ON PROGRAM OBFUSCATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Sidharth Durgesh Telang

February 2016

ON PROGRAM OBFUSCATION

Sidharth Durgesh Telang, Ph.D.

Cornell University 2016

Program obfuscation is an exciting new area of research with wide-ranging applications and implications throughout cryptography and computer science. The most widely accepted security notion for obfuscation is indistinguishability obfuscation (*iO*), which on one hand is surprisingly useful for many applications [134, 42] and on the other hand can plausibly exist [70]. Despite this amazing progress, there are still some barriers to its ambition as a "central hub" in cryptography. Firstly, all known obfuscation mechanisms incur an overhead proportional to the circuit size of the program being obfuscated, leading to a major source of inefficiency. Secondly, there are no plausible, natural intractability assumptions on which these mechanisms can be based.

In this thesis we study the above issues.

- We construct an indistinguishability obfuscator for Turing machines and RAM programs, where the obfuscation overhead is independent of the running time of the program.
- We introduce a natural assumption on multilinear encodings, a candidate for which was provided in the seminal work of [66], and show this assumption, together with other standard hardness assumptions, suffices to construct *iO* for circuits.
- We show methods to bootstrap *iO* from quantitatively weaker notions of *iO* (in particular, notions with significantly relaxed efficiency guarantees).

In the process of obtaining the above results, we discover fascinating connections between obfuscation and randomized encodings [8, 10]. In particular, we study the notion of randomized encodings in a new setting where the time to encode a program is independent of the running time of a program. We show that this setting for randomized encoding is not only interesting in its own right from a theoretical perspective, but also has many applications in cryptography.

BIOGRAPHICAL SKETCH

Sidharth got his B.Tech in Computer Science and Engineering from the Indian Institute of Technology Delhi in 2011, and joined the Computer Science department at Cornell as a Ph.D. student soon after. His research is in the area of cryptography, where he was advised by Prof. Rafael Pass. In 2013, Sidharth followed his advisor to Cornell Tech in New York City, where he finished his Ph.D. in 2015. Along the way, Sidharth spent a few summers at Microsoft Research India, IBM Research and Google.

To Mom, Papa and Isha

This thesis is dedicated to you.

ACKNOWLEDGEMENTS

Firstly, I would like to thank Rafael Pass, for not only teaching me what I know about cryptography, but also for his influence on how I have grown as a person these past four years. I continue to be inspired by his endless energy, ability to extract the essence of an idea from a sea of details, sheer creative power and above all his patience and constant encouragement. For all this and more, thank you!

I would like to thank my other collaborators: Karn Seth, Rachel Lin, Kai-Min Chung, Nir Bitansky and Sanjam Garg. Apart from being great researchers, they are all fantastic at communicating and brainstorming complex ideas in a very simple intuitive manner, which was crucial to building my own intuition and thinking style.

A special thank you to Vipul Goyal and Tal Malkin for hosting me for two very memorable summers. I believe I grew a lot as a researcher from those experiences. Also, I would like to thank Vladimir Kolovski and Miguel Andres for a fantastic summer at Google, where they guided me out into the real world and provided a lot of new perspective.

I would also like to thank my committee members: Eva Tardos, Ari Juels, Joe Halpern and Vitaly Shmatikov for their helpful feedback. A special thanks to Eva for her frank and patient advice on research and life in general.

Another special thank you to my office-mates Karn, Stavros, Zhi and Adith, and my room-mates Karn, Banga, Joshi and TJ. It's amazing I got all this done with you guys around.

Last but farthest from the least, I would like to thank my parents for their constant support and encouragement. *This* is what I've been upto all these years away from home.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Program Obfuscation

The goal of program obfuscation is to scramble the code of a program, hiding its implementation details and hence making it hard to "reverse engineer", while preserving the functionality (*i.e.* input-output behavior) of the program. While program obfuscation is a useful primitive in practice, with several heuristic solutions widely prevalent, it is also potentially very useful in cryptography. For example, in their seminal paper on public-key cryptography [64], Diffie and Hellman envision the use of a program obfuscator (called a "one-way compiler" in their paper [1]) to heuristically construct a public-key encryption scheme from any private-key encryption scheme: the public key is simply the obfuscated private-key encryption algorithm (with the private key hardcoded in it). Indeed, the prospect of program obfuscation has potential "dream" applications in almost all areas of cryptography. However, formally defining what it means to "scramble" a program is non-trivial: on the one hand, we want a definition that can be plausibly satisfied, on the other hand, we want a definition that is useful for applications.

A first formal definition of such program obfuscation was provided by Hada [98]: roughly speaking, Hada's definition—let us refer to it as *strongly virtual black-box*—is formalized using the simulation paradigm. It requires that anything an attacker can learn from the obfuscated code, could be simulated using

[1] [64] describe an obfuscator as a compiler "which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language".

just black-box access to the functionality. Unfortunately, as noted by Hada, only learnable functionalities can satisfy such a strong notion of obfuscation: if the attacker simply outputs the code it is given, the simulator must be able to recover the code by simply querying the functionality and thus the functionality must be learnable.

An in-depth study of program obfuscation was initiated in the seminal work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [17]. Their central result shows that even if we consider a more relaxed simulation-based definition of program obfuscation—called *virtual black-box (VBB)* obfuscation—where the attacker is restricted to simply outputting a single bit, impossibility can still be established. Their result is even stronger, demonstrating the existence of families of functions such that given black-box access to f_s (for a randomly chosen s), not even a *single* bit of s can be guessed with probability significantly better than $1/2$, but given the code of any program that computes f_s, the entire secret s can be recovered. Thus, even quite weak simulation-based notions of obfuscation are impossible.

But weaker notions of obfuscation may be achievable, and may still suffice for (some) applications. Indeed, Barak *et al.* [17] also suggested such a notion called *indistinguishability obfuscation*. Roughly speaking, this notion requires that obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ of any two functionally *equivalent* circuits C_1 and C_2 (i.e. whose outputs agree on all inputs) from some class $\mathcal{C}$ are computationally indistinguishable. In a breakthrough result, Garg, Gentry, Halevi, Raykova, Sahai, and Waters [70] provided the first candidate constructions of indistinguishability obfuscators for all polynomial-size circuits, based on so-called *multilinear (a.k.a. graded) encodings* [38, 133, 66]—for which candidate constructions

were discovered in the seminal work of Garg, Gentry and Halevi [66].

Applicability of Indistinguishability Obfuscation

It may not be immediately clear why indistinguishability obfuscation is a useful security notion: it only guarantees that the obfuscation of a program does not leak more information than what an obfuscation of a functionally equivalent program would leak. However, there has been a long line of work that shows indistinguishability obfuscation suffices for many of the "dream" applications of obfuscation (see e.g., [134, 99, 42, 99, 40, 67, 88, 111, 110]). Most notable among these is the work of Sahai and Waters [134] (and the "punctured program" paradigm it introduces) which shows that for many interesting applications of virtual black-box obfuscation (including turning private-key primitives into public-key one), the weaker notion of indistinguishability obfuscation suffices. Furthermore, as shown by Goldwasser and Rothblum [96], indistinguishability obfuscators provide a very nice "best-possible" obfuscation guarantee: if a functionality can be VBB obfuscated (even non-efficiently!), then any indistinguishability obfuscator for this functionality is VBB secure.

Despite this amazing progress, a key obstacle in the applicability of the candidate obfuscation mechanisms is that they are highly inefficient. A key source of inefficiency is that these mechanisms require the program being obfuscated be expressed as a boolean circuit, and these obfuscators incur a polynomial overhead in the size of the circuit. As a result, the time to obfuscate a program, and the size of the obfuscated program, is polynomial in the running time of the program. Not only does this make the use of obfuscation in the above applications inefficient but also limits its use in a number of settings. For example, consider the scenario of a weak client wishing to delegate some computation to

a powerful albeit untrusted server. This computation may have some private information in its implementation which it wishes to hide from the server. The natural solution of the client uploading an obfuscation of the program does not work because it is more expensive than running the computation itself.

In this thesis we will address the following question.

Can we obfuscate programs such that the obfuscator running time is independent of the running time of the program?

Foundations of Indistinguishability Obfuscation

We now turn our attention to the goal of basing indistinguishability obfuscation on some natural intractability assumption. While the construction of indistinguishability obfuscation of Garg et al is based on *some* intractability assumption on the underlying multilinear encoding scheme, the assumption is very tightly tied to their scheme—in essence, the assumption stipulates that their scheme is a secure indistinguishability obfuscator. In particular, their obfuscator proceeds in two steps: They first provide a candidate construction of an indistinguishability obfuscator for NC^1. Their assumption essentially stipulates that this construction is secure. Next, they demonstrate a "bootstrapping" theorem showing how to use fully homomorphic encryption (FHE) schemes [79] and indistinguishability obfuscators for NC^1 to obtain indistinguishability obfuscators for all polynomial-size circuits.

Further constructions of obfuscators for NC^1 were subsequently provided by Brakerski and Rothblum [50] and Barak, Garg, Kalai, Paneth and Sahai [16]—in fact, these constructions achieve the even stronger notion of virtual-black-

box obfuscation in idealized "generic" multilinear encoding models. These models require all attacks to treat multilinear encoding as if they were "physical envelopes" on which multilinear operations can be performed. But at the same time, non-generic attacks against their scheme are known—since general-purpose VBB obfuscation is impossible. Thus, it is not clear to what extent security arguments in the generic multilinear encoding model should make us more confident that these constructions satisfy a notion of indistinguishability obfuscation.

Therefore, at this point, the following question still remains open:

> *Can the security of a general-purpose indistinguishability obfuscator be reduced to some "natural" intractability assumption?*

In this thesis, we initiate a study of the above-mentioned question. In particular we study two approaches towards this goal.

Approach 1: Basing obfuscation directly on multilinear encodings. We are concerned with the question of whether some *succinct* and *general* assumption (that is interesting in its own right, and is not "tailored" to a particular obfuscation construction) on multilinear encodings can be used to obtain indistinguishability obfuscation. More importantly, we are interested in *reducing* the security of the obfuscation to some *simpler* assumption, not just in terms of "description size" but in terms of computational complexity—that is, we are not interested in assumptions that "directly" (without any security reduction) imply security of the obfuscation.

Approach 2: Bootstrapping obfuscation from weaker notions. In this ap-

proach, we are interested in studying *weaker* notions of indistinguishability obfuscation that can amplified/bootstrapped into the standard notion of indistinguishability obfuscation:

> *Identify quantitatively weaker notions of indistinguishability obfuscation ($i\mathcal{O}$) that can be amplified into the "standard" notion of $i\mathcal{O}$.*

Not only is this interesting from a theoretical perspective, but we believe this has the potential to simplify future constructions of $i\mathcal{O}$. Future constructions from multilinear encodings can focus on the simpler task of achieving these weaker notions, opening the possibility of better constructions and assumptions to base $i\mathcal{O}$ on. Also, it may be possible to achieve these weaker notions, and hence $i\mathcal{O}$, from entirely different primitives.

Note that this approach is orthogonal to the aforementioned bootstrapping theorem of [70], which studies whether $i\mathcal{O}$ for "weak" *classes of circuits* (e.g., NC^1 circuits) can be bootstrapped to $i\mathcal{O}$ for all polynomial-size circuits. Instead, this approach is similar to e.g., the hardness amplification approach of Yao [137] of amplifying a *weak one-way function* into a strong one.

While uncovering answers to the aforementioned questions, we discover intriguing connections between obfuscation and a cryptographic primitive called randomized encodings, introduced in the next section.

1.2 Randomized Encodings

The beautiful notion of a *randomized encoding (RE)*, introduced by Ishai and Kushilevitz [100], aims to trade the computation of a "complex" (deterministic) function Π on a given input x for the computation of a "simpler" randomized function—the "encoding algorithm"—whose output distribution $\hat{\Pi}(x)$ encodes $\Pi(x)$ (from which $\Pi(x)$ can be efficiently decoded, or "evaluated"). Furthermore, the encoding $\hat{\Pi}(x)$ should not reveal anything beyond $\Pi(x)$; this is referred to as the *privacy*, or *security*, property of randomized encodings and is typically defined through the simulation paradigm [95].

Most previous work have focused on randomized encodings where encodings can be computed in lower parallel-time complexity than what is required for computing the original function Π. For instance, all log-space computations have *perfectly-secure* randomized encodings in $\mathbf{NC}^0$ [100, 102, 8], and assuming low-depth pseudo-random generators, this extends to all polynomial-time computations (with computational security) [10, 138]. Such randomized encodings have been shown to have various applications to parallel cryptography, secure computation, verifiable delegation, etc. (see [6] for a survey).

In this thesis, we focus on another natural complexity measure: *the time required to compute* $\widehat{f}(x)$. Specifically, given the description of f and the input x, we would like to compute the encoding $\widehat{f}(x)$ in time $\widehat{T}$ that is significantly smaller than the time T required to compute $f(x)$. Decoding time, in contrast, would be as large as T, perhaps with some tolerable overhead. For this goal to be achievable, f has to be given in some succinct representation that is smaller than T, and cannot be given by, say, a size-T circuit. Concretely, we focus on the natural

case that f is represented by a succinct program Π, e.g., a Turing machine (TM) or a random-access machine (RAM). For concreteness we will consider the case of Turing machines in the rest of this section.

Besides being interesting from a purely complexity-theoretic perspective, such randomized encodings may have powerful applications analogous to those of the known randomized encodings. One such immediate application is private delegation of computation: a weak client that wishes to use the aid of a server to run a long computation Π on a short private input x, may quickly compute a randomized encoding $\widehat{\Pi}(x)$, and have the server decode the result $\Pi(x)$, without the server learning anything regarding x (with a little more effort, we can even ensure privacy of the output, and be able to verify that the server computed correctly).

Given a description of a Turing machine Π and input x with running time T and output length l, we consider three notions of efficiency for randomized encodings $\hat{\Pi}(x)$ of $\Pi(x)$.

- *succinct RE*: Encoding time (and thus also size of the encodings) is $\text{poly}(|\Pi|, |x|, l)$.
- *compact RE*: Encoding time (and thus also size) is $\text{poly}(|\Pi|, |x|, \log T)$
- *sublinear RE*: Encoding time (and thus also size) is bounded by $\text{poly}(|\Pi|, |x|) * T^{1-\epsilon}$, for some $\epsilon > 0$.

We assume without loss of generality that the randomized encoding $\hat{\Pi}(x)$ of Π, x itself is a program, and that the decoding/evaluation algorithm simply executes $\hat{\Pi}(x)$.

Randomized encodings with compact or sublinear efficiency can be thought of as *output-compressing* randomized encodings. For a machine with long outputs, an encoding of the machine will be shorter than the output, and hence can be thought of as a compression of the output which can de-compressed by decoding/evaluating the encoding.

It is easy to see that for output-compressing randomized encodings, the standard simulation-based notion of security is impossible to achieve—roughly speaking, the simulator given just $\Pi(x)$ needs to output a "compressed" version of it, which is impossible if $\Pi(x)$ has high pseudo-Kolmogorov complexity (e.g., if Π is a PRG). Consequently, we consider weaker indistinguishability-based notions of privacy.

- One natural indistinguishability based notion of privacy simply requires that encoding $\hat{\Pi}_0(x_0)$ and $\hat{\Pi}_1(x_1)$ are indistinguishable as long as $\Pi_0(x_0) = \Pi_1(x_1)$ and $\mathsf{Time}(\Pi_0(x_0)) = \mathsf{Time}(\Pi_1(x_1))$, where $\mathsf{Time}(\Pi(x))$ is the running-time of $\Pi(x)$.
- In this work, we consider a stronger notion which requires indistinguishability of $\hat{\Pi}_0(x_0)$ and $\hat{\Pi}_0(x_1)$ as long as Π_0, x_0 and Π_1, x_1 are sampled from some distributions such that $\Pi_0(x_0), \mathsf{Time}(\Pi_0(x_0))$ and $\Pi_1(x_1), \mathsf{Time}(\Pi_1(x_1))$ are indistinguishable. We refer to this notion as *distributional indistinguishability security*.

At this point, it is worthwhile to note that randomized encoding schemes for Turing machines that satisfy the first, weaker indistinguishability security notion can be thought of as indistinguishability obfuscators for a very restricted class of Turing machines: machines that have a fixed input hardcoded in them and take no further input.

In addition to the above, we highlight more fascinating connections between obfuscation and randomized encodings as we describe our results on obfuscation in the next section.

1.3 Our Contributions

We organize the contents of this thesis as answers to the three central questions raised in Section 1.1, reproduced here for convenience.

- Can we obfuscate Turing machines and RAM programs directly? That is, can we obfuscate programs such that the obfuscator running time is independent of the running time of the program?
- Can the security of a general-purpose indistinguishability obfuscator be reduced to some "natural" intractability assumption on the underlying multilinear encoding scheme?
- What are quantitatively weaker notions of indistinguishability obfuscation ($i\mathcal{O}$) that can be amplified into the "standard" notion of $i\mathcal{O}$?

The different notions of randomized encodings introduced in Section 1.2 will serve as very useful intermediate notions in the following results, with applications and deeper implications of their own.

1.3.1 $i\mathcal{O}$ for Turing machines and RAM programs

In Chapter 2 of this thesis we describe how to construct an indistinguishability obfuscator for both Turing machines and RAM programs from an indistin-

guishability obfuscator from circuits. While our obfuscator running time is essentially independent of the running time of the program being obfuscated Π, it depends polynomially on the space complexity of Π.

Our core contribution is a succinct randomized encoding (relying on *iO* for *circuits*) for any class of *a-priori bounded-space computations*. That is, the time to encode depends on the space complexity of the computation, but is essentially independent of its time complexity.

Theorem 1 (Main Theorem, Informally Stated)**.** *Assume the existence of iO for* $\mathsf{P/poly}$ *and one-way functions. Then, for every polynomial* $s(\cdot)$*, there exists a succinct randomized encoding (or garbling scheme) for all polynomial-time programs* Π *with space-complexity* $S(n) \leq s(n)$*. Specifically, the time to encode depends polynomially on the size of* Π*, the lengths* (n, m) *of its input and output, and the space bound* $s(n)$*, but only polylogarithmically on* Π*'s running-time.*

We then demonstrate the power of succinct randomized encodings in several applications, some new, and some analogous to previous applications of randomized encodings, but with new succinctness properties. Examples include succinct functional encryption, succinct NIZKs and publicly verifiable delegation (for more details see Chapter 2). However, our strongest application of succinct randomized encodings is *iO* for bounded-space computations. Indistinguishability here means that the obfuscations of two programs that have the same output and running time on all input of some apriori-bounded length n are computationally indistinguishable. The construction is based on subexponentially-secure *iO* for circuits.

Theorem 2 (Informally Stated)**.** *Assume the existence of succinct randomized encodings for space-bounded programs, one-way functions, and iO for* $\mathsf{P/poly}$ *that are all*

subexponentially-secure. Then, for every polynomial $s(\cdot)$, there exists a succinct $i\mathcal{O}$ for all polynomial-time programs Π with space-complexity $S(n) \leq s(n)$, Specifically, the time to obfuscate Π depends polynomially on the size of Π, the input length n, and the space bound $s(n)$, but only polylogarithmically on Π's running-time and output length m.

In concurrent work, Canetti, Holmgren, Jain, and Vaikuntanathan [54] construct $i\mathcal{O}$ for RAMs assuming subexponentially-secure $i\mathcal{O}$ for $\mathsf{P/poly}$. The complexity of their $i\mathcal{O}$ is also such that obfuscation depends on an a-priori bound on space, but not on the running time. This, in particular, implies a succinct randomized encoding with similar parameters. In a beautiful subsequent work, Koppula, Lewko, and Waters [113] construct fully-succinct randomized encodings from $i\mathcal{O}$ *i.e.* succinct randomized encodings without the dependence on the space complexity. Such a scheme yields an indistinguishability obfuscator with similar parameters by the same transformation of Theorem 2.

1.3.2 $i\mathcal{O}$ from Semantically-secure Multilinear Encodings

In Chapter 3, we introduce a a new assumption on multilinear (a.k.a. graded) encoding schemes called semantic security and show how to construct indistinguishability obfuscators for NC^1 from semantically -secure schemes.

Multilinear encoding schemes, roughly speaking, are schemes to encode ring elements in a way that hides the ring elements but enables computations of certain restricted set of arithmetic circuits on the elements and finally determine whether the output of the circuit is 0 or not; we refer to these circuits as the *legal arithmetic circuits*.

In essence, our notion of *semantical security* attempts to capture the intuition that encodings of ring elements m_0 and m_1 are indistinguishable in the presence of encodings of "auxiliary" ring elements $\vec{z}$, as long as $m_0, m_1, \vec{z}$ are sampled from *any* "nice" distribution D [2]. Defining what makes a distribution D "nice" turns out to be quite non-trivial: A first (and minimal) approach—similar to e.g., the uber assumption of [35] in the context of bilinear maps—would be to simply require that D samples elements $\vec{m}_0, \vec{m}_1, \vec{z}$ such that no generic attacker (that is, an attacker who can access these encodings only by querying *legal* arithmetic circuits on them) can distinguish $\vec{m}_0, \vec{z}$ and $\vec{m}_1, \vec{z}$. It turns out that the most natural formalization of this approach can be attacked assuming standard cryptographic hardness assumptions!

However, the distribution D used in the above attack is a bit "unnatural": encodings of $\vec{m}_b, \vec{z}$ actually leak information about $\vec{m}_b$ even to generic attackers. Our notion of a *nice* message distribution disallows such information leakage w.r.t. generic attacks. More precisely, we require that the output of every (even unbounded-size) legal arithmetic circuit C not only be the same on $(m_0, \vec{z})$ and $(m_1, \vec{z})$ but be *constant* with overwhelming probability over $(m_b, \vec{z})$. We refer to any distribution D satisfying this property as being *valid*, and our formal definition of semantical security now only quantifies over such valid message distributions.

Our central result shows how to construct indistinguishability obfuscators for NC^1 based on the existence of *constant-message* semantically-secure multilinear encodings (where m_0 and m_1 are replced with constant-length sequences of elements).

[2] This is reminiscent of the DDH assumption, where $\vec{z}$ is a vector of independent uniform elements, m_0 is the product of the elements in $\vec{z}$ and m_1 is an independent uniform element.

Theorem 1 (Informally stated)**.** *Assume the existence of semantically secure multilinear encodings. Then there exists an indistinguishability obfuscator for* NC^1.

As far as we know, this is the first result presenting indistinguishability obfuscators for NC^1 based on any type of assumption with a "non-trivial" security reduction w.r.t. arbitrary attackers as opposed to generic attackers.

1.3.3 *iO* from Exponentially-efficient *iO*

In Chapter 5, we introduce a substantial weakening of *iO* called **XiO** and show how to bootstrap **XiO** into standard *iO* (additionally assuming the LWE assumption). The notion of sublinear randomized encodings, with distributional indistinguishability plays a key role in this result as an intermediate primitive.

Recall that indistinguishability obfuscators with running time

$$T_0(|C|, \lambda, n) = \mathrm{poly}(|C|, \lambda) \cdot 2^n,$$

and size

$$\mathsf{Size}_0(|C|, \lambda, n) = \mathrm{poly}(|C|, \lambda) \cdot 2^n,$$

where C is the circuit to be obfuscated, λ is the security parameter, and n is the input length of C, exists *unconditionally*—simply output the function table of C (i.e., the output of C on all possible inputs). Such inefficient *iO* , however, are not useful for applications.

We here consider *iO* with just "slightly non-trivial" running-time; namely, we allow the running time to be

$$T_0(|C|, \lambda, n) = \mathrm{poly}(|C|, \lambda) \cdot 2^n,$$

but require the *size* of the obfuscation to be

$$\mathsf{Size}_\epsilon(|C|, \lambda, n) = \mathrm{poly}(|C|, \lambda) \cdot 2^{n(1-\epsilon)}$$

where $\epsilon > 0$. We refer to this notion as *iO with exponential efficiency*, or simply *exponentially-efficient iO (XiO)* (Recall that, in contrast, for "standard" *iO*, the running time and size of the obfuscator is required to be $\mathrm{poly}(|C|, \lambda)$). In essence, **XiO** requires the obfuscator to be just slightly smaller than a brute-force canonicalization of the circuit. Note that **XiO** obfuscators are only efficiently computable for circuits that take short inputs; we thus here restrict our attention to **XiO** for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$, the class of polynomial size circuits with logarithmic input length..

Theorem 3. *Assume subexponential security of the LWE assumption, and the existence of subexponentially-secure XiO for* $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$. *Then there exists subexponentially-secure iO for* P/poly.

A crucial intermediate notion used to prove Theorem 3 is sublinear randomized encodings with distributional indistinguishability. In fact, the theorem only relies on randomized encoding schemes that work in a weaker CRS model (a one-time trusted setup is performed, generating public parameters which are required when encoding machines). Such randomized encoding schemes are an interesting primitive on its own and are extensively studied in Chapter 4. There it is shown that in the standard model (*i.e.* without a setup phase) sublinear randomized encodings are in fact not possible.

Theorem 4. *Assume the existence of subexponentially secure one-way functions. Then, there do not exists subexponentially-secure sublinear RE with distributional indistinguishability.*

A crucial part of the proof of both Theorem 3 and Theorem 4 is a transformation from a sublinear randomized encoding scheme with distributional indistinguishability to *iO* for circuits.

Lemma 1. *The existence of subexponentially-secure sublinear (resp. compact) RE with distributional indistinguishability and one-way functions implies the existence of subexponentially-secure iO for circuits (resp. for Turing machines).*

Recall that compact randomized encodings can be viewed as *iO* for a restrictive class of programs that take no input. In other words, while an obfuscated program can be evaluated multiple times on different inputs, an encoded program is only good for one-time evaluation. Hence, the above lemma can be thought of as going from a one-time use primitive (randomized encodings) to a multi-time use primitive (obfuscation). This is reminiscent of the result of [87] which constructs a pseudorandom function from a pseudorandom generator, and indeed our construction is conceptually very similar to theirs. This is indicative of a more general technique, which could find use in other areas of cryptography and theoretical computer science.

1.4 Bibliography

The subsequent chapters are based on the following papers.

Chapter 2: Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 439–448, 2015

Chapter 3: Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 500–517, 2014

Chapter 4: Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. *Proceedings of the Thirteen Theory of Cryptography Conference, TCC 2016, Tel-Aviv, Israel, January 10-13, 2016*, To appear

Chapter 5: Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Non-trivially efficient indistinguishability obfuscation. *Proceedings of the Nineteenth International Conference on the Theory and Practice of Public-Key Cryptography, PKC 2016, Taipei, Taiwan, March 6-9, 2016*, To appear

CHAPTER 2

SUCCINCT RANDOMIZED ENCODINGS AND APPLICATIONS

This chapter contains joint work with Nir Bitansky (MIT), Sanjam Garg (UC Berkeley), Huijia Lin (UCSB) and Rafael Pass (Cornell University).

2.1 Introduction

The notion of a *randomized encoding*, coined by Ishai and Kushilevitz [101], aims to trade the computation of a "complex" function $f(x)$ for the computation of a "simpler" randomized function whose output distribution $\widehat{f}(x)$ encodes $f(x)$, but hides anything else regarding f and x. The "complexity" of computing f is shifted to a decoding procedure that extracts $f(x)$ from $\widehat{f}(x)$.

The privacy of the function f and input x is naturally captured by an efficient simulator $\mathsf{Sim}(f(x))$, who given only the output $f(x)$, produces a simulated encoding indistinguishable from $\widehat{f}(x)$; privacy can be perfect, statistical, or computational, according to the attained indistinguishability. Capturing what it means to "simplify the computation of $f(x)$" may take quite different forms according to the complexity measure of interest. Most previous work have focused on computing the randomized encoding $\widehat{f}(x)$ with lower parallel-time complexity than required for computing the original function f, and has been quite successful. In particular, all log-space computations were shown to have perfectly-private randomized encodings in NC^0 [101, 103, 9]. When settling for privacy against computationally bounded adversaries, and assuming low-depth pseudo-random generators, the latter extends to arbitrary poly-time computations [11], which was already demonstrated in Yao's seminal work on

garbling circuits [138]. The constructed randomized encodings were in turn shown to have various strong applications to parallel cryptography, secure computation, verifiable delegation, and more (see [6] for a survey).

Succinct Randomized Encodings. In this work, we focus on another natural complexity measure: *the time required to compute* $\widehat{f}(x)$. Specifically, given the description of f and the input x, we would like to compute the encoding $\widehat{f}(x)$ in time $\widehat{T}$ that is significantly smaller than the time T required to compute $f(x)$. Decoding time, in contrast, would be as large as T, perhaps with some tolerable overhead. For this goal to be achievable, f has to be given in some succinct representation that is smaller than T, and cannot be given by, say, a size-T circuit. Concretely, we focus on the natural case that f is represented by a succinct program Π, e.g., a Turing machine (TM) or a random-access machine (RAM).

Besides being interesting from a purely complexity-theoretic perspective, such *succinct randomized encodings* may have powerful applications analogous to those of the known randomized encodings. One such immediate application is private delegation of computation: a weak client that wishes to use the aid of a server to run a long computation Π on a short private input x, may quickly compute a succinct randomized encoding $\widehat{\Pi}(x)$, and have the server decode the result $\Pi(x)$, without the server learning anything regarding x (with a little more effort, we can even ensure privacy of the output, and be able to verify that the server computed correctly).

Beyond shifting computation from weak parties to strong parties, succinct randomized encodings may sometimes save in communication and computation altogether. For instance, one of the first demonstrated applications of randomized encodings [101, 103] was to achieve such savings in multi-party com-

putation (MPC). Indeed, most known MPC solutions explicitly utilize the circuit $C_f(x_1, \ldots, x_m)$ representing a function $f(x_1, \ldots, x_m)$, and the overhead they incur, e.g. in communication, may depend on the circuit size $|C|$. When the function f is succinctly represented by a program Π, we may have the parties compute first a succinct randomized $\widehat{\Pi}(x_1, \ldots, x_m)$, and only decode at the end, thereby making communication overhead proportional to the smaller circuit that computes $\widehat{\Pi}$. Furthermore, the effort of decoding (proportional to Π's running time) falls only on the parties that obtain the output. If the overhead of decoding is small, it may reduce the computational complexity of the MPC protocol as well. (For instance, now only a single party, rather than each one of the parties, has to invest resources proportional to the running time of f.)

Do Succinct Randomized Encodings Exist? Under commonly believed complexity-theoretic assumptions, perfectly-private randomized encodings for all of $\mathcal{P}$ are unlikely to be computable too fast, e.g. in fixed polynomial time.[1] In contrast, restricting attention to privacy against computationally-bounded adversaries, no lower bounds or barriers are known. In fact, succinct indistinguishability obfuscation ($i\mathcal{O}$) for any model of computation (e.g., $i\mathcal{O}$ for Turing machines) would directly imply corresponding succinct randomized encodings.[2] Still, constructions of succinct $i\mathcal{O}$ [43, 1], or direct constructions of succinct randomized encodings [90, 83] are based on considerably strong computational

[1] Specifically, it can be shown that, for a language L, recognized by a given $T(n)$-time Turing machine Π, succinct randomized encodings with perfect-privacy computable in time $t(n) \ll T(n)$, would imply that L has 2-message interactive proofs with a $O(t(n))$-time verifier, which already suggests that $t(n)$ should at least depend on the space (or depth) of the computation. Furthermore, under commonly believed derandomization assumptions (used to show that $\mathsf{AM} \subseteq \mathsf{NP}$ [109, 121]), the above would imply that L can be non-deterministically decided in time $\mathrm{poly}(t(n))$, for some fixed polynomial poly. Thus, any speedup in encoding would imply related speedup by non-determinism, whereas significant speedup is believed to be unlikely.

[2] To encode (Π, x) simply obfuscate a program that given no input computes $\Pi(x)$. This can be simulated from $y = \Pi(x)$ by obfuscating a program that only performs dummy steps and outputs y.

assumptions such as extractable witness encryption, succinct non-interactive arguments, and differing-inputs obfuscation. In the language of [125] these assumptions are not *efficiently-falsifiable*; furthermore, in some cases they have been shown implausible [45, 29, 73].

2.1.1 Contributions

Our core contribution is a succinct randomized encoding relying on (non-succinct) $i\mathcal{O}$ for *circuits*, for any class of *a-priori bounded-space computations*. That is, the time to encode depends on the space complexity of the computation, but is essentially independent of its time complexity. The construction, in fact, satisfies the enhanced guarantee of *a succinct garbling schemes* [138, 11, 22], with the extra feature that inputs can be encoded independently of the program and its complexity.

Theorem 5 (Main Theorem, Informally Stated)**.** *Assume the existence of $i\mathcal{O}$ for $\mathsf{P/poly}$ and one-way functions. Then, for every polynomial $s(\cdot)$, there exists a succinct randomized encoding (or garbling scheme) for all polynomial-time programs Π with space-complexity $S(n) \leq s(n)$. Specifically, the time to encode depends polynomially on the size of Π, the lengths (n, m) of its input and output, and the space bound $s(n)$, but only polylogarithmically on Π's running-time.*

On the Underlying Assumption: Assuming puncturable pseudo-random functions in NC^1 (known based on various hardness assumptions, such as the hardness of the learning with errors problem [36]), and restricting attention to any class of computations with a-priori-bounded running time $t(n)$, we can settle for $i\mathcal{O}$ for circuits in NC^1 with input size $O(\log(t(n))$ (which is a $\mathrm{poly}(t(n))$-time falsi-

fiable assumption on its own). Obtaining $i\mathcal{O}$ for this class may be done based on qualitatively weaker assumptions; indeed, for any polynomial $t(\cdot)$ the construction of Gentry et al. [84] would imply $i\mathcal{O}$ for the corresponding class based on a polynomial hardness assumption on multi-linear maps.[3]

We then demonstrate the power of succinct randomized encodings in several applications, some new, and some analogous to previous applications of randomized encodings, but with new succinctness properties.

Application 1: Succinct Indistinguishability Obfuscation. Our first (and somewhat strongest) application of succinct randomized encodings is succinct $i\mathcal{O}$ for bounded-space computations. Indistinguishability here means that the (succinct) obfuscations of two programs that have the same output and running time on all inputs inp of some apriori-bounded length n are computationally indistinguishable. The construction is based on subexponential $i\mathcal{O}$, whereas any form of succinct $i\mathcal{O}$ realized so far [1, 43, 104] relies on differing-inputs obfuscation in conjunction with succinct non-interactive arguments (which already entail strong succinctness properties); as mentioned before, these are considered very strong up to implausible in certain settings.

Theorem 6 (Informally Stated)**.** *Assume the existence of succinct randomized encodings for space-bounded programs, one-way functions, and $i\mathcal{O}$ for $\mathsf{P/poly}$ that are all subexponentially-secure. Then, for every polynomial $s(\cdot)$, there exists a succinct $i\mathcal{O}$ for all polynomial-time programs Π with space-complexity $S(n) \leq s(n)$, Specifically, the time to obfuscate Π depends polynomially on the size of Π, the input length n, and the space bound $s(n)$, but only polylogarithmically on Π's running-time and output length*

[3] More generally, one of the challenges in basing $i\mathcal{O}$ on an efficient black-box reduction is that the reduction may have to exhaust the input space to check if the challenge circuits are functionally equivalent. In the above case, this can be done in time $\mathrm{poly}(t(n))$.

m.

The theorem is somewhat the succinct analog of previous bootstrapping theorems [7, 56] who show how (non-succinct) randomized encodings and pseudo-random functions in NC^1, together with obfuscation for NC^1 circuits, imply obfuscation for $\mathsf{P/poly}$. Here, through succinct randomized encodings, we reduce $i\mathcal{O}$ for arbitrarily long computations to $i\mathcal{O}$ for circuits of fixed polynomial size.

Application 2: Succinct Functional Encryption and Reusable Garbling. The recent leap in the study of obfuscation has brought with it a corresponding leap in functional encryption (FE). Today, (indistinguishability-based) functional encryption for all circuits can be constructed from IO [71, 136], or even from concrete (and efficiently falsifiable) assumptions on composite order multilinear graded-encodings [74]. For models of computation with succinct representations, we may hope to have *succinct FE*, where a secret key sk_Π, allowing to decryption $\Pi(x)$ from an encryption of x, can be computed faster than the running time of Π. However, here the state-of-art was similar to succinct randomized encodings, or succinct $i\mathcal{O}$, requiring essentially the same strong (non-falsifiable) assumptions.

One can replace $i\mathcal{O}$ for circuits, in the above FE constructions, with the succinct $i\mathcal{O}$ from Theorem 6, and obtain FE where computing sk_Π is comparable to (succinctly) obfuscating Π. This, however, will require the same sub-exponential hardness of $i\mathcal{O}$ for circuits. Based on existing non-succinct functional encryption schemes, we show that succinct FE can be constructed without relying on sub-exponentially hard primitives.

Theorem 7 (Informally Stated)**.** *Assume the existence of succinct randomized encod-*

ings for space-bounded programs, one-way functions, and iO for $\mathsf{P/poly}$. *Then, for every polynomial* $s(\cdot)$, *there exists a succinct FE where a functional key* sk_Π *could be generated for any polynomial-time program* Π *with space-complexity* $S(n) \leq s(n)$, *and can decrypt encryption of messages of apriori-bounded length. The time to derive* sk_Π *depends polynomially on the size of* Π, *the input and output lengths* (n, m), *and space bound* $s(n)$, *but only polylogarithmically on* Π*'s running-time.*

The scheme is selectively-secure. Assuming also puncturable pseudo-random functions in NC^1, *and the same assumptions on multi-linear maps made in [74], results in full (adaptive) security.*

As observed in previous work [83, 58, 91], FE (even indistinguishability-based) directly implies an enhanced version of randomized encodings known as *reusable garbling*. Here reusability means that an encoding consists of two parts: The first part $\widehat{\Pi}$ is independent of any specific input, and only depends on the machine Π. $\widehat{\Pi}$ can then be "reused" together with a second part $\widehat{\mathsf{inp}}$ encoding any input inp. We get succinct reusable garbling for space-bounded computations: encoding Π depends on the space, but is done once, subsequent input-encodings depend only on the input size n and not on space.

Application 3: Publicly Verifiable Delegation and succinct NIZKs. Succinct randomized encodings directly imply a one-round delegation scheme for polynomial-time computations with bounded space complexity. A main feature of the scheme is *public-verifiability*, meaning that given the verifier's message σ anyone can verify the proof π from the prover, without requiring any secret verification state. Previous publicly-verifiable schemes relied on strong knowledge assumptions [94, 25, 63, 26] or proven secure only in the random oracle

model [120].[4] Another prominent feature of the scheme is that it guarantees input privacy for the verifier. (While this can generically be guaranteed with fully homomorphic encryption, the generic solution requires the prover to convert the computation into a circuit, which could incur quadratic blowup; in our solution, the complexity of the prover corresponds to decoding complexity, which could be made quasi-linear. See further discussion below.)

The delegation scheme is based only on randomized encodings (and one-way functions), and thus as explained above, can be based only on polynomial assumptions. Assuming also iO , we can make the verifier's message reusable; namely, the verifier can publish his message σ once and for all, and then get non-interactive proofs for multiple computations.[5]

Theorem 8 (Informally Stated)**.**

1. *Assume the existence of succinct randomized encodings for space-bounded programs and one-way functions. Then, there exists a publicly-verifiable 2-message delegation scheme with input privacy where verifying a computation given by a program* Π *and input* x*, is polynomial in the size of* Π*, input length and output lengths* (n, m)*, and the space* S *required to compute* $\Pi(x)$*, but only polylogarithmic in* Π*'s running-time.*

2. *Assuming also* iO *for* $\mathsf{P/poly}$*, the verifier message* σ *is made reusable for computations with a-priori bounded space* $s(n)$*. Furthermore, only the one-time generation of* σ *depends on* $s(n)$*, whereas subsequent verification depends only on the*

[4]Notably, in the setting of private-verification Kalai, Raz, and Rothblum give a solution based on the subexponential learning with errors assumption [106].

[5]Our transformation for reusing the verifier's message is, in fact, a generic one that can be applied to any delegation scheme, including privately-verifiable schemes (e.g., [106]). For privately-verifiable schemes, the transformation has an additional advantage: it removes what is known as *the verifier rejection problem*; specifically, in the transformed scheme, soundness holds even against provers with a verification oracle.

input size n (and the security parameter).

Plugging in our succinct iO into the perfect non-interactive zero-knowledge (NIZK) arguments of Sahai-Waters [134] directly yields a construction of perfect succinct NIZK for bounded-space NP from iO for P/poly and one-way functions that are both sub-exponentially-secure. The NIZK has a succinct common reference string whose size is independent of the time required to verify the NP statement to be proven, and only depends on the space, and the size of the input and witness (verification time depends only on the length of the statement as in [134]).

iO for NC^1 is enough: We note that in all three theorems above, the assumption of iO for P/poly can replaced with assuming iO for NC^1 and puncturable pseudorandom functions in NC^1. Indeed, in the above applications the obfuscated circuit is dominated by computing a succinct randomized encoding and a puncturable PRF. Here we can rely on the observation that randomized encodings can be composed [11]. Concretely, we can consider an outer layer of a non-succinct shallow randomized encoding (like Yao [138]) that computes an inner succinct randomized encoding.

Other Applications. We reinspect additional previous applications of (non-succinct) randomized encodings and note the resulting succinctness features.

One application, briefly mentioned above, is to multiparty computation [101, 103], where we can reduce the communication overhead from depending on the circuit size required to compute a multiparty function $f(x_1, \ldots, x_m)$ to depending on the space required to compute f, which can be much smaller. When focusing merely on communication this problem has by now general one-round

solutions based on (multi-key) fully-homomorphic encryption [80, 14, 117, 68]. Succinct randomized encodings allow in addition to shift the work load to one party (the decoder) that obtains the output, without inducing extra rounds. (With one extra message, outputs to weak parties can also be delivered, while guaranteeing their privacy and correctness.)

Another application is to amplification of *key-dependent message security* (KDM). In KDM encryption schemes, semantic security needs to hold, even when the adversary obtains encryptions of functions of the secret key taken from a certain class F. Applebaum [5] shows that any scheme that is KDM-secure with respect to some class of functions F can be made resilient to a bigger class $\mathsf{F}' \supseteq \mathsf{F}$, if functions in F' can be randomly encoded in F. Our succinct randomized encodings will essentially imply that KDM-security for circuits of any fixed polynomial size $s(\cdot)$ (such as the scheme of [18]) can be amplified to KDM-security for functions that can be computed by programs with space $S \ll s(n)$, but could potentially have larger running time.

Dependence on the output length. As stated above, the size of our basic randomized encodings grows with the output of the underlying computation. Such dependence can be easily shown to be inherent as long as we require simulation-based security (using a standard incompressibility argument). Nevertheless, this dependence can be removed if we settle for a weaker indistinguishability-based guarantee saying that randomized encodings of two computations leading to the same output are indistinguishable. This guarantee, in fact, suffices, and allows removing output-dependence, in all of the applications above, except for the multi-party application (which requires simulation on its own).

Optimizing Decoding Time. While we have so far concentrated on how fast

can a randomized encoding be computed, one may also be interested in optimizing the time and space complexity of decoding. Ideally the complexity of decoding should be as close as possible to that of the original computation. In our basic scheme, decoding $\widehat{\Pi}(x)$ of a T-time S-space computation $\Pi(x)$, where S is a-priori bounded by some polynomial $s(n)$, requires roughly time $T \cdot \mathrm{poly}(s(n))$ and space $\mathrm{poly}(s(n))$, while encoding takes only time $\mathrm{poly}(s(n))$ (up to polynomial factors in the security parameter). This complexity is naturally inherited by all our applications of randomized encodings: for instance, the time to obfuscate a program Π is roughly $\mathrm{poly}(s(n))$, and the time to evaluate the obfuscation (given by Theorem 6) on an input x is proportional to the decoding time for $\widehat{\Pi}(x)$.

We show how to optimize our randomized encodings to improve decoding time to roughly $T + s(n)$. This optimization further reduces the encoding time from $\mathrm{poly}(s(n))$ to $\tilde{O}(s(n))$.

Proposition 1 (Improved Efficiency, Informally Stated)**.** *Assume the existence of $i\mathcal{O}$ for $\mathsf{P/poly}$ and one-way functions. Then, for every polynomial $s(\cdot)$, there exists a succinct randomized encoding (or garbling scheme) for all polynomial-time RAM Π with space-complexity $S(n) \leq s(n)$. Specifically, the time to encode is* **quasi-linear** *in the size of Π, input length n, and the space bound $s(n)$. The time to decode $\widehat{\Pi}(x)$ is polynomial in the size of Π, and* **quasi-linear** *in the space bound $s(n)$ and the time T for evaluating $\Pi(x)$.*

The improvement in encoding and decoding efficiency leads to improved efficiency for our applications of succinct randomized encoding. For instance, we obtain a succinct $i\mathcal{O}$ for bounded space RAM that takes time roughly $s(n)$ to obfuscate, and $T + s(n)$ to evaluate. Other applications such as FE, delegation, MPC directly inherit the improved decoding complexity (leading to better

decryption time, prover efficiency, and computational complexity respectively).

We note that the above efficiency optimizations are inspired by a concurrent work of Canetti, Holmgren, Jain, and Vaikuntanathan [54], who constructed succinct $i\mathcal{O}$ for bounded space RAM, where evaluation takes time roughly $T + s(n)$. We investigated these optimizations after being made aware that they achieve this feature.

2.1.2 Techniques

We next overview our construction of succinct randomized encodings for bounded space programs. Beyond $i\mathcal{O}$, the main tool on which we rely is existing *non-succinct* randomized encodings, or more accurately their enhanced version of *garbling schemes*. As mentioned before, garbling schemes have the extra feature that the input x can be encoded separately of the program Π given a shared (short) string **key** [138, 22]. When considering (non-succinct) garbling, e.g. where Π is a circuit, a salient advantage of this separation is that the time to compute the encoded $\widehat{x}$ depends on the length of x, but not on the typically larger running time (or circuit size) of Π. In contrast, the time to compute the encoded $\widehat{\Pi}$ may be as large as its running time. This feature of "independent input encoding" is crucial for our construction.

We construct succinct randomized encodings, or in fact, succinct garbling schemes, in two steps: we first construct a non-succinct garbling scheme for bounded-space computations, with the property that the garbled program consists of many "small garbled pieces" that can be generated separately. In the second step, we use $i\mathcal{O}$ to "compress" the size of the garbled program, by pro-

viding an obfuscated program that takes an index as input and generates the "garbled piece" corresponding to that index. As a result, the final garbled program (namely the obfuscated program) is small and can be efficiently computed. It is only at evaluation time that the underlying non-succinct garbled program is unravelled, by running the obfuscated program on every index, and decoded.

The Non-succinct Garbling Scheme. We outline the *non succinct garbling scheme* for bounded computations, based on any one-way function. For concreteness, we shall focus on Turing machines. (The solution extends to any model of bounded-space computation, e.g. RAM, as long as a computation can be decomposed into a sequence of steps operating on one memory.)

A "trivial" approach towards such garbling is to simply transform any polynomial-time Turing machine into a circuit and then garble the circuit. While our construction in essence relies on this principle, it will in fact invoke garbling for "small" fixed-sized circuits. Concretely, we rely on the existence of a circuit garbling scheme satisfying two additional properties. First, we require that the shared string **key**, and thus also the input encoding, are generated independently of the circuit to be garbled (e.g., **key** is sampled at random and given to both the input-encoding and circuit-garbling procedures). Second, we require that encoded inputs can be simulated, given only the input size, whereas the garbled program is simulated using the result $\Pi(x)$ of the computation (and the randomness used to simulate the encoded input). We refer to such schemes as *garbling schemes with independent input encoding* and note that Yao's basic scheme [138] satisfies the two properties.

Our non-succinct garbling scheme now proceeds as follows. Let Π be a Turing machine with bounded space complexity $s(\cdot)$, running-time $t(\cdot)$, and inputs

of length n. We construct a "chain" of $t(n)$ garbled circuits that evaluate Π step by step. More precisely, we first generate keys $\mathbf{key}_1, \ldots, \mathbf{key}_{t(n)}$ for the $t(n)$ garbled circuits. The j^{th} garbled circuit (which is computed using key $\mathbf{key}_j$) takes as input some state of Π and computes the next state (ie., the state after one computation step); if the next state is a final state, it returns the output generated by Π, otherwise it outputs an *encoding* of this new state using the next key $\mathbf{key}_{j+1}$. (Note that after $t(n)$ steps we are guaranteed to get to a final state and thus this process is well-defined.)

To encode the input, we simply encode the initial state of Π, including the input x, using $\mathbf{key}_1$. To evaluate the garbled program, we sequentially evaluate each garbled circuit, using the encodings generated in the previous one as inputs to the next one, and so on until the output is generated.

Security of the Non-Succinct Scheme: an Overview. To show that this construction is a secure (non-succinct) garbling scheme we need to exhibit a simulator that, given just the output $y = \Pi(x)$ of the program Π on input x and *the number of steps t^* taken by* $\Pi(x)$, can simulate the encoded input and program. (The reason we provide the simulation with the number of steps t^* is that we desire a garbling scheme with a "per-instance efficiency"—that is, the evaluation time is polynomial in the actual running-time t^* and not just the worst-case running-time. To achieve such "per-instance efficiency" requires leaking the running-time, which is why the simulator gets access to it.) Towards this, we start by simulating the $t^{*\text{th}}$ garbled circuit with the output being set to y; this simulation generates an encoded input $\widetilde{\text{conf}}_{t^*-1}$ and a garbled program $\tilde{\Pi}_{t^*}$.

We then iteratively in descending order simulate the j^{th} garbled circuits $\tilde{\Pi}_j$ with the output being set to $\widetilde{\text{conf}}_{j+1}$ generated in the previously simulated gar-

bled circuit. We finally simulate the remaining $j > t^*$ garbled circuits $\tilde{\Pi}_j$ with the output being set to some arbitrary output in the range of the circuit (e.g., the output y). The simulated encoded input is then $\widetilde{\text{conf}}_1$ and the simulated garbled program is $(\tilde{\Pi}_1, \ldots \tilde{\Pi}_{t(n)})$.[6]

To prove indistinguishability of the simulated garbling and the real garbling, we consider a sequence of hybrid experiments $H_0, \ldots, H_{t(n)}$, where in H_j the first j garbled circuits are simulated, and the remaining $t(n) - j$ garbled circuits are honestly generated. To "stitch together" the simulated circuits with the honestly generated ones, the j^{th} garbled circuit is simulated using as output an honest encoding $\widehat{\text{conf}}_j$ of the actual configuration conf_j of the TM Π after j steps.

It follows from the security of the garbling scheme that hybrids H_j and H_{j+1} are indistinguishable and thus also H_0 (i.e., the real experiment) and $H_{t(n)}$.

Let us finally note a useful property of the above-mentioned simulation. Due to the fact that we rely on a garbling scheme with *independent input encoding*, each garbled circuit can in fact be *independently simulated*—recall that the independent input encoding property guarantees that encoded inputs can be simulated without knowledge of the circuit to be computed and thus all simulated encoded inputs $\widetilde{\text{conf}}_1, \ldots \widetilde{\text{conf}}_{t(n)}$ can be generated in an initial step. Next, the garbled circuits can be simulated in any order.

The Succinct Garbling Scheme: an Overview. We now show how to make this garbling scheme succinct. The idea is simple: instead of providing the actual garbled circuits in the clear, we provide an obfuscation of the *randomized* program that generates these garbled circuits. More precisely, we provide an

[6]This "layered" simulation strategy resembles that of Applebaum, Ishai, and Kushilevitz in the context of arithmetic garbling [13].

$i\mathcal{O}$ of a program $\mathbf{\Pi}^{s,s'}(\cdot)$ where s and s' are seeds for a PRF F: $\mathbf{\Pi}^{s,s'}(j)$, given a "time-step" $j \in [t(n)]$, generates the j^{th} garbled circuit in the non-succinct garbling of Π using pseudo-random coins generated by the PRF with seed s and s'. Specifically, it uses $\mathsf{F}(s, j)$ and $\mathsf{F}(s, j+1)$ as randomness to generate $\mathbf{key}_j$ and $\mathbf{key}_{j+1}$ (recall that the functionality of the j^{th} circuit depends on $\mathbf{key}_{j+1}$), and uses $\mathsf{F}(s', j)$ as randomness for garbling the j^{th} circuit.

Now, the new succinct garbled program is the obfuscated program $\mathbf{\Lambda} \xleftarrow{\$} i\mathcal{O}(\mathbf{\Pi}^{s,s'})$, and the encoding $\hat{x}$ of x remains the same as before, except that now it is generated using pseudo-random coins $\mathsf{F}(s, 1)$. Given such a garbled pair $\mathbf{\Lambda}$ and $\hat{x}$, one can compute the output by gradually generating the non-succinct garbled program, *one garbled circuit at a time*, by computing $\mathbf{\Lambda}$ on every time step j, and evaluating the produced garbled circuit with $\hat{x}$ until the output is produced. (This way, the evaluation still has "per-instance efficiency".)

Security of the Succinct Scheme: an Overview. Given that the new succinct garbled program $\mathbf{\Lambda}$ produces "pieces" of the non-succinct garbled program, the natural idea for simulating the succinct garbled program is to obfuscate a program that produces "pieces" of the simulated non-succinct garbled program. The above-mentioned "independent simulation" property of the non-succinct garbling (following from independent input encoding) enables to fulfill this idea.

More precisely, given an output y and the running-time t^* of $\Pi(x)$, the simulator outputs the obfuscation $\widetilde{\mathbf{\Lambda}}$ of a program $\widetilde{\mathbf{\Pi}}^{y,t^*,s,s'}$ that, given input j, outputs a simulated j^{th} garbled circuit, using randomness $\mathsf{F}(s, j+1)$ to generate $\widetilde{\text{conf}}_{j+1}$ as the output, and $\mathsf{F}(s, j)$ and $\mathsf{F}(s', j)$ as the extra randomness needed to simulate

the input $\widetilde{\mathsf{conf}}_j$ and the garbled Π_j. [7] The encoded input $\tilde{x}$ is simulated as in the non-succinct garbling scheme, but using pseudo-random coins $\mathsf{F}(s, 1)$.

It is not hard to see that this simulation works if the obfuscation is virtually black-box secure, as (non-succinct) garbling security guarantees that the entire truth tables of the two programs $\mathbf{\Pi}^{s,s'}$ and $\widetilde{\mathbf{\Pi}}^{y,t^*,s,s'}$ are indistinguishable given an encoding of x, when the hardwired PRF keys s, s' are chosen at random. Our goal, however, is to show that $i\mathcal{O}$ suffices. Towards this goal, we consider a sequence of hybrid experiments $H'_0, \ldots, H'_{t(n)}$ with a corresponding sequence of obfuscated programs $\widetilde{\mathbf{\Pi}}_0^{s,s'}, \ldots, \widetilde{\mathbf{\Pi}}_{t(n)}^{s,s'}$ that "morph" gradually from the real $\mathbf{\Pi}$ to the fully simulated $\widetilde{\mathbf{\Pi}}$. Specifically, the program $\widetilde{\mathbf{\Pi}}_j^{s,s'}$ obfuscated in H'_j produces a non-succinct *hybrid* garbled program as in hybrid H_j in the proof of the non-succinct garbling scheme, except that pseudo-random coins are used instead of truly random coins. That is, for the first j inputs, $\widetilde{\mathbf{\Pi}}_j$ produces simulated garbled circuits, and for the rest of the inputs, it produces honestly generated garbled circuits, having hardwired the true configuration conf_{j+1}.

To prove indistinguishability of any two consecutive hybrids H'_j and H'_{j+1}, we rely on the punctured program technique of Sahai and Waters [134] to replace pseudo-random coins $\mathsf{F}(s, j+1)$, $\mathsf{F}(s', j+1)$ for generating the $j+1^{\text{st}}$ simulated garbled circuit with truly random coins, and then rely on the indistinguishability of the simulation of the $j+1^{\text{st}}$ garbled circuit. A bit more concretely, at each step we puncture the seeds s, s' only on the (three) points corresponding to the $j+1^{\text{st}}$ step, and hardwire instead the corresponding outputs generated by $\widetilde{\mathbf{\Pi}}_j^{s,s'}$; next, relying on the puncturing guarantee, we can sample these outputs using true independent randomness. At his point, we can already replace the real hardwired

[7] Recall that simulating a garbled circuit requires both the output and the randomness for simulating the input encoding.

garbling with a simulated one. Finally, we go back to generating the hardwired value pseudorandomly as part of the circuit's logic, now identical to $\widetilde{\mathbf{\Pi}}^{s,s'}_{j+1}$, and "unpuncture" the seeds s, s'. We note that each such step requires hardwiring a new (real) intermediate configuration conf_{j+1} (used to simulate the $j+1^{\text{st}}$ garbling), but now the previous hardwired configuration conf_j can be "forgotten" and blowup is avoided.

***iO* for a Simple Class of Circuits is Enough.** The obfuscated circuits in the construction are of a special kind—their input size is $O(\log t(n))$. Canetti et al. [56] show that *iO* for NC^1 can be bootstrapped to obtain *iO* for all circuits, assuming puncturable PRFs in NC^1 [36], and incurring a security loss that is exponential in the size of the input. Accordingly, for polynomial $t(n)$, it suffices to assume (polynomially-secure) *iO* for classes in NC^1 with logarithmic-size inputs.

Generalizing and Optimizing. The solution described above does not apply uniquely to Turing machines, but rather to any model of computation that can be divided into sequential steps using one memory, for instance random access machines (RAMs). Thus it directly gives a succinct garbling scheme for bounded space RAMs.

Also note that, in the described solution, we can in fact replace the underlying circuit garbling scheme, with any garbling scheme, as long as it admits independent input encoding. For instance, in the case the program Π is a RAM, we may use previous *garbled RAM* solutions [118, 82, 76]. The benefit is that this allows optimizing the efficiency of our scheme. Indeed, in the solution described above, each step of the machine is translated to a garbled circuit of size $O(s(n))$ (up to polynomial factors in the security parameter), which means that

the complexity of encoding is $\text{poly}_{iO}(s(n))$, where $\text{poly}_{iO}(\cdot)$ is the overhead due to obfuscation, and the complexity of decoding for a T-time computation $\Pi(x)$ is at least $T \cdot \text{poly}_{iO}(s(n))$, which may be significantly larger than the original computation.

In contrast, known garbled RAM solutions provide a more efficient way of garbling RAMs than converting them into circuits, taking into consideration the RAM structure, and guaranteeing that encoding and decoding require essentially the same time and space as the original RAM computation. Aiming to leverage this efficiency in our solution, instead of partitioning a RAM computation into $t(n)$ steps, each implemented by a circuit of size $s(n)$, we can partition it to $t(n)/s(n)$ pieces, where each piece is an $s(n)$-step RAM. The encoding and decoding time for each piece are essentially linear in its running time $O(s(n))$ (whereas a circuit implementing any such piece might be of size $\Omega(s(n)^2)$).

This modification on its own may still be insufficient; indeed, obfuscating the circuit that produces the garbled RAM may incur non-linear overhead $\text{poly}_i O(\cdot)$, so that eventually decoding may take time $\text{poly}_i O(s(n)) \cdot t(n)/s(n)$ which may be again as large as $t(n) \cdot s(n)$.

To circumvent this blowup, and as a result of independent interest, we show how to bootstrap any iO for circuits to one that has quasi-linear blowup. Overall, in the new solution, for a T-time S-space computation computation $\Pi(x)$ where $S < s(n)$, encoding takes time $\tilde{O}(s(n))$ and decoding $\widehat{\Pi}(x)$ takes time $O(T + s(n))$.

Main Ideas behind the Applications

We briefly sketch the main ideas behind our main applications of succinct randomized encodings.

Succinct *iO* . The construction of succinct *iO* from randomized encoding and exponential *iO* for circuits is a natural instantiation of the bootstrapping approach suggested by Applebaum [7]. There, the goal is to reduce obfuscation of general circuits to obfuscation of NC^1 circuits; our goal is to reduce obfuscation of programs with large running time (but bounded space) to obfuscation of significantly smaller circuits. To obfuscate a succinct program Π with respect to inputs of size at most n, we obfuscate a small circuit $C^{\Pi,K}$ that has a hardwired seed K for a PRF, and given input inp, applies the PRF to inp to derive randomness, and then computes a succinct randomized encoding of $\widehat{\Pi}(\mathsf{inp})$. The obfuscated $iO(\Pi)$, given input inp computes the encoding, decodes it, and returns the result.

The analysis in [7] establishes security in case that the circuit obfuscator *iO* is virtually black-box secure. We show that if *iO* has $2^{-\lambda^\varepsilon}$-security for security parameter $\lambda \gg n^\varepsilon$, and the PRF is puncturable that is also $2^{-\lambda^\varepsilon}$-secure, then a similar result holds for *iO* (rather than virtual black-box). The proof is based on a general *probabilistic iO* argument, an abstraction recently made by Canetti et al. [56].

Succinct FE. The construction of succinct functional encryption follows rather directly by plugging in our randomized encodings into previous constructions of non-succinct functional encryption. Concretely, starting with the scheme of Gentry et al. [83], we can replace the non-succinct randomized encodings for RAM in their construction with our succinct randomized encodings, and ob-

tain selectively-secure FE.[8] Alternatively, starting from the scheme of Garg et al. [74], we can replace randomized encodings for circuits in their construction with our succinct randomized encodings, and get an adaptively secure succinct FE scheme. (Here we also need to rely on the fact that succinct randomized encodings can be computed in low depth, which is required in their construction.) We note that in both cases, our succinct randomized encodings already satisfy the required security for their security proof to go through, and only the succinctness features change.

Publicly-Verifiable Delegation. Finally, we sketch the basic ideas behind the delegation scheme. The delegation scheme is pretty simple and similar in spirit to previous delegation schemes (in a weaker processing model) [12, 77, 127, 91]. To delegate a computation, given by Π and inp, the verifier simply sends the prover a randomized encoding $\widehat{\Pi'}(\mathsf{inp}, r)$, where Π' is a machine that returns r if and only if it accepts inp, and r is a sufficiently long random string. The security of the randomized encoding implies that the prover learns nothing of r, unless the computation is accepting. The scheme can be easily made publicly verifiable by publishing $f(r)$ for some one-way function f. Furthermore, the scheme ensures input-privacy for the verifier.

We then propose a simple transformation that can be applied to any delegation scheme in order to make the first verifier message reusable. The idea is natural: we let the verifier's first message be an obfuscation of a circuit C^K that has a hardwired key K for a puncturable PRF, and given a computation (Π, inp), applies the PRF to derive randomness, and generates a first message for the del-

[8] Formally, their construction is given in terms of garbling for RAM rather than randomized encodings, but these are actually used as randomized encodings, without making special use of independent input encoding.

egation scheme. Thus, for each new computation, a first message is effectively sampled afresh. Relying on iO and the security of the puncturable PRF, we can show that (non-adaptive) soundness is guaranteed. The transformation can also be applied to privately-verifiable delegation schemes, such as the one of [106] and maintains soundness, even if the prover has a verification oracle.

2.1.3 Concurrent and Subsequent Work

In concurrent work, Canetti, Holmgren, Jain, and Vaikuntanathan construct succinct iO for RAMs assuming subexponentially-secure iO for $\mathsf{P/poly}$. The complexity of their succinct iO is also such that obfuscation depends on an a-priori bound on space, but not on the running time. This, in particular, implies a succinct randomized encoding with similar parameters.

The technique that they employ is quite different from ours, and requires stronger computational assumptions. Their main step is also the construction of a succinct garbling scheme for RAMs; however, their succinct garbling scheme is very different. At a high-level, in our solution, the obfuscation is only responsible for garbling (or encoding); the evaluation of the garbled components (or decoding) is done "externally" by the evaluator; encoding and decoding themselves are implemented using existing garbling schemes. In their solution, the obfuscation deals not only with encoding, but also with decoding, getting as input at every step the encrypted and authenticated current state of the computation. They implement this mechanism by designing a primitive that they call *Asymmetrically Constrained Encapsulation*, in a careful combination with an oblivious RAM scheme. (In our basic solution, oblivious RAMs are not needed

as we rely on garbling for circuits, which are already an oblivious model of computation, but an inefficient one that touches all of the state in every step. In our optimizations, the use of oblivious RAM is abstracted by the underlying garbled RAMs, which are indeed implemented in [118, 82, 76] using oblivious RAMs.)

A disadvantage of their approach is that the circuit deals with inputs of size proportional to the security parameter (due to encryption and authentication of state bits), whereas in our case the circuit just takes a logarithmic size index (representing a time point in the computation); as discussed above, iO for logarithmic length input seems to be a weaker assumption (in particular, it is falsifiable), and can be based on polynomial assumptions on multilinear maps. On the other hand, performing the entire evaluation "inside the obfuscation" as in their approach would eventually lead to a fully succinct solution in subsequent work (see below).

Full Succinctness. At first glance, our approach seems to suggest a natural way to achieve full succinctness, without any dependence on space. Instead of garbling a sequence of transition circuits, we can garble each gate in the circuit representation of the computation separately; indeed, the circuit corresponding to the computation can be succinctly represented by a small circuit that can output each gate and its corresponding neighbours. More accurately, as in the previous solution, we will garbled an augmented gate that encodes the output under the keys corresponding to its (constant number of) neighbours (towards the output gate). Again, garbling will be derandomized using a pseudo-random function.

This approach will, in fact, give a fully succinct garbling scheme if we assume virtual black-box security for the above "gate garbler", as once again the truth tables of a real and a simulated garbling will be computationally indistin-

guishable. However, assuming $i\mathcal{O}$ it is not clear how to achieve any advantage over the previous solution. Intuitively, whenever we invoke $i\mathcal{O}$ we cannot "forget" an intermediate value in the computation, before all the connected gates in the layer above are simulated (inducing new values to remember). In the worst-case, we are forced to remember an entire configuration.

