in the mistake bound model

Mistake bound model:

Examples arrive in a stream: x1, x2, … (adversarially ordered)

After observing x, the learner needs to predict c(x)

After every mistake the learner makes, they can update their concept

Goal: make at most M mistakes.

Learning in the mistake bound model

Theorem [V87]:

If a concept class can be learnt in the mistake bound model, it can be exact learnt using equivalence and membership queries.

Learning in the mistake bound model

Applications:

If P is learnable in the mistake bound model, and implementing equivalence and membership queries for programs in P is easy, then we can obfuscate.

Learning in the mistake bound model

Applications:

If P is learnable in the mistake bound model, and implementing equivalence and membership queries for programs in P is easy, then we can obfuscate.

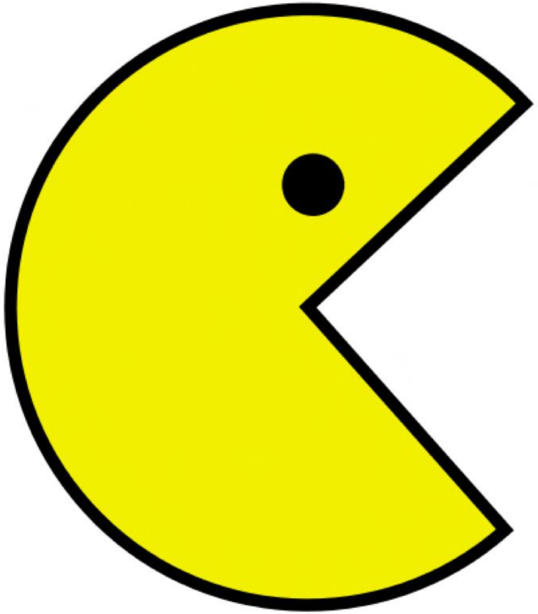
=> Quasi-polynomial time canonization algorithm for DTs.

Our learning framework together with Learning DT in the mistake bound model [S95].

Conceptually easier than the algorithm in [AKKRV15].

Result 2: negative results using PAC learning

"The PAC attack"



3CNF
Proper iO

The case of 3CNFs

The equivalence problem for 3CNFs is coNP hard, and so we cannot hope for statistical obfuscation.

The case of 3CNFs

The equivalence problem for 3CNFs is coNP hard, and so we cannot hope for statistical obfuscation.

What about (computationally) proper obfuscation for 3CNFs?

(The output of the obfuscator is also a 3CNF)

PAC Learning for Negative Results

Theorem 2:

If one-way functions exist, there is no polynomial-time iO of 3-CNF formulas by 3-CNF formulas with computational indistinguishability.

PAC Learning for Negative Results

Theorem 2:

If one-way functions exist, there is no polynomial-time iO of 3-CNF formulas by 3-CNF formulas with computational indistinguishability.

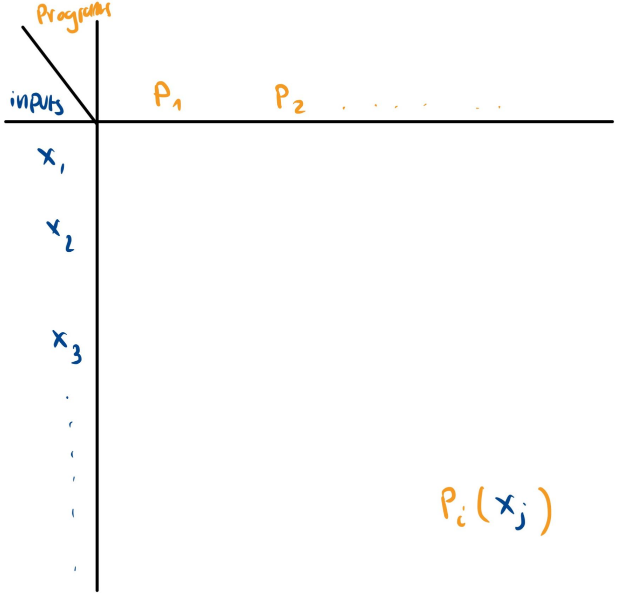
Our proof gives a stronger statement:

If one-way functions exist, there is no polynomial-time iO of 3-CNF formulas by a program class whose dual class is PAC learnable, with computational indistinguishability.

PAC Learning for Negative Results

Theorem 2:

The dual class of a program class:



PAC Learning for Negative Results

Theorem 2:

The dual class of a program class:

$$\mathrm{Eval}(P, x) = P(x)$$

$\mathrm{Eval}(P, \cdot)$ $\qquad\qquad$ $\mathrm{Eval}(\cdot, x)$

P $\qquad\qquad$ dual

PAC Learning for Negative Results

Theorem 2:

If one-way functions exist, there is no polynomial-time iO of 3-CNF formulas by a program class whose dual class is PAC learnable, with computational indistinguishability.

