Limits of Extractability Assumptions with Distributional Auxiliary Input

Elette Boyle[*]
Technion Israel
eboyle@alum.mit.edu

Rafael Pass[†]
Cornell University
rafael@cs.cornell.edu

August 24, 2015

Abstract

Extractability, or "knowledge," assumptions have recently gained popularity in the cryptographic community, leading to the study of primitives such as extractable one-way functions, extractable hash functions, succinct non-interactive arguments of knowledge (SNARKs), and (public-coin) differing-inputs obfuscation ((PC-)$di\mathcal{O}$), and spurring the development of a wide spectrum of new applications relying on these primitives. For most of these applications, it is required that the extractability assumption holds even in the presence of attackers receiving some *auxiliary information* that is sampled from some *fixed* efficiently computable distribution $\mathcal{Z}$.

We show that, assuming the existence of public-coin collision-resistant hash functions, there exists an efficient distributions $\mathcal{Z}$ such that either

- PC-$di\mathcal{O}$ for Turing machines does not exist, or
- extractable one-way functions w.r.t. auxiliary input $\mathcal{Z}$ do not exist.

A corollary of this result shows that additionally assuming existence of fully homomorphic encryption with decryption in NC^1, there exists an efficient distribution $\mathcal{Z}$ such that either

- SNARKs for NP w.r.t. auxiliary input $\mathcal{Z}$ do not exist, or
- PC-$di\mathcal{O}$ for NC^1 circuits does not exist.

To achieve our results, we develop a "succinct punctured program" technique, mirroring the powerful punctured program technique of Sahai and Waters (STOC'14), and present several other applications of this new technique. In particular, we construct succinct perfect zero knowledge SNARGs and give a universal instantiation of random oracles in full-domain hash applications, based on PC-$di\mathcal{O}$.

As a final contribution, we demonstrate that *even in the absence of auxiliary input*, care must be taken when making use of extractability assumptions. We show that (standard) $di\mathcal{O}$ w.r.t. any distribution $\mathcal{D}$ over programs and bounded-length auxiliary input is directly implied by any

[*]This work was primarily completed while the first author was a postdoc at Cornell University, supported in part by AFOSR YIP Award FA9550-10-1-0093. This research has received funding from the European Union's Tenth Framework Programme (FP10/ 2010-2016) under grant agreement no. 259426 ERC-CaC.

[†]Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2- 0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

[†]Part of this work was done while visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

obfuscator that satisfies the weaker indistinguishability obfuscation (i$\mathcal{O}$) security notion and $di\mathcal{O}$ for a slightly modified distribution $\mathcal{D}'$ of programs (of slightly greater size) and no auxiliary input. As a consequence, we directly obtain negative results for (standard) $di\mathcal{O}$ in the absence of auxiliary input.

1 Introduction

Extractability Assumptions. Extractability, or "knowledge," assumptions (such as the "knowledge-of-exponent" assumption), have recently gained in popularity, leading to the study of primitives such as extractable one-way functions, extractable hash-functions, SNARKs (succinct non-interactive arguments of knowledge), and differing-inputs obfuscation:

- **Extractable OWF:** An extractable family of one-way (resp. collision-resistant) functions [Dam91, HT98, CD09], is a family of one-way (resp. collision-resistant) functions $\{f_i\}$ such that any attacker who outputs an element y in the range of a randomly chosen function f_i given the index i must "know" a pre-image x of y (i.e., $f_i(x) = y$). This is formalized by requiring for every adversary $\mathcal{A}$, the existence of an "extractor" $\mathcal{E}$ that (with overwhelming probability) given the view of $\mathcal{A}$ outputs a pre-image x whenever $\mathcal{A}$ outputs an element y in the range of the function.

  For example, the "knowledge-of-exponent" assumption of Damgard [Dam91] stipulates the existence of a particular such extractable one-way function.
- **SNARKs:** Succinct non-interactive arguments of knowledge (SNARKs) [Mic94, Val08, BCCT12] are communication-efficient (i.e., "short" or "succinct") arguments for NP with the property that if a prover generates an accepting (short) proof, it must "know" a corresponding (potentially long) witness for the statement proved, and this witness can be efficiently "extracted" out from the prover.
- **Differing-inputs obfuscation:** [BGI$^+$12, BCP14, ABG$^+$13] A public-coin differing-inputs obfuscator $\mathcal{O}$ for program-pair distribution $\mathcal{D}$ is an efficient procedure which ensures if any efficient attacker $\mathcal{A}$ can distinguish obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ of programs C_1, C_2 generated via $\mathcal{D}$ given the randomness r used in sampling, then it must "know" an input x such that $C_1(x) \neq C_2(x)$, and this input can be efficiently "extracted" from $\mathcal{A}$.

  A recently proposed (weaker) variant known as *public-coin differing-inputs obfuscation* [IPS15] additionally provides the randomness used to sample the programs $(C_0, C_1) \leftarrow \mathcal{D}$ to the extraction algorithm (and to the attacker $\mathcal{A}$).

The above primitives have proven extremely useful in constructing cryptographic tools for which instantiations under complexity-theoretic hardness assumptions are not known (e.g., [HT98, BCCT12, GLR11, DFH12, BCP14, ABG$^+$13, IPS15]).

Extraction with (Distribution-Specific) Auxiliary Input. In all of these applications, we require a notion of an *auxiliary-input* extractable one-way function [HT98, CD09], where both the attacker and the extractor may receive an auxiliary input. The strongest formulation requires extractability in the presence of an *arbitrary* auxiliary input. Yet, as informally discussed already in the original work by Hada and Tanaka [HT98], extractability w.r.t. an arbitrary auxiliary input is an "overly strong" (or in the language of [HT98], "unreasonable") assumption. Indeed, a recent result of Bitansky, Canetti, Rosen and Paneth [BCPR14] (formalizing earlier intuitions from [HT98, BCCT12]) demonstrates that assuming the existence of indistinguishability obfuscators for the class of polynomial-size circuits[1] there cannot exist auxiliary-input extractable one-way functions that remain secure for an arbitrary auxiliary input.

[1] The notion of indistinguishability obfuscation [BGI$^+$12] requires that obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ of any two *equivalent* circuits C_1 and C_2 (i.e., whose outputs agree on all inputs) from some class $\mathcal{C}$ are computationally indistinguishable. A candidate construction for general-purpose indistinguishability obfuscation was recently given by Garg et al [GGH$^+$13].

However, for most of the above applications, we actually do not require extractability to hold w.r.t. an arbitrary auxiliary input. Rather, as proposed by Bitansky et al [BCCT12, BCCT13], it often suffices to consider extractability with respect to specific distributions $\mathcal{Z}$ of auxiliary input.[2] More precisely, it would suffice to show that for every desired output length $\ell(\cdot)$ and distribution $\mathcal{Z}$ there exists a function family $\mathcal{F}_{\mathcal{Z}}$ (which, in particular, may be tailored for $\mathcal{Z}$) such that $\mathcal{F}_{\mathcal{Z}}$ is a family of extractable one-way (or collision-resistant) functions $\{0,1\}^k \to \{0,1\}^{\ell(k)}$ with respect to $\mathcal{Z}$. In fact, for some of these results (e.g., [BCCT12, BCCT13]), it suffices to just assume that extraction works for just for the *uniform* distribution.

In contrast, the result of [BCPR14] can be interpreted as saying that (assuming $i\mathcal{O}$), there do not exist extractable one-way functions with respect to *every* distribution of auxiliary input: That is, for every candidate extractable one-way function family $\mathcal{F}$, there exists *some* distribution $\mathcal{Z}_{\mathcal{F}}$ of auxiliary input that breaks it.

Our Results. In this paper, we show limitations of extractability primitives with respect to *distribution-specific* auxiliary input (assuming the existence of public-coin collision-resistant hash functions (CRHF) [HR04]). Our main result shows a conflict between public-coin differing-inputs obfuscation for Turing machines [IPS15] and extractable one-way functions.

Theorem 1.1 (Main Theorem – Informal)**.** *Assume the existence of public-coin collision-resistant hash functions. Then for every polynomial ℓ, there exists an efficiently computable distribution $\mathcal{Z}$ such that one of the following two primitives does not exist:*

- *extractable one-way functions $\{0,1\}^k \to \{0,1\}^{\ell(k)}$ w.r.t. auxiliary input from $\mathcal{Z}$.*
- *public-coin differing-inputs obfuscation for Turing machines.*

By combining our main theorem with results from [BCCT12] and [IPS15], we obtain the following corollary:

Theorem 1.2 (Informal)**.** *Assume the existence of public-coin CRHF and fully homomorphic encryption with decryption in NC^1.*[3] *Then there exists an efficiently computable distribution $\mathcal{Z}$ such that one of the following two primitives does not exist:*

- *SNARKs w.r.t. auxiliary input from $\mathcal{Z}$.*
- *public-coin differing-inputs obfuscation for NC^1 circuits.*

To prove our results, we develop a new proof technique, which we refer to as the "succinct punctured program" technique, extending the "punctured program" paradigm of Sahai and Waters [SW14]; see Section 1.1 for more details. This technique has several other interesting applications, as we discuss in Section 1.3.

As a final contribution, we demonstrate that *even in the absence of auxiliary input*, care must be taken when making use of extractability assumptions. Specifically, we show that differing-inputs obfuscation ($di\mathcal{O}$) for any distribution $\mathcal{D}$ of programs and bounded-length auxiliary inputs, is directly implied by any obfuscator that satisfies a weaker indistinguishability obfuscation ($i\mathcal{O}$) security notion (which is not an extractability assumption) and $di\mathcal{O}$ security for a related distribution $\mathcal{D}'$ of programs (of slightly greater size) which does *not contain auxiliary input*. Thus, negative results ruling out existence of $di\mathcal{O}$ with bounded-length auxiliary input directly imply negative results for $di\mathcal{O}$ in a setting without auxiliary input.

[2]As far as we know, the only exceptions are in the context of zero-knowledge simulation, where the extractor is used in the simulation (as opposed to being used as part of a reduction), and we require simulation w.r.t. arbitrary auxiliary inputs. Nevertheless, as pointed out in the works on zero-knowledge [HT98, GS12], to acheive "plain" zero-knowledge [GMR89, BLV06] (where the verifier does not receive any auxiliary input), weaker "bounded" auxiliary input assumptions suffice.

[3]As is the case for nearly all existing FHE constructions (e.g., [GSW13, BV14]).

Theorem 1.3 (Informal)**.** *Let $\mathcal{D}$ be a distribution over pairs of programs and ℓ-bounded auxiliary input information $\mathcal{P} \times \mathcal{P} \times \{0,1\}^\ell$. There exists diO with respect to $\mathcal{D}$ if there exists an obfuscator satisfying iO in addition to diO with respect to a modified distribution $\mathcal{D}'$ over $\mathcal{P}' \times \mathcal{P}'$ for slightly enriched program class $\mathcal{P}'$, and* no auxiliary input.

Our transformation applies to a recent result of Garg *et al.* [GGHW14], which shows that based on a new assumption (pertaining to special-purpose obfuscation of Turing machines) general-purpose *diO* w.r.t. auxiliary input cannot exist, by constructing a distribution over circuits and bounded-length auxiliary inputs for which no obfuscator can be *diO*-secure. Our resulting conclusion is that, assuming such special-purpose obfuscation exists, then general-purpose *diO* cannot exist, *even in the absence of auxiliary input.*

We view this as evidence that public-coin differing inputs may be the "right" approach definitionally, as restrictions on auxiliary input without regard to the programs themselves will not suffice.

Interpretation of Our Results. Our results suggest that one must take care when making extractability assumptions, even in the presence of specific distributions of auxiliary inputs, and in certain cases even in the absence of auxiliary input. In particular, we must develop a way to distinguish "good" distributions of instances and auxiliary inputs (for which extractability assumptions may make sense) and "bad" ones (for which extractability assumptions are unlikely to hold). As mentioned above, for some applications of extractability assumptions, it in fact suffices to consider a particularly simple distribution of auxiliary inputs—namely the *uniform* distribution.[4] We emphasize that our results do not present any limitations of extractable one-way functions in the presence of uniform auxiliary input, and as such, this still seems like a plausible assumption at this point.

Comparison to [GGHW14]. An interesting subsequent[5] work of Garg *et al.* [GGHW13, GGHW14] contains a related study of differing-inputs obfuscation. In [GGHW14], the authors propose a new "special-purpose" circuit obfuscation assumption, and demonstrate based on this assumption an auxiliary input distribution (whose size grows with the desired circuit size of circuits to be obfuscated) for which general-purpose *diO* cannot exist. Using similar techniques of hashing and obfuscating Turing machines as in the current work, they further conclude that if the new obfuscation assumption holds also for *Turing machines*, then the "bad" auxiliary input distribution can have bounded length (irrespective of the circuit size).

Garg *et al.* [GGHW14] show the "special-purpose" obfuscation assumption is a falsifiable assumption (in the sense of [Nao03]) and is implied by virtual black-box obfuscation for the relevant restricted class of programs, but plausibility of the notion in relation to other primitives is otherwise unknown. In contrast, our results provide a direct relation between existing, studied topics (namely, *diO*, EOWFs, and SNARKs). Even in the case that the special-purpose obfuscation assumption *does* hold, our primary results provide conclusions for *public-coin diO*, whereas Garg *et al.* [GGHW14] consider (stronger) standard *diO*, with respect to auxiliary input.

And, utilizing our final observation (which occurred subsequent to [GGHW14]), we show that based on their same special-purpose obfuscation assumption for Turing machines, we can in fact rule out general-purpose *diO* for circuits even *in the absence of auxiliary input.*

1.1 Proof Techniques

To explain our techniques, let us first explain earlier arguments against the plausibility of extractable one-way functions with auxiliary input. For simplicity of notation, we focus on extractable one-way function over $\{0,1\}^k \to \{0,1\}^k$ (as opposed to over $\{0,1\}^k \to \{0,1\}^{\ell(k)}$ for

[4]Note that this is not the case for all applications; e.g. [HT98, GKP$^+$13, BGI14, GS12] require considering more complicated distributions.

[5]A version of our paper with Theorem 1 and 2 for (standard) differing-inputs obfuscation in the place of public-coin *diO* has been on ePrint since October 2013 [BP13].

some polynomial ℓ), but emphasize that the approach described directly extends to the more general setting.

Early Intuitions. As mentioned above, already the original work of Hada and Tanaka [HT98], which introduced auxiliary input extractable one-way functions (EOWFs) (for the specific case of exponentiation), argued the "unreasonableness" of such functions, reasoning informally that the auxiliary input could contain a program that evaluates the function, and thus a corresponding extractor must be able to "reverse-engineer" *any* such program. Bitansky et al [BCCT12] made this idea more explicit: Given some candidate EOWF family $\mathcal{F}$, consider the distribution $\mathcal{Z}_{\mathcal{F}}$ over auxiliary input formed by "obfuscating" a program $\Pi^s(\cdot)$ for uniformly chosen s, where $\Pi^s(\cdot)$ takes as input a function index e from the alleged EOWF family $\mathcal{F} = \{f_i\}$, applies a pseudorandom function (PRF) with hardcoded seed s to the index i, and then outputs the evaluation $f_i(\mathsf{PRF}_s(i))$. Now, consider an attacker $\mathcal{A}$ who, given an index i, simply runs the obfuscated program to obtain a "random" point in the range of f_i. If it were possible to obfuscate Π^s in a "virtual black-box (VBB)" way (as in [BGI$^+$12]), then it easily follows that any extractor $\mathcal{E}$ for this particular attacker $\mathcal{A}$ can invert f_i. Intuitively, the VBB-obfuscated program hides the PRF seed s (revealing, in essence, only black-box access to Π^s), and so if $\mathcal{E}$ can successfully invert f_i on $\mathcal{A}$'s output $f_i(\mathsf{PRF}_s(i))$ on a pseudorandom input $\mathsf{PRF}_s(i)$, he must also be able to invert for a *truly* random input. Formally, given an index i and a random point y in the image of f_i, we can "program" the output of $\Pi^s(i)$ to simply be y, and thus E will be forced to invert y.

The problem with this argument is that (as shown by Barak et al [BGI$^+$12]), for large classes of functions VBB program obfuscation simply does not exist.

The Work of [BCPR14] and the "Punctured Program" Paradigm of [SW14]. Intriguingly, Bitansky, Canetti, Rosen and Paneth [BCPR14] show that by using a particular PRF and instead relying on indistinguishability obfuscation, the above argument still applies! To do so, they rely on the powerful "punctured-program" paradigm of Sahai and Waters [SW14] (and the closely related work of Hohenberger, Sahai and Waters [HSW14] on "instantiating random oracles"). Roughly speaking, the punctured program paradigm shows that if we use indistinguishability obfuscation to obfuscate a (function of) a special kind of "puncturable" PRF[6] [BW13, BGI14, KPTZ13], we can still "program" the output of the program on *one* input (which was used in [SW14, HSW14] to show various applications of indistinguishability obfuscation). Bitansky et al. [BCPR14] show that by using this approach, then from any alleged extractor $\mathcal{E}$ we can construct a one-way function inverter Inv by "programming" the output of the program Π^s at the input i with the challenge value y. More explicitly, mirroring [SW14, HSW14], they consider a hybrid experiment where $\mathcal{E}$ is executed with fake (but indistinguishable) auxiliary input, formed by obfuscating a *"punctured"* variant $\Pi^s_{i,y}$ of the program Π^s that contains an i-punctured PRF seed s^* (enabling evaluation of $\mathsf{PRF}_s(j)$ for any $j \neq i$) and directly outputs the hardcoded value $y := f_i(\mathsf{PRF}_s(i))$ on input i: indistinguishability of this auxiliary input follows by the security of indistinguishability obfuscation since the programs $\Pi^s_{i,y}$ and Π^s are equivalent when $y = f_i(\mathsf{PRF}_s(i)) = \Pi^s(i)$. In a second hybrid experiment, the "correct" hardcoded value y is replaced by a *random* evaluation $f_i(u)$ for uniform u; here, indistinguishability of the auxiliary inputs follows directly by the security of the punctured PRF. Finally, by indistinguishability of the three distributions of auxiliary input in the three experiments, it must be that $\mathcal{E}$ can extract an inverse to y with non-negligible probability in each hybrid; but, in the final experiment this implies the ability to invert a random evaluation, breaking one-wayness of the EOWF.

The Problem: Dependence on $\mathcal{F}$. Note that in the above approach, the auxiliary input distribution is selected *as a function* of the family $\mathcal{F} = \{f_j\}$ of (alleged) extractable one-way

[6]That is, a PRF where we can surgically remove one point in the domain of the PRF, keeping the rest of the PRF intact, and yet, even if we are given the seed of the punctured PRF, the value of the original PRF on the surgically removed point remains computationally indistinguishable from random.

functions. Indeed, the obfuscated program Π^s must be able to evaluate f_j given j. One may attempt to mitigate this situation by instead obfuscating a universal circuit that takes as input both $\mathcal{F}$ and the index j, and appropriately evaluates f_j. But here still the *size* of the universal circuit must be greater than the running time of f_j, and thus such an auxiliary input distribution would only rule out EOWFs with a-priori bounded running time. This does not suffice for what we aim to achieve: in particular, it still leaves open the possibility that for every distribution of auxiliary inputs, there may exist a family of extractable one-way functions that remains secure for that particular auxiliary input distribution (although the running time of the extractable one-way function needs to be greater than the length of the auxiliary input).