In a beautiful subsequent work, Koppula, Lewko, and Waters [113] construct fully-succinct randomized encodings from $i\mathcal{O}$. Their solution takes a similar route to that of Canetti et al. [54] in that each step of the computation is done "under the obfuscation". To overcome the space barrier, they introduce a clever "selective enforcement mechanism" that allows avoiding storage of the entire state, by storing a special purpose succinct commitment. In the analysis, this commitment can be indistinguishably replaced with a commitment that statistically binds some selected location in the memory corresponding to a given step of the computation, and is thus "$i\mathcal{O}$ -friendly" in their terminology.

Organization In Section 2.2, we provide preliminaries, including: different models of computation considered in the paper, definitions of garbling schemes and $i\mathcal{O}$ with different efficiency levels. In Section 2.3, we construct succinct garbling schemes for bounded space Turing machines. We then generalize this construction to any model of bounded space computation, in particular, RAM, and optimize the decoding efficiency in Section 2.4. Finally, in Section 2.5, we present applications of succinct randomized encodings to succint $i\mathcal{O}$ and delegation; we omit details for other applications that are achieved by directly plugging in randomized encodings in previous works. In Appendix 2.6, we show how to bootstrap any circuit $i\mathcal{O}$ to one with quasi-linear blowup.

2.2 Preliminaries

Let $\mathcal{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \ldots, n\}$. We denote by PPT probabilistic polynomial time Turing machines. The term negligible is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from non-negative integers to reals is called negligible if for every constant $c > 0$ and all sufficiently large n, it holds that $\nu(n) < n^{-c}$.

2.2.1 Models of Computation

In this work we will consider different models of computation. Below we define formally different classes of algorithms; we will start by defining classes of deterministic algorithms of fixed polynomial size, and then move to define classes of randomized algorithms and classes of algorithms of arbitrary polynomial size.

Classes of deterministic algorithms of fixed polynomial size.

Polynomial-time Circuits. For every polynomial D, the class $\mathsf{CIR}[D] = \{C_\lambda\}$ of include all deterministic circuits of size at most $D(\lambda)$.

NC^1 **Circuits.** For every constant c and polynomial D, the class $\mathsf{NC}_c[D] = \{C_\lambda\}$ of polynomial-sized circuits of depth $c \log \lambda$ include all deterministic circuits of size $D(\lambda)$ and depth at most $c \log \lambda$.

Exponential-time Turing Machines. We consider a canonical representation of Turing machines $M = (M', n, m, S, T)$ with $|n| = |m| = |S| = |T| = \lambda$ and

$n, m \leq S \leq T$; M takes input x of length n, and runs $M'(x)$ using S space for at most T steps, and finally outputs the first m bits of the output of M'. (If $M'(x)$ does not halt in time T or requires more than S space, M outputs $\perp$.) In other words, given the description M of a Turing machine in this representation, one can efficiently read off its bound parameters denoted as $(M.n, M.m, M.S, M.T)$.

Now we define the class of exponential time Turing machines. For every polynomial D, the class $\mathsf{TM}[D] = \{\mathcal{M}_\lambda\}$ includes all deterministic Turing machines Π_M containing the canonical representation of a Turing machine M of size $D(\lambda)$; $\Pi_M(x, t)$ takes input x and t of length $M.n$ and λ respectively, and runs $M(x)$ for t steps, and finally outputs what M returns.

Remark: Note that machine $\Pi_M(x, t)$ on any input terminates in $t < 2^\lambda$, and hence its output is well-defined. Furthermore, for any two Turing machines M_1 and M_2, they have the same functionality if and only if they produce identical outputs and run for the same number of steps for every input x. This property is utilized when defining and constructing indistinguishability obfuscation for Turing machines, as in previous work [43].

Exponential-time RAM Machines. We consider a canonical representation of RAM machines $R = (R', n, m, S, T)$ identical to the canonical representation of Turing machines above.

For every polynomial D, the class $\mathsf{RAM}[D] = \{\mathcal{R}_\lambda\}$ of polynomial-sized RAM machines include all deterministic RAM machines Π_R, defined as Π_M above for Turning machines, except that the Turing machine M is replace with a RAM machine R.

Classes of randomized algorithms: The above defined classes contain only

deterministic algorithms. We define analogously these classes for their corresponding randomized algorithms. Let $\mathcal{X}[D]$ be any class defined above, we denote by $\mathsf{r}\mathcal{X}[D]$ the corresponding class of randomized algorithms. For example $\mathsf{rCIR}[D]$ denote all randomized circuits of size $D(\lambda)$, and $\mathsf{rTM}[D]$ denote all randomized turning machine of size $D(\lambda)$.

Classes of (arbitrary) polynomial-sized algorithms: The above defined classes consist of algorithms of a fixed polynomial D description size. We define corresponding classes of arbitrary polynomial size. Let $\mathcal{X}[D]$ be any class defined above, we simply denote by $\mathcal{X} = \cup_{\text{poly } D}\mathcal{X}[D]$ the corresponding class of algorithms of arbitrary polynomial size. For instance, CIR and rCIR denotes all deterministic and randomized polynomial-sized circuits, and TM denotes all polynomial-sized Turing machines.

In the rest of the paper, when we write a family of algorithms $\{AL_\lambda\} \in \mathcal{X}$, we mean $\{AL_\lambda\} \in \mathcal{X}[D]$ for some polynomial D. This means, the size of the family of algorithms is bounded by some polynomial. Below, for convenience of notation, when $\mathcal{X}$ is a class of algorithms of arbitrary polynomial size, we write $AL \in \mathcal{X}_\lambda$ as a short hand for $\{AL_\lambda\} \in \{\mathcal{X}_\lambda\}$.

Classes of well-formed algorithms: In the rest of the preliminary, we define various cryptographic primitives. In order to avoid repeating the definitions for different classes of machines, we provide definitions for general classes of algorithms $\{\mathcal{AL}_\lambda\}$ that can be instantiated with specific classes defined above. In particular, we will work with classes of algorithms that are **well-formed**, satisfying the following properties:

1. For every $AL \in \mathcal{AL}_\lambda$, and input x, AL on input x terminates in 2^λ steps.

Note that this also implies that AL has bounded input and output lengths.

2. the size of every ensemble of algorithms $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda\}$ is bounded by some polynomial D in λ, and
3. given the description of an algorithm $AL \in \mathcal{AL}_\lambda$, one can efficiently read off the bound parameters $AL.n, AL.m, AL.S, AL.T$.

All above defined algorithm classes are well-formed. Below, we denote by $T_{AL}(x)$ the running time of AL on input x, and T_{AL} the worst case running time of AL. Note that well-formed algorithm classes are not necessarily efficient; for instance the class of polynomial-sized Turing machines TM contain Turing machines that run for exponential time. In order to define cryptographic primitives for only polynomial-time algorithms, we will use the notation $\mathsf{ALG}^T = \left\{\mathcal{AL}_\lambda^T\right\}$ to denote the class of algorithms in $\mathsf{ALG} = \{\mathcal{AL}_\lambda\}$ that run in time $T(\lambda)$ (in particular, these with $AL_\lambda.T < T(\lambda)$).

In the rest of the paper, all algorithm classes are well-formed.

2.2.2 Indistinguishability Obfuscation

We recall the definition of indistinguishability obfuscation, adapting to arbitrary classes of algorithms. As before, we first define the syntax, correctness and security of $i\mathcal{O}$, and then discuss about different efficiency guarantees.

Definition 1 (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** *A uniform machine $i\mathcal{O}$ is a indistinguishability obfuscator for a class of* deterministic *algorithms $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$, if the following conditions are satisfied:*

Correctness: *For all security parameters $\lambda \in \mathbb{N}$, for all $AL \in \mathcal{AL}_\lambda$, for all input x, we have that*

$$\Pr[AL' \leftarrow i\mathcal{O}(1^\lambda, AL) \ : \ AL'(x) = AL(x)] = 1$$

Security: *For every polynomial T, every* non-uniform PPT *samplable distribution $\mathcal{D}$ over the support $\left\{\mathcal{AL}_\lambda^T \times \mathcal{AL}_\lambda^T \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$, and adversary $\mathcal{A}$, there is a negligible function μ, such that, for sufficiently large $\lambda \in \mathbb{N}$, if*

$$\Pr[(AL_1, AL_2, z) \leftarrow \mathcal{D}(1^\lambda) \ : \ \forall x, \ AL_1(x) = AL_2(x), T_{AL'}(x) = T_{AL}(x),$$
$$(|AL|, AL.n, AL.m, AL.S, AL.T) = (|AL'|, AL'.n, AL'.m, AL'.S, AL'.T)] > 1 - \mu(\lambda)$$

Then,

$$\Big|\Pr[(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) \ : \ \mathcal{A}(i\mathcal{O}(1^\lambda, AL_1), z)]$$
$$- \Pr[(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) \ : \ \mathcal{A}(i\mathcal{O}(1^\lambda, AL_2), z)]\Big| \ \leq \mu(\lambda)$$

where μ is called the **distinguishing gap** *for $\mathcal{D}$ and $\mathcal{A}$.*

*Furthermore, we say that $i\mathcal{O}$ is δ-***indistinguishable** *if the above security condition holds with a distinguishing gap μ bounded by δ. Especially, $i\mathcal{O}$ is* **sub-exponentially indistinguishable** *if $\mu(\lambda)$ is bounded by $2^{-\lambda^\varepsilon}$ for a constant ε.*

Note that in the security guarantee above, the distribution $\mathcal{D}$ samples algorithms AL_1, AL_2 that has the same functionality, and matching bound parameters. This means, an obfuscated machine "reveals" the functionality (as desired) and these bound parameters. We remark that the leakage of the latter is without loss of generality: In the case of circuits, all bound parameters are set to 2^λ. In the case of other algorithm classes, say Turing and RAM machines. If an $i\mathcal{O}$ scheme ensures that one parameter, say $AL.S$, is not revealed, one can simply

consider a representation that always sets that parameter to 2^λ; then security definition automatically ensures privacy of that parameter. See Remark 1 for more details.

Definition 2 (Different Levels of Efficiency of IO)**.** *We say that an indistinguishability obfuscator $i\mathcal{O}$ of a class of algorithms $\{\mathcal{AL}_\lambda\}$ has* **optimal efficiency***, if there is a universal polynomial p such that for every $\lambda \in \mathbb{N}$, and every $AL \in \mathcal{AL}_\lambda$, $i\mathcal{O}(1^\lambda, AL)$ runs in time $p(\lambda, |AL|)$.*

Additionally, we say that $i\mathcal{O}$ has **input- / space- / linear-time- dependent complexity***, if $i\mathcal{O}(1^\lambda, AL)$ runs in time* $\text{poly}(\lambda, |AL|, AL.n)$ / $\text{poly}(\lambda, |AL|, AL.S)$ / $\text{poly}(\lambda, |AL|)AL.T$.

We note that unlike the case of garbling schemes, the optimal efficiency of an $i\mathcal{O}$ scheme does not need to depend on the length of the output. Loosely speaking, the stems from the fact that indistinguishability-based security does not require "programming" outputs, which is the case in simulation-based security for garbling.

***iO* for Specific Algorithm Classes.** We recall the definition of $i\mathcal{O}$ for polynomial-sized circuits, NC^1 [17]; and give definitions of $i\mathcal{O}$ for polynomial time Turing machines [43] and RAM machines with different efficiency guarantees.

Definition 3 (Indistinguishability Obfuscator for Poly-sized Circuits and NC^1)**.** *A uniform* PPT *machine $i\mathcal{O}_{\mathsf{CIR}}(\cdot,\cdot)$ is an indistinguishability obfuscator for polynomial-sized circuits if it is an indistinguishability obfuscator for* CIR *with optimal efficiency.*

A uniform PPT *machine $i\mathcal{O}_{\mathsf{NC}^1}(\cdot,\cdot,\cdot)$ is an indistinguishability obfuscator for NC^1 circuits if for all constants $c \in \mathcal{N}$, $i\mathcal{O}_{\mathsf{NC}^1}(c,\cdot,\cdot)$ is an indistinguishability obfuscator for NC_c with optimal efficiency.*

Definition 4 (*iO* for Turing Machines)**.** *A uniform machine* $i\mathcal{O}_{\mathsf{TM}}(\cdot,\cdot)$ *is a indistinguishability obfuscator for polynomial-time Turing machines, with optimal efficiency or input- / space-dependent complexity, if it is an indistinguishability obfuscator for the class* TM *with the same efficiency.*

Recently, the works by [43, 1] give constructions of *iO* for Turing machines[9] with input-dependent complexity assuming FHE, differing-input obfuscation for circuits, and P-certificates [60]; furthermore, the dependency on input lengths can be removed—leading to a scheme with optimal efficiency—if assuming SNARK instead of P-certificates.

Definition 5 (*iO* for RAM Machines)**.** *A uniform machine* $i\mathcal{O}_{\mathsf{TM}}(\cdot,\cdot)$ *is a indistinguishability obfuscator for polynomial-time Turing machines, with optimal efficiency or linear-time-dependent complexity, if it is an indistinguishability obfuscator for the class* RAM *with the same efficiency.*

Remark 1 (Explicit v.s. Implicit Bound Parameters)**.** *In the above definitions of Garbling Scheme and iO for general algorithms, we considered a canonical representation of algorithms AL that gives information of various bound parameters of the algorithm, specifically, the size* $|AL|$*, bound on input and output lengths* $AL.n, AL.m$*, space complexity* $AL.S$*, and time complexity* $AL.T$*. This representation allows us to define, in a* unified *way, different garbling and iO schemes that depend on different subsets of parameters. For instance,*

- *The Garbling and iO schemes for* TM *that we construct in Section 2.3 and 2.5.1 (from iO and sub-exp iO for circuits respectively) has complexity*

[9] Their works actually realize the stronger notion of differing-input, or extractability, obfuscation for Turing machines

$\mathrm{poly}(|AL|, AL.S, \log(AL.T))$. *(In particular, the size of the garbled TM and obfuscated TM is of this order.)*

- *The garbling scheme for* TM *constructed (from iO for* TM*) sketched in the introduction has complexity* $\mathrm{poly}(|AL|, AL.n, AL.m, \log(AL.T))$.
- *The garbling scheme for* RAM *from one-way functions by [118, 82] has complexity scales polynomially in* $(|AL|, AL.n, AL.m)$ *and quasi-linearly in* $AL.T$. *This construction leads to an iO for* RAM *(from sub-exp iO for circuits) of the same complexity in 2.5.1.*

By using the canonical representation, our general definition allows the garbling or iO scheme to depend on any subset of parameters flexibly. Naturally, if a scheme depends on a subset of parameters, the resulting garbled or obfuscated machines may "leak" these parameters (in the above three examples above, the size of the garbled or obfuscated machines leaks the parameters they depend on); thus, the security definitions must reflect this "leakage" correspondingly. The general security definitions 7 and 1 captures this by allowing leakage of all parameters $|AL|, AL.n, AL.m, AL.S, AL.T$. *However, this seems to "overshoot", as if a specific scheme does not depend on a particular parameter (e.g.* $AL.S$*), then this parameter should be kept private. This can be easily achieved, by simply considering an algorithm representation that always set that parameter to* 2^λ *(e.g.* $AL.S = 2^\lambda$*).*

2.2.3 Garbling Schemes

In this section, we define garbling schemes; our definition is adapted from the definition of [22]. As explained in the introduction the main difference between garbling schemes and randomized encodings is that in garbling schemes the

input is encoded separately from the program. These extra properties will be utilized in our constructions of succinct randomized encodings (or more generally succinct garbling schemes). Our applications will only require randomized encodings; their definition is given in Section 2.5, and is a direct projection of the definition of garbling schemes.

Definition 6 (Garbling Scheme)**.** *A Garbling scheme $\mathcal{GS}$ for a class of (well-formed) deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of algorithms $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ satisfying the following properties:*

Syntax: *For every $\lambda \in \mathbb{N}$, $AL \in \mathcal{AL}_\lambda$ and input x,*

- Garb *is probabilistic and on input $(1^\lambda, AL)$ outputs a pair $(\widehat{AL}, \mathbf{key})$.*[10]
- Encode *is deterministic and on input $(\mathbf{key}, x)$ outputs $\hat{x}$.*
- Eval *is deterministic and on input $(\widehat{AL}, \hat{x})$ produced by* $\mathsf{Garb}, \mathsf{Encode}$ *outputs y.*

Correctness: *For every polynomial T and every family of algorithms $\{AL_\lambda\} \in \left\{\mathcal{AL}_\lambda^T\right\}$ and sequence of inputs $\{x_\lambda\}$, There exists a negligible function μ, such that, for every $\lambda \in \mathbb{N}$, $AL = AL_\lambda$, $x = x_\lambda$,*

$$\Pr[(\widehat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL),\ \hat{x} \xleftarrow{\$} \mathsf{Encode}(\mathbf{key}, x)\ :\ \mathsf{Eval}(\widehat{AL}, \hat{x}) \neq AL(x)] \leq \mu(\lambda)$$

Definition 7 (Security of a Garbling Scheme)**.** *We say that a Garbling scheme $\mathcal{GS}$ for a class of deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$ is secure if the following holds.*

Security: *There exists a uniform machine* Sim*, such that, for every non-uniform* PPT *distinguisher $\mathcal{D}$, every polynomial T', every sequence of algorithms $\{AL_\lambda\} \in$*

[10](Note that as the algorithm class is well-formed, Garb implicitly has all bound parameters of AL.

$\{\mathcal{AL}_\lambda^{T'}\}$, *and sequence of inputs* $\{x_\lambda\}$ *where* $x_\lambda \in \{0,1\}^{AL_\lambda.n}$, *there exists a negligible function* μ, *such that, for every* $\lambda \in \mathbb{N}$, $AL = AL_\lambda$, $x = x_\lambda$ *the following holds:*

$$\Big|\Pr[(\widehat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL),\ \hat{x} \xleftarrow{\$} \mathsf{Encode}(\mathbf{key}, x)\ :\ \mathcal{D}(\widehat{AL}, \hat{x}) = 1]$$
$$-\Pr[(\tilde{AL}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, |x|, |AL|, (n, m, S, T), T_{AL}(x), AL(x))\ :\ \mathcal{D}(\tilde{AL}, \tilde{x}) = 1]\Big| \leq \mu(\lambda)$$

where $(n, m, S, T) = (AL.n, AL.m, AL.S, AL.T)$ *and* Sim *runs in time* $\mathrm{poly}(\lambda, T)$. μ *is called the* **distinguishing gap**.

Furthermore, we say that $\mathcal{GS}$ *is* δ**-indistinguishable** *if the above security condition holds with a distinguishing gap* μ *bounded by* δ. *Especially,* $\mathcal{GS}$ *is* **sub-exponentially indistinguishable** *if* $\mu(\lambda)$ *is bounded by* $2^{-\lambda^\varepsilon}$ *for a constant* ε.

We note that the sub-exponentially indistinguishability defined above is weaker than usual sub-exponential hardness assumptions in that the distinguishing gap only need to be small for PPT distinguisher, rather than sub-exponential time distinguishes.

We remark that in the above definition, simulator Sim receives many inputs, meaning that, a garbled pair $\widehat{AL}, \hat{x}$ reveals nothing but the following: The output $AL(x)$, instance running time $T_{AL}(x)$, input length $|x|$ and machine size $|AL|$, together with various parameters (n, m, S, T) of AL. We note that the leakage of the instance running time is necessary in order to achieve instance-based efficiency (see efficiency guarantees below). The leakage of $|AL|$ can be avoided by padding machines if an upper bound on their size is known. The leakage of parameters (n, m, S, T) can be avoided by setting them to 2^λ; see Remark 1 for more details. In particular, when the algorithms are circuits, inputs to the simula-

tion algorithm can be simplified to $(1^\lambda, |x|, |C|, AL(x))$, since all bound parameters n, m, S, T can be set to 2^λ.

Efficiency Guarantees. we proceed to describe the efficiency requirements for garbling schemes. When considering only circuit classes, all algorithms Garb, Encode, Eval should be polynomial time machines, that is, the complexity of Garb, Eval scales with the size of the circuit $|C|$, and that of Encode with the input length $|x|$. However, when considering general algorithm classes, since the description size $|AL|$ could be much smaller than the running time $AL.T$, or even other parameters $AL.S, AL.n, AL.m$, there could be different variants of efficiency guarantees, depending on what parameters the complexity of the algorithms depends on. Below we define different variants.

Definition 8 (Different Levels of Efficiency of Garbling Schemes)**.** *We say that a garbling scheme $\mathcal{GS}$ for a class of deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$ has succinctness or I/O / space / time-dependent complexity if the following holds.*

Optimal efficiency: *There exists universal polynomials $p_{\mathsf{Garb}}, p_{\mathsf{Encode}}, p_{\mathsf{Eval}}$, such that, for every $\lambda \in \mathbb{N}$, $AL \in \mathcal{AL}_\lambda$ and input $x \in \{0,1\}^{AL.n}$,*

- $(\hat{A}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL)$ *runs in time* $p_{\mathsf{Garb}}(\lambda, |AL|, AL.m)$,[11]
- $\hat{x} = \mathsf{Encode}(\mathbf{key}, x)$ *runs in time* $p_{\mathsf{Encode}}(\lambda, |x|, AL.m)$*, and*
- $y = \mathsf{Eval}(\widehat{AL}, \hat{x})$ *runs in time* $p_{\mathsf{Eval}}(\lambda, |AL|, |x|, AL.m) \times T_{AL}(x)$*, with overwhelming probability over the random coins of* Garb*. We note that* Eval *has instance-based efficiency.*

[11] Note that the running time of Garb and similarly other algorithms that takes AL as an input, implicitly depends logarithmically on the time bound of AL, as its description contains the time bound $AL.T$.

I/O-dependent complexity: *The above efficiency conditions hold with* p_{Garb}, p_{Encode}, p_{Eval} *taking* $AL.n$ *as additional parameters.*

Space-dependent complexity: *The above efficiency conditions hold with* p_{Garb}, p_{Encode}, p_{Eval} *taking* $AL.S$ *as an additional parameter.*

Linear-time-dependent complexity: *The above efficiency conditions hold with* $p_{\mathsf{Garb}}, p_{\mathsf{Encode}}$ *taking* $AL.T$ *as an additional parameter and depending (quasi-)linearly in* $AL.T$*, and the running time of* Eval *is bounded by* $p_{\mathsf{Eval}}(\lambda, |AL|, |x|)AL.T$.

Furthermore, we say that the garbling scheme $\mathcal{GS}$ *has* **succinct input encodings** *if the encoding algorithm* $\mathsf{Encode}(\mathbf{key}, x)$ *runs in time* $p_{\mathsf{Encode}}(1^\lambda, |x|)$.

We say that a garbling scheme is "succinct" if its complexity depends only poly-logarithmically on the time bound. Thus a scheme with space-dependent complexity is succinct for a class of algorithms whose space usage is bounded by a fixed polynomial.

On the dependency on the length of the output. Note that in the optimal efficiency defined above, the complexity of the algorithms depends on the length of their respective inputs and the bound on their output lengths $AL.m$. We argue that this is necessary. This is because that the garbling of an algorithm $\widehat{AL}$ together with an encoding of an input $\hat{x}$ encodes the output $AL(x)$, while leaking nothing beyond $AL(x)$. ($\widehat{AL}, \hat{x}$ is a randomized encoding of AL, x.) Then, assuming the existence of pseudorandom generators G, the total size of the garbled function $\widehat{G}$ and encoded input $\widehat{x}$ must be at least the length of the output of the function. Otherwise, the simulator can "compress" random strings with overwhelming probability, which is a contradiction. Therefore, we allow the complexity of the

algorithms to depend on the length of the output in optimal efficiency.

Static v.s. Adaptive Security Throughout this work, we consider statically secure garbling scheme, that is, the privacy guarantees only hold when the entire computation (AL, x) to be garbled is chosen statically. In the literature, stronger privacy guatantees have been considered [21, 20], allowing the input x to be chosen maliciously and adaptively depending on the garbling of the algorithm AL. In fact, such adaptive security can be further strengthed, when considering garbling schemes that are "input decomposable", that is, the garbling consists of k parts $\{F_i(x_i; r)\}_{i\in[k]}$ each depending on one input. In this more general setting, adaptive security allows inputs to be gradually and adaptively choosen depending on the garbling (or encoding) of previously chosen inputs.

In this work, we however consider only the, weaker, static security, which suffices for all our applications. We leave open the question of constructing succinct adaptively secure garbling schemes.

Garbling Schemes for Specific Algorithm Classes. Next we instantiate the above definition of garbling scheme for general algorithm classed with concrete classes.

Definition 9 (Garbling Scheme for Polynomial-sized Circuits)**.** *A triplet of algorithms* $\mathcal{GS}_{\mathsf{CIR}} = (\mathsf{Garb}_{\mathsf{CIR}}, \mathsf{Encode}_{\mathsf{CIR}}, \mathsf{Eval}_{\mathsf{CIR}})$ *is a garbling scheme (with linear-time-dependent complexity) for polynomial sized circuits if it is a garbling scheme for class* CIR *(with linear-time-dependent complexity).*

We note that in the case of circuits, succinctness means the complexity scales polynomially in $|C|$, whereas linear-time-dependency means the complexity scales linearly with $|C|$.

Definition 10 (Garbling Schemes for Polynomial Time Turing Machines)**.** *A triplet* $\mathcal{GS}_{\mathsf{TM}} = (\mathsf{Garb}_{\mathsf{TM}}, \mathsf{Encode}_{\mathsf{TM}}, \mathsf{Eval}_{\mathsf{TM}})$ *of algorithms is a garbling scheme with optimal efficiency or I/O- / space- / linear-time-dependent complexity (and succinct input encodings) for Turing machines, if it is a garbling scheme for class* TM*, with the same level of efficiency.*

Different efficiency requirements impose qualitatively different restrictions. In this work, we will construct a garbling scheme for Turing machines with space-dependent complexity assuming indistinguishability obfuscation for circuits. The construction of garbling scheme from *iO* for Turing machines, sketched in the introduction, has I/O-dependent complexity. On the other hand, we show that a scheme with is impossible; in particular, the complexity of the scheme must scale with the bound on the output length.

Definition 11 (Garbling Schemes for Polynomial Time RAM Machines)**.** *A triplet* $\mathcal{GS}_{\mathsf{RAM}} = (\mathsf{Garb}_{\mathsf{RAM}}, \mathsf{Encode}_{\mathsf{RAM}}, \mathsf{Eval}_{\mathsf{RAM}})$ *of algorithms is a garbling scheme for polynomial-time RAM machines with optimal efficiency or I/O- / space- / linear-time-dependent-complexity, (and succinct input encodings), if it is a garbling scheme for class* RAM*, with the same level of efficiency.*

Recently, the works by [118, 82] give construction of a garbling scheme for RAM machines with linear-time-dependent complexity and succinct input encodings, assuming only one-way functions.

Garbled Circuits with independent input encoding. In this work, we will make use of a garbling scheme for circuits with a special structural property. In Definition 9, the key **key** for garbling inputs is generated depending on the circuit (by $\mathsf{Garb}(1^\lambda, C)$); the special property of a circuit garbling scheme is that

the **key** can be generated depending only on the length of the input $1^{|x|}$ and the security parameter, which implies that the garbled inputs $\hat{x}$ can also be generated depending only on the plain input x and the security parameter λ, independently of the circuit—we call this *independent input encoding*.

Definition 12 (Garbling Scheme for Circuits with Independent Input Encoding)**.** *A Garbling scheme* $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ *for a deterministic circuit class* $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ *has* **independent input encoding** *if the following holds: For every* $\lambda \in \mathbb{N}$*, and every* $C \in C_\lambda$*,*

- *The algorithm* Garb *on input* $(1^\lambda, C)$ *invokes first* $\mathbf{key} \xleftarrow{\$} \mathsf{Gen}(1^\lambda, 1^{|x|})$ *and then* $\widehat{C} \xleftarrow{\$} \mathsf{Gb}(\mathbf{key}, C)$*, where* Gen *and* Gb *are all* PPT *algorithms.*
- *The security condition holds w.r.t. a simulator* Sim *that on input* $(1^\lambda, 1^{|x|}, 1^{|C|}, T_C(x), C(x))$ *invokes first* $(\tilde{x}, \mathbf{st}) \xleftarrow{\$} \mathsf{Sim}\text{·}\mathsf{Gen}(1^\lambda, |x|)$ *and then* $\tilde{C} \xleftarrow{\$} \mathsf{Sim}\text{·}\mathsf{Gb}((1^\lambda, |x|, |C|, C(x), \mathbf{st})$*, where* $\mathsf{Sim}\text{·}\mathsf{Gen}$ *and* $\mathsf{Sim}\text{·}\mathsf{Gb}$ *runs in time* $\mathrm{poly}(\lambda, |x|)$ *and* $\mathrm{poly}(\lambda, |C|)$ *respectively.*

It is easy to check that many known circuit garbling schemes, in particular the construction by Yao [138], has independent input encoding.

Proposition 2. *Assume the existence of one-way functions that are hard to invert in* Γ *time. Then, there exists a garbling scheme* $\mathcal{GS}_{\mathsf{CIR}}$ *for polynomial-sized circuits with independent input encoding that is* $\Gamma^{-\varepsilon}$*-indistinguishable for some constant* $\varepsilon \in (0, 1)$*.*

2.2.4 Puncturable Pseudo-Random Functions

We recall the definition of puncturable pseudo-random functions (PRF) from [134]. Since in this work, we only uses puncturing at one point, the defini-

tion below is restricted to puncturing only at one point instead of at a polynomially many points.

Definition (Puncturable PRFs)**.** *A puncturable family of PRFs is given by a triple of uniform PPT machines* $(\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F})$*, and a pair of computable functions* $n(\cdot)$ *and* $m(\cdot)$*, satisfying the following conditions:*

Correctness. *For all outputs* K *of* $\mathsf{PRF{\cdot}Gen}(1^\lambda)$*, all points* $i \in \{0,1\}^{n(\lambda)}$*, and* $K(-i) = \mathsf{PRF{\cdot}Punc}(K, i)$*, we have that* $\mathsf{F}(K(-i), x) = \mathsf{F}(K, x)$ *for all* $x \neq i$*.*

Pseudorandom at punctured point. *For every* PPT *adversary* $(\mathcal{A}_1, \mathcal{A}_2)$*, there is a negligible function* μ*, such that in an experiment where* $\mathcal{A}_1(1^\lambda)$ *outputs a point* $i \in \{0,1\}^{n(\lambda)}$ *and a state* σ*,* $K \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ *and* $K(i) = \mathsf{PRF{\cdot}Punc}(K, i)$*, the following holds*

$$\left|\Pr[\mathcal{A}_2(\sigma, K(i), i, \mathsf{F}(K, i)) = 1] - \Pr[\mathcal{A}_2(\sigma, K(i), i, U_{m(\lambda)}) = 1]\right| \leq \mu(\lambda)$$

where μ *is called the* **distinguishing gap** *for* $(\mathcal{A}_1, \mathcal{A}_2)$*.*

Furthermore, we say that the puncturable PRF is δ**-indistinguishable** *if the above pseudorandom property holds with a distinguishing gap* μ *bounded by* δ*. Especially, the puncturable PRF is* **sub-exponentially indistinguishable** *if* $\mu(\lambda)$ *is bounded by* $2^{-\lambda^\varepsilon}$ *for a constant* ε*.*

As observed by [39, 44, 107], the GGM tree-based construction of PRFs [87] from pseudorandom generators (PRGs) yields puncturable PRFs. Furthermore, it is easy to see that if the PRG underlying the GGM construction is sub-exponentially hard (and this can in turn be built from sub-exponentially hard OWFs), then the resulting puncturable PRF is sub-exponentially pseudorandom.

2.3 Succinct Garbling for Bounded-Space Turing Machines

In this section, we construct a garbling scheme for the class of Turing machines TM with space-dependent complexity. Thus when the space complexity of the TM is bounded, it yields a succinct scheme. We will see in the next section that our construction for Turing machines directly applies to general bounded space computation.

Theorem 9. *Assuming the existence of iO for circuits and one-way functions. There exists a garbling scheme for* TM *with space-dependent complexity.*

Towards this, we proceed in two steps: In the first step, we construct a *non-succinct* garbling scheme for TM, which satisfies the correctness and security requirements of Definition 6 and 7, except that the garbling and evaluation algorithms can run in time polynomial in both the time and space complexity, $M.T$ and $M.S$, of the garbled Turing machine M (as well as the simulation algorithm); the produced garbled Turing machine is of size in the same order. In the second step, we show how to reduce the complexity to depend only on the space complexity $M.S$, leading to a garbling scheme with space-dependent complexity. Since in this section, only the space and time bound parameters matter, we will simply write S and T as $M.S$ and $M.T$, and we use the notion D to represent the description size of M.

2.3.1 A Non-Succinct Garbling Scheme

Overview. The execution of a Turing machine M consists of a sequence of steps, where each step t depends on the description of the machine M and its current

configuration conf_t, and produces the next configuration conf_{t+1}. In the Turing machine model, each step takes constant time, independent of the size of the Turing machine and its configuration. However, each step can be implemented using a circuit $\mathsf{Next}^{D,S}$ that on input (M, conf_t) with $|M| \le D, |\mathrm{conf}_t| \le S$, outputs the next configuration conf_{t+1}—we call this circuit the "universal next-step circuit". The size of the circuit is a fixed polynomial p_{Next} in the size of the machine and the configuration, that is, $p_{\mathsf{Next}}(D, S)$. The whole execution of $M(x)$ can be carried out by performing at most T evaluations of $\mathsf{Next}^{D,S}(M, \cdot)$, producing a chain of configurations denoted by,

> $\mathsf{CONFIG}(M, x) = (T^*, \mathrm{conf}_1, \cdots, \mathrm{conf}_T, \mathrm{conf}_{T+1})$, where $T^* = T_M(x)$, conf_1 is the initial configuration with input x $\{\mathrm{conf}_1, \cdots, \mathrm{conf}_{T^*-1}, \mathrm{conf}_{T^*}\}$ are the sequence of configurations until $M(x)$ halts (conf_t is the configuration before the t^{th} step starts), and $\{\mathrm{conf}_{T^*}, \cdots, \mathrm{conf}_{T+1}\}$ are simply set to the output $y = M(x)$.
>
> We note that the initial configuration conf_1 can be derived efficiently from x, conf_{T^*} is called the final configuration, which can be efficiently recognized and from which an output y can be extracted efficiently.

When succinctness is not required, the natural idea to garble a T-step Turing machine computation of $M(x)$ is to produce a chain of T garbled circuits $(\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T)$, for evaluating the next step circuit $\mathsf{Next}^{D,S}(M, \cdot)$ for M. The t^{th} circuit $\mathbf{C}_t$ is designated to compute from the t^{th} configuration conf_t (as input) to the next conf_{t+1}; if the produced conf_{t+1} is a final configuration, then it simply outputs the output y; otherwise, to enable the evaluation of the next garbled circuit $\widehat{\mathbf{C}}_{t+1}$, it translates conf_{t+1} into the corresponding garbled inputs $\widehat{\mathrm{conf}}_{t+1}$ for $\widehat{\mathbf{C}}_{t+1}$—

we call $\mathbf{C}_t$ the t^{th} *step-circuit*. Then evaluation propagates and the intermediate configurations of the execution of M on x is implicitly computed one by one, until it reaches the final configuration, in which case, an output is produced explicitly (without translating into the garbled inputs of the next garbled circuit). Since each computation step is garbled, and all intermediate configurations, except from the final output y, are "encrypted" as garbled inputs, the entire chain of garbled circuits can be simulated given only the output y.

Finally, we note that each step-circuit $\mathbf{C}_t$ evaluates $\mathsf{Next}^{D,S}(M,\cdot)$ and has the capability of garbling an input for the next garbled circuit $\widehat{\mathbf{C}}_t$; this can only be achieved if the circuit garbling scheme has independent input encoding, which ensures that the input garbling can be done independently of the circuit garbling, and only takes time polynomial in the length of the input (rather than, in the size of the circuit).

Our Non-Succinct Garbling Scheme. We now describe formally our non-succinct garbling scheme $\mathcal{GS}_{ns} = (\mathsf{Garb}_{ns}, \mathsf{Encode}_{ns}, \mathsf{Eval}_{ns})$. We rely on a garbling scheme for polynomial-sized circuits with independent input encoding.

- Let $\mathcal{GS}_{\mathsf{CIR}} = (\mathsf{Garb}_{\mathsf{CIR}}, \mathsf{Encode}_{\mathsf{CIR}}, \mathsf{Eval}_{\mathsf{CIR}})$ be a garbling scheme for polynomial-sized circuits, and $\mathsf{Sim}_{\mathsf{CIR}}$ the simulation algorithm. We require $\mathcal{GS}_{\mathsf{CIR}}$ to have independent input encoding, that is, $\mathsf{Garb}_{\mathsf{CIR}} = (\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Gb}_{\mathsf{CIR}})$, and $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim}{\cdot}\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Sim}{\cdot}\mathsf{Gb}_{\mathsf{CIR}})$ as described in Definition 12.

Let $\mathsf{Next}^{D,S}$ be the universal next step circuit for machine of size at most D and space complexity at most S; it has a fixed polynomial size $p_{\mathsf{Next}}(D,S)$ and can be generated efficiently given D and S. For every λ and $M \in \mathsf{TM}_\lambda$, our

scheme proceeds as follows:

The garbling algorithm $\mathsf{Garb}_{ns}(1^\lambda, M)$**:**

Let $S = M.S$, $T = M.T$ and $D = |M|$.

Sample $2T$ sufficiently long random strings $\alpha_1, \cdots, \alpha_t$ and $\beta_1, \cdots \beta_t$; produce a chain of T garbled circuits using $\mathsf{Garb}_{\mathsf{CIR}}$ by running the following program for every $t \in [T]$.

Program $\mathbf{P}^{\lambda,S,M}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$:

1. *Generate the key* key_{t+1} *for the next garbled circuit:*

   If $t < T$, compute the key for the $t + 1^{\text{st}}$ garbled circuit $\mathrm{key}_{t+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_{t+1})$ using randomness α_{t+1}. (Note that key_t is generated for inputs of length S.)

2. *Prepare the step-circuit* $\mathbf{C}_t$*:*

   $Step_t$ on a S-bit input conf_t (i) compute $\mathrm{conf}_{t+1} = \mathsf{Next}^{D,S}(M, \mathrm{conf}_t)$; (ii) if conf_{t+1} is a final configuration, simply outputs the output y contained in it[12]; (iii) otherwise, translate conf_{t+1} to the garbled inputs of the $t + 1^{\text{st}}$ garbled circuit, by computing $\widehat{\mathrm{conf}}_{t+1} = \mathsf{Encode}_{\mathsf{CIR}}(\mathrm{key}_{t+1}, \mathrm{conf}_{t+1})$.

3. *Garble the step-circuit* $\mathbf{C}_t$*:*

   Compute the key using randomness α_t, $\mathrm{key}_t = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_t)$, and garble $\mathbf{C}_t$ using randomness β_t, $\widehat{\mathbf{C}}_t = \mathsf{Gb}_{\mathsf{CIR}}(\mathrm{key}_t, \mathbf{C}_t; \beta_t)$,

4. *Output* $\widehat{\mathbf{C}}_t$*.*

Generate **key** as follows: Compute the key for the first garbled circuit using randomness α_1, $\mathrm{key}_1 = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_1)$; set $\mathbf{key} = \mathrm{key}_1 \| 1^S$.

[12] Pad y with 0 if it is not long enough

Finally, output $\hat{M} = (\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T), \mathbf{key}$.

The encoding algorithm $\mathsf{Encode}_{ns}(\mathbf{key}, x)$**:** Let $\mathrm{conf}_1 \in \{0,1\}^S$ be the initial configuration of M with input x; compute $\hat{x} = \widehat{\mathrm{conf}}_1 = \mathsf{Encode}_{\mathsf{CIR}}(\mathrm{key}_1, \mathrm{conf}_1)$.

The evaluation algorithm $\mathsf{Eval}_{ns}(\hat{M}, \hat{x})$**:** Evaluate the chain of garbled circuits $\hat{M} = (\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T)$ in sequence in T iterations: In iteration t, compute $z = \mathsf{Eval}_{\mathsf{CIR}}(\widehat{\mathbf{C}}_t, \widehat{\mathrm{conf}}_t)$; if z is the garbled inputs $\widehat{\mathrm{conf}}_{t+1}$ for the next garbled circuit $\widehat{\mathbf{C}}_{t+1}$, proceed to the next iteration; otherwise, terminate and output $y = z$.

Next, we proceed to show that $\mathcal{GS}_{ns}$ is a non-succinct garbling scheme for TM.

Efficiency. We summarize the complexity of different algorithms of the non-succinct scheme. It is easy to see that for any Turing machine M with $D = |M|$, $S = M.S$ and $T = M.T$, the garbling algorithm Garb_{ns} runs in time $\mathrm{poly}(\lambda, D, S) \times T$, and produces a garbling machine of size in the same order. Thus the garbling scheme is non-succinct. On the other hand, the encoding and evaluation algorithms Encode_{ns} and Eval_{ns} are all deterministic polynomial time algorithms. Finally, the simulation run in time $\mathrm{poly}(\lambda, D, S) \times T$ as the garbling algorithm.

Correctness. We show that for every polynomial T', every sequence of algorithms $\{M = M_\lambda\} \in \{\mathsf{TM}_\lambda^{T'}\}$, and sequence of inputs $\{x = x_\lambda\}$ where $x_\lambda \in \{0,1\}^{M.n}$, there exists a negligible function μ, such that,

$$\Pr[(\mathbf{key}, \hat{M}) \xleftarrow{\$} \mathsf{Garb}_{ns}(1^\lambda, M),\ \hat{x} = \mathsf{Encode}_{ns}(\mathbf{key}, x)\ :\ \mathsf{Eval}_{ns}(\hat{M}, \hat{x}) \neq M(x)] \leq \mu(\lambda)$$

Let $\mathsf{CONFIG}(M, x) = (T^*, \mathrm{conf}_1, \cdots, \mathrm{conf}_T, \mathrm{conf}_{T+1})$ be the sequence of configurations generated in the computation of $M(x)$, where $T \leq T'(\lambda)$. It follows from

the correctness of the circuit garbling scheme $\mathsf{Garb}_{\mathsf{CIR}}$ that with overwhelming probability (over the randomness of Garb_{ns}), the following is true: (1) for every $t < T^*$, the garbled circuit $\widehat{\mathbf{C}}_t$, if given the garbled input $\widehat{\mathrm{conf}}_t$ corresponding to conf_t, computes the correct garbled inputs $\widehat{\mathrm{conf}}_{t+1}$ corresponding to conf_{t+1}, and (2) for $t = T^*$, the garbled circuit $\widehat{\mathbf{C}}_{T^*}$, if given the garbled input $\widehat{\mathrm{conf}}_{T^*-1}$ corresponding to conf_{T^*-1}, produces the correct output y. (Note that the evaluation procedure terminates after T^* iterations and circuits $\widehat{\mathbf{C}}_t$ for $t > T^*$ are never evaluated). Then since the garbled input $\hat{x}$ equals to the garbled initial configuration $\widehat{\mathrm{conf}}_1$, by conditions (1) and (2), the evaluation procedure produces the correct output with overwhelming probability.

Security. Fix any polynomial T', any sequence of algorithms $\{M = M_\lambda\} \in \{\mathsf{TM}_\lambda^{T'}\}$, and any sequence of inputs $\{x = x_\lambda\}$ where $x_\lambda \in \{0,1\}^{M.n}$. Towards showing the security of $\mathcal{GS}_{ns}$, we construct a simulation algorithm Sim_{ns}, and show that the following two ensembles are indistinguishable: For convenience of notation, we suppress the appearance of $M.n$ and $M.m$ as input to Sim.

$$\left\{\mathsf{real}_{ns}(1^\lambda, M, x)\right\} = \left\{(\hat{M}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}_{ns}(1^\lambda, M),\ \hat{x} = \mathsf{Encode}_{ns}(\mathbf{key}, x)\ :\ (\hat{M}, \hat{x})\right\}_\lambda \quad (2.1)$$

$$\left\{\mathsf{simu}_{ns}(1^\lambda, M, x)\right\} = \left\{(\tilde{M}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}_{ns}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T_M(x), M(x))\ :\ (\tilde{M}, \tilde{x})\right\}_\lambda \quad (2.2)$$

Below we describe the simulation algorithm. Observe that the garbled machine $\hat{M}$ consists of T garbled circuits $(\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T)$ and the garbled input $\hat{x}$ is simply the garbled input of the initial configuration conf_0 (corresponding to x) for the first garbled circuit $\widehat{\mathbf{C}}_1$. Naturally, to simulate them, the algorithm Sim_{ns} needs to utilize the simulation algorithm $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}, \mathsf{Sim{\cdot}Gb}_{\mathsf{CIR}})$ of the circuit garbling scheme, which requires knowing the output of each garbled circuit. In a real evaluation with $\hat{M}, \hat{x}$, the output of the $(T^*)^{\text{th}}$ garbled circuit is $y = M(x)$, the output of the garbled circuits $t < T^*$ is the garbled input $\widehat{\mathrm{conf}}_{t+1}$ for

next garbled circuit $t+1$, and the garbled circuits $t > T^*$ are not evaluated, but for which y is a valid output. Thus, in the simulation, garbled circuits $t = T^*, \cdots, T$ can be simulated using output y; whereas garbled circuits $t = 1, \cdots, T^* - 1$ will be simulated using the *simulated garbled inputs* for circuit $t + 1$. More precisely,

The simulation algorithm $\mathsf{Sim}_{ns}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T^* = T_M(x), y = M(x))$**:**

Sample $2T$ sufficiently long random strings $\alpha_1, \cdots, \alpha_T, \beta_1, \cdots, \beta_T$. Simulate the chain of garbled circuits by running the following program for every $t \in [T]$.

Program $\mathbf{Q}^{\lambda, S, |M|, T^*, y}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$:

1. *Prepare the output out_t for the t^{th} simulated circuit $\widetilde{\mathbf{C}}_t$:*

   If $t \geq T^*$, $out_t = y$. Otherwise, if $t < T^*$, set the output as the garbled input for the next garbled circuits, that is, $out_t = \widetilde{\mathrm{conf}}_{t+1}$ computed from $(\widetilde{\mathrm{conf}}_{t+1}, \mathbf{st}_{t+1}) = \mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_{t+1})$ using randomness α_{t+1}.

2. *Simulate the t^{th} step-circuit $\widetilde{\mathbf{C}}_t$:*

   Given the output out_t, simulate the t^{th} garbled circuit $\widetilde{\mathbf{C}}_t$ by computing first $(\widetilde{\mathrm{conf}}_t, \mathbf{st}_t) = \mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_t)$ and then $\widetilde{\mathbf{C}}_t = \mathsf{Sim{\cdot}Gb}_{\mathsf{CIR}}(1^\lambda, S, q, out_t, \mathbf{st}_t\ ;\ \beta_t)$, using randomness α_t, β_t where $q = q(\lambda, S)$ is the size of the circuit $\mathbf{C}_t$.

3. *Output $\widetilde{\mathbf{C}}_t$.*

Simulate the garbled input $\tilde{x}$ by computing again $(\widetilde{\mathrm{conf}}_1, \mathbf{st}_1) = \mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_1)$ using randomness α_1, and setting $\tilde{x} = \widetilde{\mathrm{conf}}_1$.

Finally, output $(\tilde{M} = (\widetilde{\mathbf{C}}_1, \cdots, \widetilde{\mathbf{C}}_T), \tilde{x})$.

Towards showing the indistinguishability between honestly generated garbling $(\hat{M}, \hat{x})$ and the simulation $(\tilde{M}, \tilde{x})$, we will consider a sequence of hybrids

$\mathsf{hyb}^0_{ns}, \cdots, \mathsf{hyb}^T_{ns}$, where hyb^0_{ns} samples $(\hat{M}, \hat{x})$ honestly, while hyb^T_{ns} generates the simulated garbling $(\tilde{M}, \tilde{x})$. In every intermediate hybrid hyb^γ_{ns}, a hybrid simulator $\mathsf{HSim}^\gamma_{ns}$ is invoked, producing a pair $(\tilde{M}_\gamma, \tilde{x}_\gamma)$. At a high-level, the γ^{th} hybrid simulator on input $(1^\lambda, M, x)$ simulate the first $\gamma - 1$ garbled circuits using the program **Q**, generates the last $T - \gamma$ garbled circuits honestly using the program **P**, and simulates the γ^{th} garbled circuits using the program **R** described below, which "stitches" together the first $\gamma - 1$ simulated circuits with the last $T - \gamma$ honest circuits into a chain that evaluates to the correct output. More precisely, we will denote by

> $\mathsf{COMBINE}[(P_1, S_1), \cdot, (P_\ell, S_\ell)]$ a merged circuit that on input x in the domain X, computes $P_j(x)$ if $x \in S_j$, where $S_1, \cdots, S_\ell$ is a partition of the domain X.

The hybrid simulation algorithm $\mathsf{HSim}^\gamma_{ns}(1^\lambda, M, x)$ **for** $\gamma = 0, \cdots, T$**:**

Compute $T^* = T_M(x)$ and $y = M(x)$, and the intermediate configuration $\mathrm{conf}_{\gamma+1}$ as defined by $\mathsf{CONFIG}(M, x)$.

Sample $2T$ sufficiently long random strings $\{\alpha_t, \beta_t\}_{t \in [T]}$. Simulate the chain of garbled circuits by running the following program for every $t \in [T]$, which combines programs **P**, **Q** and **R** as below.

Program $\mathbf{M}^\gamma = \mathsf{COMBINE}\left[(\mathbf{Q}, [\gamma - 1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma + 1, T])\right](t\,;\,(\alpha_t, \alpha_{t+1}, \beta_t))$:

- If $t \leq \gamma - 1$, compute $\widetilde{\mathbf{C}}_t = \mathbf{Q}^{\lambda, S, |M|, T^*, y}(t\,;\,(\alpha_t, \alpha_{t+1}, \beta_t))$; output $\widetilde{\mathbf{C}}_t$.
- If $t \geq \gamma + 1$, compute $\widehat{\mathbf{C}}_t = \mathbf{P}^{\lambda, S, M}(t\,;\,(\alpha_t, \alpha_{t+1}, \beta_t))$; output $\widehat{\mathbf{C}}_t$.
- If $t = \gamma$, compute $\widetilde{\mathbf{C}}_t = \mathbf{R}^{\lambda, S, \mathrm{conf}_{\gamma+1}}(\gamma\,;\,(\alpha_\gamma, \alpha_{\gamma+1}, \beta_\gamma))$ define as follow:
  1. *Prepare the output out_γ of the simulated γ^{th} circuit $\widetilde{\mathbf{C}}_t$:*

Set the output out_γ to y if $\mathrm{conf}_{\gamma+1}$ is a final configuration. Otherwise, the output should be the garbled input corresponding to $\mathrm{conf}_{\gamma+1}$ for the next garbled circuit; since the $\gamma+1^{\text{st}}$ circuit is generated honestly, we compute $out_\gamma = \widehat{\mathrm{conf}}_{\gamma+1}$ by first computing $\mathrm{key}_{\gamma+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S \ ; \ \alpha_{\gamma+1})$, and then encoding $\widehat{\mathrm{conf}}_{\gamma+1} = \mathsf{Encode}_{\mathsf{CIR}}(\mathrm{key}_{\gamma+1}, \mathrm{conf}_{\gamma+1})$.

(Note that the difference between program $\mathbf{Q}$ and $\mathbf{R}$ is that the former prepares the output out_γ using simulated garbled input $\widetilde{\mathrm{conf}}_{t+1}$, whereas the latter using honestly generated garbled input $\widehat{\mathrm{conf}}_{\gamma+1}$.)

2. *Simulate the γ^{th} circuit $\widetilde{\mathbf{C}}_t$:*

   Given the output out_γ, simulate the γ^{th} garbled circuit $\widetilde{\mathbf{C}}_\gamma$ by computing $(\widetilde{\mathrm{conf}}_\gamma, \mathbf{st}_\gamma) = \mathsf{Sim}\text{·}\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, S \ ; \ \alpha_\gamma)$ and $\widetilde{\mathbf{C}}_t = \mathsf{Sim}\text{·}\mathsf{Gb}_{\mathsf{CIR}}(1^\lambda, S, q, out_\gamma, \mathbf{st}_\gamma \ ; \ \beta_\gamma)$, where $q = q(\lambda, S)$ is the size of the circuit $\mathbf{C}_t$.

If $\gamma > 0$, simulate the garbled input $\tilde{x}_\gamma$ as Sim_{ns} does. Otherwise, if $\gamma = 0$, generate the garbled input $\tilde{x}_0$ honestly as in Garb_{ns} and Encode_{ns}.

Finally, output $(\tilde{M}_\gamma = (\widetilde{\mathbf{C}}_1, \cdots, \widetilde{\mathbf{C}}_\gamma, \widehat{\mathbf{C}}_{\gamma+1}\widehat{\mathbf{C}}_T), \tilde{x}_\gamma)$.

We overload notation $\mathsf{hyb}^\gamma_{ns}(1^\lambda, M, x)$ as the output distribution of the hybrid simulator $\mathsf{HSim}^\gamma_{ns}$. By construction, in $\mathsf{HSim}^\gamma_{ns}$, when $\gamma = 0$, $\mathbf{M}^0 = \mathbf{P}$ and the garbled input $\tilde{x}_0$ is generated honestly; thus, $\{\mathsf{hyb}^0_{ns}(1^\lambda, M, x)\} = \{\mathsf{real}_{ns}(1^\lambda, M, x)\}$ (where real_{ns} is the distribution of honestly generated garbling; see equation (2.1)); furthermore, when $\gamma = T$, $\mathbf{M}^0 = \mathbf{Q}$ and the garbled input $\tilde{x}_\gamma$ is simulated; thus $\left\{\mathsf{hyb}^\gamma_{ns}(1^\lambda, M, x)\right\} = \left\{\mathsf{simu}_{ns}(1^\lambda, M, x)\right\}$ (where simu_{ns} is the distribution of simulated garbling; see equation (2.2)). Thus to show the indistinguishability between

$\{\mathsf{real}_{ns}(1^\lambda, M, x)\}$ and $\{\mathsf{simu}_{ns}(1^\lambda, M, x)\}$, it suffices to show the following claim:

Claim 1. *For every $\gamma \in \mathbb{N}$, the following holds*

$$\left\{\mathsf{hyb}^{\gamma-1}_{ns}(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{hyb}^{\gamma}_{ns}(1^\lambda, M, x)\right\}_\lambda$$

Proof. Fix a $\gamma \in \mathbb{N}$, a sufficiently large $\lambda \in \mathbb{N}$, an $M = M_\lambda$ and a $x = x_\lambda$. The only difference between the garbling $(\tilde{M}_{\gamma-1}, \tilde{x}_{\gamma-1})$ sampled by $\mathsf{hyb}^{\gamma-1}_{ns}(1^\lambda, M, x)$ and the garbling $(\tilde{M}_\gamma, \tilde{x}_\gamma)$ sampled by $\mathsf{hyb}^{\gamma}_{ns}(1^\lambda, M, x)$ is the following: Let conf_γ be the intermediate configuration at the beginning of step γ.

- In $\mathsf{hyb}^{\gamma-1}_{ns}$, the γ^{th} garbled circuit $\widehat{\mathbf{C}}_\gamma$ is generated honestly using program $\mathbf{P}$. The circuit $\mathbf{C}_\gamma$ (as described in algorithm Garb_{ns}) is the composition of the circuit $\mathsf{Next}^{\lambda,S}(M, \cdot)$ and the encoding algorithm $\mathsf{Encode}_{\mathsf{CIR}}(\mathrm{key}_{\gamma+1}, \cdot)$, where $\mathrm{key}_{\gamma+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_{\gamma+1})$ is generated honestly.

  Furthermore, the first $\gamma - 1$ garbled circuits are simulated using $\mathbf{R}$ and $\mathbf{Q}$. The simulation of the first $\gamma - 1$ circuits as well as the generation of the garbled input $\tilde{x}_\gamma$ depends potentially on the garbled input $\widehat{\mathrm{conf}}_\gamma$ corresponding to conf_γ for $\widehat{\mathbf{C}}_\gamma$ (when conf_γ is not a final configuration; see Step 1 in $\mathbf{R}$).

  In other words, the output of $\mathsf{hyb}^{\gamma-1}_{ns}$ can be generated by the following alternative sampling algorithm:

  - Generate garbled circuits $\gamma + 1, \cdots, T$ honestly using program $\mathbf{P}$; prepare the γ^{th} circuit $\mathbf{C}_\gamma$ using $\mathrm{key}_{\gamma+1}$.
  - Receive externally honest garbling $(\widehat{\mathbf{C}}_\gamma, \widehat{\mathrm{conf}}_\gamma)$ of $(\mathbf{C}_\gamma, \mathrm{conf}_\gamma)$.
  - Simulate the first $\gamma - 1$ circuits using $\mathbf{R}$ and $\mathbf{Q}$, with $\widehat{\mathrm{conf}}_\gamma$ hardwired in $\mathbf{R}$.

- In $\mathsf{hyb}_{ns}^{\gamma}$, the γ^{th} garbled circuit $\widetilde{\mathbf{C}}_{\gamma}$ is simulated using program $\mathbf{R}$; the output out_{γ} used for simulation is set to either y (if $\text{conf}_{\gamma+1}$ is a final configuration) or the honestly generated gabled input $\widehat{\text{conf}}_{\gamma+1}$. In other words, $out_{\gamma} = \mathbf{C}_{\gamma}(\text{conf}_{\gamma})$, where $\mathbf{C}_{\gamma}$ is prepared in the same way as above.

  Furthermore, the previous $\gamma - 1$ garbled circuits are also simulated using program $\mathbf{Q}$. Their simulation as well as the generation of the garbled input $\tilde{x}_{\gamma+1}$ depends potentially on the corresponding simulated garbled input $\widetilde{\text{conf}}_{\gamma}$ of $\widetilde{\mathbf{C}}_{\gamma}$.

  In other words, the output of $\mathsf{hyb}_{ns}^{\gamma}$ can be generated by the same alternative sampling algorithm above, except that the second step is modified to:

  - Receive externally simulated garbling $(\widetilde{\mathbf{C}}_{\gamma}, \widetilde{\text{conf}}_{\gamma})$ generated using output $\mathbf{C}_{\gamma}(\text{conf}_{\gamma})$.

Then it follows from the security of the circuit garbling scheme $\mathcal{GS}_{\mathsf{CIR}}$ that the distributions of $(\widehat{\mathbf{C}}_{\gamma}, \widehat{\text{conf}}_{\gamma})$ and $(\widetilde{\mathbf{C}}_{\gamma}, \widetilde{\text{conf}}_{\gamma})$ received externally by the alternative sampling algorithm above are computationally indistinguishable, and thus the distributions of outputs of $\mathsf{hyb}_{ns}^{\gamma-1}$ and $\mathsf{hyb}_{ns}^{\gamma}$, which can be efficiently constructed from them, are also indistinguishable □

Finally, by the above claim, it follows from a hybrid argument over γ, that $\{\mathsf{real}_{ns}(1^{\lambda}, M, x)\}$ and $\{\mathsf{simu}_{ns}(1^{\lambda}, M, x)\}$ are indistinguishable; Hence, $\mathcal{GS}_{ns}$ is a secure garbling scheme for TM.

2.3.2 A Garbling Scheme for TM with Space-dependent Complexity

In this section, we construct a garbling scheme $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ for TM with space-dependent complexity. This scheme will rely on the non-succinct garbling scheme $\mathcal{GS}_{ns} = (\mathsf{Garb}_{ns},$
$\mathsf{Encode}_{ns}, \mathsf{Eval}_{ns})$ in a non-black-box, but largely modular, way.

Overview. The garbling scheme $\mathcal{GS}_{ns}$ described in the previous section is non-succinct because its garbling algorithm Garb_{ns} runs in time proportional to the time-bound T (and generates a garbling of size proportional to T.) Our first observation is that the "bulk" of the computation of Garb_{ns} is evaluating the *same randomized* program $\mathbf{P}(\cdot)$ for T times *with coordinated random coins*, to create a chain of garbled circuits:

$$\hat{M} = (\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T), \qquad \widehat{\mathbf{C}}_t = \mathbf{P}(t; \alpha_t, \alpha_{t+1}, \beta_t)$$

The complexity of each garbled circuit depends only on the size of M and its space complexity S, that is, $\mathrm{poly}(D, S)$ (independent of T). Our main idea towards constructing a garbling scheme $\mathcal{GS}$ with space-dependent complexity is to *defer* the T executions of $\mathbf{P}$, from garbling time (that is, in Garb), to evaluation time (that is, in Eval), by using an indistinguishability obfuscator iO for circuits. More specifically, instead of computing the chain of garbled circuits $\hat{M}$ directly, the new garbling algorithm Garb generates an obfuscation of the program $\mathbf{P}$, that is $\overline{\mathbf{P}} = iO(\mathbf{P})$, and use that as the new garbled machine; (since $\mathbf{P}$ has size $\mathrm{poly}(D, S)$, the obfuscation is "succinct" and so is the new garbling algorithm). The procedure for creating garbled inputs $\hat{x}$ remains the same as in the non-succinct scheme $\mathcal{GS}_{ns}$. Then, on input $(\overline{\mathbf{P}}, \hat{x})$, the new evaluation algorithm Eval

first generates the chain of garbled circuits $\hat{M} = (\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T)$ by evaluating $\overline{\mathbf{P}}$ on inputs from $1, \cdots T$; once the chain $\hat{M}$ of garbled circuits is generated, the output can be computed by evaluating $\mathsf{Eval}_{ns}(\hat{M}, \hat{x})$ as in the non-succinct scheme $\mathcal{GS}_{ns}$. (Note that to make sure that evaluation algorithm has instance-based efficiency, the algorithm Eval actually generates and evaluates $\widehat{\mathbf{C}}_t$'s one by one, and terminates as soon as an output is produced.)

To make the above high-level idea go through, a few details need to be taken care of. First, the program $\mathbf{P}$ is randomized, whereas indistinguishability obfuscators only handles deterministic circuits. This issue is resolved by obfuscating, instead, a wrapper program $\mathbb{P}(t)$ that runs $\mathbf{P}(t)$ with pseudo-random coins generated using a PRF on input t. In fact, the use of pseudo-random coins also allows coordinating the random coins used in different invocations of $\mathbf{P}$ on different inputs, so that they will produce coherent garbled circuits that can be run together. The second question is how to simulate the new garbled machine $\overline{\mathbb{P}} \xleftarrow{\$} i\mathcal{O}(\mathbb{P})$. In the non-succinct scheme the chain $\hat{M}$ of garbled circuits is simulated by running the program $\mathbf{Q}$ for T times (again with coordinated random coins),

$$\tilde{M} = (\widetilde{\mathbf{C}}_1, \cdots, \widetilde{\mathbf{C}}_T) \qquad \widehat{\mathbf{C}}_t = \mathbf{Q}(t; \alpha_t, \alpha_{t+1}, \beta_t)$$

Naturally, in the succinct scheme, the simulation creates $\overline{\mathbb{Q}} \xleftarrow{\$} i\mathcal{O}(\mathbb{Q})$ (where $\mathbb{Q}$ is the de-randomized version for $\mathbf{Q}$, as $\mathbb{P}$ is for $\mathbf{P}$). By the pseudo-randomness of PRF and the security of garbled circuits, we have that the truth tables $\hat{M}$ and $\tilde{M}$ of $\mathbb{P}$ and $\mathbb{Q}$ are indistinguishable; but this does not directly imply that their obfuscations are indistinguishable. We bridge the gap by considering the obfuscation of a sequence of hybrid programs (as in the security proof of the non-succinct garbling scheme).

$$\forall \gamma \in [0, T+1], \quad \mathbf{M}^{\gamma} = \mathsf{COMBINE}\left[(\mathbf{Q}, [\gamma - 1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma + 1, T])\right], \quad \overline{\mathbb{M}}^{\gamma} \xleftarrow{\$} i\mathcal{O}(\mathbb{M}^{\gamma})$$

The sequence of hybrid programs "morphs" gradually from program $\mathbf{P} = \mathbf{M}^0$ to program $\mathbf{Q} = \mathbf{M}^{T+1}$; since every pair of subsequent programs $\mathbf{M}^{\gamma-1}, \mathbf{M}^{\gamma}$ differs only at two inputs ($\gamma - 1$ and γ) with indistinguishable outputs, we can use standard techniques such as puncturing and programming to show that their obfuscations are indistinguishable, and hence so are $\overline{\mathbb{P}}$ and $\overline{\mathbb{Q}}$.

Our Succinct Garbling Scheme. We now describe the formal construction, which relies on the following building blocks.

- A garbling scheme for polynomial-sized circuits, with independent input encoding: $\mathcal{GS}_{\mathsf{CIR}} = (\mathsf{Garb}_{\mathsf{CIR}}, \mathsf{Encode}_{\mathsf{CIR}}, \mathsf{Eval}_{\mathsf{CIR}})$, where $\mathsf{Garb}_{\mathsf{CIR}} = (\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Gb}_{\mathsf{CIR}})$ and its the simulation algorithm is $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim}{\cdot}\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Sim}{\cdot}\mathsf{Gb}_{\mathsf{CIR}})$.
- An indistinguishability obfuscator $i\mathcal{O}_{\mathsf{CIR}}(\cdot,\cdot)$ for polynomial-sized circuits.
- A puncturable PRF $(\mathsf{PRF}{\cdot}\mathsf{Gen}, \mathsf{PRF}{\cdot}\mathsf{Punc}, \mathsf{F})$ with input length $n(\lambda)$ and output length $m(\lambda)$, where $n(\lambda)$ can be set to any super-logarithmic function $n(\lambda) = \omega(\log \lambda)$, and m is a sufficiently large polynomial in λ.

