On Low-End Obfuscation and Learning

Elette Boyle
Reichman University, Herzliya, Israel
NTT Research, Sunnyvale, CA, USA

Yuval Ishai
Technion, Haifa, Israel

Pierre Meyer
Reichman University, Herzliya, Israel
IRIF, Université Paris Cité, CNRS, France

Robert Robere
McGill University, Montreal, Canada

Gal Yehuda
Technion, Haifa, Israel

Abstract

Most recent works on cryptographic obfuscation focus on the high-end regime of obfuscating *general circuits* while guaranteeing *computational indistinguishability* between functionally equivalent circuits. Motivated by the goals of simplicity and efficiency, we initiate a systematic study of "low-end" obfuscation, focusing on simpler representation models and information-theoretic notions of security. We obtain the following results.

- **Positive results via "white-box" learning.** We present a general technique for obtaining perfect indistinguishability obfuscation from exact learning algorithms that are given restricted access to the representation of the input function. We demonstrate the usefulness of this approach by obtaining simple obfuscation for decision trees and multilinear read-k arithmetic formulas.
- **Negative results via PAC learning.** A *proper obfuscation* scheme obfuscates programs from a class $\mathcal{C}$ by programs from the same class. Assuming the existence of one-way functions, we show that there is no proper indistinguishability obfuscation scheme for k-CNF formulas for any constant $k \geq 3$; in fact, even obfuscating 3-CNF by k-CNF is impossible. This result applies even to computationally secure obfuscation, and makes an unexpected use of PAC learning in the context of *negative* results for obfuscation.
- **Separations.** We study the relations between different information-theoretic notions of indistinguishability obfuscation, giving cryptographic evidence for separations between them.

2012 ACM Subject Classification Theory of computation → Cryptographic primitives

Keywords and phrases Indistinguishability obfuscation, cryptography, learning

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.23

Funding *Elette Boyle*: supported by AFOSR Award FA9550-21-1-0046, a Google Research Award, and ERC Project HSS (852952).
Yuval Ishai: supported by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20.
Pierre Meyer: supported by ERC Project HSS (852952).
Robert Robere: supported by NSERC.

Acknowledgements We thank Geoffroy Couteau for helpful pointers and suggestions.

1 Introduction

The study of program obfuscation and its applications has been a central research theme in cryptography for the past two decades. Following the initial impossibility results for a strong form of "virtual black-box" (VBB) obfuscation [6, 25], the notion of *indistinguishability obfuscation* (iO) emerged as one that is liberal enough to be realized under well-studied cryptographic assumptions [21, 32], yet strong enough for a surprisingly wide variety of applications [42]. In fact, iO is a best-possible form of obfuscation, in the sense that if any obfuscation scheme suffices for a given application then iO does as well [27].

More concretely, an iO scheme is an efficient randomized algorithm that maps a source program P into a functionally equivalent target program $\hat{P}$, while satisfying the following intuitive security guarantee: for every two equivalent programs P, P' of the same size, the corresponding obfuscated programs $\hat{P}, \hat{P'}$ are computationally indistinguishable.

At this point, almost all research on iO has been focused on what we call the *high-end regime.* In this regime, the source program P and/or the target program P' are taken from program classes with high representational power, such as boolean circuits or branching programs. While recent work in the area has led to a variety of general constructions, they are all quite complex, and inherently rely on cryptographic assumptions.

In this work, we deviate from prior research and instead focus on *low-end obfuscation.* Here, by "low-end" we mean (1) the input program is in a *weak* representation model, such as a decision trees or low-width CNF formulas, and (2) the obfuscation algorithm should be *proper*,[1] in the sense that the output respects the representation of the source program. For instance, if the source program P is a decision tree, then the target obfuscated program P' should also be a decision tree. While low-end obfuscation restricts the scope of the "best possible" guarantee of iO to other proper obfuscation schemes, it can still provide a meaningful and useful notion of security.

Of course, one can always apply high-end obfuscation schemes to weaker computation models; however, this has several quite significant drawbacks:

- **Inefficiency.** Current obfuscation constructions for general circuits (or even for weaker computation models, e.g. [11, 5, 12, 28, 48, 20]) have a large concrete overhead, typically making them infeasible to implement.
- **Incompatible formats.** Current constructions do not respect the *structure* of their source programs: for example, even for a simple program such as a CNF formula or decision tree, the obfuscated program is still a general boolean circuit. Simple representation models are often desirable in practice.
- **Computational vs. information-theoretic security.** As soon as testing satisfiability of a program is NP-Hard (alternatively, testing equivalence is coNP-hard), information-theoretic iO for the class implies that the polynomial-time hierarchy collapses [27, 10]. This means that computational indistinguishability is the strongest guarantee we can hope to get for rich models like boolean circuits; this is not true for weaker representation models.

To summarize, current high-end obfuscation schemes may be "overkill" for weak representation models, where much more efficient algorithms can exist that have a proper output and achieve information-theoretic security guarantees.

While we believe that low-end obfuscation is natural to study in and of itself, we note that there are alternative motivations for studying it. One natural motivation is *private data analysis.* Low-end obfuscation provides a utility-preserving (albeit much weaker) alternative to differential privacy [17]: instead of replacing a learning algorithm with a differentially private one, we could leave the learning algorithm unchanged and then obfuscate the output representation in order to eliminate unnecessary information leakage about the training set. In this context, iO provides a means for maximizing privacy without sacrificing utility. This is similar in spirit to history-independent data structures, which aim to achieve a similar goal in a different context [39, 41, 40].

[1] We borrow the name "proper" from the corresponding notion of *proper learning* in learning theory where the output hypothesis must be from the same concept class.

On a more conceptual level, a second motivation for studying low-end obfuscation is that it reveals interesting new connections between obfuscation and learning theory. Even in the original work introducing iO [6] it was observed that any exact learning algorithm using membership queries implies the strong notion of information-theoretic VBB obfuscation. This raises the question – which we address in the present work – of pinning down the "right" learning model for such positive results when settling for the weaker notion of information-theoretic iO. As we will see in the sequel, an unexpected consequence of the present work is that learning algorithms can also imply *negative* results for low-end obfuscation.

We note that several prior works have studied various notions of obfuscation for simple types of source programs [14, 38, 47, 31, 2, 29, 15, 11, 5, 12, 34, 13, 16, 8, 7, 20], but with different goals in mind. Indeed, the prior works did not impose any restrictions on the target program, and the presented constructions continued to suffer from the drawbacks discussed above. One prior example for *proper* obfuscation from the literature is for the class of (P)OBDDs [27], which was also used to separate proper iO from proper VBB obfuscation. We will discuss other examples of this kind later (cf. Section 3).

1.1 Our Contributions

Before stating our results, let us cover some necessary background. If iO is an indistinguishability obfuscation scheme for a program class $\mathcal{P}$, we say that iO is *proper* if $\mathsf{iO}(P) \in \mathcal{P}$ for all $P \in \mathcal{P}$. As we have discussed above, when studying weaker representation models obtaining information-theoretic security guarantees suddenly becomes an achievable goal. In the present work, we will be interested in variations of information-theoretic security. The weakest notion of information-theoretic iO that we consider is *statistical* iO, which guarantees that for any two equivalent programs $P \equiv P'$ of the same size, the output distributions $\mathsf{iO}(P)$ and $\mathsf{iO}(P')$ are close in statistical distance.

Once we enter the low-end regime, however, much stronger notions of iO are attainable. For a motivating example let us consider the class of 2-CNF formulas. It is well-known that testing equivalence of 2-CNF formulas is computable in polynomial time, and so the complexity-theoretic barrier for information-theoretic iO does not apply to this class. However, for 2-CNF formulas it is not hard to see (cf. Section 3) that we can design an obfuscation algorithm that has *much* stronger indistinguishability guarantees than mere statistical obfuscation. Indeed, the output of the obfuscation is not only statistically secure but *canonizing*, in the sense that iO is a deterministic algorithm such that for *any* pair of equivalent 2-CNF formulas $F \equiv F'$ (not even of the same size) we have $\mathsf{iO}(F) = \mathsf{iO}(F')$. This is the strongest notion of indistinguishability that we consider.

In light of this we found it useful to systematically introduce further refinements of information-theoretic indistinguishability. We define the following notions: if $P \equiv P'$ are two functionally equivalent programs of the same size then an indistinguishability obfuscation scheme iO is

- *statistical*, if the output distributions $\mathsf{iO}(P)$ and $\mathsf{iO}(P')$ are close in statistical distance,
- *perfect*, if $\mathsf{iO}(P)$ and $\mathsf{iO}(P')$ are identically distributed,
- *deterministic*, if iO is a deterministic algorithm, and
- *canonizing*, if iO is deterministic and if $\mathsf{iO}(P) = \mathsf{iO}(P')$ for *every* pair of functionally equivalent programs $P \equiv P'$ (not only those of the same size[2]).

[2] Therein lies the difference between deterministic obfuscation and canonization: the output of a deterministic indistinguishability obfuscation scheme is only guaranteed to be the same on two equivalent

Positive results via “white-box” learning

Our first contribution is a refinement of the connection between learning and information-theoretic obfuscation. First, let us recall the observation of [6] connecting exact learning to the very strong model of *virtual black box* (VBB) obfuscation. In the model of *exact learning with membership queries* [3] we are given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$ chosen from some concept class $\mathcal{H}$, and the goal is to use the oracle in order to produce a concept $h \in \mathcal{H}$ such that $h \equiv f$. For instance, if we have a deterministic polynomial-time learning algorithm in this model then it is not hard to see that we also have a polynomial-time canonizer for the class $\mathcal{H}$. This is because in the canonization problem we are given a program P representing f as input and can therefore use P to simulate the calls to the membership oracle of f. Crucially, since the oracle calls depend only on f, and not on the particular representation of f as a program P, the output of the underlying learning algorithm will be the same for all functionally equivalent programs. And since membership queries can be simulated given oracle access to f, we actually get an information-theoretic VBB *obfuscator* for the program class.

Our main observation is that the weaker notion of information-theoretic iO is captured by a stronger learning model. Concretely, since we are given the program P as input in the obfuscation task, we are not restricted only to using membership queries and can instead use *any* efficiently computable property of P that depends only on the underlying function f. Indeed, for weak models of computation like decision trees we can make many inferences from the representation that are impossible for strong models of computation such as boolean circuits – for example, testing equivalence between two input programs. To capture this notion, for a class of programs $\mathcal{P}$ we define a *representation-oblivious hint function* (cf. Section 3) to be, roughly speaking, any efficiently computable function $h : \mathcal{P} \times \{0,1\}^* \to \{0,1\}^*$ such that for any functionally equivalent programs $P \equiv P'$ of the same size and for any $x \in \{0,1\}^*$ we have $h(P,x) = h(P',x)$. Now, it is natural to define the model of *exact learning with hint functions*, where the learning algorithm is given oracle access to a predefined hint function $h(P,\cdot)$ for some program P – but *not* the program P itself – and it must recover a hypothesis $P' \in \mathcal{P}$ such that $P' \equiv P$ using queries to $h(P,\cdot)$. Informally speaking, we think of a hint function as representing the information that we can compute about f when given the representation of f by P. Note that while membership queries are one obvious example of hint functions, there are many other possibilities, such as equivalence queries or even more esoteric operations.

We now state our first contribution, which exactly characterizes low-end obfuscation in terms of exact learning with hint functions.

▶ **Theorem 1** (Informal). *For any class of programs $\mathcal{P}$, $\mathcal{P}$ admits a polynomial-time perfect (resp. deterministic, canonical)* iO *if and only if $\mathcal{P}$ 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.*

Using this connection, we observe that several learning results from the literature in various models of learning imply low-end obfuscation schemes. For instance, we prove a general result stating that if a concept class $\mathcal{C}$ is learnable in the well-studied *mistake-bound*

programs *of same size*. For a separating example, consider a family of programs where no two circuits have the same size; the identity function, mapping each program to itself, is then trivially a deterministic obfuscation scheme, but is not a canonizer as soon as two different programs (of different size) in the class are functionally equivalent. In particular, deterministic obfuscation for a program class does not *a priori* imply efficient equivalence testing.

learning model then $\mathcal{C}$ admits a canonical iO scheme. We believe this result is interesting for two reasons. First, it uses both membership *and* equivalence queries, which reaches beyond the membership query algorithms due to [6] (indeed, note that 2-CNF formulas are not exactly learnable with membership queries alone). Second, it immediately implies canonizing algorithms for decision trees,[3] 2-CNF formulas, and multilinear read-k arithmetic formulas. This is particularly interesting for the case of decision trees, as it simplifies an ad-hoc canonizing algorithm for decision trees given by [4]. We refer to Section 3 for details.