PAC Learning for Negative Results

Theorem 2:

If one-way functions exist, there is no polynomial-time iO of 3-CNF formulas by a program class whose dual class is PAC learnable, with computational indistinguishability.

Cannot obfuscate 3CNFs by O(1)-CNFs

PAC Learning for Negative Results

Theorem 2:

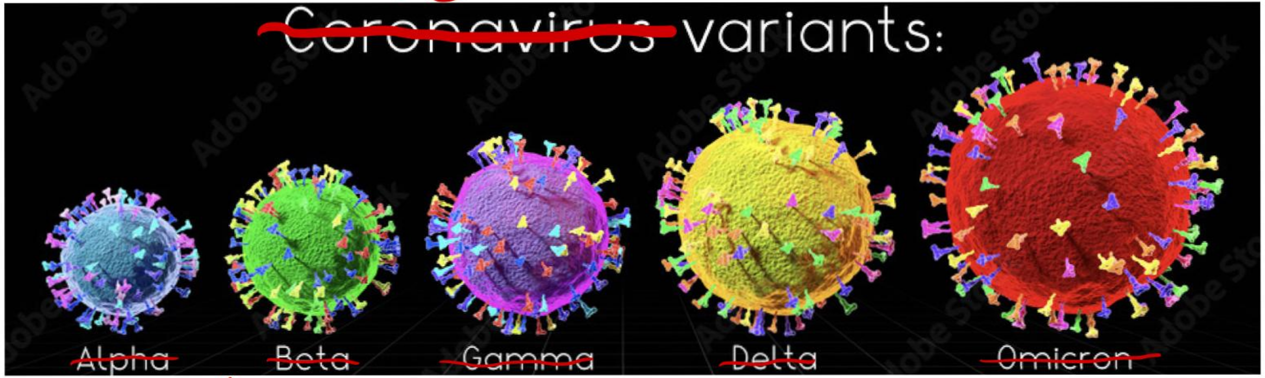
If one-way functions exist, there is no polynomial-time iO of 3-CNF formulas by a program class whose dual class is PAC learnable, with computational indistinguishability.

Cannot obfuscate 3CNFs by O(1)-CNFs

(Unlike what we heard yesterday by Mikito Nanashima, 3CNFs are easy to dualize)

Result 3: separating variants of iO

iO



computational iO

stat iO

Perfect iO ≠ Deterministic iO ≠ Canonizer

Result 3: Separating Notions of Information-Theoretic iO

Deterministic iO:

A ~~probabilistic~~ polynomial time algorithm iO is an *indistinguishability obfuscator* , if for all pairs of "programs" C_1, C_2 satisfying $C_1 \equiv C_2$ and $|C_1| = |C_2|$ we have that $\mathrm{iO}(C_1) \underset{=}{\cancel{\approx}} \mathrm{iO}(C_2)$.

Result 3: Separating Notions of Information-Theoretic iO

<u>Canonizer</u>:

A ~~probabilistic~~ polynomial time algorithm iO is an *indistinguishability obfuscator* , if for all pairs of "programs" C_1, C_2 satisfying $C_1 \equiv C_2$ ~~and $|C_1| = |C_2|$~~ we have that $\mathrm{iO}(C_1) \cancel{\approx} \mathrm{iO}(C_2)$.

$=$

Result 3: Separating Notions of Information-Theoretic iO

Separating canonizer and deterministic iO:

Program class with one circuit for every size, and equivalence is hard.

Result 3: Separating Notions of Information-Theoretic iO

<u>"Abstract obfuscation"</u>

$$R_n \subseteq \{0,1\}^n \times \{0,1\}^n$$

(Intuition): "Two strings are in R iff they represent the same program"

Result 3: Separating Notions of Information-Theoretic iO

"Abstract obfuscation"

Canonizer

$(x, Can(x)) \in R_n$

$(x, y) \in R_n \Rightarrow Can(x) = Can(y)$

Result 3: Separating Notions of Information-Theoretic iO

“Abstract obfuscation”

Perfect iO

$$iO_R(x) \sim \{y : (x,y) \in R_n\}$$

Result 3: Separating Notions of Information-Theoretic iO

"Abstract obfuscation"

$f : \{0,1\}^* \to \{0,1\}^*$ has a

random self reduction

Result 3: Separating Notions of Information-Theoretic iO

"Abstract obfuscation"