For every λ and $M \in \mathsf{TM}_\lambda$, the garbling scheme $\mathcal{GS}$ proceeds as follows:

Circuit $\mathbb{P} = \mathbb{P}^{\lambda,S,M,K_\alpha,K_\beta}$**:** On input $t \in [T]$, does:

Generates pseudo-random strings $\alpha_t = \mathsf{F}(K_\alpha, t)$, $\alpha_{t+1} = \mathsf{F}(K_\alpha, t+1)$ and $\beta_t = \mathsf{F}(K_\beta, t)$;

Compute $\widehat{\mathbf{C}}_t = \mathbf{P}^{\lambda,S,M}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$ and output $\widehat{\mathbf{C}}_t$.

Circuit $\mathbb{Q} = \mathbb{Q}^{\lambda,S,|M|,T^*,y,K_\alpha,K_\beta}$**:** On input $t \in [T]$, does:

Generate pseudo-random strings $\alpha_t = \mathsf{F}(K_\alpha, t)$, $\alpha_{t+1} = \mathsf{F}(K_\alpha, t+1)$ and $\beta_t = \mathsf{F}(K_\beta, t)$;

Compute $\widetilde{\mathbf{C}}_t = \mathbf{Q}^{\lambda,S,|M|,T^*,y}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$ and output $\widetilde{\mathbf{C}}_t$.

The circuits in Figure 2.1, 2.2 and 2.3 are padded to their maximum size.

Figure 2.1: Circuits used in the construction and simulation of $\mathcal{GS}$

The garbling algorithm $\mathsf{Garb}(1^\lambda, M)$**:**

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$.

2. *Obfuscate the circuit* $\mathbb{P}$*:*

   Obfuscate the circuit $\mathbb{P}(t) = \mathbb{P}^{\lambda,S,M,K_\alpha,K_\beta}(t)$ as described in Figure 2.1, which is essentially a wrapper program that evaluates $\mathbf{P}$ on t using pseudo-random coins generated using K_α and K_β as described above. Obtain $\overline{\mathbb{P}} \xleftarrow{\$} i\mathcal{O}(1^\lambda, \mathbb{P})$.

3. *Generate the key for garbling input:*

   Compute **key** in the same way as the garbling scheme Garb_{ns} does, but using pseudo-random coins generated using K_α. That is, Compute the key for the first garbled circuit using randomness $\alpha_1 = \mathsf{F}(K_\alpha, 1)$, $\mathrm{key}_1 = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_1)$; set $\mathbf{key} = \mathrm{key}_1 \| 1^S$.

4. *Finally, output* $(\overline{\mathbb{P}}, \mathbf{key})$.

The encoding algorithm $\mathsf{Encode}(\mathbf{key}, x)$**:** Compute $\hat{x} = \mathsf{Encode}_{ns}(\mathbf{key}, x)$.

The evaluation algorithm $\mathsf{Eval}(\overline{\mathbb{P}}, \hat{x})$**:** Generate and evaluate the garbled circuits in the non-succinct garbling $\hat{M}$ one by one; terminate as soon as an output is produced. More precisely, evaluation proceeds in T iterations as follows:

At the beginning of iteration $t \in [T]$, previous $t-1$ garbled circuits has been generated and evaluated, producing garbled input $\widehat{\mathrm{conf}}_t$ ($\widehat{\mathrm{conf}}_1 = \hat{x}$). Then, compute $\widehat{\mathbf{C}}_t = \overline{\mathbb{P}}(t)$; evaluate $z = \mathsf{Eval}_{\mathsf{ClR}}(\widehat{\mathbf{C}}_t, \widehat{\mathrm{conf}}_t)$; if z is a valid output, terminate and output $y = z$; otherwise, proceed to the next iteration $t + 1$ with $\widehat{\mathrm{conf}}_{t+1} = z$.

Next, we proceed to show that $\mathcal{GS}$ is a garbling scheme for TM with space-dependent complexity.

Correctness. Fix any machine $M \in \mathsf{TM}$ and input x. Recall that the garbling algorithm Garb generates a pair $(\overline{\mathbb{P}}, \mathbf{key})$; the latter is later used by the encoding algorithm Encode to obtain garbled input $\hat{x}$, while the former is later used by the evaluation algorithm Eval to create the non-succinct garbling $\hat{M} = \{\widehat{\mathbf{C}}_t = \overline{\mathbb{P}}(t)\}_{t\in[T]}$; the non-succinct garbling $\hat{M}$ is then evaluated with $\hat{x}$ using algorithm Eval_{ns}. The distribution of the garbled input and the non-succinct garbling recovered by Eval is as follows:

$$\mathcal{D}_1 = \left\{(\overline{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, M) \ : \ \left(\hat{x} = \mathsf{Encode}(\mathbf{key}, x), \quad \hat{M} = \left\{\widehat{\mathbf{C}}_t = \overline{\mathbb{P}}(t)\right\}_{t\in[T]}\right)\right\}$$

It follows from the construction of Garb, Encode and the correctness of the indistinguishability obfuscator that the above distribution $\mathcal{D}_1$ is identical to the distribution $\mathcal{D}_2$ of a garbled pair $(\hat{M}', \hat{x}')$ generated by the algorithms Garb_{ns}, Encode_{ns} of the non-succinct scheme, *using pseudo-random coins*, formalized below.

$$\mathcal{D}_2 = \Big\{K_\alpha, K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda), \quad \forall t \in [T],\ \alpha_t = \mathsf{F}(K_\alpha, t),\ \beta_t = \mathsf{F}(K_\beta, t)\ :$$
$$\Big(\hat{x}' = \mathsf{Encode}_{ns}(\mathbf{key}' = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_1),\ x), \quad \hat{M}' = \Big\{\widehat{\mathbf{C}}_t = \mathbf{P}(t; \alpha_t, \alpha_{t+1}, \beta_t)\Big\}_{t\in[T]}\Big)\Big\}$$

By the pseudo-randomness of PRF, distribution $\mathcal{D}_2$ is computationally indistinguishable from the garbled pair generated by $\mathsf{Garb}_{ns}, \mathsf{Encode}_{ns}$, using truly random coins.

$$\mathcal{D}_3 = \Big\{(\hat{M}'', \mathbf{key}'') \xleftarrow{\$} \mathsf{Garb}_{ns}(1^\lambda, M)\ :\ \Big(\hat{x}'' = \mathsf{Encode}_{ns}(\mathbf{key}'', x), \quad \hat{M}''\Big)\Big\}$$

The correctness of the non-succinct garbling scheme $\mathcal{GS}_{ns}$ guarantees that with overwhelming probability, evaluating $\hat{M}''$ with $\hat{x}''$ produces the correct output $y = M(x)$; furthermore, the correct output y is produced after evaluating only the first $T^* = T_M(x)$ garbled circuits. Thus, it follows from the indistinguishability between $\mathcal{D}_1$ and $\mathcal{D}_3$ that, when evaluating a garbled pair $(\hat{M}, \hat{x})$ sampled from $\mathcal{D}_1$, the correct output y is also produced after evaluating the first T^* garbled circuits. Given that $\mathcal{D}_1$ is exactly the distribution of the non-succinct garbled pairs generated in Eval, we have that correctness holds.

Efficiency. We show that the garbling scheme $\mathcal{GS}$ has space-dependent complexity.

- The garbling algorithm $\mathsf{Garb}(1^\lambda, M)$ runs in time $\mathrm{poly}(\lambda, |M|, S)$. This is because Garb produces an obfuscation of the program $\mathbb{P}$ (a de-randomized version of P) which garbles circuits $\mathbf{C}_t$ using pseudo-random coins for every input $t \in [T]$. Since the program $\mathbf{C}_t$ has size $q = \mathrm{poly}(\lambda, |M|, S)$ as analyzed in the non-succinct garbling scheme, so does $\mathbf{P}$ and $\mathbb{P}$ (note that the input range T of these two programs are contained as part of the description of M, and hence $|M| > \log T$). Therefore, Garb takes time $\mathrm{poly}(\lambda, |M|, S)$

to produced the obfuscation of $\mathbb{P}$. Additionally, notice that Garb generates the **key** as the algorithm Garb_{ns} does, which in turn runs $\mathsf{Garb}_{\mathsf{CIR}}(1^\lambda, 1^S)$ and takes time $\text{poly}(\lambda, S)$. Overall, Garb runs in time $\text{poly}(\lambda, |M|, S)$ as claimed.

- Encode run in time the same as the Encode_{ns} algorithm which is $\text{poly}(\lambda, |M|, S)$.
- The evaluation algorithm Eval on input $(\overline{\mathbb{P}}, \hat{x})$ produced by $(\overline{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, 1^S)$ and $\hat{x} = \mathsf{Encode}(\mathbf{key}, x)$ runs in time $\text{poly}(\lambda, |M|, S) \times T^*$, $T^* = T_M(x)$, with overwhelming probability.

  It follows from the analysis of correctness of $\mathcal{GS}$ that with overwhelming probability over the coins of Garb, the non-succinct garbling $\hat{M}$ defined by $\overline{\mathbb{P}}$ satisfies that when evaluated with $\hat{x}$, the correct output is produced after T^* iterations. Since Eval does not compute the entire non-succinct garbling $\hat{M}$ in one shot, but rather, generates and evaluates the garbled circuits in $\hat{M}$ one by one. Thus it terminates after producing and evaluating T^* garbled circuits. Since the generation and evaluation of each garbled circuit takes $\text{poly}(\lambda, |M|, S)$ time, overall Eval runs in time $T_M(x) \times \text{poly}(\lambda, |M|, S)$ as claimed.

Security. Fix any polynomial T', any sequence of algorithms $\{M = M_\lambda\} \in \{\mathsf{TM}_\lambda^{T'}\}$, and any sequence of inputs $\{x = x_\lambda\}$ where $x_\lambda \in \{0,1\}^{M.n}$. Towards showing the security of $\mathcal{GS}$, we construct a simulator Sim, satisfying that the following two ensembles are indistinguishable in λ:

$$\left\{\mathsf{real}(1^\lambda, M, x)\right\} = \left\{(\overline{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, M),\ \hat{x} = \mathsf{Encode}(\mathbf{key}, x) \ :\ (\overline{\mathbb{P}}, \hat{x})\right\}_\lambda \quad (2.3)$$

$$\left\{\mathsf{simu}(1^\lambda, M, x)\right\} = \left\{(\overline{\mathbb{Q}}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, |x|, |M|, S, T, T_M(x), M(x)) \ :\ (\overline{\mathbb{Q}}, \tilde{x})\right\}_\lambda \quad (2.4)$$

As discussed in the overview, the simulation will obfuscate the program **Q**

used for simulating the non-succinct garbled machine $\tilde{M} = (\widetilde{\mathbf{C}}_1, \cdots, \widetilde{\mathbf{C}}_T)$. More precisely,

The simulation algorithm $\mathsf{Sim}(1^\lambda, |x|, |M|, S, T, T^* = T_M(x), y = M(x))$**:**

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{Q}$*:*

   Obfuscate the circuit $\mathbb{Q}(t) = \mathbb{Q}^{\lambda, S, |M|, T^*, y, K_\alpha, K_\beta}(t)$ as described in Figure 2.1, which is essentially a wrapper program that evaluates $\mathbf{Q}$ on t, using pseudo-random coins $\{\alpha_t, \beta_t\}$ generated by evaluating F on keys K_α and K_β and inputs $t \in [T]$. Obtain $\overline{\mathbb{Q}} \xleftarrow{\$} i\mathcal{O}(1^\lambda, \mathbb{Q})$.
3. *Simulate the garbled input:*

   Simulate the garbled input $\tilde{x}$ in the same way as simulator Sim_{ns} does, but using pseudo-random coins. That is, compute $(\widetilde{\mathsf{conf}}_1, \mathbf{st}_1) = \mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}(1^\lambda, S \ ; \ \alpha_1)$, where $\alpha_1 = \mathsf{F}(K_\alpha, 1)$; set $\tilde{x} = \widetilde{\mathsf{conf}}_1$.
4. *Finally, output* $(\overline{\mathbb{Q}}, \tilde{x})$.

The simulator $\mathsf{Sim}(1^\lambda, |x|, |M|, S, T, T^*, y = M(x))$ runs in time $\mathrm{poly}(\lambda, |M|, S)$. This follows because the simulator simulates the garbled Turing machine by obfuscating the program $\mathbb{Q}$. As the program $\mathbb{Q}$ simply runs $\mathbf{Q}$ using pseudo-random coins, its size is $\mathrm{poly}(\lambda, |M|, S)$; thus obfuscation takes time in the same order. On the other hand, Sim simulates the garbled input $\tilde{x}$ as the simulator Sim_{ns} does, which simply invokes $\mathsf{Sim}_{\mathsf{CIR}}(1^\lambda, S)$ of the circuit garbling scheme, which takes time $\mathrm{poly}(\lambda, S)$. Therefore, overall the simulation takes time $\mathrm{poly}(\lambda, |M|, S)$ as claimed.

Towards showing the indistinguishability between honestly generated garbling $(\overline{\mathbb{P}}, \hat{x}) \xleftarrow{\$} \mathsf{real}(1^\lambda, M, x)$ and the simulation $(\overline{\mathbb{Q}}, \tilde{x}) \xleftarrow{\$} \mathsf{simu}(1^\lambda, M, x)$ (see equa-

tion (2.3) and (2.4) for formal definition of real and simu), we will consider a sequence of hybrids $\mathsf{hyb}^0, \cdots, \mathsf{hyb}^T$, where the output distribution of hyb^0 is identical to real, while that of hyb^T is identical to simu. In every intermediate hybrid hyb^γ, a hybrid simulator HSim^γ is invoked, producing a pair $(\overline{\mathbb{M}}^\gamma, \tilde{x}^\gamma)$, where $\overline{\mathbb{M}}^\gamma$ is the obfuscation of (the de-randomized wrapper of) a merged program $\mathbf{M}^\gamma$ that produces a hybrid chain of garbled circuit as in the security proof of the non-succinct garbling scheme, where the first γ garbled circuits are simulated and the rest are generated honestly. More precisely,

The hybrid simulation algorithm $\mathsf{HSim}^\gamma(1^\lambda, M, x)$ **for** $\gamma = 0, \cdots, T$**:**

Compute $T^* = T_M(x)$ and $y = M(x)$, and the intermediate configuration $\mathsf{conf}_{\gamma+1}$ as defined by $\mathsf{CONFIG}(M, x)$.

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$.

2. *Obfuscate the circuit* $\mathbb{M}^\gamma$*:*

   Obfuscate the circuit $\mathbb{M}^\gamma(t) = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \mathsf{conf}_{\gamma+1}, K_\alpha, K_\beta}(t)$ as described in Figure 2.1, which is essentially a wrapper program that evaluates the combined program

   $$\mathbf{M}^\gamma = \mathsf{COMBINE}\left[(\mathbf{Q}, [\gamma - 1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma + 1, T])\right](t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t)),$$

   using pseudo-random coins $\{\alpha_t, \beta_t\}$ generated using K_α and K_β. Obtain $\overline{\mathbb{M}}^\gamma \xleftarrow{\$} iO(1^\lambda, \mathbb{M}^\gamma)$.

3. *Simulate the garbled input:*

   If $\gamma > 0$, simulate the garbled input $\tilde{x}^\gamma$ in the same way as in Sim. Otherwise, if $\gamma = 0$, generate $\tilde{x}^0$ honestly, using Garb and Encode.

4. *Finally, output* $(\overline{\mathbb{M}}^\gamma, \tilde{x}^\gamma)$.

Circuit $\mathbb{M}^\gamma = (\mathbb{M}^\gamma)^{\lambda,S,M,T^*,y,\text{conf}_{\gamma+1},K_\alpha,K_\beta}$**:** On input $t \in [T]$, does:

Generate pseudo-random strings $\alpha_t = \mathsf{F}(K_\alpha, t)$, $\alpha_{t+1} = \mathsf{F}(K_\alpha, t+1)$ and $\beta_t = \mathsf{F}(K_\beta, t)$;

Compute $\widetilde{\mathbf{C}}_t = \mathbf{M}^\gamma(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$ and output $\widetilde{\mathbf{C}}_t$, where $\mathbf{M}^\gamma$ is:

$$(\mathbf{M}^\gamma)^{\lambda,S,M,T^*,y,\text{conf}_{\gamma+1}} =$$
$$\mathsf{COMBINE}\left[(\mathbf{Q}, [\gamma-1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma+1, T])\right](t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$$

The circuits in Figure 2.1, 2.2 and 2.3 are padded to their maximum size.

Figure 2.2: Circuits used in the security analysis of $\mathcal{GS}$

We overload the notation $\mathsf{hyb}^\gamma(1^\lambda, M, x)$ as the output distribution of the γ^{th} hybrid. By construction, when $\gamma = 0$, $\mathbf{M}^0 = \mathbf{P}$ and the garbled input $\tilde{x}^0$ is generated honestly; thus, $\{\mathsf{hyb}^0(1^\lambda, M, x)\} = \{\mathsf{real}(1^\lambda, M, x)\}$; furthermore, when $\gamma = T$, $\mathbf{M}^T = \mathbf{Q}$ and the garbled input $\tilde{x}^T$ is simulated; thus $\left\{\mathsf{hyb}^T(1^\lambda, M, x)\right\} = \left\{\mathsf{simu}(1^\lambda, M, x)\right\}$. Therefore, to show the security of $\mathcal{GS}$, it boils down to proving the following claim:

Claim 2. *For every $\gamma \geq 0$, the following holds*

$$\left\{\mathsf{hyb}^\gamma(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{hyb}^{\gamma+1}(1^\lambda, M, x)\right\}_\lambda$$

Proof. Fix a $\gamma \in \mathbb{N}$, a sufficiently large $\lambda \in \mathbb{N}$, an $M = M_\lambda$ and a $x = x_\lambda$. Note that the only difference between $(\overline{\mathbb{M}}^\gamma, \tilde{x}^\gamma) \xleftarrow{\$} \mathsf{hyb}^\gamma$ and $(\overline{\mathbb{M}}^{\gamma+1}, \tilde{x}^{\gamma+1}) \xleftarrow{\$} \mathsf{hyb}^{\gamma+1}$ is the following:

- For every γ, the underlying obfuscated programs $\mathbb{M}^\gamma, \mathbb{M}^{\gamma+1}$ differ on their implementation for at most two inputs, namely $\gamma, \gamma+1$, and,
- when $\gamma = 0$, the garbled input $\tilde{x}^0$ is generated honestly in hyb^0, whereas $\tilde{x}^1$ is simulated in hyb^1.

To show the indistinguishability of the two hybrids, we consider a sequence of sub-hybrids from $\mathsf{H}_0^\gamma = \mathsf{hyb}^\gamma$ to $\mathsf{H}_7^\gamma = \mathsf{hyb}^{\gamma+1}$. Below we describe these hybrids $\mathsf{H}_0^\gamma, \cdots \mathsf{H}_7^\gamma$, and argue that the output distributions of any two subsequent hybrids are indistinguishable. We denote by $(\overline{\mathbb{M}}_i^\gamma, \tilde{x}_i^\gamma)$ the garbled pair produced in hybrid H_i^γ for $i = 0, \cdots, 7$. For convenience, below we suppress the superscript γ, and simply use notations $\mathsf{H}_i = \mathsf{H}_i^\gamma$, $\overline{\mathbb{M}}_i = \overline{\mathbb{M}}_i^\gamma$, $\mathbb{M}_i = \mathbb{M}_i^\gamma$ and $\tilde{x}_i = \tilde{x}_i^\gamma$.

Hybrid H_1: Generate a garbled pair $(\overline{\mathbb{M}}_1, \tilde{x}_1)$ by running a simulation procedure that proceeds identically to HSim^γ, except from the following modifications:

- In the first step, puncture the two PRF keys K_α, K_β at input $\gamma + 1$, and obtain $K_\alpha(\gamma+1) = \mathsf{PRF{\cdot}Punc}(K_\alpha, \gamma+1)$ and $K_\beta(\gamma+1) = \mathsf{PRF{\cdot}Punc}(K_\beta, \gamma+1)$. Furthermore, compute $\alpha_{\gamma+1} = \mathsf{F}(K_\alpha, \gamma+1)$ and $\beta_{\gamma+1} = \mathsf{F}(K_\beta, \gamma+1)$.
- In the second step, obfuscate a circuit $\mathbb{M}_1$ slightly modified from $\mathbb{M}^\gamma$: Instead of having the full PRF keys K_α, K_β hardwired in, $\mathbb{M}_1$ has the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$ and the PRF values $\alpha_{\gamma+1}, \beta_{\gamma+1}$ hardwired in; $\mathbb{M}_1$ proceeds identically to $\mathbb{M}_1$, except that it uses the punctured PRF keys to generate pseudo-random coins corresponding to input $t \neq \gamma+1$ and directly use $\alpha_{\gamma+1}, \beta_{\gamma+1}$ as the coins for input $t = \gamma+1$. See Figure 2.1 for a description of $\mathbb{M}_1 = \mathbb{M}_1^\gamma$.

By construction, H_1 only differs from hyb^γ at which underlying program is obfuscated, and program $\mathbb{M}_1$ has the same functionality as $\mathbb{M}^\gamma$. Thus it follows from the security of indistinguishability obfuscator *iO* that, the obfuscated programs $\overline{\mathbb{M}}^\gamma$ and $\overline{\mathbb{M}}_1$ are indistinguishable. (Furthermore, the garbled inputs $\tilde{x}^\gamma$ and $\tilde{x}_1$ in these two hybrids are generated in the same

way.) Thus, we have that the output $(\overline{\mathbb{M}}_1, \tilde{x}_1)$ of H_1 is indistinguishable from the output $(\overline{\mathbb{M}}^\gamma, \tilde{x}^\gamma)$ of hyb^γ. That is,

$$\left\{\mathsf{hyb}^\gamma(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_0(1^\lambda, M, x)\right\}_\lambda$$

Hybrid H_2: Generate a garbled pair $(\overline{\mathbb{M}}_2, \tilde{x}_2)$ by running the same simulation procedure as in H_1 except from the following modifications: Instead of using pseudo-random coins $\alpha_{\gamma+1}$ and $\beta_{\gamma+1}$, hybrid H_2 samples two sufficiently long truly random string $\alpha'_{\gamma+1}, \beta'_{\gamma+1} \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(\lambda)}$ and replace $\alpha_{\gamma+1}, \beta_{\gamma+1}$ with these truly random strings. More specifically, H_2 obfuscates a program $\mathbb{M}_2$ that is identical to $\mathbb{M}_1$, but with $(K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1})$ hardwired in; furthermore, if $\gamma = 0$, α'_1 (as opposed to α_1) is used to generate the garbled input $\tilde{x}_2$. Since only the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$ are used in the whole simulation procedure, it follows from the pseudorandomness of the punctured PRF that the output $(\overline{\mathbb{M}}_2, \tilde{x}_2)$ of H_2 is indistinguishable from that $(\overline{\mathbb{M}}_1 \tilde{x}_1)$ of hyb_1. That is,

$$\left\{\mathsf{H}_1(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_2(1^\lambda, M, x)\right\}_\lambda$$

Hybrid H_3: Generate a garbled pair $(\overline{\mathbb{M}}_3, \tilde{x}_3)$ by running the same simulation procedure as in H_2 with the following modifications:

- Observe that in program $\mathbb{M}_2$, $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$ are used in the evaluation of at most two inputs, γ and $\gamma+1$:

  For input $\gamma+1$, program $\mathbf{P}$ is invoked with input $\gamma+1$ and randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$, in which a circuit $\mathbf{C}_{\gamma+1}$ is prepared depending on $\alpha_{\gamma+2}$, and then obfuscated by computing

$$\mathrm{key}_{\gamma+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha'_{\gamma+1}) \qquad \widehat{\mathbf{C}}_{\gamma+1} = \mathsf{Gb}_{\mathsf{CIR}}(\mathrm{key}_{\gamma+1}, \mathbf{C}_{\gamma+1}; \beta'_{\gamma+1})$$

If $\gamma > 0$, for input γ , program $\mathbf{R}$ is invoked with input γ and randomness $\alpha_\gamma, \alpha'_{\gamma+1}, \beta_\gamma$, in which a garbled circuit $\widetilde{\mathbf{C}}_\gamma$ is simulated; the output out_γ used for the simulation depends potentially on an honest garbling of $\mathrm{conf}_{\gamma+1}$, that is,

$$\widehat{\mathrm{conf}}_{\gamma+1} = \mathsf{Encode}_{\mathsf{CIR}}\left(\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha'_{\gamma+1}),\ \mathrm{conf}_{\gamma+1}\right)$$

Using out_γ, $\widetilde{\mathbf{C}}_\gamma$ is simulating using randomness $\alpha_\gamma, \beta_\gamma$.

First modification: Hybrid H_3 receives externally the above pair $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\mathrm{conf}}_{\gamma+1}$. Instead of obfuscating $\mathbb{M}_2$ (which computes $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\mathrm{conf}}_{\gamma+1}$ internally), H_3 obfuscates $\mathbb{M}_3$ that has $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\mathrm{conf}}_{\gamma+1}$ directly hardwired in (as well as $K_\alpha(\gamma+1), K_\beta(\gamma+1)$). $\mathbb{M}_3$ on input $\gamma+1$, directly outputs $\widehat{\mathrm{conf}}_{\gamma+1}$; on input γ, it uses $\widehat{\mathrm{conf}}_{\gamma+1}$ to compute $\widetilde{\mathbf{C}}_\gamma$; on all other inputs, it proceeds identically as $\mathbb{M}_2$. (See Figure 2.1 for a description of $\mathbb{M}_3$.) It is easy to see that when the correct values $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\mathrm{conf}}_{\gamma+1}$ are hardwired, the program $\mathbb{M}_3$ has the same functionality as $\mathbb{M}_2$.

- In H_2, if $\gamma = 0$, α'_1 is used for garbling the input,

$$\mathrm{key}_1 = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha'_1) \qquad \widehat{\mathrm{conf}}_1 = \mathsf{Encode}_{\mathsf{CIR}}(\mathrm{key}_1, \mathrm{conf}_1)$$

  where conf_1 is the initial state corresponding to x.

  Second modification: Instead, if $\gamma = 0$, hybrid H_3 receives $\widehat{\mathrm{conf}}_1$ externally, and directly outputs it as the garbled inputs $\hat{x}_3 = \widehat{\mathrm{conf}}_1$.

When H_3 receives the correct values of $(\widehat{\mathrm{conf}}_{\gamma+1}, \widehat{\mathbf{C}}_{\gamma+1})$ externally, it follows from the security of *iO* that the output distribution of H_3 is indistinguishable from that of H_2. That is,

$$\left\{\mathsf{H}_2(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_3(1^\lambda, M, x)\right\}_\lambda$$

Hybrid H_4: Generate a garbled pair $(\overline{\mathbb{M}}_4, \tilde{x}_4)$ by running the same procedure as in H_3, except that H_4 receives externally a simulated pair $(\widetilde{\mathrm{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ produced as follows:

$$(\widetilde{\mathrm{conf}}_{\gamma+1}, \mathbf{st}_{\gamma+1}) = \mathsf{Sim}\text{-}\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, S; \alpha'_{\gamma+1}) \tag{2.5}$$

$$\widetilde{\mathbf{C}}_{\gamma+1} = \mathsf{Sim}\text{-}\mathsf{Gb}_{\mathsf{CIR}}\left(1^\lambda, S, 1^q, out_{\gamma+1}, \mathbf{st}_{\gamma+1}; \beta'_{\gamma+1}\right) \tag{2.6}$$

where $out_{\gamma+1}$ is set to be the output of circuit $\mathbf{C}_{\gamma+1}$ on input $\mathrm{conf}_{\gamma+1}$. Thus, it follows from the security of the circuit garbling scheme $\mathcal{GS}_{\mathsf{CIR}}$ that the simulated pair $(\widetilde{\mathrm{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ that hybrid H_4 receives externally is indistinguishable to the honest pair $(\widehat{\mathrm{conf}}_{\gamma+1}, \widehat{\mathbf{C}}_{\gamma+1})$ that H_3 receives externally. Since these two hybrids only differ in which pair they receive externally, it follows that:

$$\left\{\mathsf{H}_3(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_4(1^\lambda, M, x)\right\}_\lambda$$

Hybrid H_5: Generate a garbled pair $(\overline{\mathbb{M}}_5, \tilde{x}_5)$ by running the same procedure as in H_4, except that instead of receiving $(\widetilde{\mathrm{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ externally, it computes them internally using truly random coins $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$. More precisely,

- It obfuscate a program $\mathbb{M}_5$ that have $K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}$ hardwired in:

  On input $\gamma+1$, it computes $\widetilde{\mathbf{C}}_{\gamma+1}$ using the program $\mathbf{R}$ with randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$ (which computes $\widetilde{\mathbf{C}}_{\gamma+1}$ as described in equations (2.5) and (2.6)).

  On input γ, it computes $\widetilde{\mathbf{C}}_\gamma$ using the program $\mathbf{Q}$ with randomness $\alpha_\gamma, \alpha'_{\gamma+2}, \beta_\gamma$ (which computes internally $\widetilde{\mathrm{conf}}_{\gamma+1}$ as described in equation (2.5)).

  On other inputs $t \neq \gamma, \gamma+1$, it computes as $\mathbb{M}_4$ does.

- If $\gamma = 0$, α'_1 is used for computing $\widetilde{\mathsf{conf}}_1$ as described in equation (2.5), and then output $\tilde{x}_4 = \widetilde{\mathsf{conf}}_1$.

It follows from the fact that $\mathbb{M}_5$ computes $(\widetilde{\mathsf{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ correctly internally, it has the same functionality as $\mathbb{M}_4$; thus, the obfuscation of these two programs are indistinguishable. Combined with the fact that the distribution of the garbled inputs $\tilde{x}_4$ is identical to $\tilde{x}_3$, we have that

$$\left\{\mathsf{H}_4(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_5(1^\lambda, M, x)\right\}_\lambda$$

Hybrid H_6: Generate a garbled pair $(\overline{\mathbb{M}}_6, \tilde{x}_6)$ by running the same procedure as in H_5, except that instead of using truly random coins $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$, use pseudo-random coins $\alpha_{\gamma+1} = \mathsf{F}(K_\alpha, \gamma+1)$ and $\beta_{\gamma+1} = \mathsf{F}(K_\beta, \gamma+1)$. In particular, H_6 obfuscates a program $\mathbb{M}_6$ that is identical to $\mathbb{M}_5$ except that $K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}$ are hardwired in, and if $\gamma = 0$, α_1 is used to generate the garbled input $\tilde{x}_6$. It follows from the pseudo-randomness of the punctured PRF that:

$$\left\{\mathsf{H}_6(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_5(1^\lambda, M, x)\right\}_\lambda$$

Hybrid H_7: Generate a garbled pair $(\overline{\mathbb{M}}_7, \tilde{x}_7)$ by running the hybrid simulator $\mathsf{HSim}^{\gamma+1}$. Note that the only difference between $\mathsf{HSim}^{\gamma+1}$ and the simulation procedure in H_6 is that instead of obfuscating $\mathbb{M}_6$ that has tuple $(K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1})$ hardwired in, $\mathsf{HSim}^{\gamma+1}$ obfuscates $\mathbb{M}^{\gamma+1}$ that has the full PRF keys K_α, K_β hardwired in and evaluates $\alpha_{\gamma+1}, \beta_{\gamma+1}$ internally.

Since $\mathbb{M}^{\gamma+1}$ and $\mathbb{M}_6^\gamma$ has the same functionality, it follows from the security of *iO* that

$$\left\{\mathsf{H}_6(1^\lambda, M, x)\right\}_\lambda \approx \left\{\mathsf{H}_5(1^\lambda, M, x)\right\}_\lambda$$

Finally, by a hybrid argument, we conclude the claim. □

Given the above claim, by a hybrid argument over γ, we have that $\{\mathsf{real}(1^\lambda, M, x)\}$ and $\{\mathsf{simu}(1^\lambda, M, x)\}$ are indistinguishable; Hence, $\mathcal{GS}$ is a secure garbling scheme for TM.

2.4 Succinct Garbling in Other Models of Computation

In the section, we observe that our approach for constructing a succinct garbling scheme for bounded space TM in the previous two sections applies generally to any *bounded space computation* (e.g., bounded-space RAM). This immediately yields a garbling scheme for any model of computation with space-dependent complexity.

Theorem 10. *Assuming the existence of iO for circuits and one-way functions. There exists a garbling scheme for any abstract model of sequential computation, such as* TM *and* RAM*, with space-dependent complexity.*

A Garbing Scheme for Any Bounded Space Computation: Given an underlying circuit garbling scheme $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ with independent input encoding, to construct a garbling scheme $\mathcal{GS}^A$ for $\{\mathcal{AL}_\lambda\}$, proceed in the following two steps:

Step 1: Construct a non-succinct garbling scheme: Observe that the computation of a machine AL of $AL.T$ steps can be divided into $AL.T$ *1-step "blocks"* that transforms the current configuration to the next; therefore, to garble AL, it suffices to produce a sequence of *"garbled blocks"*, one for each 1-step

block. The actual programs being garbled is an *"augmented block"*, whose execution consists of a 1-step block followed by the encoding algorithm of $\mathcal{GS}$ that encodes the output configuration for the next garbled block (when an output is produced, it is output directly without encoding). The final garbling then consists of a sequence of T garbled blocks.

Step 2: Compress the size using IO: As before, we then use iO to "compress" the size of the non-succinct garbling constructed in the first step, by giving the obfuscation of the algorithm that on input t, runs Garb to garble the t^{th} augmented block, producing the t^{th} garbled block. The obfuscated program is the succinct garbled program.

The efficiency and security analysis remains the same as before. This concludes Theorem 10.

2.4.1 Improved Construction and Analysis

Notice that our construction of $\mathcal{GS}^A$ uses the underlying circuit garbling scheme $\mathcal{GS}$ in a black-box way. In fact, the scheme does not even require the underlying garbling scheme to be for circuits—*any garbling scheme for any class of algorithms that is "complete", in particular can be used to implement the augmented blocks suffices*. Below we show that by plugging in the one-time garbled RAM of [118, 82], and modifying the construction of Theorem 10 slightly, we can improve the efficiency of $\mathcal{GS}^A$ when the algorithm class is RAM. More precisely, we show the following theorem.

Theorem 11. *Assuming the existence of iO for circuits and one-way functions. There exists a garbling scheme $\mathcal{GS}^{\mathsf{RAM}}$ for* RAM *with* linear*-space-dependent complexity. Fur-*

thermore, for any RAM R and input x, evaluation of a garbled pair $(\widehat{R}, \hat{x})$ produced by $\mathcal{GS}^{\mathsf{RAM}}$ takes time $\mathrm{poly}(\lambda, |R|) \times (T_R(x) + S)$.

Towards the theorem, we rely on a basic RAM garbling scheme with two properties, independent input encoding and linear complexity. For completeness, we describe the two properties in details below and note that they are satisfied by the construction of garbled RAM of [118, 82].

The Basic RAM Garbling Scheme $\mathcal{GS}'$: Theorem 11 relies on a basic garbling scheme $\mathcal{GS}' = (\mathsf{Garb}', \mathsf{Encode}', \mathsf{Eval}')$ with the following properties. Let R be a RAM machine with parameters n, m, S, T.

Independent input encoding. $\mathcal{GS}'$ has independent input encoding as defined in Definition 12 with a slight strengthening. We repeat the definition and highlight the strengthening.

- The garbling algorithm Garb' consists of:

$$(\mathbf{key}, \widehat{R}) \xleftarrow{\$} \mathsf{Garb}'(1^\lambda, R) \ : \ \mathbf{key} \xleftarrow{\$} \mathsf{Gen}'(1^\lambda), \ \widehat{R} \xleftarrow{\$} \mathsf{Gb}'(\mathbf{key}, R)$$

  Strengthening: Different from Definition 12, the PPT key generation algorithm Gen' depends only on the security parameter 1^λ and not on the length of the input $1^{|x|}$. As a result, the length of **key** produced is bounded by $\mathrm{poly}(\lambda)$.

- The simulation procedure Sim' consists of[13]:

$$(\tilde{R}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}'(1^\lambda, (|x|, |R|, n, m, S, T), R(x)) \ :$$

$$(\tilde{x}, \mathbf{st}) \xleftarrow{\$} \mathsf{Sim{\cdot}Gen}'(1^\lambda, |x|), \ \ \tilde{R} \xleftarrow{\$} \mathsf{Sim{\cdot}Gb}'(1^\lambda, (|x|, |R|, n, m, S, T), R(x), \mathbf{st})$$

[13]Note that the simulation procedure described here does not receive the instance running time $T_R(x)$. This is because, as seen shortly, the complexity of the basic RAM garbling scheme is linear in the time complexity of the RAM machine being garbled, and thus does not have instance based efficiency.

Linear complexity. The complexity of algorithms in the garbling scheme is:

- The garbling algorithm $\mathsf{Gb}'(1^\lambda, R)$ and evaluation algorithm $\mathsf{Eval}'(\widehat{R}, \hat{x})$ run in time $\text{poly}(\lambda, |R|) \times T$. Note that unlike previous efficiency requirements, this complexity bound here does not explicitly depend on the lengths of input and output.
- The input encoding algorithm $\hat{x} \xleftarrow{\$} \mathsf{Encode}'(\mathbf{key}, x)$ runs in time linear in the length of the input $\text{poly}(\lambda)|x|$.

Instantiation of the Basic Garbling Scheme. We observe that the construction of [118, 82] satisfies the above three properties, with some small modifications.

- *independent input encoding:* The construction of [118, 82] is based on Yao's garbled circuits. The latter has independent input encoding Gen that depends on the length of the input. The construction of [118, 82] inherits this property. To remove the dependence on the length of the input, one can modify the scheme as follows: Let the new key generation algorithm Gen' sample a PRF seed as the key $\mathbf{key} = k$ (which depends only on the security parameter), and then augment the garbing and encoding algorithms to first generate the actual key using Gen with pseudo-random coins produced with k and then proceed as before.

  After the modification, the run-time of the garbling and encoding algorithms increase by $\text{poly}(\lambda)T_{\mathsf{Gen}}(\lambda, |x|)$, where $T_{\mathsf{Gen}}(\lambda, |x|)$ is the time used by the original key generation algorithm.
- *Linear Complexity:* The complexity of the garbling, evaluation and encoding algorithms of the construction of [118, 82] is exactly as required above,

namely $\text{poly}(\lambda, |R|) \times T$ for the first two and $\text{poly}(\lambda)|x|$ for the last one.[14] Furthermore, its key generation algorithm runs in time linear in the input length $\text{poly}(\lambda)|x|$.

After applying the modification above, we remove the dependency of the key generation algorithm on the input length (and reduce its run-time to $\text{poly}(\lambda)$), while the complexity of garbling, evaluation and encoding remain at the same order as desired.

More Efficient Garbling Scheme for Bounds Space RAM: Let $\mathcal{GS}' = (\mathsf{Garb}' = (\mathsf{Gen}', \mathsf{Gb}'),$
$\mathsf{Encode}', \mathsf{Eval}')$ be a basic garbling scheme as described above, with simulation procedure $\mathsf{Sim}' = (\mathsf{Sim{\cdot}Gen}', \mathsf{Sim{\cdot}Gb}')$. we now construct a garbling scheme $\mathcal{GS}$ for bounded space RAM with improved efficiency. In particular, it has (1) *linear*-space dependent complexity and (2) produces garbled RAM with $\text{poly}(\lambda, |R|)$ overhead (that is, evaluation of $\widehat{R}, \hat{x}$ takes $\text{poly}(\lambda, |R|)T_R(x)$ steps). In comparison, the previous general construction has *polynomial* space dependent complexity and $\text{poly}(\lambda, |R|, S)$ overhead. Towards this, we plug in $\mathcal{GS}'$ and Sim' into our general construction, and make the following modifications.

Modification to Step 1: As before, the first step is constructing a non-succinct garbling scheme, by dividing a RAM computation into small blocks and garbling all of them using $\mathcal{GS}'$.

The only, and key, difference is, instead of dividing a T step RAM computation into T 1-step "blocks", dividing it into $\lceil T/S \rceil$ S-step "blocks".

[14] In [118, 82], the overhead of garbling is $\text{poly}(\lambda) \times |R| \times \text{poly}\log(n)$, where n is the size of the persistent memory data. Since here we do not consider RAM machine with persistent memory data, we ignore this term.

As before, each block is then augmented with the encoding algorithm $\mathsf{Encode}'(\mathbf{key}, \cdot)$ for garbling the output configuration; and each augmented block is garbled using Garb', producing garbled blocks.

Efficiency. We now analyze various efficiency parameters.

- Each augmented block, say the t^{th}, is a RAM consisting of S steps of computation of R followed by $\mathsf{Encode}'(\mathbf{key}_{t+1}, \cdot)$[15]—denote the augmented block as $B(t, R, \mathbf{key}_{t+1}, \cdot)$. Since $\mathbf{key}_{t+1}$ has size $\mathrm{poly}(\lambda)$, we have,

$$\Psi = |B| = |R| + \mathrm{poly}(\lambda), \quad T_B = \mathrm{poly}(\lambda)S$$

  The latter follows since encoding of an intermediate configuration of R of size S takes $\mathrm{poly}(\lambda)S$ steps.

- By the efficiency of Gb', each garbled block has size

$$\Phi = \mathrm{poly}(\lambda, |B|)T_B = \mathrm{poly}(\lambda, |R|)S$$

- Overall, there are $\lceil T/S \rceil$ blocks, resulting in a non-succinct garbled RAM $\widehat{R}$ of size

$$|\widehat{R}| = \lceil T/S \rceil \times \Phi = \mathrm{poly}(\lambda, |R|)T$$

- We note that for any input x of instance complexity T^*, the output $R(x)$ is produced after evaluating $\lceil T^*/S \rceil$ garbled blocks, taking $\mathrm{poly}(\lambda, |R|)(T^* + S)$ steps.

Modification to Step 2: As before, the second step is using obfuscation to "compress" the size of the non-succinct garbling scheme constructed in Step 1. However, if using any obfuscator to obfuscate the program that generates each of $\lceil T/S \rceil$ garbled blocks, it leads to an obfuscated program

[15] It also has the additional logic for deciding whether the output configuration is a final configuration, and returns the output if so.

of size at least $\text{poly}(\lambda, \Phi) = \text{poly}(\lambda, |R|, S)$. In this case, the complexity of the new garbling scheme is not linear in S, and the overhead of the produced garbled RAM is at least $\text{poly}(\lambda, |R|, S)$.

Better efficiency: To avoid the polynomial overhead due to obfuscation, we instead use an iO for circuits with quasi-linear complexity $|C|\,\text{poly}(\lambda, n)$, where $|C|$ is the size of the circuit obfuscated and n is the length of the input. As shown in Appendix 2.6, such an scheme can be constructed generically from any iO (for circuits), puncturable PRF, and randomized encoding that is local (as defined in Appendix 2.6 and satisfied, for instance, by Yao's garbled circuits), all with $2^{-(n+\omega(\log\lambda))}$-security.

Efficiency. Since the obfuscated programs $\mathbb{P}_i$, $\mathbb{Q}_i$ and $\mathbb{R}_i$ take input a time index t of length $O(\log T)$, and outputs a garbled block computated in time $\text{poly}(\lambda, |R|)S$ (roughly the same as Φ). Therefore, the size of the new garbled RAM (and the complexity for generating it) is,

$$\text{size of garbled RAM} = \text{poly}(\lambda, |R|)S \times \text{poly}(\lambda, \log T) = \text{poly}(\lambda, |R|) \times S \ ,$$

which is linear in the space complexity of R.

Moreover, evaluation of an input x of instance complexity T^* requires generating and evaluating $\lceil T^*/S \rceil$ garbled blocks, which takes time

$$\text{run-time of garbled RAM} = \lceil T^*/S \rceil \times \text{poly}(\lambda, |R|) \times S \ = \text{poly}(\lambda, |R|) \times (S + T^*) \ .$$

This concludes Theorem 11.

Remark 2 (RAM Garbling Scheme with Complexity Linear in the Program Size)**.** *The RAM garbling scheme of Theorem 11 produces garbled RAM of size* $\text{poly}(\lambda, |R|) \times S$ *and run-time* $\text{poly}(\lambda, |R|)T^*$ *(for an input of instance complexity* T^**); both depending polynomially in the description size of the underlying RAM* $|R|$*. We show that the*

complexity can be improved to depending linearly on $|R|$, that is, the garbled RAM has size $\mathrm{poly}(\lambda) \times (|R| + S)$ *and run-time* $\mathrm{poly}(\lambda) \times (|R| + S + T^*)$.

To achieve this, we need to rely on a basic RAM garbling scheme that satisfies the properties, independent input encoding and linear complexity, described above, and the following strengthening: The complexity of the garbling algorithm Gb' *depends linearly on $|R|$, that is,* $\mathrm{poly}(\lambda)(|R|+T)$ *(as opposed to* $\mathrm{poly}(\lambda, |R|)T$*). To obtain such a basic RAM garbling scheme, we observe that there is a universal RAM M, such that, any RAM computation $R(x)$ can be transformed into computing $M^R(x, |R|)$, where the description of R is provided as a part of the initial memory. The universal machine M has constant size and $M^R(x, |R|)$ takes at most $cT_R(x)$ steps for some constant c (since each step of R depends on at most a constant number of bits of the description of R). Then applying the construction of [118, 82] to M with a persistent database R yields a garbled RAM of size* $\mathrm{poly}(\lambda)(T + |R|)$ *(where* $\mathrm{poly}(\lambda)T$ *corresponds to the size of garbling of M and* $\mathrm{poly}(\lambda)|R|$ *corresponds to the garbling of the persistent database R).*

Now, instantiate the construction of Theorem 11 with such a basic RAM garbling scheme, and an additional modification: In Step 1, instead of dividing a T step RAM computation into $\lceil T/S \rceil$ S-step blocks, divide it into $\lceil T/(S + |R|) \rceil$ $(S + |R|)$-step blocks; the rest of the construction follows identically. We now argue that this construction indeed has complexity linear in $|R|$. Each augmented block has the same size as before $\mathrm{poly}(\lambda) + |R|$*, but a longer run-time of* $\mathrm{poly}(\lambda)(S + |R|)$*. By the complexity of the (new) basic RAM garbling scheme, each of the garbled block has size* $\mathrm{poly}(\lambda)(S + |R|)$*. Therefore, when obfuscating using a $i\mathcal{O}$ with quasi-linear complexity, the program that produces the garbled blocks, it leads to a new garbled RAM of size* $\mathrm{poly}(\lambda)(S + |R|)$*. The evaluation of such a garbled RAM with an input x of instance complexity T^* takes time* $\lceil T^*/(S + |R|) \rceil \times \mathrm{poly}(\lambda)(S + |R|) = \mathrm{poly}(\lambda)(S + |R| + T^*)$*. Since the construction and analysis is*

essentially the same as in Theorem 11, we omit the details here.

2.5 Applications

In this section, we address two of our main applications of succinct garbling schemes: succinct $i\mathcal{O}$ and publicly-verifiable delegation and $\mathcal{SNARG}$s. (The rest of the applications outlined in the introduction, follow directly by plugging-in our succinct garbling into previous work.) In fact, all of our applications can be instantiated with succinct randomized encodings; namely, they do not required separate input encoding. We first recall the syntax and properties of randomized encodings.

Randomized Encodings: A randomized encoding scheme $\mathcal{RE} = (\mathsf{RE}, \mathsf{Dec})$ for $\{\mathcal{AL}_\lambda\}$ consists of a randomized encoding algorithm RE and a decoding algorithm Dec. $\mathsf{RE}(1^\lambda, AL, x)$, given any function $AL \in \mathcal{AL}_\lambda$ and input x returns the encoded computation $\widehat{AL}(x)$. Given such an encoding, Dec can decode the result $AL(x)$. Any garbling scheme $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ for $\{\mathcal{AL}_\lambda\}$ can be projected to a corresponding randomized encoding where $\mathsf{RE} = \mathsf{Garb} \circ \mathsf{Encode}$ is given by

$$(\widehat{AL}, \hat{x}) \xleftarrow{\$} \mathsf{RE}(1^\lambda, AL, x), \text{ where } (\widehat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL, x),\ \hat{x} = \mathsf{Encode}(\mathbf{key}, x)$$

and the evaluation algorithm Eval is the decoding algorithm Dec.

In accordance, the correctness, security, and efficiency properties are all defined similarly to garbling schemes, as defined in Section 2.2.3 (in particular, it will be convenient to consider randomized encodings that like garbling schemes also guarantee the privacy of the program and not just the input). When project-

ing a garbling scheme to a randomized encoding scheme as above, the randomized encoding inherits the corresponding efficiency properties of the garbling scheme.

2.5.1 From Randomized Encodings to $i\mathcal{O}$

We present a generic transformation from a garbling scheme for an algorithm class $\{\mathcal{AL}_\lambda\}$ to an indistinguishability obfuscator for $\{\mathcal{AL}_\lambda\}$, assuming sub-exponentially indistinguishability obfuscators for circuits. We require that the algorithm class to have the property that for any $\lambda < \lambda' \in \mathbb{N}$, it holds that every algorithm $AL \in \mathcal{AL}_\lambda$ is also contained in $\mathcal{AL}_{\lambda'}$—we say that such a class is "monotonically increasing". For instance, the class of Turing machines TM and RAM machines RAM are all monotonically increasing.

Proposition 3. *Let $\{\mathcal{AL}_\lambda\}$ be any monotonically increasing class of deterministic algorithms. It holds that if there are*

- i) *a sub-exponentially indistinguishable $i\mathcal{O}$, $i\mathcal{O}^C$, for circuits, and*

  ii) *a sub-exponentially indistinguishable randomized encoding $\mathcal{RE}$ for $\{\mathcal{AL}_\lambda\}$.*
- **then**, *there is an indistinguishability obfuscator $i\mathcal{O}^A$ for $\{\mathcal{AL}_\lambda\}$.*

Furthermore, the following efficiency preservation holds.

- *if $\mathcal{RE}$ has optimal efficiency or I/O-dependent complexity, $i\mathcal{O}^A$ has I/O-dependent complexity.*
- *If $\mathcal{RE}$ has space-dependent complexity, so does $i\mathcal{O}^A$.*

- *If $\mathcal{RE}$ and $i\mathcal{O}^C$ have linear-time-dependent complexity, so does $i\mathcal{O}^A$.*

Before moving to the proof of the proposition, we first note that combining Proposition 3 with constructions of garbling schemes for TM and RAM in Section 2.3 and 2.4, we directly obtain $i\mathcal{O}$ for TM and RAM with space-dependent complexity.

Theorem 12. *Assume a sub-exponentially indistinguishable $i\mathcal{O}$ for circuits and sub-exponentially secure OWF. There is an indistinguishability obfuscator for* TM *and* RAM *with space-dependent complexity.*

Proof of Proposition 3. This result relies on the following natural way of obfuscating probabilistic circuits, abstracted in [56].

Probabilistic $i\mathcal{O}$. Let $i\mathcal{O}$ and F be 2^{λ^ε}-indistinguishable $i\mathcal{O}$ and puncturable PRF. Given a *probabilistic* circuit C, obfuscate it in the following way: Consider another circuit $\Pi^{C,k}$ that on input x, computes C using pseudo-random coins $\mathsf{F}(k,x)$ generated with a hard-wired PRF key k, that is, $\Pi^{C,k}(x) = C(x;\mathsf{F}(k,x))$. The obfuscation of C, denoted as $pi\mathcal{O}(1^\lambda, C)$, is an $i\mathcal{O}$ obfuscation of $\Pi^{C,k}$ for a randomly sampled key C, that is,

$$\widehat{C} \xleftarrow{\$} pi\mathcal{O}(1^\lambda, C), \text{ where } k \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^{\lambda'});\ \widehat{C} \xleftarrow{\$} i\mathcal{O}(1^{\lambda'}, \Pi^{C,k})$$

where $\lambda' = (\lambda + n)^{1/\varepsilon}$ for $n = C.n$, so that $i\mathcal{O}$ and F are $\mathrm{negl}(\lambda)2^n$-indistinguishable. The work of [56] showed that the above obfuscations are indistinguishable for circuits whose output distributions are strongly indistinguishable for every input. More specifically, circuits C_1 and C_2 with the same input length n are strongly indistinguishable (w.r.t. auxiliary input z) if for every input $x \in \{0,1\}^n$, the outputs $C_1(x)$ and $C_2(x)$ are $\mathrm{negl}(\lambda)2^{-n}$ indistinguishable (given z). Summarizing,

Lemma 2 (**piO** for Circuits [56]). *Assume sub-exponentially indistinguishable iO for circuits iO^C, and sub-exponentially indistinguishable OWF. Then, for every class $\{C_\lambda\}$ of polynomial-size circuits, and every non-uniform* PPT *samplable distribution $\mathcal{D}$ over the support of $\left\{C_\lambda \times C_\lambda \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$, if it holds that for every non-uniform* PPT *adversary $\mathcal{R}$, and input x,*

$$\Big|\Pr[(C_1, C_2, z) \xleftarrow{\$} \mathcal{D}_\lambda,\ y \xleftarrow{\$} C_1(x)\ :\ \mathcal{R}(C_1, C_2, x, y, z) = 1] - \Pr[(C_1, C_2, z) \xleftarrow{\$} \mathcal{D}_\lambda,\ y \xleftarrow{\$} C_2(x)\ :\ \mathcal{R}(C_1, C_2, x, y, z) = 1]\Big| \leq \mathrm{negl}(\lambda)2^{-n}$$

the following ensembles are computationally indistinguishable:

$$\left\{C_1, C_2, piO(1^\lambda, C_1), z\right\}_\lambda \approx \left\{C_1, C_2, piO(1^\lambda, C_2), z\right\}_\lambda$$

For completeness, we include a proof sketch of the lemma.

Proof Sketch of Lemma 2. The lemma essentially follows from complexity leveling. To see the proof, first consider a simpler case, where the two circuits C_1 and C_2 have identical implementation on all but one input x^*, and the outputs on x^*, $C_1(x^*)$ and $C_2(x^*)$, are indistinguishable. In this case, it follows directly from the security iO that obfuscation of C_b, $\hat{C}_b \xleftarrow{\$} piO(1^\lambda, C_1)$ is indistinguishable to the obfuscation of $C'_b \xleftarrow{\$} iO(C'_b)$ where C'_b has a punctured key $k(x^*)$ and $C_b(x^*; \mathsf{F}(k, x^*))$ hardwired in; then, it follows from the pseudo-randomness of puncturable PRF and the indistinguishability of $C_1(x^*)$ and $C_2(x^*)$ that $iO(C'_0)$ and $iO(C'_1)$ are indistinguishable. Therefore, overall obfuscation of C_1 and C_2 are indistinguishable.

Now consider the case where C_1 and C_2 are sampled from $\mathcal{D}(1^\lambda)$, and their output distributions for every input are $\mathrm{negl}(\lambda)2^{-n}$-indistinguishable. To show that their $\mathsf{p}IO$ obfuscation are indistinguishable, consider an exponential, 2^n,

number of hybrids, where in each hybrid, a circuit C_i is obfuscated, which outputs $C_2(x)$ for every input $x \leq i$ and outputs $C_1(x)$ for every input $x > i$. Since in every two neighboring hybrids, C_i and C_{i+1} are the same except on one input $x^* = i + 1$. By the argument above, neighboring hybrids have a distinguishing gap $O(\mathrm{negl}(\lambda)2^{-n})$. Thus, by a hybrid argument, obfuscations of C_1 and C_2 are indistinguishable. This concludes the lemma. □

Construction of $i\mathcal{O}$ for General Algorithms. Using Lemma 2, we now prove Proposition 3.

Given $2^{-\lambda^\varepsilon}$-indistinguishable $i\mathcal{O}$ $i\mathcal{O}^C$ and $2^{-\lambda^\varepsilon}$-indistinguishable randomized encoding $\mathcal{RE}$, let $pi\mathcal{O}$ be the obfuscator for probabilistic circuits constructed from $i\mathcal{O}^C$ (and a sub-exponentially secure puncturable PRF implies by sub-exponentially secure $\mathcal{RE}$). Our $i\mathcal{O}$ for the a general algorithm class $\{\mathcal{AL}_\lambda\}$ is defined as follows,

$$\widehat{AL}(\cdot) \xleftarrow{\$} i\mathcal{O}^A(1^\lambda, AL) \text{ where } \widehat{AL}(\cdot) \xleftarrow{\$} pi\mathcal{O}(\lambda, \mathsf{RE}(1^{\lambda'}, AL, \cdot))$$

where the security parameter $\lambda' = (\lambda + n)^{1/\varepsilon}$ for $n = AL.n$ so that RE is $\mathrm{negl}(\lambda)2^n$-indistinguishable. (Note that the reason that we can use the security parameter $\lambda' > \lambda$ is because the algorithm class is monotonically increasing and thus $AL \in \mathcal{AL}_\lambda$ also belongs to $\mathcal{AL}_{\lambda'}$.) The correctness of $i\mathcal{O}^A$ follows from the correctness of $\mathcal{RE}$ and $i\mathcal{O}^C$ underlying $pi\mathcal{O}$. Next, we show the security of $i\mathcal{O}^A$.

Security. Fix a polynomial T, a *non-uniform* PPT samplable distribution $\mathcal{D}$ over the support $\left\{\mathcal{AL}_\lambda^T \times \mathcal{AL}_\lambda^T \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$, such that, with overwhelming probability, $(AL_1, AL_2, z) \leftarrow \mathcal{D}(1^\lambda)$ satisfies that AL_1 and AL_2 are functionally equivalent and has matching parameters. We want to show that the following distributions

are indistinguishable.

$$\left\{(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) \ : \ (i\mathcal{O}^A(1^\lambda, AL_1), z)\right\}_\lambda$$
$$\left\{(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) \ : \ (i\mathcal{O}^A(1^\lambda, AL_2), z)\right\}_\lambda$$

By construction of $i\mathcal{O}^A$, this is equivalent to showing

$$\left\{(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) \ : \ (pi\mathcal{O}(1^\lambda, \ \mathsf{RE}(1^{\lambda'}, AL_1, \cdot)), z)\right\}_\lambda$$
$$\left\{(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) \ : \ (pi\mathcal{O}(1^\lambda, \ \mathsf{RE}(1^{\lambda'}, AL_2, \cdot)), z)\right\}_\lambda$$

Consider the sampler $\mathcal{D}'(1^\lambda)$ that outputs C'_1, C'_2, z, by sampling $(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and setting $C'_b = \mathsf{RE}(1^{\lambda'}, AL_b, \cdot)$. It follows from the security of $\mathcal{RE}$ that for every non-uniform adversary $\mathcal{R}$, and every input x, the output distributions of $C'_1(x)$ and $C'_2(x)$ are $\mathrm{negl}(\lambda)2^n$-indistinguishable, given x, z, C'_1, C'_2. Thus, it follows from Lemma 2 that the above two ensembles are indistinguishable, as well as the obfuscations of AL_1 and AL_2.

Efficiency. Finally, we analyze the efficiency of $i\mathcal{O}^A$. It is easy to see that $pi\mathcal{O}(1^\lambda, C)$ runs in time $T_{pIO}(\lambda, C.n, |C|)$, where $T_{\mathsf{p}IO}$ is a polynomial depending on the running time of the underlying $i\mathcal{O}$ and PRF as well as the parameters of their sub-exponential security; moreover, if the underlying $i\mathcal{O}$ has linear-time-dependent complexity, p_{pIO} also depends linearly in $|C|$ (still polynomially in λ and $C.n$). Let $T_{\mathsf{RE}}(\lambda', |AL|, n, m, S, T)$ be the running time of $\mathsf{RE}(1^{\lambda'}, AL, x)$. Overall, the running time of $i\mathcal{O}^A(1^\lambda, AL)$ is,

$$T_{pIO}(\lambda, n, \ T_{\mathsf{RE}}(\lambda', |AL|, n, m, S, T)) \text{ where } \lambda' = \mathrm{poly}(\lambda, n)$$

Therefore,

- If $\mathcal{RE}$ has optimal efficiency (that is, T_{RE} depends only on m) or I/O-dependent complexity (that is, T_{RE} does not depend on S, T), $i\mathcal{O}^A$ has I/O-dependent complexity.

- If $\mathcal{RE}$ has space-dependent complexity (that is, T_{RE} does not depend on T), so does iO^A.
- If $\mathcal{RE}$ and the underlying iO has linear-time-dependent complexity (that is, T_{RE} depends linearly on T and T_{pIO} depends linearly on $|C|$), so does iO^A.

This concludes the proof of Proposition 3.

More Efficient Construction. Evaluating the iO for TM and RAM obtained in Theorem 12 on input x, involves evaluating the obfuscated program on x once to obtain a randomized encoding $\widehat{AL}(x)$, and then decode it. When relying on an arbitrary randomized encoding with space-dependent complexity, the overall evaluation takes time $T_{AL}(x) \times \text{poly}(\lambda, |AL|, S)$. When the space is large, the overhead on run-time is large.

We now improve the evaluation efficiency by combining Proposition 3 with the specific RAM garbling scheme of Theorem 11.

Theorem 13. *Assume a sub-exponentially indistinguishable iO for circuits and sub-exponentially secure OWFs. There is an indistinguishability obfuscator for* TM *and* RAM *with, where obfuscation of a machine R takes time linear in the space complexity* $\text{poly}(\lambda, n, |R|) \times S$*, and evaluation of the obfuscated program on input x takes time* $\text{poly}(\lambda, n, |R|) \times (T_R(x) + S)$*, with $n = R.n$ and $S = R.S$.*

Towards the above theorem, consider instantiating the Proposition 3 using the RAM garbling scheme of Theorem 11 and a sub-exponentially secure iO scheme with quasi-linear complexity (implied by sub-exponentially secure iO and OWF as shown in Appendix 2.6). Recall that the RAM garbling scheme of Theorem 11 has linear-space-dependent complexity $\text{poly}(\lambda, |R|) \times S$ and evaluation time $\text{poly}(\lambda, |R|) \times (T^* + S)$ with $T^* = T_R(x)$; such a garbling scheme leads

to a randomized encoding algorithm RE with the same encoding and decoding complexity. By the same efficiency analysis as in Proposition 3, this instantiation yields an $i\mathcal{O}$ for RAM with linear-space-dependent complexity, namely $\text{poly}(\lambda, |R|, n)S$. Therefore, its evaluation time is now $\text{poly}(\lambda, |R|, n) \times S + \text{poly}(\lambda, |R|) \times (T^* + S)$, which is $\text{poly}(\lambda, |R|, n) \times (S + T^*)$.

Remark 3 (Indistinguishability Obfuscation with Complexity Linear in the Program Size)**.** *In remark 2, we showed that the efficiency of our RAM garbling scheme can be improved to depend only linearly in program description size $|R|$, namely, it has garbling complexity of* $\text{poly}(\lambda)(|R|+S)$ *and evaluation complexity of* $\text{poly}(\lambda)(|R|+S+T^*)$*. When using such a RAM garbling scheme as the underlying scheme in our construction of IO scheme for RAM, we obtain an $i\mathcal{O}$ scheme with complexity* $\text{poly}(\lambda, n)(|R| + S)$ *and evaluation time* $\text{poly}(\lambda, n)(|R| + S + T^*)$*.*

2.5.2 Publicly-Verifiable Delegation, $\mathcal{SNARG}$s for $\mathcal{P}$, and Succinct NIZKs for NP

We now present the publicly-verifiable delegation scheme for bounded-space computations, following from our succinct randomized encodings, as well as a general transformation from delegation schemes to succinct non-interactive arguments. We also note the implications to succinct NIZKs as a corollary of our succinct $i\mathcal{O}$ and the work of [135].

P-delegation

A delegation system for $\mathcal{P}$ is a 2-message protocol between a verifier and a prover. The verifier consists of two algorithms $(\mathcal{G}, \mathcal{V})$, given a (well-formed) algorithm, input, and security parameter $z = (AL, \mathsf{inp}, \lambda)$, $\mathcal{G}$ generates a message σ. The prover, given (z, σ), produces a proof π attesting that AL accepts inp within $AL.T$ steps. $\mathcal{V}$ then verifies the proof. In a privately-verifiable system, the $\mathcal{G}$ produces, in addition to the (public) message σ, a secret verification state τ, and verification by $\mathcal{V}$ requires (z, σ, τ, π). In a publicly-verifiable scheme, τ can be published (together with σ), without compromising soundness.

We shall require that the running time of $(\mathcal{G}, \mathcal{V})$ will be significantly smaller than $AL.T$, and that the time to prove is polynomially related to $AL.T$.

Definition ($\mathcal{P}$-Delegation)**.** *A prover and verifier* $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ *constitute a delegation scheme for* $\mathcal{P}$ *if it satisfies:*

1. **Completeness:** *for any* $z = (AL, \mathsf{inp}, \lambda)$*, such that AL accepts* inp *within AL.T steps:*

$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{c} (\sigma, \tau) \leftarrow \mathcal{G}(z) \\ \pi \leftarrow \mathcal{P}(z, \sigma) \end{array}\right] = 1 \quad .$$

2. **Soundness:** *for any poly-size prover* $\mathcal{P}^*$*, polynomial* $T(\cdot)$*, there exists a negligible* $\alpha(\cdot)$ *such that for any* $z = (AL, \mathsf{inp}, \lambda)$*, such that* $AL.T \leq T(\lambda)$*, and AL does* **not** *accept* inp *within AL.T steps:*

$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{c} (\sigma, \tau) \leftarrow \mathcal{G}(z) \\ \pi \leftarrow \mathcal{P}^*(z, \sigma) \end{array}\right] \leq \alpha(\lambda) \quad .$$

3. **Optimal verification and instance-based prover efficiency:** *There exists a (universal) polynomial* p *such that for every* $z = (AL, \mathsf{inp}, \lambda)$*:*

- *the verifier algorithms* $(\mathcal{G}, \mathcal{V})$ *run in time* $p(\lambda, |AL|, |\mathsf{inp}|, \log AL.T)$*;*
- *the prover* $\mathcal{P}$ *runs in time* $p(\lambda, |AL|, |\mathsf{inp}|)T_{AL}(x)$.