A First Idea: Using Turing Machine Obfuscators. At first sight, it would appear this problem could be solved if we could obfuscate *Turing machines*. Namely, by obfuscating a universal Turing machine in the place of a universal circuit in the construction above, the resulting program Π^s would depend only on the size of the PRF seed s, and not on the runtime of $f_j \in F$.

But there is a catch. To rely on the punctured program paradigm, we must be able to obfuscate the program Π^s in such a way that the result is indistinguishable from an obfuscation of a related "punctured" program $\Pi^s_{i,y}$; in particular, the *size* of the obfuscation must be at least as large as $|\Pi^s_{i,y}|$. Whereas the size of Π^s is now bounded by a polynomial in the size of the PRF seed s, the description of this punctured program must specify a punctured input i (corresponding to an index of the candidate EOWF $\mathcal{F}$) and hardcoded output value y, and hence must grow with the size of $\mathcal{F}$. We thus run into a similar wall: even with obfuscation of Turing machines, the resulting auxiliary input distribution $\mathcal{Z}$ would only rule out EOWF with a-priori bounded index length.

Our "Succinct Punctured Program" Technique. To deal with this issue, we develop a "succinct punctured program" technique. That is, we show how to make the size of the obfuscation be independent of the length of the input, while still retaining its usability as an obfuscator. The idea is two-fold: First, we modify the program Π^s to *hash* the input to the PRF, using a collision-resistant hash function h. That is, we now consider a program $\Pi^{h,s}(j) = f_j(PRF_s(h(j)))$. Second, we make use of *differing-inputs obfuscation*, as opposed to just indistinguishability obfuscation. Specifically, our constructed auxiliary input distribution $\mathcal{Z}$ will sample a uniform s and a random hash function h (from some appropriate collection of collision-resistant hash functions) and then output a differing-inputs obfuscation of $\Pi^{h,s}$.

To prove that this "universal" distribution $\mathcal{Z}$ over auxiliary input breaks *all* alleged extractable one-way functions over $\{0,1\}^k \to \{0,1\}^k$, we define a one-way function inverter Inv just as before, except that we now feed the EOWF extractor $\mathcal{E}$ the obfuscation of the "punctured" variant $\Pi^{h,s}_{i,y}$ which contains a PRF seed punctured at point $h(i)$. The program $\Pi^{h,s}_{i,y}$ proceeds just as $\Pi^{h,s}$ except on all inputs j such that $h(j)$ is equal to this special value $h(i)$; for those inputs it simply outputs the hardcoded value y. (Note that the index i is no longer needed to specify the function $\Pi^{h,s}_{i,y}$—rather, just its hash $h(i)$—but is included for notational convenience). As before, consider a hybrid experiment where y is selected as $y := \Pi^{h,s}(i)$.

Whereas before the punctured program was equivalent to the original, and thus indistinguishability of auxiliary inputs in the different experiments followed by the definition of indistinguishability obfuscation, here it is *no longer* the case that if $y = \Pi^{h,s}(i)$, then $\Pi^{h,s}_{i,y}$ is equivalent to $\Pi^{h,s}$—in fact, they may differ on many points. More precisely, the programs may differ in all points j such that $h(j) = h(i)$, but $j \neq i$ (since f_j and f_i may differ on the input $PRF_s(h(i))$). Thus, we can no longer rely on indistinguishability obfuscation to provide indistinguishability of these two hybrids.

We resolve this issue by relying *differing-inputs obfuscation* instead of just indistinguishability obfuscation. Intuitively, if obfuscations of $\Pi^{h,s}$ and $\Pi^{h,s}_{i,y}$ can be distinguished when y is set to $\Pi^{h,s}(i)$, then we can efficiently recover some input j where the two programs differ. But, by construction, this must be some point j for which $h(j) = h(i)$ (or else the two program are the same), *and* $j \neq i$ (since we chose the hardcoded value $y = \Pi^{h,s}(i)$ to be consistent with $\Pi^{h,s}$ on input i. Thus, if the obfuscations can be distinguished, we can find a collision in h, contradicting its collision resistance.

To formalize this argument using just public-coin $di\mathcal{O}$, we require that h is a *public-coin* collision-resistant hash functions [HR04].

1.2 Removing Auxiliary Input in $di\mathcal{O}$

The notion of public-coin $di\mathcal{O}$ is weaker than "general" (not necessarily public-coin) $di\mathcal{O}$ in two aspects: 1) the programs M_0, M_1 are sampled using only public randomness, and 2) we consider only a very specific auxiliary input that is given to the attacker—namely the randomness of the sampling procedure.

In this section, we explore another natural restriction of $di\mathcal{O}$ where we simply disallow auxiliary input, but allow for "private" sampling of M_0, M_1. We show that "bad side information" cannot be circumvented simply by simply disallowing auxiliary input, but rather such information can appear in the *input-output behavior* of the programs to be obfuscated.

More precisely, we show that for any distribution $\mathcal{D}$ over $\mathcal{P} \times \mathcal{P} \times \{0,1\}^\ell$ of programs $\mathcal{P}$ and bounded-length auxiliary input, the existence of $di\mathcal{O}$ w.r.t. $\mathcal{D}$ is directly implied by the existence of any *indistinguishability* obfuscator ($i\mathcal{O}$) that is $di\mathcal{O}$-secure for a slightly enriched distribution of programs $\mathcal{D}'$ over $\mathcal{P}' \times \mathcal{P}'$, *without* auxiliary input.

Intuitively, this transformation works by embedding the "bad auxiliary input" into the *input-output behavior* of the circuits to be obfuscated themselves. That is, the new distribution $\mathcal{D}'$ is formed by sampling first a triple (P_0, P_1, z) of programs and auxiliary input from the original distribution $\mathcal{D}$, and then instead considering the tweaked programs P_0^z, P_1^z that have a special additional input x^* (denoted later as "$\mathsf{mode} = *$") for which $P_0^z(x^*) = P_1^z(x^*)$ is defined to be z. This introduces no new differing inputs to the original program pair P_0, P_1, but now there is no hope of preventing the adversary from learning z without sacrificing correctness of the obfuscation scheme.

A technical challenge arises in the security reduction, however, in which we must modify the obfuscation of the z-embedded program P_b^z to "look like" an obfuscation of the original program P_b. Interestingly, this issue is solved by making use of a *second layer* of obfuscation, and is where the $i\mathcal{O}$ security of the obfuscator is required. We refer the reader to Section C for details.

1.3 Other Applications of the "Succinct Punctured Program" Technique

As mentioned above, the "punctured program" paradigm of [SW14] has been used in multiple applications (e.g., [SW14, HSW14, GGHR, BZ14]). Many of them rely on punctured programs in an essentially identical way to the approach described above, and in particular follow the same hybrids within the security proof. Furthermore, for some of these applications, there are significant gains in making the obfuscation succinct (i.e., independent of the input size of the obfuscated program). Thus, for these applications, if we instead rely on public-coin differing-inputs obfuscation (and the existence of public-coin collision-resistant hash functions), by using our succinct punctured program technique, we can obtain significant improvements. For instance, relying on the same approach as above, we can show based on these assumptions:

- "Succinct" Perfect Zero-Knowledge Non-Interactive *Universal* Argument System (with communication complexity k^ϵ for every ϵ), by relying on the non-succinct Perfect NIZK construction of [SW14].
- A *universal* instantiation of Random Oracles, for which the Full Domain Hash (FDH) signature paradigm [BR93] is (selectively) secure for *every* trapdoor (one-to-one) function (if hashing not only the message but also the index of the trapdoor function), by relying on

the results of [HSW14] showing how to provide a trapdoor-function specific instantiation of the random oracle in the FDH.[7]

We explore these applications further in Section B.

1.4 Overview of Paper

We focus in this submission on the primary result: the conflict between public-coin differing inputs obfuscation and extractable OWFs (and SNARKs). Further preliminaries, applications of our succinct punctured programs technique, and our transformation removing auxiliary input in differing-inputs obfuscation are deferred to the Supplementary Materials (which appear at the end of this file).

2 Preliminaries

2.1 Public-Coin Differing-Inputs Obfuscation

The notion of public-coin differing-inputs obfuscation was introduced by Ishai *et al.* [IPS15] as a refinement of (standard) differing-inputs obfuscation [BGI$^+$12] to exclude certain cases whose feasibility has been called into question. We now present the definitions of [IPS15], focuing only on Turing machine obfuscation; the definition easily extends also to circuits.

Definition 2.1 (Public-Coin Differing-Inputs Sampler for TMs)**.** An efficient non-uniform sampling algorithm $\mathsf{Samp} = \{\mathsf{Samp}_k\}$ is called a *public-coin differing inputs sampler* for the parameterized collection of TMs $\mathcal{M} = \{\mathcal{M}_k\}$ if the output of Samp_k is always a pair of Turing machines $(M_0, M_1) \in \mathcal{M}_k \times \mathcal{M}_k$ such that $|M_0| = |M_1|$ and for every efficient non-uniform algorithm $\mathcal{A} = \{\mathcal{A}_k\}$ there exists a negligible function ϵ such that for all $k \in \mathbb{N}$,

$$\Pr\Big[r \leftarrow \{0,1\}^*; (M_0, M_1) \leftarrow \mathsf{Samp}_k(r); (x, , 1^t) \leftarrow \mathcal{A}_k(r) \\ : \big(M_0(x) \neq M_(x)\big) \wedge \big(\mathsf{steps}(M_0, x) = \mathsf{steps}(M_1, x)\big)\Big] \leq \epsilon(k).$$

Definition 2.2 (Public-Coin Differing-Inputs Obfuscator for TMs)**.** A uniform PPT algorithm $\mathcal{O}$ is a *public-coin differing-inputs obfuscator for the collection* $\mathcal{M} = \{\mathcal{M}_k\}$ if the following requirements hold:

- **Correctness:** For every $k \in \mathbb{N}$, every $M \in \mathcal{M}_k$, and every x, we have that $\Pr[\tilde{M} \leftarrow \mathcal{O}(1^k, M) : \tilde{M}(x) = M(x)] = 1$.
- **Security:** For every public-coin differing-inputs sampler $\mathsf{Samp} = \{\mathsf{Samp}_k\}$ for the ensemble $\mathcal{M}$, every efficient non-uniform distinguishing algorithm $\mathcal{D} = \{\mathcal{D}_k\}$, there exists a negligible function ϵ such that for all k,

$$\big|\Pr[r \leftarrow \{0,1\}^*; (M_0, M_1) \leftarrow \mathsf{Samp}_k(r); \tilde{M} \leftarrow \mathcal{O}(1^k, M_0) : \mathcal{D}_k(r, \tilde{M}) = 1] - \\ \Pr[r \leftarrow \{0,1\}^*; (M_0, M_1) \leftarrow \mathsf{Samp}_k(r); \tilde{M} \leftarrow \mathcal{O}(1^k, M_1) : \mathcal{D}_k(r, \tilde{M}) = 1]\big| \leq \epsilon(k).$$

2.2 Differing-Inputs Obfuscation

For each distribution $\mathcal{D} = \{D_k\}$ over $\mathcal{M}_k \times \mathcal{M}_k \times \{0,1\}^*$ (specifying program pairs and auxiliary input), we define differing-inputs obfuscation secure with respect to $\mathcal{D}$. We present a definition as formalized in [BCP14], which is a variant of the notion of "differing-inputs" obfuscation of [BGI$^+$12].[8]

[7] That is, [HSW14] shows that for every trapdoor one-to-one function, there exists some way to instantiate the random oracle so that the resulting scheme is secure. In constrast, our results shows that there exists a single instantation that works no matter what the trapdoor function is.

[8] Formally, our notion of differing-inputs obfuscation departs from differing-inputs obfuscation of [BGI$^+$12] in three ways: First, we consider a setting in which the adversary (and extractor) may receive auxiliary input. Second, [BGI$^+$12]

Definition 2.3 (Differing-Inputs Obfuscator w.r.t. Distribution $\mathcal{D}$)**.** A uniform PPT machine $di\mathcal{O}$ is a *differing-inputs obfuscator w.r.t. distribution* $\mathcal{D}$ over $\mathcal{M}_k \times \mathcal{M}_k \times \{0,1\}^*$ if it satisfies the following correctness and security properties:

- **Correctness:** There exists a negligible function $\mathsf{negl}(k)$ such that for every security parameter $k \in \mathbb{N}$, for all $M \in \mathcal{M}_k$, for all inputs x, we have

$$\Pr[\tilde{M} \leftarrow di\mathcal{O}(1^k, M) : \tilde{M}(x) = M(x)] = 1 - \mathsf{negl}(k).$$

- **Security:** For every non-uniform PPT adversary $\mathcal{A}$ and polynomial $p(k)$, there exists a non-uniform PPT extractor E and polynomial $q(k)$ such that for every $k \in \mathbb{N}$, it holds with overwhelming probability over $(M_0, M_1, z) \leftarrow \mathcal{D}$ that

$$\Pr\left[b \leftarrow \{0,1\}; \tilde{M} \leftarrow di\mathcal{O}(1^k, M_b) : \mathcal{A}(1^k, \tilde{M}, z) = b\right] \geq \frac{1}{2} + \frac{1}{p(k)}$$
$$\implies \quad \Pr\left[w \leftarrow E(1^k, M_0, M_1, z) : M_0(w) \neq M_1(w)\right] \geq \frac{1}{q(k)}.$$

Definition 2.4 (Differing-Inputs Obfuscator for TM)**.** A uniform PPT machine $di\mathcal{O}_{\mathsf{TM}}$ is called a *differing-inputs obfuscator for the class* TM *of polynomial-size Turing machines* if it satisfies the following. For each k, let $\mathcal{M}_k$ be the class of Turing machines Π containing a description of a Turing machine M of size bounded by k, such that Π takes two inputs, (t, x), with $|t| = k$, and the output of $\Pi(t, x)$ is defined to be the the output of running the Turing machine $M(x)$ for t steps. Then $di\mathcal{O}_{\mathsf{TM}}$ is a differing-inputs obfuscator for $\{\mathcal{M}_k\}$.[9]

2.3 Extractable One-Way Functions

For this work, we consider a slightly weakened version of EOWFs w.r.t. *distributional* auxiliary input information, from some distribution $\mathcal{Z}$. Namely, we require one-wayness and extractability to hold with overwhelming probability over auxiliary input z sampled from $\mathcal{Z}$; in contrast, typical definitions require these properties to hold for *any* auxiliary input z in some auxiliary input set $\mathcal{Z}$ (e.g., *all* length-bounded values). For example, "standard" auxiliary-input-secure EOWFs [CD08] are required to be $\mathcal{Z}$-auxiliary-input EOWFs for *every* (possibly non-uniformly generated) distribution $\mathcal{Z}$.

We present a non-uniform version of the definition, in which both one-wayness and extractability are with respect to *non-uniform* polynomial-time adversaries.

Definition 2.5 ($\mathcal{Z}$-Auxiliary-Input EOWF)**.** Let ℓ, m be polynomially bounded length functions. An efficiently computable family of functions $\mathcal{F} = \left\{ f_i : \{0,1\}^k \to \{0,1\}^{\ell(k)} \;\middle|\; i \in \{0,1\}^{m(k)}, k \in \mathbb{N} \right\}$, associated with an efficient probabilistic key sampler $\mathcal{K}_{\mathcal{F}}$, is a *$\mathcal{Z$-auxiliary-input extractable one-way function* if it satisfies:

require the extractor E to extract a differing input for M_0, M_1 given *any* pair of programs M_0', M_1' evaluating equivalent functions. Third, [BGI+12] consider also adversaries who distinguish with negligible advantage $\epsilon(k)$, and require that extraction still succeeds in this setting, but within time polynomial in $1/\epsilon$. In contrast, we restrict our attention only to adversaries who succeed with noticeable advantage.

[9]Note that applying the properties of differing-inputs obfuscation to this class of Turing machines $\{\mathcal{M}_k\}$ implies that for programs $\Pi_0, \Pi_1 \in \mathcal{M}_k$ defined above (corresponding to underlying size-k Turing machines M_0, M_1), efficiently distinguishing between obfuscations of Π_0 and Π_1 implies that one can efficiently extract an input *pair* (t', x') for which $\Pi_0(t', x') \neq \Pi_1(t', x')$. In particular, either $M_0(x') \neq M_1(x')$ or $\mathit{Runtime}(M_0, x') \neq \mathit{Runtime}(M_1, x)$. Thus, if restricting attention to a subclass of $\mathcal{M}_k$ for which each pair of programs satisfies $\mathit{Runtime}(M_0, x) = \mathit{Runtime}(M_1, x)$ for each input x, then "standard" extraction is guaranteed (i.e., such that the extracted input contains x' satisfying $M_0(x') \neq M_1(x')$). In the sequel, when referring to a differing-inputs obfuscation of a Turing machine M, we will implicitly mean the related program Π_M as above, but will suppress notation of the additional input t. For our application, it will be the case for the relevant class of Turing machines that every pair of programs M_0, M_1 has same runtime per input (and thus we will achieve "standard" input-extraction guaranteed).

- **One-wayness:** For non-uniform polynomial-time $\mathcal{A}$ and sufficiently large $k \in \mathbb{N}$,

$$\Pr\left[z \leftarrow \mathcal{Z}_k;\ i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ x \leftarrow \{0,1\}^k;\ x' \leftarrow \mathcal{A}(i, f_i(x); z) : f_i(x') = f_i(x)\right] \leq \mathsf{negl}(k).$$

- **Extractability:** For any non-uniform polynomial-time adversary $\mathcal{A}$, there exists a non-uniform polynomial-time extractor $\mathcal{E}$ such that, for sufficiently large security parameter $k \in \mathbb{N}$:

$$\Pr\left[z \leftarrow \mathcal{Z}_k;\ i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ y \leftarrow \mathcal{A}(i; z);\ x' \leftarrow \mathcal{E}(i; z) : \exists x \text{ s.t. } f_i(x) = y \wedge f_i(x') \neq y\right] \leq \mathsf{negl}(k).$$

3 Public-Coin Differing-Inputs Obfuscation or Extractable One-Way Functions

In this section, we present our main result: a conflict between extractable one-way functions (EOWF) w.r.t. a particular distribution of auxiliary information and public-coin differing-inputs obfuscation ("$\mathsf{PC}\text{-}di\mathcal{O}$") (for Turing Machines).

3.1 From PC-$di\mathcal{O}$ to Impossibility of $\mathcal{Z}$-Auxiliary-Input EOWF

We demonstrate a bounded polynomial-time uniformly samplable distribution $\mathcal{Z}$ (with bounded poly-size output length) and a public-coin differing-inputs sampler for Turing Machines $\mathcal{D}$ (over $\mathsf{TM} \times \mathsf{TM}$) such that if there exists public-coin differing-inputs obfuscation for Turing machines (and, in particular, for the program sampler $\mathcal{D}$), and there exist public-coin collision-resistant hash functions (CRHF), then there do *not* exist extractable one-way functions (EOWF) w.r.t. auxiliary information sampled from distribution $\mathcal{Z}$. In our construction, $\mathcal{Z}$ consists of an obfuscated Turing machine.

We emphasize that we provide a single distribution $\mathcal{Z}$ of auxiliary inputs for which *all* candidate EOWF families $\mathcal{F}$ with given output length will fail. This is in contrast to the result of [BCPR14], which show for each candidate family $\mathcal{F}$ that there exists a tailored distribution $\mathcal{Z}_{\mathcal{F}}$ (whose size grows with $|\mathcal{F}|$) for which $\mathcal{F}$ will fail.