Negative results for proper iO via PAC learning

Our next contributions are new impossibility results for low-end obfuscation. In particular, we show that proper obfuscation for the class of 3-CNF formulas is impossible, even when we settle for computational indistinguishability and even if we allow the obfuscation algorithm to output an $O(1)$-CNF as the output.

▶ **Theorem 2** (Informal). *If one-way functions exist, there is no polynomial-time* iO *of* 3*-CNF formulas by* $O(1)$*-CNF formulas with computational indistinguishability.*

Several remarks on this result are in order. First, we note that this theorem is tight in the sense that 2-CNF formulas can be canonized (and therefore admit proper indistinguishability obfuscation). Second, the proof of this theorem also relies on the usage of learning algorithms: this time, the known PAC learning algorithms for $O(1)$-CNFs [46]. In fact, the proof of this result follows from a much more general statement: namely, that it is impossible to obfuscate 3-CNF formulas by *any* class of programs for which the dual class (in the sense of learning theory) is PAC learnable. While it has been long-known that learning algorithms imply *positive* results in obfuscation, to the best of our knowledge this is the first example of a learning algorithm giving a *negative* result in obfuscation, which we believe is of independent interest. The high-level idea is to use the PAC learning algorithm to break a public-key encryption scheme that could be built from a one-way function using the low-end iO. We refer to Section 4 for details.

Separating notions of information-theoretic iO

The feasibility of information-theoretic iO in the low-end regime motivates a more refined study of the distinction between the different flavors of information-theoretic iO discussed above. Given the previous positive examples, one might be tempted to conjecture that information-theoretic iO can always be made deterministic (possibly under derandomization assumptions).

Towards a counterexample, consider a program class $\mathcal{P}$ as an equivalence relation R between bit-strings representing equivalent programs. A separation between perfect and deterministic iO implies R for which it is possible to efficiently sample uniformly from the equivalence class of a given program P, but there is no efficient deterministic algorithm mapping each P to a canonical representative of its class. It turns out that such a relation R is quite easy to construct under standard cryptographic assumptions. Concretely, consider a non-interactive commitment scheme which is *dense* in the sense that every bit-string is a valid commitment of some message and is *rerandomizable* in the sense that given a commitment one can efficiently sample a random commitment of the same message. Such a commitment

[3] In the case of decision trees, the canonizing algorithm runs in *quasi*-polynomial time.

scheme can be based on the standard Decisional Diffie-Hellman (DDH) assumption. Now, let R be an equivalence relation on n-bit strings, where two bit-strings are equivalent if they represent commitments to the same message. Then an efficient sampling algorithm as above follows directly from the rerandomization property. On the other hand, an efficient algorithm for finding a canonical representative could be used to test equality of two committed messages, violating the secrecy property of the commitment scheme.

While a relation R as above is necessary for the desired separation, it is not sufficient. Indeed, so far we ignored the fact that each program P should have concrete *semantics* defined by an efficient evaluation algorithm, where equivalent programs must have the same semantics. This requirement is at odds with the secrecy property of the commitment scheme. To get around this issue we take a different approach, which we can only instantiate using strong forms of (computational, VBB) obfuscation or alternatively in a generic "ideal obfuscation" model. The equivalence relation R is defined via a pseudorandom partition of n-bit strings, where each program class is assigned a distinct "evasive" function, say a random point function, as its semantics. (We could alternatively use a random *constant* function if we only wanted to rule out *proper* deterministic iO.) Given the bit-string representing P, the obfuscation gives oracle access to its assigned evasive function. To enable perfect iO without allowing deterministic iO, we connect the programs in each equivalence class by a random cycle of length N, and implement a "fast-forward" oracle that given program P and integer k returns the program P' obtained by taking k steps along the cycle. Perfect iO is then realized by invoking the fast-forward oracle with a uniformly random k, $0 \leq k < N$. On the other hand, a deterministic iO in this setting can be shown to be at least as hard as solving the discrete logarithm problem in Shoup's generic group model [43]. This separation can be formalized by assuming either ideal obfuscation or strong forms of (computational) obfuscation, where a common reference string (CRS) is used for defining the pseudorandom graph.

▶ **Theorem 3** (Informal). *If one-way functions exist, and under a special-purpose* VBB *obfuscation assumption (alternatively, using ideal obfuscation), there is a program class $\mathcal{P}$ such that $\mathcal{P}$ admits perfect (proper)* iO *in the CRS model but not deterministic (even improper)* iO *in the same model.*

A somewhat unexpected feature of the above separation is that it *uses* a computational notion of obfuscation to separate between information-theoretic notions of obfuscation. That being said, the separation requires a special-purpose VBB obfuscation assumption, which means that it can either be cast in an idealized model (such as ideal multilinear maps) or instantiated heuristically using existing general-purpose iO constructions. A similar use of special-purpose VBB obfuscation was made in other contexts (e.g., [22, 9, 30, 1]).

1.2 Open Problems

This work suggests many open problems. We record a few of the most interesting here:

1. What other natural representation models admit information-theoretic iO?
2. By using known learning algorithms we obtain quasipolynomial-time canonizing obfuscation algorithms for decision trees, but there likely does not exist *polynomial*-time canonizing algorithms for decision trees since this would violate known hardness-of-approximation results for decision tree size. Is it possible to obtain polynomial-time proper iO for decision trees using our generalized exact learning framework?

3. Can 3-CNF formulas be properly obfuscated by AC^0-circuits, or even (unbounded-width) CNF? Note that the known learning algorithms for AC^0 [35] only work over the uniform distribution, which is not strong enough to apply the techniques from Section 4.
4. Can we separate between information-theoretic notions of iO using *natural* program classes $\mathcal{P}$ or under cleaner assumptions?

2 Preliminaries

We recall the standard notions of perfect, statistical, and computational indistinguishability of distribution ensembles.

▶ **Definition 4** (Notions of 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:*

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

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*.*

2.1 Obfuscation

Towards a unified definition of proper obfuscation, we capture different representations of functions (such as circuits, CNF formulas, or decision trees) via the following general notion of a *program class.*

▶ **Definition 5** (Program class). *A* program class *is a pair* $\mathcal{P} = (\mathsf{Eval}, \mathsf{Size})$ *with the following syntax.*

- $\mathsf{Eval} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \cup \{\bot\}$ *is a polynomial-time algorithm such that* $\mathsf{Eval}(P,x)$ *returns the output of program* P *on input* x*. The algorithm* Eval *returns* $\bot$ *if* P *is not a valid program or* x *is not a valid input for* P*.*
- $\mathsf{Size} : \{0,1\}^* \to \mathbb{N}$ *is a polynomial-time algorithm returning a size parameter* $\mathsf{Size}(P) \le |P|$ *for a given program* P*.*

The programs P, P' *are* functionally equivalent*, written* $P \equiv_{\mathcal{P}} P'$*, if* $\mathsf{Eval}(P,x) = \mathsf{Eval}(P',x)$ *for all* $x \in \{0,1\}^*$*. For notational convenience, when the program class is known from context we will simply write* $P \equiv P'$ *to denote equivalent programs, use the shorthand* $P(x) := \mathsf{Eval}(P,x)$*, and also write* $P \in \mathcal{P}$ *to denote that* P *is a valid program chosen from the class* $\mathcal{P}$*.*

Standard representation models for functions, such as (boolean or arithmetic) circuits, formulas and decision trees, can all be cast as instances of the above definition. For these classes, the size parameter is typically defined as the number of nodes in the corresponding graph.

We next define a general notion of indistinguishability obfuscation that captures the notions of proper and improper obfuscation we will consider in this work.

▶ **Definition 6** (Indistinguishability Obfuscation (iO)). *A computational (resp. statistical, perfect) indistinguishability obfuscation scheme of program class* $\mathcal{P} = (\mathsf{Eval}_{\mathcal{P}}, \mathsf{Size}_{\mathcal{P}})$ *(Definition 5) by program class* $\mathcal{P}' = (\mathsf{Eval}_{\mathcal{P}'}, \mathsf{Size}_{\mathcal{P}'})$ *is a p.p.t. algorithm* iO *satisfying the following:*

- ***Syntax:*** *For any security parameter* λ *and a program* P, $\mathsf{iO}(1^\lambda, P) \in \mathcal{P}$.
- ***Completeness:*** *For every* $\lambda \in \mathbb{N}$ *and program* P *we have:*

  $$\Pr\left[P \equiv \mathsf{iO}(1^\lambda, P)\right] = 1$$

  where the probability is taken over the internal randomness of iO.
- ***Indistinguishability:*** *For every pair of program sequences* P^0_λ, P^1_λ *such that* $P^0_\lambda \equiv P^1_\lambda$ *and* $\mathsf{Size}(P^0_\lambda) = \mathsf{Size}(P^1_\lambda)$, *the following distribution ensembles are computationally (resp. statistically, perfectly) indistinguishable:*

  $$\{\mathsf{iO}(1^\lambda, P^0_\lambda)\}_{\lambda\in\mathbb{N}} \qquad \textit{and} \qquad \{\mathsf{iO}(1^\lambda, P^1_\lambda)\}_{\lambda\in\mathbb{N}} .$$

The obfuscation scheme iO *is said to be* proper *if* $\mathcal{P} = \mathcal{P}'$ *and* improper *otherwise.*

The standard notion of iO considered in the cryptographic literature corresponds to proper computational iO for the class of boolean circuits. We note that perfect iO implies statistical iO, which in turn implies computational iO. Next we define two deterministic models of iO.

▶ **Definition 7.** *An* iO *scheme for a program class* $\mathcal{P}$ *is* deterministic *if it is computed by a deterministic algorithm. Formally, for any two equivalent programs* $P \equiv P'$ *in* $\mathcal{P}$ *satisfying* $\mathsf{Size}(P) = \mathsf{Size}(P')$ *it holds that* $\mathsf{iO}(P) = \mathsf{iO}(P')$.

A canonization scheme *for the program class* $\mathcal{P}$ *is a deterministic* iO *scheme such that* $\mathsf{iO}(P) = \mathsf{iO}(P')$ *for* all *equivalent programs* $P \equiv P'$ *in* $\mathcal{P}$.

A canonization scheme for a program class $\mathcal{P}$ implies very strong structural properties for $\mathcal{P}$ – for example, it implies that equivalence testing for programs in $\mathcal{P}$ is easy. Despite this, we show in the next section that canonization is achievable for several natural program classes. Furthermore, we remark that deterministic iO is weaker than canonization since it does not imply efficient equivalence testing for program classes.

We will also consider a natural extension of the above notions of iO to the common reference string (CRS) model. Here the CRS is generated by a trusted setup algorithm and is available both to programs and to distinguishers.