3′. **Space-dependent verification complexity:** *The scheme has space-dependent verification complexity if the running time of the* $(\mathcal{G}, \mathcal{V})$ *may also depend on space; concretely: there exists a (universal) polynomial* p *such that for every* $z = (AL, \mathsf{inp}, \lambda)$*:*

- *the verifier algorithms* $(\mathcal{G}, \mathcal{V})$ *run in time* $p(\lambda, |AL|, \log AL.T, AL.S)$.

The system is said to be **publicly-verifiable** *if soundness is maintained when the malicious prover* $\mathcal{P}^*$ *is also given the verification state* τ.

Remark 4 (Input Privacy)**.** *Our construction achieves an additional property of input privacy which states that the first message of the delegation scheme* σ *leaks no information about the input* x *on which the computation of* AL *is being delegated, beyond the output* $AL(x)$*. This ensures that, in the outsourcing computation application, the server performing the computation learns no more than is necessary about the input to the computation.*

We next present a publicly-verifiable delegation with fast verification based on any succinct randomized encoding, and one-way functions.

The scheme. Let f be a one-way function and $(\mathsf{RE}, \mathsf{Dec})$ be a randomized encoding scheme. We describe $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ as follows. Let $z = (AL, \mathsf{inp}, \lambda)$ be a tuple consisting of an algorithm, input, and security parameter.

Generator $\mathcal{G}(z)$**:**

For $r \leftarrow \{0,1\}^\lambda$, let $AL'(\mathsf{inp}, r)$ be the machine that returns r if $AL(\mathsf{inp}) = 1$ and $\perp$ otherwise. $\mathcal{G}$ generates and outputs $\sigma \leftarrow \mathsf{RE}(1^\lambda, \Pi', (\mathsf{inp}, r))$ and $\tau = f(r)$.

Prover $\mathcal{P}(z, \sigma)$:

$\mathcal{P}$ simply runs $\pi \leftarrow \mathsf{Dec}(\sigma)$ and outputs π.

Verifier $\mathcal{V}(z, \sigma, \tau, \pi)$:

$\mathcal{V}$ outputs 1 if and only if $f(\pi) = \tau$.

We prove that $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ is a P-delegation scheme as follows.

Theorem 14. *If* $(\mathsf{RE}, \mathsf{Dec})$ *is a randomized encoding scheme with optimal complexity (resp. space dependent complexity), then* $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ *as described above is a publicly verifiable P-delegation scheme with optimal verification (resp. space-dependent verification).*

Proof. The completeness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ follows directly from the correctness of $(\mathsf{RE}, \mathsf{Dec})$. Also, note that the running time of the verifier algorithms $(\mathcal{G}, \mathcal{V})$ is related to the running time of RE. Therefore, it also follows directly that if $(\mathsf{RE}, \mathsf{Dec})$ has optimal complexity (resp. space dependent complexity) then $(\mathcal{G}, \mathcal{V})$ satisfies the property of optimal verification (resp. space-dependent verification), and the instance-based prover efficiency follows from the fact the randomized encoding has instance-efficiency. It remains to show the soundness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$.

To show soundness, we will rely on the security of $(\mathsf{RE}, \mathsf{Dec})$ and the one-wayness of f. Assume for contradiction there exists poly-size prover $\mathcal{P}^*$ and

polynomial $p(\cdot)$ such that for infinitely many $z = (AL, \mathsf{inp}, \lambda)$ where AL does not accept inp and $AL.T \leq p(\lambda)$, we have that

$$\Pr\left[\mathcal{V}(z,\sigma,\tau,\pi) = 1 \;\middle|\; \begin{array}{c}(\sigma,\tau) \leftarrow \mathcal{G}(z)\\ \pi \leftarrow \mathcal{P}^*(z,\sigma,\tau)\end{array}\right] \geq \frac{1}{p(\lambda)} \quad .$$

Let $\mathcal{Z}$ be the sequence of such $z = (AL, \mathsf{inp}, \lambda)$ and consider any $z \in \mathcal{Z}$. Recall that $\mathcal{G}(z)$ samples $r \leftarrow \{0,1\}^\lambda$ and outputs $\sigma \leftarrow \mathsf{RE}(1^\lambda, AL', (\mathsf{inp}, r))$ and $\tau \leftarrow f(r)$. Since AL does not accept inp, we have that $AL'(\mathsf{inp}, r)$ outputs $\perp$. By the security of $(\mathsf{RE}, \mathsf{Dec})$, there exists a PPT simulator Sim such that the ensembles $\{\mathsf{RE}(1^\lambda, AL', (\mathsf{inp}, r)\}_{r\in\{0,1\}^\lambda, z\in\mathcal{Z}}$ and $\{\mathsf{Sim}(1^\lambda, \perp, AL', 1^{|\mathsf{inp}|+|r|})\}_{r\in\{0,1\}^\lambda, z\in\mathcal{Z}}$ are indistinguishable. Therefore, given a simulated $\sigma \leftarrow \mathsf{Sim}(1^\lambda, \perp, AL', 1^{|\mathsf{inp}|+|r|})$ we have that $\mathcal{P}^*$ still convinces $\mathcal{V}$ with some noticeable probability. More formally, for infinitely many $z \in \mathcal{Z}$, we have that

$$\Pr\left[\mathcal{V}(z,\sigma,\tau,\pi) = 1 \;\middle|\; \begin{array}{r}r \leftarrow \{0,1\}^\lambda\\ \sigma \leftarrow \mathsf{Sim}(1^\lambda, \perp, AL', 1^{|\mathsf{inp}|+|r|})\\ \tau \leftarrow f(r)\\ \pi \leftarrow \mathcal{P}^*(z,\sigma,\tau)\end{array}\right] \geq \frac{1}{p(\lambda)} - \alpha(\lambda) \quad .$$

for some negligible function $\alpha(\cdot)$.

Recall that $\mathcal{V}$ outputs 1 if and only if $f(\pi) = \tau$. Therefore $\mathcal{V}(z,\sigma,\tau,\pi) = 1$ implies that $\mathcal{P}^*$ when given $\tau = f(r)$ outputs π which is in the pre-image of $f(r)$. Hence $\mathcal{P}^*$ can be used to break the one-wayness of f and we have a contradiction. This completes the proof of the theorem. □

$\mathcal{SNARG}$s for P

A succinct non-interactive argument system ($\mathcal{SNARG}$) for $\mathcal{P}$ is a delegation system where the first message σ is *reusable*, it is independent of any specific computation, and can be used to verify an unbounded number of computations. In a privately-verifiable $\mathcal{SNARG}$, soundness might not be guaranteed if the prover learns the result of verification on different inputs, which can be seen as certain leakage on the private state τ (this is sometimes referred to as the *verifier rejection problem*). Accordingly, in this case, we shall also address a strong soundness requirement, which says that soundness holds, even in the presence of a verification oracle.

Definition ($\mathcal{SNARG}$)**.** *A $\mathcal{SNARG}$ $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ is defined as a delegation scheme, with the following change to the syntax of $\mathcal{G}$: the generator $\mathcal{G}$ now gets as input a security parameter, time bound, and input bound $\lambda, T, n \in \mathbb{N}$, and does not get AL, inp as before. We require that*

1. **Completeness:** *for any $z = (AL, \mathsf{inp}, \lambda)$, such that $AL.T \leq T$ and $|AL, \mathsf{inp}| \leq n$, and AL accepts* inp*:*

$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{r} (\sigma, \tau) \leftarrow \mathcal{G}(\lambda, T, n) \\ \pi \leftarrow \mathcal{P}(z, \sigma) \end{array}\right] = 1 \quad .$$

2. **Soundness:** *for any poly-size prover $\mathcal{P}^*$, polynomials $T(\cdot), n(\cdot)$, there exists a negligible $\alpha(\cdot)$ such that for any $z = (AL, \mathsf{inp}, \lambda)$, where $AL.T \leq T(\lambda)$, $|AL, \mathsf{inp}| \leq n(\lambda)$, and AL does* **not** *accept* inp*:*

$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{r} (\sigma, \tau) \leftarrow \mathcal{G}(\lambda, T(\lambda), n(\lambda)) \\ \pi \leftarrow \mathcal{P}^*(z, \sigma) \end{array}\right] \leq \alpha(\lambda) \quad .$$

2*. **Strong soundness:** *for any poly-size oracle-aided prover $\mathcal{P}^*$, polynomials $T(\cdot), n(\cdot)$, there exists a negligible $\alpha(\cdot)$ such that for any $z = (AL, \mathsf{inp}, \lambda)$, where $AL.T \leq T(\lambda)$, $|AL, \mathsf{inp}| \leq n(\lambda)$, and AL does* **not** *accept* inp*:*

$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{r} (\sigma, \tau) \leftarrow \mathcal{G}(\lambda, T(\lambda), n(\lambda)) \\ \pi \leftarrow \mathcal{P}^{*\mathcal{V}(\cdot,\sigma,\tau,\cdot)}(z, \sigma) \end{array}\right] \leq \alpha(\lambda) \ .$$

3. **Optimal verification and instance-based prover efficiency:** *There exists a (universal) polynomial p such that for every $z = (AL, \mathsf{inp}, \lambda)$:*

   - *the verifier algorithms $(\mathcal{G}, \mathcal{V})$ run in time $p(\lambda, n(\lambda), \log AL.T)$;*
   - *the prover $\mathcal{P}$ runs in time $p(\lambda, |AL|, |\mathsf{inp}|)T_{AL}(\mathsf{inp}))$.*

As before, the system is said to be publicly-verifiable if soundness is maintained when the malicious prover is also given the verification state τ. (In this case, strong soundness follows from standard soundness.) Also, we can naturally extend the definition for the case of semi-succinctness, in which case, $\mathcal{G}$ will also get a space bound S, and the running time of algorithms $(\mathcal{G}, \mathcal{V})$ may also depend on S

Remark 5 (Non-adaptive soundness)**.** *Note that in the definition above and in our construction, we will consider only* non-adaptive *soundness, as opposed to* adaptive *soundness where the malicious prover $\mathcal{P}^*$ can pick the statement z after seeing the first message σ.*

We now show a simple transformation, based on IO, that takes any 2-message delegation scheme (e.g., the one constructed above), and turns it into a $\mathcal{SNARG}$ for $\mathcal{P}$. The transformation works in either the public or private verification setting. Furthermore, it always results in a $\mathcal{SNARG}$ with strong soundness,

even the delegation we start with does not have strong soundness (such as the scheme of [106]).

The scheme. Let $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ be a P-delegation scheme, $(\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F})$ be a puncturable PRF scheme, and $i\mathcal{O}$ be an indistinguishability obfuscator. We describe a $\mathcal{SNARG}(\mathcal{P}, (\mathcal{G}, \mathcal{V})))$ as follows.

Let $z = (AL, \mathsf{inp}, \lambda)$ be a tuple consisting of an algorithm, input and security parameter such that $|AL, \mathsf{inp}| \leq n$ and $AL.T \leq T$. For notational convenience, we decompose $\mathcal{G}_d$ into $(\mathcal{G}_{\sigma_d}, \mathcal{G}_{\tau_d})$ where $\mathcal{G}_{\sigma_d}(z)$ only outputs the message σ_d and $\mathcal{G}_{\tau_d}(z)$ only outputs the secret verification state τ_d.

Generator $\mathcal{G}(\lambda, T, n)$:

1. $\mathcal{G}$ samples a puncturable PRF key $K \leftarrow \mathsf{PRF{\cdot}Gen}(1^\lambda)$.
2. Let $C_\sigma[K]$ be a circuit that on input z, runs $r \leftarrow \mathsf{F}(K, z)$ and outputs $\sigma_d \leftarrow \mathcal{G}_{\sigma_d}(z; r)$. That is, C_σ runs $\mathcal{G}_{\sigma_d}$ to generate a first message of the delegation scheme, using randomness from the PRF key K. Similarly, $C_\tau[K]$ on input z runs $r \leftarrow \mathsf{F}(K, z)$ and outputs $\tau_d \leftarrow \mathcal{G}_{\tau_d}(z; r)$. $\mathcal{G}$ generates the circuits $C_\sigma[K]$ and $C_\tau[K]$, and pads them to be of size ℓ_σ and ℓ_τ respectively which will be specified exactly later in the analysis. For now, we mention that if we use a delegation scheme with optimal verification then $\ell_\sigma, \ell_\tau \leq \mathrm{poly}(\lambda, n, \log T)$. We subsequently assume the circuits C_σ and C_τ are padded.
3. $\mathcal{G}$ runs $\sigma \leftarrow i\mathcal{O}(1^\lambda, C_\sigma[K])$, $\tau \leftarrow i\mathcal{O}(1^\lambda, C_\tau[K])$ and outputs (σ, τ).

Prover $\mathcal{P}(z, \sigma)$:

$\mathcal{P}$ runs σ on input z to get $\sigma_d \leftarrow \sigma(z)$. Note that σ_d is a first message of the underlying delegation scheme $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$. Next, $\mathcal{P}$ generates the corresponding proof of the delegation scheme $\pi \leftarrow \mathcal{P}_d(z, \sigma_d)$ and outputs π.

Verifier $\mathcal{V}(z, \sigma, \tau, \pi)$:

$\mathcal{V}$ runs $\sigma_d \leftarrow \sigma(z)$, $\tau_d \leftarrow \tau(z)$, and outputs the result of $\mathcal{V}_d(z, \sigma_d, \tau_d, \pi)$.

Theorem 15. *If $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ is a privately verifiable (resp. publicly verifiable) P-delegation scheme, then $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ as described above is a privately verifiable (resp. publicly verifiable) SNARG with strong soundness. Moreover, if the delegation scheme has optimal or space-dependent verification and relative prover efficiency, then so does the SNARG.*

Proof. The completeness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ follows directly from that of $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ and the correctness of $i\mathcal{O}$. The running time of $\mathcal{G}(\lambda, T, n)$ is polynomial in λ and the maximum running time of $\mathcal{G}_d$ on inputs $z = (AL, \mathsf{inp}, \lambda)$ where $|AL, \mathsf{inp}| \leq n$ and $AL.T \leq T$. Similarly, the running times of $\mathcal{P}$ and $\mathcal{V}$ are polynomial in λ and the running times of $\mathcal{P}_d$ and $\mathcal{V}_d$ respectively. Therefore, the optimal (or space-dependent) verification and prover efficiency of $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ implies that the same properties hold for $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$.

To show strong soundness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$, we will rely on the soundness of $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$, and the security of $i\mathcal{O}$ and the punctured PRF $(\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F})$. We will first consider the privately verifiable setting. Assume for contradiction there exists poly-size oracle-aided prover $\mathcal{P}^*$, polynomials $T(\cdot)$,$n(\cdot)$,$p(\cdot)$ such that for infinitely many $z = (AL, \mathsf{inp}, \lambda)$, where $AL.T \leq$

$T(\lambda)$, $|AL, \mathsf{inp}| \leq n(\lambda)$, and AL does not accept inp:

$$\Pr\left[\mathcal{V}(z,\sigma,\tau,\pi)=1 \;\middle|\; \begin{array}{r}(\sigma,\tau)\leftarrow\mathcal{G}(\lambda,T(\lambda),n(\lambda))\\ \pi\leftarrow\mathcal{P}^{*\mathcal{V}(\cdot,\sigma,\tau,\cdot)}(z,\sigma)\end{array}\right]\geq\frac{1}{p(\lambda)}\ .$$

We will refer to the above probability as the advantage $\mathcal{A}(z,\mathcal{P}^*)$. We will now construct a malicious prover $\mathcal{P}_d^*$ to break the soundness of the delegation scheme. $\mathcal{P}_d^*$ gets as input z and σ_d which is some first message of the delegation scheme. $\mathcal{P}^*$ runs a subroutine $\mathcal{D}$ described in the following paragraph, on input (z,σ_d), to obtain a "fake" SNARG message and verification state (σ,τ) which it will then use to run $\mathcal{P}^*$ and answer its queries. That is, $\mathcal{P}_d^*$ runs $(\sigma,\tau)\leftarrow\mathcal{D}(z,\sigma_d)$, $\pi\leftarrow\mathcal{P}^{*\mathcal{V}(\cdot,\sigma,\tau,\cdot)}(z,\sigma)$ and outputs π. The subroutine $\mathcal{D}$ is defined as follows:

Subroutine $\mathcal{D}(z,\sigma_d)$:

1. $\mathcal{D}$ samples a puncturable PRF key $K\leftarrow\mathsf{PRF{\cdot}Gen}(1^\lambda)$ and punctures it at the input z to obtain a punctured key $K_z\leftarrow\mathsf{PRF{\cdot}Punc}(K,z)$.
2. Let $C_\sigma^*[K_z,\sigma_d]$ be a circuit that on input z^* behaves as follows: if $z^*=z$ then C_σ^* simply outputs the hardwired value σ_d. Otherwise, C_σ^* runs $r\leftarrow\mathsf{F}(K_z,z^*)$ and outputs the result of $\mathcal{G}_{\sigma_d}(z^*;r)$.
3. Similarly, let $C_\tau^*[K_z]$ be a circuit that on input z^* behaves as follows: if $z^*=z$ then C_τ^* simply outputs $\perp$. Otherwise, C_τ^* runs $r\leftarrow\mathsf{F}(K_z,z^*)$ and outputs the result of $\mathcal{G}_{\tau_d}(z^*;r)$.
4. $\mathcal{D}$ generates the circuits $C_\sigma^*[K_{z^*},\sigma_d]$ and $C_\tau^*[K_{z^*}]$ and pads them to sizes ℓ_σ and ℓ_τ respectively, where ℓ_σ is the maximum size of the circuits $C_\sigma^*[K_{z^*},\sigma_d^*]$ and $C_\sigma[K]$ and ℓ_τ is the maximum size of the circuits $C_\tau^*[K_{z^*}]$ and $C_\tau[K]$. We subsequently assume the circuits C_σ^* and C_τ^* are padded.
5. $\mathcal{D}$ generates $\sigma\leftarrow i\mathcal{O}(1^\lambda,C_\sigma^*[K_z,\sigma_d])$, $\tau\leftarrow i\mathcal{O}(1^\lambda,C_\tau^*[K_z])$ and outputs (σ,τ).

Note that when $\mathcal{P}_d^*$ uses τ, as generated by $\mathcal{D}$ above, to answer $\mathcal{P}^*$'s verification oracle queries on the input z then, unlike a "real" verification state, τ simply outputs $\perp$. In this case, $\mathcal{P}_d^*$ answers the query with the bit 0 (rejecting the proof submitted in the query).

We now analyze the success probability of $\mathcal{P}_d^*$. We want to show there exists a polynomial p' such that for infinitely many $z = (AL, \mathsf{inp}, \lambda)$ where AL does not accept inp the following holds:

$$\mathcal{A}_d(z, \mathcal{P}_d^*) = \Pr\left[\mathcal{V}_d(z, \sigma_d, \tau_d, \pi) = 1 \;\middle|\; \begin{array}{c}(\sigma_d, \tau_d) \leftarrow \mathcal{G}_d(z) \\ \pi \leftarrow \mathcal{P}_d^*(z, \sigma_d)\end{array}\right] \geq \frac{1}{p'(\lambda)} \quad .$$

Let $\mathcal{Z}$ be the sequence of such $z = (AL, \mathsf{inp}, \lambda)$.

To show $\mathcal{P}_d^*$ succeeds with noticeable probability, we will consider a hybrid malicious prover $\mathcal{P}_d^{\mathsf{Hyb}}$ that is very similar to $\mathcal{P}_d^*$ except that it also gets the secret verification state τ_d as input and uses it in a different subroutine $\mathcal{D}^{\mathsf{Hyb}}$. We will first show that for every $z \in \mathcal{Z}$, $\mathcal{A}_d(z, \mathcal{P}_d^*) = \mathcal{A}_d(z, \mathcal{P}_d^{\mathsf{Hyb}})$. Next, we show that relying on the security of the indistinguishability obfuscator and the puncturable PRF, $\mathcal{A}_d(z, \mathcal{P}_d^{\mathsf{Hyb}})$ is negligibly close to $\mathcal{A}(z, \mathcal{P}^*)$ for all $z \in \mathcal{Z}$. By assumption, $\mathcal{A}(z, \mathcal{P}^*)$, is noticeable and hence we have that $\mathcal{A}_d(z, \mathcal{P}_d^*)$ is noticeable, contradicting the soundness of the P-delegation scheme.

We now describe the hybrid malicious prover $\mathcal{P}_d^{\mathsf{Hyb}}$. $\mathcal{P}_d^{\mathsf{Hyb}}$ gets as input z and both σ_d and τ_d. It uses the hybrid subroutine $\mathcal{D}^{\mathsf{Hyb}}$ on input (z, σ_d, τ_d) to generate a hybrid "fake" (σ, τ) to run $\mathcal{P}^*$ and answer its queries. However, unlike $\mathcal{P}_d^*$, it uses τ to answer *all* of $\mathcal{P}^*$'s queries (including those on input z). $\mathcal{D}^{\mathsf{Hyb}}$ is defined as follows.

Subroutine $\mathcal{D}^{\mathsf{Hyb}}(z, \sigma_d, \tau_d)$:

1. $\mathcal{D}^{\mathsf{Hyb}}$ samples K_z and generates σ exactly as in $\mathcal{D}$. The only difference is in the generation of τ.
2. Let $C^*_\tau[K_z, \tau_d]$ be a circuit that on input z^* behaves as follows: if $z^* = z$ then C^*_τ simply outputs the hardwired value τ_d. Otherwise, C^*_τ runs $r \leftarrow \mathsf{F}(K_z, z^*)$ and outputs the result of $\mathcal{G}_{\tau_d}(z^*; r)$.
3. $\mathcal{D}^{\mathsf{Hyb}}$ generates $C^*_\tau[K_z, \tau_d]$, pads it to the maximum size of $C^*_\tau[K_z, \tau_d]$ and $C_\tau[K]$ and generates $\tau \leftarrow i\mathcal{O}(1^\lambda, C^*_\tau[K_z, \tau_d])$. $\mathcal{D}^{\mathsf{Hyb}}$ outputs (σ, τ).

We now observe that for every $z \in \mathcal{Z}$, $\mathcal{A}_d(z, \mathcal{P}^*_d) = \mathcal{A}_d(z, \mathcal{P}^{\mathsf{Hyb}}_d)$. The only difference in the two experiments is in the view of $\mathcal{P}^*$: when run by $\mathcal{P}^*_d$ then its oracle responses are answered using τ as generated by $\mathcal{D}$ and when run by $\mathcal{P}^{\mathsf{Hyb}}_d$, its oracle responses are answered using τ as generated by $\mathcal{D}^{\mathsf{Hyb}}$. However, we claim that the responses are distributed identically in both cases. They could only potentially differ on queries on the input z, but since z is a "false" input, *i.e.* AL does not accept inp, in both cases, the verification oracle response on such queries is 0 (reject).

Next we show that there is a negligible function $\alpha(\cdot)$ such that for every $z \in \mathcal{Z}$,

$$|\mathcal{A}_d(z, \mathcal{P}^{\mathsf{Hyb}}_d) - \mathcal{A}(z, \mathcal{P}^*)| \leq \alpha(\lambda)$$

. We first observe that in the experiment corresponding to $\mathcal{A}_d(z, \mathcal{P}^{\mathsf{Hyb}}_d)$, the event $\mathcal{V}_d(z, \sigma_d, \tau_d, \pi) = 1$ is equivalent to the event $\mathcal{V}(z, \sigma, \tau, \pi) = 1$ where $(\sigma, \tau) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z, \sigma_d, \tau_d)$. This follows directly from the construction of $\mathcal{V}$ and the fact that

σ and τ are hardwired to output σ_d and τ_d on input z. Hence we have that

$$\mathcal{A}_d(z, \mathcal{P}_d^{\mathsf{Hyb}}) = \Pr\left[\mathcal{V}(z,\sigma,\tau,\pi) = 1 \;\middle|\; \begin{array}{r} (\sigma_d,\tau_d) \leftarrow \mathcal{G}_d(z) \\ (\sigma,\tau) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z,\sigma_d,\tau_d) \\ \pi \leftarrow \mathcal{P}^{*\mathcal{V}(\cdot,\sigma,\tau,\cdot)}(z,\sigma) \end{array}\right].$$

Now viewed this way, we can observe that the only difference between the above experiment and that of $\mathcal{A}(z, \mathcal{P}^*)$ is in how (σ, τ) are generated. In the above experiment, (σ, τ) comes from $\mathcal{D}^{\mathsf{Hyb}}$ and $\mathcal{G}_d$ whereas in the experiment for $\mathcal{A}(z, \mathcal{P}^*)$, (σ, τ) comes from $\mathcal{G}$. It suffices to show the following claim:

Claim 2. *The following ensembles are computationally indistinguishable.*

$$\{(\sigma,\tau) : (\sigma,\tau,\pi) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z,\sigma_d,\tau_d), (\sigma_d,\tau_d) \leftarrow \mathcal{G}_d(z)\}_{z\in\mathcal{Z}} \tag{2.7}$$

$$\approx_c \{(\sigma,\tau) : (\sigma,\tau) \leftarrow \mathcal{G}(\lambda, T(\lambda), n(\lambda))\}_{z\in\mathcal{Z}} \tag{2.8}$$

Proof. Recall that in ensemble (2.7), $\sigma \leftarrow i\mathcal{O}(C^*_\sigma[K_z, \sigma_d])$ where K_z is a PRF key punctured at input z and C^*_σ on all input z outputs σ_d and on all other inputs z^* outputs $\mathcal{G}_d(z^*; \mathsf{F}(K_z, z^*))$. However, in ensemble (2.8), $\sigma \leftarrow i\mathcal{O}(C_\sigma[K])$ where C on input z^* outputs $\mathcal{G}_d(z^*; \mathsf{F}(K, z^*))$. The difference between τ in ensembles (2.7) and (2.8) is the same. Indistinguishability follows from the security of $i\mathcal{O}$ and that of $(\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F})$ in the standard way. We provide a brief overview.

Consider a hybrid ensemble that is identical to ensemble (2.7) except that instead of uniform randomness $\mathcal{G}_d$ uses randomness from $\mathsf{F}(K, z)$ where K is a PRF key. K is then punctured at input z and given to $\mathcal{D}^{\mathsf{Hyb}}$ to use as K_z. By the security of the punctured PRF, this hybrid ensemble is indistinguishable from ensemble (2.7). Furthermore, the circuits obfuscated as σ and τ in this hybrid ensemble and in ensemble (2.8) are functionally equivalent. Hence, by

the security of $i\mathcal{O}$, ensemble (2.8) is indistinguishable from the hybrid ensemble. A hybrid argument completes the proof of the claim. □

This completes the proof of strong soundness in the privately-verifiable setting. Proving strong soundness in the publicly-verifiable setting is very similar . The malicious prover for the SNARG $\mathcal{P}^*$ now also requires τ as input to generate the convincing proof π. On the other hand the prover we want to construct for the delegation scheme $\mathcal{P}_d^*$ gets τ_d as input from the challenger. $\mathcal{P}_d^*$ uses the same strategy as $\mathcal{P}_d^{\mathsf{Hyb}}$ to generate τ and simply gives it to $\mathcal{P}^*$. Using the same proof as above, we have that if $\mathcal{P}^*$ succeeds with noticeable probability then so does $\mathcal{P}_d^*$. □

Succinct Perfect NIZK for NP

In this section we briefly note that using the succinct indistinguishability obfuscator from Section 2.5.1 in the construction of [134] we can obtain a NIZK argument scheme for any NP language $\mathcal{R}_L$ that is perfect zero-knowledge and additionally *succinct* in the following sense: Let $\Pi^{\mathcal{R}}$ be a uniform programs that computes the NP relation $\mathcal{R}(x, w)$, and let $\tau(n)$ and $s(n)$ be respectively bounds on the length of witness and space needed by $\Pi^{\mathcal{R}}$ for n-bit statements. The length of the CRS of the scheme for proving n-bit statements grows polynomially with n, $\tau(n)$, and $s(n)$, (and is essentially independent of the verification time of the language). Below We provide a brief overview of the [134] construction and how it can be made succinct using succinct indistinguishability obfuscation.

In [134], the NIZK scheme relies on indistinguishability obfuscation for circuits as follows: the CRS contains an obfuscation of two circuits that contain the

same PRF key. The first obfuscation is used by the Prover to generate proofs: the circuit takes as input a statement and witness (x, w) of lengthes n and $\tau(n)$, and outputs the image of the input under the PRF as the proof if the witness is valid, that is, $\Pi^{\mathcal{R}}(x, w) = 1$. The second obfuscation is used by the Verifier to check if this proof is valid. [134] show how to use this idea relying on indistinguishability obfuscation and puncturable PRFs. In their construction, the length of the proof is succinct: it depends only on the security parameter. However, the length of the CRS is related to the size of the circuits obfuscated in the CRS, which is related to the verification time. We note that by obfuscating the pair of Turing machines that perform the above functionality, and using our succinct indistinguishability obfuscator instead, the length of the CRS can be made to depend on the statement and witness lengthes, as well as the space complexity of the verification program, independent of the verification time.

Note that this succinct construction relies on our succinct indistinguishability obfuscator which in turn relies on sub-exponentially secure *iO* for circuits (as opposed to standard IO for circuits which the [134] construction is based on).

Theorem 16. *(Follows from [134]) Assuming sub-exponentially secure iO for circuits and sub-exponentially secure OWFs, there exists a NIZK argument scheme for every NP language determined by a uniform polynomial-time program $\Pi^{\mathcal{R}}$ with the following properties:*

1. *The scheme is perfectly zero knowledge.*
2. *The scheme has adaptive soundness*[16].

[16]The perfect NIZK construction of [134] only satisfies non-adaptive soundness. But by a standard complexity leveraging trick, it can be made to satisfy adaptive soundness. Since we anyway assume sub-exponential security of the *iO* this comes at no cost for us.

3. *There are universal polynomials p, p' and p'', such that the length of the CRS of the scheme for verifying statements x of length n is $p(\lambda, n, \tau(n), s(n))$, where λ is the security parameter, $\tau(n)$ is a bound on the length of witness, and $s(n)$ is the space complexity of $\Pi^{\mathcal{R}}$ for verifying n-bit statements. The length of the proof is $p'(\lambda)$. The run-time of the prover for statement and witness (x, w) is $p''(\lambda, n, \tau(n), s(n))T$, where T is the run-time of $\Pi^{\mathcal{R}}(x, w)$.*

Remark 6 (Improved Efficiency of Delegation and $\mathcal{SNARG}$s)**.** *When plugging in the more efficient garbling scheme of Remark 2, we directly obtain a publicly verifiable delegation scheme that the verification complexity is* $\mathrm{poly}(\lambda)(|AL| + AL.S + |x|)$ *and the prover complexity for a T-time computation $AL(x)$ is* $\mathrm{poly}(\lambda)(|AL| + AL.S + T)$*. Furthermore, combining this delegation scheme with Theorem 15, while applying the trick of "obfuscating in a piecemeal fashion" as in Section 2.4.1 and 2.5.1, we obtain a $\mathcal{SNARG}$ for $\mathcal{P}$ with the same verification and prover efficiency.*

Finally, using the more efficiency succinct $i\mathcal{O}$ of Remark 3 in Theorem 16, the size of the CRS can be improved to $\mathrm{poly}(\lambda, n, \tau(n)) \cdot s(n)$*, and the prover efficiency can be improved to* $\mathrm{poly}(\lambda, n, \tau(n))(s(n) + T)$*.*

Acknowledgements

We thank Boaz Barak and Guy Rothblum for their input regarding the plausibility of interactive proofs with fast verification (relevant to the plausibility of perfectly-private succinct randomized encodings). We thank Daniel Wichs for discussing several aspects of [83]. We thank Stefano Tessaro for many delightful discussions at the early stage of the project. Finally we thank the anonymous reviewers of STOC for their valuable comments.

2.6 Obfuscating Circuits with Quasi-Linear Blowup

In general, when considering iO for circuits, the size of an obfuscation $|iO(C)|$ (or more generally the time to required to obfuscate) is allowed to be an arbitrary polynomial in the original circuit-size $|C|$. In known candidate constructions (e.g., [70]) the blow-up is quadratic (see discussion in [78]). In this section, we show how to construct from iO and one-way functions, iO for circuits where the blowup is quasi-linear.

The high-level idea. The transformation relies on similar ideas to those used in Section 2.5.1 to construct succinct iO from succinct randomized encodings, which in turn go back to the bootstrapping technique of Applebaum [7]. Concretely, we rely on plain randomized encodings [102, 11] for circuits are known to have the following basic locality property: given a circuit C with s gates and n-bit input x, computing a randomized encoding $\widehat{C}(x;R)$ can be decomposed into s computations $\widehat{C}_1(x;R),\ldots,\widehat{C}_s(x;R)$, each of fixed size ℓ independent of the circuit size $|C|$. In particular, each such computation $\widehat{C}_i(x;R)$ involves at most ℓ bits of the shared randomness R.

Similarly to the transformation in Section 2.5.1, the transformation here is based on the basic idea of obfuscating the circuit that computes the randomized encoding $\widehat{C}(x;r)$ for any input x, while deriving the randomness R, by applying a puncturable PRF to the input x. The only difference is that, rather than obfuscating this circuit as a whole, we separately obfuscate s smaller circuits computing the corresponding $\widehat{C}_i(x;R)$. To make sure that deriving the randomness is also local, we associate $r = |R|$ PRF keys $K_1,\ldots,K_r$ with each of the bits of the shared randomness R. Each one of the s obfuscated circuits only includes the PRF keys

required for its local computation. The corresponding bits of randomness are again derived by applying the corresponding PRFs to the input x.

The gain is that the size of the resulting obfuscated circuit is thus $s \cdot \text{poly}(\lambda, n)$ as required. The proof of security relies on a variant of the probabilistic *iO* argument invoked in Section 2.5.1, with the difference that puncturing is performed simultaneously across all r PRF keys. Accordingly, it incurs a 2^n security loss in the input length n (which is polynomial when considering circuits with logarithmic-size inputs, as is often the case in this work).

We next describe the transformation in more detail, and sketch the proof of security. We start by defining the required notion of locality for the randomized encoding.

Definition (Locality of Randomized Encodings)**.** *A randomized encoding* $\mathcal{RE} = (\mathsf{RE}, \mathsf{Dec})$ *for circuits is said to be local if*

$$\mathsf{RE}(1^\lambda, C, x; R) = \widehat{C}_1(x; R|_{S_1}), \ldots, \widehat{C}_s(x; R|_{S_s}) \ ,$$

where $s = \Theta(|C|)$, $S_i \subseteq \{1, \ldots, |R|\}$, $R|_{S_i}$ *is the restriction of* R *to* S_i*, and the following properties are satisfied:*

- $\widehat{C}_i$ *is a circuit of fixed size* $\ell(\lambda, |x|) = \text{poly}(\lambda, |x|)$*, independent of* $|C|$.
- *The circuits* $\left\{\widehat{C}_i\right\}$ *and sets* $\{S_i\}$ *can be computed from* C *in time* $|C| \cdot \text{poly}(\lambda, |x|)$.
- *Decoding can be done in time* $|C| \cdot \text{poly}(\lambda, |x|)$.

Such randomized encodings can be constructed based on any one-way function [138, 11].

A qausi-linear obfuscator $i\mathcal{O}^*$. we now describe the new obfuscator. The obfuscator relies on the following building blocks:

- A randomized encoding $\mathcal{RE} = (\mathsf{RE}, \mathsf{Dec})$ for circuits that is local and which used randomness of length at most $r = r(|C|, \lambda)$.
- An indistinguishability obfuscator $i\mathcal{O}$ for circuits (with arbitrary polynomial blowup).
- A puncturable PRF $(\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F})$.

All building blocks are assumed to be $2^{-n+\omega(\log \lambda)}$-secure.

The obfuscator $i\mathcal{O}^*(1^\lambda, C)$ proceeds as follows:

1. Compute the circuits $\widehat{C}_1(\cdot;\cdot), \ldots, \widehat{C}_s(\cdot;\cdot)$ and sets $S_1, \ldots, S_s$.
2. Sample PRF keys $K_1, \ldots, K_r \leftarrow \mathsf{PRF{\cdot}Gen}(1^\lambda)$.
3. For each $i \in [s]$, obfuscate using $i\mathcal{O}$ the circuit $\mathbb{C}_i$ that has hardwired $\left\{K_j : j \in S_i\right\}$ and given x operates as follows:
   - Derive randomness $R|_{S_i}$ by invoking $\mathsf{F}_{K_j}(x)$ for $j \in S_i$.
   - Output $\widehat{C}_i(x, R|_{S_i})$.

   The circuit is further padded to be of total size is $\ell(\lambda, x)$, where ℓ is determined in the analysis.
4. Output the obfuscations $i\mathcal{O}(\mathbb{C}_1), \ldots, i\mathcal{O}(\mathbb{C}_s)$.

To evaluate $i\mathcal{O}^*(1^\lambda, C)$ on input x, first evaluate each $i\mathcal{O}(\mathbb{C}_i)$ on x, obtain the randomized encoding

$$\widehat{C}(x) = \widehat{C}_1(x; R|_{S_1}), \ldots, \widehat{C}_s(x; R|_{S_s}) \ ,$$

end decode to obtain the result $C(x)$.

Proposition. *$i\mathcal{O}^*$ is a circuit obfuscator with qausi-linear blowup.*

Proof sketch. The functionality of $i\mathcal{O}^*$ follows directly from the functionality of the underlying $i\mathcal{O}$ and the correctness of decoding. The fact that the size $|i\mathcal{O}^*(1^\lambda, C)|$ obfuscated circuit is $|C| \cdot \text{poly}(\lambda, |x|)$, follows from the locality of the randomized encoding. We next sketch the security.

We use a probabilistic $i\mathcal{O}$ argument similar to the one used in Section 2.5.1. Concretely, given two circuits C, C' of the same size and functionality, we consider $2^n + 1$ hybrids that transition from $i\mathcal{O}^*(1^\lambda, C)$ to $i\mathcal{O}^*(1^\lambda, C')$. In the j^{th} hybrid, the s obfuscations are with respect to hybrid circuits $\mathbb{C}_1^j, \ldots, \mathbb{C}_s^j$ where $\mathbb{C}_i^j$ uses $\widehat{C}_i$ for all inputs $x < j$ and $\widehat{C'}_i$ for all inputs $x \geq j$. Each two consecutive hybrids only differ on a single point j. Similarly to Section 2.5.1, we puncture the underlying PRFs at this point j, and hardwire the values $\widehat{C}_1(x; R|_{S_1}), \ldots, \widehat{C}_s(x; R|_{S_s})$ (or $\widehat{C'}_1(x; R|_{S_1}), \ldots, \widehat{C'}_s(x; R|_{S_s})$, respectively), using true randomness instead of pseudo-randomness. Then we can rely on the security of the randomized encodings to switch between the two.

The padding parameter $\ell(\lambda, |x|)$ is chosen to account for the above hybrids (and only induces quasi-linear blowup). □

We describe circuits $\mathbb{M}_1^\gamma$ to $\mathbb{M}_6^\gamma$. They all have parameters $\lambda, S, M, T^*, y, \text{conf}_{\gamma+1}$ hardwired in; for simplicity, we suppress these parameters in the superscript.

Circuit $\mathbb{M}_1^\gamma = (\mathbb{M}_1^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$**:** On input $t \in [T]$, does:

If $t \neq \gamma$, generate pseudo-random string $\alpha_{t+1} = \mathsf{F}(K_\alpha(\gamma+1), t+1)$.

If $t \neq \gamma+1$, generate pseudo-random strings $\alpha_{t+1} = \mathsf{F}(K_\alpha(\gamma+1), t)$ and $\beta_t = \mathsf{F}(K_\beta(\gamma+1), t)$.

Proceed as $\mathbb{M}^\gamma$ does using random coins $\alpha_t, \alpha_{t+1}, \beta_t$.

Circuit $\mathbb{M}_2^\gamma = (\mathbb{M}_2^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$**:**

Identical to $(\mathbb{M}_1^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$, with $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$ sampled at random.

Circuit $\mathbb{M}_3^\gamma = (\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widehat{\mathbf{C}}_{\gamma+1}, \widehat{\text{conf}}_{\gamma+1}}$**:** On input $t \in [T]$, does:

If $t = \gamma+1$, output $\widehat{\mathbf{C}}_{\gamma+1}$.

If $t = \gamma$, set out_γ using $\widehat{\text{conf}}_{\gamma+1}$ as in Step 1 of program $\mathbf{R}$; simulate and output $\widetilde{\mathbf{C}}_\gamma$ as in Step 2 of $\mathbf{R}$.

Otherwise, compute as $\mathbb{M}_2^\gamma$ does using the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$.

Circuit $\mathbb{M}_4^\gamma = (\mathbb{M}_4^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$**:**

Identical to $(\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$, with simulated garbling pair $\widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}$.

Circuit $\mathbb{M}_5^\gamma = (\mathbb{M}_5^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$**:** On input $t \in [T]$, does:

If $t = \gamma+1$, compute $\widetilde{\mathbf{C}}_{\gamma+1}$ using program $\mathbf{R}$ with randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$.

If $t = \gamma$, compute $\widetilde{\mathbf{C}}_\gamma$ using program $\mathbf{Q}$, which internally computes $\widetilde{\text{conf}}_{\gamma+1}$ for setting the output out_γ using randomness $\alpha'_{\gamma+1}$.

Otherwise, compute as $\mathbb{M}_4^\gamma$ does using the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$.

Circuit $\mathbb{M}_6^\gamma = (\mathbb{M}_6^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$**:**

Identical to $(\mathbb{M}_5^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$, with $\alpha_{\gamma+1} = \mathsf{F}(K_\alpha, \gamma+1)$, $\beta_{\gamma+1} = \mathsf{F}(K_\beta, \gamma+1)$

The circuits in Figure 2.1, 2.2 and 2.3 are padded to their maximum size.

Figure 2.3: Circuits used in the security analysis of $\mathcal{GS}$, continued

CHAPTER 3

INDISTINGUISHABILITY OBFUSCATION FROM SEMANTICALLY-SECURE MULTILINEAR ENCODINGS

This chapter contains joint work with Rafael Pass (Cornell University) and Karn Seth (Cornell University).

3.1 Introduction

The goal of *program obfuscation* is to "scramble" a computer program, hiding its implementation details (making it hard to "reverse-engineer"), while preserving the functionality (i.e, input/output behavior) of the program. Precisely defining what it means to "scramble" a program is non-trivial: on the one hand, we want a definition that can be plausibly satisfied, on the other hand, we want a definition that is useful for applications.

A first formal definition of such program obfuscation was provided by Hada [98]: roughly speaking, Hada's definition—let us refer to it as *strongly virtual black-box*—is formalized using the simulation paradigm. It requires that anything an attacker can learn from the obfuscated code, could be simulated using just black-box access to the functionality.[1] Unfortunately, as noted by Hada, only learnable functionalities can satisfy such a strong notion of obfuscation: if the attacker simply outputs the code it is given, the simulator must be able to recover the code by simply querying the functionality and thus the functionality must be learnable.

[1] Hada actually considered a slight distributional weakening of this definition.

An in-depth study of program obfuscation was initiated in the seminal work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [17]. Their central result shows that even if we consider a more relaxed simulation-based definition of program obfuscation—called *virtual black-box (VBB)* obfuscation—where the attacker is restricted to simply outputting a single bit, impossibility can still be established.[2] Their result is even stronger, demonstrating the existence of families of functions such that given black-box access to f_s (for a randomly chosen s), not even a *single* bit of s can be guessed with probability significantly better than $1/2$, but given the code of any program that computes f_s, the entire secret s can be recovered. Thus, even quite weak simulation-based notions of obfuscation are impossible.

But weaker notions of obfuscation may be achievable, and may still suffice for (some) applications. Indeed, Barak *et al.* [17] also suggested two such notions:

- The notion of *indistinguishability obfuscation*, first defined by Barak *et al.* [17] and explored by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [70], roughly speaking 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.
- The notion of *differing-input obfuscation*, first defined by Barak *et al.* [17] and explored by Boyle, Chung and Pass [42] and by Ananth, Boneh, Garg, Sahai and Zhandry [2] strengthens the notion of indistinguishability obfuscation to also require that even if C_1 and C_2 are not equivalent circuits, if an attacker can distinguish obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$, then the at-

[2] A similar notion of security (without referring to obfuscation) was considered even earlier by Canetti [53] in the special case of what is now referred to as *point-function obfuscation*.

tacker must "know" an input x such that $C_1(x) \neq C_2(x)$, and this input can be efficiently "extracted" from the attacker.

In a very recent breakthrough result, Garg, Gentry, Halevi, Raykova, Sahai, and Waters [70] provided the first candidate constructions of indistinguishability obfuscators for all polynomial-size circuits, based on so-called *multilinear (a.k.a. graded) encodings* [38, 133, 66]—for which candidate constructions were recently discovered in the seminal work of Garg, Gentry and Halevi [66], and more recently, alternative constructions were provided by Coron, Lepoint and Tibouchi [61].

The obfuscator construction of Garg et al proceeds in two steps. They first provide a candidate construction of an indistinguishability obfuscator for NC^1 (this construction is essentially assumed to be secure); next, they demonstrate a "bootstrapping" theorem showing how to use fully homomorphic encryption (FHE) schemes [79] and indistinguishability obfuscators for NC^1 to obtain indistinguishability obfuscators for all polynomial-size circuits. Further constructions of obfuscators for NC^1 were subsequently provided by Brakerski and Rothblum [50] and Barak, Garg, Kalai, Paneth and Sahai [16]—in fact, these constructions achieve the even stronger notion of virtual-black-box obfuscation in idealized "generic" multilinear encoding models. Additionally, Boyle, Chung and Pass [42] present an alternative bootstrapping theorem, showing how to employ differing-input obfuscations for NC^1 to obtain differing-input (and thus also indistinguishability) obfuscation for both circuits and Turing machines. (Ananth et al [2] also provide Turing machine differing-input obfuscators, but start instead from differing-input obfuscators for polynomial-size circuits.)

In parallel with the development of candidate obfuscation constructions,

several surprising applications of both indistinguishability and differing-input obfuscations have emerged (see e.g., [70, 134, 99, 41, 69, 42], and more recently [28, 46, 88, 111, 110]). Most notable among these is the work of Sahai and Waters [134] (and the "punctured program" paradigm it introduces) which shows that for some interesting applications of virtual black-box obfuscation (such as turning private-key primitives into public-key one), the weaker notion of indistinguishability obfuscation suffices. Furthermore, as shown by Goldwasser and Rothblum [96], indistinguishability obfuscators provide a very nice "best-possible" obfuscation guarantee: if a functionality can be VBB obfuscated (even non-efficiently!), then any indistinguishability obfuscator for this functionality is VBB secure. Finally, as shown by Boyle, Chung and Pass [42] indistinguishability obfuscation in fact implies a notion of differing-input obfuscation (when restricted to programs that differ on polynomially-many inputs); and this notion already suffices for some applications of differing-input obfuscation (see e.g., [23], [51], [52]).

3.1.1 Towards "Provably-Secure" Obfuscation

But despite these amazing developments, the following question remains open:

> *Can the security of general-purpose indistinguishability obfuscator be reduced to some "natural" intractability assumption?*

The principal goal of the current paper is to make progress toward addressing this question. Note that while the construction of indistinguishability obfuscation of Garg et al is based on *some* intractability assumption, the assumption

is very tightly tied to their scheme—in essence, the assumption stipulates that their scheme is a secure indistinguishability obfuscator. The VBB constructions of Brakerski and Rothblum [50] and Barak et al [16] give us more confidence in the plausible security of their obfuscators, in that they show that at least "generic" attacks—that treat multilinear encoding as if they were "physical envelopes" on which multilinear operations can be performed—cannot be used to break security of the obfuscators. But at the same time, non-generic attacks against their scheme are known—since general-purpose VBB obfuscation is impossible. Thus, it is not clear to what extent security arguments in the generic multilinear encoding model should make us more confident that these constructions satisfy e.g., a notion of indistinguishability obfuscation. In particular, the question of to what extent one can capture "real-world" security properties from security proofs in the generic model through a "meta-assumption" (regarding multilinear encodings) was raised (but not investigated) in [16]; see Remark 1 there.

In this work, we initiate a study of the above-mentioned question:

- We are concerned with the question of whether some *succinct* and *general* assumption (that is interesting in its own right, and is not "tailored" to a particular obfuscation construction) about some *low-level primitive* for which candidate constructions are known (e.g., multilinear encodings), can be used to obtain indistinguishability obfuscation.
- More importantly, we are interested in *reducing* the security of the obfuscation to some *simpler* assumption, not just in terms of "description size" but in terms of computational complexity—that is, we are not interested in assumptions that "directly" (without any security reduction) imply security

of the obfuscation.

- Finally, ideally, we would like the assumption to be *efficiently falsifiable* [123], so that it is possible to efficiently check whether the assumption is broken. This is particularly pressing since the assumption that a particular scheme (e.g., one of the schemes of [70, 50, 16]) is an indistinguishability obfuscator is not an efficiently falsifiable assumption, making it hard to check whether they can be broken or not: a presumed attacker must exhibit two *functionally-equivalent* circuits C_1 and C_2 that it can distinguish obfuscations of; but checking whether two circuits are functionally equivalent may not be polynomial-time computable. (In fact, assuming the existence of indistinguishability obfuscation and one-way functions, it is easy to come up with a method to sample C_1, C_2, z such that with high probability $C_1(z) \neq C_2(z)$ (and thus, given z, we can easily distinguish obfuscations of them), yet the pair of circuits (C_1, C_2) are indistinguishable from a pair of functionally equivalent circuits.[3] Thus, there are "fake attacks" on indistinguishability obfuscation that cannot be efficiently distinguished from a real attack.)

3.1.2 Security of Multilinear (Graded) Encodings

Towards explaining the assumptions we consider, let us start by briefly recalling multilinear (a.k.a. graded) encoding schemes [66, 70]. Roughly speaking, such

[3] In particular, (mirroring the ideas from the lower bound for witness encryption of [75]), given a statement x, let C^x_b be an obfuscation of a circuit that given a witness w outputs b iff w is an NP-witness for the statement x (and $\perp$ otherwise). If x is false, then by the indistinguishability obfuscation property, (C^x_0, C^x_1) is indistinguishable from two obfuscations of the *same* constant $\perp$ function. This still holds even if we sample a true x (and its associated witness z) from a hard-on-the-average language (as long as we do not give z to the distinguisher). Yet given the trapdoor z, we can clearly distinguish C^x_0, C^x_1 and also obfuscations of them.

schemes enable anyone that has access to a *public parameter* pp and *encodings* $E_S^x = \mathsf{Enc}(x, S)$, $E_S^y = \mathsf{Enc}(y, S')$ of ring elements x, y under the sets $S, S' \subset [k]$ to *efficiently*:[4]

- compute an encoding $E_{S \cup S'}^{x \cdot y}$ of $x \cdot y$ under the set $S \cup S'$, as long as $S \cap S' = \emptyset$;
- compute an encoding E_S^{x+y} of $x + y$ under the set S as long as $S = S'$;
- compute an encoding E_S^{x-y} of $x - y$ under the set S as long as $S = S'$.

(Given just access to the public-parameter pp, generating an encoding to a particular element x may not be efficient; however, it can be efficiently done given access to the *secret parameter* sp.) Additionally, given an encoding E_S^x where the set S is the whole universe $[k]$—called the "target set"—we can efficiently check whether $x = 0$ (i.e., we can "zero-test" encodings under the target set $[k]$.) In essence, multilinear encodings enable computations of certain restricted set of arithmetic circuits (determined by the sets S under which the elements are encoded) and finally determine whether the output of the circuit is 0; we refer to these as the *legal arithmetic circuits*.

Semantical Security of Multilinear (Graded) Encodings The above description only explains the *functionality* of multilinear encodings, but does not discuss *security*. As far as we are aware, there have been two approaches to defining security of multilinear encodings. The first approach, initiated in [66], stipulates specific hardness assumptions closely related to the DDH assumption. The second approach instead focuses on *generic attackers* and assumes that the attacker does

[4]Just as [50, 16], we here rely on "set-based" graded encoding; these were originally called "generalized" graded encodings in [66]. Following [70, 16] (and in particular the notion of a "multilinear jigsaw puzzles" in [70]), we additionally enable anyone with the secret parameter to encode *any* elements (as opposed to just *random* elements as in [66]).

not get to see the actual encodings but instead can only access them through legal arithmetic circuits.

In this work, we consider the first approach, but attempt to capture a general *class* of algebraic "decisional" assumptions (such as the the graded DDH assumption of [66]) which hold against generic attackers (and as such, it can be viewed as a merge of the two approaches). In essence, our notion of (single-message) *semantical security* attempts to capture the intuition that encodings of elements m_0 and m_1 (under the set S) are indistinguishable in the presence of encodings of "auxiliary" elements $\vec{z}$ (under sets $\vec{T}$), as long as $m_0, m_1, \vec{z}$ are sampled from *any* "nice" distribution D; in the context of a graded DDH assumption, think of $\vec{z}$ as a vector of independent uniform elements, m_0 as the product of the elements in $\vec{z}$ and m_1 as an independent uniform element. We analogously consider stronger notions of *constant-message* and *multi-message* semantical security, where m_0, m_1 (and S) are replaced by either constant-length or arbitrary polynomial-length vectors $\vec{m}_0, \vec{m}_1$ of elements (and sets $\vec{S}$).

Defining what makes a distribution D "nice" turns out to be quite non-trivial: A first (and minimal) approach—similar to e.g., the uber assumption of [35] in the context of bilinear maps—would be to simply require that D samples elements $\vec{m}_0, \vec{m}_1, \vec{z}$ such that no generic attacker can distinguish $\vec{m}_0, \vec{z}$ and $\vec{m}_1, \vec{z}$. As we discuss in Section 3.1.3, the most natural formalization of this approach can be attacked assuming standard cryptographic hardness assumptions. The distribution D considered in the attack, however, is "unnatural" in the sense that encodings of $\vec{m}_b, \vec{z}$ actually leak information about $\vec{m}_b$ even to generic attackers (in fact, this information fully determines the bit b, it is just that it cannot be computed in polynomial time).

Our notion of a *valid* message distribution disallows such information leakage w.r.t. generic attacks. More precisely, we require that every (even unbounded-size) legal arithmetic circuit C is *constant* over $(m_b, \vec{z})$, $b \in \{0,1\}$ with overwhelming probability; that is, there exists some bit c such that with overwhelming probability over $m_0, m_1, \vec{z} \leftarrow D$, $C(m_b, \vec{z}) = c$ for $b \in \{0,1\}$ (recall that a legal arithmetic circuit needs to end with a zero-test and thus the output of the circuit will be either 0 or 1). We refer to any distribution D satisfying this property as being *valid*, and our formal definition of semantical security now only quantifies over such valid message distributions.

Obfuscation from Semantically-Secure Multilinear Encodings As a starting point, we observe that slight variants of the constructions of [50, 16] can be shown to satisfy indistinguishability obfuscation for NC^1 assuming *multi-message* semantically-secure multilinear encodings. In fact, any VBB secure obfuscation in the generic model where the construction only releases encodings of elements (as the constructions of [50, 16] do) satisfies indistinguishability obfuscation assuming a slight strengthening of multi-message semantical security where validity only considers *polynomial-size* (as opposed to arbitrary-size) legal arithmetic circuits:[5] let $\vec{m}_0$ denote the elements corresponding to an obfuscation of some program Π_0, and $\vec{m}_1$ the elements corresponding to an obfuscation of some functionally equivalent program Π_1. VBB security implies that all polynomial-size legal arithmetic circuits are constant with overwhelming probability over both $\vec{m}_0$ and $\vec{m}_1$ (as any such query can be simulated given black-box access to the functionality of the program), and thus encodings of $\vec{m}_0$ and $\vec{m}_1$ (i.e., obfuscations of Π_0 and Π_1) are indistinguishable. By slightly tweaking

[5] We thank Sanjam Garg for this observation.

the construction of [16] and the analysis[6], we can extend this to hold against *all* (arbitrary-size) legal arithmetic circuits, and thus indistinguishability of the encodings (which implies indistinguishability of the obfuscations) follows as a direct consequence of the multi-message security assumption.

While this observation does takes us a step closer towards basing the security of obfuscation on a simple, natural, assumption, it is unappealing in that the assumption itself directly implies the security of the scheme (without any security reduction); that is, it does not deal with our second desiderata of *reducing* security to a *simpler* assumption—in particular, simply assuming that the (slight variant of the) scheme of [16] is secure is a special case of the multi-message security assumption.

Our central result shows how to construct indistinguishability obfuscators for NC^1 based on the existence of *constant-message* semantically-secure multilinear encodings; in the sequel, we simply refer to such schemes as being semantically secure (dropping "constant-message" from the notation). Note that the constant-message restriction not only simplifes (and reduces the complexity) of the assumption, it also takes us a step closer to the more standard GDDH assumption. (As far as we know, essentially all DDH-type assumptions in "standard"/bilinear or multilinear settings consider a constant-message setting, stipulating indistinguishability of either a *single* or a *constant* number of elements in the presence of polynomially many auxiliary elements. It is thus safe to say that such constant-message indistinguishability assumptions are significantly better understood their multi-message counterpart.)

Theorem 3 (Informally stated)**.** *Assume the existence of semantically secure multi-*

[6]Briefly, we need to tweak the construction to ensure a "perfect" simulation property.

linear encodings. Then there exists an indistinguishability obfuscator for NC^1.

As far as we know, this is the first result presenting indistinguishability obfuscators for NC^1 based on any type of assumption with a "non-trivial" security reduction w.r.t. arbitrary nuPPT attackers.

The core of our result is a general technique for transforming any obfuscator for matrix branching programs that satisfies a weak notion of *neighboring-matrix* indistinguishability obfuscation—which roughly speaking only requires indistinguishability of obfuscations of branching programs that differ only in a constant number of matrices—into a "full-fledged" indistinguishability obfuscator. (We emphasize that this first result is unconditional—it does not pertain to any particular construction and does not rely on any computational assumptions—and we thus hope it may be interesting in its own right.) We next show how to adapt the construction of [16] and its analysis to satisfy neighboring-matrix indistinguishability obfuscation based on semantical secure multilinear encodings; on a high-level, the security analysis in the generic model is useful for proving that the particular message distribution we consider is "valid".[7]

If additionally assuming the existence of a leveled FHE [132, 79] with decryption in NC^1—implied, for instance, by the LWE assumption [47, 48]—this construction can be bootstrapped up to obtain indistinguishability obfuscators for all polynomial-size circuits by relying on the technique from [70].

Theorem 4 (Informally stated)**.** *Assume the existence of semantically secure multilinear encodings and a leveled FHE with decryption in* NC^1*. Then there exists indistin-*

[7] As we explain in more details later, to use our transformation, we need to deal with branching programs that satisfy a slightly more liberal definition of a branching program than what is used in earlier works. This is key reason why we need to modify the construction and analysis from [16].

guishability obfuscators for P/poly.

Semantical Security w.r.t. Restricted Classes of Distributions Our most basic notion of semantical security requires indistinguishability to hold w.r.t. to *any* "valid" message distribution. This may seem like a strong assumption. Firstly, such a notion can clearly not be satisfied by a *deterministic* encoding schemes (as envisioned in the original work of [38])—we can never expect encodings of 0 and 1 (under a non target set, and without any auxiliary inputs) to be indistinguishable. Secondly, even if we have a randomized encoding scheme in mind (such as the candidates of [66, 61]), giving the attacker access to encodings of *arbitrary* elements may be dangerous: As mentioned in [66], attacks (referred to as "weak discrete logarithm attacks") on their scheme are known in settings where the attacker can get access to "non-trivial" encodings of 0 under any *non-target* set $S \subset [k]$. (We mention that, as far as we know, no such attacks are *currently* known on the candidate construction of [61].)

For the purposes of the results in our paper, however, it suffices to consider a notion of semantical security w.r.t. *restricted classes of distributions* D. In particular, to deal with both of the above issues, we consider "high-entropy" distributions D that sample elements $\vec{m}_0, \vec{m}_1, \vec{z}$ such that 1) each individual element has high-entropy, and 2) any element, associated with a *non-target* set $S \subset [k]$, that can be obtained by applying "legal" algebraic operations to $(\vec{m_b}, \vec{z})$ (for $b \in \{0, 1\}$) has high-entropy (and thus is non-zero with overwhelming probability).[8] We refer to such message distributions as being *entropically valid*.

Basing Security on a Single Falsifiable Assumption The assumption that a

[8]Technically, by high-entropy, we here mean that the min-entropy is at least $\log |R| - O(\log \log |R|)$ where R is the ring associated with the encodings; that is, the min-entropy is "almost" optimal (i.e., $\log |R|$).

scheme satisfies semantical security may be viewed as an (exponential-size) *class* of algebraic "decisional" assumptions (or as a "meta"-assumption, just like the "uber assumption" of [35]): we have one assumption for each valid message distributions D. Indeed, to prove indistinguishability of obfuscations of two circuits C_0, C_1, we rely on an instance in this class that is a function of the circuits C_0, C_1—in the language of [75, 86], security is thus based on an "instance-dependent" assumption.

This view-point also clarifies that semantical security is not an *efficiently falsifiable* assumption [123]: the problem is that there may not exist an efficient way of checking whether a distribution D is valid (as this requires checking that *all* legal arithmetic circuits are constant with overwhelming probability, which in our particular case would require checking whether C_0 and C_1 are functionally equivalent).

We finally observe that both of these issues can be overcome if we make subexponential hardness assumptions: there exists a single (uniform PPT samplable) distribution Sam over nuPPT message distributions D that are *provably* entropically valid such that it suffices to assume the existence of an encoding scheme that is entropic semantically secure w.r.t., this particular distribution with *subexponentially small indistinguishability gap*.[9] Note that this is a single, non-interactive and efficiently falsifiable, decisional assumption.

[9]These results were added to our e-Print report April 25, 2014, motivated in part by [86] (which bases witness encryption [75] on an instant-independent assumption) and a question asked by Amit Sahai.

3.1.3 Alternative Security Notions for Multilinear Encodings

We finally investigate various ways of defining a "super" (or uber) assumption for multilinear encodings. As mentioned above, a natural way of defining security of multilinear encodings would be to require that for specific classes of problems, generic attacks cannot be beaten (this is the approach alluded to in [16]). Perhaps the most natural instantiation of this in the context of a multilinear DDH assumption would be to require that for any distribution D over $\vec{m}_0, \vec{m}_1, \vec{z}$ (where $\vec{m}_0, \vec{m}_1$ are constant-length sequences), if encodings of $\vec{m}_0, \vec{z}$ and and $\vec{m}_0, \vec{z}$ are indistinguishable w.r.t. to generic attackers, then they are also indistinguishable w.r.t. arbitrary nuPPT attackers; in essence, "if an algebraic decisional assumption holds w.r.t. to generic attacks, then it also holds with respect to nuPPT attackers". We refer to this notion of security as *extractable uber security*.[10]

Our second main result shows that, assuming the existence of a leveled FHE with decryption in NC^1, there do not exist extractable uber-secure multilinear encodings (even if we only require security to hold w.r.t high-entropy distributions D).

Theorem 5 (Informally stated)**.** *Assume the existence of a leveled FHE with decryption in* NC^1*. Then no multilinear encodings can satisfy extractable (entropic) uber security.*

The high-level idea behind this result is to rely on the "conflict" between the feasibility of VBB obfuscation in the generic model of [16] and the impos-

[10] We use the adjective "extractable" as this security notion implies that if an nuPPT attacker can distinguish encodings, then the arithmetic circuits needed to distinguish the elements can be efficiently extracted out.

sibility of VBB obfuscation in the "standard" model [17]: we let $\vec{m}_b, \vec{z}$ contain a generically-secure VBB obfuscation of a program Π_b that hides b given just black-box access to Π_b, yet b can be recovered given the code of Π_b. By generic security of the obfuscation, it follows that *efficient* generic attackers cannot distinguish $\vec{m}_0, \vec{z}$ and $\vec{m}_1, \vec{z}$ yet, "non-generic" (i.e., standard PPT) attackers can. In our formal treatment, to rule out *constant-message* (as opposed to multi-message) security, we rely on a variant of the obfuscator presented in this paper, enhanced using techniques from [16].

We emphasize that in the above attack it is cruicial that we restrict to efficient (nuPPT) generic attacks. We finally consider several plausible ways of defining uber security for multilinear encodings, which circumvent the above impossibility results by requiring indistinguishability of encodings only if the encodings are *statistically* close w.r.t. *unbounded* generic attackers (that are restricted to making polynomially many zero-test queries). We highlight that none of these assumptions are needed for our construction of an indistinguishability obfuscation and are stronger than semantical security, but they may find other applications.

3.1.4 Construction Overview

The Basic Obfuscator We start by providing a construction of a "basic" obfuscator; our final construction will then rely on the basic obfuscator as a black-box. The construction of this obfuscator closely follows the design principles laid out in the original work by Garg et al [70] and follow-up constructions [50, 16] (in fact, the basic obfuscator may be viewed as a simplified version of the obfusca-

tor from [16]). As these works, we proceeds in three steps:

Following the original work of Garg et al (as well as subsequent works), the basic obfuscator proceeds in three steps:

- We view the NC^1 circuit to be obfuscated as a *branching program BP* (using Barrington's Theorem [19])—that is, the program is described by m pairs of matrices $(B_{i,0}, B_{i,1})$, each one labelled with an input bit $\mathsf{inp}(i)$. The program is evaluated as follows: for each $i \in [m]$, we choose one of the two matrices $(B_{i,0}, B_{i,1})$, based on the input. Next, we compute the product of the chosen matrices, and based on the product determine the output—there is a unique "accept" (i.e., output 1) matrix, and a unique "reject" (i.e., output 0) matrix.
- The branching program *BP* is *randomized* using Kilian's technique [108] (roughly, each pair of matrices is appropriately multiplied with the same random matrix R while ensuring that the output is the same), and then "randomized" some more—each individual matrix is multiplied by a random *scalar* α. Let us refer to this step as Rand.
- Finally the randomized matrices are encoded using multilinear encodings with the sets selected appropriately. We here rely on a (simple version) of the *straddling set* idea of [16] to determine the sets. We refer to this step as Encode.