Theorem 3.1. *For every polynomial ℓ, there exists an efficient, uniformly samplable distribution $\mathcal{Z}$ such that, assuming the existence of public-coin collision-resistant hash functions and public-coin differing-inputs obfuscation for Turing machines, then there* cannot *exist $\mathcal{Z}$-auxiliary-input extractable one-way functions $\{f_i : \{0,1\}^k \to \{0,1\}^{\ell(k)}\}$.*

Proof. We construct an adversary $\mathcal{A}$ and desired distribution $\mathcal{Z}$ on auxiliary inputs, such that for any alleged EOWF family $\mathcal{F}$, there cannot exist an efficient extractor corresponding to $\mathcal{A}$ given auxiliary input from $\mathcal{Z}$ (assuming public-coin CRHFs and $\mathsf{PC}\text{-}di\mathcal{O}$).

The Universal Adversary $\mathcal{A}$. We consider a universal PPT adversary $\mathcal{A}$ that, given $(i, z) \in \{0,1\}^{\mathsf{poly}(k)} \times \{0,1\}^{n(k)}$, parses z as a Turing machine and returns $z(i)$. Note that in our setting, i corresponds to the index of the selected function $f_i \in \mathcal{F}$, and (looking ahead) the auxiliary input z will contain an obfuscated program.

The Auxiliary Input Distribution $\mathcal{Z}$. Let $\mathcal{PRF} = \{\mathsf{PRF}_s : \{0,1\}^{m(k)} \to \{0,1\}^k\}_{s \in \{0,1\}^k}$ be a puncturable pseudorandom function family, and $\mathcal{H} = \{\mathcal{H}_k\}$ a public-coin collision-resistant hash function family with $h : \{0,1\}^* \to \{0,1\}^{m(k)}$ for each $h \in \mathcal{H}_k$. (Note that by Theorem A.5, punctured PRFs for these parameters exist based on OWFs, which are implied by CRHF). We begin by defining two classes of *Turing machines*:

$$\mathcal{M} = \left\{\Pi^{h,s} \;\middle|\; s \in \{0,1\}^k,\ h \in \mathcal{H}_k,\ k \in \mathbb{N}\right\},$$
$$\mathcal{M}^* = \left\{\Pi^{h,s}_{i,y} \;\middle|\; s \in \{0,1\}^k,\ y \in \{0,1\}^{\ell(k)},\ h \in \mathcal{H}_k,\ k \in \mathbb{N}\right\},$$

Turing Machine $\Pi^{h,s}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, PRF seed $s \in \{0,1\}^k$.

Inputs: Circuit description f_i

1. Hash the index: $v = h(i)$.
2. Compute the PRF on this hash: $x = \mathsf{PRF}_s(v)$.
3. Output the evaluation of the universal Turing machine on inputs f_i, x: i.e., $y = U_k(f_i, x)$.

Figure 1: Turing machines $\Pi^{h,s} \in \mathcal{M}$.

Auxiliary Input Distribution $\mathcal{Z}_k$:

1. Sample a hash function $h \leftarrow \mathcal{H}_k$ and PRF seed $s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k)$.
2. Output an obfuscation $\tilde{\Pi} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(\Pi^{h,s})$.

Figure 2: The auxiliary input distribution $\mathcal{Z}_k$.

which we now describe. We assume without loss of generality for each k that the corresponding collection of Turing machines $\Pi^{h,s} \in \mathcal{M}_k$, $\Pi^{h,s}_{i,y} \in \mathcal{M}^*_k$ are of the *same size*; this can be achieved by padding. (We address the size bound of each class of machines below). In a similar fashion, we may further assume that for each k the runtime of each $\Pi^{h,s}$ and $\Pi^{h,s}_{i,y}$ on any given input f_i is equal (see discussion in Section 2.2).

At a high level, each machine $\Pi^{h,s}$ accepts as input a poly-size circuit description of a function f_i (with canonical description, including a function index i), computes the hash of the corresponding index i w.r.t. the hardcoded hash function h, applies a PRF with hardcoded seed s to the hash, and then evaluates the circuit f_i on the resulting PRF output value x: that is, $\Pi^{h,s}_{i,y}(f_i)$ outputs $U_k(f_i, \mathsf{PRF}_s(h(i)))$, where U_k is the universal Turing machine. See Figure 1. Note that each $\Pi^{h,s}$ can be described by a Turing machine of size $O(|s| + |h| + |U_k|)$, which is bounded by $p(k)$ for some fixed polynomial p.

The machines $\Pi^{h,s}_{i,y}$ perform a similar task, except that instead of having the entire PRF seed s hardcoded, they instead only have a *punctured* seed s^* derived from s by puncturing it at the point $h(i)$ (i.e., enabling evaluation of the PRF on all points except $h(i)$). In addition, it has hardwired an output y to replace the punctured result. More specifically, on input a circuit description f_j (with explicitly specified index j), the program $\Pi^{h,s}_{i,y}$ first computes the hash $h = h(j)$, continues computation as usual for any $h \neq h(i)$ using the punctured PRF key, and for $h = h(i)$, it skips the PRF and U_k evaluation steps and directly outputs y. Note that because h is not injective, this puncturing may change the value of the program on multiple inputs f_j (corresponding to functions $f_j \in \mathcal{F}$ with $h(j) = h(i)$). When the hardcoded value y is set to $y = f_i(\mathsf{PRF}_s(h(i)))$, then $\Pi^{h,s}_{i,y}$ agrees with $\Pi^{h,s}$ additionally on the input f_i, but not necessarily on the other inputs f_j for which $h(j) = h(i)$. (Indeed, whereas the hash of their indices collide, and thus their corresponding PRF outputs, $\mathsf{PRF}(h(j))$, will agree, the final step will apply *different* functions f_j to this value).

We first remark that indistinguishability obfuscation arguments will thus not apply to this scenario, since we are modifying the computed functionality. In contrast, differing-inputs obfuscation would guarantee that the two obfuscated programs are indistinguishable, since otherwise we could efficiently *find* one of the disagreeing inputs, which would correspond to a collision in the CRHF. But, most importantly, this argument holds even if *the randomness used to sample the program pair* $(\Pi^{h,s}, \Pi^{h,s}_{i,y})$ *is revealed.* Namely, we consider a program sampler that generates pairs $(\Pi^{h,s}, \Pi^{h,s}_{i,y})$ of the corresponding distribution; this amounts to sampling a hash function h, an EOWF challenge index i, and a PRF seed s, and a $h(i)$-puncturing of the seed, s^*. All remaining values specifying the programs, such as $y = f_i(\mathsf{PRF}_s(h(i)))$, are deterministically computed given (h, i, s, s^*). Now, since $\mathcal{H}$ is a public-coin CRHF family, revealing the randomness used to sample

Turing Machine $\Pi^{h,s}_{i,y}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, punctured PRF seed $s^* \in \{0,1\}^k$, punctured point $h(i)$, bit string $y \in \{0,1\}^{\ell(k)}$.

Input: Circuit description f_j (containing index j)

```
1. Hash the index: $v = h(j)$.
2. If $v \neq h(i)$, compute $x = \mathsf{PRF}_{s^*}(v)$, and output $U_k(f_j, x)$.
3. If $v = h(i)$, output $y$.
```

Figure 3: "Punctured" Turing machines $\Pi^{h,s}_{i,y} \in \mathcal{M}^*$.

Auxiliary Input Distribution $\mathcal{Z}_k(i,y)$:

1. Sample a hash function $h \leftarrow \mathcal{H}_k$ and PRF seed $s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k)$.
2. Sample a punctured PRF seed $s^* \leftarrow \mathsf{Punct}(s, h(i))$, punctured at point $h(i)$.
3. Compute the "correct" punctured evaluation: $y = f_i(\mathsf{PRF}_s(h(i)))$.
4. Output an obfuscation $\tilde{M} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(\Pi^{h,s}_{i,y})$, where $\Pi^{h,s}_{i,y}$ is defined from (h, s^*, y), as in Figure 13.

Figure 4: The "punctured" distribution $\mathcal{Z}_k(i,y)$.

$h \leftarrow \mathcal{H}$ is not detrimental to its collision resistance. And, the values i, s, and s^* are completely *independent* of the CRHF security (i.e., a CRHF adversary reduction could simply generate them on its own in order to break h). Therefore, we ultimately need only rely on *public-coin* $di\mathcal{O}$.

We finally consider the size of the program(s) to be obfuscated. Note that each $\Pi^{h,s}_{i,y}$ can be described by a Turing machine of size $O(|s^*| + |h| + |y| + |U_k|)$. Recall by Theorem A.5 the size of the punctured PRF key $|s^*| \in O(m'(k)\ell(k))$, where the PRF has input and output lengths $m'(k)$ and $\ell(k)$. In our application, note that the input to the PRF is not the function index i itself (in which case the machine $\Pi^{h,s}_{i,y}$ would need to grow with the size of the alleged EOWF family), but rather the *hashed* index $h(i)$, which is of fixed polynomial length. Thus, collectively, we have $|\Pi^{h,s}_{i,y}|$ is bounded by a fixed polynomial $p'(k)$, and finally that there exists a single fixed polynomial bound on the size of *all* programs $\Pi^{h,s} \in \mathcal{M}, \Pi^{h,s}_{i,y} \in \mathcal{M}^*$. This completely determines the auxiliary input distribution $\mathcal{Z} = \{\mathcal{Z}_k\}$, described in full in Figure 2. (Note that the size of the auxiliary output generated by $\mathcal{Z}$, which corresponds to an obfuscation of an appropriately padded program $\Pi^{h,s}$ is thus also bounded by a fixed polynomial in k).

$\mathcal{A}$ Has No Extractor. We show that, based on the assumed security of the underlying tools, the constructed adversary $\mathcal{A}$ given auxiliary input from the constructed distribution $\mathcal{Z} = \{Z_k\}$, cannot have an extractor $\mathcal{E}$ satisfying Definition 2.5:

Proposition 3.2. *For any non-uniform polynomial-time candidate extractor $\mathcal{E}$ for $\mathcal{A}$, it holds that $\mathcal{E}$ fails with overwhelming probability: i.e.,*

$$\Pr\left[z \leftarrow \mathcal{Z}_k;\ i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ y \leftarrow \mathcal{A}(i;z);\ x' \leftarrow \mathcal{E}(i;z) : \exists x \text{ s.t. } f_i(x) = y \wedge f_i(x') \neq y\right] \geq 1 - \mathsf{negl}(k).$$

Proof. First note that given auxiliary input $z \leftarrow \mathcal{Z}_k$, $\mathcal{A}$ produces an element in the image of the selected f_i with high probability. That is,

$$\Pr\left[z \leftarrow \mathcal{Z}_k; i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k); y \leftarrow \mathcal{A}(i;z) : \exists x \text{ s.t. } f_i(x) = y\right] \geq 1 - \mathsf{negl}(k).$$

Indeed, by the definition of $\mathcal{A}$ and $\mathcal{Z}_k$, and the correctness of the obfuscator $\mathsf{PC}\text{-}di\mathcal{O}$, then we

Program Pair Sampler $\mathsf{Samp}(1^k, r)$:

1. Sample a hash function $h = \mathcal{H}_k(r_h)$.
2. Sample an EOWF index $i = K_{\mathcal{F}}(1^k; r_i)$.
3. Sample a PRF seed $s = K_{\mathsf{PRF}}(1^k; r_s)$.
4. Sample a punctured PRF seed $s^* = \mathsf{Punct}(s, h(i); r_*)$.
5. Let $y = f_i(\mathsf{PRF}_s(h(i)))$.
6. Denote $r := (r_h, r_i, r_s, r_*)$.
7. Output program pair $(\Pi^{h,s}, \Pi^{h,s}_{i,y})$, defined by h, i, s, s^*, y as above (and padded to equal length).

Figure 5: Program pair sampler algorithm, to be used in public-coin differing inputs security step.

have with overwhelming probability

$$\mathcal{A}(i; z) = \tilde{M}(f_i) = \Pi^{h,s}(f_i) = f_i(\mathsf{PRF}_s(h(i))),$$

where $z = \tilde{M}$ is an obfuscation of $\Pi^{h,s} \in \mathcal{M}$; i.e., $z = \tilde{M} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(\Pi^{h,s})$.

Now, suppose for contradiction that there exists a non-negligible function $\epsilon(k)$ such that for all $k \in \mathbb{N}$ the extractor $\mathcal{E}$ successfully outputs a preimage corresponding to the output $\mathcal{A}(i; z) \in Range(f_i)$ with probability $\epsilon(k)$: i.e.,

$$\Pr\left[z \leftarrow \mathcal{Z}_k;\ i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ x' \leftarrow \mathcal{E}(i; z) : f_i(x') = \mathcal{A}(i; z) = f_i(\mathsf{PRF}_s(h(i)))\right] \geq \epsilon(k).$$

where as before, s, h are such that $z = \mathsf{PC}\text{-}di\mathcal{O}(\Pi^{h,s})$. We show that this cannot be the case, via three steps.

Step 1: Replace $\mathcal{Z}$ with "punctured" distribution $\mathcal{Z}(i, y)$. For every index i of the EOWF family $\mathcal{F}$ and $k \in \mathbb{N}$, consider an alternative distribution $\mathcal{Z}_k(i, y)$ that, instead of sampling and obfuscating a Turing machine $\Pi^{h,s}$ from the class $\mathcal{M}$, as is done for $\mathcal{Z}$, it does so with a Turing machine $\Pi^{h,s}_{i,y} \in \mathcal{M}^*$ as follows. First, it samples a hash function $h \leftarrow \mathcal{H}_k$ and PRF seed s as usual. It then generates a *punctured* PRF key $s^* \leftarrow \mathsf{Punct}(s, h(i))$ that enables evaluation of the PRF on all points except the value $h(i)$. For the specific index i, it computes the correct full evaluation $y := f_i(\mathsf{PRF}_s(h(i)))$. Finally, $\mathcal{Z}_k(i, y)$ outputs an obfuscation of the constructed program $\Pi^{h,s}_{i,y}$ as specified in Figure 13 from the values (h, s^*, y): i.e., $\tilde{M} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(\Pi^{h,s}_{i,y})$. See Figure 4 for a full description of $\mathcal{Z}(i, y)$.

We now argue that the extractor $\mathcal{E}$ must also succeed in extracting a preimage when given a value $z^* \leftarrow \mathcal{Z}_k(i, y)$ from this modified distribution instead of $\mathcal{Z}_k$.

Consider the Turing Machine program sampler algorithm Samp as in Figure 5.

We first argue that, by the (public-coin) collision resistance of the hash family $\mathcal{H}$, the sampler algorithm Samp is a *public-coin differing-inputs sampler*, as per Definition **??**.

Claim 3.3. Samp *is a public-coin differing-inputs sampler. That is, for all efficient non-uniform* $\mathcal{A}_{\mathsf{PC}}$*, there exists a negligible function* ϵ *such that for all* $k \in \mathbb{N}$*,*

$$\Pr\left[r \leftarrow \{0,1\}^*; (M_0, M_1) \leftarrow \mathsf{Samp}(1^k, r); (x, 1^t) \leftarrow \mathcal{A}_{\mathsf{PC}}(1^k, r) : \right.$$
$$\left. M_0(x) \neq M_1(x) \wedge \mathsf{steps}(M_0, x) = \mathsf{steps}(M_1, x) = t\right] \leq \epsilon(k). \quad (1)$$

Proof. Suppose, to the contrary, there exists an efficient (non-uniform) adversary $\mathcal{A}_{\mathsf{PC}}$ and non-negligible function $\alpha(k)$ for which the probability in Equation 1 is greater than $\alpha(k)$. We show such an adversary contradicts the security of the (public-coin) CRHF. Consider an adversary $\mathcal{A}_{\mathsf{CR}}$ in the CRHF security challenge. Namely, for a challenge hash function $h \leftarrow \mathcal{H}_k(r_h)$, the adversary $\mathcal{A}_{\mathsf{CR}}$ receives h, r_h, and performs the following steps:

CRHF adversary $\mathcal{A}_{\mathsf{CR}}(1^k, h, r_h)$:

1. Imitate the remaining steps of Samp. That is, sample an EOWF index $i = K_{\mathcal{F}}(1^k; r_i)$; a PRF seed $s = K_{\mathsf{PRF}}(1^k; r_s)$; and a punctured PRF seed $s^* = \mathsf{Punct}(s, h(i); r_*)$. Define $y = f_i(\mathsf{PRF}_s(h(i)))$ and $r = (r_h, r_i, r_s, r_*)$, and let $M_0 = \Pi^{h,s}$ and $M_1 = \Pi^{h,s}_{i,y}$.
2. Run $\mathcal{A}_{\mathsf{PC}}(1^k, r)$ on the collection of randomness r used above. In response, $\mathcal{A}_{\mathsf{PC}}$ returns a pair $(x, 1^t)$.
3. $\mathcal{A}_{\mathsf{CR}}$ outputs the pair (i, x) as an alleged collision in the challenge hash function h.

Now, by assumption, the value x generated by $\mathcal{A}_{\mathsf{PC}}$ satisfies (in particular) that $M_0(x) \neq M_1(x)$. From the definition of M_0, M_1 (i.e., $\Pi^{h,s}, \Pi^{h,s}_{i,y}$), this must mean that $h(i) = h(x)$ (since all values with $h(x) \neq h(i)$ were not changed from $\Pi^{h,s}$ to $\Pi^{h,s}_{i,y}$), *and* that $i \neq x$ (since $\Pi^{h,s}_{i,y}(i)$ was specifically "patched" to the correct output value $\Pi^{h,s}(i)$). That is, $\mathcal{A}_{\mathsf{CR}}$ successfully identifies a collision with the same probability $\alpha(k)$, which must thus be negligible.

□

We now show that this implies, by the security of the public-coin $di\mathcal{O}$, that our original EOWF extractor $\mathcal{E}$ must succeed with nearly equivalent probability in the EOWF challenge when instead of receiving (real) auxiliary input from $\mathcal{Z}_k$, both $\mathcal{E}$ and $\mathcal{A}$ are given auxiliary input from the fake distribution $\mathcal{Z}_k(i, y)$. (Recall that ϵ is assumed to be $\mathcal{E}$'s success in the same experiment as below but with $z \leftarrow \mathcal{Z}_k$ instead of $z^* \leftarrow \mathcal{Z}_k(i, y)$).

Lemma 3.4. *It holds that*

$$\Pr\Big[i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ z^* \leftarrow \mathcal{Z}_k(i, y);\ x' \leftarrow \mathcal{E}(i; z^*) : \\ f_i(x') = \mathcal{A}(i; z^*) = f_i(\mathsf{PRF}_s(h(i)))\Big] \geq \epsilon(k) - \mathsf{negl}(k). \quad (2)$$

Proof. Note that given $z^* \leftarrow \mathcal{Z}_k(i, y)$ (which corresponds to an obfuscated program of the form $\Pi^{h,s}_{i,y}$) our EOWF adversary $\mathcal{A}$ indeed will still output $\Pi^{h,s}_{i,y}(i) = y := f_i(\mathsf{PRF}_s(h(i)))$ (see Figures 13,4).

Now, suppose there exists a non-negligible function $\alpha(k)$ for which the probability in Equation (2) is less than $\epsilon(k) - \alpha(k)$. We directly use such $\mathcal{E}$ to design another adversary $\mathcal{A}_{di\mathcal{O}}$ to contradict the security of the public-coin $di\mathcal{O}$ with respect to the program pair sampler Samp (which we showed in Claim 3.3 to be a void public-coin differing inputs sampler). Recall the $di\mathcal{O}$ challenge samples a program pair $(\Pi^{h,s}, \Pi^{h,s}_{i,y}) \leftarrow \mathsf{Samp}(1^k, r)$, selects a random $M \leftarrow \{\Pi^{h,s}, \Pi^{h,s}_{i,y}\}$ to obfuscate as $\tilde{M} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, M)$, and gives as a challenge the pair $(r, \tilde{M})$ of the randomness used by Samp and obfuscated program. Define $\mathcal{A}_{di\mathcal{O}}$ (who wishes to distinguish which program was selected) as follows.