▶ **Definition 8** (Virtual Black Box Obfuscation (VBB), Adapted from [6]). *An (efficient) virtual black box obfuscation scheme of program class* $\mathcal{P} = (\mathsf{Eval}_{\mathcal{P}}, \mathsf{Size}_{\mathcal{P}})$ *(Definition 5) by program class* $\mathcal{P}' = (\mathsf{Eval}_{\mathcal{P}'}, \mathsf{Size}_{\mathcal{P}'})$ *is a p.p.t. algorithm* VBB *satisfying the following:*

- ***Syntax:*** *On input a security parameter* 1^λ *and a program* P, VBB *outputs a program* P'.
- ***Preserving Functionality:*** *For every* $\lambda \in \mathbb{N}$, *program* P, *and input* x, *the following holds:*

  $$\Pr\left[\mathsf{Eval}_{\mathcal{P}'}(P', x) = \mathsf{Eval}_{\mathcal{P}}(P, x) \colon P' \xleftarrow{\$} \mathsf{VBB}(1^\lambda, P)\right] = 1$$
- ***Polynomial Slowdown:*** *There is a polynomial* p *such that for every program* P, *the following holds:*

  $$\Pr\left[\mathsf{Size}(P') \leq \mathsf{Size}(P) \colon P' \xleftarrow{\$} \mathsf{VBB}(1^\lambda, P)\right] = 1$$

- ***"Virtual Black Box":*** *For any p.p.t. $\mathcal{A}$, there is a p.p.t. $\mathcal{S}$ and a negligible function α such that for all programs P, the following holds:*

$$|\Pr[\mathcal{A}(\mathsf{VBB}(P)) = 1] - \Pr\left[\mathcal{S}^{P}(1^{\mathsf{Size}(P)} = 1)\right]| \leq \alpha(\mathsf{Size}(P)) .$$

The VBB obfuscation scheme is said to be proper *if $\mathcal{P} = \mathcal{P}'$ and* improper *otherwise. If we omit the program class $\mathcal{P}$, then it is assumed to be the collection of all circuits.*

3 Low-End Obfuscation from Learning

As we have discussed above, several connections between various models of learning and obfuscation have been previously observed, such as the connections between exact learning and membership queries [6], and also with PAC learning [10]. The goal of this section is refine the connection between learning theory and information-theoretic indistinguishability obfuscation, and show how to use learning algorithms to give strong proper iO guarantees for some program classes. The key idea is to think of indistinguishability obfuscation as an exact learning task where we are given a representation of the function as a program, but, we can only use this program to query *semantic* properties of the function[4]. Intuitively speaking, this re-frames proper obfuscation as understanding which semantic properties we can learn about functions when given their representations as programs. The next definition formalizes this idea.

▶ **Definition 9** (Representation-Oblivious Hint Function). *Let $\mathcal{C} = (\mathsf{Size}, \mathsf{Eval})$ be a program class. A* representation-oblivious hint function *$h : \mathcal{C} \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ is a randomized polynomial-time algorithm such that for any equivalent programs $c \equiv c'$ with $\mathsf{Size}(c) = \mathsf{Size}(c')$ and for any any input $x \in \{0,1\}^*$ the output distributions of $h(c,x)$ and $h(c',x)$ are the same. The hint function is* deterministic *if it is computed by a deterministic algorithm, and is* canonical *if it is deterministic and if for* any *equivalent programs $c \equiv c'$ and any input x we have $h(c,x) = h(c',x)$.*

For example, the evaluation function Eval for any program class $\mathcal{C}$ is a hint function by definition (indeed, it is just a membership query oracle by a different name). For the family of decision trees, a slightly more sophisticated example is the function that takes as an input a decision tree T and outputs a uniformly random point $x \in T^{-1}(1)$. Note that we can compute this efficiently[5] and also it does not depend on the representation of the decision tree, but only on the function the tree computes. Another example is an *equivalence query*: it takes as input two decision trees T_1 and T_2 and either outputs "equivalent" if $T_1 \equiv T_2$ or outputs an input x such that $T_1(x) \neq T_2(x)$; such queries are also well-known to be efficiently computable for decision trees.

▶ **Definition 10** (Exact Learning with Hint Functions). *A program class $\mathcal{C}$ is* exactly learnable with a hint function *if there is a hint function h for $\mathcal{C}$ and a polynomial-time algorithm L such that the following holds. As input, the algorithm L receives $|c|$ and oracle access to the hint function $h(c,\cdot)$ for some unknown $c \in \mathcal{C}$, and satisfies the following property: for every $c \in \mathcal{C}$, L halts and outputs a hypothesis c' such that $c' \equiv c$. We say the learning algorithm is* proper *if $c' \in \mathcal{C}$, and is* improper *otherwise.*

[4] Note that otherwise the learning task is obviously trivial since we could output the program we were given.

[5] This can be done by taking a random walk on the tree, weighted by the number of 1-leaves in the sub-tree rooted by the current vertex in the walk.

We emphasize that in the previous definition the algorithm L only receives access to the program c through the hint function $h(c,\cdot)$, and is not given c explicitly. Moreover, since the hint function can only compute semantic properties of the function $\mathsf{Eval}(c,\cdot)$ corresponding to c, it means that the learning algorithm cannot do something naive like simply outputting c. Indeed, the main theorem of this section shows that this notion of learning is equivalent to information-theoretic indistinguishability obfuscation.

▶ **Theorem 11.** *A program class $\mathcal{C}$ can be exactly learned with a (deterministic, canonical, resp.) hint function if and only if it can be perfectly (deterministically, canonically, resp.) obfuscated. Moreover, the obfuscation is proper if and only if the exact learning algorithm is proper.*

Proof. First let $\mathcal{C}$ be a program class which admits an exact learning algorithm L with respect to a hint function h. Consider the randomized algorithm $\mathcal{O}$ which, on input a program $c \in \mathcal{C}$, runs the learning algorithm L – answering its queries by calling $h(c,\cdot)$ – and outputs the hypothesis produced by L. $\mathcal{O}$ is a perfect obfuscation scheme for program class $\mathcal{C}$: completeness follows from the correctness of the learning algorithm, and perfect indistinguishability follows from representation-obliviousness of h. Indeed, for any two equivalent programs of same size c_1 and c_2, $h(c_1,\cdot) = h(c_2,\cdot)$ and therefore the distributions $\mathcal{O}(c_1)$ and $\mathcal{O}(c_2)$ are identical. Furthermore, it follows from inspection that if L is proper then so is $\mathcal{O}$, and that if L and h are both deterministic or canonical, then so is $\mathcal{O}$.

To see the other direction, note that if a class of programs $\mathcal{C}$ has a perfect iO (deterministic or statistical) Obf, then the class $\mathcal{C}$ can also be exactly learnt with a hint function: the hint oracle simply sends $\mathsf{Obf}(c)$ to the learner. Thus, exact learning with hint functions precisely captures the notion of proper obfuscation. ◀

It is natural to compare this result to the argument of [6] that exact learning with membership queries implies VBB obfuscation. Since information-theoretic iO is weaker than VBB obfuscation it makes sense that an easier learning task should capture it. Indeed, any program class for which equivalence queries are efficiently implementable but which are not exactly learnable with membership queries alone are not VBB obfuscatable, but they can potentially admit information-theoretic iO. In the remainder of this section we will see three such examples.

First, we will use Theorem 11 to prove a very general result, showing that any efficient learning algorithm in the online *mistake-bound model* implies canonization, as long as we can implement equivalence queries efficiently. We consider this an interesting result since canonization is, arguably, the *strongest* form of indistinguishability obfuscation. For this we must recall the definition of mistake-bound learning, introduced by [36].

▶ **Definition 12.** *Let $\mathcal{C}$ be a class of programs. The task of* mistake-bound learning *of $\mathcal{C}$ is defined as follows. There is a fixed* target program *$c \in \mathcal{C}$ that we must learn. The learner starts with an* initial hypothesis *$h_0 \in \mathcal{C}$, and then receives a stream of inputs $x_1, x_2, x_3, \ldots$ for their hypotheses. After receiving input x_i, the learner outputs the value of $h_{i-1}(x_i)$ and learns if $h_{i-1}(x_i) = c(x_i)$. Afterwards, if the learner makes a* mistake *(i.e. $h_{i-1}(x_i) \neq c(x_i)$), the learner updates their hypothesis to $h_i \in \mathcal{C}$.*

An algorithm A learns *$\mathcal{C}$ in the mistake-bound model with $M = M(n, |c|)$* mistakes *if for any $c \in \mathcal{C}$ and any ordering of inputs the algorithm A makes at most M mistakes. We say that A is a* polynomial-time *learning algorithm if A runs in polynomial-time in each learning stage, and that $\mathcal{C}$ is* efficiently learnable *if there is a polynomial-time learning algorithm A that makes at most polynomially-many mistakes.*

Littlestone proved that learning in the mistake-bound model is closely related to exact learning with equivalence and membership queries [36]. In particular, these models are essentially identical up to the efficiency of implementing equivalence queries. For our purposes we will only need one direction of this equivalence, which we record in the next theorem.

▶ **Theorem 13** ([36]). *Let $\mathcal{C}$ be a class of programs. If there is an algorithm that learns $\mathcal{C}$ in the mistake-bound model with at most M mistakes, then there is an algorithm exactly learning $\mathcal{C}$ with membership and equivalence queries that makes at most $M+1$ equivalence queries and at most $M+1$ membership queries.*

With these results in hand we prove the following theorem.

▶ **Theorem 14.** *Let $\mathcal{C}$ be a class of programs such that equivalence queries for $\mathcal{C}$ can be implemented in polynomial time. If $\mathcal{C}$ can be learned in polynomial time in the mistake-bound model with M mistakes, then $\mathcal{C}$ can be canonically obfuscated in time $O(M \cdot \mathsf{poly}(n))$.*

Proof. By Theorem 13, there is an algorithm exactly learning $\mathcal{C}$ with membership and equivalence queries that makes $M+1$ equivalence and $M+1$ membership queries. We use this to give an exact learning algorithm for $\mathcal{C}$ with a canonical hint function. Let $c \in \mathcal{C}$ be our target program. Our learning algorithm will simulate the exact learning algorithm with membership and equivalence queries. To do this, the canonical hint function $h(c, \cdot)$ will simulate the membership queries and equivalence queries on c. Both of these queries can be implemented in polynomial time by assumption, however, we must ensure that the equivalence queries are themselves canonical in that whenever we make an equivalence query to nonequivalent programs $a \not\equiv c$ the program will always return some canonical counterexample. Formally, we require the following guarantee: for any programs $a, b, d \in \mathcal{C}$, if $a \equiv b$, $c \equiv d$, and $a \not\equiv c$, then counterexamples returned by the equivalence queries $a \equiv c$ and $b \equiv d$ must be the same.

Fortunately, this is easy to ensure. We show something stronger: since the equivalence queries for $\mathcal{C}$ can be carried out in polynomial time, it follows that for any nonequivalent $a, b \in \mathcal{C}$ we can find the lexicographically-first counterexample x such that $a(x) \neq b(x)$. To see this, let a, b be non-equivalent programs and let $x_1, x_2, \dots, x_n$ be the n input variables of a and b. Given a partial restriction ρ to the input variables, let $a \restriction \rho$ be the program obtained by substituting ρ for the appropriate input variables to a. Consider the following algorithm. Initially, let $i = 1$ be a counter, let $\rho = *^n$ be the empty restriction, and repeat the following process. Query if $a \restriction (\rho \cup (x_i \leftarrow 0)) \equiv b \restriction (\rho \cup (x_i \leftarrow 0))$. If they are not equivalent, then update $i = i + 1$ and $\rho = \rho \cup (x_i \leftarrow 0)$; otherwise, update $\rho = \rho \cup (x_i \leftarrow 1)$. Once all variables are assigned it is easy to see that the (now total) assignment ρ is the lexicographically-first input x witnessing that $a(x) \neq b(x)$. By using this equivalence query algorithm in place of the original one we can make the hint function $h(c, \cdot)$ canonical, and this completes the proof. ◀

Using this theorem we can now use known results in the learning literature to canonize some well-known program classes. First, we observe that polynomial-size decision trees can be canonized in *quasipolynomial* time. This was proved directly using an ad-hoc argument in [4], but thanks to our general result it follows automatically from known learning results.

▶ **Corollary 15.** *The class of size-s decision trees on n variables can be canonized in time $n^{O(\log s)}$.*

Proof. Simon [45] proved that rank-r decision trees can be learned by a polynomial-time algorithm in the mistake-bound model with at most $M = n^{O(r)}$ mistakes, and it is known that the rank of a size-s decision tree is at most $\log s$ [18]. Furthermore, it is known that given two decision trees T_1, T_2 we can compute the decision tree $T_1 \oplus T_2$ in polynomial-time; this means that we can efficiently perform equivalence testing of decision trees. The corollary then follows immediately from Theorem 14. ◀

A second easy corollary is that 2-CNF formulas can be canonized.