(The original construction as well as the subsequent works also consisted of several other steps, but for our purposes these will not be needed.) The obfuscated program is now evaluated by using the multilinear operations to evaluate the branching program and finally appropriately use the zero-test to determine the

output of the program.

Roughly speaking, the idea behind the basic obfuscator is that the multilinear encodings *intuitively* ensure that any attacker getting the encoding needs to multiply matrices along paths that corresponds to some input to the branching program (the straddling sets are used to ensure that the input is used consistently in the evaluation)[11]; the scalars α, roughly speaking, ensure that a potential attacker without loss of generality can use a *single* "multiplication-path" and still succeed with roughly the same probability, and finally, Kilian's randomization steps ensures that if an attacker *only* operates on matrices along a single path that corresponds to some input x (in a consistent way), then its output can be perfectly simulated given just the output of the circuit on input x. (The final step relies on the fact that the output of the circuit uniquely determines product of the branching program along the path, and Kilian's randomization then ensures that the matrices along the path are random conditioned on the product being this unique value.) Thus, if an attacker can tell apart obfuscations of two programs BP_0, BP_1, there must exist some input on which they produce different outputs. The above intuitions can indeed be formalized w.r.t. *generic attackers* (that only operate on the encodings in a legal way, respecting the set restrictions), relying on arguments from [50, 16]. This already suffices to prove that the basic obfuscator is an indistinguishability obfuscator assuming the encodings are *multi-message* semantically secure.[12]

The Merge Procedure To base security on the weaker assumption of (constant-

[11] The encodings, however, still permit an attacker to add elements within matrices.

[12] As mentioned above, there are still some minor subtleties involved in doing this: the analyses of [50, 16] implicitly show that all *polynomial-size* legal arithmetic circuits are constant with overwhelming probability, but by slightly tweaking the constructions and the analyses to ensure a "perfect" simulation property, we can extend these arguments to hold against *all* (arbitrary-size) legal arithmetic circuits and thus base security on multi-message semantical security.

message) semantical security, we will add an additional program transformation steps before the Rand and Encode steps. Roughly speaking, we would like to have a method $\mathsf{Merge}(BP_0, BP_1, b)$ that "merges" BP_0 and BP_1 into a single branching program that evaluates BP_b; additionally, we require that $\mathsf{Merge}(BP_0, BP_1, 0)$ and $\mathsf{Merge}(BP_0, BP_1, 1)$ only differ in a constant number of matrices. We achieve this merge procedure by connecting together BP_0, BP_1 into a branching program of double width and adding two "switch" matrices in the beginning and the end, determining if we should go "up" or "down". Thus, to switch between $\mathsf{Merge}(BP_0, BP_1, 0)$ (which is functionally equivalent to BP_0) and $\mathsf{Merge}(BP_0, BP_1, 1)$ (which is functionally equivalent to BP_1) we just need to switch the "switch matrices". More precisely, given branching programs BP_0 and BP_1 described respectively by pairs of matrices $\{(B^0_{i,0}, B^0_{i,1}), (B^1_{i,0}, B^1_{i,1})\}_{i\in[m]}$, we construct a merged program $\mathsf{Merge}(BP_0, BP_1, b)$ described by $\{(\hat{B}^0_{i,0}, \hat{B}^0_{i,1})\}_{i\in[m+2]}$ such that

$$\hat{B}^0_{i,b} = \hat{B}^1_{i,b} = \begin{pmatrix} B^0_{(i-1),b} & 0 \\ 0 & B^1_{(i-1),b} \end{pmatrix} \quad \text{for all } 2 \le i \le m+1 \text{ and } b \in \{0,1\}$$

and the first and last matrices are given by:

$$\hat{B}^0_{1,b} = \hat{B}^0_{m+2,b} = I_{2w\times 2w} \qquad \text{for } b \in \{0,1\}$$

$$\hat{B}^1_{1,b} = \hat{B}^1_{m+2,b} = \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \qquad \text{for } b \in \{0,1\}$$

It directly follows from the construction that $\mathsf{Merge}(BP_0, BP_1, 0)$ and $\mathsf{Merge}(BP_0, BP_1, 1)$ differ only in the first and the last matrices (i.e., the "switch" matrices). Furthermore, it is not hard to see that $\mathsf{Merge}(BP_0, BP_1, b)$ is functionally equivalent to BP_b.

Our candidate obfuscator is now defined as

$$i\mathcal{O}(B) = \mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP, I, 0)))$$

where I is simply a "dummy" program of the same size as BP.[13]

The idea behind the merge procedure is that to prove that obfuscations of two programs BP_0, BP_1 are indistinguishable, we can come up with a sequence of hybrid experiments that start with $i\mathcal{O}(BP_0)$ and end with $i\mathcal{O}(BP_1)$, but between any two hybrids only changes a constant number of encodings, and thus we may rely on semantic security of multilinear encodings to formalize the above intuitions. At a high level, our strategy will be to matrix-by-matrix, replace the dummy branching program in the obfuscation of BP_0 with the branching program for BP_1. Once the entire dummy branching program has been replaced by BP_1, we flip the "switch" so that the composite branching program now computes the branching program for BP_1. We then replace the branching program for BP_0 with BP_1, matrix by matrix, so that we have two copies of the branching program for BP_1. We now flip the "switch" again, and finally restore the dummy branching program, so that we end up with one copy of BP_1 and one copy of the dummy, which is now a valid obfuscation of BP_1. In this way, we transition from an obfuscation of BP_0 to an obfuscation of BP_1, while only changing a small piece of the obfuscation in each step. (On a very high-level, this approach is somewhat reminiscient of the Naor-Yung "two-key" approach in the context of CCA security [126] and the "two-key" bootstrapping result for indistinguishability obfuscation due to Garg et al [70]—in all these approaches the length of the scheme is artificially doubled to facilitate a hybrid argument. It is perhaps even more reminiscient of the Feige-Shamir "trapdoor witness"

[13]This description oversimplifies a bit. Formally, the Rand step needs to depends on the field size used in the Encode steps, and thus in our formal treatment we combine these two steps together.

approach for constructing zero-knowledge arguments [65], whereby an additional "dummy" trapdoor witness is introduced in the construction to enable the security proof.)

More precisely, consider the following sequence of hybrids.

- We start off with $i\mathcal{O}(BP_0) = \mathsf{Enc}(\mathsf{Rand}(\mathsf{Merge}(BP_0, I, 0)))$
- We consider a sequence of hybrids where we gradually change the dummy program I to become BP_1; that is, we consider $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_0, BP', 0)))$, where BP' is "step-wise" being populated with elements from BP_1.
- We reach $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_0, BP_1, 0)))$.
- We turn the "switch" : $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_0, BP_1, 1)))$.
- We consider a sequence of hybrids where we gradually change the BP_0 to become BP_1; that is, we consider $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP', BP_1, 1)))$, where BP' is "step-wise" being populated with elements from BP_1.
- We reach $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, BP_1, 1)))$.
- We turn the "switch" back: $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, BP_1, 0)))$.
- We consider a sequence of hybrids where we gradually change the second BP_1 to become I; that is, we consider $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, BP', 0)))$, where BP' is "step-wise" being populated with elements from I.
- We reach $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, I, 0))) = i\mathcal{O}(BP_1)$.

By construction we have that if BP_0 and BP_1 are functionally equivalent, then so will all the hybrid programs–the key point is that we only "morph" between two branching programs on the "inactive" part of the merged branching program.

Furthermore, by construction, between any two hybrids we only change a constant number of elements. Thus, if some distinguisher can tell apart $i\mathcal{O}(BP_0)$ and $i\mathcal{O}(BP_1)$, it must be able to tell apart two consecutive hybrids. But, by semantic security it then follows that some "legal" arithmetic circuit can tell apart the encodings in the two hybrids. Roughly speaking, we can now rely on simulation security of the basic obfuscator w.r.t. to just *legal* arithmetic circuits to complete the argument. A bit more precisely, based on BP_0, BP_1 and the hybrid index i, we can define a message distribution D_{i,BP_0,BP_1} that is valid (by the simulation arguments in [16]) as long as BP_0 is functionally equivalent to BP_1, yet our distinguisher manages to distinguish messages samples from D_{i,BP_0,BP_1}, contradicting semantical security.

Dealing with branching programs with non-unique outputs There is a catch with the final step though. Recall that to rely on Kilian's simulation argument it was crucial that there are *unique* accept and reject matrices. For our "merged" programs, this is no longer the case (the output matrix is also a function of the second "dummy" program), and thus it is no longer clear how to prove that the message distribution above is valid. We overcome this issue by noting that the *first column* of the output matrix actually is unique, and this is all we need to determine the output of the branching program; we refer to such branching programs as *fixed output-column branching programs*. Consequently it suffices to release encodings of the *just* first column (as opposed to the whole matrices) of the last matrix pair in the branching program, and we can still determine the output of the branching program. As we show, for such a modified scheme, we can also simulate the (randomized) matrices along an "input-path" given just the first column of the output matrix.

A Modular Analysis: Neighboring-Matrix Indistinguishability Obfuscation In the actual proof, we provide a more modular analysis of the above two steps (that may be interesting in its own right).

- We define a notion of *neighboring-matrix indistinguishability obfuscation*, which relaxes indistinguishability obfuscation by only requiring security to hold w.r.t. any two functionally equivalent branching programs that differ in at most a constant number of matrices.
- We then use the above merge procedure (and the above hybrid argument) to show that the existence of a neighboring-matrix $i\mathcal{O}$ for all "fixed output column" branching programs implies the existence of a "full-fledged" $i\mathcal{O}$.
- We finally use the "basic obfuscator" construction to show how to construct a neighboring-matrix $i\mathcal{O}$ for all fixed output column branching programs based on (constant-message) semantical security.

Basing Security on a (Single) Falsifiable Assumption To base security on a falsifiable assumption, we rely on a different merge procedure from the work of Boyle, Chung and Pass [42]: Given two NC_1 circuits C_0, C_1 taking (at most) n-bit inputs, and a string z, let $\widehat{\mathsf{Merge}}(C_0, C_1, z)$ be a circuit that on input x runs $C_0(x)$ if $x \geq z$ and $C_1(x)$ otherwise; in essence, this procedure lets us "traverse" between C_0 and C_1 while provably only changing the functionality on at most one input. ([42] use this type of merged circuits to perform a binary search and prove that indistinguishability obfuscation implies differing-input obfuscation for circuits that differ in only polynomially many inputs.) We now define a notion of *neighboring-input* $i\mathcal{O}$, which relaxes $i\mathcal{O}$ by only requiring that security holds with respect to "neigboring-input" programs $\widehat{\mathsf{Merge}}(C_0, C_1, z)$, $\widehat{\mathsf{Merge}}(C_0, C_1, z+1)$

that are functionally equivalent. Note that checking whether $\widehat{\mathsf{Merge}}(C_0, C_1, z)$, $\widehat{\mathsf{Merge}}(C_0, C_1, z+1)$ are functionally equivalent is easy: they are equivalent iff $C_0(z) = C_1(z)$. (As such, the assumption that a scheme satisfies neighboring-input $i\mathcal{O}$ is already an efficiently falsfiable assumption.) Furthermore, by a simple hybrid argument over $z \in \{0,1\}^n$, *exponentially-secure* neighboring-input $i\mathcal{O}$ implies "full" $i\mathcal{O}$—exponential security is needed since we have 2^n hybrids. (We mention a very recent work by Gentry, Lewko and Waters [86] in the context of *witness encryption* [75] that similarly defines a falsifiable primitive "positional witness encryption" that implies the full-fledged notion with an exponential security loss.)

Additionally, note that to show that our construction satisfies exponentially-secure neigboring-input $i\mathcal{O}$, we only need to rely on exponentially-secure semantical security w.r.t. classes of sets and message distributions corresponding to programs of the form $\widehat{\mathsf{Merge}}(C_0, C_1, z)$, $\widehat{\mathsf{Merge}}(C_0, C_1, z+1)$. Equivalently, it suffices to rely on exponentially-secure semantical security w.r.t. a *single* distribution over sets and message distributions corresponding to uniformly selected programs $\widehat{\mathsf{Merge}}(C_0, C_1, z)$, $\widehat{\mathsf{Merge}}(C_0, C_1, z+1)$ (i.e., z, C_0, C_1 are picked at random); again, this only results in an exponential security loss. Finally, by padding the security parameter of the multilinear encodings in the construction, it actually suffices to rely on subexponential security.

3.1.5 Discussion and Future Work

We have introduced a new security notion, *semantical security*, for multilinear (a.k.a. graded) encodings, which captures a general (but quite restrictive) *class*

of algebraic decisional assumption over multilinear encodings. Our main result demonstrates the existence of indistinguishability obfuscators ($i\mathcal{O}$) assuming the existence of semantically secure multilinear encodings and the LWE assumption; as far as we know, this yields the first construction of $i\mathcal{O}$ based on a "simple-to-state" assumption about some algebraic primitive (namely, multilinear encodings) for which candidate constructions are known.

We additionally show that it suffices to assume the existence of encoding schemes that satisfy a *specific, falsifiable, instance* of semantical security (i.e., that a specific assumption in the class holds w.r.t. the encoding scheme); this time, however, we need to assume *subexponentially-hard* semantical security. This shows that under subexponential reductions, indistinguishability obfuscation can be based on a single, non-interactive and falsifiable, assumption.

We finally consider various strengthenings of semantical security, which (among other things) motivate why in our definition of semantical security, we restrict the class of algebraic decisional assumptions: we show that the assumption that "every non-interactive algebraic decisional assumption that holds against generic attackers holds against nuPPT attackers" is false.

Our work leaves open several interesting questions:

- Can we base $i\mathcal{O}$ on *polynomial-hardness* of a falsfiable (and preferrably non-interactive) assumption (using a security-preserving reduction)? Note that for many *applications* of $i\mathcal{O}$ (e.g., functional encryption [70]) it suffices to require indistinguishability for restricted distributions of programs that (with overwhelming probability) are *provably* functionally equivalent; for these applications, our proof already shows they can be based on specific,

falsifiable, instances of semantical security (without assuming subexponential hardness).

- Even in the regime of subexponential hardness, the specific assumption that we use—although it is a special case of semantical security—is not particularly natural, and does not have a particularly "simple" description. In essence, we consider semantical security with respect to distributions over elements that describe the obfuscation of a random branching program. (As such, in our eyes, perhaps the best reason to believe this assumption is true that it is a falsifiable special case of semantical security). It would be much more desirable to base security on semantical security w.r.t. a single *simple* and *natural* distribution over $\vec{m}_0, \vec{m}_1, \vec{z}$, where, for instance, similar to the GDDH assumption, $\vec{z}$ are uniformly random elements. We conjecture that our assumption actually can be "massaged" into a nicer looking assumption, closer in spirit to the GDDH assumption.

  Two recent works take a major step in this direction. The elegant work of Gentry, Lewko and Waters [86][14] bases *witness encryption* [75] on exponential hardness of some simple assumptions over multinear encodings—the "multilinear subgroup eliminations assumption" and the "multilinear subgroup decision assumption" (which are closer in spirit to the GDDH assumption); however, in contrast to our work they rely on multilinear (graded) encodings over composite-order rings (for which the only candidate is a modified variant of [61]), or require more complex assumptions over prime-order rings (that still are false for the [66] construction); furthermore, they require several additional functionalities from graded encodings—in particular, "subring generation", and "subring sampling",

[14]This result is subsequent to our results on iO from (entropic) semantical security, but preceeds our results on iO from single-distribution semantical security.

which require releasing additional "auxiliary elements" and thus challenges security (which is why a variant of the [61] construction is needed).

Even more recently, the beautiful work by Gentry, Lewko, Sahai and Waters [85] manages to demonstrate also *iO* from just the multilinear subgroup assumption over composite-order rings.[15] Just as [86] they require the additional functionalities from graded encoding scheme (and as such the only candidate construction currently know is the variant of the [61] scheme introduced in [86]). Although the implementation details are quite different, the construction in [85] follows our general approach of "merging" threads of branching programs (we here consider only two threads, whereas they consider multiple), and using a switch between "active" and "inactivate" threads. (Additionally, their notion of a "positional" *iO* is closely related to our notion of neighboring-input *iO* .)[16]

- Another interesting question is finding other applications of entropic semantically secure multilinear encodings. Our impossibility results—which show that there exist algebraic decisional assumptions that are false despite being true w.r.t. generic attackers—present a further challenge to the practice of arguing the plausibility of an assumption (even a "DDH-type" assumption) through security arguments in the generic model. At this point it seems that checking whether some specific algebraic assumption falls within the class of assumptions considered by entropic semantical security (or perhaps even just uber security) may be a viable replacement to the standard "sanity check" of arguing security in the generic

[15] This result is subsequent to our results on *iO* from (entropic) semantical security, and appears to be concurrent to (appearing on e-Print only a few days after) our results on *iO* from single-distribution semantical security.

[16] But as mentioned above, the results relying on neighboring-input *iO* were not part of our original manuscript and appear to be concurrent to the ones in [85].

model.

- In this paper we have focused on indistinguishability obfuscation. An interesting problem is basing stronger notions of obfuscation on some succinct and natural assumption on a low-level primitive. We mention that our result that any scheme satisfying iO security w.r.t. "neighboring-matrix" programs can be turned in a "fully" secure scheme, applies also to differing-input security.

  A recent beautiful work of Bitansky, Canetti, Kalai and Paneth [27] introduces a strengthening of semantical security (called "strong-sampler" semantical security) which also consider non-samplable (i.e., computationally unbounded) message distributions (as opposed to nuPPT distributions as we consider here); their key result demonstrated the existence of VGB (virtual grey-box secure) [24] obfuscators for NC^1 assuming strong-sampler semantical security. VGB security is a strengthening of iO ; but it is not known how to bootstrap VGB for NC^1 to all polynomial-size circuits.

3.1.6 Outline of the Paper

We provide some preliminaries in Section 3.2. We define semantical security of multilinear (aka graded) encodings in Section 3.3. Our construction of an indistinguishability obfuscator and its proof of security is provided in Section 3.4. We show how to slightly modify the construction to be based on a single (falsifiable) instance of semantical security in Section 3.5. We finally study alternative notions of security for multilinear encodings in Section 3.6.

3.2 Preliminaries

Let $\mathcal{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \ldots, n\}$. Let $\mathcal{Z}$ denote the integers, and $\mathbb{Z}_p$ the integers modulo p. Given a string x, we let $x[i]$, or equivalently x_i, denote the i-th bit of x. For a matrix M, we let $M[i, j]$ denote the entry of M in the ith row and jth column. We use $\mathbf{e}_k$ to denote the vector that is 1 in position k, and 0 in all other positions. The length of $\mathbf{e}_k$ is generally clear from the context. We use $I_{w\times w}$ to denote the identity matrix with dimension $w \times w$.

By a probabilistic algorithm we mean a Turing machine that receives an auxiliary random tape as input. If M is a probabilistic algorithm, then for any input x, $M(x)$ represents the distribution of outputs of $M(x)$ when the random tape is chosen uniformly. $M(x; r)$ denotes the output of M on input x when the random tape is fixed to r. An oracle algorithm M^O is a machine M that gets oracle access to another machine O, that is, it can access O's functionality as a black-box.

By $x \leftarrow S$, we denote an element x is sampled from a distribution S. If F is a finite set, then $x \leftarrow F$ means x is sampled uniformly from the set F. To denote the ordered sequence in which the experiments happen we use semicolon, e.g. $(x \leftarrow S; (y, z) \leftarrow A(x))$. Using this notation we can describe probability of events. For example, if $p(\cdot, \cdot)$ denotes a predicate, then $\Pr[x \leftarrow S; (y, z) \leftarrow A(x) : p(y, z)]$ is the probability that the predicate $p(y, z)$ is true in the ordered sequence of experiments $(x \leftarrow S; (y, z) \leftarrow A(x))$. The notation $\{(x \leftarrow S; (y, z) \leftarrow A(x) : (y, z))\}$ denotes the resulting probability distribution $\{(y, z)\}$ generated by the ordered sequence of experiments $(x \leftarrow S; (y, z) \leftarrow A(x))$. We define the support of a distribution $\mathsf{supp}(S)$ to be $\{y : \Pr[x \leftarrow S : x = y] > 0\}$.

By isZero, we denote the predicate such that $\mathsf{isZero}(x) = 1$ exactly when $x = 0$, and $\mathsf{isZero}(x) = 0$ otherwise.

3.2.1 Obfuscation

We recall the definition of indistinguishability obfuscation due to [17].

Definition 1 (Indistinguishability Obfuscator)**.** *A uniform PPT machine $i\mathcal{O}$ is an* indistinguishability obfuscator *for a class of circuits $\{C_n\}_{n\in\mathbb{N}}$ if the following conditions are satisfied*

- **Correctness:** *There exists a negligible function ε such that for every $n \in \mathcal{N}$, for all $C \in C_n$, we have*

$$\Pr[C' \leftarrow i\mathcal{O}\,(1^n, C) : \forall x,\ C'(x) = C(x)] \geq 1 - \varepsilon(n)$$

- **Security:** *For every pair of circuit ensembles $\{C_n^0\}_{n\in\mathbb{N}}$ and $\{C_n^1\}_{n\in\mathbb{N}}$ such that for all $n \in \mathbb{N}$, for every pair of circuits $C_n^0, C_n^1 \in C_n$ such that $C_n^0(x) = C_n^1(x)$ for all x the following holds: For every nuPPT adversary A there exists a negligible function ε such that for all $n \in \mathbb{N}$,*

  $$|Pr[C' \leftarrow i\mathcal{O}\,(1^n, C_n^0) : A(1^n, C') = 1] - Pr[C' \leftarrow i\mathcal{O}\,(1^n, C_n^1) : A(1^n, C') = 1]| \leq \varepsilon(n)$$

  We additionally say that $i\mathcal{O}$ is subexponentially-secure *if there exists some constant $\alpha > 0$ such that for every nuPPT A the above indistinguishability gap is bounded by $\varepsilon(n) = 2^{-O(n^\alpha)}$.*

Note: We observe that the above definition allows for a negligible correctness error. That is, for any circuit C, there is a negligible fraction of "bad" randomness r such that $i\mathcal{O}\,(C; r)$ is not functionally equivalent to C. However, if we can

efficiently check if r is "bad", we can modify $i\mathcal{O}$ so that $i\mathcal{O}(C; r)$ outputs C in the clear if r is "bad". Then the modified $i\mathcal{O}$ has perfect correctness, and its security remains intact since only a negligible fraction of r are "bad". We note that our construction, as well as all previous ones, have the property that a "bad" r can be efficiently detected, and thus these schemes can be modified to have perfect correctness.

We now recall the definitions of $i\mathcal{O}$ for NC^1 and $\mathsf{P/poly}$.

Definition 2 (Indistinguishability Obfuscator for NC^1)**.** *A uniform PPT machine $i\mathcal{O}$ is an* indistinguishability obfuscator for NC^1 *if for every constant c, $i\mathcal{O}(c, \cdot, \cdot)$ is an indistinguishability obfuscator for the class of circuits $\mathcal{C}^c = \{C_n^c\}_{n\in\mathcal{N}}$ where C_n^c is the set of circuits that have size at most n^c, and have depth at most $c \log n$.*

Definition 3 (Indistinguishability Obfuscator for $\mathsf{P/poly}$)**.** *A uniform PPT machine $i\mathcal{O}$ is an* indistinguishability obfuscator for $\mathsf{P/poly}$ *if for every constant c, $i\mathcal{O}(c, \cdot, \cdot)$ is an indistinguishability obfuscator for the class of circuits $\mathcal{P}^c = \{\mathcal{P}_n^c\}_{n\in\mathcal{N}}$ where $\mathcal{P}_n^c$ is the set of circuits that have size at most n^c.*

The following simple lemma will be useful in the sequel.

Lemma 6. *Let $i\mathcal{O}$ be a (subexponentially-secure) indistinguishability obfuscator for $\mathcal{C}^1$. Then $i\mathcal{O}'$ defined as $i\mathcal{O}'(c, 1^n, C) = i\mathcal{O}(1^{n^c}, C)$ is a (subexponentially-secure) indistinguishability obfuscator for NC^1.*

Proof. Consider any pair of circuit ensembles $\{C_n^0\}_{n\in\mathbb{N}}, \{C_n^1\}_{n\in\mathbb{N}}$ in $\mathcal{C}^c$. Assume for contradiction that there exists some nuPPT A and a polynomial $p(\cdot)$ such that $A(1^n)$ distinguishes $i\mathcal{O}'(c, 1^n, C_n^0) = i\mathcal{O}(1^{n^c}, C_n^0)$ and $i\mathcal{O}'(c, 1^n, C_n^1) = i\mathcal{O}(1^{n^c}, C_n^1)$ with probability $1/p(n)$ for infinitely many n. Note that for every n, $C_n^0, C_n^1 \in$

$C^1_{n^c}$. Thus, for infinitely many $n \in \mathcal{N}$, there exists circuits $C^0_n, C^1_n \in C^1_{n^c}$ such that $A(1^n)$ distinguishes $i\mathcal{O}(1^{n^c}, C^0_n)$ and $i\mathcal{O}(1^{n^c}, C^1_n)$ with probability $1/p(n)$. In other words, for infinitely many $n' \in \mathbb{N}$ of the form $n' = n^c$, there exist circuits $\tilde{C}^0_{n'} = C^0_n$, $\tilde{C}^1_{n'} = C^1_n$ such that the nuPPT $A'(1^{n'}) = A(1^n)$ distinguishes $i\mathcal{O}(1^{n'}, \tilde{C}^0_{n'})$ and $i\mathcal{O}(1^{n'}, \tilde{C}^1_{n'})$ with probability $1/p(n) = 1/p(n'^{1/c})$), which contradicts that $i\mathcal{O}$ is an indistinguishability obfuscator for C^1.

The same argument also works in the context of subexponential security. □

3.2.2 Branching programs for NC^1

We recall the notion of a branching program.

Definition 4 (Matrix Branching Program)**.** *A* branching program *of width w and length m for n-bit inputs is given by a sequence:*

$$BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1})\}_{i=1}^{m}$$

where each $B_{i,b}$ is a permutation matrix in $\{0,1\}^{w \times w}$, $\mathsf{inp}(i) \in [n]$ is the input bit position examined in step i.

Then the output of the branching program on input $x \in \{0,1\}^n$ is as follows:

$$BP(x) \stackrel{def}{=} \begin{cases} 1, & \text{if } (\prod_{i=1}^{m} B_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1 = \mathbf{e}_1. \\ 0, & \text{otherwise} \end{cases}$$

The branching program is said to be oblivious *if* $\mathsf{inp} : [m] \to [n]$ *is a fixed function, independent of the function being evaluated.*

The above definition differs slightly from the definition of matrix branching programs generally used, which have the slightly stronger requirement that $\prod_{i=1}^{n} B_{i,x[\mathsf{inp}(i)]} = I_{w\times w}$ when $BP(x)$ is accepting, and $\prod_{i=1}^{n} B_{i,x[\mathsf{inp}(i)]} = \mathsf{P}_{\mathsf{reject}}$ for some fixed permutation matrix $\mathsf{P}_{\mathsf{reject}} \neq I_{w\times w}$ when $BP(x)$ is rejecting. More generally,

Definition 5. *The branching program is said to have* fixed accept and reject matrices $\mathsf{P}_{\mathsf{accept}}$ *and* $\mathsf{P}_{\mathsf{reject}}$ *if, for all* $x \in \{0,1\}^n$,

$$\prod_{i=1}^{m} B_{i,x[\mathsf{inp}(i)]} = \begin{cases} \mathsf{P}_{\mathsf{accept}} & \text{when } BP(x) = 1 \\ \mathsf{P}_{\mathsf{reject}} & \text{when } BP(x) = 0 \end{cases}$$

We now have the following theorem due to Barrington:

Theorem 7. *([19]) There exist* 5×5 *permutation matrices* $\mathsf{P}_{\mathsf{accept}}$ *and* $\mathsf{P}_{\mathsf{reject}}$ *with* $\mathsf{P}_{\mathsf{accept}} \cdot \mathbf{e}_1 = \mathbf{e}_1$, *and* $\mathsf{P}_{\mathsf{reject}} \cdot \mathbf{e}_1 = \mathbf{e}_k$ *where* $k \neq 1$ *such that the following holds. For any depth* d *and input length* n, *there exists a length* $m = 4^d$, *a labeling function* $\mathsf{inp} : [m] \to [n]$ *such that, for every fan-in* 2 *boolean circuit* C *of depth* d *and* n *input bits, there exists an oblivious matrix branching program* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^{m}$, *of width* 5 *and length* m *that computes the same function as the circuit* C.

In particular, every circuit in NC^1 has a polynomial length branching program of width 5. Further, two circuits of the same depth d will have the same fixed accepting and rejecting permutations $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$, and a fixed labelling function $\mathsf{inp} : [m] \to [n]$.

The branching programs we consider in this work will not have fixed output matrices. However, the first column of their output matrices will be fixed

and depend only on the output of the program. That is, the first column of the output matrix is either $\mathsf{p}_{\mathsf{accept}}$ or $\mathsf{p}_{\mathsf{reject}}$, depending on whether the program accepts or rejects. Furthermore, we will consider ensembles of classes of programs where these fixed columns $\mathsf{p}_{\mathsf{accept}}$ and $\mathsf{p}_{\mathsf{reject}}$ are the same for all programs in every class in the ensemble.

Definition 6 (Fixed Output Column Ensemble)**.** *An ensemble of classes of branching programs $\mathcal{B} = \{\mathcal{B}_n\}_{n\in\mathcal{N}}$ where $\mathcal{B}_n$ contains branching programs of constant width w, is a* fixed output column ensemble *if there exists vectors $\mathsf{p}_{\mathsf{accept}}, \mathsf{p}_{\mathsf{reject}} \in \{0,1\}^w$ such that for every $n \in \mathcal{N}$, every branching program $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1})\}_{i=1}^{m} \in \mathcal{B}_n$ and every input x it holds that*

$$(\prod_{i=1}^{m} B_{i,x[\mathsf{inp}(i)]})) \cdot \mathbf{e}_1 = \begin{cases} \mathsf{p}_{\mathsf{accept}} & \text{when } BP(x) = 1 \\ \mathsf{p}_{\mathsf{reject}} & \text{when } BP(x) = 0 \end{cases}$$

Subsequently, whenever we refer to an ensemble of classes of branching programs we will implicitly be referring to a fixed output column ensemble.

3.3 Semantically Secure Graded Encoding Schemes

In this section we define what it means for a graded encoding scheme to be semantically secure. We start by recalling the notion of graded encoding schemes due to Garg, Gentry and Halevi [66].

3.3.1 Graded Encoding Schemes

Graded (multilinear) encoding schemes were originally introduced in the work of Garg, Gentry and Halevi [66]. Just as [50, 16], we here rely on "set-based" (or "asymmetric") graded encoding; these were originally called "generalized" graded encodings in [66]. Following [70, 16] and the notion of "multilinear jigsaw puzzles" from [70], we additionally enable anyone with the secret parameter to encode *any* elements (as opposed to just *random* elements as in [66]).

Definition 7 ((k, R)-Graded Encoding Scheme)**.** *A (k, R)-graded encoding scheme for $k \in \mathbb{N}$ and ring R is a collection of sets $\{E_S^\alpha : \alpha \in R, S \subseteq [k]\}$ with the following properties*

- *For every $S \subseteq [k]$ the sets $\{E_S^\alpha : a \in R\}$ are disjoint.*
- *There are associative binary operations $\oplus$ and $\ominus$ such that for every $\alpha_1, \alpha_2 \in R$, $S \subseteq [k]$, $u_1 \in E_S^{\alpha_1}$ and $u_2 \in E_S^{\alpha_2}$ it holds that $u_1 \oplus u_2 \in E_S^{\alpha_1+\alpha_2}$ and $u_1 \ominus u_2 \in E_S^{\alpha_1-\alpha_2}$ where '+' and '−' are the addition and subtraction operations in R.*
- *There is an associative binary operation $\otimes$ such that for every $\alpha_1, \alpha_2 \in R$, $S_1, S_2 \subseteq [k]$ such that $S_1 \cap S_2 = \emptyset$, $u_1 \in E_{S_1}^{\alpha_1}$ and $u_2 \in E_{S_2}^{\alpha_2}$ it holds that $u_1 \otimes u_2 \in E_{S_1 \cup S_2}^{\alpha_1 \cdot \alpha_2}$ where '$\cdot$' is multiplication in R.*

Definition 8 (Graded Encoded Scheme)**.** *A graded encoding scheme $\mathcal{E}$ is associated with a tuple of PPT algorithms,* $(\mathsf{InstGen}_\mathcal{E}, \mathsf{Enc}_\mathcal{E}, \mathsf{Add}_\mathcal{E}, \mathsf{Sub}_\mathcal{E}, \mathsf{Mult}_\mathcal{E}, \mathsf{isZero}_\mathcal{E})$ *which behave as follows:*

- *Instance Generation:* $\mathsf{InstGen}_\mathcal{E}$ *takes as input the security parameter 1^n and multilinearity parameter 1^k, and outputs secret parameters* sp *and public parameters*

pp *which describe a* (k, R)*-graded encoding scheme* $\{E_S^\alpha : \alpha \in R, S \subseteq [k]\}$*. We refer to* E_S^α *as the set of encodings of the pair* (α, S)*. We restrict to graded encoding schemes where* R *is* $\mathbb{Z}_p$ *and* p *is a prime exponential in* n *and* k*.*

- *Encoding:* $\mathsf{Enc}_{\mathcal{E}}$ *takes as input the secret parameters* sp*, an element* $\alpha \in R$ *and set* $S \subseteq [k]$*, and outputs a random encoding of the pair* (α, S)*.*
- *Addition:* $\mathsf{Add}_{\mathcal{E}}$ *takes as input the public parameters* pp *and encodings* $u_1 \in E_{S_1}^{\alpha_1}, u_2 \in E_{S_2}^{\alpha_2}$*, and outputs an encoding of the pair* $(\alpha_1 + \alpha_2, S)$ *if* $S_1 = S_2 = S$ *and outputs* $\perp$ *otherwise.*
- *Negation:* $\mathsf{Sub}_{\mathcal{E}}$ *takes as input the public parameters* pp *and encodings* $u_1 \in E_{S_1}^{\alpha_1}, u_2 \in E_{S_2}^{\alpha_2}$*, and outputs an encoding of the pair* $(\alpha_1 - \alpha_2, S)$ *if* $S_1 = S_2 = S$ *and outputs* $\perp$ *otherwise.*
- *Multiplication:* $\mathsf{Mult}_{\mathcal{E}}$ *takes as input the the public parameters* pp *and encodings* $u_1 \in E_{S_1}^{\alpha_1}, u_2 \in E_{S_2}^{\alpha_2}$*, and outputs an encoding of the pair* $(\alpha_1 \cdot \alpha_2, S_1 \cup S_2)$ *if* $S_1 \cap S_2 = \emptyset$ *and outputs* $\perp$ *otherwise.*
- *Zero testing:* $\mathsf{isZero}_{\mathcal{E}}$ *takes as input the public parameters* pp *and an encoding* $u \in E_S(\alpha)$*, and outputs 1 if and only if* $\alpha = 0$ *and* S *is the universe set* $[k]$*.*[17]

Whenever it is clear from the context, to simplify notation we drop the subscript $\mathcal{E}$ *when we refer to the above procedures (and simply call them* $\mathsf{InstGen}, \mathsf{Enc}, \ldots$*).*

In known candidate constructions [66, 61], encodings are "noisy" and the noise level increases with each operation; the parameters, however, are set so

[17] In the candidate scheme given by [66], isZero may not have perfect correctness: the generated instances $(\mathsf{pp}, \mathsf{sp})$ can be "bad" with some negligible probability, so that there could exist an encoding u of a nonzero element where $\mathsf{isZero}(\mathsf{pp}, u) = 1$. However, these "bad" parameters can be efficiently detected during the execution of $\mathsf{InstGen}$. We can thus modify the encoding scheme to simply set $\mathsf{Enc}(\mathsf{pp}, e) = e$ whenever the parameters are "bad" (and appropriately modify $\mathsf{Add}, \mathsf{Sub}, \mathsf{Mult}$ and isZero so that the operate on "unencoded" elements. This change ensures that, for every pp, including "bad" ones, the zero test procedure isZero works with perfect correctness. We note that since bad parameters occur only with negligible probability, this change does not affect the security of the encodings.

that any poly(n, k) operations can be performed without running into trouble. For convenience of notation (and just like all other works in the area), we ignore this noise issue.[18]

Note that the above procedures allow algebraic operations on the encodings in a restricted way. Given the public parameters and encodings made under the sets $\vec{S}$, one can only perform algebraic operations that are allowed by the structure of the sets in $\vec{S}$. We call such operations $\vec{S}$-respecting and formalize this notion as follows:

Definition 9 (Set Respecting Arithmetic Circuits)**.** *For any sequence $\vec{S}$ of subsets of $[k]$, we say that an arithmetic circuit C (i.e. gates perform only ring operations $\{+, -, \cdot\}$) is* $\vec{S}$-respecting *if it holds that*

- *Eevery input wire of C is tagged with some set in $\vec{S}$.*
- *For every $+$ and $-$ gate in C, if the tags of the two input wires are the same set S then the output wire of the gate is tagged with S. Otherwise the output wire is tagged with $\perp$.*
- *For every $\cdot$ gate in C, if the tags of the two input wires are sets S_1 and S_2 and $S_1 \cap S_2 = \emptyset$ then the output wire of the gate is tagged with $S_1 \cup S_2$. Otherwise the output wire is tagged with $\perp$.*
- *It holds that the output wire is tagged with the universe set $[k]$.*[19]

We say that a circuit C is weakly $\vec{S}$-respecting *if all the above conditions hold except the last, that is, the output wire may be tagged with some set $T \subseteq [k]$, where T*

[18] The above definition can be easily generalized to deal with the candidates by only requiring that the above conditions hold when u_1, u_2 have been obtained by poly(n, k) operations.

[19] For ease of notation, we assume that the description of a set S also contains a description of the universe set $[k]$.

is not necessarily equal to $[k]$. *We say that* C *is* non terminal $\vec{S}$-respecting *if* T *is a strict subset of* $[k]$.

3.3.2 Semantical Security

We now turn to defining semantical security of graded encoding schemes. Towards explaining our notion of semantical security, let us first consider a "DDH-type" assumption for (asymmetric) multilinear encodings, similar in spirit to the "graded DDH" assumption of Garg et al [66] (which was in the contex of symmetric multilinear encodings, whereas we here consider asymmetric ones). Consider a distribution D sampling n random elements $\vec{z}$, and let $m_0 = \prod_{i\in[n]} z_i$ be the product of the elements in $\vec{z}$, and $m_1 = z'$ be just a random element. A DDH-type assumption—let us refer to it as the "asymmetric graded DDH assumption (aGDDH)"—would require that encodings of $m_0, \vec{z}$ and $m_1, \vec{z}$ under the sets $S, \vec{T}$ are indistinguishable as long as (a) S is the target set $[k]$, and (b) S is not the disjoint union of the sets in $\vec{T}$; that is, the set-restrictions prohibit "legally" multiplying all the elements of $\vec{z}$ and subtracting them from m_0 or m_1. $\vec{z}$.

Note that for any such sets $S, \vec{T}$, the particular (joint) distribution D over $m_0, m_1, \vec{z}$ has a nice "zero-knowledge" property w.r.t. generic attacker: for every $(S, \vec{T})$-respecting circuit C, $\mathsf{isZero}(C(\cdot))$ is *constant* over $(m_b, \vec{z})$, $b \in \{0, 1\}$ with overwhelming probability: that is, there exists some bit c such that with overwhelming probability over $m_0, m_1, \vec{z} \leftarrow D$, $\mathsf{isZero}(C(m_b, \vec{z})) = c$ for $b \in \{0, 1\}$, and as (except with negligible probability) no zero-test query leaks *anything* to a generic attacker. To see this, note that any such $\mathsf{isZero}(C(m, \vec{z})$ function is of the form

$\mathsf{isZero}(a \cdot m + p(\vec{z}))$ where $p(\cdot)$ is a polynomial of degree at most $n-1$. If $a = 0$ and $p(\cdot)$ is the zero-polynomial, then clearly the function evaluates to 1. If either $a = 1$ or $p(\cdot)$ is a non-zero polynomial, then no matter whether $m = m_0$ or $m = m_1$, $\mathsf{isZero}(C(\cdot,\cdot))$ is evaluating a non-zero polynomial of degree at most n at a random point; by the Schwartz-Zippel lemma, with overwhelming probability (proportional to the field size), both these polynomials will evaluate to a non-zero value, and thus the zero-test will output 0.

We refer to any distribution D satisfying the above "zero-knowledge w.r.t. generic attackers" property as being *valid* w.r.t. $S, \vec{T}$. We formalize this notion through what we refer to as a *$(S, \vec{T})$-respecting message sampler*. As mentioned in the introduction, for our purposes, we need to consider a more general setting where m_0,m_1, and S are replaced by *constant-length* vectors $\vec{m_0}, \vec{m_1}, \vec{S}$; for generality, we provide a definition that considers arbitrary length vectors of messages.

Definition 10 (Set-Respecting Operations)**.** *Let $\{k_n\}_{n\in\mathcal{N}}$ be an ensemble where $k_n \in \mathcal{N}$. We say $f = \{f_n\}_{n\in\mathcal{N}}$ is an ensemble of* set-respecting operations *if for every $n \in \mathcal{N}$, and every pair of sequences of sets $\vec{S}, \vec{T}$ over $[k_n]$ we have that $f_n(\vec{S}, \vec{T})$ outputs a $(\vec{S}, \vec{T})$-respecting arithmetic circuit.*

Definition 11 (Valid Message Sampler)**.** *Let $\mathcal{E}$ be a graded encoding scheme. We say that a nuPPT M is a* valid message sampler *if*

- *M on input 1^n and a public parameter $\mathsf{pp} \in \mathsf{InstGen}(1^n, 1^{k_n})$ computes the ring R associated with pp and next based on only 1^n, 1^{k_n} and R generates and outputs*
  - *a pair $(\vec{S}, \vec{T})$ of sequences of sets over $[k_n]$ and*
  - *a pair $(\vec{m_0}, \vec{m_1})$ of sequences of $|S|$ ring elements and a sequence $\vec{z}$ of $|T|$ ring elements.*

- *There exists a polynomial $Q(\cdot,\cdot)$ such that for every ensemble $\{k_n\}_{n\in\mathcal{N}}$ and ensemble of set-respecting operations $\{f_n\}_{n\in\mathcal{N}}$, for every $n \in \mathcal{N}$ there exists a constant $c \in \{0,1\}$ such that that for any $b \in \{0,1\}$,*

$$Pr[(\vec{m}_0, \vec{m}_1, \vec{z}, \vec{S}, \vec{T}) \leftarrow M(1^n, \mathsf{pp}); C \leftarrow f_n(\vec{S}, \vec{T}) : \mathsf{isZero}(C(\vec{m}_b, \vec{z})) = c]$$
$$\geq 1 - Q(n, k_n)/|R|.$$

Let us comment that Definition 11 allows the message sampler M to select $\vec{m}_0, \vec{m}_1, \vec{z}$ based on the ring $R = \mathcal{Z}_p$; note that this is needed even to model the aGDDH assumption (or else we could not define what it means to pick a uniform element in the ring). On the other hand, to make the notion of valid message samplers as restrictive as possible, we prevent the message selection from depending on pp in any other way. Looking ahead, this restriction makes the notion somewhat nicer behaved; see Lemma 8.

We can now define what it means for a graded encoding scheme to be semantically secure. Roughly speaking, we require that encodings of $(\vec{m}_0, \vec{z})$ and $(\vec{m}_1, \vec{z})$ under the sets $(\vec{S}, \vec{T})$ are indistinguishable as long as $(\vec{m}_0, \vec{m}_1, \vec{z})$ is sampled by a message sampler that is valid w.r.t. $(\vec{S}, \vec{T})$.

Definition 12 (Semantic Security)**.** *Let $\mathcal{E}$ be a graded encoding scheme and $q(\cdot)$ and $c(\cdot)$ be polynomials. We say a graded encoding scheme $\mathcal{E}$ is* (c, q)-semantically secure *if for every polynomial $k(\cdot)$, every ensemble $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$ where $\vec{S}_n$ and $\vec{T}_n$ are sequences of subsets of $[k(n)]$ of length $c(k(n))$ and $q(k(n))$ respectively, for every $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-set-respecting message sampler M and every nuPPT adversary A, there exists a negligible function ϵ such that for every security parameter $n \in \mathbb{N}$,*

$$|Pr[\boldsymbol{Output}_0(1^n) = 1] - Pr[\boldsymbol{Output}_1(1^n) = 1]| \leq \epsilon(n)$$

where $\boldsymbol{Output}_b(1^n)$ is A's output in the following game:

- *Let* $(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k(n)})$.
- *Let* $\vec{m}_0, \vec{m}_1, \vec{z} \leftarrow M(1^n, \mathsf{pp})$.
- *Let* $\vec{u_b} \leftarrow \{\mathsf{Enc}(\mathsf{sp}, \vec{m}_0[i], \vec{S}_n[i])\}_{i=1}^{c(k_n)}, \{\mathsf{Enc}(\mathsf{sp}, \vec{z}[i], \vec{T}_n[i])\}_{i=1}^{q(k(n))}$.
- *Finally, run* $A(1^n, \mathsf{pp}, \vec{u_b})$.

We say that $\mathcal{E}$ *is* (constant-message) semantically secure *if it is* $(O(1), O(k))$*-semantically secure; we say that* $\mathcal{E}$ multi-message semantically secure *if it is* $(O(k), O(k))$*-semantically secure. We additionally say that* $\mathcal{E}$ *is* subexponentially-hard semantically secure *if there exists some exists some constant* $\alpha > 0$ *such that for every nuPPT A the above indistinguishability gap is bounded by* $\varepsilon(n) = 2^{-O(n^\alpha)}$.[20]

In analogy with the GDDH assumption, our notion of semantical security restricts to the case when the number of elements encoded is $O(k)$.[21] As the following lemma (whose proof is delegated to Section 3.9) shows, any such encoding scheme can be modified to one that is secure as long as the number of elements in $\vec{z}$ is (a-priori) polynomially bounded.

Lemma 8. *Let* c, ϵ *be constants and let* $\mathcal{E}$ *be a* (c, k^ϵ)*-semantically secure encoding scheme. Then for every polynomial* $q(k)$ *there exists a* $(c, q(k))$*-semantically secure encoding scheme.*

Also, note that our notion of semantical security requires that security holds w.r.t. to *any* polynomial multilinearity parameter $k(\cdot)$; again, this is without

[20] We could also have considered an even stronger notion where the adversary A is allowed to be of subexponential-size; this will not be needed for our result, but may be useful in other contexts.

[21] This restriction was suggested in [27] and independently by Hoeteck Wee; our original formulation of semantical security considered an unbounded polynomial number of elements in $\vec{z}$ (but our proof of security only relied on security for $O(k)$ elements). We now refer to this stronger notion as *unbounded semantical security*; see below.

loss of generality: Any encoding scheme $\mathcal{E}$ that is semantically secure for any multilinearity parameter $k(n) \leq n$, can be turned into a new scheme $\mathcal{E}'$ that is (full-fledged) semantically secure, by simply letting $\mathsf{InstGen}'(1^n, 1^k) = \mathsf{InstGen}(1^{n+k}, 1^k)$.

Finally, one may also consider a notion of *unbounded semantical security* (that is provably stronger than semantical security)[22] which requires that $\mathcal{E}$ is $(O(1), q(k))$-semantically secure for *every* polynomial $q(k)$; this notion is not needed for our results. A recent result by [27] shows that for natural *special cases* of message samplers, *unbounded* single-message semantical security implies multi-message semantical security; we mention that this result only applies in the regime of polynomial security (and in particular does not apply for subexponential-hard semantical security).

Let us end this section by remarking that (sub-exponentially hard) semantical security trivially holds against polynomial-time "generic" attackers that are restricted to "legally" operating on the encodings—in fact, it holds even against *unbounded* generic attackers that are restricted to only making polynomially (or even subexponentially) many zero-test queries: recall that each legal zero-test query is constant with overwhelming probability (whether we operate on $\vec{m}_0, \vec{z}$ or $\vec{m}_1, \vec{z}$) and thus by a Union Bound, the output of any generic attacker restricted to polynomially many zero-test queries is also constant with overwhelming probability; see Section 3.6 for a formal statement.

Semantical Security w.r.t. Restricted Classes of Message Samplers For our

[22] Any semantically secure encoding scheme $\mathcal{E}$ can be modified into a new encoding scheme $\mathcal{E}'$ that still is semantically secure but not unbounded semantically secure. Simply let each encoding additionally release a random share of a secret-sharing of sp. If few shares are released (i.e., $\vec{z}$ is small) security is untouched, but if many shares are released security is trivially broken.

specific construction of indistinguishability obfuscators it suffices to assume the existence of *semantically secure encodings w.r.t. restricted classes of message samplers* M, where the $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-respecting condition on M is replaced by some stronger restriction on M. It particular, it suffices to restrict to message samplers M that induce a *high-entropy* distribution over $\vec{m}_0, \vec{m}_1, \vec{z}$—not only the individual elements have high min-entropy but also any element computed by applying a "non-terminal" sequence of legal arithmetic operations to $\vec{m}_b, \vec{z}$ (for $b \in \{0,1\}$). More precisely, we say that a M is a *H-entropic* $\{(\vec{S}_n, \vec{T}_n))\}_{n\in\mathbb{N}}$*-respecting message sampler* if M is $\{(\vec{S}_n, \vec{T}_n))\}_{n\in\mathbb{N}}$-respecting, where the sets S_n and T_n are over the universe set $[k_n]$ and additionally:

- For every security parameter n, every $\mathsf{pp} \in \mathsf{InstGen}(1^n, 1^{k_n})$ describing a ring R, every non-terminal $(\vec{S}_n, \vec{T}_n)$-respecting arithmetic circuit C that computes a non-zero polynomial in its inputs, it holds that for $b \in \{0,1\}$,

$$H_\infty(C(\vec{m}_b, \vec{z})) \geq H(\log |R|)$$

  where $(\vec{m}_0, \vec{m}_1, \vec{z}) \leftarrow M(1^n, \mathsf{pp})$.

We here focus on "very" high entropy message samplers, where $H(n) = n - O(\log n)$, and refer to such message samplers as simply *entropic* $\{\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$*-respecting message sampler* (or entropically valid), and refer to encoding schemes satisfying semantical security w.r.t. such restricted message samplers as *entropic semantically secure*.

Additionally, for our purposes, we may consider semantic security with respect to even more restricted types of message samplers M and sets $(\vec{S}_n, \vec{T}_n)$. In particular, where: (1) Each individual element sampled is statistically close to

a uniform ring element; (2) Elements sampled are "almost" pair-wise independent: each pair of elements encoded is statistically close to two uniform ring elements;[23] (3) The sets contained in the sequences $\vec{S}_n, \vec{T}_n$ are pairs of indices $\{i, j\}$, $i, j \in [k_n]$. Properties 1, 2 are natural abstractions of what happens in the GDDH assumption (property 2 is a relaxation of the independence, as opposed to just pair-wise independence, property satisfied by the GDDH assumption). Property 3 implies that (if we consider a arithemtic circuit) exactly $k/2$ multiplications on the elements must be performed before a zero-testing can be done; combined with the above entropic message sampler condition, this implies that any set-respecting arithmetic circuit of multiplicative degree smaller than $k/2$ produces a high-entropy element when applied to the sampled elements.[24]

3.4 $i\mathcal{O}$ from Semantically Secure Multilinear Encodings

In this section we prove that semantically secure multilinear encodings implies indistinguishability obfuscators for NC^1. We will show this through the following steps.

- We first introduce a weaker notion of indistinguishability obfuscation for branching programs, which we call *neighboring-matrix indistinguishability obfuscation*. Roughly speaking, this notion guarantees that the obfuscations of any pair of functionally equivalent branching programs that *differ in only a few matrices* are computationally indistinguishable.

[23] We thank Hoeteck Wee to suggesting to consider independence properties among the elements.

[24] We thank Shai Halevi for this observation (and more generally for suggesting that we consider the output of low-degree arithmetic circuits as an alternative to our entropic condition.).

- We show that any neighboring-matrix indistinguishability obfuscator for branching programs can be transformed into an *full* indistinguishability obfuscator for NC^1.
- Finally, we show that assuming the existence of semantically secure multilinear encodings, there exists a neighboring-matrix indistinguishability obfuscator for branching programs.

3.4.1 Neighboring-Matrix Indistinguishability Obfuscation (*nm-iO*)

We introduce a weaker notion of indistinguishability obfuscation for branching programs. This notion is similar to indistinguishability obfuscation except that instead of requiring security to hold with respect to *any* pair of functionally equivalent programs, we require security to hold with respect to any pair of *neighboring* programs that are functionally equivalent. We say a pair of branching programs are neighboring if they differ in only a few matrices.

Definition 13 (Neighboring-Matrix Branching Programs)**.** *We say that BP_0 and BP_1 are a pair of* neighboring-matrix branching programs *if they differ in at most 4 matrices. We say that $\{BP_n^0\}_{n\in\mathcal{N}}$ and $\{BP_n^1\}_{n\in\mathcal{N}}$ are a pair of* neighboring-matrix branching program ensembles *if for every $n \in \mathcal{N}$, BP_n^0 and BP_n^1 are a pair of neighboring-matrix branching programs.*

Definition 14 (Neighboring-Matrix Indistinguishability Obfuscator)**.** *A uniform PPT machine* Obf *is an* neighboring-matrix indistinguishability obfuscator *for an ensemble of classes of branching programs $\{\mathcal{B}_n\}_{n\in\mathbb{N}}$ if it satisfies the same correctness and security conditions as in Definition 1 except that the security condition quantifies only*

over pairs of neighboring-matrix branching program ensembles (as opposed to pairs of arbitrary circuit ensembles as in Definition 1).

3.4.2 From *nm-iO* to *iO*

In this section we show that any neighboring-matrix indistinguishability obfuscator for a particular ensemble of classes of branching programs can be transformed into *full* indistinguishability obfuscators for NC^1.

Roughly speaking, the indistinguishability obfuscator iO will use the neighboring-matrix indistinguishability obfuscator Obf in the following way: iO on input a circuit C, first converts it to an oblivious branching program BP using Theorem 7. Next, iO doubles the width of BP by "merging" it with a dummy branching program that computes the constant 1, and then adds a branch at the very start that chooses whether to use the true program or the dummy, based on a "switch". iO simply returns the obfuscation of the above "merged" branching program as produced by Obf.

At a high level, to show indistinguishability of $iO(C_1)$ and $iO(C_2)$, our strategy will be to obfuscate (using Obf) the "merged" branching program for C_1, and then, matrix by matrix, replace the dummy branching program with the branching program for C_2. Once the entire dummy branching program has been replaced by C_2, we flip the "switch" so that the composite branching program now computes the branching program for C_2. We then replace the branching program for C_1 with C_2, matrix by matrix, so that we have two copies of the branching program for C_2. We now flip the "switch" again, and finally restore the dummy branching program, so that we end up with one copy of C_2 and one

copy of the dummy. In this way, we transition from $i\mathcal{O}(C_1)$ to $i\mathcal{O}(C_2)$, while only changing a small piece of the branching program being obfuscated under Obf in each step, and keeping the functionality the same. If Obf is a neighboring-matrix indistinguishability obfuscator then each step of these transitions must be indistinguishable, hence showing $i\mathcal{O}$ is an *full* indisntinguishability obfuscator.

Merging Branching Programs

We first describe a method Merge for combining any two matrix branching programs together to create a composite branching program of double width, in a way that enables switching by changing only a small number of matrices.

Construction 1 (Merging branching programs)**.** *Let* $BP_0 = \{\mathsf{inp}(i), B^0_{i,0}, B^0_{i,1}\}_{i=1}^m$ *and* $BP_1 = \{\mathsf{inp}(i), B^1_{i,0}, B^1_{i,1}\}_{i=1}^m$ *be oblivious matrix branching programs, each of width* w *and length* m *for* n *input bits. (We assume that the same labelling function* $\mathsf{inp} : [m] \to [n]$ *is used for each of* BP_0 *and* BP_1*, and this is without loss of generality because we can add extra dummy levels so that this property holds.)*

Define branching programs $\widehat{BP}_0 = \{\mathsf{inp}'(i), \hat{B}^0_{i,0}, \hat{B}^0_{i,1}\}_{i=1}^{m+2}$ *and* $\widehat{BP}_1 = \{\mathsf{inp}'(i), \hat{B}^1_{i,0}, \hat{B}^1_{i,1}\}_{i=1}^{m+2}$*, each of width* $2w$ *and length* $m+2$ *on* l *input bits, where:*

$$\mathsf{inp}'(i) \stackrel{def}{=} \begin{cases} 1, & \text{when } i = 1 \\ \mathsf{inp}(i-1), & \text{when } 2 \le i \le m+1 \\ 1, & \text{when } i = m+2 \end{cases}$$

and, for all levels except the first and the last, $\widehat{BP}_0$ *and* $\widehat{BP}_1$ *agree, given by:*

$$\hat{B}^0_{i,b} = \hat{B}^1_{i,b} \stackrel{def}{=} \begin{pmatrix} B^0_{(i-1),b} & 0 \\ 0 & B^1_{(i-1),b} \end{pmatrix} \quad \textit{for all } 2 \le i \le m+1 \textit{ and } b \in \{0,1\}$$

and the first and last levels are given by:

$$\hat{B}^0_{1,b} = \hat{B}^0_{m+2,b} = I_{2w \times 2w} \qquad \textit{for } b \in \{0,1\}$$

$$\hat{B}^1_{1,b} = \hat{B}^1_{m+2,b} = \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \qquad \textit{for } b \in \{0,1\}$$

We define Merge *so that* $\mathsf{Merge}(BP_0, BP_1, 0) = \hat{BP}_0$ *and* $\mathsf{Merge}(BP_0, BP_1, 1) = \hat{BP}_1$.

We will show that $\hat{BP}_0$ and $\hat{BP}_1$ are matrix branching programs that compute the same functions as BP_0 and BP_1 respectively, with the additional feature that $\hat{BP}_0$ and $\hat{BP}_1$ differ from each other in only two levels, namely the first and the last. Further, since inp' does not depend on the function being computed, $\hat{BP}_0$ and $\hat{BP}_1$ are *oblivious* matrix branching programs.

Accordingly, with respect to $\mathsf{Merge}(BP_0, BP_1, b)$ we will often use the phrase *active branching program* to refer to BP_b.

Claim 9. *For* $BP_0 = \{\mathsf{inp}(i), B^0_{i,0}, B^0_{i,1}\}_{i=1}^m$ *and* $BP_1 = \{\mathsf{inp}(i), B^1_{i,0}, B^1_{i,1}\}_{i=1}^m$ *each of width* w *and length* m *on* n *input bits, define* $\widehat{BP}_0$ *and* $\widehat{BP}_1$ *as above. Then, for each* $b \in \{0,1\}$, $x \in \{0,1\}^n$,

$$\prod_{i=1}^{m+2} \widehat{B}^b_{i,x[\mathsf{inp}'(i)]} = \begin{pmatrix} \prod_{i=1}^m B^b_{i,x[\mathsf{inp}(i)]} & 0 \\ 10 & \prod_{i=1}^m B^{1-b}_{i,x[\mathsf{inp}(i)]} \end{pmatrix}$$

Proof. We observe that $\widehat{BP}_0$ and $\widehat{BP}_1$ agree on each level except the first and last, that is,

$$\widehat{B}^0_{i,b} = \widehat{B}^1_{i,b} = \begin{pmatrix} B^0_{(i-1),b} & 0 \\ 0 & B^1_{(i-1),b} \end{pmatrix} \qquad \forall \quad i : 2 \le i \le m+1, \quad b \in \{0,1\}$$

Then we have, for any $x \in \{0,1\}^n$,

$$\prod_{i=2}^{m+1} \widehat{B}^0_{i,x[\mathsf{inp}'(i)]} = \prod_{i=2}^{m+1} \widehat{B}^1_{i,x[\mathsf{inp}'(i)]} = \prod_{i=2}^{m+1} \begin{pmatrix} B^0_{(i-1),x[\mathsf{inp}'(i)]} & 0 \\ 0 & B^1_{(i-1),x[\mathsf{inp}'(i)]} \end{pmatrix}$$
$$= \prod_{i=1}^{m} \begin{pmatrix} B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix}$$
$$= \begin{pmatrix} \prod_{i=1}^{m} B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^{m} B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix}$$

Where the change of indices in the second step follows because $\mathsf{inp}'(i) = \mathsf{inp}(i-1)$ when $2 \le i \le m+1$. We now consider the two case for $b \in \{0,1\}$.

Case 1: (b = 0)

In this case,

$$\prod_{i=1}^{m+2} \widehat{B}^0_{i,x[\mathsf{inp}'(i)]} = I_{2w\times 2w} \cdot \begin{pmatrix} \prod_{i=1}^{m} B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^{m} B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \cdot I_{2w\times 2w}$$
$$= \begin{pmatrix} \prod_{i=1}^{m} B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^{m} B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix}$$

as required.

Case 2: (b = 1)

In this case,

$$\prod_{i=1}^{m+2} \widehat{B}^1_{i,x[\mathsf{inp}'(i)]} = \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \cdot \begin{pmatrix} \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \cdot \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \\ \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix}$$

$$= \begin{pmatrix} \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} \end{pmatrix}$$

as required. □

Claim 10. *For all BP_0 and BP_1 each of width w and length m on n input bits, for each $b \in \{0,1\}$, for all $x \in \{0,1\}^n$,*

$$\mathsf{Merge}(BP_0, BP_1, b)(x) = BP_b(x)$$

Proof. Let $BP_0 = \{\mathsf{inp}(i), B^0_{i,0}, B^0_{i,1}\}_{i=1}^m$ and $BP_1 = \{\mathsf{inp}(i), B^1_{i,0}, B^1_{i,1}\}_{i=1}^m$. Define $\hat{BP}_0 = \mathsf{Merge}(BP_0, BP_1, 0)$ and $\hat{BP}_1 = \mathsf{Merge}(BP_0, BP_1, 1)$ as above. We observe that for any $x \in \{0,1\}^n$,

$$\begin{aligned}
&\mathsf{Merge}(BP_0, BP_1, b)(x) = 1 \\
\iff &(\prod_{i=1}^{m+2} \widehat{B}^b_{i,x[\mathsf{inp}'(i)]}) \cdot \mathbf{e}_1 = \mathbf{e}_1 \\
\iff &\begin{pmatrix} \prod_{i=1}^m B^b_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^{1-b}_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \cdot \mathbf{e}_1 = \mathbf{e}_1 \qquad \text{(from Claim 9)} \\
\iff &(\prod_{i=1}^{m} B^b_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1 = \mathbf{e}_1 \\
\iff &BP_b(x) = 1
\end{aligned}$$

Thus $\mathsf{Merge}(BP_0, BP_1, b)(x) = BP_b(x)$. □

The following claim illustrates some useful properties of the Merge procedure that we will use later. Firstly it notes that changing the bit Merge gets as input changes only the "switch" matrices in the first and last level of the program Merge outputs. Secondly, changing one level of a program Merge gets as input changes the output program in one level only. Finally, the first column of the output matrix of the widened program output by Merge depends only on the first column of the output matrix of the active program. The claim follows by observing the definition of Merge.

Claim 11. *Let BP_0 and BP_1 be length m, width w branching programs, with input length n.*

- $\mathsf{Merge}(BP_0, BP_1, 0)$ *and* $\mathsf{Merge}(BP_0, BP_1, 1)$ *differ in only 4 matrices, the matrices corresponding to the first and last level.*
- *Let BP'_1 be a length m branching program that differs from BP_1 in only the i^{th} level for some $i \in [m]$. Then for both $b \in \{0,1\}$, $\mathsf{Merge}(BP_0, BP_1, b)$ and $\mathsf{Merge}(BP_0, BP'_1, b)$ also differ only in the i^{th} level. A similar statement holds for branching programs BP'_0 that differ from BP_0 in only one level.*
- *For any $b \in \{0,1\}$, let $BP = \mathsf{Merge}(BP_0, BP_1, b)$, and $\mathsf{P_{out}}^{BP}(\cdot)$ and $\mathsf{P_{out}}^{BP_b}(\cdot)$ be the functions computing the output matrices on a given input for BP and BP_b respectively. Then for every input $x \in \{0,1\}^n$,*

$$\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{out}}^{BP_b}(x)))$$

where extend *extends a length w vector by appending w zeroes to the end.*

Let us emphasize that even if BP_0 and BP_1 have fixed accept and reject matrices, $\mathsf{Merge}(BP_0, BP_1, b)$ may no longer be a branching program with fixed accept and reject matrices; however, it will be a branching program having fixed output column (as required by Definition 6).

The Construction

In this section we show how to construct an indistinguishability obfuscator for the class C^1, given a neighboring-matrix indistinguishability obfuscator for branching programs Obf. By Lemma 6, iO can be converted into indistinguishability obfuscator for NC^1.

Description of $iO\,(1^n, C)$:

1. iO verifies that input $C \in C_n^1$ (that is, C is a circuit with size at most n and depth at most $\log(n)$), and aborts otherwise.
2. iO uses Barrington's Theorem to convert C into an oblivious width 5 permutation branching program. It pads this branching program as follows: First, it increases the number of input bits to the branching program to n. Next, it adds dummy levels to the end of the branching program until its length is the same as the longest branching program for a circuit in C_n^1 (which is $O(4^{\log(n)}) = O(n^2)$). Then, for every level in the branching program, it replaces it with n dummy levels that read every bit of the input in sequential order, inserting the original level into the corresponding position in this sequence.