PC-$di\mathcal{O}$ adversary $\mathcal{A}_{di\mathcal{O}}(1^k, r, \tilde{M})$:

1. Parse the given randomness r used in Samp as $r = (r_h, r_i, r_s, r_*)$ (see Figure 5).
2. Recompute the corresponding "challenge index" $i = K_{\mathcal{F}}(1^k; r_i)$. Let $z^* = \tilde{M}$.
3. Run the extractor algorithm $\mathcal{E}(i; z^*)$, and receive back an alleged preimage x'.
4. Recompute $h = \mathcal{H}_k(r_h)$, $s = K_{\mathsf{PRF}}(1^; r_s)$, again using the randomness from r.
5. If $f_i(x') = f_i(\mathsf{PRF}_s(h(i)))$ — i.e., if $\mathcal{E}$ succeeded in extracting a preimage — then $\mathcal{A}_{di\mathcal{O}}$ outputs 1. Otherwise, $\mathcal{A}_{di\mathcal{O}}$ outputs 0.

Now, if $\tilde{M}$ is an obfuscation of $\Pi^{h,s}$, then this experiment corresponds directly to the EOWF challenge where $\mathcal{E}$ (and $\mathcal{A}$) is given auxiliary input $z \leftarrow \mathcal{Z}_k$. On the other hand, if $\tilde{M}$ is an obfuscation of $\Pi^{h,s}_{i,y}$, then the experiment corresponds directly to the same challenge where $\mathcal{E}$ (and $\mathcal{A}$) is given auxiliary input $z^* \leftarrow \mathcal{Z}_k(i, y)$. Thus, $\mathcal{A}_{di\mathcal{O}}$ will succeed in distinguishing these two cases with probability at least $[\epsilon(k)] - [\epsilon(k) - \alpha(k)] = \alpha(k)$. By the security of $\mathsf{PC}\text{-}di\mathcal{O}$, it hence follows that $\alpha(k)$ must be negligible.

□

Step 2: Replace "correct" hardcoded y in $\mathcal{Z}(i,y)$ with random f_i evaluation. Next, we consider another experiment where $\mathcal{Z}_k(i,y)$ is altered to a nearly identical distribution $\mathcal{Z}_k(i,u)$ where, instead of hardcoding the "correct" i-evaluation value $y = f_i(\mathsf{PRF}_s(h(i)))$ in the generated "punctured" program $\Pi^{h,s}_{i,y}$, the distribution $\mathcal{Z}_k(i,u)$ now simply samples a random f_i output $y = f_i(u)$ for an independent random $u \leftarrow \{0,1\}^k$. We claim that the original EOWF extractor $\mathcal{E}$ still succeeds in finding a preimage when given this new auxiliary input distribution:

Lemma 3.5. *It holds that*

$$\Pr\Big[i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ z^{**} \leftarrow \mathcal{Z}_k(i,u);\ x' \leftarrow \mathcal{E}(i;z^{**}) : \\ f_i(x') = \mathcal{A}(i;z^{**}) = f_i(u)\Big] \geq \epsilon(k) - \mathsf{negl}(k). \quad (3)$$

Proof. This follows from the fact that $\mathsf{PRF}_s(h(i))$ is pseudorandom, even given the $h(i)$-punctured key s^*.

Formally, consider an algorithm $\mathcal{A}^0_{\mathsf{PRF}}$ which, on input the security parameter 1^k, a pair of values i, h, and a pair s^*, x (that will eventually correspond to a challenge punctured PRF key, and either $\mathsf{PRF}_s(h(i))$ or random u), performs the following steps.

Algorithm $\mathcal{A}^0_{\mathsf{PRF}}(1^k, i, h, s^*, x)$:

1. Take $y = f_i(x)$, and obfuscate the associated program $\Pi^{h,s}_{i,y}$: i.e., $z^{**} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, \Pi^{h,s}_{i,y})$.
2. Run the EOWF extractor given index i and auxiliary input z^{**}: $x' \leftarrow \mathcal{E}(i;z^{**})$.
3. Output 0 if $\mathcal{E}$ succeeds in extracting a valid preimage: i.e., if $f_i(x') = y^* = f_i(x)$. Otherwise, output a random bit $b \leftarrow \{0,1\}$.

Now, suppose Lemma 3.5 does not hold: i.e., the probability in Equation (3) differs by some non-negligible amount from $\epsilon(k)$. Then, expanding out the sampling procedure of $\mathcal{Z}_k(i,y)$ and $\mathcal{Z}_k(i,u)$, we have for some non-negligible function $\alpha(k)$ that

$$\Pr\Big[i \leftarrow \mathcal{K}_{\mathcal{F}}(1^k);\ h \leftarrow \mathcal{H}_k;\ s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k);\ s^* \leftarrow \mathsf{Punct}(s, h(i));\ u \leftarrow \{0,1\}^k; \\ b \leftarrow \{0,1\} : \mathcal{A}^0_{\mathsf{PRF}}(1^k, i, h, x_b) = b\Big] \geq \frac{1}{2} + \alpha(k), \quad (4)$$

where $x_0 := \mathsf{PRF}_s(h(i))$ and $x_1 := u$. Indeed, in the case $b = 0$, the auxiliary input z^{**} generated by $\mathcal{A}_{\mathsf{PRF}}$ and given to $\mathcal{E}$ has distribution exactly $\mathcal{Z}(i,y)$, whereas in the case $b = 1$, the generated z^{**} has distribution exactly $\mathcal{Z}(i,u)$.

In particular, there exists a polynomial $p(k)$ such that for infinitely many k, there exists an index i_k and hash function $h_k \in \mathcal{H}_k$ with

$$\Pr\Big[s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k);\ s^* \leftarrow \mathsf{Punct}(s, h(i_k));\ u \leftarrow \{0,1\}^k; \\ b \leftarrow \{0,1\} : \mathcal{A}^0_{\mathsf{PRF}}(1^k, i_k, h, x_b) = b\Big] \geq \frac{1}{2} + \frac{1}{p(k)}, \quad (5)$$

where x_0, x_1 are as before.

Consider a non-uniform punctured-PRF adversary $\mathcal{A}^I_{\mathsf{PRF}}$ (with the ensemble $I = \{i_k, h_k\}$ hardcoded) that first selects the challenge point $h_k(i_k)$; receives the PRF challenge information (s^*, x) for this point; executes $\mathcal{A}^0_{\mathsf{PRF}}$ on input $(1^k, i_k, h_k, s^*, x)$, and outputs the corresponding bit b output by $\mathcal{A}^0_{\mathsf{PRF}}$. Then by (5), it follows that $\mathcal{A}^I_{\mathsf{PRF}}$ breaks the security of the punctured PRF. □

Step 3: Such an extractor breaks one-wayness of EOWF. Finally, we observe that this means that $\mathcal{E}$ can be used to break the one-wayness of the original function family $\mathcal{F}$. Indeed, given a random key i and a challenge output $y = f_i(u)$, an inverter can simply sample a hash function h and $h(i)$-punctured PRF seed s^* on its own, construct the program $\Pi^{h,s}_{i,y}$ with its challenge y hardcoded in, and sample an obfuscation $z^{**} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(\Pi^{h,s}_{i,y})$. Finally, it runs $\mathcal{E}(i, z^{**})$ to invert y^*, with the same probability $\epsilon(k) - \mathsf{negl}(k)$.

□

This concludes the proof of Theorem 3.1. □

3.2 PC-$di\mathcal{O}$ or SNARKs

We link the existence of public-coin differing-inputs obfuscation for NC^1 and the existence of succinct non-interactive arguments of knowledge (SNARKs), via an intermediate step of *proximity* extractable one-way functions (PEOWFs), a notion related to EOWFs, introduced in [BCCT12]. Namely, *assume the existence of fully homomorphic encryption (FHE) with decryption in NC^1 and public-coin collision-resistant hash functions.* Then, building upon the results of the previous subsection, and the results of [IPS15, BCCT12], we show:

1. Assuming SNARKs for NP, there exists an efficient distribution $\mathcal{Z}$ such that public-coin differing-inputs obfuscation for NC^1 implies that there *cannot* exist PEOWFs $\{f : \{0,1\}^k \to \{0,1\}^k\}$ w.r.t. $\mathcal{Z}$.
2. PEOWFs $\{f : \{0,1\}^k \to \{0,1\}^k\}$ w.r.t. this auxiliary input distribution $\mathcal{Z}$ are *implied by* the existence of SNARKs for NP secure w.r.t. a second efficient auxiliary input distribution $\mathcal{Z}'$, as shown in [BCCT12].
3. Thus, one of these conflicting hypotheses must be false. That is, there exists an efficient distribution $\mathcal{Z}'$ such that assuming existence of FHE with decryption in NC^1 and collision-resistant hash functions, then either: (1) public-coin differing-inputs obfuscation for NC^1 does not exist, or (2) SNARKS for NP w.r.t. $\mathcal{Z}'$ do not exist.

Note that we focus on the specific case of PEOWFs with k-bit inputs and k-bit outputs, as this suffices to derive the desired contradiction; however, the theorems following extend also to the more general case of PEOWF output length (demonstrating an efficient distribution $\mathcal{Z}$ to rule out each potential output length $\ell(k)$).

3.2.1 Proximity EOWFs

We begin by defining Proximity EOWFs.

Proximity Extractable One-Way Functions (PEOWFs). In a Proximity EOWF (PEOWF), the extractable function family $\{f_i\}$ is associated with a "proximity" equivalence relation $\sim$ on the range of f_i, and the one-wayness and extractability properties are modified with respect to this relation. The one-wayness is strengthened: not only must it be hard to find an exact preimage of v, but it is also hard to find a preimage of any equivalent $v \sim v'$. The extractability requirement is weakened accordingly: the extractor does not have to output an exact preimage of v, but only a preimage of of some equivalent value $v' \sim v$.

As an example, consider functions of the form $f : x \mapsto (f_1(x), f_2(x))$ and equivalence relation on range elements $(a, b) \sim (a, b')$ whose first components agree. Then the proximity extraction property requires for any adversary $\mathcal{A}$ who outputs an image element $(a, b) \in Range(f)$ that there exists an extractor $\mathcal{E}$ finding an input x s.t. $f(x) = (a, b')$ for some b' not necessarily equal to b.

In this work, we allow the relation $\sim$ to depend on the function index i, but require that the relation $\sim$ is *publicly* (and efficiently) testable. We further consider non-uniform adversaries and extraction algorithms, and (in line with this work) auxiliary inputs coming from a specified distribution $\mathcal{Z}$.

Definition 3.6 ($\mathcal{Z}$-Auxiliary-Input Proximity EOWFs)**.** Let ℓ, m be polynomially bounded length functions. An efficiently computable family of functions

$$\mathcal{F} = \left\{ f_i : \{0,1\}^k \to \{0,1\}^{\ell(k)} \;\middle|\; i \in \{0,1\}^{m(k)}, k \in \mathbb{N} \right\},$$

associated with an efficient probabilistic key sampler $\mathcal{K}_\mathcal{F}$, is a *$\mathcal{Z}$-auxiliary-input proximity extractable one-way function* if it satisfies the following (strong) one-wayness, (weak) extraction, and public testability properties:

- **(Strengthened) One-wayness:** For non-uniform polynomial-time $\mathcal{A}$ and sufficiently large security parameter $k \in \mathbb{N}$,

$$\Pr\left[z \leftarrow \mathcal{Z}_k;\ i \leftarrow \mathcal{K}_\mathcal{F}(1^k);\ x \leftarrow \{0,1\}^k;\ x' \leftarrow \mathcal{A}(i, f_i(x); z) : f_i(x') \sim f_i(x)\right] \leq \mathsf{negl}(k).$$

- **(Weakened) Extractability:** For any non-uniform polynomial-time adversary $\mathcal{A}$, there exists a non-uniform polynomial-time extractor $\mathcal{E}$ such that, for sufficiently large security parameter $k \in \mathbb{N}$,

$$\Pr\left[z \leftarrow \mathcal{Z}_k;\ i \leftarrow \mathcal{K}_\mathcal{F}(1^k);\ y \leftarrow \mathcal{A}(i; z);\ x' \leftarrow \mathcal{E}(i; z) : \exists x \text{ s.t. } f_i(x) = y \wedge f_i(x') \not\sim y\right] \leq \mathsf{negl}(k).$$

- **Publicly Testable Relation:** There exists a deterministic polytime machine $\mathcal{T}$ such that, given the function index i, $\mathcal{T}$ accepts $y, y' \in \{0,1\}^{\ell(k)}$ if and only if $y \sim_k y'$.

3.2.2 (PC-$di\mathcal{O}$ for NC^1 + PC-CRHF + FHE + SNARK) $\Rightarrow$ *No* $\mathcal{Z}$-PEOWF

We now show that, assuming the existence of public-coin collision-resistant hash functions (CRHF) and fully homomorphic encryption (FHE) with decryption in NC^1,[10] then for some efficiently computable distributions $\mathcal{Z}_\mathsf{SNARK}, \mathcal{Z}_\mathsf{PEOWF}$, if there exist public-coin differing-inputs obfuscators for NC^1 circuits, and SNARKs w.r.t. auxiliary input $\mathcal{Z}_\mathsf{SNARK}$, then there *cannot* exist PEOWFs w.r.t. auxiliary input $\mathcal{Z}_\mathsf{PEOWF}$. This takes place in two steps.

First, we remark that an identical proof to that of Theorem 3.1 rules out the existence of $\mathcal{Z}$-auxiliary-input *proximity EOWFs* in addition to standard EOWFs, based on the same assumptions: namely, assuming public-coin differing-inputs obfuscation for Turing machines, and public-coin collision-resistant hash functions. Indeed, assuming the existence of a PEOWF extractor $\mathcal{E}$ for the adversary $\mathcal{A}$ and auxiliary input distribution $\mathcal{Z}$ (who extracts a "related" preimage to the target value), the same procedure yields a PEOWF inverter who similarly extracts a "related" preimage to any challenge output. In the reduction, it is merely required that the success of $\mathcal{E}$ is efficiently and publicly testable (this is used to construct a distinguishing adversary for the differing-inputs obfuscation scheme, in Step 1). However, this is directly implied by the public testability of the PEOWF relation $\sim$, as specified in Definition 3.6.

Theorem 3.7. *There exist an efficient, uniformly samplable distribution $\mathcal{Z}$ such that, assuming the existence of public-coin collision-resistant hash functions and public-coin differing-inputs obfuscation for polynomial-size Turing machines, there cannot exist (publicly testable) $\mathcal{Z}$-auxiliary-input PEOWFs $\{f_i : \{0,1\}^k \to \{0,1\}^k\}$.*

Now, in [IPS15], it was shown that public-coin differing-inputs obfuscation for the class of all polynomial-time Turing machines can be achieved by bootstrapping up from public-coin differing-inputs obfuscation for circuits in the class NC^1, assuming the existence of FHE with decryption in NC^1, public-coin CRHF, and public-coin SNARKs for NP.

Putting this together with Theorem 3.7, we thus have the following corollary.

Corollary 3.8. *There exists an efficient, uniformly samplable distribution $\mathcal{Z}$ s.t., assuming existence of public-coin SNARKs and FHE with decryption in NC^1, then assuming the existence of public-coin differing-inputs obfuscation for NC^1, there cannot exist PEOWFs $\{f_i : \{0,1\}^k \to \{0,1\}^k\}$ w.r.t. auxiliary input $\mathcal{Z}$.*

[10] As is the case for nearly all existing FHE constructions (e.g., [GSW13, BV14]).

3.2.3 (SNARK + CRHF) $\implies$ $\mathcal{Z}$-PEOWF

As shown in [BCCT12], Proximity EOWFs (PEOWFs) with respect to an auxiliary input distribution $\mathcal{Z}$ are *implied by* collision-resistant hash functions (CRHF) and SNARKs secure with respect to a related auxiliary input distribution $\mathcal{Z}'$.[11]

Loosely, the transformation converts any CRHF family $\mathcal{F}$ into a PEOWF by appending to the output of each $f \in \mathcal{F}$ a succinct SNARK argument π_x that there exists a preimage x yielding output $f(x)$. (If the Prover algorithm of the SNARK system is randomized, then the function is also modified to take an additional input, which is used as the random coins for the SNARK generation). The equivalence relation on outputs is defined by $(y, \pi) \sim (y', \pi')$ if $y = y'$ (note that this relation is publicly testable). More explicitly, consider the new function family $\mathcal{F}'$ composed of functions

$$f'_{\mathsf{crs}}(x, r) = \big(f(x), \mathsf{Prove}(1^k, \mathsf{crs}, f(x), x; r)\big)\,,$$

where a function $f'_{\mathsf{crs}} \in \mathcal{F}'$ is sampled by first sampling a function $f \leftarrow \mathcal{F}$ from the original CRHF family, and then sampling a CRS for the SNARK scheme, $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^k)$.

Now (as proved in [BCCT12]), the resulting function family will be a PEOWF with respect to auxiliary input $\mathcal{Z}$ if the underlying SNARK system is secure with respect to an augmented auxiliary input distribution $\mathcal{Z}_{\mathsf{SNARK}} := (\mathcal{Z}, h)$, formed by concatenating a sample from $\mathcal{Z}$ with a function index h sampled from the collision-resistant hash function family $\mathcal{F}$. (Note that we will be considering public-coin CRHF, in which case h is uniform).

Theorem 3.9 ([BCCT12])**.** *There exist efficient, uniformly samplable distributions $\mathcal{Z}, \mathcal{Z}_{\mathsf{SNARK}}$ such that, assuming the existence of collision-resistant hash functions and SNARKs for* NP *secure w.r.t. auxiliary input distribution $\mathcal{Z}_{\mathsf{SNARK}}$, then there exist PEOWFs $\{f_i : \{0,1\}^k \to \{0,1\}^k\}$ w.r.t. $\mathcal{Z}$.*

3.2.4 Reaching a Standoff

Observe that the conclusions of Corollary 3.8 and Theorem 3.9 are in direct contradiction. Thus, it must be that one of the two sets of assumptions is false. Namely,

Corollary 3.10. *Assuming the existence of public-coin collision-resistant hash functions and fully homomorphic encryption with decryption in NC^1, there exists an efficiently samplable distribution $\mathcal{Z}_{\mathsf{SNARK}}$ such that one of the following two objects cannot exist:*

- *SNARKs w.r.t. auxiliary input distribution $\mathcal{Z}_{\mathsf{SNARK}}$.*
- *Public-coin differing-inputs obfuscation for NC^1.*

More explicitly, we have that $\mathcal{Z}_{\mathsf{SNARK}} = (\mathcal{Z}, U)$, where $\mathcal{Z}$ is composed of an obfuscated program, and U is a uniform string (corresponding to a randomly sampled index from a public-coin CRHF family).

References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *EUROCRYPT*, pages 171–188, 2004.

[11] [BCCT12] consider the setting of arbitrary auxiliary input; however, their construction directly implies similar results for specific auxiliary input distributions.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC*, pages 52–73, 2014.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC 2014*, pages 505–514, 2014.

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography*, pages 501–519, 2014.