▶ **Corollary 16.** *The class of* 2*-CNF formulas can be canonized in polynomial time.*

Proof. It is well known that equivalence testing of 2-CNF formulas is computable in polynomial time so we focus on constructing an efficient mistake-bound algorithm for 2-CNFs. The algorithm is just the standard "monotone disjunction" learning algorithm from learning theory. Let n be the number of variables of the target CNF formula F. Our initial hypothesis H is the trivial 2-CNF that contains every width-2 clause on n variables. Let x be the most recently arrived input on the stream. If $H(x) = 0$ but $F(x) = 1$ then we remove all clauses from H that are not satisfied; otherwise, we do not update H. Observe that $H \implies F$ at each stage of this algorithm (in other words, H contains every clause that F contains), and thus at no point will $H(x) = 1$ but $F(x) = 0$. Finally, this algorithm achieves a mistake bound of $O(n^2)$ since we start with $O(n^2)$ clauses in H and each mistake leads to the removal of at least one clause. ◀

Finally, we give an example of exact learning with hint functions from algebraic circuit complexity that does *not* use the mistake-bound model. Let $\mathbb{F}$ be a field and consider the polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$. An *arithmetic circuit* Φ is circuit with input gates labelled with variables from $x_1, \ldots, x_n$ or field elements from $\mathbb{F}$, and whose internal gates are labelled with multiplication $(\cdot)$ or addition $(+)$ operations over the field. An arithmetic circuit computes a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ in the natural way. An arithmetic circuit is a *formula* if the underlying graph is a tree, it is *read-k* if each input variable labels at most k leaves, and finally it is *multilinear* if the polynomial output by the circuit has individual degree at most 1 for every variable.

It is interesting to note that for the class of multilinear, constant-read arithmetic formulas there is a super-polynomial gap between the running time of the best known learning algorithm with only membership queries (i.e. oracle access to the formula) and a learning algorithm that is allowed to use more powerful queries. Shpilka and Volkovich proved that if Φ is a multilinear read-k formula, then given black-box access (i.e. membership queries) to any arithmetic circuit C computing Φ there is a deterministic algorithm that constructs a read-k multilinear polynomial computing Φ in time $n^{k^{O(k)} + k \log n}$ [44, Theorem 1.6]. In other words, this is an exact learning algorithm with membership queries, and we have already observed that membership queries are hint functions above.

However, more interestingly, Shpilka and Volkovich observed that knowing the *discrete partial derivatives* of the polynomial computed by Φ, there is an algorithm that can reconstruct Φ in time $n^{k^{O(k)}}$. The discrete partial derivatives are a semantic property of the polynomial Φ, and moreover they can be efficiently evaluated given an arithmetic circuit for Φ and thus they can be implemented as hint functions. We therefore have the following theorem:

▶ **Theorem 17** (cf. Theorem 1.6 in [44])**.** *The class of multilinear read-k arithmetic formulas over n variables can be properly obfuscated in deterministic time $n^{k^{O(k)}}$.*

4 Impossibility of Obfuscating 3-CNF to k-CNF

This section is dedicated to ruling out the existence of (computational) proper obfuscation for the class of 3-CNF. In fact we show that there is no obfuscation from 3-CNFs to k-CNF for any constant k (or even for slightly super-constant k, but at the cost of only ruling out obfuscation with slightly super-polynomial security). The astute reader will note by reading between the lines that we in fact show a more general statement (however we restrict our proof and formalism to $\mathcal{C}_1 = \text{3-CNF}$):

> If a program class $\mathcal{C}_1$ is expressive enough to evaluate sparse pseudorandom generators[6] or injective one-way functions, then it cannot be properly obfuscated to a class $\mathcal{C}_2$ whose dual (*c.f.* Definition 18) is PAC-learnable[7].

▶ **Definition 18** (Dual Class of Boolean Circuits). *Let $\mathcal{C}_n$ be a class of n-input boolean formulae. We define the* dual class *of $\mathcal{C}_n$ as the class of circuits taking as input a circuit in $\mathcal{C}_n$ and evaluating it on a fixed, hardcoded input $x \in \{0,1\}^n$,* i.e.

$$\mathcal{C}'_n := \left\{ \begin{array}{cccc} \psi_x : & \mathcal{C}_n & \to & \{0,1\} \\ & \phi & \mapsto & \phi(x) \end{array} : x \in \{0,1\}^n \right\}.$$

Note that $\mathcal{C}_n$ and $\mathcal{C}'_n$ can both be cast as program classes as defined in Definition 5.

Before we proceed, let us clarify that in the course of the proof we will be using the fact that the dual class is PAC-learnable from random samples with respect to *any*[8] distribution, and not just *e.g.* the uniform distribution. For this reason, the impossibility result does not *a priori* rule out the existence of obfuscation from 3-CNF to AC^0.

▶ **Theorem 19** (PAC Attack). *Assuming the existence of a one-way function, there is a universal constant $c \in (1,4]$ such that if $\mathcal{C}'_n$ (the dual class of $\mathcal{C}_n$) is $T(\cdot,\cdot,\cdot)$-PAC learnable then for any $\epsilon(\cdot), \delta(\cdot) \in (0,1)^{\mathbb{N}}$ there is no $(T(\epsilon(n^c), \delta(n^c), n^c),\ \epsilon(n^c) + \delta(n^c))$-secure* iO *scheme from n-input* 3-CNF*s to $\mathcal{C}_n$.*

Before we proceed with the proof of Theorem 19, let us observe that, because the dual class of k-CNF is $(\mathsf{k}+1)$-CNF – which has been shown to be n^{k+1}-PAC learnable by [46] – it implies the following corollaries.

▶ **Corollary 20** (PAC Attack against iO of 3-CNFs to k-CNFs). *Assuming the existence of a one-way function, there is a uniform time-$\mathcal{O}(n^{4\cdot(k+1)})$ attack with success probability $1 - 1/\Theta(n^{4\cdot(k+1)})$ against any* iO *scheme from n-input* 3-CNF*s to k-*CNF*s.*

▶ **Corollary 21** (Impossibility of Properly Obfuscating 3-CNF to $\mathcal{O}(1)$-CNF). *Let $k \geq 3$ be a constant. Assuming the existence of a one-way function, there is no* iO *scheme from n-input* 3-CNF*s to k-*CNF*s with security against uniform p.p.t. adversaries.*

---

[6] A PRG is said to be sparse if a random string is outside its image with overwhelming probability, *i.e.* if it has superlogarithmic stretch.

[7] In fact it suffices for the impossibility result to go through that $\mathcal{C}_2$ be learnable on the distribution of ciphertexts of random bits.

[8] We do not require the ability to learn with respect to any distribution *per se*, but rather only with respect to a single "esoteric" (*i.e.* not uniform or any other distribution likely to have been studied on its own) and unknown distribution.

In order to prove Theorem 19, we first show that assuming the existence of a one-way function and a secure iO scheme from 3-CNFs to $\mathcal{C}_n$, we can build a semantically secure public-key encryption scheme whose decryption circuit, once the key has been hardcoded, is in $\mathcal{C}'_n$ (the dual class of the target class $\mathcal{C}_n$ of the obfuscator) with constant probability over the randomness of the key generation algorithm. This means $\mathcal{C}'_n$ cannot be PAC-learnable without contradicting the security of the PKE, *i.e.* without contradicting the security of (the OWF or) the iO. Note that this idea of opposing "easiness of learning" to cryptographic assumptions has been exploited before, notably in [33]. Recall that semantic security is equivalent to CPA-security as shown by [26], and that CPA security implies that security must hold even if the adversary is allowed to make encryption queries. In particular the adversary can use this power to sample random plaintexts and make encryption queries in order to collect the positive samples required to PAC-learn the decryption circuit on the distribution of valid encryptions of a random bit: this contradicts security of the PKE.

The PKE scheme we present is very similar in spirit to that in [19, Fig. 1], which makes use of a sparse PRG rather than an injective one-way function. Whether one uses a sparse PRG or an injective OWF, the idea is to have the encryption process be the obfuscation of a "compute-and-compare" to guard access to a payload which is hard to retrieve without a trapdoor (the secret key). However, it may not be immediately clear how this is linked to obfuscation of 3-CNF unless the PRG or injective OWF is 3-local (which allows the "compute-and-compare" circuit to be expressed as a 3-CNF). The main difference with [19, Fig. 1] is that we add a specification of how a "compute-and-compare" formula is converted to 3-CNF using the Tseytin transformation in order to control the form of the decryption circuit. Because the Tseytin transform only preserves equisatisfiability, not equivalence, special care must be taken in order to reduce security of our PKE scheme to security of the iO for 3-CNF.

▶ **Lemma 22** (PKE with Simple Decryption from 3-CNF Obfuscation)**.** *For $n \in \mathbb{N}$, let $\mathcal{C}_n$ be a class of n-input polynomial-size boolean formulae. Let $\mathcal{C}'_n$ denote the dual class of $\mathcal{C}_n$,* i.e. $\mathcal{C}'_n := \left\{ \begin{array}{rccc} \psi_x \colon & \mathcal{C}_n & \to & \{0,1\} \\ & \phi & \mapsto & \phi(x) \end{array} : x \in \{0,1\}^n \right\}$.

Assuming the existence of an injective one-way function and of a iO[9] *scheme from n-input* 3-CNF*s to $\mathcal{C}_n$, there is a* PKE *scheme whose decryption circuit is in $\mathcal{C}'_N \cup (1 \oplus \mathcal{C}'_N)$, where N is a fixed polynomial in n which only depends on the choice of the injective one-way function and where $1 \oplus \mathcal{C}'_N := \{1 \oplus f : f \in \mathcal{C}'_N\}$ is the class of negations of $\mathcal{C}'_N$. Moreover, an explicit construction is provided in Figure 1.*

Proof. The proof of Lemma 22 is given by considering the PKE construction of Figure 1, then proving its correctness and security. The construction is in two parts. The construction relies on an "functionality-preserving conversion of compute-and-compare formulae to 3-CNF" primitive whose properties have been abstracted out in Figure 1. Then Figure 2 provides an explicit instantiation of this primitive.

[9] The theorem holds using null-iO instead of iO.

Public-Key Encryption Scheme

Tseytin-based Conversion to 3-CNF – Idealized

Notations. $\mathsf{BF}_n := \{0,1\}^{(\{0,1\}^n)}$ denotes the set of all n-input boolean formulae, and 3-CNF_n denotes the set of all 3-CNF with n inputs.

Requirements. The below conversion procedure is parametrized by a pair of boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$, and applies to boolean formulae of the form $\phi_{f,g,y,m}(x_1,\dots,x_n) := (f(x_1,\dots,x_n) == y) \wedge (g(x_1,\dots,x_n) \oplus m)$, where $y \in \{0,1\}^n$, and $m \in \{0,1\}$. It is convenient to introduce a third parameter, which depends only on f and on g: N is some polynomial in n to be made explicit.

Syntax. The conversion to 3-CNF is a pair $\pi\colon \{0,1\}^n \to \{0,1\}^N$ (the variable conversion) and $\mathsf{Cvrt}\colon \mathsf{BF}_n \to$ 3-CNF_{n+N} (the formula conversion).

Properties (Functionality-Preserving Conversion). π should be injective and independent of y and m, and $\forall x \in \{0,1\}^n, \phi_{f,g,y,m}(x) = \mathsf{Cvrt}(\phi_{f,g,y,m})(\pi(x))$.

An explicit construction of this primitive is given in Figure 2.

– –

Public-Key Encryption

Requirements and Notations.
- An injective one-way function f, and a hardcore predicate hc for f.
- A iO scheme $\mathcal{O}$ which, on input an n-input 3-CNF, outputs a circuit in $\mathcal{C}_n$.
- (Cvrt, π) is defined as above for the choice of functions $(f, g) \leftarrow (f, \mathsf{hc})$.