   This procedure ensures that every padded branching program for a circuit in C_n^1 has the same length, same number of input bits, and the same input

labelling function inp as the padded branching program for any other circuit in C^1_n. Let the padded branching program be $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^{m}$.

3. iO generates a dummy width-5 branching program $I = \{\mathsf{inp}(i), I_{5\times5}, I_{5\times5}\}_{i=1}^{m}$ of length m, where each permutation matrix at each level is the identity matrix. iO then computes $\widehat{BP} = \mathsf{Merge}(BP, I, 0)$.

4. iO outputs $\mathsf{Obf}(\widehat{BP})$.

Proof of security

Theorem 12. *There exists a ensemble of classes of branching programs $\mathcal{B}$ such that if there exists a neighboring-matrix indistinguishability obfuscator for $\mathcal{B}$ then there exist indistinguishability obfuscators for* NC^1.

Proof. We first define the ensemble of classes $\mathcal{B} = \{\mathcal{B}_n\}_{n\in\mathcal{N}}$. The class $\mathcal{B}_n$ is simply the class of all matrix branching programs of width 10, and length $n^3 + 2$ such that for every input x it holds that

$$(\prod_{i=1}^{m} B_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1 = \begin{cases} \mathbf{e}_1 & \text{when } BP(x) = 1 \\ \mathbf{e}_k & \text{when } BP(x) = 0 \end{cases}$$

where $k \neq 1$ is such that $\mathbf{e}_k = \mathsf{extend}(\mathsf{P}_{\mathsf{reject}} \cdot \mathbf{e}_1)$ and $\mathsf{P}_{\mathsf{reject}}$ is the rejecting matrix from Theorem 7.

Let Obf be a neighboring-matrix indistinguishability obfuscator for $\mathcal{B}$, and let iO be the obfuscator relying on Obf constructed in Section 3.4.2 . We will show iO is a indistinguishability obfuscator for C^1; by Lemma 6, this implies the existence of indistinguishability obfuscators for NC^1.

Assume for contradiction that there exists a nuPPT distinguisher D and polynomial p such that for infinitely many n, there exist functionally equivalent circuits $C_n^0, C_n^1 \in \mathcal{C}_n^1$ such that D distinguishes $i\mathcal{O}(1^n, C_n^0)$ and $i\mathcal{O}(1^n, C_n^1)$ with advantage $1/p(n)$. For any $n \in \mathbb{N}$, let BP_0 and BP_1 be the branching programs of length $m = poly(n)$ obtained by applying Theorem 7 to the circuits C_n^0 and C_n^1 respectively, and padding them so they have the same length and same input labelling function.

Let Hyb_i be a procedure that takes as input two length m branching programs P_0 and P_1 (with the same labeling function) and outputs a "hybrid" length m branching program whose first i levels are identical to the first i levels of P_0 and all the other levels are identical to those of P_1. Formally, let $P_0 = \{\mathsf{inp}(j), B_{j,0}, B_{j,1}\}_{j\in[m]}$ and $P_1 = \{\mathsf{inp}(j), B'_{j,0}, B'_{j,1}\}_{j\in[m]}$.

$$\mathsf{Hyb}_i(P_0, P_1) = \{\mathsf{inp}(j), B_{j,0}, B_{j,1}\}_{j=1}^{i}, \{\mathsf{inp}(j), B'_{j,0}, B'_{j,1}\}_{j=i+1}^{m}$$

For every $n \in \mathbb{N}$ we define hybrid distributions in the following way.

- We start with H_0 which is the obfuscation of the circuit C_n^0.

$$H_0 = i\mathcal{O}(1^n, C_n^0) = \mathsf{Obf}(\mathsf{Merge}(BP_0, I, 0))$$

- For $i = 1, 2 \ldots m$, let

$$H_i = \mathsf{Obf}(\mathsf{Merge}(BP_0, \mathsf{Hyb}_i(BP_1, I), 0))$$

  We change, one level at a time, the second branching program Merge takes as input from I to BP_1.

- We have that $H_m = \mathsf{Obf}(\mathsf{Merge}(BP_0, BP_1, 0))$. We change the "switch" input to Merge so that the second branching program BP_1 is active.

$$H_{m+1} = \mathsf{Obf}(\mathsf{Merge}(BP_0, BP_1, 1))$$

- For $i = 1, 2 \ldots m$, let

$$H_{m+i+1} = \mathsf{Obf}(\mathsf{Merge}(\mathsf{Hyb}_i(BP_1, BP_0), BP_1, 1))$$

  We change the first program Merge takes as input from BP_0 to BP_1, one level at a time as before.

- We have that $H_{2m+1} = \mathsf{Obf}(\mathsf{Merge}(BP_1, BP_1, 1))$. We switch back so that the first program is active (which in this case is the same as the second program BP_1)

$$H_{2m+2} = \mathsf{Obf}(\mathsf{Merge}(BP_1, BP_1, 0))$$

- For $i = 1, 2 \ldots m$, let

$$H_{2m+i+2} = \mathsf{Obf}(\mathsf{Merge}(BP_1, \mathsf{Hyb}_i(I, BP_1), 0))$$

  We change the second program Merge takes as input from BP_1 to I, one level at a time as before. Finally we get

$$H_{3m+2} = i\mathcal{O}\,(1^n, C_n^1) = \mathsf{Obf}(\mathsf{Merge}(BP_1, I, 0))$$

  which is the obfuscation of the circuit C_n^1.

Recall that by assumption D distinguishes between $\{i\mathcal{O}\ (1^n, C_n^0)\}_{n\in\mathbb{N}}$ and $\{i\mathcal{O}\ (1^n, C_n^1)\}_{n\in\mathbb{N}}$. That is, there is a polynomial p such that for infinitely many n

$$|Pr[D(1^n, H_0) = 1] - Pr[D(1^n, H_{3m+2})]| > 1/p(n)$$

By the above hybrid argument, D must distinguish between a pair of consecutive hybrids. That is, there exists some $i \in \{0, 1, \ldots 3m + 1\}$ such that

$$|Pr[D(1^n, H_i) = 1] - Pr[D(1^n, H_{i+1})]| > 1/4mp(n)$$

We now show that H_i and H_{i+1} can be expressed as the $\mathsf{Obf}(BP)$ and $\mathsf{Obf}(BP')$ respectively where BP and BP' are relaxed matrix branching programs that differ in at most 4 matrices, agree on all inputs and come from $\mathcal{B}_n$.

Claim 13. *For every n, there exist branching programs $BP, BP' \in \mathcal{B}_n$ such that*

- $H_i = \mathsf{Obf}(BP)$ *and* $H_{i+1} = \mathsf{Obf}(BP')$.
- *BP and BP' differ in at most 4 matrices.*
- *For all x, $BP(x) = BP'(x)$.*

Proof. We consider three cases: when i is equal to m, $2m + 1$ and otherwise.

Case 1: $i = m$: By definition of H_i and H_{i+1}, the branching programs BP and BP' are $\mathsf{Merge}(BP_0, BP_1, 0)$ and $\mathsf{Merge}(BP_0, BP_1, 1)$ respectively. By Claim 11, BP and BP' differ in the "switch" matrices, which make up 4 matrices (the first and last level). Furthermore, BP and BP' compute BP_0 and BP_1 respectively which are equivalent programs by assumption. It remains to show that $BP, BP' \in \mathcal{B}_n$. Note that BP and BP' have width 10 and length $n^3 + 2$. By Claim 11, the first column of the output matrix for a merged branching program only depends on the first column of the output matrix of the active program. Hence, for every input x, $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP_0}(x)))$. By Theorem 7, $\mathsf{P}_{\mathsf{out}}{}^{BP_0}(x)$ is either $\mathsf{P}_{\mathsf{accept}}$ or $\mathsf{P}_{\mathsf{reject}}$ depending on the output $BP_0(x)$. Therefore, for all inputs x such that $BP(x) = 0$,

$$\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P}_{\mathsf{reject}})) = \mathbf{e}_k$$

Similarly, for all inputs x such that $BP(x) = 1$,

$$\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P}_{\mathsf{accept}})) = \mathbf{e}_1$$

The same argument holds for BP' too, in which case BP_1 is active and has the same accepting and rejecting permutations $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$ by Theorem 7.

Case 2: $i = 2m + 1$**:** By definition of H_i and H_{i+1}, the branching programs BP and BP' are $\mathsf{Merge}(BP_1, BP_1, 0)$ and $\mathsf{Merge}(BP_1, BP_1, 1)$ respectively. As before, these programs differ in the 4 matrices only. Furthermore, both BP and BP' compute the same function, as the active program is the same (BP_1). Also as before, from Claim 11 and Theorem 7 we have that for all inputs x,

$$\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP}(x)) = \mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP'}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP_1}(x))) = \mathbf{e}_t$$

where $t = 1$ if $BP_1(x) = 1$ and $t = k$ otherwise.

Case 3: $i \neq m$ **and** $i \neq 2m + 1$**:** First, consider the subcase when $i < m$ or $i > 2m + 1$. The programs BP and BP' are of the form $\mathsf{Merge}(BP_0, P_i)$ and $\mathsf{Merge}(BP_0, P_{i+1})$ respectively where P_i and P_{i+1} are branching programs that differ only in the $i + 1^{th}$ level. By Claim 11, BP and BP' differ only in the $i + 1^{th}$ level too. Furthermore, in both BP and BP', the active program is BP_0. Hence BP and BP' compute the same function and similarly as the previous case, we have that for all inputs x,

$$\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP}(x)) = \mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP'}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}{}^{BP_0}(x))) = \mathbf{e}_t$$

where $t = 1$ if $BP_1(x) = 1$ and $t = k$ otherwise. The case when $m < i < 2m + 1$ follows similarly. This concludes the proof of the claim. □

Therefore we have that there is a polynomial p' such that for infinitely many n there exist functionally equivalent branching programs $BP, BP' \in \mathcal{B}_n$ that differ in only a few matrices such that

$$|Pr[D(1^n, \mathsf{Obf}(BP)) = 1] - Pr[D(1^n, \mathsf{Obf}(BP'))]| > 1/p'(n)$$

This implies Obf is not a neighboring-matrix indistinguishability obfuscator for $\mathcal{B}$ and hence a contradiction. □

3.4.3 From Semantic Security to *nm-iO*

In this section we show that assuming the existence of semantically secure multilinear encodings, there exists a neighboring-matrix indistinguishability obfuscator for any ensemble of classes of branching programs.

As in previous works [70, 50, 16], the strategy for our construction will be to apply Kilian's randomization technique to the matrices, and then encode these matrices using the graded encoding scheme. The encoding will be using a so-called "straddling set system" (as in [16]) that will enforce that any arithmetic circuit operating on these encodings can be decomposed into a sum of terms such that each term can be expressed using only encodings that come from one branch of the branching program (more specifically, from the path through the branching program corresponding to evaluating a particular input x to the branching program).

As mentioned in the introduction, although we will closely follow techniques from [50, 16] (our obfuscator may be viewed as a simplified version of the obfuscator from [16]), we cannot directly rely on their proofs for two reasons:

1. The proofs in [50, 16] rely on the fact that we are only obfuscating branching programs with fixed accept and reject matrices; as mentioned, we need to handle more general classes of branching programs.
2. The proofs in [50, 16] only reason about *polynomial-size* generic attackers.

In contrast, to rely on semantical security, we need to reason about *unbounded* arithmetic circuits.

Randomizing Branching Programs

We start by describing Kilian's randomization technique [108] for a branching program, adapted to our setting, by defining a process Rand that randomizes the matrices of a branching program BP. We will decompose the randomization into two parts, Rand^B and Rand^α, defined below, and define Rand as their composition.

Definition 15 (Rand^B)**.** *Let $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^m$ be a branching program of width w and length m, with length-n inputs. Let p be a prime exponential in n. Then the process $\mathsf{Rand}^B(BP, p)$ samples m random invertible matrices $R_1, R_2, \ldots, R_m \in Z_p^{w\times w}$ uniformly and independently, and computes*

$$\tilde{B}_{i,b} = R_{(i-1)} \cdot B_{i,b} \cdot R_i^{-1} \quad \text{for every } i \in [m], \text{ and } b \in \{0,1\}$$

where R_0 is defined as $I_{w\times w}$, and

$$\mathbf{t} = R_m \cdot \mathbf{e}_1$$

Rand^B *then outputs*

$$(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}, p)$$

Definition 16 (Rand^α)**.** *Let $(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}, p)$ be the output of $\mathsf{Rand}^B(BP, p)$ as defined above. On this input, $\mathsf{Rand}^\alpha(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, p)$ samples $2m$ non-zero scalars $\{\alpha_{i,b} \in \mathcal{Z}_p : i \in [m], b \in \{0,1\}\}$ uniformly and independently, and outputs*

$$(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$$

Definition 17 (Rand). *Let* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^{m}$ *be a branching program of width* w *and length* m*, with length-*n *inputs. Let* p *be a prime exponential in* n*. Then we define* $\mathsf{Rand}(BP, p)$ *to be:*

$$\begin{aligned}\mathsf{Rand}(BP, p) &= (\mathsf{Rand}^{\alpha}(\mathsf{Rand}^{B}(BP, p))) \\ &= (\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})\end{aligned}$$

Where $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$ *are as computed in the executions of* Rand^{α} *and* Rand^{B}*.*

Execution of a randomized branching program: To compute $BP(x)$ from the output of $\mathsf{Rand}(BP, p)$, given some input labelling function $\mathsf{inp} : [m] \to [n]$, and $x \in \{0,1\}^n$, we compute

$$\mathsf{Out}(x) = (\prod_{i=1}^{m} \alpha_{i,x[\mathsf{inp}(i)]} \cdot \tilde{B}_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{t}$$

Where $\mathsf{Out} \in Z_P^w$ is a $w \times 1$ vector. The intermediate multiplications cause each R_i^{-1} to cancel each R_i, and $R_0 = I_{w\times w}$, so the above computation can also be expressed as:

$$\mathsf{Out}(x) = (\prod_{i=1}^{m} \alpha_{i,x[\mathsf{inp}(i)]} \cdot B_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1$$

When $BP(x) = 1$, we have that

$$\prod_{i=1}^{m} \alpha_{i,x[\mathsf{inp}(i)]} \cdot B_{i,x[\mathsf{inp}(i)]} \cdot \mathbf{e}_1 = (\prod_{i=1}^{m} \alpha_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1$$

When $BP(x) = 0$, we have that

$$\prod_{i=1}^{m} \alpha_{i,x[\mathsf{inp}(i)]} \cdot B_{i,x[\mathsf{inp}(i)]} \cdot \mathbf{e}_1 = (\prod_{i=1}^{m} \alpha_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_k$$

for $k \neq 1$. Hence, to compute $BP(x)$, we compute $\mathsf{Out}(x)$ and output 0 if $\mathsf{Out}(x)[1] = 0$, and 1 otherwise.

Simulating a randomized branching program: Previous works ([16, 50]) followed [108] to show how to simulate the distribution of any single path corresponding to an input x using just $BP(x)$. However, the simulator required that branching programs have unique accept and reject matrices $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$.

We would also like a theorem, along the lines of [108], that shows that any single path through a randomized branching program BP corresponding to an input x can be simulated knowing just the accept/reject behavior of BP on x (i.e. by knowing whether $BP(x) = 1$).

In our setting, however, branching programs only meet the relaxed requirement that the output matrix $\mathsf{P}_{\mathsf{out}}(x)$ computed by evaluating BP on input x satisfies $\mathsf{P}_{\mathsf{out}}(x) \cdot \mathbf{e}_1 = \mathbf{e}_1 \iff BP(x) = 1$. There can thus be multiple accept and reject matrices, and the particular accept or reject matrix output by BP can depend both on x and on the specific implementation of BP (and not simply its accept/reject behavior). In contrast, in previous works, because $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$ were unique, knowing just the accept/reject behavior of BP on x also determines $\mathsf{P}_{\mathsf{out}}(x)$.

What we will show is that, for the particular randomization scheme chosen above, we can simulate any single path through a randomized branching program BP corresponding to an input x without knowing the exact accept/reject matrix $\mathsf{P}_{\mathsf{out}}(x)$, but rather just knowing the first column $\mathsf{p}_{\mathsf{out}}(x) = \mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$.

This will be sufficient for our applications, because the class of branching programs we randomize will have the property that there are fixed columns $\mathsf{p}_{\mathsf{accept}}$ and $\mathsf{p}_{\mathsf{reject}} \in \mathbb{Z}_p^w$ such that for all $x \in \{0,1\}^n$, if $BP(x) = 1$ then $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = \mathsf{p}_{\mathsf{accept}}$, and if $BP(x) = 0$ then $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = \mathsf{p}_{\mathsf{reject}}$. In the case of such programs,

$\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$ is determined solely by $BP(x)$, and not the particular implementation of BP. Thus, for these programs, we can simulate given only $BP(x)$.

Before we show this theorem, we define notation for a path through a branching program corresponding to an input x.

Definition 18 (proj_x)**.** *Let* $\mathsf{inp} : [m] \to [n]$ *be an input labelling function, and, for any* $x \in \{0,1\}^n$*, define* proj_x*, relative to* inp*, such that for any branching program BP with labelling function* inp*, for any prime* $p \in \mathcal{N}$*, and for any* $(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}^B(BP, p)$

$$\mathsf{proj}_x(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = (\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t}),$$

that is, proj_x *selects the elements from* $(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$ *used when evaluating input x.*

We now show a version of Kilian's theorem, adapted to our construction.

Theorem 14. *There exists an efficient simulator* KSim *such that the following holds. Let* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i\in[m]}$ *be a width-w branching program of length m on n bit inputs, and p a prime exponential in n. Let* $x \in \{0,1\}^n$ *be an input to BP, and let* $b_i = x[\mathsf{inp}(i)]$ *for each* $i \in [m]$*. Let* $\mathsf{P}_{\mathsf{out}}(x) = \prod_{i=1}^{m} B_{i,b_i}$ *denote the matrix obtained by evaluating BP on x, and let* $\mathsf{p}_{\mathsf{out}}(x) = \mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$ *denote the first column of this output. Let* $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$ *be defined respecting the labelling function* inp*. Then* $\mathsf{KSim}(1^m, p, \mathsf{p}_{\mathsf{out}}(x))$ *is identically distributed to* $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$*.*

Proof. We begin by defining $\mathsf{KSim}(1^n, p, BP(x))$ as follows:

- For each i, KSim selects $\tilde{B}_{i,b_i}$ to be a uniformly random invertible matrix in $Z_p^{w\times w}$.

- KSim selects $\mathbf{t} \in \mathcal{Z}_p^w$ solving

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot \mathbf{t} = \mathsf{p}_{\mathsf{out}}(x) \tag{3.1}$$

  where $b_i = x[\mathsf{inp}(i)]$ for each i.

- KSim outputs $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, \mathbf{t}\}$

We want to show that the distribution output by KSim matches the real distribution of $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, \mathbf{t}\}$ in the output of $\mathsf{Rand}^B(BP, p)$. But from [108], we have the following:

Claim 15. *The distribution of $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, R_m\}$ can be exactly sampled given $\mathsf{P}_{\mathsf{out}}(x)$, by sampling $\{\tilde{B}_{i,b_i}\}_{i\in[m]}$, R_m to be uniformly random and independent invertible matrices in $\mathcal{Z}_p^{w\times w}$ subject to*

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot R_m = \mathsf{P}_{\mathsf{out}}(x) \tag{3.2}$$

The above claim implies the following:

Claim 16. *The distribution of $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, R_m\}$ can be sampled by independently choosing each $\tilde{B}_{i,b_i}$ uniform and invertible, and fixing R_m solving equation (3.2).*

Proof. This follows because for every choice of invertible $\tilde{B}_{i,b_i}$, there exists R_m solving equation (3.2) given by

$$R_m = (\prod_{i\in[m]} \tilde{B}_{i,b_i}))^{-1} \cdot \mathsf{P}_{\mathsf{out}}(x) \tag{3.3}$$

Further, every solution to equation (3.2) can be represented as invertible $\tilde{B}_{i,b_i}$, and an R_m solving equation (3.3). Thus choosing a random solution to equation

(3.2) corresponds to independently choosing each $\tilde{B}_{i,b_i}$ uniformly and invertible, and fixing R_m solving equation (3.3). □

From the above argument, we have that the distribution of $\mathsf{proj}_x(\mathsf{Rand}(BP, p))$ is exactly the same as the distribution produced by independently choosing each $\tilde{B}_{i,b_i}$ uniform and invertible, fixing R_m solving equation (3.3), setting $\mathbf{t}$ to be the first column of R_m, and outputting $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, \mathbf{t}\}$. But note that each column $\mathsf{col}_i(R_m), i \in [w]$ is the unique solution to

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot \mathsf{col}_i(R_m) = \mathsf{col}_i(\mathsf{P}_{\mathsf{out}}(x))$$

Thus we have that each $\tilde{B}_{i,b_i}$ is independent, uniform, and invertible, and, using $i = 1$, $\mathbf{t}$ is the unique solution to

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot \mathbf{t} = \mathsf{p}_{\mathsf{out}}(x)$$

and, in particular, that $\mathbf{t}$ is determined by *only* the first column of $\mathsf{P}_{\mathsf{out}}(x)$. Thus, we see that the distribution of $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$ is exactly the same as that output by KSim. □

Choosing a Set System

In this section we will describe how to choose a collection of sets under which to encode a randomized branching program using the graded encoding scheme. Our selection of sets will closely follow [16], in that we use straddling set systems. However, one difference is that while they use dual input branching programs, we restrict our attention to single-input schemes. As a consequence, the sets will be simpler and consist of fewer elements.

We first define straddling set systems.

Definition 19 (Straddling Set Systems [16])**.** *A* straddling set system *with n entries is a collection of sets* $\mathbb{S}_n = \{S_{i,b} : i \in [n], b \in \{0,1\}\}$ *over a universe U, such that:*

$$\bigcup_{i\in[n]} S_{i,0} = \bigcup_{i\in[n]} S_{i,1} = U$$

and for every distinct non-empty sets $C, D \subseteq \mathbb{S}_n$*, we have that if:*

1. *(Disjoint Sets:) C contains only disjoint sets. D contains only disjoint sets.*
2. *(Collision:)* $\bigcup_{S\in C} S = \bigcup_{S\in D} S$

Then it must be that $\exists b \in \{0,1\}$ *such that:*

$$C = \{S_{j,b}\}_{j\in[n]} \quad , \quad D = \{S_{j,(1-b)}\}_{j\in[n]}$$

Informally, the guarantee provided by a straddling set system is that only way to exactly cover U using elements from $\mathbb{S}_n$ is to use either all sets $\{S_{i,0}\}_{i\in n}$ or all sets $\{S_{i,1}\}_{i\in n}$. We use a slight variant of their construction, choosing U to be $[2n]$, each $S_{i,0}$ to be one of $\{1,2\},\{3,4\},\ldots,\{2n-1,2n\}$, and each $S_{i,1}$ to be one of $\{1,2n\},\{2,3\},\{4,5\}\ldots,\{2n-2,2n-1\}$.[25] By a proof exactly following [16], we have that this construction is a straddling set system.

Theorem 17 (Following Construction 1 in [16])**.** *For every* $n \in N$*, there exists a straddling set system* $\mathbb{S}_n$ *with n entries, over a universe U of* $2n$ *elements; furthermore, each set in the straddling set system has size exactly two.*

We now define the process SetSystem which takes as input the length m of a branching program, the number of input bits n, and the input labelling function

[25] In the construction of [16], $U = [2n-1]$, and each $S_{i,0}$ is one of $\{1\},\{2,3\},\ldots,\{2n-2,2n-1\}$, and each $S_{i,1}$ is one of $\{1,2\},\{3,4\},\ldots,\{2n-1\}$. We could have also worked with this construction, but modify it slightly to ensure that all encodings are under sets of size exactly two.

$\mathsf{inp} : [m] \to [n]$ for a branching program. SetSystem will output the collection of straddling set systems that we will use to encode any branching program of length m on n input bits, with labelling function inp.

Execution of SetSystem(m, n, inp)**:**

We let n_j denote the number of levels that inspect the jth input bit in inp. That is,

$$n_j = |\{i \in [m] : \mathsf{inp}(i) = j\}|$$

For every $j \in [n]$, SetSystem chooses $\mathbb{S}^j$ to be a straddling set system with n_j entries over a set U_j, such that the sets $U_1, \ldots, U_n$ are disjoint. Let $U = \bigcup_{j\in[n]} U_j$. SetSystem then chooses S_t be a set of two elements[26], disjoint from U. We associate the set system $\mathbb{S}^j$ with the j'th input bit of the branching program corresponding to inp. SetSystem then re-indexes the elements of $\mathbb{S}^j$ to match the steps of the branching program as described by inp, so that:

$$\mathbb{S}^j = \{S_{i,b} : \mathsf{inp}(i) = j, b \in \{0, 1\}\}$$

By this indexing, we also have that $S_{i,b} \in \mathbb{S}^{\mathsf{inp}(i)}$ for every $i \in [m]$, for every $b \in \{0, 1\}$.

Let $k = |U \cup S_t|$, and WLOG, assume that the U^js and S_t are disjoint subsets of $[k]$ (otherwise SetSystem relabels the elements to satisfy this property).

SetSystem then outputs

$$k, \quad \{S_{i,b}\}_{i\in[m],b\in\{0,1\}}, \quad S_t$$

[26]We make this choice to ensure that every set in the output of SetSystemconsists of exactly two indices $\{i, j\}$ for $i, j \in [k]$

The Construction

We finally describe our neighboring-matrix indistinguishability obfuscator Obf for branching programs. Obf will use Rand and SetSystem as subroutines.

Description of Obf(BP) :

Input. Obf takes as input an oblivious permutation branching program $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^{m}$ of width w, length m and taking n input bits.

Choosing sets. Obf runs SetSystem(m, n, inp) and receives $k, \{S_{i,b}\}_{i\in[m+2],b\in\{0,1\}}, S_t$.

Initializing the GES. Obf runs InstGen($1^n, 1^k$) and receives secret parameters sp and public parameters pp which describe a (k, R)-graded encoding scheme. We assume the ring R is equal to $\mathbb{Z}_p$ for some p exponential in n and k.

Randomizing BP. Obf executes Rand(BP, p), and obtains its output, $\{\{\mathsf{inp}(i), \alpha_{i,0} \cdot \tilde{B}_{i,0}, \alpha_{i,1} \cdot \tilde{B}_{i,1}\}_{i\in[m]}, \mathbf{t}\}$

Output. Obf outputs:

$$\mathsf{pp}, \quad \{\mathsf{inp}(i), \quad \mathsf{Enc}(\mathsf{sp}, \alpha_{i,0}\cdot\tilde{B}_{i,0}, S_{i,0}), \quad \mathsf{Enc}(\mathsf{sp}, \alpha_{i,0}\cdot\tilde{B}_{i,0}, S_{i,1})\}_{i\in[m]}, \quad \mathsf{Enc}(\mathsf{sp}, \mathbf{t}, S_t)$$

We also define a generic version of Obf, which we refer to as GObf. Its output will be used to initialize an oracle $\mathcal{M}$ for the idealized version of the graded encoded scheme. GObf(BP, pp) acts exactly as Obf(BP), except that it works with a fixed public parameter pp supplied as input, and in the **Output** step, GObf outputs

$$\mathsf{pp}, \quad \{\mathsf{inp}(i), (\alpha_{i,0} \cdot \tilde{B}_{i,0}, S_{i,0}), (\alpha_{i,1} \cdot \tilde{B}_{i,1}, S_{i,1})\}_{i\in[m]}, \quad (\mathbf{t}, S_t)$$

that is, the output before it is encoded under the multilinear encoding scheme.

Proof of security

We show that Obf defined in Section 3.4.3 is a neighboring-matrix indistinguishability obfuscator for any ensemble of classes of branching programs, if the underlying multilinear encodings are semantically secure.

Theorem 18. *Assume the existence of an entropic semantically secure multilinear encoding scheme. Then there exist a neighboring-matrix indistinguishability obfuscator for any ensemble of classes of branching programs.*

Proof. Consider any ensemble $\mathcal{B} = \{\mathcal{B}_n\}_{n\in\mathcal{N}}$ of classes of branching programs. We show that the obfuscator Obf is a neighboring-matrix indistinguishability obfuscator for $\mathcal{B}$. Assume for contradiction there exist a pair of ensembles $\{BP_n^0\}_{n\in\mathcal{N}}, \{BP_n^1\}_{n\in\mathcal{N}}$ nuPPT D and polynomial p such that for infinitely many n, BP_n^0, BP_n^1 are functionally equivalent programs in $\mathcal{B}_n$ that differ in at most 4 matrices and

$$|Pr[D(1^n, \mathsf{Obf}(BP_n^0)) = 1] - Pr[D(1^n, \mathsf{Obf}(BP_n^1))]| > 1/p(n)$$

We will show that the semantic security of the multilinear encodings used by Obf implies a contradiction. In particular, we construct a message sampler M which samples $(\vec{m}_0, \vec{m}_1, \vec{z})$ such that $\mathsf{Obf}(BP_n^0)$ is simply the encoding of $(\vec{m}_0, \vec{z})$ and $\mathsf{Obf}(BP_n^1)$ is the encoding of $(\vec{m}_1, \vec{z})$. We then show that if BP_n^0 and BP_n^1 agree on all inputs, then the message sampler M is valid in the sense of Definition 11 and therefore D breaks the semantic security of the encoding scheme used, hence a contradiction.

Fix $n \in \mathbb{N}$, and let $BP_n^0 = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i\in[m]}$ and $BP_n^1 = \{\mathsf{inp}(i), B'_{i,0}, B'_{i,1}\}_{i\in[m]}$. Let $L \subset [m]\times\{0,1\}$ be the set of indices of those matrices in which BP_n^0 and BP_n^1 differ.

Note that by assumption $|L| = 4$. All other matrices of BP_n^0 and BP_n^1 are the same.

Let $(k, \{S_{i,b}\}_{i\in[m],b\in\{0,1\}}, S_{\mathbf{t}}) = \mathsf{SetSystem}(m, n', \mathsf{inp})$ where n' is the input length of the branching programs BP_n^0, BP_n^1, and let

$$\vec{S}_n = \{S_l\}_{l\in L}$$

$$\vec{T}_n = (\{S_l\}_{l\notin L}, S_{\mathbf{t}})$$

We now define a message sampler M as follows. When run with security parameter 1^n, M gets BP_n^0 and BP_n^1 as non-uniform advice. On input 1^n, public parameters pp that describe a $(k, \mathbb{Z}_p)$-graded encoding scheme, M samples m random invertible 10×10 matrices over $\mathbb{Z}_p$, $\{R_i\}_{i\in[m]}$ and $2m$ random scalars from $\mathbb{Z}_p$, $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$. M then uses these matrices and scalars to randomize BP_n^0 and BP_n^1 as described by $\mathsf{Rand}(\cdot, p)$ to obtain $\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m'],b\in\{0,1\}}$, $\{\alpha_{i,b} \cdot \tilde{B}'_{i,b}\}_{i\in[m'],b\in\{0,1\}}$ and $\mathbf{t}$. M outputs

$$\vec{m_0} = \{\alpha_l \cdot \tilde{B}_l\}_{l\in L}$$

$$\vec{m_1} = \{\alpha_l \cdot \tilde{B'}_l\}_{l\in L}$$

$$\vec{z} = (\{\alpha_l \cdot \tilde{B}_l\}_{l\notin L}, \mathbf{t})$$

We observe that $D(1^n, \mathsf{Obf}(BP_n^b))$ is simply the output of D when playing the semantic security game in Definition 12 parameterized by the bit b with the message sampler M and sets $(\vec{S}_n, \vec{T}_n)$ (as defined above). To see this, observe that the distribution of $(\vec{m_0}, \vec{z})$ is identical to $\mathsf{Rand}(BP_n^0, p)$ and the distribution of $(\vec{m_1}, \vec{z})$ is identical to $\mathsf{Rand}(BP_n^1, p)$. When these elements are encoded under sets $\vec{S}_n, \vec{T}_n$ then we obtain the distributions $\mathsf{Obf}(BP_n^0)$ and $\mathsf{Obf}(BP_n^1)$ respectively.

Recall that for infinitely many n,

$$|Pr[D(1^n, \mathsf{Obf}(BP_n^0)) = 1] - Pr[D(1^n, \mathsf{Obf}(BP_n^1))]| > 1/p(n)$$

Since the graded encoding scheme is semantically secure, and $|\vec{S}_n| \in O(1)$ and $|\vec{T}_n| \in O(k)$, it must be that M is not a $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting message sampler. In the remainder of the proof we show that if BP and BP' agree on all inputs then M is a $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting message sampler, hence implying a contradiction. Similar statements were shown in [16] and [50]. In particular, GObf is a simplified version of the obfuscator of [16], which [16] shows is VBB secure against algebraic adversaries. We will follow the structure of the proof in [16], but cannot use it in a black-box way due to the differences in the construction and the fact that their proof only works for branching programs that have unique accepting and rejecting output matrices. The branching programs we consider may not have this property.

To prove that M is a $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting message sampler we need to show that there exists a polynomial Q such that for every $n \in \mathbb{N}$, every $(\mathsf{sp}, \mathsf{pp})$ in the support of $\mathsf{InstGen}(1^n, 1^k)$, and every $(\vec{S}_n, \vec{T}_n)$-respecting arithmetic circuit C, there exists a constant $c \in \{0,1\}$ such that for any $b \in \{0,1\}$,

$$Pr[(\vec{m}_0, \vec{m}_1, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : \mathsf{isZero}(C(\vec{m}_b, \vec{z})) = c] \geq 1 - Q(n,k)/|R|.$$

where R is the ring associated with pp. We show that the result of applying any $(\vec{S}_n, \vec{T}_n)$-respecting arithmetic circuit C on $(\vec{m}_0, \vec{z})$ (resp. $(\vec{m}_1, \vec{z})$), can be *simulated* with overwhelming probability given just BP^0_n. This implies (by a union bound over $b \in \{0,1\}$) that for every such C there exists some bit c such that with overwhelming probability $C(\vec{m}_b, \vec{z}) = c$ for $b \in \{0,1\}$, and thus M is $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting. It suffices to show the following lemma and to note that BP^0_n and BP^1_n are functionally equivalent.

Lemma 19. *There exists a Turing machine* CSim *such that for every* $m, n, w \in \mathbb{N}$, $v_0, v_1 \in \{0,1\}^w$, *labeling function* $\mathsf{inp} : [m] \to [n]$, *prime number* p, *and* $\vec{S}$*-respecting*

arithmetic circuit C where $\vec{S} = \mathsf{SetSystem}(m, n, \mathsf{inp})$, the following holds. For every branching program BP of length m, width w and labeling function inp *for which on every input x, $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = v_{BP(x)}$ it holds that*

$$Pr[\mathsf{isZero}(C(\mathsf{Rand}(BP, p))) \neq \mathsf{CSim}^{BP}(1^m, p, C, v_0, v_1)] \leq 32wm/p$$

The proof of the lemma follows the structure of the VBB simulation in [16], appropriately adapted to deal with the fact that our branching programs do not have a unique output by relying on Theorem 14.

Proof. Roughly speaking the lemma follows from the the property that $\vec{S}$-respecting arithmetic circuits, due to the straddling set systems in $\vec{S}$, can only evaluate expressions that are "consistent" with some inputs. In particular, following [16], the polynomial evaluated by C can be expressed as the sum of *single-input terms* where each *single-input term* is a function of elements that are consistent with some single input to the branching program. Next, we rely on Theorem 14 to show that the sum of these single-input terms will depend only on the value of the branching program on these inputs.

The following proposition states that the function a $\vec{S}$-respecting arithmetic circuit computes can be expressed as the sum of several *single-input* terms. This decomposition is very similar to the one shown in [16].[27]

Proposition 1. *Fix $m, n, w \in \mathcal{N}$ and* $\mathsf{inp} : [m] \to [n]$*. Let $\vec{S} = \mathsf{SetSystem}(m, n, \mathsf{inp}) = (\{S_{i,b}\}_{i\in[m],b\in\{0,1\}}, S_t)$, and let C be any $\vec{S}$-respecting arithmetic circuit. There exists a set $X \subseteq \{0,1\}^n$ of inputs such that*

[27] The key difference is that [16] proves such a decomposition for "dual-input" branching program, and use the "dual-input" property to show that there are only polynomially many terms in the decomposition.

(i)

$$C(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \equiv \sum_{x\in X} C_x(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$$

where each C_x is a $\vec{S}$-respecting arithmetic circuit, whose input wires are labelled only with sets respecting a single input $x \in \{0,1\}^n$, that is, only with sets $\in \{S_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]} \cup \{S_t\}$.

(ii) *For each C_x above, for every branching program BP of width w and length m on n input bits, with input labelling function* inp*, every prime p, and every $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m,b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$*

$$C_x(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_x \cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$$

where p_x is some polynomial, and $\alpha_x = (\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})$. Furthermore, when p_x is viewed as a sum of monomials, each monomial contains exactly one entry from each $\tilde{B}_{i,x[\mathsf{inp}(i)]}$, and one entry from $\mathbf{t}$.

The proof of Proposition 1 uses the following lemma:

Lemma 20. *Fix $m, n, w \in \mathcal{N}$ and* $\mathsf{inp} : [m] \to [n]$*. Let $\vec{S} = \mathsf{SetSystem}(m, n, \mathsf{inp}) = (\{S_{i,b}\}_{i\in[m],b\in\{0,1\}}, S_t)$, and let C be any weakly $\vec{S}$-respecting arithmetic circuit whose output wire is tagged with $T \subseteq [k]$. Then there exists a set $U \subseteq \{0,1,*\}^m$ such that for every branching program BP of width w and length m on n input bits, with input tagging function* inp*, every prime p, and every $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m,b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$,*

(i)

$$C(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \equiv \sum_{u\in U} C_u(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$$

where each C_u is a weakly $\vec{S}$-respecting arithmetic circuit, whose input wires are tagged only with sets $\in \{S_{i,u[i]}\}_{i\in[m]:u[i]\neq *} \cup \{S_t\}$, and whose output wire is tagged with T.

(ii) *Each C_u above is the sum of several "monomial" circuits, where each monomial circuit performs only multiplications of elements in $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m, b\in\{0,1\}}, \mathbf{t})$, is weakly $\vec{S}$-respecting, and has output wire tagged with T.*

(iii) *For each C_u above,*

$$C_u(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_u \cdot p_u(\{\tilde{B}_{i,u[i]}\}_{i\in[m]:u[i]\neq *}, \mathbf{t})$$

where p_u is some polynomial, and $\alpha_u = (\prod_{i\in[m]:u[i]\neq *} \alpha_{i,u[i]})$. Furthermore, when p_u is viewed as a sum of monomials, each monomial contains exactly one entry from each $\tilde{B}_{i,u[i]}$ such that $u[i] \neq *$, and possibly one entry from $\mathbf{t}$. Further, p_u can be computed by a weakly $\vec{S}$-respecting circuit whose output wire is tagged with T.

The lemma can be proved using a simple induction. We provide a complete proof of the lemma in Section 3.8. Given this lemma, the proof of Proposition 1 is as follows:

Proof. **Part (i)** We consider the special case of Lemma 20 part (i), in which C is $\vec{S}$-respecting (as opposed to only weakly $\vec{S}$-respecting). In this case, we have that each C_u in the decomposition of C is also $\vec{S}$-respecting, and in particular, each C_u for $u \in U$ has its output wire tagged with the universe set $[k]$.

We first observe that for any C_u in the decomposition of C, u cannot contain $*$. This is because the output of C_u is tagged with $[k]$, and thus must have at least one input wire tagged with either of $S_{i,0}$ or $S_{i,1}$ for each i, or else the straddling set $\mathbb{S}^{\mathsf{inp}(i)}$ will be incomplete, and thus the output wire cannot be tagged with $[k]$.

Further, we observe that for every $u \in U$, for every $j \in [n]$, there must be a bit $b_j \in \{0,1\}$ such that for every $i \in [m]$ such that $\mathsf{inp}(i) = j$, $u[i] = b_j$. This can be seen by considering any monomial circuit in C_u individually. Recall from Lemma 20

part (ii) that C_u is formed by summing some number of monomials circuits, each of which is $\vec{S}$-respecting and has output wire tagged with $[k]$. This means that $\mathbb{S}^j \subseteq [k]$ is covered by the elements of the monomial. However, since $\mathbb{S}^j$ is constructed as a straddling set, the only way to cover $\mathbb{S}^j$ in a monomial circuit that only contains multiplication gates, is by using either all sets from $\{S_{i,0} : \mathsf{inp}(i) = j\}_{i \in m}$ or all sets from $\{S_{i,1} : \mathsf{inp}(i) = j\}_{i \in m}$. This means, correspondingly, that u must be such that there is a bit $b_j \in \{0,1\}$, for every $i \in [m]$ such that $\mathsf{inp}(i) = j$, $u[i] = b$. Define $x \in \{0,1\}^n$ so that $x[j] = b_j$ for all $j \in [n]$. In this way, we can define a one-to-one correspondence from each $u \in U$ to corresponding $x \in \{0,1\}^n$, and we simply relabel each C_u to the corresponding C_x to get the desired decomposition of C. We observe that the additional conditions on each C_x can be achieved from the corresponding conditions on C_u as guaranteed by Lemma 20.

Part (ii) Part (ii) follows directly from Part (i) of this proposition, together with Lemma 20 part (iii), and the observation that each C_u in Lemma 20 is relabelled to C_x for some $x \in \{0,1\}^n$ in Part (i) of this proposition. □

Now we are ready to describe the simulator CSim. CSim gets as input 1^m, prime p, a $\vec{S}$-respecting circuit C, vectors v_0, v_1 and has oracle access to a length m branching program BP. Let X be the set of inputs and $\{p_x\}_{x \in X}$ be the single-input polynomials corresponding to the decomposition of C. For every $x \in X$, CSim queries BP on x, samples $d_x \leftarrow \mathsf{KSim}(1^m, p, v_{BP(x)})$ and checks whether $p_x(d_x) = 0$. CSim outputs 1 if and only if for every input $x \in X$, $p_x(d_x) = 0$.

Now we prove correctness of our simulation. First, we prove some claims that will be useful. In each of these claims, let proj_x be defined with respect to the labeling function inp of the branching program BP. The following claim states

that if $C(\mathsf{Rand}(BP, p))$ is always zero, then every single-input term is always zero.

Claim 21. *If* $Pr[C(\mathsf{Rand}(BP, p) = 0] = 1$ *then for every input* $x \in X$*,*

$$Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] = 1$$

Proof. Consider a fixed $d = (\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$ in the support of $\mathsf{Rand}^B(BP, p)$ and let $C_d(\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}) = C(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$. By Proposition 1, we know that

$$C_d(\{\alpha_{i,b}\}) = \sum_{x\in X}(\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})p_x(\mathsf{proj}_x(d))$$

and C_d is a degree $m + 2$ polynomial. By assumption, $C(\mathsf{Rand}(BP, p))$ is always zero (over the support of $\mathsf{Rand}(BP, p)$); hence, $C_d(\{\alpha_{i,b}\}) = 0$ for all non-zero $\{\alpha_{i,b}\}$. By the Schwartz-Zippel lemma, this can happen only if C_d is the zero polynomial. By the structure of C_d, this implies that for every $x \in X$, $p_x(\mathsf{proj}_x(d)) = 0$. This argument works for every fixed value of d, hence we have that for every $x \in X$, $\Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] = 1$. □

The next claim states that if $C(\mathsf{Rand}(BP, p))$ is not always zero, then it is zero with small probability. Furthermore, there exists a single-input term that is zero with small probability.

Claim 22. *For any* $\vec{S}$*-respecting circuit* C*, if* $Pr[C(\mathsf{Rand}(BP, p)) = 0] < 1$ *then the following holds.*

1. $Pr[C(\mathsf{Rand}(BP, p)) = 0] \leq 16wm/p$
2. *There exists* $x \in X$ *such that* $Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] \leq 16wm/p$*, where* X *is obtained from the decomposition of* C *by Proposition 1.*

Proof. We start by showing part 1.

Part 1: If $\mathsf{Rand}(BP, p) = \mathsf{Rand}^{\alpha}(\mathsf{Rand}^{B}(BP, p))$ can be expressed as a low-degree ($\leq 2w$) polynomial on uniformly random values in $\mathbb{Z}_p$—namely, the α's and the randomization matrices R_i's—then by the Schwartz-Zippel lemma the first part of the claim directly follows. However, there are two barriers to applying this argument:

- Rand^{B} does not sample uniformly random matrices $\{R_i\}_{i\in[m]}$; rather, it chooses uniformly random *invertible* matrices R_i. Similarly, Rand^{α} does not sample uniformly random $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$; rather, it chooses uniformly random *non-zero* $\alpha_{i,b}$.
- Rand^{B} also needs to compute inverses R_i^{-1} to R_i for every $i \in [m]$ (which may no longer be expressed as low degree polynomials in the matrices $\{R_i\}_{i\in[m]}$).

To handle the second issue, consider the distribution $\mathsf{Rand}^{B}_{adj}(BP, p)$ that is defined exactly as $\mathsf{Rand}^{B}(BP, p)$ except that for every $i \in [m]$ it uses $adj(R_i) = R_i^{-1}det(R_i)$ instead of R_i^{-1}. Note that every entry of the adjoint of a $w \times w$ matrix M is some cofactor of M (obtained by the determinant of the $w-1 \times w-1$ matrix obtained by deleting some row and column of A). Hence every entry of $adj(R_i)$ can be expressed as a degree w polynomial in R_i. Let $\mathsf{Rand}_{adj}(BP, p) = \mathsf{Rand}^{\alpha}(\mathsf{Rand}^{B}_{adj}(BP, p))$. It follows that $\mathsf{Rand}_{adj}(BP, p)$ is computed by degree (at most) $2w$ polynomial in the matrices $\{R_i\}_{i\in[m]}$ and scalars $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$.

Furthermore, we show that $Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] = Pr[C(\mathsf{Rand}(BP, p)) = 0]$. Recall that by Proposition 1,

$$C \equiv \sum_{x\in X} C_x$$

and for each C_x above and every $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$,

$$C_x(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_x \cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$$

where $\alpha_x = (\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})$ and p_x is a polynomial such that, when viewed as a sum of monomials, each monomial contains exactly one entry from each $\tilde{B}_{i,x[\mathsf{inp}(i)]}$, and one entry from $\mathbf{t}$. Recall that for every $i \in [m]$,

$$\tilde{B}_{i,x[\mathsf{inp}(i)]} = R_{i-1} B_{i,x[\mathsf{inp}(i)]} R_i^{-1}$$

For every $i \in [m]$, replacing R_i^{-1} with $adj(R_i)$ has the effect of multiplying each monomial in p_x with the scalar $det(R_i)$. Hence

$$C_x(\mathsf{Rand}_{adj}(BP, p)) = (\prod_{i\in[m]} det(R_i)) \cdot C_x(\mathsf{Rand}(BP, p))$$

Since C is the sum of such C_x terms, it holds that $C(\mathsf{Rand}_{adj}(BP, p)) = (\prod_{i\in[m]} det(R_i))C(\mathsf{Rand}(BP, p))$. For every $i \in [m]$, by invertibility, $det(R_i) \neq 0$ and hence

$$Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] = Pr[C(\mathsf{Rand}(BP, p)) = 0]$$

So far, we have that $\mathsf{Rand}_{adj}(BP, p)$ is computed by a degree $2w$ polynomial in the matrices $\{R_i\}_{i\in[m]}$ and scalars $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$. However the first issue remains: each R_i is uniformly random invertible and each $\alpha_{i,b}$ is uniformly random non-zero, whereas we need them to be uniformly random. Consider the distribution $\mathsf{Rand}_{adj,U}(BP, p)$ that is obtained by the computing the same polynomial on uniformly random matrices $\{R_i\}_{i\in[m]}$ and scalars $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$ over $\mathbb{Z}_p$. In Claim 31, we show that the statistical distance between $\mathsf{Rand}_{adj}(BP, p)$ and $\mathsf{Rand}_{adj,U}(BP, p)$ is at most $8wm/p$. Furthermore, the support of $\mathsf{Rand}_{adj,U}(BP, p)$ contains the support of $\mathsf{Rand}_{adj}(BP, p)$. This implies that if $Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] < 1$ then $Pr[C(\mathsf{Rand}_{adj,U}(BP, p)) = 0] < 1$.

We now turn to proving the statement of the claim. Using facts shown above, we have that

$$Pr[C(\mathsf{Rand}(BP,p))=0]<1 \implies Pr[C(\mathsf{Rand}_{adj}(BP,p))=0]<1$$
$$\implies Pr[C(\mathsf{Rand}_{adj,U}(BP,p))=0]<1$$

By Proposition 1, C evaluates a $m+1$ degree polynomial, and $\mathsf{Rand}_{adj,U}(BP,p)$ is computed by a degree $2w$ polynomial in uniformly random values in $\mathbb{Z}_p$. By the Schwartz-Zippel lemma,

$$Pr[C(\mathsf{Rand}_{adj,U}(BP,p))=0]<1$$
$$\implies Pr[C(\mathsf{Rand}_{adj,U}(BP,p)=0]\leq 2w(m+1)/p\leq 8wm/p$$

We have that the statistical distance between $\mathsf{Rand}_{adj,U}(BP,p)$ and $\mathsf{Rand}_{adj}(BP,p)$ is at most $8wm/p$. Therefore, $Pr[C(\mathsf{Rand}(BP,p))=0]=Pr[C(\mathsf{Rand}_{adj}(BP,p))=0]\leq 16wm/p$ thus proving the first part of the claim. We proceed to show part 2.

Part 2: By Proposition 1, for every $x\in X$, there exists a $\vec{S}$-respecting arithmetic circuit C_x such that for every $(\{\alpha_{i,b}\cdot\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}},\mathbf{t})\leftarrow\mathsf{Rand}(BP,p)$,

$$C_x(\{\alpha_{i,b}\cdot\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}},\mathbf{t})=\alpha_x\cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]},\mathbf{t})$$

where $\alpha_x=(\prod_{i\in[m]}\alpha_{i,x[\mathsf{inp}(i)]})$ and $C=\sum_{x\in X}C_x$. In particular, $p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]},\mathbf{t})=0$ iff $C_x(\{\alpha_{i,b}\cdot\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}},\mathbf{t})=0$ (since $\alpha_{i,b}$ is non-zero).

Thus, we have that

$$Pr[C(\mathsf{Rand}(BP,p)))=0]=Pr[C_x(\mathsf{Rand}^{\alpha}(\mathsf{Rand}^{B}(BP,p)))=0]$$
$$=Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^{B}(BP,p)))=0]$$

There must exist an input $x\in X$ such that $Pr[C_x(\mathsf{Rand}(BP,p)))=0]<1$ or else $Pr[C(\mathsf{Rand}(BP,p)))=0]=1$. By the first part of the claim, it follows that

$$Pr[C(\mathsf{Rand}(BP,p)))=0]\leq 16wm/p,$$

which concludes the proof. □

Now we analyze the correctness of the simulator CSim. We consider the following two cases: when $C(\mathsf{Rand}(BP, p))$ is always zero, and otherwise.

Case 1: $Pr[C(\mathsf{Rand}(BP, p)) = 0] = 1$**:** In this case we will show that the simulation always succeeds. If $Pr[C(\mathsf{Rand}(BP, p)) = 0] = 1$ then by Claim 21, for every $x \in X$, $Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] = 1$. Recall that $\mathsf{KSim}(1^m, p, v_{BP(x)})$ simulates $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$ perfectly. Therefore, CSim always outputs 1 and hence succeeds.

Case 2: $Pr[C(\mathsf{Rand}(BP, p)) = 0] < 1$**:** In this case, by the first part of Claim 22 we have that

$$Pr[\mathsf{isZero}(C(\mathsf{Rand}(BP, p))) = 1] \leq 16wm/p$$

By the perfect simulation of KSim, we have that

$$Pr[\mathsf{CSim}^{BP} = 1] = Pr[\forall x\ (d_x \leftarrow \mathsf{proj}_x(\mathsf{Rand}^B(BP, p)) : p_x(d_x) = 0)]$$

By second part of Claim 22 there exists input x_C such that $Pr[p_{x_C}(\mathsf{proj}_{x_C}(\mathsf{Rand}^B(BP, p))) = 0] \leq 16wm/p$. Therefore,

$$Pr[\mathsf{CSim}^{BP} = 1] \leq Pr[p_{x_C}(\mathsf{proj}_{x_C}(\mathsf{Rand}^B(BP, p))) = 0] \leq 16wm/p$$

Therefore, by a union bound we have that

$$Pr[\mathsf{isZero}(C(\mathcal{D})) = \mathsf{CSim}^{BP} = 0] > 1 - 32wm/p$$

This concludes the proof of the lemma. □

□

Restricting to Entropic Message Samplers

We here show that the message samper M in the previous section satisfies the required high-entropy condition (required by the notion of entropic semantical security); that is, M is entropically valid.

Recall that the message sampler M in the proof of Theorem 18 gets as input the description of a ring $R = \mathbb{Z}_p$ and samples $(\vec{m}_0, \vec{m}_1, \vec{z})$ such that $(\vec{m}_0, \vec{z})$ and $(\vec{m}_1, \vec{z})$ are the "randomizations" (as defined in the description of Rand) of fixed branching programs. We now show the following proposition, which combined with the fact that the length m of the branching programs is polynomial in $\log |R|$ (recall that $R = \mathbb{Z}_p$ where p is a prime exponential in the multilinearity parameter k which is $< 3m$), implies that the output of a non-terminal set-respecting circuit on input $(\vec{m}_b, \vec{z})$ (for both $b \in \{0, 1\}$) has min-entropy $\log |R| - O(\log \log |R|)$, as required.

Proposition 2. *Let BP be a branching program of length m, width w, input length n and input labeling function* inp. *Let p be a prime and* $\vec{S} = \mathsf{SetSystem}(m, n, \mathsf{inp})$. *Let C be a non-terminal* $\vec{S}$*-respecting arithmetic circuit that computes a non-zero polynomial. Then we have that*

$$H_\infty(C(\mathsf{Rand}(BP, p))) \geq \log(\frac{p}{12wm})$$

or equivalently, for any fixed output $a \in \mathbb{Z}_p$

$$Pr[C(\mathsf{Rand}(BP, p)) = a] \leq 12wm/p$$

Proof. Let T be the set that tags the output wire of C as per the construction given in Definition 9. Since C is non-terminal $\vec{S}$-respecting, we have that T is a strict subset of $[k]$ where $(k, \vec{S}) = \mathsf{SetSystem}(m, n, \mathsf{inp})$. By Lemma 20 part (iii), there

exists a set U of labels $u \in \{0, 1, *\}$ such that for every $(\{\alpha_{j,b} \cdot \tilde{B}_{j,b}\}_{j\in[m],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$ we have that

$$C(\{\alpha_{j,b} \cdot \tilde{B}_{j,b}\}_{j\in[m],b\in\{0,1\}}, \mathbf{t}) = \sum_{u\in U} \alpha_u \cdot p_u(\{\tilde{B}_{j,u[j]}\}_{j\in[m]:u[j]\neq *}, \mathbf{t}) \tag{3.4}$$

where $\alpha_u = \prod_{j\in[m]:u[j]\neq *} \alpha_{j,u[j]}$. Furthermore, each p_u is computed by a weakly $\vec{S}$-respecting circuit whose output wire is also tagged with T. Since C computes a non-zero polynomial, there must exist $v \in U$ such that p_v is a non-zero polynomial. We now have the following claim.

Claim 23. *$Pr[p_v(\{\tilde{B}_{j,v[j]}\}_{j\in[m]:v[j]\neq *}, \mathbf{t}) = 0] \leq 10wm/p$.*

Proof. To see this, we first observe that since T is a strict subset of $[k]$ and p_v is computed by a $\vec{S}$-respecting circuit whose output wire is tagged with T, either p_v does not operate on some level of the branching program or it does not operate on $\mathbf{t}$; that is, either,

- there exists $j \in [m]$ such that $v[j] = *$, or
- p_v is not a function of $\mathbf{t}$.

In the first case, by an argument similar to that in Claim 16, we can show that the distribution $(\{\tilde{B}_{j,v[j]}\}_{j\in[m]:v[j]\neq *}, \mathbf{t})$ is identical to the distribution $(\{R_j\}_{j\in[m]:v[j]\neq *}, \mathsf{col}_1(R_{m+1}))$ where $\{R_j\}_{j=1}^{m+1}$ are random invertible matrices over $\mathbb{Z}_p^{w\times w}$. By Claim 31, this distribution is statistically $8wm/p$-close to the distribution where each matrix entry is uniformly random in $\mathbb{Z}_p$. Furthermore, since p_v is computed by a $\vec{S}$-respecting circuit, it is of degree at most $m + 1 < 2wm$. By the Schwartz Zippel lemma, the evaluation of p_v on such random inputs from $\mathbb{Z}_p$ is zero with probability at most $2wm/p$. All in all, we have $Pr[p_v(\{\tilde{B}_{j,v[j]}\}_{j\in[m]:v[j]\neq *}, \mathbf{t}) = 0] \leq 10wm/p$.

In the second case, p_v acts on the $\{\tilde{B}_{j,v[j]}\}_{j\in[m]}$. Following Claim 16, this distribution is identical to that of m random invertible matrices over $\mathbb{Z}_p^{w\times w}$. Similarly to the first case, it follows that $Pr[p_v(\{\tilde{B}_{j,v[j]}\}_{j\in[m]}) = 0] \leq 10wm/p$. □

Let E be the event that $p_v(\{\tilde{B}_{j,v[j]}\}_{j\in[m]:v[j]\neq *}, \mathbf{t}) \neq 0$. For any fixed output $a \in \mathbb{Z}_p$ we have that

$$Pr[C(\mathsf{Rand}(BP, p)) = a] \leq Pr[C(\mathsf{Rand}(BP, p)) = a|E] + Pr[\bar{E}] \tag{3.5}$$

For a fixed $\{\tilde{B}_{j,b}\}_{j\in[m],b\in\{0,1\}}$ let $q_{(\tilde{B},a)}$ be a polynomial in variables $\{\alpha_{j,b}\}_{j\in[m],b\in\{0,1\}}$ such that

$$q_{(\tilde{B},a)}(\{\alpha_{j,b}\}_{j\in[m],b\in\{0,1\}}) = C(\{\alpha_{j,b}\cdot\tilde{B}_{j,b}\}_{j\in[m],b\in\{0,1\}}) - a$$

When the event E occurs, we claim that the resulting polynomial $q_{(\tilde{B},a)}$ is a non-zero polynomial of degree at most m. This can be easily seen given the decomposition of C in (3.4). When $q_{(\tilde{B},a)}$ is a non-zero polynomial then by the Schwartz Zippel lemma, its evaluation on uniformly random non-zero inputs $\{\alpha_{j,b}\}_{j\in[m],b\in\{0,1\}}$ is zero with probability at most $m/p - 1 \leq 2wm/p$. Therefore, we have

$$Pr[C(\mathsf{Rand}(BP, p)) = a|E] = Pr[q_{\tilde{B}}(\{\alpha_{j,b}\}) = 0|E] \leq \frac{2wm}{p} \tag{3.6}$$

Combining (3.6) and (3.5) and Claim 23, we have $Pr[C(\mathsf{Rand}(BP, p)) = a] \leq 12wm/p$. □

3.4.4 Achieving Obfuscation for Arbitrary Programs

[70] show that any indistinguishability obfuscation scheme for NC^1 can be bootstrapped into an indistinguishability obfuscation scheme for all poly-sized cir-

cuits using FHE. That is, they prove the following theorem.

Theorem 24 ([70])**.** *Assume the existence of indistinguishability obfuscators $i\mathcal{O}$ for* NC^1 *and a leveled Fully Homomorphic Encryption scheme with decryption in* NC^1*. Then there exists an indistinguishability obfuscator $i\mathcal{O}'$ for* $\mathsf{P/poly}$*.*

Applying their construction to our indisinguishability obfuscator yields an indistinguishability obfuscator for arbitrary polynomial size circuits:

Theorem 25. *Assume the existence of a entropic semantically secure multilinear encoding scheme and a leveled Fully Homomorphic Encryption scheme with decryption in* NC^1*. Then there exists indistinguishability obfuscators for* $\mathsf{P/poly}$*.*

3.5 $i\mathcal{O}$ from Single-Distribution Semantical Security

The assumption that a scheme satisfies semantical security w.r.t. some class of message samplers may perhaps be best viewed as a *class of assumptions* (or a "meta-assumption", just like the "uber assumption" of [35]), or alternatively as an *interactive assumption*, where the attacker first selects the sets $\vec{S}, \vec{T}$ and the message sampler M, and then gets a challenge according to the message sampler.

This view point also clarifies that even for the above-mentioned restricted classes of message distributions, semantical security is not an *efficiently falsifiable* assumption [123]: the problem is that there may not exist an efficient way of checking whether a message sampler is valid (which requires checking that all set-respecting circuits are constant with overwhelming probability).

We here show that a single, falsifiable, instance of this class of assumptions suffices for proving security of indistindinguishability obfuscator, albeit at the cost of subexponential hardness.

3.5.1 Single-Distribution Semantical Security

Let us start by formalizing a "single-distribution" version of semantical security, where we restrict semantical security to hold w.r.t. to a single *efficiently samplable* distribution over pairs of message samplers M, and sets $\vec{S}, \vec{T}$. We call this distribution over message samplers and sets an *instance sampler*. Analogously to the notion of a valid message sampler, we now define a notion of a *valid instance sampler* as follows:

Definition 20. *We say that a PPT* Sam *is a* (c, q)-(entropically) valid instance sampler *if*

- *There exist a polynomial* $k(\cdot)$*, such that for every* $n \in \mathbb{N}$*, for every* $r_n \in \{0,1\}^\infty$ *,* $\mathsf{Sam}(1^n, r_n)$ *outputs a tuple* $(\vec{S}_n, \vec{T}_n, M_n)$*, where* $\vec{S}_n, \vec{T}_n$ *are sequences of sets over* $[k(n)]$ *with* $|\vec{S}_n| = c(k(n))$ *and* $|\vec{T}_n| = q(k(n))$*.*
- *For every sequence of random tapes* $\{r_n\}_{n\in\mathcal{N}}$*,* $\{M_n\}_{n\in\mathcal{N}}$ *is (entropically)* $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$*-respecting, where for every* $n \in \mathcal{N}$*,* $(\vec{S}_n, \vec{T}_n, M_n) \leftarrow \mathsf{Sam}(1^n; r_n)$*.*

Definition 21 (Single-distribution Semantic Security)**.** *Let* $\mathcal{E}$ *be a graded encoding scheme and* Sam *be a* (c, q)*-valid instance sampler. We say that* $\mathcal{E}$ *is* semantically secure w.r.t. Sam *if for every nuPPT adversary* A*, there exists a negligible function* ϵ *such that for every security parameter* $n \in \mathbb{N}$*,*

$$|Pr[\boldsymbol{Output'}_0(1^n) = 1] - Pr[\boldsymbol{Output'}_1(1^n) = 1]| \leq \epsilon(n)$$

where $\textbf{Output'}_b(1^n)$ *is A's output in the following game:*

- *Let* $\vec{S}_n, \vec{T}_n, M_n \leftarrow \mathsf{Sam}(1^n)$.
- *Let* k_n *be such that* $\vec{S}_n$ *and* $\vec{T}_n$ *are sequences of sets over* $[k_n]$. *Let* $(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k_n})$.
- *Let* $\vec{m_0}, \vec{m_1}, \vec{z} \leftarrow M_n(1^n, \mathsf{pp})$.
- *Let* $\vec{u_b} \leftarrow \{\mathsf{Enc}(\mathsf{sp}, \vec{m_0}[i], \vec{S}_n[i])\}_{i=1}^{c(n)}, \{\mathsf{Enc}(\mathsf{sp}, \vec{z}[i], \vec{T}_n[i])\}_{i=1}^{q(n)}$.
- *Finally, run* $A(1^n, \mathsf{pp}, (\vec{S}_n, \vec{T}_n), M_n, \vec{u_b})$.

Note that given an $(O(1), O(k))$-valid instance sampler Sam, the assumption that $\mathcal{E}$ is semantically-secure w.r.t. Sam is a special case of the assumption that $\mathcal{E}$ is (constant-message) semantically secure; if $\mathcal{E}$ is not semantically secure w.r.t. Sam, there exists ensembles $\{r_n\}_{n\in\mathcal{N}}$, $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$ and $\{M_n\}_{n\in\mathcal{N}}$ such that $\vec{S}_n, \vec{T}_n, M_n = \mathsf{Sam}(1^n; r_n)$ (and thus $\{M_n\}_{n\in\mathcal{N}}$ is a valid message sampler for $\{\vec{S}_n\vec{T}_n\}_{n\in\mathcal{N}}$, yet the nuPPT $A(1^n, \cdot, \vec{S}_n, \vec{T}_n, M_n, \cdot)$ breaks semantical security when considering $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$ and $\{M_n\}_{n\in\mathcal{N}}$.

Furthermore, that given an $(O(1), O(k))$-(entropically) valid instance sampler Sam, the assumption that $\mathcal{E}$ is semantically-secure w.r.t. Sam is a non-interactive and efficiently falsifiable (decisional) assumption—in essence, it is a specific instance of a DDH-type assumption over multilinear encodings.

3.5.2 Basing Security on Single-Distribution Semantical Security

We now show how to slightly modify the construction $i\mathcal{O}$ from Section 3.4.3 so that we can base it on single-distribution semantical security assumption. This time, however, we require subexponentially-hard semantical security (and as such the assumption is incomparable to the one needed for the scheme from Section 3.4.3.)