[BLV06] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. *J. Comput. Syst. Sci.*, 72(2):321–391, 2006.

[BP13] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. Cryptology ePrint Archive, Report 2013/703, 2013.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 1–12, 2014.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security*, pages 280–300, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*, pages 480–499, 2014.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Extractable perfectly one-way functions. In *ICALP (2)*, pages 449–460, 2008.

[CD09] Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 595–613. Springer, 2009.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGHR] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC*.

[GGHW13] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. Cryptology ePrint Archive, Report 2013/860, 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *CRYPTO 2014*, pages 518–535, 2014.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–, 2003.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO (2)*, pages 536–553, 2013.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GS12] Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. Cryptology ePrint Archive, Report 2012/572, 2012. http://eprint.iacr.org/.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO (1)*, pages 75–92, 2013.

[HR04] Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In *CRYPTO*, pages 92–105, 2004.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT 2014*, pages 201–220, 2014.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1998.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *TCC 2015*, pages 668–697, 2015.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *CCS'13*, pages 669–684, 2013.

[KS98] B. Kaliski and J. Staddon. PKCS #‘: RSA cryptography specifications version 2.0, 1998.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC 2014*, pages 475–484, 2014.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

A Additional Preliminaries

A.1 Succinct Non-Interactive Arguments (SNARGs and SNARKs)

We focus attention to *publicly verifiable* succinct arguments.

We consider succinct non-interactive arguments *of knowledge* (SNARKs) with adaptive soundness in Section 3.2, and consider the case of specific distributional auxiliary input.

Definition A.1 ($\mathcal{Z}$-Auxiliary Input Adaptive SNARK)**.** A triple of algorithms $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ is a *publicly verifiable, adaptively sound succinct non-interactive argument of knowledge (SNARK)* for the relation $\mathcal{R}$ if the following conditions are satisfied for security parameter k:

- **Completeness:** For any $(x, w) \in \mathcal{R}$,

$$\Pr[\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^k); \pi \leftarrow \mathsf{Prove}(x, w, \mathsf{crs}) : \mathsf{Verify}(x, \pi, \mathsf{crs}) = 1] = 1.$$

  In addition, $\mathsf{Prove}(x, w, \mathsf{crs})$ runs in time $\mathsf{poly}(k, |y|, t)$.

- **Succinctness:** The length of the proof π output by $\mathsf{Prove}(x, w, \mathsf{crs})$, as well as the running time of $\mathsf{Verify}(x, \pi, \mathsf{crs})$, is bounded by $p(k + |X|)$, where p is a universal polynomial that does not depend on $\mathcal{R}$. In addition, $\mathsf{CRSGen}(1^k)$ runs in time $\mathsf{poly}(k)$: in particular, crs is of length $\mathsf{poly}(k)$.

- **Adaptive proof of knowledge:** For any non-uniform polynomial-size prover P^* there exists a non-uniform polynomial-size extractor $\mathcal{E}_{P^*}$, such that for all sufficiently large $k \in \mathbb{N}$ and auxiliary input $z \leftarrow \mathcal{Z}$, it holds that

$$\Pr[z \leftarrow \mathcal{Z};\ \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^k);\ (x, \pi) \leftarrow P^*(z, \mathsf{crs});\ (x, w) \leftarrow \mathcal{E}_{P^*}(z, \mathsf{crs}) : \\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge w \notin R(x)] \leq \mathsf{negl}(k).$$

We also consider the following notion of zero-knowledge (ZK) succinct non-interactive arguments (SNARGs) without the extraction property, in Section B.1.

Definition A.2 (Perfect ZK-SNARG)**.** A triple of algorithms $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ is a *publicly verifiable perfect zero-knowledge (ZK) succinct non-interactive argument (SNARG)* for the relation $\mathcal{R}$ (corresponding to language L) if it satisfies the Correctness and Succinctness properties as in Definition A.1, in addition to the following properties:

- **(Non-Adaptive) Soundness:** For every PPT P^*, for every $x \notin L$, it holds that

$$\Pr[\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^k);\ \pi \leftarrow P^*(1^k, x, \mathsf{crs}) : \mathsf{Verify}(x, \pi, \mathsf{crs}) = 1] \leq \mathsf{negl}(k).$$

- **Perfect Zero Knowledge:** There exist PPT algorithms $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}})$ such that for any polynomial collection $(x_i, w_i) \in \mathcal{R}$, $i \in [\ell(k)]$, the following distributions are *identical*:

$$\{\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^k);\ \pi_1 \leftarrow \mathsf{Prove}(x_1, w_1, \mathsf{crs}); \cdots;\ \pi_\ell \leftarrow \mathsf{Prove}(x_\ell, w_\ell, \mathsf{crs}) : (\mathsf{crs}, \pi_1, \ldots, \pi_\ell)\}.$$

$$\{(\mathsf{crs}^{\mathsf{sim}}, \mathsf{td}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k);\ \pi_1^{\mathsf{sim}} \leftarrow \mathcal{S}^{\mathsf{Proof}}(x_1, \mathsf{crs}, \mathsf{td}); \cdots;\ \pi_\ell^{\mathsf{sim}} \leftarrow \mathcal{S}^{\mathsf{Proof}}(x_\ell, \mathsf{crs}, \mathsf{td}) : (\mathsf{crs}^{\mathsf{sim}}, \pi_1^{\mathsf{sim}}, \ldots, \pi_\ell^{\mathsf{sim}})\}.$$

Definition A.3 ((Perfect ZK) Universal Arguments)**.** [BG08] We say that $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ is a *(perfect zero-knowledge) universal argument* if it is a (Perfect ZK) SNARG for the universal relation R_U, defined to be the set of instance-witness pairs (y, w), where $y = (M, x, t)$, $|w| \leq t$, and M is a Turing machine, such that M accepts (x, w) after at most t steps.

Note that while the witness w for each instance $y = (M, x, t)$ in the relation R_U is of size at most t, there is no a-priori polynomial bounding t in terms of $|x|$.

A.2 Puncturable PRFs

Our result makes use of puncturable PRFs, which are PRFs with an extra capability to generate keys that allow one to evaluate the function on all bit strings of a certain length, except for any polynomial-size set of inputs. We focus on the simple case of puncturing PRFs at a single point. The definition is formulated as in [SW14], following the specific exposition of [BCPR14].

Definition A.4 (Puncturable PRFs)**.** Let m', ℓ be polynomially bounded length functions. An efficiently computable family of functions

$$\mathcal{PRF} = \left\{ \mathsf{PRF}_s : \{0,1\}^{m'(k)} \to \{0,1\}^{\ell(k)} \;\middle|\; s \in \{0,1\}^k, k \in \mathbb{N} \right\},$$

associated with an efficient (probabilistic) seed sampler $\mathcal{K}_{\mathcal{PRF}}$, is a puncturable PRF if there exists a puncturing algorithm Punct that takes as input a seed $s \in \{0,1\}^k$ and a point $x^* \in \{0,1\}^{m'(k)}$, and outputs a punctured seed s_{x^*}, so that the following conditions are satisfied:

- **Functionality is preserved under puncturing:** For every $x^* \in \{0,1\}^{m'(k)}$,

$$\Pr\left[s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k); s_{x^*} \leftarrow \mathsf{Punct}(k, x^*) : \forall x \neq x^*,\ \mathsf{PRF}_s(x) = \mathsf{PRF}_{s_{x^*}}(x)\right] = 1.$$

- **Indistinguishability at punctured points:** The following ensembles are computationally indistinguishable:

$$\{s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k), s_{x^*} \leftarrow \mathsf{Punct}(s, x^*) : x^*, s_{x^*}, \mathsf{PRF}_s(x^*)\}_{x^* \in \{0,1\}^{m(k)}, k \in \mathbb{N}}$$

$$\{s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k), s_{x^*} \leftarrow \mathsf{Punct}(s, x^*), u \leftarrow \{0,1\}^{\ell(k)} : x^*, s_{x^*}, u\}_{x^* \in \{0,1\}^{m(k)}, k \in \mathbb{N}}.$$

Note that the definition is "selectively secure," where x^* is specified before sampling the public parameters. For notational simplicity, (and without loss of generality), we will assume the punctured key s_{x^*} explicitly includes x^* in the clear.

As observed in [BW13, BGI14, KPTZ13], the GGM tree-based PRF construction [GGM86] yields puncturable PRFs as defined above, based on any one-way function. The size of such a punctured key s_{x^*} in this construction is $O(m'(k) \cdot \ell(k))$ (specifically, a punctured key at input $x^* = x_1 x_2 \cdots x_{m'(k)}$ can be attained by providing $m'(k)$ size-$\ell(k)$ partial evaluations in the GGM tree, corresponding to prefixes $(\bar{x}_1), (x_1\bar{x}_2), \ldots, (x_1 x_2 \cdots \bar{x}_{m'(k)})$.)

Theorem A.5 ([BW13, BGI14, KPTZ13])**.** *If one-way functions exist, then for all efficiently computable functions $m'(k)$ and $\ell(k)$, there exists a puncturable PRF family that maps $m'(k)$ bits to $\ell(k)$ bits, such that the size of a punctured key is $O(m'(k) \cdot \ell(k))$.*

B Applications of "Succinct Punctured Programs" Technique

We now demonstrate a variety of applications of our "succinct punctured programs" technique, building on top of public-coin $di\mathcal{O}$.

B.1 Perfect NIZK Universal Arguments

In [SW14], Sahai and Waters demonstrated a construction of a Non-Interactive Zero-Knowledge (NIZK) argument system with perfect zero knowledge from indistinguishability obfuscation, supporting fixed NP languages with statements (and witnesses) up to an a-priori bounded size. Using our succinct punctured programs technique, we achieve (using public-coin $di\mathcal{O}$) a non-interactive *universal argument system*, also perfectly zero knowledge, for languages and statements of unbounded polynomial size (see Definition A.3).

Theorem B.1. *Assume the existence of public-coin differing-inputs obfuscation for* TM *and public-coin collision-resistant hash functions. Then for any constant $\epsilon > 0$, there exists a perfect zero-knowledge universal argument system, as in Definition A.3.*

NIZK of [SW14]. We first recall the NIZK construction of [SW14]. The system consists of two obfuscated circuits (serving as the CRS):

- A Prove circuit, which has hardcoded a PRF seed s, takes as input a statement and witness pair (x, w) and outputs the PRF evaluation $\mathsf{PRF}_s(x)$ on x if w is a valid witness (i.e., $R(x, w) = 1$).
- A Verify circuit, which has hardcoded the same PRF seed s and a description of a one-way function f, takes as input a statement and alleged proof (x, π), and outputs 1 exactly if $f(\pi)$ is equal to $f(\mathsf{PRF}_s(x))$. (The introduction of the one-way function f is not needed for correctness, but rather to argue security by use of the "punctured programs" technique).

Sahai and Waters show this scheme satisfies perfect zero knowledge and (non-adaptive) soundness assuming the obfuscation scheme used is a secure indistinguishability obfuscator [SW14].

Note that here the size of the Prove and Verify circuits must grow with the size of potential statements x, thus inherently fixing an upper bound on the handled statement size at the time of CRS generation.

Succinct ZK Universal Arguments. Following our technique of succinct punctured programs, we modify the construction of [SW14] in two ways.

First, we replace the obfuscated Prove and Verify circuits (which depend on a fixed NP relation R) with obfuscated *Turing machines*, which can accept arbitrary polynomial-size relations as part of their input. That is, we now consider instances of the universal relation R_U, as described in Definition A.3. To obtain obfuscation of Turing machines *with unbounded input size*, we use (public-coin) differing-inputs obfuscation in the place of indistinguishability obfuscation. On the surface, this modification almost appears to suffice for our goal. However, one problem remains: to prove soundness of the resulting scheme for an instance $(M, x, t) \notin R_U$, we must argue that the obfuscated Turing machines are indistinguishable from obfuscations of corresponding (M, x, t)-*punctured* programs, whose size must grow with $|(M, x, t)|$. However, this requires the size of the obfuscated Turing machines to grow with $|(M, x, t)|$, thus removing our universality.

We thus incorporate a second modification to the [SW14] construction, by first computing a *collision-resistant hash* of the input statement, and proceeding with this hashed value $h(M, x, t)$. Now we need only that the obfuscated Prove and Verify programs are indistinguishable from an obfuscation of a program that is punctured at $h(M, x, t)$, whose size can now be made independent of $|(M, x, t)|$.

To execute this argument, we must take particular care that *public-coin* differing inputs obfuscation security will suffice. In particular, when replacing the obfuscated Prover and Verifier programs by their punctured versions one by one, we need that the obfuscated programs remain indistinguishable even given all the other auxiliary side information in the adversary's view, including *another obfuscated program*. At first sight, this seems an unlikely endeavor—indeed, recent works such as [BCPR14, GGHW13] have identified precisely this type of auxiliary input as problematic for security. However, we show that the *particular* obfuscated program and side information in our setting does not contain any "damaging" information. More technically, for each relevant program pair (i.e., the real Prover or Verifier program together with its associated punctured version), it is infeasible to find an input on which the programs differ directly by the security of the (public-coin) CRHF, *even given* full information on all the other tools used. This means the only "sensitive" side information is the description of the sampled CRHF function h (all other values can be simulated); since the hash family itself is a *public-coin* CRHF, public-coin $di\mathcal{O}$ will suffice. We now formalize these intuitions.

Consider the following tools:

1. PC-$di\mathcal{O}$ a public-coin differing inputs obfuscation scheme for TM (with unbounded input size).
2. $\mathcal{PRF} = \{\mathsf{PRF}_s : \{0,1\}^{m(k)} \to \{0,1\}^k\}_{s \in \{0,1\}^k}$ a puncturable PRF family.
   Note that by Theorem A.5, punctured PRFs for these parameters exist based on OWFs, which are implied by CRHF.

Turing Machine $P^{h,s}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, PRF seed $s \in \{0,1\}^k$.

Inputs: Universal statement $\vec{x} = (M, x, t)$, alleged witness w.

1. Check validity of w as witness for $\vec{x} = (M, x, t)$ by executing the Turing machine M on input (x, w) for t steps. If $M(x, w) \neq 1$, output $\perp$ and terminate. Otherwise, continue.
2. Hash the statement description: $v = h(\vec{x})$.
3. Compute the PRF on this hash: $y = \mathsf{PRF}_s(v)$.
4. Output y.

Figure 6: Turing machines $P^{h,s}$.

Punctured Turing Machine $P^{h,s}_{\vec{x},\perp}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, *punctured* PRF seed $s^* \in \{0,1\}^k$, punctured PRF input $h(\vec{x})$.

Inputs: Universal statement $\vec{x}' = (M', x', t')$, alleged witness w.

1. If $h(\vec{x}') = h(\vec{x})$ then output $\perp$ and terminate.
2. If $h(\vec{x}') \neq h(\vec{x})$ then perform the following (identical to $P^{h,s}$):
3. Check validity of w as witness for $\vec{x}' = (M', x', t')$ by executing the Turing machine M' on input (x', w) for t' steps. If $M(x', w) \neq 1$, output $\perp$ and terminate. Otherwise, continue.
4. Hash the statement description: $v = h(\vec{x})$.
5. Compute the PRF on the hash value: $y = \mathsf{PRF}_s(v)$.
6. Output y.

Figure 7: Punctured Turing machines $P^{h,s}_{\vec{x},\perp}$. Note that $P^{h,s}$ and $P^{h,s}_{\vec{x},\perp}$ differ only on inputs $(\vec{x}', w)$ for which $h(\vec{x}') = h(\vec{x})$ for the hardcoded value $h(\vec{x})$.

3. $\mathcal{H} = \{\mathcal{H}_k\}$ a CRHF family with $h : \{0,1\}^* \to \{0,1\}^{m(k)}$ for each $h \in \mathcal{H}_k$.
4. $\mathcal{F} = \{f : \{0,1\}^k \to \{0,1\}^{k^\epsilon}\}$ a OWF family.

We now present our construction.

Succinct ZK Universal Argument Construction ($\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify}$):

$\mathsf{CRSGen}(1^k)$: on input the security parameter, the CRS generation procedure samples a PRF seed $s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k)$, a hash function $h \leftarrow \mathcal{H}_k$, and a OWF $f \leftarrow \mathcal{F}_k$. It then generates obfuscations of the corresponding Turing machines $P^{h,s}$ and $V^{h,s,f}$, as defined in Figures 6 and 8. That is, $\tilde{P} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, P^{h,s})$ and $\tilde{V} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, V^{h,s,f})$. Output the pair of obfuscated programs $\mathsf{crs} = (\tilde{P}, \tilde{V})$.

$\mathsf{Prove}(\mathsf{crs}, \vec{x}, w)$: Evaluate the obfuscated program $\tilde{P} \in \mathsf{crs}$ on input $(\vec{x}, w)$: i.e., output $\tilde{P}(\vec{x}, w)$.

$\mathsf{Verify}(\mathsf{crs}, \vec{x}, \pi)$: Evaluate the obfuscated program $\tilde{V} \in \mathsf{crs}$ on input $(\vec{x}, \pi)$: i.e., output $\tilde{V}(\vec{x}, \pi)$.

Proof of Theorem B.1. Perfect zero knowledge holds as the distribution of proofs can be perfectly simulated given the PRF seed s (without knowledge of any witness).

The size and time complexities follow from a straightforward analysis of the programs $P^{h,s}, V^{h,s,f}$. In particular, for any input (M, x, t), the proof size is k^ϵ bits (independent of $|M|, t$) corresponding to the output of the OWF, and verification of a proof on (M, x, t) requires only computing a hash and PRF, and not executing M.

Turing Machine $V^{h,s,f}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, PRF seed $s \in \{0,1\}^k$, function $f \in \mathcal{F}$ from the OWF family.

Inputs: Universal statement $\vec{x} = (M, x, t)$, alleged proof π.

1. Compute the "correct" proof for $\vec{x}$: i.e., $\pi' = \mathsf{PRF}_s(h(\vec{x}))$.
2. Verify whether $f(\pi) = f(\pi')$. If so, output 1; otherwise output 0.

Figure 8: Turing machines $V^{h,s,f}$.

Punctured Turing Machine $V^{h,s,f}_{\vec{x},y}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, *punctured* PRF seed $s^* \in \{0,1\}^k$, punctured PRF input $h(\vec{x})$, punctured output y, OWF $f \in \mathcal{F}$.

Inputs: Universal statement $\vec{x}' = (M', x', t')$, alleged proof π.

1. If $h(\vec{x}') \neq h(\vec{x})$, then compute the "correct" proof for x' as $\pi' = \mathsf{PRF}_s(h(\vec{x}'))$, and output 1 if and only if $f(\pi) = f(\pi')$.
2. If $h(\vec{x}') = h(\vec{x})$, then output 1 if and only if $f(\pi) = y$, where y is hardcoded.

Figure 9: Punctured Turing machines $V^{h,s,f}_{\vec{x},y}$.

(Non-adaptive) soundness follows an analogous sequence of hybrids as Theorem 3.1, mirroring the approach of [SW14]. Namely, for $\vec{x} = (M, x, t) \notin R_U$, the obfuscated programs $P^{h,s}, V^{h,s,f}$ are sequentially replaced with their "$\vec{x}$-punctured" counterparts $P^{h,s}_{\vec{x},\perp}$ and $V^{h,s,f}_{\vec{x},y}$ (as defined in Figures 7 and 9), and then replacing the hardcoded "correct" comparison value $y = f(\mathsf{PRF}_s(h(\vec{x})))$ in the $V^{h,s,f}_{\vec{x},y}$ program with a random output $f(u)$ of the OWF f for uniform $u \in \{0,1\}^k$.