Syntax. The encryption scheme is a triple $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ which are defined as follows.
- $\mathsf{PKE.KeyGen}(1^n)$: Sample $x \xleftarrow{\$} \{0,1\}^n$, set $b \leftarrow \mathsf{hc}(x)$, $\mathsf{sk} \leftarrow (b, \pi(x))$, $\mathsf{pk} \leftarrow (f(x), \mathsf{pp})$ where public parameters pp contain a succinct description of π. Output $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{PKE.Enc}(\mathsf{pk}, m)$: On input the public key pk and a message $m \in \{0,1\}$,
  1. Parse pk as $\mathsf{pk} = (y, \mathsf{pp})$;
  2. Build the boolean formula $\phi \leftarrow (f(\cdot) == y) \wedge (\mathsf{hc}(\cdot) \oplus m)$;
  3. Run $\psi \xleftarrow{\$} \mathcal{O}(\mathsf{Cvrt}(\phi))$ and output ψ.
- $\mathsf{PKE.Dec}(\mathsf{sk}, c)$: On input the secret key sk and a ciphertext $c \in \mathcal{C}_n$,
  1. Parse sk as $\mathsf{sk} = (b, \alpha)$;
  2. Set $m' \leftarrow b \oplus c(\alpha)$ and output m'.

Figure 1 PKE from injective OWF and iO for 3-CNF.

Let us now show that the triple $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ as defined in Figure 1 is a semantically secure public-key encryption scheme.

- **Correctness:** If $(\mathsf{sk} = (\mathsf{hc}(x), g(x)), \mathsf{pk} = f(x)) \xleftarrow{\$} \mathsf{PKE.KeyGen}(1^n)$ and $c \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, m)$, then $\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{hc}(x) \oplus c(\pi(x)) = \mathsf{hc}(x) \oplus \mathsf{Cvrt}(\pi(x)) = \mathsf{hc}(x) \oplus ((f(x) == f(x)) \wedge (\mathsf{hc}(x) \oplus m)) = m$ (note that this statement becomes probabilistic in nature if $\mathcal{O}$ only preserves equivalence with probability less than 1).

- **Security:** Consider the following distributions:

$$\Delta_0 \coloneqq \{(\mathsf{pk}, c)\colon (\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{PKE.KeyGen}(1^n), c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, 0)\}$$
$$\Delta_0' \coloneqq \{(\mathsf{pk}, c)\colon \mathsf{pk} \leftarrow (f(x), \mathsf{pp}) \text{ where } x \xleftarrow{\$} \{0,1\}^n \text{ s.t. } \mathsf{hc}(x) = 0, c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, 0)\}$$
$$\Delta_1 \coloneqq \{(\mathsf{pk}, c)\colon (\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{PKE.KeyGen}(1^n), c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, 1)\}$$
$$\Delta_1' \coloneqq \{(\mathsf{pk}, c)\colon \mathsf{pk} \leftarrow (f(x), \mathsf{pp}) \text{ where } x \xleftarrow{\$} \{0,1\}^n \text{ s.t. } \mathsf{hc}(x) = 1, c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, 1)\}$$

For $\sigma \in \{0,1\}$, the difference between Δ_σ and Δ_σ' is that the secret key in the latter case is sampled (*e.g.* through rejection sampling) such that $\mathsf{hc}(x) = \sigma$. But since hc is a hardcore bit, $\Delta_0 \overset{c}{\approx} \Delta_0'$ and $\Delta_1 \overset{c}{\approx} \Delta_1'$. Indeed, $\{(f(x), \mathsf{hc}(x))\colon x \xleftarrow{\$} \{0,1\}^n\} \overset{c}{\approx} \{(f(x), 1 - \mathsf{hc}(x))\colon x \xleftarrow{\$} \{0,1\}^n\}$ (the definition of a hardcore bit) implies $\{f(x)\colon x \xleftarrow{\$} \{0,1\}^n\} \overset{c}{\approx} \{f(x)\colon x \xleftarrow{\$} \{0,1\}^n, \mathsf{hc}(x) = \sigma\}$. In turn, this implies that $\Delta_\sigma \overset{c}{\approx} \Delta_\sigma'$ for $\sigma \in \{0,1\}$.

By security of the iO scheme $\mathcal{O}$, $\Delta_0' \overset{c}{\approx} \Delta_1'$. Indeed since f is injective, if $\mathsf{hc}(x) = m$ then $(f(\cdot) == f(x)) \wedge (\mathsf{hc}(\cdot) \oplus m)$, and therefore $\mathsf{Cvrt}((f(\cdot) == f(x)) \wedge (\mathsf{hc}(\cdot) \oplus m))$, is unsatisfiable. It follows that $\{(\mathsf{pk}, \mathcal{O}(\mathsf{Cvrt}((f(\cdot) == f(x)) \wedge (\mathsf{hc}(\cdot) \oplus 0))))\colon \mathsf{pk} \leftarrow (f(x), \mathsf{pp}) \text{ where } x \xleftarrow{\$} \{0,1\}^n \text{ s.t. } \mathsf{hc}(x) = 0\} \overset{c}{\approx} \{(\mathsf{pk}, \mathcal{O}(\mathsf{Cvrt}((f(\cdot) == f(x)) \wedge (\mathsf{hc}(\cdot) \oplus 1))))\colon \mathsf{pk} \leftarrow (f(x), \mathsf{pp}) \text{ where } x \xleftarrow{\$} \{0,1\}^n \text{ s.t. } \mathsf{hc}(x) = 1\}$, *i.e.* $\Delta_0' \overset{c}{\approx} \Delta_1'$.

Therefore, $\Delta_0 \overset{c}{\approx} \Delta_0' \overset{c}{\approx} \Delta_1' \overset{c}{\approx} \Delta_1$, which concludes the proof.

Tseytin-based Conversion to 3-CNF – Construction

Notations. $\mathsf{BF}_n \coloneqq \{0,1\}^{(\{0,1\}^n)}$ denotes the set of all n-input boolean formulae, and 3-CNF$_n$ denotes the set of all 3-CNF with n inputs.

Requirements. The below conversion procedure is parametrized by a pair of boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$, and applies to boolean formulae of the form $\phi_{f,g,y,m}(x_1, \ldots, x_n) \coloneqq (f(x_1, \ldots, x_n) == y) \wedge (g(x_1, \ldots, x_n) \oplus m)$, where $y \in \{0,1\}^n$, and $m \in \{0,1\}$. It is convenient to introduce a third parameter, which depends only on f and on g: N is some polynomial in n to be made explicit.

Syntax. The conversion to 3-CNF is a pair $\pi\colon \{0,1\}^n \to \{0,1\}^N$ (the variable conversion) and $\mathsf{Cvrt}\colon \mathsf{BF}_n \to$ 3-CNF$_{n+N}$ (the formula conversion) which is defined as follows.

- Cvrt : Consider the n-variable boolean formula $\phi_{f,g,y,m}$. If $f(\cdot)_{|i}$ denotes the formula equal to the i^{th} output of f, $\phi_{f,g,y,m}$ can be rewritten as follows: $\phi_{y,m}(\cdot) \equiv (f(\cdot)_{|1} == y_1) \wedge \cdots \wedge (f(\cdot)_{|\lambda} == y_\lambda) \wedge (g(\cdot) \oplus m)$.
  Consider the formula $\phi'_{f,g,y,m}(x_1, \ldots, x_{n+\lambda+1})$ defined by:

$$\begin{aligned}\phi'_{y,m}(x_1, \ldots, x_{n+\lambda+1}) \coloneqq &(x_{n+1} \leftrightarrow f(x_1, \ldots, x_n)_{|1}) \wedge (x_{n+1} == y_1) \ \wedge \ \cdots\\ &(x_{n+\lambda} \leftrightarrow f(x_1, \ldots, x_n)_{|\lambda}) \wedge (x_{n+\lambda} == y_\lambda) \wedge\\ &(x_{n+\lambda+1} \leftrightarrow g(x_1, \ldots, x_n)) \wedge (x_{n+\lambda+1} \oplus m).\end{aligned}$$

  Using fresh variables $x_{n+\lambda+2}, x_{n+\lambda+3} \ldots$ each formula of the form $(x_{n+i} \leftrightarrow f(x_1, \ldots, x_n)_{|i})$ or $(x_{n+\lambda+1} \leftrightarrow g(x_1, \ldots, x_n))$ can be converted to 3-CNF using the Tseytin transformation (without further specification). Crucially, these conversions are independent of y, m and depend only on f, g. In turn, the $(x_{n+i} == y_i)$ and $(x_{n+\lambda+1} \oplus m)$ are simple literals, depending on y and m. Plugging all these 3-CNF into $\phi'_{f,g,y,m}$ yields an N-variable formula $\mathsf{Cvrt}(\phi_{f,g,y,m})$ where N is some fixed polynomial in n which depends only on f and g (and on the – canonical – choice of variable switches performed by the Tseytin transformation).

- $\pi\colon \{0,1\}^n \to \{0,1\}^N : g(\alpha_1, \ldots, \alpha_n)$ is defined as follows:
  - the first n outputs are equal to $(\alpha_1, \ldots, \alpha_n)$;
  - for $i = 1 \ldots \lambda$, the $(n+i)^{\text{th}}$ output is equal to $f(\alpha_1, \ldots, \alpha_n)_{|\lambda}$;
  - the $(n+\lambda+1)^{\text{th}}$ output is equal to $g(\alpha_1, \ldots, \alpha_n)$;
  - similarly the last $N - (n+\lambda)$ output values are functions of $(\alpha_1, \ldots, \alpha_n)$ given by the variable changes which introduced $(x_{n+\lambda+2}, \ldots, x_N)$ as functions of $(x_1, \ldots, x_n)$ in the definition of Cvrt.

Note that by construction, π is injective (and independent of y, m) and $\forall x \in \{0,1\}^n, \phi_{f,g,y,m}(x) = \mathsf{Cvrt}(\phi_{f,g,y,m})(\pi(x))$.

Figure 2 Construction of the Tseytin-based Conversion to 3-CNF.

◀

5 Separating Forms of Information-Theoretic iO

In this section we establish a separation between two flavors of information-theoretic obfuscation: perfect (randomized) iO and deterministic iO. Before we discuss our separation, it is first worth noting what makes this an interesting and challenging problem. In order to do this, let us begin by considering the following "toy" version of obfuscation where instead of trying to obfuscate *programs*, we simply try to obfuscate some *arbitrary equivalence relation*. More formally, consider a sequence of equivalence relations $R = \{R_n\}_n$ where $R_n \subseteq \{0,1\}^n \times \{0,1\}^n$ for each n. We can define analogous notions of "obfuscation" for R: for instance, a *canonizer* would be an algorithm Can such that for all $n \in \mathbb{N}$ and all $x \in \{0,1\}^n$ we have that $(x, \mathsf{Can}(x)) \in R_n$ and if $(x, y) \in R_n$ then $\mathsf{Can}(x) = \mathsf{Can}(y)$. Similarly, a *perfect obfuscator* iO_R would be a p.p.t. algorithm such that $\mathsf{iO}_R(x)$ outputs the uniform distribution over all y such that $(x, y) \in R_n$. From these abstract definitions we can recover the standard notions of obfuscation simply by letting R be $\equiv_{\mathcal{C}}$, the standard notion of equivalence for a program class $\mathcal{C}$.

In this more abstract setting it turns out to be quite easy to separate forms of obfuscation. For a simple example, we can roughly sketch how to separate perfect iO from canonization (a formal argument can be found in Appendix A). Consider any randomly self-reducible function $f : \{0,1\}^* \to \{0,1\}^*$. Roughly speaking[10], this means that there is a p.p.t. algorithm σ such that for all n and for all $x \in \{0,1\}^n$, $f(x) = f(\sigma(x))$ and furthermore $\sigma(x)$ is distributed uniformly at random over the set $\{y \in \{0,1\}^n : f(x) = f(y)\}$. Given such a function we can define the equivalence relation R by $(x, y) \in R$ iff $f(x) = f(y)$, and σ is a perfect iO for R. However, if checking that $f(x) = f(y)$ is hard deterministically then we cannot expect to be able to canonize the relation R. In Appendix A we construct such a function using ElGamel encryption, thus "separating" these abstract forms of perfect iO and canonization under the DDH assumption.

[10] The definition of random self reducibility is actually a bit different in multiple ways, e.g. it is usually defined as a non-uniform reduction; an input x can be mapped into polynomially many "shares", etc. Here we explain an idea, and so for simplicity do not define random self reducibility in its full generality.