Towards this, we introduce a new notion of *neighboring-input indistinguishability obfuscation*. As we shall see, the assumption that a scheme satisfies neighboring-input $i\mathcal{O}$ is already an efficiently falsifiable assumption. We then show that a) *exponentially-secure* neighboring-input $i\mathcal{O}$ implies "full" $i\mathcal{O}$, and b) exponentially-secure neighboring-input $i\mathcal{O}$ can be based on subexponentially-hard single-distribution semantic security. (We mention a very recent work by Gentry, Lewko and Waters [86] in the context of *witness encryption* [75] that similarly defines a falsifiable primitive "positional witness encryption" that implies the full-fledged notion with an exponential security loss.)

Neighboring-input Indistinguishability Obfuscation

We start by recall a different "merge" procedure from the work of Boyle, Chung and Pass [42]: Given two NC^1 circuits C_0, C_1 taking (at most) n-bit inputs, and a string z, let $\widehat{\mathsf{Merge}}(C_0, C_1, z)$ be a circuit that on input x runs $C_0(x)$ if $x \geq z$ and $C_1(x)$ otherwise. ([42] use this type of merged circuits to perform a binary search and prove that indistinguishability obfuscation implies differing-input obfuscation for circuits that differ in only polynomially many inputs.) Also,

$\widehat{\mathsf{Merge}}$ is defined such that $\widehat{\mathsf{Merge}}(C_0, C_1, 0) = C_0$ and $\widehat{\mathsf{Merge}}(C_0, C_1, 2^n) = C_1$. It is easy to see that an NC^1 circuit computing $\widehat{\mathsf{Merge}}(C_0, C_1, z)$ can be efficiently found given NC^1 circuits C_0, C_1 and z; (abusing notation) let $\widehat{\mathsf{Merge}}$ denote an efficient procedure that outputs such a circuit.

The notion of neighboring-input $i\mathcal{O}$ relaxes $i\mathcal{O}$ by only requiring that security holds with respect to "neigboring-input" programs $\widehat{\mathsf{Merge}}(C_0, C_1, z)$, $\widehat{\mathsf{Merge}}(C_0, C_1, z+1)$ that are functionally equivalent. Note that checking whether $\widehat{\mathsf{Merge}}(C_0, C_1, z)$, $\widehat{\mathsf{Merge}}(C_0, C_1, z+1)$ are functionally equivalent is easy: they are equivalent iff $C_0(z) = C_1(z)$. As such, the assumption that a scheme satisfies neighboring-input $i\mathcal{O}$ is efficiently falsfiable.

Definition 22. *A uniform PPT machine $i\mathcal{O}$ is a* neighboring-input indistinguishability obfuscator *for the class of circuits $\{C_n\}_{n\in\mathcal{N}}$ if it satisfies the same correctness condition as in Definition 1 but the security condition is replaced by:*

- **Security:** *For every nuPPT adversary A there exists a negligible function ε such that for all $n \in \mathbb{N}$, all $C_0, C_1 \in C_n^1$ and all $z \in \{0,1\}$ such that $C_0(z) = C_1(z)$,*

$$|\Pr[A(1^n, C_0', C_1', z, i\mathcal{O}(1^n, C_0') = 1] - \Pr[A(1^n, C_0', C_1', z, i\mathcal{O}(1^n, C_1') = 1]| \leq \epsilon(n)$$

where $C_b' = \widehat{\mathsf{Merge}}(C_0, C_1, z+b)$.

We additionally say that $i\mathcal{O}$ is exponentially-secure *if for every nuPPT A the above indistinguishability gap is bounded by $\varepsilon(n) = 2^{-O(n^2)}$.*

Theorem 26. *There exists an $(O(1), O(k))$-entropically valid instance sampler* Sam*, such that if there exists an encoding scheme that is subexponentially-hard semantically secure w.r.t.* Sam*, then there exists an exponentially-secure neighboring-input indistinguishability obfuscator for C^1.*

Proof. Consider the obfuscator $i\mathcal{O}(\cdot,\cdot,\cdot)$ for NC^1 presented in Section 3.4.3. We change it to run the underlying multilinear encoding scheme with security parameter $n' = n^{2/\alpha}$, where α is the subexponential security constant for the encoding scheme. Let c^* be the constant such that the sizes and depth of $\widehat{\mathsf{Merge}}(C_0, C_1, z)$ where $C_0, C_1 \in C_n^1$ and $z \in \{0,1\}^n$ are bounded by n^{c^*} and $c^* \log(n)$ respectively. We show that $i\mathcal{O}(c^*,\cdot,\cdot) = i\mathcal{O}(\cdot,\cdot) =$ is an exponentially-secure indistinguishability obfuscator for C^1 based on subexponentially-hard semantical security with respect to an instance sampler Sam.

Assume for contradiction there exists nuPPT A such that for infinitely many n, there exist $C_0, C_1 \in C_n^1$, $z \in \{0,1\}$ such that $C_0(z) = C_1(z)$ and A given $(1^n, C_0', C_1', z)$ where $C_0' = \widehat{\mathsf{Merge}}(C_0, C_1, z)$ and $C_1' = \widehat{\mathsf{Merge}}(C_0, C_1, z+1)$, distinguishes $i\mathcal{O}(1^n, C_0')$ and $i\mathcal{O}(1^n, C_1')$, with probability, say, 2^{-n^2}.

We define hybrid distributions similarly as in the proof in Section 3.4.2 corresponding to $i\mathcal{O}(1^n, C_0')$ and $i\mathcal{O}(1^n, C_1')$. Recall that each of these hybrids correspond to one step in the transition from a branching program for C_0' to a branching program C_1', where each step changes at most two levels of the branching program. Let $h(n)$ be the number of such hybrids. We have that the circuits C_0' and C_1' determine for every $j \in [h(n) - 1]$ a hybrid distribution H_j such that H_0 is identical to $i\mathcal{O}(1^n, C_0')$, $H_{h(n)}$ is identical to $i\mathcal{O}(1^n, C_1')$ and for every $j \in [h(n) - 1]$, indistinguishability of H_j and H_{j+1} follows from neighboring-matrix indistinguishability obfuscation which in turn follows from a reduction to semantic security.

We now define $\mathsf{Sam}(1^{n'}; r_{n'})$ as follows: Using random coins $r_{n'}$, Sam uniformly samples $C_0, C_1 \leftarrow C_n^1$, $z \leftarrow \{0,1\}^n$ and a random hybrid index $j \in [h(n) - 1]$. It checks whether $C_0(z) = C_1(z)$ and if not, it sets $C_1 = C_0$. Next, it generates

$C'_0 = \widehat{\mathsf{Merge}}(C_0, C_1, z)$ and $C'_1 = \widehat{\mathsf{Merge}}(C_0, C_1, z+1)$. Finally, it outputs the sets $(\vec{S}_{n'}, \vec{T}_{n'})$ and message sampler $M_{n'}$ used in the reduction to semantic security when arguing indistinguishability of hybrids H_j and H_{j+1}, as determined by the circuits C'_0 and C'_1.

Note that since the pair of the circuits C'_0, C'_1 sampled by Sam are always functionally equivalent, by the same proof as in Section 3.4.3 (more specifically, Lemma 19), we have that the messages $\vec{m}_0, \vec{m}_1, \vec{z}$ output by $M_{n'}$ are such that every $(\vec{S}_{n'}, \vec{T}_{n'})$-respecting circuit is constant on both $\vec{m}_0, \vec{z}$ and $\vec{m}_1, \vec{z}$, except with probability at most $Q(n', k)/|R|$ for some fixed polynomial $Q(\cdot, \cdot)$. Thus, for every sequence of random tapes $\{r_n\}_{n\in\mathcal{N}}$, $\{M_n\}_{n\in\mathcal{N}}$ is $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting, where for every $n \in \mathcal{N}$, $\vec{S}_n, \vec{T}_n, M_n = \mathsf{Sam}(1^n; r_n)$. We conclude that Sam is a $(O(1), O(k))$-valid instance sampler.

By assumption, there exists a $j \in [h(n) - 1]$ such that A distinguishes H_j and H_{j+1} with advantage $2^{-n^2}/h(n)$. We now define a nuPPT attacker A' for semantical security w.r.t. Sam: For each n', A' receives as non-uniform advice the index j^* and proceeds as follows: $A'(1^{n'}, \mathsf{pp}, , (\vec{S}_{n'}, \vec{T}_{n'}), M_{n'}, \vec{u_b})$ examines $M_{n'}$ and extracts the underlying circuits C^*_0, C^*_1 the underlying merge index z^* and the underlying hybrid index j^* from it. (We assume $M_{n'}$ is defined so that this information is efficiently extractable.) If $j = j^*$, $C^*_0 = C'_0, C^*_1 = C'_1$ and $z^* = z$, A' executes $A(1^n, C^*_0, C^*_1, z^*, (\mathsf{pp}, \vec{u_b}))$, and otherwise simply outputs 1.

Let us now analyze the success probability of A':

- Conditioned on the event when $j = j^*$, $C^*_0 = C'_0, C^*_1 = C'_1$ and $z^* = z$, A' distinguishes with advantage $2^{-n^2}/h(n)$.
- Otherwise A''s output is 1.

Since C_0^*, C_1^*, z^*, j^* are chosen at random, it follows that A' has a total distinguishing advantage of at least $2^{-3n} \cdot 2^{-n^2}/h(n)^2 = 2^{-O(n^2)} = 2^{-O(n'^{\alpha})}$, which contradicts the assumption that the encoding scheme is subexponentially secure with respect to Sam. □

From *ni-iO* to *iO*

Theorem 27. *If there exists PPT iO that is an exponentially-secure neighboring-input indistinguishability obfuscator for $\mathcal{C}^1$, then there exists a PPT iO′ that is a subexponentially-secure indistinguishability obfuscator for* NC^1.

Proof. Assume the existence of a PPT *iO* that is an exponentially-secure neighboring-input indistinguishability obfuscator for the class $\mathcal{C}^1$. We show that *iO* is a (subexponentially-secure) indistinguishability obfuscator for $\mathcal{C}^1$; by Lemma 6, this suffices for concluding the existence of (subexponentially-secure) indistinguishability obfuscators for NC^1.

Assume there exists some nuPPT A such that for infinitely many n, there exists a pair of functionally equivalent circuits $C_n^0, C_n^1 \in \mathcal{C}_n^1$ such that A distinguishes $iO(1^n, C_n^0)$ and $iO(1^n, C_n^1)$ with probability, say, 2^{-n}. For any such n, consider a sequence of $2^n + 1$ hybrid distributions, where

- $H_0 = iO(1^n, C_n^0) = iO(1^n, \widehat{\mathsf{Merge}}(C_n^0, C_n^1, 0))$
- $H_i = iO(1^n, \widehat{\mathsf{Merge}}(C_n^0, C_n^1, i))$ for $i \in [1, \ldots, 2^n - 1]$
- $H_{2^n} = iO(1^n, C_n^1)) = iO(1^n, \widehat{\mathsf{Merge}}(C_n^0, C_n^1, 2^n))$

There must exist some z such that A distinguishes H_z and H_{z+1} with advantage at least $2^{-n} \cdot 2^{-n} = 2^{-2n}$. Thus, there exists some sequence of programs $\{C_n^0, C_n^1\}_{n \in \mathcal{N}}$

where $C_n^0, C_n^1 \in C_n^1$ and a sequence of of inputs $\{z_n\}_{n\in\mathcal{N}}$, $z_n \in [0,\ldots,2^n-1]$, such that for infinitely many n, A distinguishes $i\mathcal{O}$ $(1^n, \widehat{\mathsf{Merge}}(C_n^0, C_n^1, z_n))$ and $i\mathcal{O}$ $(1^n, \widehat{\mathsf{Merge}}(C_n^0, C_n^1, z_n+1))$ with advantage 2^{-2n}. This directly contradicts the exponential security of the neighboring-input indistinguishability obfuscator $i\mathcal{O}$. □

Combing the above theorems, we get the following corollary.

Theorem 28. *There exists an* $(O(1), O(k))$*-entropically valid instance sampler* Sam*, such that if there exists an encoding scheme that is subexponentially-hard semantically secure w.r.t.* Sam*, then there exists a subexponentially-secure indistinguishability obfuscator for* NC^1*.*

3.6 Alternative Security Notions of Semantical Security Encodings

In this section we consider alternative ways of defining security of multilinear encodings. First, in section 3.6.1 we show that semantical security holds (in a very strong sense) w.r.t. generic attackers. Next, in section 3.6.2 we consider various "uber assumptions" (similar to the uber-assumption of [35] in the context of bilinear maps)[28] which capture the intuition that "if an algebraic decisional assumption holds w.r.t. to generic attacks, then it also holds with respect to nuPPT attackers". As we shall see the perhaps most natural formalization of this notion is *false* (under standard cryptographic assumptions)—in particular, we give a concrete example of a algebraic decisional assumption that holds in

[28] We thank Shai Halevi for pointing out the connection with [35].

the generic model but is false w.r.t. nuPPT attackers. We finally consider alternative ways for formalizing such an uber assumption.

3.6.1 Semantical Security w.r.t. Algebraic Attackers

We begin by showing that semantic security holds in the generic model. We formally define an *algebraic adversary* or *generic adversary* by considering adversaries that interact with the following oracle.

Definition 23 (Oracle $\mathcal{M}$)**.** *Let $\mathcal{M}$ be an oracle which operates as follows:*

- *$\mathcal{M}$ gets as initial input a ring R, $k \in \mathbb{N}$ and list L of m pairs $\{(\alpha_i, S_i)\}_{i=1}^m$, $\alpha \in R$ and $S \subseteq [k]$.*
- *Every oracle query to $\mathcal{M}$ is an arithmetic circuit $C : R^m \to R$. When queried with C, $\mathcal{M}$ checks whether C is a $\vec{S}$-respecting arithmetic circuit where $\vec{S} = \{S_i\}_{i=1}^m$. If not, $\mathcal{M}$ outputs $\perp$. Otherwise, $\mathcal{M}$ computes C on $\{\alpha_i\}_{i=1}^m$ and outputs 1 if and only if the output of C is zero, and outputs 0 otherwise.*

To formalize that (even subexponentially-hard) semantical security holds w.r.t. generic attackers, we define a stronger notion of a set-respecting message samplers—which requires not only that the output of every set-respecting circuit is constant with overwhelming probability, but also that this holds for the output of any *unbounded* algebraic attacker that is restricted to *polynomially-many* zero-test queries— and show that this notion in fact already is implied by the standard one. This shows that semantical security holds in a very strong sense w.r.t. to generic attackers.

Definition 24 (Strongly Respecting Message Sampler)**.** *We say that a nuPPT M is a* strongly $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-respecting message sampler *(or* strongly valid w.r.t. $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$*) if it satisfies the same conditions as in Definition 11 but where the second bullet is replaced by the following:*

- *For every polynomial p, there exists some polynomial Q such that for every $n \in \mathbb{N}$, every $(\mathsf{sp}, \mathsf{pp})$ in the support of $\mathsf{InstGen}(1^n, 1^{k_n})$, every (deterministic) oracle algorithm A that on input 1^n makes at most $p(n)$ oracle queries, there exists some string $\alpha \in \{0,1\}^*$ such that*

$$Pr[(\vec{m_0}, \vec{m_1}, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp}, \vec{p_0})}(1^n) = A^{\mathcal{M}(\mathsf{pp}, \vec{p_1})}(1^n) = \alpha]$$
$$\geq 1 - Q(n, k_n)/|R|.$$

  where $\vec{p_b} = \{(m_b[i], S_i)\}_{i=1}^{c(n)}, \{(z[i], T_i)\}_{i=1}^{q(n)}$ and $c(n)$ and $q(n)$ are the lengths of $\vec{S}_n$ and $\vec{T}_n$ respectively.

Note that validity is the special case of strong validity where we restrict to the case when $p(n) = 1$.

Theorem 29. *A message sampler M is* strongly $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-respecting *if and only it is $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-respecting.*

Proof. The "only if" direction is trivial (as mentioned, if $p(n) = 1$ strong validity collapses down to validity). To prove the "if direction", consider some M, $p(\cdot)$, security parameter $n \in \mathbb{N}$, $(\mathsf{sp}, \mathsf{pp}) \in \mathsf{InstGen}(1^n, 1^{k(n)})$ where pp defines a ring R, and oracle machine A (the algebraic adversary) such that $A(1^n)$ makes at most $p(n)$ oracle queries. From semantic security of $\mathcal{E}$, we have that there exists some polynomial $Q(\cdot, \cdot)$ such that for every $(\vec{S}, \vec{T})$-respecting arithmetic circuit C, there

exists a constant $c_C \in \{0,1\}$ such that for every $b \in \{0,1\}$,

$$Pr[(\vec{m}_0, \vec{m}_1, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : \mathsf{isZero}(C(\vec{m}_b, \vec{z})) \neq c] \leq Q(n, k(n))/|R|$$

For $b \in \{0,1\}$, consider an execution of both $A^{\mathcal{M}(\mathsf{pp}, \vec{p}_b)}(1^n)$ where $\vec{m}_0, \vec{m}_1, \vec{z}$ are sampled by M. Note that except with probability $Q(n, k(n))/|R|$ it holds the first oracle query C_1 by A is answered as c_{C_1}. Analogously, if the first i queries $C_1, \ldots, C_i$ were answered as $c_{C_1}, \ldots c_{C_i}$, then except with probability $Q(n, k(n))/|R|$, the $(i+1)$th query C_{i+1} will be answered as $c_{C_{i+1}}$. It follows that except with probability $p(n)Q(n, k(n))/|R|$ over $\vec{m}_0, \vec{m}_1, \vec{z}$, the output of A is identical to the output of an execution of A where *every* oracle query C is answered by the bit c_C. Thus, for every algebraic attacker A there exists some string α—namely the output of A where every oracle query C is answered by c_C—such that for $b \in \{0,1\}$, except with probability $p(n)Q(n, k(n))/|R|$, the output of $A^{\mathcal{M}(\mathsf{pp}, \vec{p}_b)}(1^n)$ is α. □

Note that for the above proof to go through it is cruicial that we restrict the algebraic attacker to making polynomially-many (or subexponentially-many) oracle queries. This is not just an anomaly of the proof: if we allow the attacker to make an unbounded number of queries, then strong validity would no longer imply validity; we discuss this point further in Section 3.6.2.

3.6.2 Uber Assumptions for Multilinear Encodings

A natural question is whether there are reasonable qualitative strengthenings of semantical security that can be used to achieve stronger notions of obfuscation, such as differing-input (a.k.a. extractability) obfuscation. We here consider such a strengthening.

At first sight, it may seem like the most natural way of defining security of multilinear encodings would be to require that for specific classes of problems, generic attacks cannot be beaten (this is the approach alluded to in [16]). A natural "uber assumption" (similar to the uber-assumption of [35] in the context of bilinear maps) would be to require that "if an algebraic decisional assumption holds w.r.t. to generic attacks, then it also holds with respect to nuPPT attackers". Let us now formalize this notion.

Extractable Uber Security

We start by defining a notion of a *computationally valid* message sampler: roughly speaking, we want to capture the intuition that no generic attacker can distinguish $\vec{m}_0, \vec{z}$ from $\vec{m}_1, \vec{z}$. To get a definition that is a strong as possible, we require indistinguishability to hold in a *pointwise* sense: with overwhelming probability, the output of $A^{\mathcal{M}(\mathsf{pp},\vec{p_0})}(1^n, \mathsf{pp})$ is required to be the same as the output of $A^{\mathcal{M}(\mathsf{pp},\vec{p_1})}(1^n, \mathsf{pp})$.

Definition 25 (Computationally Respecting Message Sampler)**.** *We say that a nuPPT M is a* computationally $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-respecting message sampler *(or* computationally valid w.r.t. $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$*) if it satisfies the same conditions as in Definition 11 but where the second bullet is replaced by the following:*

- *For every nuPPT oracle machine A, there exists some negligible function ε such that for every $n \in \mathbb{N}$,*

$$Pr[(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k_n}), (\vec{m_0}, \vec{m_1}, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp},\vec{p_0})}(1^n, \mathsf{pp}) \neq A^{\mathcal{M}(\mathsf{pp},\vec{p_1})}(1^n, \mathsf{pp})] \leq \varepsilon(n)$$

where $\vec{p_b} = \{(m_b[i], S_i)\}_{i=1}^{c(n)}, \{(z[i], T_i)\}_{i=1}^{q(n)}$ *and* $c(n)$ *and* $q(n)$ *are the lengths of* $\vec{S}_n$ *and* $\vec{T}_n$ *respectively.*

Note that computational validity differs from strong validity (which is equivalent to " plain" validity) in two main aspects: 1) we no longer require the output of the algebraic attacker to be *constant* with overwhelming probability; rather, we only require that it cannot tell apart $\vec{m}_0$ and $\vec{m}_1$, and 2) the algebraic attacker is restricted to be nuPPT (as opposed to being unbounded and only making polynomially many queries).

We now define *extractable "uber security"* in exactly the same way as semantic security except that we only require the message sampler to be computationally valid (and define entropic uber security in the analogous way). In other words, extractable uber security implies that whenever $\vec{m}_0, \vec{z}$ and $\vec{m}_1, \vec{z}$ are *pointwise computationally indistinguishable* w.r.t. legal algebraic attackers, encodings of them computationally indistinguishable. (We use the term "extractable" since this notion of security requires that if encodings can be distinguished, then we can efficiently find (or "extract") set-respecting circuits that distinguish the elements.)

We now have the following theorem.

Theorem 30. *Assume the existence of a leveled Fully Homomorphic Encryption scheme with decryption in* NC^1*. Then no graded encoding scheme satisfies entropic extractable uber security.*

Proof. Consider any graded encoding scheme $\mathcal{E}$. To show that $\mathcal{E}$ is not entropic extractable uber secure we need to show that there exists an entropic computa-

tionally respecting message sampler M and PPT adversary A such that A distinguishes between encodings of $(\vec{m}_0, \vec{z})$ and $(\vec{m}_1, \vec{z})$ where $(\vec{m}_0, \vec{m}_1, \vec{z}) \leftarrow M$.

Our M will sample obfuscations of the following circuit family, that was shown to be unobfuscatable in the virtual black box setting [17].Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a semantically secure fully homomorphic encryption scheme with ciphertext size $N(\cdot)$; for simplicity of exposition, let us first assume that it is an "unleveled" FHE. For each security parameter n, consider the class of circuits

$$C_n = \{C_{n,a,b,v,\mathsf{pk},\mathsf{sk},\hat{a}}\}_{a,b\in\{0,1\}^n, v\in\{0,1\}, (\mathsf{pk},\mathsf{sk})\in\mathsf{Gen}(1^n), \hat{a}\in\mathsf{Enc}(pk,a)}$$

taking $N(n)$-bit inputs, where

$$C_{n,a,b,v,\mathsf{pk},\mathsf{sk},\hat{a}}(x) = \begin{cases} (\mathsf{pk}, \hat{a}) & \text{if } x = 0 \\ b & \text{if } x = a \\ v & \text{if } \mathsf{Dec}(\mathsf{sk}, x) = b \\ 0 & \text{otherwise} \end{cases}$$

Then $M(1^n, \mathsf{pp})$ operates as follows, given public parameters pp to a graded encoding scheme it first computes the ring $R = \mathbb{Z}_p$ associated with pp.

- M samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ and $a, b \leftarrow \{0,1\}^n$ uniformly at random, and computes $\hat{a} = \mathsf{Enc}(\mathsf{pk}, a)$.
- M generates branching programs BP_0 and BP_1 corresponding to $C_{n,a,b,0,\mathsf{pk},\mathsf{sk},\hat{a}}$ and $C_{n,a,b,1,\mathsf{pk},\mathsf{sk},\hat{a}}$ respectively, and computes $\widehat{BP}_0 = \mathsf{Merge}(BP_0, BP_1, 0)$ and $\widehat{BP}_1 = \mathsf{Merge}(BP_0, BP_1, 1)$, each of width 10 and length m. Recall, from Claim 11, that $\widehat{BP}_0$ and $\widehat{BP}_1$ differ only in levels 1 and m, and that $\widehat{BP}_0$ and $\widehat{BP}_1$ are functionally equivalent to BP_0 and BP_1 respectively.

- M samples m random invertible matrices over $\mathbb{Z}_p^{10\times 10}$, $\{R_i\}_{i\in[m]}$ and $2m$ random scalars from $\mathbb{Z}_p$, $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$. M then uses these matrices and scalars to randomize $\widehat{BP}_0$ and $\widehat{BP}_1$ as described by $\mathsf{Rand}(\cdot, p)$ to obtain $\{\alpha_{i,b}\cdot\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}$, $\{\alpha_{i,b}\cdot\tilde{B}'_{i,b}\}_{i\in[m],b\in\{0,1\}}$ and $\mathbf{t}$.
- M outputs

$$\vec{m_0} = (\{\alpha_{1,b}\cdot\tilde{B}_{1,b}\}_{b\in\{0,1\}}, \{\alpha_{m,b}\cdot\tilde{B}_{m,b}\}_{b\in\{0,1\}})$$

$$\vec{m_1} = (\{\alpha_{1,b}\cdot\tilde{B'}_{1,b}\}_{b\in\{0,1\}}, \{\alpha_{m,b}\cdot\tilde{B'}_{m,b}\}_{b\in\{0,1\}})$$

$$\vec{z} = (\{\alpha_{i,b}\cdot\tilde{B}_{i,b}\}_{i\in[m']/\{1,m\},b\in\{0,1\}}, \mathbf{t})$$

  Note that $(\vec{m}_0, \vec{z})$ is identically distributed to $\mathsf{Rand}(\widehat{BP}_0, p)$ and similarly $(\vec{m}_1, \vec{z})$ is identically distributed to $\mathsf{Rand}(\widehat{BP}_1, p)$ As a result, by Proposition 2, we have that M is an entropic message sampler.

Let $(\{S_{i,b}\}_{i\in[m],b\in\{0,1\}}, S_{\mathbf{t}}) = \mathsf{SetSystem}(m, N, \mathsf{inp})$, where inp is the labelling function for the branching programs $\widehat{BP}_0$ and $\widehat{BP}_1$, and let

$$\vec{S}_n = \{S_{1,b}, S_{m,b}\}_{b\in\{0,1\}}$$

$$\vec{T}_n = (\{S_{i,b}\}_{i\in[m']/\{1,m\},b\in\{0,1\}}, S_{\mathbf{t}})$$

We show that M is a computationally $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting message sampler, *i.e.* no nuPPT oracle machine A' can pointwise distinguish the oracles $\mathcal{M}(\vec{m}_0, \vec{z})$ and $\mathcal{M}(\vec{m}_1, \vec{z})$. We note that by Lemma 19 and a Union Bound over A''s queries, the output of $A'^{\mathcal{M}(\vec{m}_0,\vec{z})}$ (resp. $A'^{\mathcal{M}(\vec{m}_1,\vec{z})}$) can be simulated with only oracle access to BP_0 (resp. BP_1), or equivalently, to $C_{n,a,b,0,\mathsf{pk},\mathsf{sk},\hat{a}}$ (resp. $C_{n,a,b,1,\mathsf{pk},\mathsf{sk},\hat{a}}$)[29]. In fact, with high probability over the randomness of M, A' and the simulator, the simulator's output is identical to the output of A'. We observe that this simulation can be

[29]To apply the Union Bound it is important that the query response $C(\vec{m}_b, \vec{z})$ depends only on the queried arithmetic circuit C and the input-output behavior of BP_b as shown in Lemma 19

made *efficient* using the techniques introduced in [16] (i.e. by modifying BP_0 and BP_1 to be *dual-input* branching programs and correspondingly changing SetSystem); this requires encodings elements using sets of size 4 (as opposed to 2 as in our original construction). Let this efficient simulator be Sim.

We would now like to argue that with high probability over the randomness of M and Sim, $\mathsf{Sim}^{BP_0} = \mathsf{Sim}^{BP_1}$. Recall that the circuits $C_{n,a,b,0,\mathsf{pk},\mathsf{sk},\hat{a}}$ (equivalent to BP_0) and $C_{n,a,b,1,\mathsf{pk},\mathsf{sk},\hat{a}}$ (equivalent to BP_1) differ only on inputs x for which $\mathsf{Dec}(\mathsf{sk}, x) = b$ (on these inputs $C_{n,a,b,0,\mathsf{pk},\mathsf{sk},\hat{a}}(x) = 0$, whereas $C_{n,a,b,1,\mathsf{pk},\mathsf{sk},\hat{a}}(x) = 1$). Since b was randomly chosen from an exponentially large set of values, to find such an input with noticeable probability, Sim must query one of the circuits on input a with noticeable probability, otherwise its view is independent of b. However, if the original ciphertext $\hat{a}$ is an encryption of 0 instead of a, then the view of Sim is independent of a, and thus Sim can only query a with negligible probability. Thus by the semantic security of the FHE scheme, the probability that Sim can query a when given BP_0 or BP_1 is negligible. This implies that the outputs of Sim^{BP_0} and Sim^{BP_1} differ with only negligible probability.

We now have that :

- $A'^{\mathcal{M}(\vec{m}_0,\vec{z})} = \mathsf{Sim}^{BP_0}$, except with negligible probability;
- $\mathsf{Sim}^{BP_0} = \mathsf{Sim}^{BP_1}$, except with negligible probability;
- $\mathsf{Sim}^{BP_1} = A'^{\mathcal{M}(\vec{m}_1,\vec{z})}$, except with negligible probability.

By a union bound, we have that $A'^{\mathcal{M}(\vec{m}_0,\vec{z})} = A'^{\mathcal{M}(\vec{m}_1,\vec{z})}$, except with negligible probability. Thus M must be a computationally respecting sampler. Finally, it follows using identically the same argument as in Section 3.4.3 that the message

sampler satisfies the required high-entropy condition and thus is an entropic computationally respecting message sampler.

Now we will show an *nuPPT* adversary A that distinguishes between encodings of $(\vec{m}_0, \vec{z})$ and $(\vec{m}_1, \vec{z})$ when encoded under sets $(\vec{S}_n, \vec{T}_n)$ Note that given encodings of one of $(\vec{m}_0, \vec{z})$ and $(\vec{m}_1, \vec{z})$, A in fact receives either $\mathsf{Obf}(\widehat{BP}_0)$ or $\mathsf{Obf}(\widehat{BP}_1)$. Let us refer to this input to A as O.

A evaluates O on input 0 to receive $(\mathsf{pk}, \hat{a})$, and then simply homomorphically evaluates O on the ciphertext $\hat{a}$ in order to generate a valid encryption of the hidden value b, and then feeds this new ciphertext back into O to reveal the secret bit v, and then outputs v. Thus A succeeds in distinguishing $(\vec{m}_0, \vec{z})$ and $(\vec{m}_1, \vec{z})$ with probability 1. Additionally, note that since O is a constant-width branching program, O can be computed by a NC^1 circuit, thus for this argument it suffices to use a leveled FHE.

We thus have that no graded encoding scheme can satisfy entropic extractable uber security. □

"Plain" Uber Security

Due to the above impossibility result, we here consider a weaker variant of an uber security—which we simply refer to as (plain) "uber security", where we strengthen the "computational validity" condition to a "weak validity" condition where the the algebraic attacker is allowed to be unbounded while making polynomially many queries. Note that weak validity differs from strong validity only in the respect that weak validity does not require the output of the algebraic attacker is *constant* (with overwhelming probability).

Definition 26 (Weakly Respecting Message Sampler)**.** *We say that a nuPPT M is a* weakly $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$-respecting message sampler *(or* weakly valid w.r.t. $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$*) if it satisfies the same conditions as in Definition 11 but where the second bullet is replaced by the following:*

- *For every polynomial p, there exists some polynomial Q such that for every $n \in \mathbb{N}$, every $(\mathsf{sp}, \mathsf{pp})$ in the support of $\mathsf{InstGen}(1^n, 1^{k_n})$, every (deterministic) oracle algorithm A that on input 1^n makes at most $p(n)$ oracle queries,*

$$Pr[(\vec{m}_0, \vec{m}_1, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp}, \vec{p}_0)}(1^n) = A^{\mathcal{M}(\mathsf{pp}, \vec{p}_1)}(1^n)] \geq 1 - Q(n, k_n)/|R|.$$

  where $\vec{p}_b = \{(m_b[i], S_i)\}_{i=1}^{c(n)}, \{(z[i], T_i)\}_{i=1}^{q(n)}$ and $c(n)$ and $q(n)$ are the lengths of $\vec{S}_n$ and $\vec{T}_n$ respectively.

We define *"uber security"* in exactly the same way as semantic security except that we only require the message sampler to be weakly valid (and define entropic uber security in the analogous way). In other words, uber security implies that whenever $\vec{m}_0, \vec{z}$ and $\vec{m}_0, \vec{z}$ are *pointwise statistically close* w.r.t. legal algebraic attackers, encodings of them computationally indistinguishable.

Let us remark that for uber security to imply semantical security, it is important that we restrict the algebraic attacker (in the definition of a weakly valid message sampler) to only make polynomially many queries. Otherwise, even the aGDDH distribution (described in Section 3.3) is not weakly valid: With high probability over $(m_0, m_1, \vec{z})$ sampled from the aGDDH distribution, there always exists *some* legal arithmetic circuit C such that $\mathsf{isZero}(C(m_0, \vec{z})) \neq \mathsf{isZero}(C(m_1, \vec{z}))$.[30] Therefore, an unbounded-query algebraic adversary could

[30]Consider a very simple aGDDH instance, where $|\vec{z}| = 2$, $T_1 = T_2 = S = [k]$. For non-zero z_1, z_2, there always exists some a such that the circuit $C(m, z_1, z_2) = \mathsf{isZero}(m - az_1)$ yields different outputs on input $(m_0, \vec{z})$ and $(m_1, \vec{z})$—namely, $a = z_2$.

simply go over all legal arithmetic circuits and distinguish the elements.

We are not aware of any attacks (like those against *extractable* uber security) against "plain" uber security, and it thus seems like a reasonable strengthening of semantical security, which may have other applications. In fact, we may consider an even further strengthening of this notion—which we refer to as *statistical uber security*— by replacing the the weakly valid message sampler by a *super weakly valid* message sampler which only requires $\vec{m_0}, \vec{z}$ and $\vec{m_1}, \vec{z}$ to be *statistically indistinguishable* by algebraic attackers (as opposed to be *pointwise* statistically indistinguishable); that is, the second bullet in Definition 11 is replaced by:

- For every (computationally unbounded) oracle machine A that makes at most polynomially many oracle queries, there exists a negligible function ε such that for every security parameter $n \in \mathbb{N}$,

$$|Pr[(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k(n)}), (\vec{m_0}, \vec{m_1}, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp}, \vec{p_0})}(1^n, \mathsf{pp}) = 1] -$$
$$Pr[(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k(n)}), (\vec{m_0}, \vec{m_1}, \vec{z}) \leftarrow M(1^n, \mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp}, \vec{p_1})}(1^n, \mathsf{pp}) = 1]| \leq \varepsilon(n)$$

  where $\vec{p_b} = \{(m_b[i], S_i)\}_{i=1}^{c(n)}, \{(z[i], T_i)\}_{i=1}^{q(n)}$ and $c(n)$ and $q(n)$ are the lengths of $\vec{S}_n$ and $\vec{T}_n$ respectively.

3.6.3 Strong Semantical and Uber Security

Recall that in the definition of both validity and weak validity, we consider *arbitrary-size* set-respecting circuits. We may weaken both validity conditions

(and thus obtain stronger notion of semantical and uber security) by restricting attention to only polynomial-size arithmetic circuits. Note that in the context of uber security, this takes us a step closer to extractable uber security (which is impossible under reasonable assumption): we restrict to algebraic attackers that make polynomially-many queries and each query is polynomial-size, but the attacker may generate these queries (and generate its final output) in a computationally unbounded way. We refer to these notions respectively as *strong semantical security* and *strong uber security*.

3.6.4 Weak Semantic Security

We end this section by considering a *weaker* notion of semantical security—let us refer to it as *weak semantical security*—where the definition of a valid message sampler requires the the answer to every set-respecting circuit is *actually* constant (as opposed to only being constant with overwhelming probability); a similar relaxation can be applied also to uber security. While we do not know whether any of these weaker assumptions suffices for obtaining obfuscation (and they do not imply the aGDDH assumption), the weak notion of semantical security suffices for obtaining *witness encryption* [75]—roughly speaking, the notion of witness encryption enables a sender to encrypt a message m using an NP-statement x such that a) if the statement is false, then encodings of any two messages are indistinguishable, and b) if the statement is true, then anyone who has a witness w for x can recover m. Let us briefly sketch this construction:[31] As in [75], we focus on the NP-language Exact-Cover where an x instance consist of sets $S_1, \ldots, S_n \subseteq [k]$; for a true instance, there exists some "exact cover"

[31] The observation that semantically secure multilinear encoding directly implies witness encryption was obtained in a conversation with Sanjam Garg, Craig Gentry and Shai Halevi.

of $[k]$ using a subset of the sets, whereas for a false instance no such exact cover exists. Now, to encrypt the bit m under the instance $S_1, \ldots S_n$, use a multilinear encoding scheme over the set $[k+1]$, encode 1 under each of the sets $S_1, \ldots S_n$ and finally encode m under the set $\{k+1\}$. Clearly anyone who knows an exact cover can obtain an encoding of m under $[k+1]$ (by appropriately multiplying the sets corresponding to the exact cover and additionally the encoding of m under $\{k+1\}$). On the other hand, if the instance is false, there is no exact cover, and thus "legal" algebraic operation can never be used to obtain an encoding under the full set $[k+1]$ and thus zero-testing can never be used; thus indistinguishability of encryptions follows by weak semantical security.

3.7 Technical Lemma

Claim 31. *Fix $m, w \in \mathcal{N}$, and let $p \in \mathcal{N}$ be a prime. Let $\mathcal{D}_0$ be the following distribution:*

$$\mathcal{D}_0 = \{\{R_i\}_{i\in[m]}, \{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}\}$$

where each R_i is a uniformly random invertible matrix in $\mathbb{Z}_p^{w\times w}$ (i.e $det(R_i) \neq 0$, and each $\alpha_{i,b}$ is a uniformly random non-zero scalar in $\mathbb{Z}_p$.

Let $\mathcal{D}_1$ be a distribution defined identically to $\mathcal{D}_0$, except with each R_i being a uniformly random (not necessarily invertible) matrix in $\mathbb{Z}_p^{w\times w}$, and each $\alpha_{i,b}$ a uniformly random (not necessarily non-zero) scalar in $\mathbb{Z}_p$.

Then:

$$\Delta(\mathcal{D}_0, \mathcal{D}_1) \leq 8wm/p$$

where $\Delta(\mathcal{D}_0, \mathcal{D}_1)$ denotes the statistical distance between distributions $\mathcal{D}_0$ and $\mathcal{D}_1$.

Proof. Note that $\mathcal{D}_0$ and $\mathcal{D}_1$ are each uniformly distributed on their respective supports, and that $\mathsf{supp}(\mathcal{D}_0) \subseteq \mathsf{supp}(\mathcal{D}_1)$. Then the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ can be computed as follows:

$$\begin{aligned}
\Delta(\mathcal{D}_0, \mathcal{D}_1) &= \sum_{d \in \mathsf{supp}(\mathcal{D}_0) \cup \mathsf{supp}(D_1)} |\Pr[\mathcal{D}_0 = d] - \Pr[\mathcal{D}_1 = d]| \\
&= \sum_{d \in \mathsf{supp}(\mathcal{D}_0)} |\Pr[\mathcal{D}_0 = d] - \Pr[\mathcal{D}_1 = d]| + \sum_{d \in \mathsf{supp}(\mathcal{D}_1) \setminus \mathsf{supp}(\mathcal{D}_0)} |\Pr[\mathcal{D}_1 = d]| \\
&= \sum_{d \in \mathsf{supp}(\mathcal{D}_0)} |\frac{1}{|\mathsf{supp}(\mathcal{D}_0)|} - \frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}| + \sum_{d \in \mathsf{supp}(\mathcal{D}_1) \setminus \mathsf{supp}(\mathcal{D}_0)} |\frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}| \\
&= (|\mathsf{supp}(\mathcal{D}_0)| \cdot |\frac{1}{|\mathsf{supp}(\mathcal{D}_0)|} - \frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}|) + \\
&\quad (|\mathsf{supp}(\mathcal{D}_1) \setminus \mathsf{supp}(\mathcal{D}_0)| \cdot |\frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}|) \\
&= 2 \cdot (1 - \frac{|\mathsf{supp}(\mathcal{D}_0)|}{|\mathsf{supp}(\mathcal{D}_1)|})
\end{aligned}$$

But notice that $(1 - \frac{|\mathsf{supp}(\mathcal{D}_0)|}{|\mathsf{supp}(\mathcal{D}_1)|})$ can be interpreted as $\Pr[\exists i \in [m], b \in \{0,1\} : \det(R_i) = 0 \vee \alpha_{i,b} = 0]$. For each $i \in [m]$, the probability $det(R_i) = 0$ can be bounded by applying the Schwartz-Zippel lemma to the $det(\cdot)$, which is a polynomial of degree w. Thus we have that $\Pr[det(R_i) = 0] \leq w/p$. Further, each $\alpha_{i,b}$ is zero with probability $1/p$. Hence, applying a union bound, we have that

$$\begin{aligned}
\Delta(\mathcal{D}_0, \mathcal{D}_1) &= 2 \cdot (1 - \frac{|\mathsf{supp}(\mathcal{D}_0)|}{|\mathsf{supp}(\mathcal{D}_1)|}) \\
&\leq 2 \cdot (2m/p + mw/p) \\
&\leq 8wm/p
\end{aligned}$$

□

3.8 Proof of Lemma 20

In this section, we prove Lemma 20, restated below for clarity:

Lemma 24. *Fix $m, n, w \in \mathcal{N}$ and* $\mathsf{inp} : [m] \to [n]$. *Let* $\vec{S} = \mathsf{SetSystem}(m, n, \mathsf{inp}) = (\{S_{i,b}\}_{i\in[m],b\in\{0,1\}}, S_t)$, *and let C be any weakly $\vec{S}$-respecting arithmetic circuit whose output wire is tagged with $T \subseteq [k]$. Then there exists a set $U \subseteq \{0,1,*\}^m$ such that for every branching program BP of width w and length m on n input bits, with input tagging function* inp, *every prime p, and every* $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m,b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$,

(i)

$$C(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \equiv \sum_{u\in U} C_u(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$$

where each C_u is a weakly $\vec{S}$-respecting arithmetic circuit, whose input wires are tagged only with sets $\in \{S_{i,u[i]}\}_{i\in[m]:u[i]\neq *} \cup \{S_t\}$, and whose output wire is tagged with T.

(ii) *Each C_u above is the sum of several "monomial" circuits, where each monomial circuit performs only multiplications of elements in* $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m,b\in\{0,1\}}, \mathbf{t})$, *is weakly $\vec{S}$-respecting, and has output wire tagged with T.*

(iii) *For each C_u above,*

$$C_u(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_u \cdot p_u(\{\tilde{B}_{i,u[i]}\}_{i\in[m]:u[i]\neq *}, \mathbf{t})$$

where p_u is some polynomial, and $\alpha_u = (\prod_{i\in[m]:u[i]\neq *} \alpha_{i,u[i]})$. Furthermore, when p_u is viewed as a sum of monomials, each monomial contains exactly one entry from each $\tilde{B}_{i,u[i]}$ such that $u[i] \neq *$, and possibly one entry from $\mathbf{t}$. Further, p_u can be computed by a weakly $\vec{S}$-respecting circuit whose output wire is tagged with T.

Proof. **Part (i)** We begin by expressing the circuit C as a polynomial in variables $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m,b\in\{0,1\}}, \mathbf{t})$, in the form of a sum of monomials (possibly exponentially many). We do so recursively: we associate each wire w of the circuit with a multiset S_w of pairs of monomials and signs ("+1" or "-1"), such that the sum

of the monomials multiplied by their respective signs computes the same value as the value computed by the circuit at that wire. We eventually output the multiset of monomial pairs corresponding to the output wire. We compute the sets of monomials as follows:

- Any input wire of the circuit reading input variable v can be represented as the set $\{(v, +)\}$.
- The output wire of an addition gate can be represented as the union of the multisets of monomial pairs representing the gates left and right children.
- The output wire of an subtraction gate can be similarly represented as the union of the multisets of the gate's left input wire, and of its right input wire with the "sign" component of every pair negated (from "+1" to "-1" and vice versa), to reflect subtraction.
- For the output wire of a multiplication gate, for each pair (M_1, s_1) in the multiset of its left input and each pair (M_2, s_2) in the multiset of its right input, we add $(M_1 \cdot M_2, s_1 \cdot s_2)$ to the multiset of the output wire.

We note that it holds inductively in the above process that the sum of the monomials in the multiset associated with each wire w in C, multiplied by its appropriate sign, equals the value computed on that wire w.

We also show that each monomial in the set corresponding to a wire can be computed by a weakly $\vec{S}$-respecting circuit whose output wire has the same tag as the wire. This can again be seen inductively:

- This property holds at any input wire of C, since the only monomial in the set can be computed using the input wire itself as the "monomial circuit".

- This property also holds at any output wire of an addition or subtraction gate, since the circuit corresponding to any monomial in this wire's set is the same as the circuit for the monomial from the corresponding incoming wire to the gate.
- Finally, at the output wire of a multiplication gate G, for any monomial M in this wire's set computed as the product of monomials $M1$ and $M2$, the circuit for M is simply the circuit for each of $M1$ and $M2$, joined by a multiplication gate. Since G performs a set respecting multiplication, and the output wires of $M1$ and $M2$'s circuits have the same tags as the input wires of G, we have that the multiplication joining $M1$ and $M2$'s circuits to produce M's circuit is set-respecting, and so the circuit corresponding to M is a weakly $\vec{S}$-respecting circuit whose output wire has the same tag as the output wire of G.

Thus each of the monomials in the decomposition of C can be represented as a weakly set-respecting arithmetic circuit with output wire tagged with T, where this circuit simply multiplies together all terms in the monomial in some order, and performs no additions. Finally, the tags of the input wires of these monomial circuits must be mutually disjoint, otherwise the monomial circuit would perform a non-set-respecting multiplication at some level.

We label each monomial M with an element $u \in \{0, 1, *\}^m$, where $u[i] = b$ if $S_{i,b}$ is the label on one of input wires in M's circuit representation, and $u[i] = *$ if neither $S_{i,0}$ and $S_{i,1}$ are labels on any of M's input wires. We note that no monomial can have both $S_{i,0}$ and $S_{i,1}$ on its input wires because these two sets are not disjoint, and the tags of the input wires of the monomial circuits must be mutually disjoint.

We now let C_u be the circuit representing the subtraction of all momonials in the the decomposition of C labelled with u and sign (-1) from the sum of all momonials in the the decomposition of C labelled with u and sign $(+1)$. Since each monomial can be represented as a weakly set-respecting circuit with output wire tagged with T, adding several monomials together is a set-respecting operation, as is subtracting several monomials from the sum, and thus each C_u is a weakly set-respecting circuit. Further, since each monomial circuit has output wire tagged with T, each C_u also has output wire tagged with T. Further, by the way we labelled each monomial, each of the input wires of C_u is tagged only with sets $\in \{S_{i,u[i]}\}_{i\in[m]:u[i]\neq *} \cup \{S_t\}$. Finally, if we sum over all the u, we capture all the monomials in the decomposition of C multiplied by their respective signs, so we have that $\sum_u C_u = C$.

Part (ii) We observe that by construction of C_u, it is a sum of several monomial circuits each of which performs only multiplications of its inputs, is weakly $\vec{S}$-respecting, and has output wire tagged with T.

Part (iii) From part (ii), we have that for each C_u, it is a sum of several monomial circuits each of which performs only multiplications of its inputs, is weakly $\vec{S}$-respecting, and has output wire tagged with T. Furthermore, for each such monomial circuit the input tags are drawn from sets $\in \{S_{i,u[i]}\}_{i\in[m]:u[i]\neq *} \cup \{S_t\}$. In fact, each of these monomials must contain exactly one input wire tagged with each of the sets in $\{S_{i,u[i]}\}_{i\in[m]:u[i]\neq *}$, and exactly one set tagged with S_t if and only if $S_t \subseteq T$. This means that each of these monomials is the product of one element chosen from each of the matrices $(\{\alpha_{i,u[i]} \cdot \tilde{B}_{i,u[i]}\}_{i\in m:u[i]\neq *}$, and possibly one element from $\mathbf{t}$. Thus each monomial in the decomposition of C_u has a common factor of $\alpha_u = (\prod_{i\in[m]:u[i]\neq *} \alpha_{i,u[i]})$.

We can now write C_u as a polynomial (namely the sum of its monomials multiplied by their respective signs), and by factoring α_u from each of it monomials and letting p_u be the remaining polynomial, we have, as required, that

$$C_u(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_u \cdot p_u(\{\tilde{B}_{i,u[i]}\}_{i\in[m]:u[i]\neq *}, \mathbf{t})$$

Finally, we note that computing p_u is the same as computing C_u if the alphas are set to 1. Since C_u is $\vec{S}$-respecting, we thus have that p_u can be computed by a weakly $\vec{S}$-respecting circuit whose output wire is tagged with T. □

3.9 Proof of Lemma 8

In this section we prove Lemma 8, restated below for clarity.

Lemma 32. *Let $c, \varepsilon \in \mathbb{N}$ and $\mathcal{E}$ be an (c, k^{ε})-semantically secure encoding scheme. Then for every polynomial $q(k)$ there exists a $(c, q(k))$-semantically secure encoding scheme.*

Proof. Consider any polynomial $q(\cdot)$ and constants c, ε. Given a (c, k^{ε})-semantically secure encoding $\mathcal{E}$, we construct a new multilinear encoding scheme $\mathcal{E}'$ and prove that $\mathcal{E}'$ is $(c, q(k))$-semantically secure. Let $(\mathsf{InstGen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Sub}, \mathsf{Mult}, \mathsf{isZero})$ be the algorithms associated with $\mathcal{E}$. We define a new encoding scheme $\mathcal{E}' = (\mathsf{InstGen}', \mathsf{Enc}', \mathsf{Add}', \mathsf{Sub}', \mathsf{Mult}', \mathsf{isZero}')$ as follows.

- $\mathsf{InstGen}'$ on input $(1^n, 1^k)$ runs $(\mathsf{pp}, \mathsf{sp}) \leftarrow \mathsf{InstGen}(1^n, 1^{(q(k)+1)^{1/\varepsilon}})$ and generates an encoding of a uniformly random non-zero element e under the set $\{k+1, \ldots (q(k)+1)^{1/\varepsilon}\}$ by running $u^1 \leftarrow \mathsf{Enc}(\mathsf{sp}, e, \{k+1, \ldots (q(k)+1)^{1/\varepsilon}\})$. $\mathsf{InstGen}'$ outputs (pp, u^1) as the public parameters and sp as the secret parameters.

- $\mathsf{Enc}', \mathsf{Add}', \mathsf{Sub}', \mathsf{Mult}'$ are identical to $\mathsf{Enc}, \mathsf{Add}, \mathsf{Sub}, \mathsf{Mult}$ respectively.
- isZero' takes as input public parameters (pp, u^1) and an encoding u under the set $[k]$ to zero-test. isZero' simply outputs $\mathsf{isZero}(\mathsf{Mult}(\mathsf{pp}, u, u^1))$. The correctness of isZero' follows from that of isZero and the fact that $\mathsf{Mult}(\mathsf{pp}, u, u^1)$ returns an encoding, under the set $[(q(k)+1)^{1/\varepsilon}]$, of an element which is zero if and only if u is an encoding of zero.

It is easy to see that the correctness of $\mathcal{E}'$ follows from that of $\mathcal{E}$.

We now show that $\mathcal{E}'$ is $(c, q(k))$-semantically secure. Assume for contradiction there exists a polynomial $k'(\cdot)$, ensemble $\{\vec{S}'_n, \vec{T}'_n\}_{n\in\mathcal{N}}$ of sets where $|\vec{S}'_n| = c$, $|\vec{T}'_n| = q(k'(n))$, $\{\vec{S}'_n, \vec{T}'_n\}_{n\in\mathcal{N}}$-respecting message sampler M' and nuPPT adversary A' such that for sufficiently large n, A' distinguishes encodings of elements as described in the semantic security game in Definition 12.

Let $k(\cdot)$ be a polynomial such that $k(n) = (q(k'(n))+1)^{1/\varepsilon}$. For every $n \in \mathbb{N}$, let $\vec{S}_n, \vec{T}_n$ be a sequence of sets over $[k(n)]$ where $\vec{S}_n = \vec{S}'_n$ and $\vec{T}_n = (\vec{T}'_n, \{k'(n)+1, \ldots k(n)\})$. We will construct a $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting message sampler M and nuPPT adversary A such that (M, A) breaks the (c, k^{ε})-semantic security of $\mathcal{E}$.

We define the message sampler M as follows: on input 1^n, $\mathsf{pp} \in \mathsf{InstGen}(1^n, 1^{k(n)})$, M samples $(\vec{m}_0, \vec{m}_1, \vec{z}) \leftarrow M'(1^n, \mathsf{pp})$. and outputs the elements $(\vec{m}_0, \vec{m}_1, (\vec{z}, e))$ where e is a uniformly random non-zero element, *i.e.* M outputs the same elements sampled by M' with an additional element e. Note that M' samples elements based only on the ring associated with the public parameters pp, which in this case, is the same ring associated with $\mathsf{pp}' \in \mathsf{InstGen}'(1^n, 1^{k'(n)})$.

To show that M is $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting, we claim that for any $(\vec{S}_n, \vec{T}_n)$-respecting circuit C acting on $(\vec{m}_0, \vec{m}_1, (\vec{z}, e))$ there exists a $(\vec{S}'_n, \vec{T}'_n)$-respecting cir-

cuit C' acting on $(\vec{m}_0, \vec{m}_1, \vec{z})$ such that $\mathsf{isZero}(C(\cdot)) = \mathsf{isZero}(C'(\cdot))$. C' is simply the circuit C computes to obtain an element corresponding to the set $[k'(n)]$, with which it must multiply an element under the set $\{k'(n)+1, \ldots k(n)\}$ to reach the target set $[k(n)]$. Since M' is $\{\vec{S}'_n, \vec{T}'_n\}_{n\in\mathcal{N}}$-respecting, the output of $\mathsf{isZero}(C'(\cdot))$ is constant with overwhelming probability. Therefore, the output of $\mathsf{isZero}(C(\cdot))$ is constant with overwhelming probability too, and M is $\{\vec{S}_n, \vec{T}_n\}_{n\in\mathcal{N}}$-respecting.

We now define a nuPPT adversary A that breaks the semantic security of $\mathcal{E}$. On input encodings $\vec{u}$ and public parameters pp, A simply removes the last encoding u from $\vec{u}$ and runs A' on input public parameters (pp, u) and the remaining encodings. Observe that for any security parameter n, the output of A in the semantic security game in Definition 12 when played with message sampler M and sets $\vec{S}_n, \vec{T}_n$ is identical to the output of A' in the game played with message sampler M' and sets $\vec{S}'_n, \vec{T}'_n$. Recall that $\vec{S}_n, \vec{T}_n$ are sequences of sets over $[k(n)]$ and $|\vec{S}_n| = c$ and $|\vec{T}_n| = k(n)^\varepsilon$. Therefore, this contradicts the (c, k^ε)-semantic security of $\mathcal{E}$. □

3.10 Acknowledgments

We are very grateful to Benny Applebaum, Omer Paneth, Ran Canetti, Kai-Min Chung, Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, abhi shelat, Hoeteck Wee and Daniel Wichs for many helpful comments. We are especially gratefeul to Shai for pointing out the connection between semantical security for multilinear encodings and the "uber" assumption for bilinear maps of [35], and for several very useful conversations about multilinear encodings and the security of the [66] constructions, to Amit for several helpful conversations about the

presentation of our results, and Benny for suggesting we make our proof more modular (which lead to the notion of neigboring-matrix branching programs). Finally thanks to the anonymous Crypto reviewers for their useful comments. Thanks so very much!

CHAPTER 4

OUTPUT-COMPRESSING RANDOMIZED ENCODINGS AND APPLICATIONS

This chapter contains joint work with Huijia Lin (UCSB), Rafael Pass (Cornell University) and Karn Seth (Cornell University).

4.1 Introduction

The beautiful notion of a *randomized encoding (RE)*, introduced by Ishai and Kushilevitz [100], aims to trade the computation of a "complex" (deterministic) function Π on a given input x for the computation of a "simpler" randomized function—the "encoding algorithm"—whose output distribution $\hat{\Pi}(x)$ encodes $\Pi(x)$ (from which $\Pi(x)$ can be efficiently decoded, or "evaluated"). Furthermore, the encoding $\hat{\Pi}(x)$ should not reveal anything beyond $\Pi(x)$; this is referred to as the *privacy*, or *security*, property of randomized encodings and is typically defined through the simulation paradigm [95].

Most previous work have focused on randomized encodings where encodings can be computed in lower parallel-time complexity than what is required for computing the original function Π. For instance, all log-space computations have *perfectly-secure* randomized encodings in $\mathbf{NC}^0$ [100, 102, 8], and assuming low-depth pseudo-random generators, this extends to all polynomial-time computations (with computational security) [10, 138]. Such randomized encodings have been shown to have various applications to parallel cryptography, secure computation, verifiable delegation, etc. (see [6] for a survey).

Bitansky, Garg, Lin, Pass and Telang [31] recently initiated a study of *succinct randomized encodings* where we require that the *time* required to compute $\hat{\Pi}(x)$ is smaller than the time required to compute $\Pi(x)$; their study focused on functions Π that have *single-bit* outputs. [31, 54, 112] show that subexponentially-secure indistinguishability obfuscators ($i\mathcal{O}$) [17, 70] and one-way functions[1] imply the existence of such succinct randomized encodings for all polynomial-time Turing machines that output just a single bit.

We here further the study of such objects, focusing on functions Π with *long* outputs. Given a description of a Turing machine Π and an input x, we consider two notions of efficiency for randomized encodings $\hat{\Pi}(x)$ of $\Pi(x)$ with running time T.

- *compact RE*: Encoding time (and thus also size of the encodings) is $\text{poly}(|\Pi|, |x|, \log T)$
- *sublinear RE*: Encoding time (and thus also size) is bounded by $\text{poly}(|\Pi|, |x|) * T^{1-\epsilon}$, for some $\epsilon > 0$.

We assume without loss of generality that the randomized encoding $\hat{\Pi}(x)$ of Π, x itself is a program, and that the decoding/evaluation algorithm simply executes $\hat{\Pi}(x)$.

It is easy to see that for such notions of efficiency, the standard simulation-based notion of security is impossible to achieve—roughly speaking, the simulator given just $\Pi(x)$ needs to output a "compressed" version of it, which is impossible if $\Pi(x)$ has high pseudo-Kolmogorov complexity (e.g., if Π is a PRG); we formalize this argument in Theorem 32 in Section 4.7. Consequently, we

[1]The one-way function assumption can be weakened to assume just that $\mathsf{NP} \not\subseteq io\mathsf{BPP}$ [110].

consider weaker indistinguishability-based notions of privacy. One natural indistinguishability based notion of privacy simply requires that encoding $\hat{\Pi}_0(x_0)$ and $\hat{\Pi}_1(x_1)$ are indistinguishable as long as $\Pi_0(x_0) = \Pi_1(x_1)$ and $\mathsf{Time}(\Pi_0(x_0)) = \mathsf{Time}(\Pi_1(x_1))$, where $\mathsf{Time}(\Pi(x))$ is the running-time of $\Pi(x)$; such a notion was recently considered by Ananth and Jain [4]. In this work, we consider a stronger notion which requires indistinguishability of $\hat{\Pi}_0(x_0)$ and $\hat{\Pi}_0(x_1)$ as long as Π_0, x_0 and Π_1, x_1 are sampled from some distributions such that $\Pi_0(x_0), \mathsf{Time}(\Pi_0(x_0))$ and $\Pi_1(x_1), \mathsf{Time}(\Pi_1(x_1))$ are indistinguishable. We refer to this notion as *distributional indistinguishability security*, and note that it easily follows that the standard simulation-based security implies distributional indistinguishability security.

The goal of this paper is to investigate compact and sublinear RE satisfying the above-mentioned distributional indistinguishability notion. For the remainder of the introduction, we refer to randomized encodings satisfying distributional indistinguishability security as simply *RE*. For comparison, we refer to randomized encodings with the weaker (non-distributional) indistinguishability security as *weak RE*.

We note here that [78] previously introduced a very similar notion of distributional indistinguishability in the setting of reusable garbled circuits. In the CRS model, reusable garbled circuits are equivalent to a "secret key" version of randomized encoding, thus the [78] security notion can be seen as very related to ours. Indeed, similar to our approach in this work, [78] also used distributional indistinguishability to achieve reusable garbled circuits with long outputs, and in doing so, circumvent an impossibility result. [2]

Compact RE v.s. Obfuscation Before turning to describe our results, let us point

[2]We thank Daniel Wichs for pointing out the connection to [78].

out that RE can be viewed as (a degenerate form) of obfuscation for special classes of programs.

Recall that an indistinguishability obfuscator (*iO*) [17, 70] is a method O for "scrambling" a program Π into $O(\Pi)$ such that for any two functionally equivalent programs Π_0, Π_1 (that is, their outputs and run-time are the same on all inputs,) $O(\Pi_0)$ is indistinguishable from $O(\Pi_1)$. *iO* for Turing machines [17, 42, 2] additionally requires that the size of the obfuscated code does not grow (more than polylogarithmically) with the running-time of the Turing machine.

We may also consider a useful strengthening of this notion—which we call "puncturable *iO* "—which, roughly speaking, requires indistinguishability of $O(\Pi_0)$ and $O(\Pi_1)$ as long as Π_0 and Π_1 differ *on at most one input* x^* and their outputs on input x^* are indistinguishable. More precisely, we say that a distribution D is *admissible* if there exists some x^* such that a) for every triple (Π_0, Π_1, Π) in the support of D, and every $x \neq x^*$, it holds that $\Pi_0(x) = \Pi_1(x) = \Pi(x)$, and b) $(\Pi, \Pi_0(x^*))$ and $(\Pi, \Pi_1(x^*))$ are computationally indistinguishable when (Π_0, Π_1, Π) are sampled randomly from D. Puncturable *iO* requires indistinguishability of $O(\Pi_0)$ and $iO(\Pi_1)$ for Π_0, Π_1 sampled from any admissible distribution. Interestingly, for the case of *circuits*, puncturable *iO* is equivalent to (standard) *iO* .[3] Indeed, such a notion is implicitly used in the beautiful and powerful punctured-program paradigm by Sahai and Waters [134], and all its applications. (In this context, think of Π as the "punctured" version of the programs Π_0, Π_1.)

In the case of Turing machines, when restricting to the degenerate case of

[3]To see this, consider a hybrid program $\Pi^y(x)$ that runs $\Pi(x)$ if $x \neq x^*$ and otherwise (i.e., if $x = x^*$ outputs y). By the *iO* property we have that for every Π, Π_0, Π_1 in the support of D, $O(\Pi^{\Pi_b(x^*)})$ is indistinguishable from $O(\Pi_b)$. Thus, if $O(\Pi_0)$, $O(\Pi_1)$ are distinguishable, so are $O(\Pi^{\Pi_0(x^*)})$, $O(\Pi^{\Pi_1(x^*)})$, which contradicts indistinguishability of $(\Pi, \Pi_0(x^*))$ and $(\Pi, \Pi_1(x^*))$.

Turing machines with no inputs (or more precisely, we only consider the execution of $\Pi()$ on the "empty" input), the notion of $i\mathcal{O}$ for Turing machines is equivalent to the notion of a compact *weak* RE. Compact RE, on the other hand, is equivalent to puncturable $i\mathcal{O}$ for Turing machines (without inputs). (Jumping ahead, as we shall see, for the case of Turing machines it is unlikely that puncturable $i\mathcal{O}$ is equivalent to standard $i\mathcal{O}$.)

4.1.1 Our results

iO from sublinear RE We start by showing that sublinear RE is an extremely useful primitive: Subexponentially-secure sublinear RE implies indistinguishability obfuscators for all polynomial-size circuits.

Theorem 17. *The existence of subexponentially-secure sublinear RE and one-way functions implies the existence of subexponentially-secure $i\mathcal{O}$ for circuits.*

Before continuing, let us mention that Theorem 1 is related to a recent beautiful result by Ananth and Jain [4] which shows that *under the LWE assumption*, subexponentially-secure compact RE (satisfying only the weak indistinguishability security) implies $i\mathcal{O}$ for circuits. Their construction goes from RE to *functional encryption* (FE) [37], and then from FE to $i\mathcal{O}$; (the first step relies on previous constructions of FE [92, 97], while the second step relies on a sequence of complex transformations and analysis). In contrast, the proof of Theorem 1 directly constructs $i\mathcal{O}$ from RE in a surprisingly simple way: We essentially use the GGM construction [87] that builds a PRF from a PRG using a tree, but replace the PRG with a RE. Let us explain in more details below.

Consider a program Π taking n-bit inputs. We consider a binary tree where the leaves are randomized encodings of the function applied to all possible inputs, and each node in the tree is a randomized encoding that generates its two children. More precisely, given a sequence of bits $x_1, \cdots, x_i$, let $\tilde{\Pi}_{R,x_1,\cdots,x_i}$ denote an (input-less) program that

- if $i = n$ simply outputs a RE of the program Π and input $(x_1, \cdots, x_n)$ using R as randomness, and
- otherwise, after expanding R_0, R_1, R_2, R_3 from R using a PRG, outputs randomized encodings of (input-less) programs $\tilde{\Pi}_{R_0,x_1,\cdots,x_i,0}$ and $\tilde{\Pi}_{R_1,x_1,\cdots,x_i,1}$ using respectively R_2, R_3 as randomness.

We associate each node in the binary tree that has index $x_1, \cdots, x_i$ with a randomized encoding of the program $\tilde{\Pi}_{R,x_1,\cdots,x_i}$, denoted as $\hat{\Pi}_{R,x_1,\cdots,x_i}$. In particular, the root of the tree is associated with a randomized encoding $\hat{\Pi}$ of the (initial) program $\tilde{\Pi}_R$ hardwired with a randomly chosen R.