$f : \{0,1\}^* \rightarrow \{0,1\}^*$ has a

random self reduction

$\exists$ p.p.t σ s.t $f(x) = f(\sigma(x))$

and $\sigma(x) \sim \{y : f(x) = f(y)\}$

Result 3: Separating Notions of Information-Theoretic iO

"Abstract obfuscation"

$f : \{0,1\}^* \to \{0,1\}^*$ has a

random self reduction

$\exists$ p.p.t σ s.t $f(x) = f(\sigma(x))$

and $\sigma(x) \sim \{y : f(x) = f(y)\}$

Hard to check if $f(x) = f(y)$

Result 3: Separating Notions of Information-Theoretic iO

<u>“Abstract obfuscation”</u>

We can use such f to create a relation that has a “perfect iO” but we cannot canonize.

$x \sim y \Leftrightarrow f(x) = f(x)$

Result 3: Separating Notions of Information-Theoretic iO

<u>“Abstract obfuscation”</u>

We can use such f to create a relation that has a “perfect iO” but we cannot canonize.

$x \sim y \Leftrightarrow f(x) = f(x)$

DDH

Result 3: Separating Notions of Information-Theoretic iO

Theorem 3:

If one-way functions exist, and under a special-purpose VBB obfuscation assumption (alternatively, using ideal obfuscation), there is a program class P such that P admits perfect (proper) iO in the CRS model but not deterministic (even improper) iO in the same model.

Result 3: Separating Notions of Information-Theoretic iO

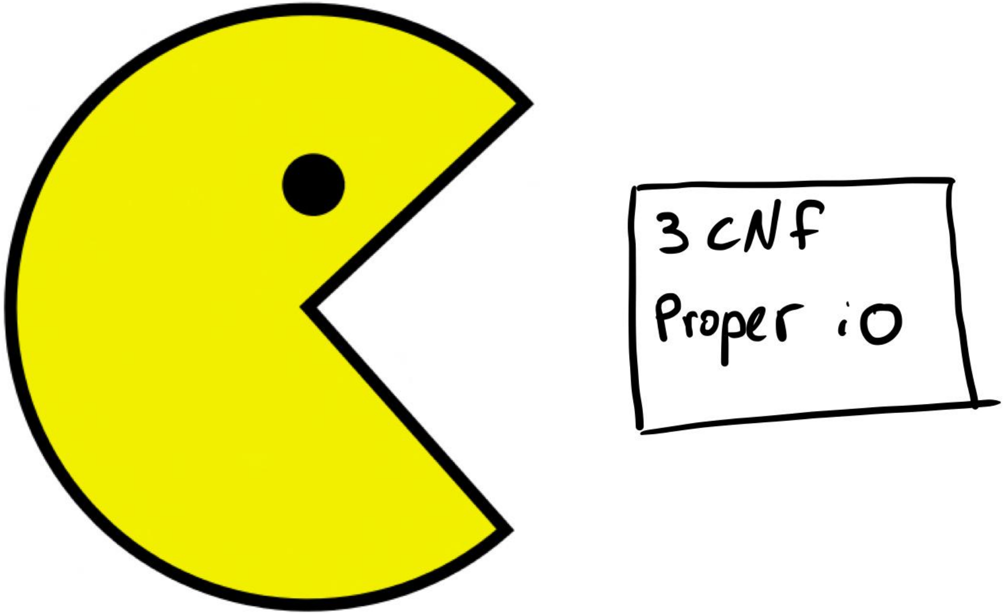
Theorem 3:

If one-way functions exist, and under a special-purpose VBB obfuscation assumption (alternatively, using ideal obfuscation), there is a program class P such that P admits perfect (proper) iO in the CRS model but not deterministic (even improper) iO in the same model.

“Oracle separation”

The PAC attack explained

"The PAC attack"



PAC Learning for Negative Results

Proof idea: "PAC Attack"

Given a OWF and a proper iO for 3CNFs, we can construct a PKE with a decryption circuit (with the secret key hardcoded) that is O(1)-CNF.

PAC Learning for Negative Results

Proof idea: “PAC Attack”

Given a OWF and a proper iO for 3CNFs, we can construct a PKE with a decryption circuit (with the secret key hardcoded) that is O(1)-CNF.

But O(1)-CNFs can be PAC learnt! [V84]

PAC Learning for Negative Results

Proof idea: “PAC Attack”

Given a OWF and a proper iO for 3CNFs, we can construct a PKE with a decryption circuit (with the secret key hardcoded) that is O(1)-CNF.

But O(1)-CNFs can be PAC learnt! [V84]

That means we can learn the decryption circuit and break the PKE.