So, what makes separating the “standard” notions of iO more difficult? In short, it is due to the fact that the elements of the underlying equivalence relation must be programs from some class, and two programs are equivalent if and only if they compute the same underlying boolean functions. In our constructions below we will show already that it is hard to distinguish between program classes which can only compute two “functions”: a point function on the one hand, and the trivially 0 function on the other.

5.1 Separating Perfect and Deterministic Obfuscation in the CRS Model

We start by introducing the notion of iO in the CRS model in Definition 23, then state our separation result in Theorem 24.

In the CRS model we allow program classes to be defined as a function of the CRS. Therefore, concretely, the separation is of the form: there exists a CRS generation algorithm Setup which yields a world in which there is a program class for which there exists a perfect iO scheme, but no deterministic iO. In fact, we show a slightly stronger statement: there is a proper perfect iO, but no (even improper) deterministic iO.

▶ **Definition 23** (iO in the CRS model). *A computational (resp. statistical, perfect) iO scheme in the CRS model for $\mathcal{P} = (\mathsf{Eval}_{\mathcal{P}}, \mathsf{Size}_{\mathcal{P}})$ by $\mathcal{P}' = (\mathsf{Eval}_{\mathcal{P}'}, \mathsf{Size}_{\mathcal{P}'})$ is a pair of p.p.t. algorithms* $(\mathsf{Setup}, \mathsf{iO})$ *satisfying the following requirements:*

- ***Syntax:***
  - Setup : *On input a security parameter 1^λ,* Setup *outputs a common reference string* CRS.
  - iO : *On input a security parameter 1^λ, a common reference string* CRS*, and a program P,* iO *outputs a program P'.*
- ***Completeness:*** *For every $\lambda \in \mathbb{N}$, common reference string* $\mathsf{CRS} \in \mathsf{Supp}(\mathsf{Setup}(1^\lambda))$*, program P, and input x, the following holds:*

$$\Pr\left[\mathsf{Eval}_{\mathcal{P}'}((\mathsf{CRS}, P'), x) = \mathsf{Eval}_{\mathcal{P}}((\mathsf{CRS}, P), x) \colon P' \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{CRS}, P)\right] = 1$$

- ***Indistinguishability:*** *For every polynomial-time nonuniform (resp. unbounded) adversary A such that $A(\mathsf{CRS})$ outputs a pair of programs P^0, P^1 where $(CRS, P^0) \equiv (CRS, P^1)$ and $\mathsf{Size}(P^0) = \mathsf{Size}(P^1)$, the following distribution ensembles are computationally (resp. statistically, perfectly) indistinguishable:*

$$\{\mathsf{iO}(1^\lambda, \mathsf{CRS}, P^0)\}_{\lambda\in\mathbb{N}} \qquad \textit{and} \qquad \{\mathsf{iO}(1^\lambda, \mathsf{CRS}, P^1)\}_{\lambda\in\mathbb{N}}.$$

  Here the probability space is over CRS *generated by* $\mathsf{Setup}(1^\lambda)$ *and P^0, P^1 generated by* $A(\mathsf{CRS})$.

▶ **Theorem 24** (Conditional Separation between Perfect and Deterministic Obfuscation with Setup). *Assuming the existence of one-way functions and VBB obfuscation for a specific circuit class (the class $\{\mathsf{Helper}_k \colon k \in \{0,1\}^\lambda\}$, as defined in Figure 3), there is a program class for which there exists perfect but not deterministic iO in the CRS model.*

The rest of this section is dedicated to proving Theorem 24, which we do by providing a Setup algorithm (Figure 4) and defining a program class $\mathcal{P}_{\mathsf{sep}}$ (Figure 5) for which there exists a perfect iO scheme (with respect to Setup) but no deterministic iO scheme (again, with respect to Setup). Abstracting out all semantic details for now, for each $\mathsf{CRS} \in \mathsf{Supp}(\mathsf{Setup})$ there are 2^n different programs in $\mathcal{P}_{\mathsf{sep}}(\mathsf{CRS})$, each described by a $2n$-bit string. These programs are partitioned into $2^{n/2}$ different equivalence classes of size $2^{n/2}$ each. Specifically, the CRS generated by Setup is the description of a VBB obfuscated program we call the "helper program" Helper, defined in Figure 3, which can be seen as succinctly described oracle, answering two types of queries:

1. *Evaluation queries:* These allow us to define $\mathcal{P}_{\mathsf{sep}}$; evaluating a program in $\mathcal{P}_{\mathsf{sep}}$ is done by making a corresponding evaluation query to the helper program.
2. *Jump queries:* These are used to facilitate perfect (but not deterministic) obfuscation.

The helper program has hardcoded the key to a strong pseudo-random permutation, mapping the 2^n valid program descriptions to the nodes of $2^{n/2}$ different length-$2^{n/2}$ directed cycles in such a way that each cycle corresponds to an equivalence class of $\equiv_{\mathcal{P}_{\mathsf{sep}}}$ (*i.e.* two programs are functionally equivalent if any only if they label nodes in the same cycle): each program is a point function, equal to 1 on the index of the cycle to which it belongs, and 0 elsewhere. We are now ready to clarify what "jump queries" are: on input the description of a program, and $d \in \{0,1\}^{n/2}$, the helper program returns the label of the d^{th} successor on the cycle of the node labeled $\widetilde{P}$. This way, the CRS can be used to achieve perfect iO in a straightforward way: on input a program P, sample $d \xleftarrow{\$} \{0,1\}^{n/2}$ and output the d^{th} successor of P on the cycle.

Program Helper_k

Parameters: Helper_k has hardcoded the security parameter λ, an input length parameter n, a strong PRP $F(\cdot,\cdot)\colon \{0,1\}^{\lambda} \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ which admits an efficiently computable "reverse permutation family"[a] $G(\cdot,\cdot)\colon \{0,1\}^{\lambda} \times \{0,1\}^{2n} \to \{0,1\}^{2n}$, and a key $k \in \{0,1\}^{\lambda}$. These parameters define two functions f and g:

$$\begin{array}{rccc} f\colon & \{0,1\}^n & \to & \{0,1\}^{2n} \\ & x & \mapsto & F(k, x \| 0^n) \end{array} \qquad \begin{array}{rccl} g\colon & \{0,1\}^{2n} & \to & \{0,1\}^n \cup \{\perp\} \\ & y & \mapsto & \begin{cases} [G(k,y)]_{1\ldots n} & \text{if } [G(k,y)]_{n+1\ldots 2n} = 0^n \\ \perp & \text{otherwise} \end{cases} \end{array}$$

// When convenient we will see g as a function from $\{0,1\}^{2n}$ to $\{0,1\}^n$ but which is only defined on $f(\{0,1\}^n) \subseteq \{0,1\}^{2n}$.

Evaluation: Helper_k tries to match its input with one of the following patterns:

```
■ On input ("jump", $\widetilde{P}$, $d$), where $\widetilde{P} \in \{0,1\}^{2n}$ and $d \in \{0,1\}^{n/2}$:
    1. Check that $g(\widetilde{P}) \neq \perp$ and output $\perp$ otherwise.
    2. Parse $g(\widetilde{P})$ as $(x_0, x_1)$, with $|x_0| = |x_1| = n/2$ .
    3. Output $f(x_0 \| (x_1 \oplus d))$ .
■ On input ("eval", $\widetilde{P}$, $\alpha$), where $\widetilde{P} \in \{0,1\}^{2n}$ and $\alpha \in \{0,1\}^{n/2}$:
    1. Check that $g(\widetilde{P}) \neq \perp$ and output $\perp$ otherwise.
    2. Compute $x_0 \leftarrow [g(\widetilde{P})]_{1\ldots\frac{n}{2}}$ .
    3. Output $(x_0 == \alpha)$ .
■ On input ("IsWellDefined", $\widetilde{P}$), where $\widetilde{P} \in \{0,1\}^{2n}$:
        Output $\neg(g(\widetilde{P}) == \perp)$ .
```

Helper_k outputs $\perp$ if it receives an incorrectly formatted input.

[a] For every choice of key $k \in \{0,1\}^{\lambda}$, $G(k,\cdot)$ is the inverse of $F(k,\cdot)$, *i.e.* $\forall k \in \{0,1\}^{\lambda}, \forall x \in \{0,1\}^{2n}, F(k, G(k,x)) = G(k, F(k,x)) = x$.

Figure 3 The helper program defining the evaluation of programs in $\mathcal{P}_{\mathsf{sep}}$.

Observe that Helper_k is parameterized by a strong PRP which can be efficiently inverted given the key. Such a permutation exists under the assumption there exists a one-way function. By combining the seminal results of [24, 23, 37], one-way functions (constructively) imply strong pseudorandom permutations. By inspection of this construction of a PRP, we can observe that the resulting PRP has the desired properties. Indeed, the permutation is defined as the sequential composition of four rounds of the Feistel network, where a single round is the application of $\mathsf{rd}_k\colon (x_0, x_1) \mapsto (x_1, x_0 \oplus F(k, x_1))$ where $|x_0| = |x_1|$ and with $F(\cdot,\cdot)$ being a pseudorandom function, which can be evaluated efficiently by definition. Each round can be inverted by the application of $\mathsf{rd}_k^{-1}\colon (y_0, y_1) \mapsto (y_1 \oplus F(k, y_0), y_0)$ and therefore the reverse permutation can also be efficiently computed given k as $(\mathsf{rd}_k^{-1})^4$.

Program Setup

Parameters: Setup is parameterized by a special-purpose virtual black-box obfuscation scheme VBB for the class $\{\mathsf{Helper}_k\}_{k\in\{0,1\}^\lambda}$.

Evaluation: On input the security parameter 1^λ, Setup samples a key $k \xleftarrow{\$} \{0,1\}^\lambda$, computes $H \xleftarrow{\$} \mathsf{VBB}(\mathsf{Helper}_k)$, sets $\mathsf{CRS} \leftarrow \widetilde{H}$ where $\widetilde{H}$ is a description of H, and outputs CRS.

Figure 4 The setup procedure with respect to which there exists perfect but not deterministic iO for $\mathcal{P}_{\mathsf{sep}}$.

Program $\mathcal{P}_{\mathsf{sep}}$

Parameters: Setup is defined as in Figure 4. $\mathcal{P}_{\mathsf{sep}}$ is parameterized by some hardcoded value α, an input size n, and a string CRS of polynomial size.

Evaluation: On input $((\mathsf{CRS}, P), x)$ where $\widetilde{P} \in \{0,1\}^{2n}$ and $x \in \{0,1\}^{n/2}$, $\mathcal{P}_{\mathsf{sep}}$ first tries to parse CRS as $\mathsf{CRS} = \widetilde{H}$, where $\widetilde{H}$ is the description of some program H, and outputs $\perp$ if the parsing fails; $\mathcal{P}_{\mathsf{sep}}$ outputs $H(\text{"eval"}, \widetilde{P}, x)$.

Figure 5 The program class $\mathcal{P}_{\mathsf{sep}}$ leading to a (conditional) separation between perfect and deterministic iO in the CRS model.

▶ **Lemma 25** (Existence of perfect iO for $\mathcal{P}_{\mathsf{sep}}$). *The procedure* iO *of Figure 6 is a perfect iO scheme for $\mathcal{P}_{\mathsf{sep}}$.*

Proof. Completeness follows from the fact that $\forall x_0, x_1, d \in \{0,1\}^{n/2}, [\pi^{-1}(\pi(x_0 \| (x_1 \oplus d) \| 0^n))]_{1\ldots n/2} = x_0$. Perfect indistinguishability follows from the fact that for all $x_0, x_1 \in \{0,1\}^{n/2}$, if d is sampled uniformly at random in $\{0,1\}^{n/2}$ then $\pi(x_0 \| (x_1 \oplus d) \| 0^n)$ follows the uniform distribution on $\{\pi(x_0 \| \alpha \| 0^n)\colon \alpha \in \{0,1\}^{n/2}\}$. ◀

Obfuscator Perfect Indistinguishability Obfuscation for $\mathcal{P}_{\mathsf{sep}}$

Setup: Setup is defined as in Figure 4.

iO: On input a security parameter 1^λ, a common reference string CRS, a program description $\widetilde{P} \in \{0,1\}^{2n}$ from the class $\mathcal{P}_{\mathsf{sep}}$,

1. Parse the CRS to retrieve the description of a program H which, on input a program in $\mathcal{P}_{\mathsf{sep}}$ outputs a tuple of $n/2$ programs in $\mathcal{P}_{\mathsf{sep}}$ (and output $\perp$ if the parsing fails).
2. If $\neg H(\text{“}\texttt{IsWellDefined}\text{”}, \widetilde{P})$ then output $\perp$.
3. Sample $d \xleftarrow{\$} \{0,1\}^{n/2}$, and compute $\widetilde{P}' \leftarrow H(\text{“}\texttt{Jump}\text{”}, \widetilde{P}, d)$.
4. Output $\widetilde{P}_{\mathsf{tmp}}$.