Hybrid 0: The "real-world" setting. Namely, the CRS is generated via the (honest) process CRSGen described above.

Hybrid 1: The CRS generation is replaced by the following procedure CRSGen_1. The algorithm CRSGen_1 samples a PRF seed $s \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^k)$, hash function $h \leftarrow \mathcal{H}_k$, and OWF $f \leftarrow \mathcal{F}_k$ as before (in CRSGen). But, from the seed s it generates an $h(x)$-punctured seed $s^* \leftarrow \mathsf{Punct}(s, h(x))$, and then obfuscates the corresponding *punctured* program $\tilde{P} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, P^{h,s}_{\vec{x},\perp})$. Note that as $R(x,w) \neq 1$ for all possible witnesses (as $x \notin L$), the punctured output is inherently $\perp$. CRSGen_1 generates the second obfuscated program $\tilde{V}$ precisely as in CRSGen, and outputs the pair $(\tilde{P}, \tilde{V})$.

Hybrid 2: The CRS generation is replaced by the following procedure CRSGen_2. The algorithm CRSGen_2 generates $\tilde{P}$ precisely as CRSGen_1 described above (i.e., obfuscating the punctured prover program). However, the obfuscation $\tilde{V}$ of the program $V^{h,s,f}$ is replaced by an obfuscation of a corresponding *punctured* program $V^{h,s,f}_{\vec{x},y}$, with punctured input $h(\vec{x})$ and "correct" comparison value $y := f(\mathsf{PRF}_s(h(\vec{x})))$ hardcoded. That is, $\tilde{V} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, V^{h,s,f}_{\vec{x},y})$.

Hybrid 3: The CRS generation is replaced by the following procedure CRSGen_3, which proceeds identically to CRSGen_2, except that the program $\tilde{V}$ is generated as an obfuscation of the program $V^{h,s,f}_{\vec{x},y}$ for $y := f(u)$ for *uniform* sampled u.

For each $i \in \{0,1,2,3\}$ and $\vec{x} = (M, x, t) \notin R_U$, we denote by $\mathsf{FalseProof}_i(\mathcal{A}, 1^k, \vec{x})$ the probability that adversary $\mathcal{A}$ succeeds in generating a false proof π for $\vec{x}$ given crs as generated as in Hybrid i:

$$\mathsf{FalseProof}_i(\mathcal{A}, 1^k, \vec{x}) := \Pr\left[\mathsf{crs} \leftarrow \mathsf{CRSGen}_i(1^k); \pi \leftarrow \mathcal{A}(1^k, \vec{x}, \mathsf{crs}) : \mathsf{Verify}(x, \pi, \mathsf{crs}) = 1\right].$$

Prover-Replacement: Program Pair Sampler $\mathsf{Samp}_P(1^k, \vec{x}, r)$:

1. Sample a hash function $h = \mathcal{H}_k(r_h)$.
2. Sample a PRF seed $s = K_{\mathsf{PRF}}(1^k; r_s)$.
3. Sample a punctured PRF seed $s^* = \mathsf{Punct}(s, h(\vec{x}); r_*)$.
4. Denote $r := (r_h, r_s, r_*)$.
5. Output program pair $(P^{h,s}, P^{h,s}_{\vec{x},\perp})$, defined by $\vec{x}, h, s, s^*$ as above (and padded to equal length and per-input runtime). See Figures 6,7.

Figure 10: Program pair sampler algorithm used in public-coin differing inputs security step, replacing the obfuscated Prover program with the *punctured* version.

Verifier-Repacement: Program Pair Sampler $\mathsf{Samp}_V(1^k, \vec{x}, f, r)$:

1. Sample a hash function $h = \mathcal{H}_k(r_h)$.
2. Sample a PRF seed $s = K_{\mathsf{PRF}}(1^k; r_s)$.
3. Sample a punctured PRF seed $s^* = \mathsf{Punct}(s, h(\vec{x}); r_*)$.
4. Let $y = f(\mathsf{PRF}_s(h(\vec{x})))$.
5. Denote $r := (r_h, r_s, r_*)$.
6. Output program pair $(V^{h,s,f}, V^{h,s,f}_{\vec{x},y})$, defined by $\vec{x}, h, s, s^*, y$ sampled above (and padded to equal length and per-input runtime). See Figures 8,9.

Figure 11: Program pair sampler algorithm, to be used in second public-coin differing inputs security step (replacing the obfuscated Verifier program with the punctured version).

The proof provides by showing that $\mathsf{FalseProof}_i(\mathcal{A}, 1^k, \vec{x})$ cannot grow by more than a negligible amount as i from 0 to 3, and then finally that $\mathsf{FalseProof}_3(\mathcal{A}, 1^k, \vec{x})$ *must* be negligible. For the present work, we focus only on the first two steps, which rely on the security of the public-coin differing-inputs obfuscation. The proofs of the two remaining steps (relying on the security of the punctured PRF and the OWF) are identical to those in [SW14].

We begin by considering the step from Hybrid 0 to Hybrid 1. In Figure 10 we define the program sampler algorithm to be used for the PC-$di\mathcal{O}$ argument. Note that the programs output by Samp_P correspond to the "standard" and "punctured" versions of the Prove procedure.

Claim B.2. *The "prover-replacement" sampler* Samp_P *(Figure 10) is a public-coin differing-inputs sampler. That is, for all efficient non-uniform* $\mathcal{A}_{\mathsf{PC}}$*, there exists a negligible function* ϵ *such that for all* $k \in \mathbb{N}$*,*

$$\Pr\big[r \leftarrow \{0,1\}^*; (M_0, M_1) \leftarrow \mathsf{Samp}_P(1^k, r); (x, 1^t) \leftarrow \mathcal{A}_{\mathsf{PC}}(1^k, r) : \\ M_0(x) \neq M_1(x) \wedge \mathsf{steps}(M_0, x) = \mathsf{steps}(M_1, x) = t\big] \leq \epsilon(k). \quad (6)$$

Proof. This claim will hold by the public-coin collision resistance of the CRHF. Namely, suppose to the contrary, there exists an efficient (non-uniform) adversary $\mathcal{A}_{\mathsf{PC}}$ and non-negligible function $\alpha(k)$ for which the probability in Equation 6 is greater than $\alpha(k)$. We show such an adversary contradicts the security of the (public-coin) CRHF. Consider an adversary $\mathcal{A}_{\mathsf{CR}}$ in the CRHF security challenge. Namely, for a challenge hash function $h \leftarrow \mathcal{H}_k(r_h)$, the adversary $\mathcal{A}_{\mathsf{CR}}$ receives h, r_h, and performs the following steps:

CRHF adversary $\mathcal{A}_{\mathsf{CR}}(1^k, h, r_h)$:

1. Imitate the remaining steps of Samp_P. That is, sample a PRF seed $s = K_{\mathsf{PRF}}(1^k; r_s)$ and a punctured PRF seed $s^* = \mathsf{Punct}(s, h(\vec{x}); r_*)$. Define $r = (r_h, r_s, r_*)$, and let $P_0 = P^{h,s}$ and $P_1 = P^{h,s}_{\vec{x},\perp}$.
2. Run $\mathcal{A}_{\mathsf{PC}}(1^k, r)$ on the collection of randomness r used above. In response, $\mathcal{A}_{\mathsf{PC}}$ returns a pair $((\vec{x}', w), 1^t)$.
3. $\mathcal{A}_{\mathsf{CR}}$ outputs the pair $(\vec{x}, \vec{x}')$ as an alleged collision in the challenge hash function h.

With probability $\alpha(k)$, the value $(\vec{x}', w)$ generated by $\mathcal{A}_{\mathsf{PC}}$ satisfies that $P_0(\vec{x}', w) \neq P_1(\vec{x}', w)$. From the definition of P_0, P_1 (i.e., $P^{h,s}, P^{h,s}_{\vec{x},\perp}$), this must mean that $h(\vec{x}) = h(\vec{x}')$, since the programs operate identically on all other inputs. Further, it must be that $\vec{x} \neq \vec{x}'$, since $P^{h,s}(\vec{x}, w) = \perp$ for any value of w (i.e., there is no valid witness since $x \notin L$) which agrees with $P^{h,s}_{\vec{x},\perp}(\vec{x}', w)$ since $h(\vec{x}') = h(\vec{x})$ (meaning its output is hardcoded to $\perp$). That is, $\mathcal{A}_{\mathsf{CR}}$ successfully identifies a collision with the same probability $\alpha(k)$, which must thus be negligible.

$\square$

Claim B.3. *For every (non-uniform) PPT* $\mathcal{A}$*, there exists a negligible function* ν *such that for every statement* $\vec{x} = (M, x, t) \notin R_U$*, it holds that* $\mathsf{FalseProof}_0(\mathcal{A}, 1^k, \vec{x}) \leq \mathsf{FalseProof}_1(\mathcal{A}, 1^k, \vec{x}) + \nu(k)$*.*

Proof. (The claim will hold by the security of the public-coin differing-inputs obfuscator). Suppose to the contrary there exists a (non-uniform) adversary $\mathcal{A}_{01}$, OWF f, and input $\vec{x}$ for which the claim does not hold. From this, we construct an adversary $\mathcal{A}_{di\mathcal{O}}$ that breaks the security of the obfuscation with respect to Samp_P (which was shown in the previous claim to be a public-coin $di\mathcal{O}$ sampler). Recall that in the $PCdi\mathcal{O}$ challenge, a pair of programs is sampled as $(P_0, P_1) \leftarrow \mathsf{Samp}(r)$, a random choice is obfuscated as $\tilde{P}$, and the adversary is given $\tilde{P}$ in addition to the sampling randomness r.

PC-$di\mathcal{O}$ adversary $\mathcal{A}_{di\mathcal{O}}(1^k, \tilde{P}, r)$:
    Hardcoded: $\vec{x}, f$.

1. Parse the given randomness r (used in Samp) as $r = (r_h, r_s, r_*)$ (see Figure 10).
2. Recompute $h = \mathcal{H}_k(r_h)$, $s = K_{\mathsf{PRF}}(1^; r_s)$.
3. Recompute $s^* = \mathsf{Punct}(s, h(\vec{x}; r_*)$.
4. Simulate the CRS generation procedure (either CRSGen or CRSGen_1) using this randomness and $\tilde{P}$.
5. Compute $y = f(\mathsf{PRF}_s(h(\vec{x})))$.
6. Let $V^{h,s,f}_{\vec{x},y}$ be the program as in Figure 9 for the above values h, s^*, f, y.
7. Obfuscate $\tilde{V} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, V^{h,s,f}_{\vec{x},y})$.
8. Let $\mathsf{crs} := (\tilde{P}, \tilde{P})$.
9. Run the adversary $\mathcal{A}_{01}(1^k, \vec{x}, \mathsf{crs})$, and receive an alleged false proof π'.
10. If $\mathsf{Verify}(\mathsf{crs}, \vec{x}, \pi) = 1$ (i.e., if $\mathcal{A}_{01}$ succeeded in generating a false proof) then $\mathcal{A}_{di\mathcal{O}}$ outputs 1. Otherwise, $\mathcal{A}_{di\mathcal{O}}$ outputs 0.
    If $\tilde{P}$ is an obfuscation of $P^{h,s}$, then $\mathcal{A}_{di\mathcal{O}}$ outputs 1 with probability $\mathsf{FalseProof}_0(\mathcal{A}, 1^k, \vec{x})$.
    If $\tilde{P}$ is an obfuscation of $P^{h,s}_{\vec{x},\perp}$, then $\mathcal{A}_{di\mathcal{O}}$ outputs 1 with probability $\mathsf{FalseProof}_1(\mathcal{A}, 1^k, \vec{x})$.
    The claim follows.

□

We now treat the step from Hybrid 1 to 2, following a similar argument.

Claim B.4. *The "verifier-replacement" sampler* Samp_V *(Figure 11) is a public-coin differing-inputs sampler. That is, for all efficient non-uniform* $\mathcal{A}_{\mathsf{PC}}$*, there exists a negligible function* ϵ *such that for all* $k \in \mathbb{N}$*,*

$$\Pr\big[r \leftarrow \{0,1\}^*; (M_0, M_1) \leftarrow \mathsf{Samp}_V(1^k, r); (x, 1^t) \leftarrow \mathcal{A}_{\mathsf{PC}}(1^k, r) : \\ M_0(x) \neq M_1(x) \wedge \mathsf{steps}(M_0, x) = \mathsf{steps}(M_1, x) = t\big] \leq \epsilon(k). \quad (7)$$

Proof. This claim will again hold by the public-coin collision resistance of the CRHF. Namely, suppose to the contrary, there exists an efficient (non-uniform) adversary $\mathcal{A}_{\mathsf{PC}}$ and non-negligible function $\alpha(k)$ for which the probability in Equation 6 is greater than $\alpha(k)$. We show such an adversary contradicts the security of the (public-coin) CRHF. Consider an adversary $\mathcal{A}_{\mathsf{CR}}$ in the CRHF security challenge. Namely, for a challenge hash function $h \leftarrow \mathcal{H}_k(r_h)$, the adversary $\mathcal{A}_{\mathsf{CR}}$ receives h, r_h, and performs the following steps:

CRHF adversary $\mathcal{A}_{\mathsf{CR}}(1^k, h, r_h)$:

1. Imitate the remaining steps of Samp_V. That is, sample a PRF seed $s = K_{\mathsf{PRF}}(1^k; r_s)$ and a punctured PRF seed $s^* = \mathsf{Punct}(s, h(\vec{x}); r_*)$. Define $y = f(\mathsf{PRF}_s(h(\vec{x})))$ and $r = (r_h, r_s, r_*)$, and let $V_0 = V^{h,s,f}$ and $V_1 = V^{h,s,f}_{\vec{x},y}$.
2. Run $\mathcal{A}_{\mathsf{PC}}(1^k, r)$ on the collection of randomness r used above. In response, $\mathcal{A}_{\mathsf{PC}}$ returns a pair $((\vec{x}', \pi), 1^t)$.
3. $\mathcal{A}_{\mathsf{CR}}$ outputs the pair $(\vec{x}, \vec{x}')$ as an alleged collision in the challenge hash function h.

With probability $\alpha(k)$, the value $(\vec{x}', \pi)$ generated by $\mathcal{A}_{\mathsf{PC}}$ satisfies that $P_0(\vec{x}', \pi) \neq P_1(\vec{x}', \pi)$. From the definition of V_0, V_1 (i.e., $V^{h,s,f}, V^{h,s,f}_{\vec{x},y}$), this must mean that $h(\vec{x}) = h(\vec{x}')$, since the programs operate identically on all other inputs. Further, it must be that $\vec{x} \neq \vec{x}'$, since the hardcoded comparison value y in $V^{h,s,f}_{\vec{x},y}$ was chosen as the "correct" comparison value for $\vec{x}$, i.e. $y = f(\mathsf{PRF}_s(h(\vec{x})))$. Therefore, $\mathcal{A}_{\mathsf{CR}}$ successfully identifies a collision in h with the same probability $\alpha(k)$, which must thus be negligible.

□

Claim B.5. *For every (non-uniform) PPT* $\mathcal{A}$*, there exists a negligible function* ν *such that for every statement* $\vec{x} = (M, x, t) \notin R_U$*, it holds that* $\mathsf{FalseProof}_2(\mathcal{A}, 1^k, \vec{x}) \leq \mathsf{FalseProof}_1(\mathcal{A}, 1^k, \vec{x}) + \nu(k)$*.*

Finally, we prove that $\mathsf{FalseProof}_2(\mathcal{A}, 1^k, \vec{x})$ must be negligible.

Claim B.6. *For every (non-uniform) PPT $\mathcal{A}$, there exists a negligible function ν such that for every statement $\vec{x} = (M, x, t) \notin R_U$, it holds that $\mathsf{FalseProof}_2(\mathcal{A}, 1^k, \vec{x}) \leq \nu(k)$.*

Proof. This proof is essentially identical to that of the previous step. Namely, given the randomness $r = (r_h, r_s, r_*)$ used to sample the program pair $(V^{h,s,f}, V^{h,s,f}_{\vec{x},y}) \leftarrow \mathsf{Samp}_V(r)$, the reduction can generate an obfuscation of the corresponding Prove program $P^{h,s}_{\vec{x},\perp}$ with the correct distribution, thus simulating either CRSGen_1 or CRSGen_2 based on his PC-$di\mathcal{O}$ obfuscation challenge. $\square$

$\square$

B.2 *Universal* Instantiation of Full-Domain Hash

The full-domain hash (FDH) signature paradigm, first proposed by Bellare and Rogaway [BR93, BR96], provides a means of building a signature scheme from any trapdoor permutation, within the heuristic random oracle model. Specifically, a signature on a message m is generated by evaluating the random oracle at input m, and then computing the inverse of the trapdoor permutation on this value $\mathsf{RO}(m)$. This work has been very influential and formed the foundation for part of the PKCS#1 standard [KS98]. However, negative results in later years called into question the rigorous implications of security proofs in the random oracle model [CGH04, GK03, BBP04], showing e.g. that for some such applications *no* concrete instantiation of the random oracle can yield security.

In a recent work, Hohenberger, Sahai, and Waters [HSW14] presented a methodology for instantiating the random oracle that provides (selective) security for full-domain hash signatures in the standard model, building on recent advances in indistinguishability obfuscation (in particular, using the punctured programs paradigm of [SW14]). They demonstrate for every trapdoor permutation f that there exists a hash function $\mathcal{R}_f$ (tailored to f) such that the FDH signature scheme is (selectively) secure in the standard model when using f and instantiating the random oracle by $\mathcal{R}_f$.

We show that our succinct punctured programs technique yields a *universal* instantiation of the random oracle providing security for full-domain hash signatures. That is, we provide a single family of Turing machines $\mathcal{R} = \{\mathcal{R}_k\}$ such that, for *any* injective trapdoor function f, the Bellare-Rogaway Full-Domain Hash signature scheme [BR93, BR96] using f and instantiating the random oracle by $\mathcal{R}$ is selectively secure in the standard model. This construction involves a tweak to the FDH signature structure, in which the random oracle takes as input a description of the trapdoor permutation f in addition to the message to be signed: i.e., $\mathsf{Sign}(m) = f^{-1}(\mathsf{RO}(m, f))$.

Our construction. Intuitively, to instantiate the random oracle, we would like for each message m to be able to sample a random image of a given trapdoor function f, without revealing information about the corresponding preimage. In [HSW14], this is done by providing an (indistinguishability) obfuscation of a circuit that computes a PRF on the input m and then evaluates f on this outcome. Using the technique of punctured programs [SW14], Hohenberger *et al.* [HSW14] show that given this obfuscated circuit, no information is revealed about the evaluation of the PRF on the (pre-selected) forgery challenge message m, so that forging on message m implies inverting a random(-looking) output $f(\mathsf{PRF}_s(m))$ of the trapdoor function, contradicting its assumed security. In this construction, however, the obfuscated circuit (i.e., the instantiation of the random oracle) inherently depends on the trapdoor function f.

We avoid this dependency by obfuscating a *Turing machine* that takes as input a message m and description of the desired trapdoor function f (described as a poly-size circuit), and outputs a seemingly random evaluation of f. As in our constructions from the previous sections, in order to allow the size of the obfuscated program to be independent of the size of the input trapdoor function (while still maintaining security), we first *hash* the input (m, f), and then proceed with this hashed value.

Program $\Pi^{h,s}$:

Hardwired: Public-coin collision-resistant hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, PRF seed $s \in \{0,1\}^k$.