PKE from iO

Similar in spirit to [FFP18]

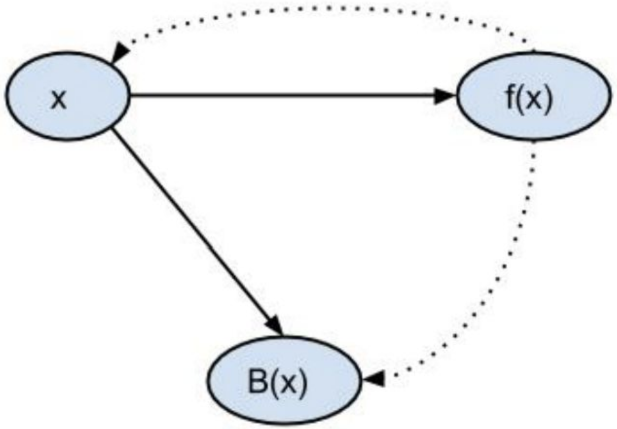
$f : \{0,1\}^n \rightarrow \{0,1\}^n$ (Injective) OWF

$hc : \{0,1\}^n \rightarrow \{0,1\}$ Hardcore Predicate for f

$m \in \{0,1\}$ A bit we want to encrypt

PKE from iO

Reminder: hardcore predicate (if you are here, probably you are not a cryptographer. They are all working for the CRYPTO deadline now…)



PKE from iO

Key Generation:

$x \xleftarrow{\$} \{0,1\}^n$ $\quad$ $sk \leftarrow (b, x)$

$b \leftarrow hc(x)$ $\quad$ $pk \leftarrow (f(x))$

PKE from iO

Encrypt:

$$\phi = (f(\cdot) == pk) \wedge (hc(\cdot) \oplus m)$$

Output $iO(\phi)$

PKE from iO

Decrypt:

output $b \oplus iO(\phi)(x)$

PKE from iO - correctness

$$b \oplus iO(\phi)(x) =$$

$$= b \oplus \left[(f(x) == f(x)) \wedge (b \oplus m) \right]$$

$$= m$$

PKE from iO - security

$\Delta_0 = \{ (pk, c) : \text{ encrypt } 0 \}$

$\Delta'_0 = \{ (pk, c) : \text{ encrypt } 0,\ hc(x) = 0 \}$

$\Delta_1 = \{ (pk, c) : \text{ encrypt } 1 \}$

$\Delta'_1 = \{ (pk, c) : \text{ encrypt } 1,\ hc(x) = 1 \}$

Goal: show that Δ0 ≈ Δ1

PKE from iO - security

$\Delta_0 = \{(pk, c) : \text{encrypt } 0\}$

$\Delta'_0 = \{(pk, c) : \text{encrypt } 0,\ hc(x) = 0\}$

$\Delta_1 = \{(pk, c) : \text{encrypt } 1\}$

$\Delta'_1 = \{(pk, c) : \text{encrypt } 1,\ hc(x) = 1\}$

$\Delta_0 \overset{c}{\approx} \Delta'_0 \quad , \quad \Delta_1 \overset{c}{\approx} \Delta'_1$ (hc is a hard core predicate)

$\Delta'_0 \overset{c}{\approx} \Delta'_1$ (Both formulas are unsatisfiable)

PKE from iO

Recall that we want the decryption circuit to be 3-CNF

PKE from iO

Recall that we want the decryption circuit to be 3-CNF

Problems:

- The OWF is not necessarily 3-local

PKE from iO

Recall that we want the decryption circuit to be 3-CNF

Problems:

- The OWF is not necessarily 3-local.
- The standard way to convert a formula to 3CNF (Tseytin transformations) only preserves equisatisfiability, and it adds auxiliary variables.

PKE from iO

Recall that we want the decryption circuit to be 3-CNF

Problems:

- The OWF is not necessarily 3-local.
- The standard way to convert a formula to 3CNF (Tseytin transformations) only preserves equisatisfiability, and it adds auxiliary variables.

We can overcome these issues

Open problems

Open problems

1. What other natural representation models admit information-theoretic iO?

Open problems

1. What other natural representation models admit information-theoretic iO?

2. Is it possible to obtain proper iO for decision trees?

Open problems

1. What other natural representation models admit information-theoretic iO?

2. Is it possible to obtain proper iO for decision trees?

3. Can we obfuscate 3CNFs by AC0 circuits?

Open problems

1. What other natural “programs” admit information-theoretic iO?

2. Is there a proper or statistical iO for decision trees?

3. Can we obfuscate 3CNFs by AC0 circuits?

4. Can we separate between information-theoretic notions of iO using natural program classes or under cleaner assumptions?

Questions?

Thank you!