Figure 6 Perfectly secure iO scheme for program class $\mathcal{P}_{\mathsf{sep}}$.

▶ **Lemma 26** (Impossibility of deterministic iO for $\mathcal{P}_{\mathsf{sep}}$). *There is no deterministic iO scheme for $\mathcal{P}_{sep}$.*

Proof. In a nutshell, the proof boils down to a reduction to the discrete logarithm problem (in cyclic groups) in Shoup's generic group model, which was shown in [43] to be impossible. At a high-level, the sequence of hybrids is the following. First, observe that a deterministic obfuscation scheme Obf (which must necessarily have perfect completeness and security) can be used to perfectly decide equivalence of two programs. This means, by security of the VBB obfuscation scheme, that this predicate of "functional equivalence" can also be decided efficiently given only oracle access to the helper program. The next step is to show that, by security of the strong PRP, Obf can also be used to decide equivalence when interacting with an idealized helper oracle, where the PRP is replaced with a truly random permutation. While there is *a priori* no reason why Obf should be an obfuscation scheme for $\mathcal{P}_{\mathsf{sep}}$ defined with respect to the ideal oracle (as this is not the same class as $\mathcal{P}_{\mathsf{sep}}$ defined with respect to "real" helper oracle), we show that it must still be able to decide equivalence of programs in an average-case sense (over the randomness of the ideal oracle and that of the choice of programs). However, because the only source of randomness for Obf is provided by the answers to the queries it makes, it must behave the same way for two different oracles whose answers match on those queries in particular. By a counting argument (and using the vertex transitivity[11] of a union of cycles), it is not hard to see that unless Obf is able (in the average-case) to build a sequence of jumps from its input to its output it will be incorrect in deciding equivalence with non-negligible probability. Because it knows the distances of the jumps, it is able to determine the relative distance on the cycle between its input and the program that it outputs. Therefore, running Obf on two random equivalent programs allows us to determine their relative distance too, by transitivity. The final step is to show that this is impossible, by a reduction to the discrete logarithm problem in Shoup's generic group model [43].

[11] A graph is vertex-transitive if its automorphism group acts transitively on its vertices. Intuitively, this means that there is no structural (*i.e.* "without using the labels") way to distinguish vertices of the graph.

We now formalize the steps sketched above. By contradiction, assume there is a deterministic iO scheme Obf for $\mathcal{P}_{\mathsf{sep}}$. By completeness of Obf and VBB,

$$\Pr\left[\begin{array}{rl} k \xleftarrow{\$} & \{0,1\}^{\lambda} \\ \mathsf{CRS} \xleftarrow{\$} & \mathsf{VBB}(\mathsf{Helper}_k) \\ x_0, x_1, x_1' \xleftarrow{\$} & \{0,1\}^{n/2} \\ \widetilde{P} \leftarrow & F(k, x_0 \| x_1 \| 0^n) \\ \widetilde{P}' \leftarrow & F(k, x_0 \| x_1' \| 0^n) \end{array} : \mathsf{Obf}(1^{\lambda}, \mathsf{CRS}, \widetilde{P}) = \mathsf{Obf}(1^{\lambda}, \mathsf{CRS}, \widetilde{P}')\right] = 1 \ . \quad (1)$$

By security of VBB there exists a simulator $\mathcal{S}$ and a negligible function negl such that:

$$\Pr\left[\begin{array}{rl} k \xleftarrow{\$} & \{0,1\}^{\lambda} \\ x_0, x_1, x_1' \xleftarrow{\$} & \{0,1\}^{n/2} \\ \widetilde{P} \leftarrow & F(k, x_0 \| x_1 \| 0^n) \\ \widetilde{P}' \leftarrow & F(k, x_0 \| x_1' \| 0^n) \end{array} : \mathcal{S}^{\mathsf{Helper}_k}(1^{\lambda}, \widetilde{P}) = \mathcal{S}^{\mathsf{Helper}_k}(1^{\lambda}, \widetilde{P}')\right] \geq 1 - \operatorname{negl}(n) \ . \quad (2)$$

The next step is to move the problem to an ideal world for ease of analysis. Consider the family of oracles $\{\mathsf{IdealHelper}_{\pi}\}_{\pi \in \mathfrak{S}(\{0,1\}^{2n})}$ as defined in Figure 7, where $\mathsf{IdealHelper}_{\pi}$ acts exactly as Helper_k except that the strong pseudorandom permutation $F(k,\cdot)$ is replaced with a truly random permutation $\pi \xleftarrow{\$} \mathfrak{S}(\{0,1\}^{2n})$. Because, by construction, the only difference between Helper_k and $\mathsf{IdealHelper}_{\pi}$ is that where the former uses the permutation $F(k,\cdot)$ (with its inverse $G(k,\cdot)$) the latter uses π (and π^{-1}), they can be described as a single algorithm[12] $\mathsf{H}^{\mathcal{O}_1,\mathcal{O}_2}$ where $\mathsf{H}^{F_k, F_k^{-1}} = \mathsf{Helper}_k$ and $\mathsf{H}^{\pi,\pi^{-1}} = \mathsf{IdealHelper}_{\pi}$.

Oracle $\mathsf{IdealHelper}_{\pi}$

Parameters: $\mathsf{IdealHelper}_{\pi}$ has hardcoded the security parameter λ, an input length parameter n. It is assumed that the oracle has sampled a truly random permutation $\pi\colon \{0,1\}^{2n} \to \{0,1\}^{2n}$. These parameters define two functions f and g:

$$\begin{array}{rccc} f\colon & \{0,1\}^n & \to & \{0,1\}^{2n} \\ & x & \mapsto & \pi(x \| 0^n) \end{array} \qquad \begin{array}{rccl} g\colon & \{0,1\}^{2n} & \to & \{0,1\}^n \cup \{\perp\} \\ & y & \mapsto & \begin{cases} [\pi^{-1}(y)]_{1\ldots n} & \text{if } [\pi^{-1}(y)]_{n+1\ldots 2n} = 0^n \\ \perp & \text{otherwise} \end{cases} \end{array}$$

// When convenient we will see g as a function from $\{0,1\}^{2n}$ to $\{0,1\}^n$ but which is only defined on $f(\{0,1\}^n) \subseteq \{0,1\}^{2n}$.

Evaluation: $\mathsf{IdealHelper}_{\pi}$ tries to match the query with one of the following patterns:

- On input (“`jump`”, $\widetilde{P}, d$), where $\widetilde{P} \in \{0,1\}^{2n}$ and $d \in \{0,1\}^{n/2}$:
  1. Check that $g(\widetilde{P}) \neq \perp$ and output $\perp$ otherwise.
  2. Parse $g(\widetilde{P})$ as (x_0, x_1), with $|x_0| = |x_1| = n/2$.
  3. Output $f(x_0 \| (x_1 \oplus d))$.
- On input (“`eval`”, $\widetilde{P}, \alpha$), where $\widetilde{P} \in \{0,1\}^{2n}$ and $\alpha \in \{0,1\}^{n/2}$:
  1. Check that $g(\widetilde{P}) \neq \perp$ and output $\perp$ otherwise.
  2. Compute $x_0 \leftarrow [g(\widetilde{P})]_{1\ldots\frac{n}{2}}$.
  3. Output $(x_0 == \alpha)$.

[12] With the notations of Figures 3 and 7: The code of Helper_k and $\mathsf{IdealHelper}$ is the exact same up to the definition of f and g, and these functions can be re-written as making oracle access to $(\mathcal{O}_1, \mathcal{O}_2)$ where $(\mathcal{O}_1, \mathcal{O}_2)$ is $(F(k,\cdot), G(k,\cdot))$ or (π, π^{-1}).

- On input ("`IsWellDefined`", $\widetilde{P}$), where $\widetilde{P} \in \{0,1\}^{2n}$:
  Output $\neg(g(\widetilde{P}) == \perp)$.

$\mathsf{IdealHelper}_\pi$ outputs $\perp$ if it receives an incorrectly formatted input.

Figure 7 The oracle $\mathsf{IdealHelper}_\pi$ is an idealized version of the helper program Helper_k, where the PRP with key hardcoded $F(k,\cdot)$ is replaced with a truly random permutation.

Now consider the following algorithm $\mathcal{D}$, interacting with a pair of oracles $(\mathcal{O}_1, \mathcal{O}_2)$ (which, skipping ahead, should be thought of as as being either of the form (F_k, F_k^{-1}) or (π, π^{-1})):

1. Sample $x_0, x_1, x_1' \xleftarrow{\$} \{0,1\}^{n/2}$
2. Set $\widetilde{P} \leftarrow \mathcal{O}_1(x_0 \| x_1 \| 0^n)$ and $\widetilde{P}' \leftarrow \mathcal{O}_1(x_0 \| x_1' \| 0^n)$
3. Run $y \leftarrow \mathcal{S}^{H^{\mathcal{O}_1,\mathcal{O}_2}}(1^\lambda, \widetilde{P})$ and $y' \leftarrow \mathcal{S}^{H^{\mathcal{O}_1,\mathcal{O}_2}}(1^\lambda, \widetilde{P}')$
4. Output $y == y'$.

By security of the strong PRP F, $\mathcal{D}$ cannot distinguish with better than negligible probability between oracle access to (F_k, F_k^{-1}) and (π, π^{-1}) over the randomness of $k \xleftarrow{\$} \{0,1\}^\lambda$ and $\pi \xleftarrow{\$} \mathfrak{S}(\{0,1\}^{2n})$. Therefore there exists a negligible function negl' such that

$$\Big|\Pr_{k \xleftarrow{\$} \{0,1\}^\lambda, \mathcal{D}}[\mathcal{D}^{F_k, F_k^{-1}}(1^n) = 1] - \Pr_{\pi \xleftarrow{\$} \mathfrak{S}(\{0,1\}^{2n}), \mathcal{D}}[\mathcal{D}^{\pi,\pi^{-1}}(1^n) = 1]\Big| \le \mathrm{negl}'(n) \,. \tag{3}$$

Combining Equations (2) and (3) yields $\Pr_{\pi \xleftarrow{\$} \mathfrak{S}(\{0,1\}^{2n}), \mathcal{D}}[\mathcal{D}^{\pi,\pi^{-1}}(1^n) = 1] \ge 1 - \mathrm{negl}(n) - \mathrm{negl}'(n)$.

In particular, this means that over the randomness of $\pi \xleftarrow{\$} \mathfrak{S}(\{0,1\}^{2n})$ (*i.e.* that of $\mathsf{CRS} \xleftarrow{\$} \mathsf{Setup}$), and for a random pair of equivalent programs P_1 and P_2, $\mathcal{S}^{\mathsf{IdealHelper}_\pi}(P_1)$ is equal to $\mathcal{S}^{\mathsf{IdealHelper}_\pi}(P_2)$ with all but negligible probability. Therefore there is an efficient algorithm which (over the randomness of π), and given two random equivalent programs in $\mathcal{P}_{\mathsf{sep}}$ is able to determine their relative distance is the cycle.