Inputs: Message m, circuit description f

1. Hash the input: $v = h(m, f)$.
2. Compute the PRF on this hash: $x = \mathsf{PRF}_s(v)$.
3. Evaluate the universal Turing machine on inputs f, x: i.e., $y = U_k(f, x)$.
4. Output y.

Figure 12: Program $\Pi^{h,s}$ that is obfuscated to form $R^{h,s} \in \mathcal{R}_k$.

Program $\Pi^{h,s}_{m,y}$:

Hardwired: Hash function $h : \{0,1\}^* \to \{0,1\}^{m(k)}$, punctured PRF seed $s^* \in \{0,1\}^k$ (punctured at point $h(m, f)$), bit string $y \in \{0,1\}^{\ell'(k)}$.

Inputs: Message m', circuit description f'

1. Hash the input: $v' = h(m', f')$.
2. If $v' \neq h(m, f)$, compute $x = \mathsf{PRF}_{s^*}(h)$, and output $U_k(f', x)$.
3. If $v' = h(m, f)$, output y.

Figure 13: "Punctured" program $\Pi^{h,s}_{m,y}$, used within the security proof.

More explicitly, each program $R^{h,s} \in \mathcal{R}_k$ in our random oracle instantiation is a (public-coin differing-inputs) obfuscation of a program $\Pi^{h,s}$, indexed by a PRF seed s and public-coin collision-resistant hash function h. $\Pi^{h,s}$ accepts as input a message m, and a description of the trapdoor function f (modeled as a polynomial-size circuit), and it functions by: (1) computing the hash of (m, f) with respect to h (using a Merkle-Damgard hash tree approach), (2) applying a PRF with hardcoded seed s to this hash value, and then (3) evaluating f on the resulting PRF output value x. That is,

$$\Pi^{h,s}(m, f) = U_k\left(f, \mathsf{PRF}_s(h(m, f))\right),$$

where U_k is the universal Turing machine, and

$$R^{h,s} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, \Pi^{h,s}).$$

(See Figure 12 for details). Denote by $\mathcal{M}$ the class of Turing machines

$$\mathcal{M} = \left\{ \Pi^{h,s} \;\middle|\; s \in \{0,1\}^k, h \in \mathcal{H}_k, k \in \mathbb{N} \right\}.$$

To prove security of the resulting FDH signature scheme, we consider a second, related class of ("punctured") Turing machines

$$\mathcal{M}^* = \left\{ \Pi^{h,s}_{m,y} \;\middle|\; s \in \{0,1\}^k, h \in \mathcal{H}_k, k \in \mathbb{N} \right\},$$

where each Turing machine $\Pi^{h,s}_{m,y}$ is defined as in Figure 13. The obfuscator we use is with respect to the class of Turing machines $\mathcal{M} \cup \mathcal{M}^*$.

Now, consider the instantiation of the full-domain hash signature scheme using a hash function $R^{h,s} \in \mathcal{R}_k$ in the place of a random oracle, and with an injective trapdoor function family $\mathcal{F}$.

$\mathsf{Setup}(1^k)$: On input the security parameter 1^k, the setup algorithm samples a program $R^{h,s} \leftarrow \mathcal{R}_k$ to be used as a random oracle. That is, it samples a random seed for a puncturable PRF $s \leftarrow \mathcal{F}_{\mathcal{PRF}}(1^k)$, and an underlying hash function $h \leftarrow \mathcal{H}_k$ from the public-coin collision-resistant hash function family. This uniquely defines a program $\Pi^{h,s}$. Then, the algorithm obfuscates this program as $R^{h,s} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, \Pi^{h,s})$.

The setup algorithm then samples $(f, f^{-1}) \leftarrow \mathsf{TDFSetup}(1^k)$ that produces a public index f and trapdoor f^{-1} (that allows inversion)

The verification key for the signature scheme is set to $\mathsf{vk} = (f, R^{h,s})$, consisting of the TDF description f and the "random oracle" $R^{h,s}$. The secret key sk is the trapdoor f^{-1} and $R^{h,s}$.

$\mathsf{Sign}(\mathsf{sk}, m)$: The signature algorithm outputs $\sigma = f^{-1}(R^{h,s}(m, f))$.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: The verification algorithm tests if $f(\sigma) \overset{?}{=} (R^{h,s}(m, f))$ and outputs accept if and only if this holds.

Theorem B.7. *Assuming the existence of public-coin collision-resistant hash functions, and public-coin differing-inputs obfuscation for Turing machines, then for* any *injective trapdoor function family $\mathcal{F}$, the scheme described above is a selectively secure signature scheme.*

Proof. The proof follows an analogous sequence of hybrids as in Theorem 3.1, mirroring the approach of [HSW14]. Namely, for challenge forgery message m and injective trapdoor function f, the hybrids are (loosely) as follows:

Hybrid 0: The real (selective) security experiment.

Hybrid 1: The obfuscated program $R^{h,s} \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, \Pi^{h,s})$ is replaced by an obfuscation of the corresponding "(m, f)-punctured" program $\Pi^{h,s}_{m,y}$ with the "correct" hardcoded output $y = \Pi^{h,s}(m, f)$. Indistinguishability follows by the security of the public-coin differing-inputs obfuscator, together with the (public-coin) collision resistance of h. (Note that in the reduction, signature queries can be simulated using the inversion trapdoor to f).

Hybrid 2: The obfuscated program is replaced by an obfuscation of $\Pi^{h,s}_{m,y}$, with hardcoded output y set to a *random* evaluation of f: i.e., $f(u)$ for uniform u. Indistinguishability follows by the pseudo randomness of PRF. (Note that in the reduction, signature queries can again be simulated using the inversion trapdoor to f).

By the indistinguishability of hybrids, it follows that a forging adversary must continue to successfully forge in this final experiment, Hybrid 2. But, this implies the adversary can invert a random output of the trapdoor function, yielding the desired contradiction.

□

C Removing Auxiliary Input in $di\mathcal{O}$

We now demonstrate that *even in the absence of auxiliary input*, care must be taken when making use of differing-inputs obfuscation ($di\mathcal{O}$). Specifically, we observe a general transformation obtaining $di\mathcal{O}$ with respect to any distribution $\mathcal{D}$ over $\mathcal{P} \times \mathcal{P} \times \{0,1\}^{p(k)}$ of program pairs and polynomial bounded-length auxiliary input from any $\mathcal{O}$ that satisfies a weaker notion of *indistinguishability* obfuscation ($i\mathcal{O}$) security (which is not an extractability assumption; see Section C.1 below), and is $di\mathcal{O}$-secure with respect to distribution $\mathcal{D}'$ over slightly enriched program pairs $\mathcal{P}' \times \mathcal{P}'$, with *no* auxiliary input.[12]

Intuitively, we demonstrate that "bad side information" cannot be avoided by disallowing auxiliary input, but rather can appear embedded within the *input-output behavior* of the programs to be obfuscated. Since any candidate obfuscator $\mathcal{O}$ must preserve the input-output functionality of the underlying programs, this "bad" information is unavoidably released.

[12]Note that standard $i\mathcal{O}$ security (for $P/poly$) is not implied by $di\mathcal{O}$ for a *specific distribution.*

Theorem C.1 (Removing Auxiliary Input in $di\mathcal{O}$)**.** *For any polynomial p and distribution $\mathcal{D}$ over $\mathcal{P} \times \mathcal{P} \times \{0,1\}^{p(k)}$, there exists a polynomial q, a class of programs $\mathcal{P}'$ with $\max_{P' \in \mathcal{P}'} |P'| \leq q(\max_{P \in \mathcal{P}} |P|)$, and a distribution $\mathcal{D}'$ over $\mathcal{P}' \times \mathcal{P}'$ for which the following holds. The existence of $di\mathcal{O}$ w.r.t. $\mathcal{D}$ (with auxiliary input) is implied by the existence of obfuscation that is $i\mathcal{O}$ secure and $di\mathcal{O}$ secure w.r.t. $\mathcal{D}'$ (without auxiliary input).*

In the following two subsections, we present a formal definition of $i\mathcal{O}$, and then present our transformation and proof for Theorem C.1.

C.1 Indistinguishability Obfuscation

We begin by defining the notion of indistinguishability obfuscation ($i\mathcal{O}$), which we rely on as a tool in our transformation. Note that in general, $i\mathcal{O}$ is weaker than (and implied by) $di\mathcal{O}$. However, in our setting, we consider general-purpose $i\mathcal{O}$, versus $di\mathcal{O}$ that is secure with respect to a particular distribution. In such case, the two notions are incomparable.

Definition C.2 (Indistinguishability Obfuscation)**.** A uniform PPT machine $i\mathcal{O}$ is an *indistinguishability obfuscator* ($i\mathcal{O}$) (w.r.t. worst-case inputs) for a class of Turing machines $\{\mathcal{M}_k\}_{k \in \mathbb{N}}$ if the following conditions are satisfied:

- **Correctness:** There exists a negligible function $\mathsf{negl}(k)$ such that for every security parameter $k \in \mathbb{N}$, for all $M \in \mathcal{M}_k$, for all inputs x, we have

$$\Pr[M' \leftarrow \mathsf{PC}\text{-}di\mathcal{O}(1^k, M) : M'(x) = M(x)] = 1 - \mathsf{negl}(k).$$

- **Security:** For every non-uniform PPT adversary $\mathcal{A}$ and polynomial $p(k)$, for every $k \in \mathbb{N}$, every pair of Turing machines $M_0, M_1 \in \mathcal{M}_k$,

$$\Pr\left[b \leftarrow \{0,1\}; \tilde{M} \leftarrow i\mathcal{O}(1^k, M_b) : \mathcal{A}(1^k, \tilde{M}) = b\right] \geq \frac{1}{2} + \frac{1}{p(k)}$$
$$\implies \quad \exists x \text{ s.t. } M_0(x) \neq M_1(x).$$

Remark C.3 ($i\mathcal{O}$ and Auxiliary Input)**.** In contrast to $di\mathcal{O}$ obfuscation, indistinguishability obfuscation directly implies security against arbitrary auxiliary input. Indeed, for any non-uniform adversary $\mathcal{A}$ and auxiliary input z, we can simply consider a modified non-uniform adversary $\mathcal{A}_z$ who has z, and the same conclusion holds. (Recall the issue in the more demanding $di\mathcal{O}$ setting is that one must demonstrate an *extractor* algorithm $\mathcal{E}$ for the original adversary $\mathcal{A}$ such that the same extractor works for any possible auxiliary input z).

C.2 $di\mathcal{O}$ with Auxiliary Input from $i\mathcal{O}$ + $di\mathcal{O}$ without Axuiliary Input

We now present our "auxiliary input removal" transformation, constituting the proof of Theorem C.1. Formally, for class of programs $\mathcal{P}$, and distribution $\mathcal{D}$ over $\mathcal{P} \times \mathcal{P} \times \{0,1\}^{p(k)}$, we define generic program modifiers, and then corresponding "enriched" variants $\mathcal{P}', \mathcal{D}'$ below. (Note that the p-notation will mostly be suppressed).

Definition C.4 (Program Modifiers)**.** For program class $\mathcal{P}$, define the following program "wrappers":

- $\mathsf{Expand}(P, z)$: For $P \in \mathcal{P}, z \in \{0,1\}^{p(k)}$, the program $\mathsf{Expand}(P, z)$ takes as input a pair (x, mode) for x a valid input to P and $\mathsf{mode} \in \{\emptyset, *\}$ (corresponding to "standard" and "special" modes), and embeds z into the program input-output as follows:
  1. If $\mathsf{mode} = \emptyset$: Output $P(x)$.
  2. If $\mathsf{mode} = *$: Output z.

  We will sometimes use shorthand notation and denote by P^z the program $\mathsf{Expand}(P, z)$

- $\mathsf{ztoZero}(P^z)$: For P^z for some $P \in \mathcal{P}, z \in \{0,1\}^{p(k)}$, the program $\mathsf{ztoZero}(P^z)$ takes as input a *pair* (x, mode) but "zeroes" out the special embedded output of P^z, as follows:
  1. If $\mathsf{mode} = \emptyset$: Output $P^z(x, \mathsf{mode})$.
  2. If $\mathsf{mode} = *$: Output 0.
- $\mathsf{pad}(P, \mathsf{size})$: For any program P (or P^z) with $|P| \leq \mathsf{size}$, the program $\mathsf{pad}(P, \mathsf{size})$ has identical input-output behavior to P, but satisfies $|\mathsf{pad}(P, \mathsf{size})| = \mathsf{size}$.
- $\mathsf{Restrict}(P^z, \mathsf{mode}^*)$: For P^z for some $P \in \mathcal{P}, z \in \{0,1\}^{p(k)}$, and $\mathsf{mode}^* \in \{\emptyset, *\}$, the program $\mathsf{Restrict}(P^z, \mathsf{mode}^*)$ takes as input a *single* value x, and outputs $P^z(x, \mathsf{mode}^*)$, with mode^* hardcoded.

Definition C.5 (Enriched Program Classes)**.** Let $\mathcal{P} = \{P\}$ be a class of programs, and $\mathcal{D}$ be a distribution over $\mathcal{P} \times \mathcal{P} \times \{0,1\}^{p(k)}$ for polynomial p. We define the (p-)enriched variants $\mathcal{P}', \mathcal{D}'$ as:

- $\mathcal{P}' := \left\{\mathsf{Expand}(P, z) : P \in \mathcal{P}, z \in \{0,1\}^{p(k)}\right\}$, where $\mathsf{Expand}(P, z)$ is as above.
- $\mathcal{D}'$ is sampled over $\mathcal{P}' \times \mathcal{P}'$ as follows:
  1. Sample $(P_0, P_1, z) \leftarrow \mathcal{D}$.
  2. Output the pair $(\mathsf{Expand}(P_0, z), \mathsf{Expand}(P_1, z)) \in \mathcal{P}' \times \mathcal{P}'$.

Proof of Theorem C.1. Suppose $\mathcal{O}'$ is a program obfuscation algorithm such that: (1) $\mathcal{O}'$ is $i\mathcal{O}$-secure, as in Definition C.2, and (2) $\mathcal{O}'$ is $di\mathcal{O}$-secure w.r.t. program distribution $\mathcal{D}'$ over $\mathcal{P}' \times \mathcal{P}'$ (with *no* auxiliary input), as in Definition 2.3. We define a new obfuscation algorithm $\mathcal{O}$ and show that it is a differing-inputs obfuscator secure w.r.t. $\mathcal{D}$ over $\mathcal{P} \times \mathcal{P} \times \{0,1\}^{p(k)}$ (which includes auxiliary input).

The Obfuscator $\mathcal{O}$: To obfuscate a program $P \in \mathcal{P}$, the algorithm $\mathcal{O}$ first "expands" P to include a new input ("$\mathsf{mode} = *$") with hardcoded output $0 \in \{0,1\}^{p(k)}$. It then pads the expanded program to an appropriate size, and runs the obfuscator $\mathcal{O}'$ on the padded program. Finally, it "wraps" the result to accept a *single* input x and run the obfuscated program with the second input mode hardcoded to $\emptyset$ (i.e., "standard" evaluation mode). More formally, let $\mathsf{size} = |\mathcal{O}'(\mathsf{ztoZero}(\mathsf{Expand}(P, z)))|$ for $P \in \mathcal{P}, z \in \{0,1\}^{p(k)}$. Then we define

$$\mathcal{O}(1^k, P; r) := \mathsf{Restrict}\left(\mathcal{O}'\big(\mathsf{pad}\big(\mathsf{Expand}(P, 0), \mathsf{size}\big); r\big), \emptyset\right).$$

Correctness: Follows directly by the correctness of $\mathcal{O}'$. Namely,

$$\begin{aligned}
&\Big[\mathsf{Restrict}\big(\mathcal{O}'\big(\mathsf{pad}\big(\mathsf{Expand}(P,0), \mathsf{size}\big); r\big), \emptyset\big)\Big](x) \\
&\quad = \mathcal{O}'\big(\mathsf{pad}\big(\mathsf{Expand}(P,0), \mathsf{size}\big); r\big)(x, \emptyset) \quad \text{by defn of } \mathsf{Restrict} \\
&\quad = \mathsf{pad}\big(\mathsf{Expand}(P,0), \mathsf{size}\big)(x, \emptyset) \quad \text{whp, by correctness of } \mathcal{O}' \\
&\quad = \mathsf{Expand}(P,0)(x, \emptyset) \quad \text{since } \mathsf{pad} \text{ does not affect input-output} \\
&\quad = P(x) \quad \text{by the defn of } \mathsf{Expand} \text{ (in "standard" mode } \emptyset).
\end{aligned}$$

Security: Consider a (non-uniform) PPT adversary $\mathcal{A}$ and polynomial $q_{\mathcal{A}}$. To prove the theorem, we must construct an extractor algorithm $\mathcal{E}$, polynomial $q_{\mathcal{E}}$, and negligible function ν, and show that for all but negligible probability $\nu(k)$ over $(P_0, P_1, z) \leftarrow \mathcal{D}$, if

$$\Pr\left[b \leftarrow \{0,1\}; \tilde{P} \leftarrow \mathcal{O}(1^k, P_b) : \mathcal{A}(1^k, \tilde{P}, z) = b\right] \geq \frac{1}{2} + \frac{1}{q_{\mathcal{A}}(k)}$$

for these P_0, P_1, z, then $\mathcal{E}$ succeeds with probability $1/q_{\mathcal{E}}(k)$ in extracting a differing input for P_0, P_1, given input $(1^k, P_0, P_1, z)$. Roughly we will do this in three steps: (1) We first show for any $\mathcal{O}$-adversary $\mathcal{A}$ that distinguishes as above, there is an $\mathcal{O}'$-adversary that distinguishes in the *di𝒪* challenge w.r.t. $\mathcal{D}'$ with similar success (making use of the *i𝒪* security of $\mathcal{O}'$ in the process). (2) By the assumed *di𝒪* security of $\mathcal{O}'$ w.r.t. $\mathcal{D}'$, this guarantees existence of an extractor $\mathcal{E}'$ who succeeds in extracting differing inputs from programs sampled as in $\mathcal{D}'$. (3) As the final step, we tweak the extractor $\mathcal{E}'$ to instead work for programs sampled as in $\mathcal{D}$, yielding the desired extractor algorithm.

From the $\mathcal{O}$-adversary $\mathcal{A}$, we begin by constructing a (non-uniform) PPT adversary $\mathcal{A}'$ for the original obfuscator $\mathcal{O}'$ for *di𝒪* security w.r.t. program distribution $\mathcal{D}'$. Recall the distribution $\mathcal{D}'$ corresponds to program pairs (P_0^z, P_1^z) where z has been "embedded" into programs P_0, P_1 (where (P_0, P_1, z) are as in $\mathcal{D}$), and with no auxiliary input.

At a high level, there are two challenges we must solve in order for $\mathcal{A}'$ to make use of the adversary $\mathcal{A}$:

- First, $\mathcal{A}'$ must generate from his challenge $\mathcal{O}'(P_b^z)$ an appropriate value of auxiliary input z to give to $\mathcal{A}$. This will be done by reading the embedded value z at the "special input" of $\mathcal{O}'(P_b^z)$.
- Second (and slightly more complicated), he must convert his $\mathcal{O}'$-obfuscated program $\mathcal{O}'(P_b^z)$ to "look like" an $\mathcal{O}$-obfuscation $\mathcal{O}(P_b)$, where recall $\mathcal{O}(P_b) = \mathsf{Restrict}(\mathcal{O}'(\mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})), \emptyset)$. Indeed, it is possible that $\mathcal{A}$ may successfully distinguish given z and obfuscations of the latter type, yet may not succeed when given z and obfuscations of the former. In particular, note that the program $\mathcal{O}'(P_b^z)$ contains information z that agrees with the auxiliary input, whereas $\mathcal{O}(P_b)$ has only 0 in this role; since we make no assumptions on the structure of the obfuscator $\mathcal{O}'$, there is no direct way to "strip off" this extra information from the obfuscation $\mathcal{O}'(P_b^z)$.

  Instead, to achieve this goal, we make use of a *second layer* of obfuscation. Namely, given the (obfuscated) code $\mathcal{O}'(P_b^z)$ in the $\mathcal{O}'$ challenge, we can generate a second program M_b^0 that has the *same input-output behavior* and size as the desired underlying program $\mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})$ that appears within $\mathcal{O}'$ in the obfuscation $\mathcal{O}(P_b)$. Explicitly,

  $$\Big[\mathsf{pad}\big(\mathsf{Expand}(P_b, 0), \mathsf{size}\big)\Big] \equiv \Big[\mathsf{ztoZero}\big(\mathcal{O}'(P_b^z)\big)\Big], \tag{8}$$

  (i.e., are functionally equivalent), where recall the second program bypasses the code of $\mathcal{O}'(P_b^z)$ on the special input $\mathsf{mode} = *$ and directly outputs 0 (see Definition C.4). Indeed, both programs evaluate to 0 on any input of the form $(x, *)$ and to $P_b(x)$ on any input of the form $(x, \emptyset)$. Define $M_b^0 := \mathsf{ztoZero}(\mathcal{O}'(P_b^z))$. Note that the value of size for pad was precisely chosen to be comparable with the size of the program M_b^0 (see Definition C.4).

  Now, because M_b^0 and $\mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})$ have identical input-output behavior (and size), by the *indistinguishability* obfuscation security of $\mathcal{O}'$ (which, recall, supports auxiliary input; see Remark C.3), it must be that their $\mathcal{O}'$-obfuscations are indistinguishable, even given z:

  $$\Big(\mathcal{O}'(M_b^0), z\Big) \overset{c}{\cong} \Big(\mathcal{O}'(\mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})), z\Big).$$

  This indistinguishability must further be preserved when applying the $\mathsf{Restrict}(\cdot, \emptyset)$ operation to both programs:

  $$\Big(\mathsf{Restrict}\big(\mathcal{O}'(M_b^0), \emptyset\big), z\Big) \overset{c}{\cong} \Big(\mathsf{Restrict}\big(\mathcal{O}'(\mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})), \emptyset\big), z\Big).$$

  But, note that the right-hand distribution is now precisely $(\mathcal{O}(P_b), z)$. We have thus succeeded in converting challenges between $\mathcal{O}'$ and $\mathcal{O}$: if the original $\mathcal{O}$-adversary $\mathcal{A}$ succeeded in predicting b from $(1^k, \mathcal{O}(P_b), z)$, he *must also succeed* when given $(1^k, \mathsf{Restrict}(\mathcal{O}'(M_b^0), \emptyset), z)$, where $\mathsf{Restrict}(\mathcal{O}'(M_b^0), \emptyset) = \mathsf{Restrict}(\mathcal{O}'(\mathsf{ztoZero}(\mathcal{O}'(P_b^z))), \emptyset)$ can be directly simulated using the $\mathcal{O}'$-challenge $\mathcal{O}'(P_b^z)$.

We now formally define the new $\mathcal{O}'$-adversary $\mathcal{A}'$. On input the security parameter 1^k and an obfuscated program $\mathcal{O}'(P_b^z)$ (where the underlying program P_b^z is sampled via $(P_0^z, P_1^z) \leftarrow \mathcal{D}'$; $b \leftarrow \{0,1\}$), $\mathcal{A}'$ performs the following:

1. Evaluate $\mathcal{O}'(P_b^z)$ on the special "$\mathsf{mode} = *$" input to learn $z := \mathcal{O}'(P_b^z)(x,*)$, for arbitrary x.
2. Construct a *new* program to simulate the $\mathcal{O}$-challenge program distribution. Namely,

```
1: Define $M_b^0 := \mathsf{ztoZero}(\mathcal{O}'(P_b^z))$.
2: Sample an obfuscation $\tilde{M}_b^0 \leftarrow \mathcal{O}'(1^k, M_b^0)$, using the original obfuscator $\mathcal{O}'$.
3: Define $M_b := \mathsf{Restrict}(\tilde{M}_b^0, \emptyset)$.
```

3. Execute the adversary $\mathcal{A}$ on input $(1^k, M_b, z)$, and output the resulting bit b'.

We now show that $\mathcal{A}'$ as constructed succeeds in the $di\mathcal{O}$ security challenge for $\mathcal{O}'$ w.r.t. $\mathcal{D}'$ with essentially as much advantage as $\mathcal{A}$ in the security challenge for $\mathcal{O}$. Let $\nu(\cdot)$ be the negligible function provided in the $i\mathcal{O}$ security guarantee for $\mathcal{O}$ (see Definition C.2).

Claim C.6. *By the $i\mathcal{O}$ security of $\mathcal{O}'$, for every triple $(P_0, P_1, z) \in \mathcal{P} \times \mathcal{P} \times \{0,1\}^{p(k)}$ in the support of $\mathcal{D}$, for $P_b^z := \mathsf{Expand}(P_b, z)$, and for the adversary $\mathcal{A}'$ constructed above, it holds that*

$$\begin{aligned}&\Pr\left[b \leftarrow \{0,1\}; \mathcal{O}'(P_b^z) \leftarrow \mathcal{O}'(1^k, P_b^z); \mathsf{guess}' \leftarrow \mathcal{A}'(1^k, \mathcal{O}'(P_b^z)) : \mathsf{guess}' = b\right]\\ &\qquad \geq \Pr\left[b \leftarrow \{0,1\}; \mathcal{O}(P_b) \leftarrow \mathcal{O}(1^k, P_b); \mathsf{guess} \leftarrow \mathcal{A}(1^k, \mathcal{O}(P_b), z) : \mathsf{guess} = b\right] - \nu(k).\end{aligned}$$

Proof. Denote these two probabilities by $\mathsf{win}_{\mathcal{A}'}$ and $\mathsf{win}_{\mathcal{A}}$. Suppose, to the contrary, there is a triple (P_0, P_1, z) for which $\mathcal{A}'$ predicts b in the $\mathcal{O}'$-challenge with probability strictly less than $\nu(k)$ below the probability that $\mathcal{A}$ succeeds in predicting b in the $\mathcal{O}$-challenge: i.e., that $\mathsf{win}_{\mathcal{A}'} < \mathsf{win}_{\mathcal{A}} - \nu(k)$. From this, we construct a (non-uniform) adversary $\mathcal{A}_{i\mathcal{O}}^z$ who has z hardcoded and breaks the $i\mathcal{O}$ security of $\mathcal{O}'$ for the pair of functionally equivalent programs constructed in Equation 8, where one program appears within the obfuscation $\mathcal{O}(P_b)$ as per the definition of $\mathcal{O}$, and the second appears within the strategy of $\mathcal{A}'$ given his challenge $\mathcal{O}'(P_b^z)$.

Recall that by the definition of $\mathcal{O}$ and of $\mathcal{A}'$, we have:

- $\mathcal{O}(P_b) \leftarrow \mathcal{O}(1^k, P_b)$ is equivalent to $\left[\mathsf{Restrict}(\tilde{P}, \emptyset) : \tilde{P} \leftarrow \mathcal{O}'(1^k, [\mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})])\right]$.
- $\mathsf{guess}' \leftarrow \mathcal{A}'(1^k, \mathcal{O}'(P_b^z))$ is equivalent to

$$\left[\mathsf{guess}' \leftarrow \mathcal{A}(1^k, \tilde{P}, z) : z = \mathcal{O}'(P_b^z)(x,*); \tilde{P} \leftarrow \mathcal{O}'(1^k, [\mathsf{ztoZero}(\mathcal{O}'(P_b^z))])\right].$$

Thus, $\mathsf{win}_{\mathcal{A}'}$ and $\mathsf{win}_{\mathcal{A}}$ correspond to the probabilities of the following experiments:

$\mathsf{win}_{\mathcal{A}'}$:		$\mathsf{win}_{\mathcal{A}}$:
$b \leftarrow \{0,1\}$;		$b \leftarrow \{0,1\}$;
$\mathcal{O}'(P_b^z) \leftarrow \mathcal{O}'(1^k, P_b^z)$;		
$z := \mathcal{O}'(P_b^z)(x,*)$;		
$M_b' := \mathsf{ztoZero}(\mathcal{O}'(P_b^z))$;		$M_b := \mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})$;
$\tilde{M}_b' \leftarrow \mathcal{O}'(1^k, M_b')$;		$\tilde{M}_b \leftarrow \mathcal{O}'(1^k, M_b)$;
$\mathsf{guess}' \leftarrow \mathcal{A}(1^k, \tilde{M}_b', z)$;		$\mathsf{guess} \leftarrow \mathcal{A}(1^k, \tilde{M}_b, z)$;
$: \mathsf{guess}' = b$		$: \mathsf{guess} = b$

If $\mathsf{win}_{\mathcal{A}'} < \mathsf{win}_{\mathcal{A}} - \nu(k)$, then in particular there must exist $b \in \{0,1\}$ and $\mathcal{O}'(P_b^z) \in \mathcal{O}'(1^k, P_b^z)$ such that for the pair of programs $M_b' := \mathsf{ztoZero}(\mathcal{O}'(P_b^z))$ and $M_b := \mathsf{pad}(\mathsf{Expand}(P_b, 0), \mathsf{size})$,

$$\begin{aligned}&\Pr[\tilde{M}_b' \leftarrow \mathcal{O}'(1^k, M_b'); \mathsf{guess}' \leftarrow \mathcal{A}(1^k, \tilde{M}_b', z) : \mathsf{guess}' = b]\\ &< \Pr[\tilde{M}_b \leftarrow \mathcal{O}'(1^k, M_b); \mathsf{guess} \leftarrow \mathcal{A}(1^k, \tilde{M}_b, z) : \mathsf{guess} = b] - \nu(k).\end{aligned}$$

But, recall that the programs M_b', M_b are *functionally equivalent* (and of comparable size). Indeed, both accept a pair of inputs (x, mode), and satisfy $M_b'(x,*) = M_b(x,*) = 0$ and $M_b'(x,\emptyset) = M_b(x,\emptyset) = P_b(x)$ (in addition, M_b was padded precisely to the appropriate value of size; see Definition C.4). Thus, the algorithm $\mathcal{A}$ directly gives us an adversary who breaks the $i\mathcal{O}$ security of $\mathcal{O}'$ for the program pair M_b', M_b. That is, consider the (non-uniform) PPT adversary $\mathcal{A}_{i\mathcal{O}}^z$ who has hardcoded the program pair M_b', M_b, the bit b, and the value z from the original triple

(P_0, P_1, z). In the $i\mathcal{O}$ security challenge, $\mathcal{A}^z_{i\mathcal{O}}$ is given as input the security parameter 1^k and a $\mathcal{O}'$-obfuscated program $\tilde{M}$, which is either generated as $\tilde{M} \leftarrow \mathcal{O}'(1^k, M'_b)$ or as $\tilde{M} \leftarrow \mathcal{O}'(1^k, M_b)$. The adversary $\mathcal{A}^z_{i\mathcal{O}}(1^k, \tilde{M})$:

```
1: Evaluate $\mathsf{guess} \leftarrow \mathcal{A}(1^k, \tilde{M}, z)$.
2: If $\mathsf{guess} = b$, output 1 (i.e., guess of $M_b$); otherwise, output 0.
```

By construction, in combination with the above, the probability that $\mathcal{A}^z_{i\mathcal{O}}$ outputs 1 in the case that $\tilde{M} \leftarrow \mathcal{O}'(1^k, M'_b)$ is *strictly smaller* than $\nu(k)$ less than the probability that $\mathcal{A}^z_{i\mathcal{O}}$ outputs 1 in the case that $\tilde{M} \leftarrow \mathcal{O}'(1^k, M_b)$. This contradicts the $i\mathcal{O}$ security of $\mathcal{O}'$. The claim follows. $\square$

Now, suppose for some $(P_0, P_1, z) \in \mathsf{supp}(\mathcal{D})$ that $\mathcal{A}$ succeeds in guessing b in the $\mathcal{O}$-obfuscation challenge:

$$\Pr\left[b \leftarrow \{0,1\}; \tilde{P} \leftarrow \mathcal{O}(1^k, P_b) : \mathcal{A}(1^k, \tilde{P}, z) = b\right] \geq \frac{1}{2} + \frac{1}{q_{\mathcal{A}}(k)}.$$

By Claim C.6, it must thus be that $\mathcal{A}'$ similarly succeeds in guessing b in the $\mathcal{O}'$-obfuscation challenge for the corresponding pair $(P^z_0, P^z_1) \in \mathsf{supp}(\mathcal{D}')$:

$$\Pr\left[b \leftarrow \{0,1\}; \tilde{P} \leftarrow \mathcal{O}'(1^k, P^z_b) : \mathcal{A}'(1^k, \tilde{P}) = b\right] \geq \frac{1}{2} + \frac{1}{q_{\mathcal{A}}(k)} - \nu(k) \qquad (9)$$

$$\geq 1/2 + 1/[2q_{\mathcal{A}}(k)]. \qquad (10)$$

By the $di\mathcal{O}$ security of $\mathcal{O}'$, we know that for this adversary $\mathcal{A}'$ and polynomial $2q_{\mathcal{A}}$, there exists an extractor $\mathcal{E}'$ and polynomial $q_{\mathcal{E}'}$ such that, with all but negligible probability over $(P^z_0, P^z_1) \leftarrow \mathcal{D}'$, if (10) holds, then with probability at least $1/q_{\mathcal{E}'}(k)$, running the extractor $(x^*, \mathsf{mode}) \leftarrow \mathcal{E}'(1^k, P^z_0, P^z_1)$ succeeds in finding a differing input (x^*, mode) for which $P^z_0(x^*) \neq P^z_1(x^*, \mathsf{mode})$.

As the final step, we wish to construct from $\mathcal{E}'$ a new extractor $\mathcal{E}$ that instead, given a tuple $(1^k, P_0, P_1, z)$, outputs a differing input x^* for which the programs P_0 and P_1 disagree. Define the algorithm $\mathcal{E}'$ that, on input $(1^k, P_0, P_1, z)$, constructs the programs $P^z_0 := \mathsf{Expand}(P_0, z)$ and $P^z_1 := \mathsf{Expand}(P_1, z)$, executes the original extractor algorithm $\mathcal{E}'$ as $(x^*, \mathsf{mode}) \leftarrow \mathcal{E}'(1^k, P^z_0, P^z_1)$, and outputs the value x^*. Since by construction, $P^z_0(x, *) = P^z_1(x, *) = z$ for every value of x (corresponding to "special" input mode), it must be that the differing input (x^*, mode) generated by $\mathcal{E}'$ satisfies $\mathsf{mode} = \emptyset$ (i.e., "standard" mode evaluation). That is, it must be that $P^z_0(x^*, \emptyset) \neq P^z_1(x^*, \emptyset)$, which means precisely that $P_0(x^*) \neq P_1(x^*)$.

Theorem C.1 follows.

$\square$

Applying the theorem to [GGHW14]. In [GGHW14], the authors propose a new "special-purpose" Turing machine obfuscation assumption, and demonstrate based on this assumption a distribution of circuits and bounded-length auxiliary input for which $di\mathcal{O}$ cannot exist. Applying Theorem C.1, we conclude that, assuming such special-purpose obfuscation exists, there exists a distribution of circuits that is $di\mathcal{O}$-unobfuscatable for any $i\mathcal{O}$ obfuscator, *even when no auxiliary input is present.*

Intuitively, their counterexample is for $di\mathcal{O}$ obfuscation of circuits that differ exactly on inputs corresponding to valid message-signature pairs (with respect to a secure digital signature scheme). Their "bad" auxiliary input distribution amounts to a "special" obfuscated program with side information on the signing key: the program takes as input a circuit description C, signs a hash of the circuit (to produce unique messages), and outputs the bit obtained by running C on the resulting message-signature pair. Access to this program allows an adversary to distinguish obfuscations of circuits as above, but (by their special obfuscation assumption) does not enable any efficient algorithm to extract a message-signature pair.

More formally, let $\mathcal{H}$ be a collision-resistant hash function family, and $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ a signature scheme (with keys denoted $(\mathsf{sk}, \mathsf{vk})$). The "special-purpose" obfuscation assumption of [GGHW14] is given below, in Definition C.7 and Conjecture C.8.

Definition C.7 (Special-Purpose TM Class $\{P^*_{H,\mathsf{sk}}\}$)**.** [GGHW14].
Program $P^*_{H,\mathsf{sk}}$: (for hardcoded $H \in \mathcal{H}$, signing key sk)
Input: circuit description C of size $s(k)$ with 1-bit output.

```
1: Compute $m = H(C)$ and $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$ .
2: Output the bit $C(m, \sigma, 0)$.
```

Conjecture C.8 (Special-Purpose TM Obfuscation [GGHW14])**.** *There exists a deterministic signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$*, collision-resistant hash function family* $\mathcal{H}$*, and an obfuscator* $\mathsf{sp}\mathcal{O}$*, such that* $\mathsf{sp}\mathcal{O}$ *satisfies correctness and the following security property. There exists a negligible function* ν *such that for any PPT attacker* $\mathcal{A}$*,*

$$\Pr\Big[(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k); H \leftarrow \mathcal{H}(1^k); \tilde{P} \leftarrow \mathsf{sp}\mathcal{O}(1^k, P^*_{H,\mathsf{sk}}); (m, \sigma) \leftarrow \mathcal{A}(1^k, \mathsf{vk}, \tilde{P}) \\ : \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1\Big] \leq \nu(k),$$

where $P^*_{H,\mathsf{sk}}$ *is the Turing machine defined in Definition C.7.*

Garg *et al.* [GGHW14] prove the following theorem.

Theorem C.9 ([GGHW14])**.** *Under the special-purpose Turing machine obfuscation conjecture (Conjecture C.8), there exists a distribution* $\mathcal{D}$ *over* $P/poly \times P/poly \times \mathcal{Z}$ *of circuit pairs and bounded-length auxiliary input information for which there cannot exist secure differing-inputs obfuscation.*

Combining this with Theorem C.1, we thus obtain the following corollary.

Corollary C.10. *Under the Special-Purpose TM Obfuscation Conjecture C.8, there exists a distribution of circuit pairs* $\mathcal{D}'$ *over* $P/poly \times P/poly$ *with respect to which no* $i\mathcal{O}$*-secure obfuscator can be* $di\mathcal{O}$*-secure,* even in the absence of auxiliary input.

In particular, this implies that if the special-purpose obfuscator conjecture holds, then general-purpose $di\mathcal{O}$ cannot exist, even without auxiliary input.

Corollary C.11. *Under the Special-Purpose TM Obfuscation Conjecture C.8, there cannot exist* $di\mathcal{O}$ *for* $P/poly$*,* even in the absence of auxiliary input.

Proof. Follows from Corollary C.10, since any $di\mathcal{O}$ obfuscator for $P/poly$ is also $i\mathcal{O}$ secure for $P/poly$ and $di\mathcal{O}$-secure for any efficiently samplable distribution of circuits $\mathcal{D}'$ over $P/poly \times P/poly$. □