Recall that in Shoup's generic group model [43], a generic algorithm for solving the discrete logarithm problem in $\mathbb{Z}_N$ on S is given the access to group elements via random labels, and may send a "group operation" oracle the labels for a and b and recover the label for $a+b$. A generic algorithm can run the aforementioned "relative distance algorithm", using fast exponentiation to emulate the $\mathsf{IdealHelper}$ queries using the "group operation" oracle. Therefore, there exists a generic algorithm which can determine the discrete logarithm of a random group element. By [43] this is impossible, which contradicts the existence of a deterministic iO scheme for $\mathcal{P}_{\mathsf{sep}}$. ◀

References

1 Shweta Agrawal, Yuval Ishai, Eyal Kushilevitz, Varun Narayanan, Manoj Prabhakaran, Vinod M. Prabhakaran, and Alon Rosen. Secure computation from one-way noisy communication, or: Anti-correlation via anti-concentration. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021 – 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 124–154. Springer, 2021. doi:10.1007/978-3-030-84245-1_5.

2 W. Erik Anderson. On the secure obfuscation of deterministic finite automata. Cryptology ePrint Archive, Report 2008/184, 2008. URL: https://eprint.iacr.org/2008/184.

3 Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988. doi:10.1023/A:1022821128753.

4 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, Gaurav Rattan, and Yadu Vasudev. On the isomorphism problem for decision trees and decision lists. *Theoretical Computer Science*, 590:38–54, 2015.

5 Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 26–51, San Diego, CA, USA, February 24–26 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54242-8_2.

6 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44647-8_1.

7 James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. New techniques for obfuscating conjunctions. In *EUROCRYPT*, pages 636–666. Springer, 2019.

8 Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. A simple obfuscation scheme for pattern-matching with wildcards. In *CRYPTO*, pages 731–752. Springer, 2018.

9 Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography – 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 662–693. Springer, 2017. doi:10.1007/978-3-319-70503-3_22.

10 Zvika Brakerski, Christina Brzuska, and Nils Fleischhacker. On statistically secure obfuscation with approximate correctness. In *Annual International Cryptology Conference*, pages 551–578. Springer, 2016.

11 Zvika Brakerski and Guy N. Rothblum. Obfuscating conjunctions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013 – 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 416–434. Springer, 2013. doi:10.1007/978-3-642-40084-1_24.

12 Zvika Brakerski and Guy N. Rothblum. Black-box obfuscation for d-CNFs. In Moni Naor, editor, *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science*, pages 235–250, Princeton, NJ, USA, January 12–14 2014. Association for Computing Machinery. doi:10.1145/2554797.2554820.

13 Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Obfuscating conjunctions under entropic ring LWE. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 147–156. ACM, 2016. doi:10.1145/2840728.2840764.

14 Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1997. doi:10.1007/BFb0052255.

15 Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 72–89, Zurich, Switzerland, February 9–11 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-11799-2_5.

16 David Bruce Cousins, Giovanni Di Crescenzo, Kamil Doruk Gür, Kevin King, Yuriy Polyakov, Kurt Rohloff, Gerard W Ryan, and Erkay Savas. Implementing conjunction obfuscation under entropic ring LWE. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 354–371. IEEE, 2018.

17 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284, New York, NY, USA, March 4–7 2006. Springer, Heidelberg, Germany. `doi:10.1007/11681878_14`.

18 Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

19 Pooya Farshim, Georg Fuchsbauer, and Alain Passelègue. Simpler constructions of asymmetric primitives from obfuscation. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology – INDOCRYPT 2020: 21st International Conference in Cryptology in India*, volume 12578 of *Lecture Notes in Computer Science*, pages 715–738, Bangalore, India, December 13–16 2020. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-65277-7_32`.

20 Steven D. Galbraith and Lukas Zobernig. Obfuscating finite automata. Cryptology ePrint Archive, Report 2020/1009, 2020. URL: `https://eprint.iacr.org/2020/1009`.

21 Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016. `doi:10.1137/14095772X`.

22 Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 404–413. IEEE Computer Society, 2014. `doi:10.1109/FOCS.2014.50`.

23 Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26 1984. IEEE Computer Society Press. `doi:10.1109/SFCS.1984.715949`.

24 Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, WA, USA, May 15–17 1989. ACM Press. `doi:10.1145/73007.73010`.

25 Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *46th Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PA, USA, October 23–25 2005. IEEE Computer Society Press. `doi:10.1109/SFCS.2005.60`.

26 Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

27 Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213, Amsterdam, The Netherlands, February 21–24 2007. Springer, Heidelberg, Germany. `doi:10.1007/978-3-540-70936-7_11`.

28 Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 528–557, Paris, France, April 30 – May 4 2017. Springer, Heidelberg, Germany. `doi:10.1007/978-3-319-56614-6_18`.

29 Satoshi Hada. Secure obfuscation for encrypted signatures. In *EUROCRYPT*, pages 92–112. Springer, 2010.

30 Ariel Hamlin, Justin Holmgren, Mor Weiss, and Daniel Wichs. On the plausibility of fully homomorphic encryption for RAMs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 589–619, Santa Barbara, CA, USA, August 18–22 2019. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-26948-7_21`.

31 Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. *J. Cryptol.*, 24(4):694–719, 2011. doi:10.1007/s00145-010-9077-7.

32 Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021. doi:10.1145/3406325.3451093.

33 Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994. doi:10.1145/174644.174647.

34 Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *55th Annual Symposium on Foundations of Computer Science*, pages 374–383, Philadelphia, PA, USA, October 18–21 2014. IEEE Computer Society Press. doi:10.1109/FOCS.2014.47.

35 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. doi:10.1145/174130.174138.

36 Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, 1987. doi:10.1007/BF00116827.

37 Michael Luby and Charles Rackoff. Pseudo-random permutation generators and cryptographic composition. In *18th Annual ACM Symposium on Theory of Computing*, pages 356–363, Berkeley, CA, USA, May 28–30 1986. ACM Press. doi:10.1145/12130.12167.

38 Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 20–39, Interlaken, Switzerland, May 2–6 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-24676-3_2.

39 Daniele Micciancio. Oblivious data structures: Applications to cryptography. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 456–464. ACM, 1997. doi:10.1145/258533.258638.

40 Moni Naor, Gil Segev, and Udi Wieder. History-independent cuckoo hashing. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 631–642, Reykjavik, Iceland, July 7–11 2008. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-70583-3_51.

41 Moni Naor and Vanessa Teague. Anti-presistence: History independent data structures. In *33rd Annual ACM Symposium on Theory of Computing*, pages 492–501, Crete, Greece, July 6–8 2001. ACM Press. doi:10.1145/380752.380844.

42 Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *SIAM J. Comput.*, 50(3):857–908, 2021. doi:10.1137/15M1030108.

43 Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15 1997. Springer, Heidelberg, Germany. doi:10.1007/3-540-69053-0_18.

44 Amir Shpilka and Ilya Volkovich. On reconstruction and testing of read-once formulas. *Theory of Computing*, 10(1):465–514, 2014.

45 Hans Ulrich Simon. Learning decision lists and trees with equivalence-queries. In Paul M. B. Vitányi, editor, *Computational Learning Theory, Second European Conference, EuroCOLT '95, Barcelona, Spain, March 13-15, 1995, Proceedings*, volume 904 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 1995. doi:10.1007/3-540-59119-2_188.

46 L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. doi:10.1145/1968.1972.

47 Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 523–532, Baltimore, MA, USA, May 22–24 2005. ACM Press. doi:10.1145/1060590.1060669.

48 Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 600–611, Berkeley, CA, USA, October 15–17 2017. IEEE Computer Society Press. doi:10.1109/FOCS.2017.61.

A Separating Perfect and Deterministic "Obfuscation" for Equivalence Relations

The goal of this section is to formally separate the "abstract" notions of perfect iO and canonization as was roughly outlined in the beginning of Section 5.

In this section, we consider a sequence of equivalence relations $\{R_n\}_{n\in\mathbb{N}}$, where $R_n \subseteq \{0,1\}^n \times \{0,1\}^n$, and define similar notions of canonization and perfect obfuscation for the equivalence classes of R_n. This can be thought of as an abstract notion of obfuscation.

▶ **Definition 27** (Canonizer for R_n). *A canonizer for $R = \{R_n\}_{n\in\mathbb{N}}$ is a polynomial time algorithm* Can*, such that for all n and for all $x \in \{0,1\}^n$ it holds that $(x, \mathsf{Can}(x)) \in R_n$. Moreover, for all n, and for all $(x,y) \in R_n$ it holds that $\mathsf{Can}(x) = \mathsf{Can}(y)$*

▶ **Definition 28** (Perfect obfuscation for R_n.). *A perfect* iO *for $R = \{R_n\}_{n\in\mathbb{N}}$ is a p.p.t. algorithm* iO*, such that for all n and for all $x \in \{0,1\}^n$,* $\mathsf{iO}(x)$ *has the uniform distribution over all the strings $y\,\{0,1\}^n$ such that $(x,y) \in R_n$.*

We recall the Decisional Diffie-Hellman problem. Let $\mathcal{G}$ be polynomial-time a group generator that takes the security parameter λ as input and outputs $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a multiplicative cyclic group of prime order q and g is a generator of the group.

▶ **Definition 29** (Decisional Diffie-Hellman problem). *Let λ be the security parameter. We say that the decisional Diffie-Hellman problem is hard relative to $\mathcal{G}$ if*

$$(\mathbb{G}, q, g, g^a, g^b, g^{ab}) \approx_c (\mathbb{G}, q, g, g^a, g^b, g^c) \ ,$$

where $(\mathbb{G}, q, g) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ and $(a,b,c) \xleftarrow{\$} \mathbb{Z}_q$.

We show that under the DDH assumption there exists a relation R with a perfect iO but without a canonizer.

▶ **Theorem 30** (Separating deterministic and perfect abstract iO). *Under DDH, there exists a relation R such that:*
- *There exists a perfect obfuscator* iO *for R.*
- *There is no canonizer for R.*

Proof. Let $G = \langle g \rangle$ be a DDH-hard group. Define the relation $R \subseteq G^3 \times G^3$ over triples of elements in G as follows: $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$ are in R if and only if

$$\frac{x_3}{x_1^{\log_g(x_2)}} = \frac{y_3}{y_1^{\log_g(y_2)}} \ ,$$

where $\log_g(h)$ is the discrete logarithm in the group G with respect to the generator g.

First, we show how, given $x = (x_1, x_2, x_3)$, to sample y uniformly at random from the equivalence class of x. In order to do so, pick k_1, k_2 uniformly at random from the set $\{1, 2, \dots, |G|\}$, and return the triple

$$(x_1 g^{k_1}, x_2 g^{k_2}, x_3 x_2^{k_1} x_1^{k_2} g^{k_1 k_2}) \ .$$

The pairs are in the same class,

$$\frac{x_3 x_2^{k_1} x_1^{k_2} g^{k_1 k_2}}{\left(x_1 g^{k_1}\right)^{\log_g(x_2 g^{k_2})}} = \frac{x_3 x_2^{k_1} x_1^{k_2} g^{k_1 k_2}}{x_1^{\log_g(x_2)} x_1^{k_2} g^{k_1 \log_g(x_2)} g^{k_1 k_2}} = \frac{x_3}{x_1^{\log_g(x_2)}} \ .$$

Moreover, the output is distributed uniformly at random over the equivalence class of x by construction.

We now prove the lower bound. We do so by showing that canonizing R would imply distinguishing between the distributions D_1 and D_2, where

$$D_1 = \{(g^a, g^b, g^{ab}) : a, b \xleftarrow{\$} \{1, 2, \dots |G|\}\} \ ,$$

and

$$D_2 = \{(g^a, g^b, g^c) : a, b, c \xleftarrow{\$} \{1, 2, \dots |G|\}\} \ ,$$

which would break the DDH assumption.

In order to distinguish between D_1 and D_2 given a canonizer for R, on input $x = (x_1, x_2, x_3)$, the distinguisher picks arbitrary $a, b \in \{1, 2, \dots, |G|\}$, and canonize the triple (g^a, g^b, g^{ab}). It then canonize its input x. If the results are equal ($\mathsf{Can}(x) = \mathsf{Can}((g^a, g^b, g^{ab}))$), the distinguisher knows the input triple was sampled from D_1, and otherwise it knows it was sampled from D_2. ◀