The obfuscation of Π is now a program with the "root" $\hat{\Pi}$ hardcoded, and given an input x, computes the path from the root to the leaf x – by recursively evaluating the randomized encodings associated with nodes on the path – and finally outputs the evaluation of the leaf. More precisely, on input x, evaluate $\hat{\Pi}$ to obtain $\hat{\Pi}_0, \hat{\Pi}_1$, next evaluate $\hat{\Pi}_{x_1}$ to obtain $\hat{\Pi}_{x_1,0}, \hat{\Pi}_{x_1,1}$, so on and so forth until $\hat{\Pi}_{x_1,\cdots,x_n}$ is evaluated, yielding the output $\Pi(x_1, \cdots, x_n)$.

Note that for any two functionally equivalent programs, the randomized encodings associated with individual leaf node are computationally indistinguishable by the indistinguishability security property (the non-distributional version suffices here). Then, by the distributional indistinguishability security,

the randomized encodings associated with tree nodes one layer above are also indistinguishable. Thus, by induction, it follows that the roots are indistinguishable, which implies that obfuscations of functionally equivalent programs are indistinguishable. Let us note that the reason that subexponential security is needed is that each time we go up one level in the tree (in the inductive argument), we lose at least a factor 2 in the indistinguishability gap (as each node generates two randomized encodings, its children). Hence, we need to ensure that encodings are at least $\text{poly}(2^n)$-indistinguishable, which can be done by scaling up the security parameter.

On the existence of Compact and Sublinear RE We next turn to investigating the existence of compact and sublinear RE. We show—assuming just the existence of subexponentially-secure one-way functions—*impossibility* of subexponentially-secure sublinear (and thus also compact) RE.[4]

Theorem 18. *Assume the existence of subexponentially secure one-way functions. Then, there do not exists subexponentially-secure sublinear RE.*

As observed above, compact RE can be interpreted as a stronger notion of $i\mathcal{O}$ (which we referred to as *puncturable* $i\mathcal{O}$) for "degenerate" input-less Turing machines, and as such Theorem 2 rules out (assuming just one-way functions) such a natural strengthening of $i\mathcal{O}$ for (input-less) Turing machines. We note that this impossibility stands in contrast with the case of *circuits* where puncturable $i\mathcal{O}$ is equivalent to $i\mathcal{O}$.

We remark that although it may seem like Theorem 2 makes Theorem 1 pointless, it turns out that Theorem 1 plays a crucial role in the proof of Theorem

[4]This result was established after hearing that Bitansky and Paneth had ruled out compact RE assuming public-coin differing-input obfuscation for Turing Machines and collision-resistant hashfunctions. We are very grateful to them for informing us of their result.

2: Theorem 2 is proven by first ruling out sublinear (even just polynomially-secure) RE *assuming* $i\mathcal{O}$ and one-way functions. Next, by using Theorem 1, the $i\mathcal{O}$ assumption comes for free if considering subexponentially-secure RE. That is, assuming one-way functions, we have the following paradigm:

$$\text{exp secure sublinear RE} \overset{\text{Theorem 1}}{\Longrightarrow} i\mathcal{O}$$

$$\Longrightarrow \text{impossibility of (poly secure) sublinear RE}$$

Let us now briefly sketch how to rule out sublinear RE assuming $i\mathcal{O}$ and one-way functions (as mentioned, Theorem 2 is then deduced by relying on Theorem 1). The idea is somewhat similar to the non-black-box zero-knowledge protocol of Barak [15].

Let $\Pi^b_{s,u}$ be a program that takes no input and outputs a sufficiently long pseudo-random string $y = \mathsf{PRG}(s)$ and an indistinguishability obfuscation $\tilde{R}^b_y$ (generated using pseudo-random coins $\mathsf{PRG}(u)$) of the program R^b_y. The program R^b_y takes input Σ of length $|y|/2$, and outputs b iff Σ, when interpreted as an input-less Turing machine, generates y; in all other cases, it outputs $\perp$.[5] We note that the size of the program $\Pi^b_{s,u}$ is linear in the security parameter λ, whereas the pseudo-random string y it generates could have length $|y| = \lambda^\alpha$ for any sufficiently large constant α.

Consider the pair of distributions $\Pi^0_{U_\lambda,U_\lambda}$ and $\Pi^1_{U_\lambda,U_\lambda}$ that samples respectively programs $\Pi^0_{s,u}$ and $\Pi^1_{s,u}$ as described above with random s and u. We first argue that their outputs are computationally indistinguishable. Recall that the output of $\Pi^b_{s,u}$ is a pair $(y, \tilde{R}^b_y)$. By the pseudorandomness of PRG, this output distribution is indistinguishable from $(X, \tilde{R}^b_X)$ where X a uniformly distributed random

[5] To enable this, we require $i\mathcal{O}$ for bounded-input Turing machines, whereas Theorem 1 only gives us $i\mathcal{O}$ for circuits. However, by the results of [31, 54, 112] we can go from $i\mathcal{O}$ for circuits to $i\mathcal{O}$ for bounded-inputs Turing machines.

variable over λ^α bit strings. With overwhelming probability X has high Kolmogorov complexity, and when this happens R^b_X is functionally equivalent to the program $R_\perp$ that always outputs $\perp$. Therefore, by the security of the iO , the output of programs sampled from $\Pi^b_{U_\lambda,U_\lambda}$ is computationally indistinguishable to $(X, \tilde{R}_\perp)$, and hence outputs of $\Pi^0_{U_\lambda,U_\lambda}$ and $\Pi^1_{U_\lambda,U_\lambda}$ are indistinguishable.

Let us now turn to showing that randomized encodings of $\Pi^0_{U_\lambda,U_\lambda}$ and $\Pi^1_{U_\lambda,U_\lambda}$ can be distinguished. Recall that a randomized encoding $\hat{\Pi}^b$ of $\Pi^b_{U_\lambda,U_\lambda}$ itself can be viewed as a (input-less) program that outputs $(y, \tilde{R}^b_y)$. Given $\hat{\Pi}^b$, the distinguisher can thus first evaluate $\hat{\Pi}^b$ to obtain $(y, \tilde{R}^b_y)$ and next evaluate $\tilde{R}^b_y(\hat{\Pi}^b)$ to attempt to recover b. Note that $\hat{\Pi}^b$ clearly is a program that generates y (as its first input); furthermore, if the RE scheme is compact, the length of the program $|\hat{\Pi}^b|$ is bounded by $\text{poly}(\lambda, \log \lambda^\alpha)$, which is far smaller than $|y|/2 = \lambda^\alpha/2$ when α is sufficiently large. Therefore, $\Sigma = \hat{\Pi}^b$ is indeed an input that makes $\tilde{R}^b_y$ output b, enabling the distinguisher to distinguish $\hat{\Pi}^0$ and $\hat{\Pi}^1$ with probability close to 1!

Finally, if the RE is only sublinear, the length of the encoding $|\hat{\Pi}^b|$ is only sublinear in the output length, in particular, bounded by $\text{poly}(\lambda)(\lambda^\alpha)^{1-\epsilon}$ for some constant $\epsilon > 0$. If $\alpha > 1/(1-\epsilon)$ (which clearly happens if ϵ is sufficiently small), then we do not get enough "compression" for the above proof to go through. We circumvent this problem by composing a sublinear RE with itself a sufficient (constant) number of times—to compose once, consider creating randomized encoding of the randomized encoding of a function, instead of the function itself; each time of composition reduces the size of the encoding to be w.r.t. a smaller exponent $1 - \epsilon'$. Therefore, it is without loss of generality to assume that ϵ is any sufficiently *big* constant satisfying $\alpha << 1/(1-\epsilon)$; so the desired compression occurs.

Sublinear RE in the CRS model from sublinear FE Despite Theorem 2, not all is lost. We remark that any sublinear functional encryption scheme (FE) [4, 33] almost directly yields a sublinear RE in the *Common Reference String (CRS)* model; roughly speaking, an FE scheme is called sublinear if the encryption time is sublinear in the size of the circuit that can be evaluated on the encrypted message.

Theorem 19. *Assume the existence of subexponentially-secure sublinear (resp. compact) FE. Then there exists a subexponentially-secures sublinear (resp. compact) RE in the CRS model.*

Furthermore, Theorem 1 straightforwardly extends also to RE in the CRS model. Taken together, these result provide an alternative, modular, simpler proof of the recent results of Ananth and Jain [4] and Bitansky and Vaikuntanathan [33] showing that subexponentially-secure sublinear FE implies subexponentially-secure iO . (All these approaches, including a related work by Brakerski, Komargodski and Segev [49] have one thing in common though: they all proceed by processing inputs one bit at a time, and hard-coding parts of input to the program.)

Theorem 20 (informal, alternative proof of [33, 4])**.** *Assume the existence of subexponentially-secure sublinear FE. Then there exists a subexponentially-secure iO for circuits.*

But there are also other ways to instantiate sublinear RE in the CRS model. We show that under the *subexponential LWE assumption* (relying on [93, 3, 97]) sublinear RE in the CRS model can be based on a significantly weaker notion of sublinear FE—namely FE schemes where the encryption time may be fully polynomial (in the size of the circuit to be evaluated) but only the *size of the ciphertext*

is sublinear in the circuit size—we refer to this notion as a *FE with sublinear ciphertexts*. Roughly speaking, we show this by 1) transforming the "succinct" FE (i.e. compact FE for 1-bit outputs) of [93, 3] into an RE which depends linearly on the output length but only polylogarithmically on the running time, 2) transforming an FE with sublinear ciphertext into an RE with "large" running-time but short output, and 3) finally composing the two randomized encodings (i.e, computing the step 1 RE of the step 2 RE).

Combining this result with (the CRS-extended version of) Theorem 1, we get:

Theorem 21 (informal)**.** *Assume the existence of subexponentially-secure FE with sublinear ciphertexts and the subexponential LWE assumption. Then there exists a subexponentially-secure iO for circuits.*

Toward Turing Machine Obfuscation with Unbounded Inputs We finally address the question of constructing indistinguishability obfuscators for Turing machines with *unbounded* inputs. (For the case of Turing machine obfuscation with unbounded-length inputs, the same obfuscated code needs to work for every input-length, and in particular, the size of the obfuscated code cannot grow with it.) Although it is known that subexponentially secure *iO* for circuits implies *iO* for Turing machines with *bounded inputs lengths* [31, 54, 112] , the only known construction of *iO* for Turing machines with unbounded inputs relies on (public-coin) differing-input obfuscation for circuits and (public-coin) SNARKs [42, 2, 105]—these are strong "extractability" assumptions (and variants of them are known to be implausible [28, 72, 32]).

We note that the construction from Theorem 1 easily extends to show that

subexponentially-secure *compact* RE implies *iO* for Turing machines with unbounded input: instead of having a binary tree, we have a ternary tree where the "third" child of a node is always a leaf; that is, for a tree node corresponding to $x_1, \cdots, x_i$, its third child is associated with a randomized encoding of program Π, and input $(x_1, \cdots, x_i)$, which can be evaluated to obtain output $\Pi(x_1, \cdots x_i)$. Then, by using a tree of *super-polynomial* depth, we can handle any polynomial-length input. Note that since obfuscating a program only involves computing the root RE (as before), the obfuscation is still efficient. Moreover, for any input, we still compute the output of the program in time polynomial in the length of the input by evaluating the "third" child of the node when all input bits have been processed.[6]

But as shown in Theorem 2, compact RE cannot exist (assuming one-way functions)! However, just as for the case of differing-inputs obfuscation and SNARKs, we may assume the existence of compact RE for *restricted* types of "nice" distributions (over programs and inputs), for which impossibility does not hold, yet the construction in Theorem 1 still works. We formalize one natural class of such distributions, and may assume that the *iO* for bounded-input Turing machines construction of [112] (based on *iO* for circuits) yields such a compact RE (for the restricted class of distributions). This yields a new candidate construction of unbounded input Turing machines (based on a very different type of assumption than known constructions).

[6]Proving security becomes slightly more problematic since there is no longer a polynomial bound on the depth of the tree (recall that we required poly(2^n)-indistinguishable RE to deal with inputs of length n). Thus issue, however, can be dealt with by using larger and larger security parameters for RE that are deeper down in the tree.

4.2 Preliminaries

Let $\mathcal{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \ldots, n\}$. We denote by PPT probabilistic polynomial time Turing machines. The term negligible is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from non-negative integers to reals is called negligible if for every constant $c > 0$ and all sufficiently large n, it holds that $\nu(n) < n^{-c}$.

Turing machine notation For any Turing machine Π, input $x \in \{0,1\}^*$ and time bound $T \in \mathbb{N}$, we denote by $\Pi^T(x)$ the output of Π on x when run for T steps. We refer to $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ as a class of Turing machines. One particular class we will consider is the class of Turing machines that have 1-bit output. We call such a machine a Boolean Turing machine. Throughout this paper, by *Turing machine* we refer to a machine with *multi-bit* output unless we explicitly mention it to be a Boolean Turing machine.

4.2.1 Concrete Security

Definition 13 ($(\lambda_0, S(\cdot))$-indistinguishability)**.** *A pair of distributions X, Y are S-indistinguishable for some $S \in \mathbb{N}$ if every S-size distinguisher D it holds that*

$$|\Pr[x \xleftarrow{\$} X : D(x) = 1] - \Pr[y \xleftarrow{\$} Y : D(y) = 1]| \leq \frac{1}{S}$$

A pair of ensembles $\{X_\lambda\}$, $\{Y_\lambda\}$ are $(\lambda_0, S(\cdot))$-indistinguishable for some $\lambda_0 \in \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$ if for every security parameter $\lambda > \lambda_0$, the distributions X_λ and Y_λ are $S(\lambda)$ indistinguishable.

Discussion on $(\lambda_0, S(\cdot))$-indistinguishability: We remark that the above definition requires that there is a universal λ_0 that works for all distinguisher D. A seemingly weaker variant could switch the order of quantifiers and only require that for every distinguisher D there is a λ_0. We show that the above definition is w.l.o.g, since it is implied by the following standard definition with auxiliary inputs in the weaker fashion.

Let U be a universal TM that on an input x and a circuit C computes $C(x)$. Let $S' : N \to N$ denote the run time $S'(S)$ of U on input a size S circuit.

Definition 14. *A pair of ensembles $\{X_\lambda\}$, $\{Y_\lambda\}$ are $S(\cdot)$-indistinguishable if for every $S' \circ S(\cdot)$-time uniform TM distinguisher D, there exists a $\lambda_0 \in N$, such that, for every security parameter $\lambda > \lambda_0$, and every auxiliary input $z = z_\lambda \in \{0,1\}^*$,*

$$|\Pr[x \xleftarrow{\$} X_\lambda : D(1^\lambda, x, z) = 1] - \Pr[y \xleftarrow{\$} Y_\lambda : D(1^\lambda, y, z) = 1]| \leq \frac{1}{S(\lambda)}$$

This definition implies $(\lambda_0, S(\cdot))$-indistinguishability. Consider a distinguisher D that on input $(1^\lambda, x, z)$ runs the universal TM $U(x, z)$, and let λ_U be the constant associated with it. For any $\lambda > \lambda_U$, and every $S(\lambda)$-size circuit C, by setting the auxiliary input $z = C$, the above definition implies that the distinguishing gap by C is at most $1/S(\lambda)$. Therefore, λ_U is effectively the universal constant that works for all (circuit) distinguisher.

Below, we state definitions of cryptographic primitives using $(\lambda_0, S(\cdot))$ indistinguishability. Traditional polynomial or sub-exponential security can be directly derived from such more concrete definitions as follows:

Definition 15 (Polynomial Indistinguishability)**.** *A pair of ensembles $\{X_\lambda\}$, $\{Y_\lambda\}$ are polynomially indistinguishable if for every polynomial $p(\cdot)$, there is a constant $\lambda_p \in N$, such that, the two ensembles are $(\lambda_p, p(\cdot))$-indistinguishable.*

Definition 16 (Sub-exponential Indistinguishability)**.** *A pair of ensembles* $\{X_\lambda\}$, $\{Y_\lambda\}$ *are sub-exponentially indistinguishable, if there is a sub-exponential function* $S(\lambda) = 2^{\lambda^\varepsilon}$ *with* $\varepsilon \in (0,1)$ *and a constant* $\lambda_0 \in N$*, such that, the two ensembles are* $(\lambda_0, S(\cdot))$*-indistinguishable.*

4.2.2 Standard cryptographic primitives

Definition 17 (Pseudorandom Generator)**.** *A deterministic PT uniform machine* PRG *is a pseudorandom generator if the following conditions are satisfied:*

Syntax *For every* $\lambda, \lambda' \in \mathbb{N}$ *and every* $r \in \{0,1\}^\lambda$, $\mathsf{PRG}(r, \lambda')$ *outputs* $r' \in \{0,1\}^{\lambda'}$

$(\lambda_0, S(\cdot))$**-Security** *For every function* $p(\cdot)$*, such that,* $p(\lambda) \leq S(\lambda)$ *for all* λ*, the following ensembles are* $(\lambda_0, S(\cdot))$ *indistinguishable*

$$\left\{ r \xleftarrow{\$} \{0,1\}^\lambda : \mathsf{PRG}(r, p(\lambda)) \right\} \left\{ r' \xleftarrow{\$} \{0,1\}^{p(\lambda)} \right\}$$

4.2.3 Indistinguishability Obfuscation

In this section, we recall the definition of indistinguishability obfuscation for Turing machines from [17, 42, 2]. Following [42], we consider two notions of obfuscation for Turing machines. The first definition, called *bounded-input* indistinguishability obfuscation, only requires the obfuscated program to work for inputs of *bounded length* and furthermore the size of the obfuscated program may depend polynomially on this input length bound. (This is the notion achieved in [31, 54, 112] assuming subexponentially-secure $i\mathcal{O}$ for circuits and one-way functions.)

The second notion considered in [42] is stronger and requires the obfuscated program to work on any arbitrary polynomial length input (and the size of the obfuscated machine thus only depends on the program size and security parameter). We refer to this notion as *unbounded-input* indistinguishability obfuscation. (This stronger notion of unbounded-input indistinguishability obfuscator for Turing machines is only known to be achievable based on strong "extractability assumptions"—namely, (public-coin) differing-input obfuscation for circuits and (public-coin) SNARKs [42, 2, 105], variants of which are known to be implausible [28, 72, 32]).

Definition 18 (Indistinguishability Obfuscator (*iO*) for a class of Turing machines)**.** *An indistinguishability obfuscator for a class of Turing machines* $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ *is a uniform machine that behaves as follows:*

$\hat{\Pi} \leftarrow i\mathcal{O}(1^\lambda, \Pi, T)$*: iO takes as input a security parameter* 1^λ*, the Turing machine to obfuscate* $\Pi \in \mathcal{M}_\lambda$ *and a time bound* T *for* Π*. It outputs a Turing machine* $\hat{\Pi}$*.*

We require the following conditions to hold.

Correctness: *For every* $\lambda \in N$, $\Pi_\lambda \in \mathcal{M}_\lambda$*, input* x_λ *and time bound* T_λ*,*

$$\Pr[(\tilde{\Pi} \xleftarrow{\$} i\mathcal{O}(1^\lambda, \Pi_\lambda, T_\lambda) : \ \tilde{\Pi}(x_\lambda) = \Pi^T(x_\lambda)] = 1 \ .$$

Efficiency: *The running times of iO and* $\hat{\Pi}$ *are bounded as follows:*

There exists polynomial p *such that for every security parameter* λ*, Turing machine* $\Pi \in \mathcal{M}_\lambda$*, time bound* T *and every obfuscated machine* $\hat{\Pi} \leftarrow i\mathcal{O}(1^\lambda, \Pi, T)$ *and input* x*, we have that*

$$\mathsf{Time}_{i\mathcal{O}}(1^\lambda, \Pi, T) \leq p(\lambda, |\Pi|, \log T)$$

$$\mathsf{Time}_{\hat{\Pi}}(x) \leq p(\lambda, |\Pi|, |x|, T)$$

$(\lambda_0, S(\cdot))$**-Security:** *For every ensemble of pairs of Turing machines and time bounds* $\{\Pi_{0,\lambda}, \Pi_{1,\lambda}, T_\lambda\}$ *where for every* $\lambda \in \mathbb{N}$, $\Pi_0 = \Pi_{0,\lambda}$, $\Pi_1 = \Pi_{1,\lambda}$, $T = T_\lambda$, *satisfying the following*

$$\Pi_0, \Pi_1 \in \mathcal{M}_\lambda \quad |\Pi_0| = |\Pi_1| \leq \text{poly}(\lambda) \quad T \leq \text{poly}(\lambda)$$
$$\forall x, \Pi_0^T(x) = \Pi_1^T(x) \,,$$

the following ensembles are $(\lambda_0, S(\cdot))$*-indistinguishable*

$$\left\{i\mathcal{O}(1^\lambda, \Pi_{0,\lambda}, T_\lambda)\right\} \left\{i\mathcal{O}(1^\lambda, \Pi_{1,\lambda}, T_\lambda)\right\} \,.$$

Definition 19 (Unbounded-input indistinguishability obfuscator for Turing machines)**.** *An unbounded-input indistinguishability obfuscator for Turing machines* $i\mathcal{O}(\cdot,\cdot,\cdot)$ *is simply an indistinguishability obfuscator for the class of all Boolean Turing machines.*

Remark 7 (Obfuscation for Boolean Turing machines is without loss of generality)**.** *The above definition is equivalent to one that considers the class of all Turing machines. Any Turing machine with output length* m *can be represented as a Boolean Turing machine that takes in an additional input* $i \in [m]$ *and returns the* i^{th} *bit of the* m*-bit long output.*

Definition 20 (Bounded-input indistinguishability obfuscator for Turing machines)**.** *A bounded-input indistinguishability obfuscator for Turing machines* $i\mathcal{O}(\cdot,\cdot,\cdot,\cdot)$ *is a uniform machine such that for every polynomial* p, $i\mathcal{O}(p,\cdot,\cdot,\cdot)$ *is an indistinguishability obfuscator for the class of Turing machines* $\{\mathcal{M}_\lambda\}$ *where* $\mathcal{M}_\lambda$ *are machines that accept only inputs of length* $p(\lambda)$*. Additionally,* $i\mathcal{O}(p, 1^\lambda, \Pi, T)$ *is allowed to run in time* $\text{poly}(p(\lambda) + \lambda + |\Pi| + \log T)$*.*

Finally, we define weaker variants of the above definitions where the size of the obfuscated program is sub-linear (instead of poly-logarithmic) in the time bound.

Definition 21 (Sub-linear efficiency for Indistinguishability Obfuscators)**.** *We say an indistinguishability obfuscator $i\mathcal{O}$ for a class of Turing machines $\{\mathcal{M}_\lambda\}$ has sub-linear efficiency if it satisfies the requirements in Definition 18 with the efficiency requirement replaced with the following.*

Efficiency: *The running times of $i\mathcal{O}$ and $\hat{\Pi}$ are bounded as follows:*

There exists polynomial p and constant $\epsilon > 0$ such that for every security parameter λ, Turing machine $\Pi \in \mathcal{M}_\lambda$, time bound T and every obfuscated machine $\hat{\Pi} \leftarrow i\mathcal{O}(1^\lambda, \Pi, T)$ and input x, we have that

$$\mathsf{Time}_{i\mathcal{O}}(1^\lambda, \Pi, T) \leq p(\lambda, |\Pi|)T^{1-\epsilon}$$

$$\mathsf{Time}_{\hat{\Pi}}(x) \leq p(\lambda, |\Pi|, |x|, T)$$

4.2.4 Functional Encryption

Definition 22 (Selectively-secure Single-Query Public-key Functional Encryption)**.** *A tuple of PPT algorithms* $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *is a selectively-secure functional encryption scheme for a class of circuits $\{C_\lambda\}$ if it satisfies the following properties.*

Completeness *For every $\lambda \in \mathbb{N}$, $C \in C_\lambda$ and message $m \in \{0,1\}^*$,*

$$\Pr\left[\begin{array}{ll} (mpk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda) & \\ c \leftarrow \mathsf{FE.Enc}(1^\lambda, m) & : C(m) \leftarrow \mathsf{FE.Dec}(sk_C, c) \\ sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C) & \end{array}\right] = 1$$

$(\lambda_0, S(\cdot))$**-Selective-security** *For every ensemble of circuits and pair of messages $\{C_\lambda, m_{0,\lambda}, m_{1,\lambda}\}$ where $C_\lambda \in C_\lambda$, $|C_\lambda|, |m_{0,\lambda}|, |m_{1,\lambda}| \leq \mathrm{poly}(\lambda)$, and $C_\lambda(m_{0,\lambda}) =$*

$C_\lambda(m_{1,\lambda})$, the following ensembles of distributions $\{D_{0,\lambda}\}$ and $\{D_{1,\lambda}\}$ are $(\lambda_0, S(\cdot))$-indistinguishable.

$$D_{b,\lambda} = \left(\begin{array}{ll} (mpk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda) & \\ \quad c \leftarrow \mathsf{FE.Enc}(1^\lambda, m_{b,\lambda}) & : mpk, c, sk_C \\ sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C_\lambda) & \end{array} \right)$$

We note that in this work, we only need the security of the functional encryption scheme to hold with respect to statically chosen challenge messages and functions.

Definition 23 (Compact Functional Encryption)**.** *We say a functional encryption scheme is* compact *if it additionally satisfies the following requirement:*

Compactness *The running time of* FE.Enc *is bounded as follows.*

There exists a polynomial p such that for every security parameter $\lambda \in \mathbb{N}$ and message $m \in \{0,1\}^*$, $\mathsf{Time}_{\mathsf{FE.Enc}}(1^\lambda, m) \leq p(\lambda, |m|, polylog(s))$, where $s = \max_{C \in C_\lambda} |C|$.

Furthermore, we say the functional encryption scheme has sub-linear compactness *if there exists a polynomial p and constant $\epsilon > 0$ such that for every security parameter $\lambda \in \mathbb{N}$ and message $m \in \{0,1\}^*$, $\mathsf{Time}_{\mathsf{FE.Enc}}(1^\lambda, m) \leq p(\lambda, |m|)s^{1-\epsilon}$.*

We also define a notion of succinctness, as follows:

Definition 24 (Succinct Functional Encryption)**.** *A compact functional encryption scheme for a class of circuits that output only a* single bit *is called a* succinct *functional encryption scheme.*

Theorem 22 ([93])**.** *Assuming (sub-exponentially secure) LWE, there exists a (sub-exponentially secure) succinct functional encryption scheme for* NC^1*.*

We note that [93] do not explicitly consider sub-exponentially secure succinct functional encryption, but their construction satisfies it (assuming sub-exponentially secure LWE).

Theorem 23 ([93, 3])**.** *Assuming the existence of symmetric-key encryption with decryption in* NC^1 *(resp. sub-exponentially secure) and succinct FE for* NC^1 *(resp. sub-exponentially secure), there exists succinct FE for* $\mathsf{P/poly}$ *(resp. sub-exponentially secure).*

We also consider an even weaker notion of sublinear-compactness, where only the ciphertext size is sublinear in the size bound s of the function being evaluation, but the encryption time can depend polynomially on s.

Definition 25 (Weakly Sublinear Compact Functional Encryption)**.** *We say a functional encryption scheme for a class of circuits* $\{C_\lambda\}$ *is* weakly sublinear compact *if there exists* $\epsilon > 0$ *such that for every* $\lambda \in \mathbb{N}$*,* $pk \leftarrow \mathsf{FE.Setup}(1^\lambda)$ *and* $m \in \{0,1\}^*$ *we have that*

$$\mathsf{Time}_{\mathsf{FE.Enc}}(pk, m) = \mathrm{poly}(\lambda, |m|, s)$$

$$\mathsf{outlen}_{\mathsf{FE.Enc}}(pk, m) = s^{1-\epsilon} \cdot \mathrm{poly}(\lambda, |m|)$$

where $s = \max_{C \in \mathcal{C}_\lambda} |C|$.

4.3 Randomized Encoding Schemes

Roughly speaking, randomized encoding schemes encodes a computation of a program Π on an input x, into an encoded computation $(\hat{\Pi}, \hat{x})$, with the following two properties: First, the encoded computation evaluates to the same output

$\Pi(x)$, while leaking no other information about Π and x. Second, the encoding is "simpler" to compute than the original computation. In the literature, different measures of simplicity have been considered. For instance, in the original works by [102, 10], the depth of computation is used and it was shown that any computation in P can be encoded in NC_1 using Yao's garbled circuits [138]. A recent line of works [31, 54, 112] uses the time-complexity as the measure and show that any *Boolean* Turing machine computation can be encoded in time poly-logarithmic in the run-time of the computation.

Traditionally, the security of randomized encoding schemes are capture via simulation. In this work, we consider a new *distributional* indistinguishability-based security notion, and show that it is implied by the transitional simulation security. Additionally, we further explore how compact the encoded computation can be: Similar to the recent works [31, 54, 112], we consider encoding whose size depends poly-logarithmically on the run-time of the encoded computation; but differently, we directly consider Turing machines with arbitrary length outputs, and require the size of the encoding to be independent of the output length. Such scheme is called a compact randomized encoding scheme.

4.3.1 Randomized Encoding with Simulation Security

In this section, we recall the traditional definition of randomized encoding with simulation security [102, 10].

Definition 26 (Randomized Encoding Scheme for a Class of Turing Machines)**.** *A Randomized Encoding scheme* RE *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *consists of two algorithms,*

- $(\hat{\Pi}, \hat{x}) \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \Pi, x, T)$*: On input a security parameter* 1^λ*, Turing machine* $\Pi \in \mathcal{M}_\lambda$*, input* x *and time bound* T*,* Enc *generates an encoded machine* $\hat{\Pi}$ *and encoded input* $\hat{x}$*.*
- $y = \mathsf{Eval}(\hat{\Pi}, \hat{x})$*: On input* $(\hat{\Pi}, \hat{x})$ *produced by* Enc, Eval *outputs* y*.*

Correctness: *The two algorithms* Enc *and* Eval *satisfy the following correctness condition: For all security parameters* $\lambda \in \mathbb{N}$*, Turing machines* $\Pi \in \mathcal{M}_\lambda$*, inputs* x *and time bounds* T*, it holds that,*

$$\Pr[(\hat{\Pi}, \hat{x}) \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \Pi, x, T) : \ \mathsf{Eval}(\hat{\Pi}, \hat{x}) = \Pi^T(x))] = 1$$

Definition 27 ($(\lambda_0, S(\cdot))$-Simulation Security)**.** *A randomized encoding scheme* RE *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *satisfies* $(\lambda_0, S(\cdot))$**-simulation security**, *if there exists a PPT algorithm* Sim *and a constant* c*, such that, for every polynomial* B*, and ensemble* $\{\Pi_\lambda, x_\lambda, T_\lambda\}$ *where* $\Pi_\lambda \in \mathcal{M}_\lambda$ *and* $|\Pi_\lambda|, |x_\lambda|, T_\lambda \leq B(\lambda)$*, the following ensembles are* $(\lambda_0, S'(\lambda))$ *indistinguishable with* $S'(\lambda) = S(\lambda) - B(\lambda)^d$ *for all* $\lambda \in N$*.*

$$\left\{(\hat{\Pi}, \hat{x}) \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \Pi, x, T) : \hat{\Pi}, \hat{x}\right\}$$
$$\left\{(\hat{\Pi}, \hat{x}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \Pi^T(x), 1^{|\Pi|}, 1^{|x|}, T) : \hat{\Pi}, \hat{x}\right\}$$

where $\Pi = \Pi_\lambda$*,* $x = x_\lambda$*, and* $T = T_\lambda$*.*

A recent line of works [31, 54, 112] constructed randomized encoding with polynomial simulation security (i.e., the simulation is polynomially indistinguishable to the honest encoding in the above definition) for the class of *Boolean* Turing machines, where the time complexity of encoding is independent of the run-time of the Turing machine.

Theorem 24 (Simulation-Based Randomized Encoding for Boolean Turing Machines [112])**.** *Assuming the existence of indistinguishability obfuscation for circuits*

and injective pseudo-random generators, there is a randomized encoding scheme $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$ *satisfying polynomial simulation security for the class of* Boolean Turing machines*, with the following efficiency:*

- *For every security parameter* λ*, Boolean Turing machine* Π*, input* x*, time bound* T *and every encoded pair* $(\hat{\Pi}, \hat{x}) \leftarrow \mathsf{Enc}(1^\lambda, \Pi, x, T)$*, we have that*

$$\mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi, x, T) = \mathrm{poly}(\lambda, |\Pi|, |x|, \log T)$$

$$\mathsf{Time}_{\mathsf{Eval}}(\hat{\Pi}, \hat{x}) = \mathrm{poly}(\lambda, |\Pi|, |x|, T)$$

In this paper, we consider the class of *all* Turing machines, including ones with arbitrarily long outputs. One can obtain a randomized encoding for Turing machines with ℓ-bit outputs, by encoding a collection of ℓ Turning machines each outputting one output bit, using a randomized encoding for Boolean Turing machines. It yields a scheme whose encoding time, as well as the size of encoding, scales linearly with the output length (and still poly-logarithmically with the run-time of the encoded computation). As we show later in the paper, this essentially is tight, meaning that there is certain computation (namely, evaluating pseudo-random generators) for which the size of the randomized encoding cannot be sub-linear in the output length. In other words, when simulation security is required, encoding has to be as long as the output length in general.

4.3.2 Distributional Indistinguishability Security

In this paper, we study randomized encoding for all Turing machine computation, whose encoding size is independent of the output length of the

computation—we say such randomized encoding schemes are **compact**. Towards this, we must consider weaker security notions than simulation security, and indistinguishability-based security notions are natural candidates. One weaker notion that has been considered in the literature requires encoding of two computation, (Π_1, x_1) and (Π_2, x_2) with the same output $\Pi_1(x_1) = \Pi_2(x_2)$, to be indistinguishable. In this work, we generalize this notion to, what called *distributional* indistinguishability security—this notion requires encoding of computations sampled from two distributions, $(\Pi_1, x_1) \xleftarrow{\$} D_1$ and $(\Pi_2, x_2) \xleftarrow{\$} D_2$, to be indistinguishable, provided that their outputs are indistinguishable.

Definition 28 (Distributional $(\lambda_0, S(\cdot))$-Indistinguishability Security)**.** *A randomized encoding scheme* RE *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *satisfies* distributional $(\lambda_0, S(\cdot))$-indistinguishability security, *(or* $(\lambda_0, S(\cdot))$*-ind-security for short) if the following is true w.r.t. some constant* $c > 0$*:*

For every ensembles of distributions $\{D_{0,\lambda}\}$ *and* $\{D_{1,\lambda}\}$ *with the following property:*

1. *there exists a polynomial* B*, such that, for every* $b \in \{0, 1\}$*,* $D_{b,\lambda}$ *is a distribution over tuples of the form* (Π_b, x_b, T_b)*, where* Π_b *is a Turing machine,* x_b *is an input and* T_b *is a time bound, and* $\lambda, |\Pi_b|, |x_b|, T_b \leq B(\lambda)$*.*
2. *there exist an integer* $\lambda_0' \geq \lambda_0$*, and a function* S' *with* $S'(\lambda) \leq S(\lambda)$ *for all* λ*, such that, the following ensembles of output distributions are* $(\lambda_0', S'(\cdot))$*-indistinguishable,*

$$\left\{(\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{0,\lambda} : \Pi_0^{T_0}(x_0), T_0, |\Pi_0|, |x_0|\right\}$$
$$\left\{(\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \Pi_1^{T_1}(x_1), T_1, |\Pi_1|, |x_1|\right\}$$

the following ensembles of encoding is $(\lambda_0', S''(\cdot))$*-indistinguishable, where* $S''(\lambda) =$

$\frac{S'(\lambda)}{\lambda^c} - B(\lambda)^c$.

$$\left\{(\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{0,\lambda} : \mathsf{Enc}(1^\lambda, \Pi_0, x_0, T_0)\right\}$$
$$\left\{(\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \mathsf{Enc}(1^\lambda, \Pi_1, x_1, T_1)\right\}$$

For convenience, in the rest of the paper, we directly refer to distributional indistinguishability security as indistinguishability security. The above concrete security directly gives the standard polynomial and sub-exponential security.

Definition 29 (Polynomial and Sub-exponential Indistinguishability Security)**.** *A randomized encoding scheme* RE *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *satisfies* polynomial ind-security *, if it satisfies* $(\lambda_p, p(\cdot))$*-indistinguishability security for every polynomial* p *and some* $\lambda_p \in N$*. Furthermore, it satisfies sub-exponential ind-security if it satisfies* $(\lambda_0, S(\cdot))$*-indistinguishability security for* $S(\lambda) = 2^{\lambda^\varepsilon}$ *with some* $\varepsilon \in (0, 1)$*.*

We note that, by definition, it holds that any randomized encoding scheme that is $(\lambda_0, S(\cdot))$-ind-secure, is also $(\lambda_0', S'(\cdot))$-ind-secure for any $\lambda_0' \geq \lambda_0$ and S' s.t. $S'(\lambda) \leq S(\lambda)$ for every λ. Therefore, naturally, sub-exponential ind-security is stronger than polynomial ind-security.

Later, in Section 4.3.4 , we show that RE schemes with ind-security are composable just as RE schemes with simulation security are. Additionally, in Section 4.3.5, we show that RE schemes satisfying simulation security also satisfy ind-security.

4.3.3 Compactness and Sublinear Compactness

With indistinguishability-security, we now define compact randomized encoding schemes for all Turing machines, whose time-complexity of encoding is in-

dependent of the output length.

Definition 30 (Compact Randomized Encoding for Turing machines)**.** *A* $(\lambda_0, S(\cdot))$*-ind-secure* compact randomized encoding *scheme for Turing machines, is a randomized encoding scheme with* $(\lambda_0, S(\cdot))$*-indistinguishability security for the class of all Turing machines, with the following efficiency:*

- *For every security parameter* λ*, Turing machine* Π*, input* x*, time bound* T *and every encoded pair* $(\hat{\Pi}, \hat{x}) \leftarrow \mathsf{Enc}(1^\lambda, \Pi, x, T)$*, it holds*

$$\mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi, x, T) = \mathrm{poly}(\lambda, |\Pi|, |x|, \log T)$$

$$\mathsf{Time}_{\mathsf{Eval}}(\hat{\Pi}, \hat{x}) = \mathrm{poly}(\lambda, |\Pi|, |x|, T)$$

In this work, we also consider a weaker variant of the above compactness requirement, where the encoding time is sub-linear (instead of poly-logarithmic) in the computation time. For our results a compact randomized encoding scheme with sub-linear efficiency will suffice.

Definition 31 (Sub-linear Compactness of Randomized Encoding schemes)**.** *We say a randomized encoding scheme* $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$ *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *has sub-linear compactness if the efficiency requirement on* Enc *in Definition 30 is relaxed to: For some constant* $\varepsilon \in (0, 1)$*,*

$$\mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi, x, T) \leq \mathrm{poly}(\lambda, |\Pi|, |x|) \cdot T^{1-\epsilon}$$

4.3.4 Composition of Ind-Security

It was shown in [103, 10] that randomized encoding schemes with simulation security are composable. We show that the same holds for indistinguishability

security. Let $RE_1 = (\mathsf{Enc}_1, \mathsf{Eval}_1)$ be a randomized encoding scheme for a class of Turing machines $\{\mathcal{M}_\lambda\}$, $\mathsf{RE}_2 = (\mathsf{Enc}_2, \mathsf{Eval}_2)$ a randomized encoding for an appropriate class of Turing machines that includes the ensembles of functions $\{G[\lambda, \Pi, T, s]\}$ defined below, and PRG a pseudo-random generator. Consider the following composed randomized encoding scheme $(\mathsf{Enc}, \mathsf{Eval})$:

$$\begin{aligned}
\mathsf{Enc}(1^\lambda, \Pi, x, T) \quad &: \quad (\hat{G}, \hat{x}) \xleftarrow{\$} \mathsf{Enc}_2(1^\lambda, G, x, T_G(x)) \\
&\qquad \text{where } G[\lambda, \Pi, T, s](x) = \mathsf{Enc}_1(1^\lambda, \Pi, x, T \ ; \ \mathsf{PRG}(s)) \\
\mathsf{Eval}(\hat{G}, \hat{x}) \quad &: \quad (\hat{\Pi}, \hat{x}') = \mathsf{Eval}_2(\hat{G}, \hat{x}), \ y = \mathsf{Eval}_1(\hat{\Pi}, \hat{x}')
\end{aligned}$$

where $T_G(x)$ is an upper bound on the run-time of G on input x, it can be efficiently calculated using an upper bound on the run-time of Enc_1 and PRG, in particular, $T_G(x) = \mathrm{poly}(\mathsf{Time}_{\mathsf{Enc}_1}(1^\lambda, \Pi, x, T))$.

Lemma 3 (Composition)**.** *If* RE_1 *is* $(\lambda_1, S_1(\cdot))$*-ind-secure, and* RE_2 *and* PRG *are* $(\lambda_2, S_2(\cdot))$*-ind-secure, with* $\lambda_2 \leq \lambda_1$ *and* $S_2(\lambda) \geq S_1(\lambda)$*. then* $(\mathsf{Enc}, \mathsf{Eval})$ *is* $(\lambda_1, S_1(\cdot))$*-ind-secure.*

Proof. Let c_1 be the constant w.r.t. which the ind-security of RE_1 holds, and c_2 the constant for RE_2. We show that the composed scheme $(\mathsf{Enc}, \mathsf{Eval})$ is $(\lambda_1, S_1(\cdot))$-ind-secure w.r.t. some sufficiently large constant c, whose value would become clear in the proof below.

To show this, consider any pair of ensembles of distributions $\{D_{0,\lambda}\}$ and $\{D_{1,\lambda}\}$ that are (λ_0', S')-indistinguishability for $\lambda_0' \geq \lambda_1$ and $S'(\lambda) \leq S_1(\lambda)$ for all λ, and all parameters $\lambda, |\Pi_b|, |x_b|, T_b \leq B(\lambda)$ for some polynomial B. We need to show that encodings generated using Enc are (λ_0', S'')-indistinguishable, where $S''(\lambda) \leq S'(\lambda)/\lambda^c - B(\lambda)^c$.

First, it follows directly from the indistinguishability security of RE_1 that the distributions H_1 and H_2 of encodings produced by Enc_1 are (λ_0', S_1)-indistinguishable, where $S_H(\lambda) = S'(\lambda)/\lambda^{c_1} - B(\lambda)^{c_1}$,

$$H_1 = \left\{(\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{0,\lambda} : \mathsf{Enc}_1(1^\lambda, \Pi_0, x_0, T_0)\right\}$$
$$H_2 = \left\{(\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \mathsf{Enc}_1(1^\lambda, \Pi_1, x_1, T_1)\right\}$$

Consider two modified distributions, where the encodings are generated using pseudo-random coins.

$$H_1^+ = \left\{s \xleftarrow{\$} \{0,1\}^\lambda,\ (\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{0,\lambda} : \mathsf{Enc}_1(1^\lambda, \Pi_0, x_0, T_0; \mathsf{PRG}(s))\right\}$$
$$H_2^+ = \left\{s \xleftarrow{\$} \{0,1\}^\lambda,\ (\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \mathsf{Enc}_1(1^\lambda, \Pi_1, x_1, T_1; \mathsf{PRG}(s))\right\}$$

Since $\lambda, |\Pi_b|, |x_b|, T_b \le B(\lambda)$, $\mathsf{Time}_{\mathsf{Enc}_1}(1^\lambda, \Pi_b, x_b, T_b)$ is bounded by $B(\lambda)^d$ for some constant d. Then, by the (λ_2, S_2)-indistinguishability of PRG, it follows that for every b no adversary of size $\tilde{S}_H(\lambda) = S_2(\lambda) - B(\lambda)^d$ can distinguish H_b^+ and H_b with probability larger than $1/S_2(\lambda)$. Therefore, (by a hybrid argument,) no adversary of size $\min(S_H(\lambda), \tilde{S}_H(\lambda))$ can distinguish H_1^+ and H_2^+ with probability larger than $2/\tilde{S}_H(\lambda) + 1/S_H(\lambda)$. Since $S'(\lambda) \le S_1(\lambda) \le S_2(\lambda)$, there is a $S_H^+(\lambda) = S'(\lambda)/\lambda^e - B(\lambda)^e$ with a sufficiently large constant e, such that, $\left\{H_1^+\right\}$ and $\left\{H_2^+\right\}$ are (λ_0', S_H^+)-indistinguishable.

Consider the following ensembles of distributions $\{\mathcal{E}_{0,\lambda}\}$ and $\{\mathcal{E}_{1,\lambda}\}$:

$$\mathcal{E}_{b,\lambda} \ : \ s \xleftarrow{\$} \{0,1\}^\lambda,\ (\Pi_b, x_b, T_b) \xleftarrow{\$} \mathcal{D}_{b,\lambda} \text{ output } (G_b, x_b, T_b')$$
$$\text{where } G_b[\lambda, \Pi_b, T_b, s](x_b) = \mathsf{Enc}_1(1^\lambda, \Pi_b, x_b, T_b\ ;\ \mathsf{PRG}(s)),\ T_b' = T_{G_b}(x_b).$$

where $T_{G_b}(x_b) = \mathrm{poly}(\lambda, |\Pi_b|, |x_b|, \log T_b)$, determined by the run time of algorithms PRG and Enc_1. By the indistinguishability of H_1^+ and H_2^+, the output distributions of $\mathcal{E}_{0,\lambda}$ and $\mathcal{E}_{1,\lambda}$ are also (λ_0', S_H^+)-indistinguishable. Moreover, all parameters T_b', $|G_b|$, $|x_b|$ are bounded by $B(\lambda)^k$ for some constant k.

Therefore, it follows from the (λ_2, S_2)-security of RE_2 (and the fact that $\lambda_2 \leq \lambda_1 \leq \lambda_0'$, and $S_H^+(\lambda) \leq S_2(\lambda)$) that, encoding of G_0 and G_1 sampled from $\mathcal{E}_{0,\lambda}$ and $\mathcal{E}_{1,\lambda}$ are (λ_0', S_3'')-indistinguishable, where $S''(\lambda) = S_H^+(\lambda)/\lambda^{c_2} - B(\lambda)^{kc_2} \geq S'(\lambda)/\lambda^c - B(\lambda)^c$ for a sufficiently large c.

$$\left\{(G_0, x_0, T_0') \xleftarrow{\$} \mathcal{E}_{0,\lambda} : \mathsf{Enc}_2(1^\lambda, G_0, x_0, T_0')\right\}$$
$$\left\{(G_1, x_1, T_1') \xleftarrow{\$} \mathcal{E}_{1,\lambda} : \mathsf{Enc}_2(1^\lambda, G_1, x_1, T_1')\right\}$$

This concludes the proof. □

The above composition lemma implies that if there is a sublinear RE with complexity scaling with T^β, one can reduce the complexity by an arbitrary polynomial factor by recursively composing the RE with itself. This fact with be very instrumental later. More precisely,

Lemma 4 (Recursive Composition)**.** *Let α and β be any constants satisfying $0 < \alpha < \beta < 1$. If there is a sublinear RE $(\mathsf{Enc}', \mathsf{Eval}')$ with time complexity*

$$\mathsf{Time}_{\mathsf{Enc}'}(1^\lambda, \Pi, x, T) \leq \mathrm{poly}(\lambda, |\Pi|, |x|)T^\beta\,,$$

then, there is a sublinear RE $(\mathsf{Enc}, \mathsf{Eval})$ with time complexity

$$\mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi, x, T) \leq \mathrm{poly}(\lambda, |\Pi|, |x|)T^\alpha\,.$$

Proof of Claim 4. Given any sublinear RE $\mathsf{RE}' = (\mathsf{Enc}', \mathsf{Eval}')$ with time complexity

$$\mathsf{Time}_{\mathsf{Enc}'}(1^\lambda, \Pi, x, T) \leq \mathrm{poly}(\lambda, |\Pi|, |x|)T^\beta$$

We show how to construct a sublinear RE $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$ with time complexity

$$\mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi, x, T) \leq \mathrm{poly}(\lambda, |\Pi|, |x|)T^\alpha$$

The new scheme RE is constructed by "composing" RE' iteratively for a sufficiently large, constant, number of times, depending on α and β.

COMPOSING ONCE: Recall that in Section 4.3.4, it was shown that given two ind-secure RE schemes for Turing machines, one can compose them into a new RE scheme that still satisfies indistinguishability security, as shown in Lemma 3. It follows from this lemma that by composing the scheme RE' with itself, we obtain a new scheme RE_1 as follows.

$$G[\lambda, \Pi, T, s](x) = \mathsf{Enc}'(1^\lambda, \Pi, x, T \; ; \; \mathsf{PRG}(s))$$

$$\mathsf{Enc}_1(1^\lambda, \Pi, x, T) \; : \; (\hat{G}, \hat{x}) \xleftarrow{\$} \mathsf{Enc}'(1^\lambda, G, x, T_G(x))$$

where $T_G(x)$ is an upper bound on the run-time of $G(x)$. We show that RE_1 is more "compact" than RE' – with a time complexity depending on T^{β^2} as opposed to T^β.

$$T_G(x) = T_{\mathsf{Enc}'}(1^\lambda, \Pi, x, T) \leq \text{poly}(\lambda, |\Pi|, |x|)T^\beta$$

$$T_{\mathsf{Enc}_1}(1^\lambda, \Pi, x, T) = T_{\mathsf{Enc}'}(1^\lambda, G, x, T_G(x)) \leq \text{poly}(\lambda, |\Pi|, |x|)T^{\beta^2}$$

COMPOSING k TIMES: Since RE_1 itself is a sublinear RE scheme, one can apply the same composition technique on it to obtain a new scheme RE_2 whose time complexity depends on $T^{(\beta^2)^2}$. More generally, applying the composition technique recursively for a constant number d of times yields a scheme RE_d with time complexity

$$T_{\mathsf{Enc}_d}(1^\lambda, \Pi, x, T) = \text{poly}(\lambda, |\Pi|, |x|)T^{\beta^e}, \quad \text{for } e = 2^d \; .$$

For a sufficiently large d, $T_{\mathsf{Enc}_d} \leq \text{poly}(\lambda, |\Pi|, |x|)T^\alpha$, which concludes the claim. □

4.3.5 Simulation Security implies Indistinguishability Security

In this section, we show that simulation security for a randomized encoding scheme implies indistinguishability security for the same scheme. More formally,

Theorem 25. *Let* RE *be a* (λ_0, S)*-simulation-secure randomized encoding scheme for a sufficiently large* λ_0*. Then* RE *is also* (λ_0, S)*-ind-secure.*

Proof. Let Sim be the PPT simulator of $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$; let c be a sufficiently large constant whose value will become clear in the proof below. To show that RE is (λ_0, S)-ind-secure w.r.t. constant c, consider arbitrary ensembles of distributions $\{D_{0,\lambda}\}$ and $\{D_{1,\lambda}\}$ whose output distributions below are (λ_0', S')-indistinguishability for $\lambda_0' \geq \lambda_0$ and $S'(\lambda) \leq S(\lambda)$ for all λ,

$$O_0 = \left((\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{0,\lambda} : \Pi_0^{T_0}(x_0), T_0, |\Pi_0|, |x_0|\right)$$
$$O_1 = \left((\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \Pi_1^{T_1}(x_1), T_1, |\Pi_1|, |x_1|\right)$$

and all parameters $\lambda, |\Pi_b|, |x_b|, T_b \leq B(\lambda)$ for some polynomial B. We show that encoding generated using Enc are (λ_0', S'')-indistinguishable, where $S''(\lambda) \leq S'(\lambda)/\lambda^c - B(\lambda)^c$.

Consider the following sequence of hybrids:

H_0**:**

$$\left((\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{0,\lambda} : \mathsf{Enc}(1^\lambda, \Pi_0, x_0, T_0)\right)$$

H_1**:**

$$\left((\Pi_0, x_0, T_0) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \mathsf{Sim}(1^\lambda, \Pi_0^{T_0}(x_0), 1^{|\Pi_0|}, 1^{|x_0|}, T_0)\right)$$

H_2:

$$\left((\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \mathsf{Sim}(1^\lambda, \Pi_1^{T_1}(x_1), 1^{|\Pi_1|}, 1^{|x_1|}, T_1)\right)$$

H_3:

$$\left((\Pi_1, x_1, T_1) \xleftarrow{\$} \mathcal{D}_{1,\lambda} : \mathsf{Enc}(1^\lambda, \Pi_1, x_1, T_1)\right)$$

In other words, our goal is to show that $\{H_0\}$ and $\{H_3\}$ are (λ_0', S'')-indistinguishable. Towards this, we argue the indistinguishability of every two neighboring hybrids and use a hybrid argument to conclude.

First, it follows directly from the (λ_0, S)-simulation security of RE (and the fact that $\lambda_0' \geq \lambda_0$) that $\{H_0\}$ and $\{H_1\}$, as well as $\{H_2\}$ and $\{H_3\}$, are $(\lambda_0', S_{0,1})$-indistinguishable with $S_{0,1}(\lambda) = S(\lambda) - B(\lambda)^d$ for some constant d.

Next, to argue the indistinguishability of H_1 and H_2, we first note that the run-time of the simulator $T_{\mathsf{Sim}} = \mathsf{Time}_{\mathsf{Sim}}(1^\lambda, \Pi_b^{T_b}(x_b), 1^{|\Pi_b|}, 1^{|x_b|}, T_b) \leq B(\lambda)^c$ with a sufficiently large constant c determined by Sim, since the length of every input argument is bounded by $B(\lambda)$. Then, it follows from the (λ_0', S')-indistinguishability of the output distributions $\{O_0\}$, $\{O_1\}$ that for every adversary of size $S_{1,2}(\lambda) = S'(\lambda) - T_{\mathsf{Sim}} = S'(\lambda) - B(\lambda)^c$, the probability that it distinguishes H_1 and H_2 is bounded by $1/S'(\lambda)$ (as otherwise one can construct a $S'(\lambda)$-size adversary that distinguishes $\{O_1\}$ and $\{O_2\}$ with probability larger than $1/S'(\lambda)$, by internally running Sim to sample from H_1 or H_2, and then the distinguisher for H_1 and H_2 to distinguish).

Therefore, it follows from a hybrid argument that for every adversary of size $\min(S_{0,1}(\lambda), S_{1,2}(\lambda))$, the probability of distinguishing H_0 and H_3 is bounded by $2/S_{0,1}(\lambda) + 1/S'(\lambda)$. For sufficiently large $c > d$ and λ_0', $\{H_0\}$ and $\{H_3\}$ are (λ_0', S'')-indistinguishable. □

4.4 Unbounded-Input IO from Compact RE

In this section, we define our succinct indistinguishability obfuscator for Turing machines. Let $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$ be a compact randomized encoding scheme for Turing machines with sub-exponential indistinguishability security. Let c be the constant for the security loss associated with the indistinguishability security of RE. We assume without loss of generality that $\mathsf{Enc}(1^\lambda, \cdot, \cdot)$ requires a random tape of length λ. Let PRG be a sub-exponentially secure pseudorandom generator and let ϵ be the constant associated with the sub-exponential security of PRG.

For every $\lambda \in \mathbb{N}$, $D \leq 2^\lambda$, define

$$l(\lambda, -1) = \lambda$$

$$l(\lambda, D) = l(\lambda, D-1) + (2d\lambda)^{1/\epsilon}$$

where $d > 0$ is any constant strictly greater than c.

Construction 2. *Consider a Turing machine Π, security parameter $\lambda \in \mathbb{N}$, and time bound T of Π. For every partial input $s \in \{0,1\}^*$ with $|s| \leq 2^\lambda$ and $R \in \{0,1\}^{2l(\lambda,|s|)}$, we recursively define a Turing machine $\tilde{\Pi}_{s,R}$ to be as follows:*

When $|s| < 2^\lambda$:

On the empty input, $\tilde{\Pi}_{s,R}$ outputs:

$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \tilde{\Pi}_{s0,R_0}, T'(\lambda, |s|+1, |\Pi|, \log(T)); R_1)$$

$$\mathsf{Enc}(1^{l(\lambda,|s|+2)}, \tilde{\Pi}_{s1,R_2}, T'(\lambda, |s|+1, |\Pi|, \log(T)); R_3)$$

$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi, s, T; R_4)$$

where $(R_0, R_1, R_2, R_3, R_4) \leftarrow \mathsf{PRG}(R, 5 \cdot 2l(\lambda, |s| + 1))$ *and* T' *is some fixed polynomial in* $\lambda, |s| + 1, |\Pi|$ *and* $\log(T)$*. In the special case when* $|s| = 2^\lambda - 1$*, the time bound used in the first two encodings is set to* T*.*

On all other inputs, $\tilde{\Pi}_{s,R}$ *outputs* $\perp$*.*

When $|s| = 2^\lambda$**:**

On the empty input, $\tilde{\Pi}_{s,R}$ *outputs* $\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi, s, T; R)$*. On all other inputs,* $\tilde{\Pi}_{s,R}$ *outputs* $\perp$*.*

We define $T'(\cdot,\cdot,\cdot,\cdot)$ *(corresponding to the bound placed on the running time of* $\tilde{\Pi}_{s,R}$*) to be the smallest polynomial such that for all* λ*,* $s \in \{0,1\}^{\leq 2^\lambda}$*,* $R \in \{0,1\}^{2l(\lambda,|s|)}$*,* Π *and* T*,*

$$\begin{aligned} T'(\lambda, |s|, |\Pi|, \log(T)) \geq\ & p(\lambda_{|s|+1}, |\tilde{\Pi}_{s0,R}|, 0, \log(T'_{|s|+1})) \\ & + p(\lambda_{|s|+1}, |\tilde{\Pi}_{s1,R}|, 0, \log(T'_{|s|+1})) \\ & + p(\lambda_{|s|+1}, |\Pi|, |s|, \log(T)) \\ & + \mathsf{Time}_{\mathsf{PRG}}(R, 5 \cdot 2l(\lambda, |s| + 1)) \end{aligned}$$

where $\lambda_{|s|+1} = l(\lambda, |s| + 1)$*,* $T'_{|s|+1} = T'(\lambda, |s| + 1, |\Pi|, \log(T))$ *(corresponding to the security parameter and time bound used for each of* $\tilde{\Pi}_{s0,R_0}$ *and* $\tilde{\Pi}_{s1,R_1}$*),* $\mathsf{Time}_{\mathsf{PRG}}$ *is the bound on the running time of the PRG, and* $p(\cdot,\cdot,\cdot,\cdot)$ *is the bound on* $\mathsf{Time}_{\mathsf{Enc}}$ *from the compactness of* RE*. We note that the polynomial* T' *exists because* p *is a polynomial, each of* $\lambda_{|s|+1}$ *and* $|\tilde{\Pi}_{s,R}|$ *are of size polynomial in* $\lambda, |s|$ *and* $|\Pi|$*, and the self-dependence of* $T'(\lambda, |s|, |\Pi|, \log(T))$ *on* $T'_{|s|+1}$ *is only poly-logarithmic.*

Remark: *We note that* $|\tilde{\Pi}_{s,R}|$ *is always* $poly(\lambda, |\Pi|, |s|, \log(T))$*. This is because* $\tilde{\Pi}_{s,R}$ *is fully described by* λ*,* Π*,* s*,* R *and* T*, and the size of each of these is bounded by* $poly(\lambda, |\Pi|, |s|, \log(T))$*.*

Given this definition of $\tilde{\Pi}_{s,R}$, we define our indistinguishability obfuscator as follows:

Construction 3 (Indistinguishability Obfuscator)**.** *On input $\lambda \in \mathbb{N}$, Turing machine Π and time bound T, define $\tilde{\Pi}$, the indistinguishability obfuscation of Π, to be*

$$\tilde{\Pi} = iO\,(1^\lambda, \Pi, T) = \mathsf{Enc}(1^{l(\lambda,0)}, \tilde{\Pi}_{\epsilon,R}, T'(\lambda, 0, |\Pi|, \log(T)))$$

Where ϵ is the empty string, and $R \xleftarrow{\$} \{0,1\}^{2l(\lambda,0)}$ and T' a fixed polynomial in $\lambda, |\Pi|$ and $\log(T)$, as described above.

Evaluation: The algorithm to evaluate $\tilde{\Pi}$ on input $x \in \{0,1\}^d, d < 2^\lambda$ proceeds as follows:

1. For every $0 \leq i \leq d$, compute encodings of $\tilde{\Pi}_{x_{\leq i},R}$ successively, starting with $\tilde{\Pi}$, an encoding of $\tilde{\Pi}_{\epsilon,R}$, and subsequently, for every $0 < i \leq d$, computing the encoding of $\tilde{\Pi}_{x_{\leq i},R}$ by evaluating the encoding of $\tilde{\Pi}_{x_{<i},R}$, and selecting the encoding of $\tilde{\Pi}_{x_{\leq i},R}$ from its output.
2. Evaluate the encoding of $\tilde{\Pi}_{x,R} = \tilde{\Pi}_{x_{\leq d},R}$ and obtain from its output $(\hat{\Pi}, \hat{x}) = \mathsf{Enc}(1^{l(\lambda,|x|+1)}, \Pi, x, T; R_4)$.
3. Run $\mathsf{Eval}(\hat{\Pi}, \hat{x})$ to obtain $\Pi(x)$.

To analyze the correctness, running time, and compactness of our iO construction, we make use of the following lemma:

Lemma 5. *Let Π be a polynomial-time TM, $\lambda \in \mathbb{N}$ be a security parameter, and $T \leq 2^\lambda$ be a running time bound. Then, for every $s \in \{0,1\}^*$ with $|s| \leq 2^\lambda$ and every $R \in \{0,1\}^{2l(\lambda,|s|)}$, the running time of $\tilde{\Pi}_{s,R}$ is bounded by $T'(\lambda, |s|, |\Pi|, \log(T))$.*

Proof. We prove the lemma by fixing a $\lambda \in \mathbb{N}$, and inducting on the size of s.

Base case: $|s| = 2^\lambda$**.** In this case, the running time T'_s of $\tilde{\Pi}_{s,R}$ is given by

$$\begin{aligned} T'_s &= \mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,|s|+1)}, \Pi, s, T) \\ &\leq p(l(\lambda, |s|+1), |\Pi|, |s|, \log(T)) \\ &\leq T'(\lambda, |s|, |\Pi|, \log(T)) \end{aligned}$$

This completes the base case.

Inductive step: $|s| < 2^\lambda$ By the induction hypothesis, we assume that the lemma holds for all s' with $|s'| = |s| + 1$, in particular, for $s' = s0$ and $s' = s1$.

Then, an execution of $\tilde{\Pi}_{s,R}$ runs a single evaluation of a PRG, and produces encodings of each of $\tilde{\Pi}_{s0,R_0}$, $\tilde{\Pi}_{s1,R_2}$ and (Π, s). The *PRG* runs in time $\mathsf{Time}_{\mathsf{PRG}}(R, 5 \cdot 2l(\lambda, |s|+1))$, while the encoding (Π, s) takes time $\mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,|s|+1)}, \Pi, s, T)$. Further, by the inductive hypothesis, the encodings of $\tilde{\Pi}_{s0,R_0}$ and $\tilde{\Pi}_{s1,R_2}$ each take time $\leq T'(\lambda, |s|+1, |\Pi|, \log(T)$. Combining these facts together, we have that the running time T'_s of $\tilde{\Pi}_{s,R}$ is given by:

$$\begin{aligned} T'_s &= p(\lambda_{|s|+1}, |\tilde{\Pi}_{s0,R}|, 0, \log(T'_{|s|+1})) \\ &\quad + p(\lambda_{|s|+1}, |\tilde{\Pi}_{s1,R}|, 0, \log(T'_{|s|+1})) \\ &\quad + p(\lambda_{|s|+1}, |\Pi|, |s|, \log(T)) \\ &\quad + \mathsf{Time}_{\mathsf{PRG}}(R, 5 \cdot 2l(\lambda, |s|+1)) \\ &\quad \leq T'(\lambda, |s|, |\Pi|, \log(T)) \end{aligned}$$

This concludes the inductive step, and the lemma follows. □

Correctness of *iO* : The correctness of *iO* applied to any polynomial-length x follows from the correctness of evaluating encodings RE, applied at every level

of the evaluation. More concretely, for each index $i \leq |x|$ of the evaluation, except with probability $\mu(l(\lambda, i))$, the evaluation of $\tilde{\Pi}_{x_{<i},R}$ correctly produces $\tilde{\Pi}_{x_{\leq i},R}$. Further, the final evaluation of $\hat{\Pi}, \hat{x}$ produces $\Pi^T(x)$ correctly except with probability $\mu(l(\lambda, i+1))$.

Crucially, the correctness at each step relies on the fact that each encoding $\tilde{\Pi}_{x_{<i},R}$ uses time bound $T'(\lambda, i, |\Pi|, \log(T))$, which, as argued above, is sufficiently large to compute $\tilde{\Pi}_{x_{\leq i},R}$ for the next level.

Overall, the probability of incorrect evaluation is $\leq \sum_{i=1}^{|x|} \mu(\lambda, i)$, which is negligible for any polynomial-length x.

Running time of *iO* : Again, considering step i, the evaluation at this step takes time $p(l(\lambda, i), |\Pi_{x_{<i},R}|, 0, T'(\lambda, i, |\Pi|, \log(T))$, which is $\text{poly}(\lambda, i, |\Pi|, \log(T))$. Further, the evaluation of $\hat{\Pi}, \hat{x}$ is $p(l(\lambda, |x|+1), |x|, |\Pi|, T)$, which is $\text{poly}(\lambda, |x|, |\Pi|, T)$. Therefore, overall the running time is $\text{poly}(\lambda, |x|, |\Pi|, T)$.

Efficiency of *iO* : The size of $iO(1^\lambda, \Pi, T)$ is the same as the size of $\tilde{\Pi}_{\epsilon,R}$, which itself is bounded by $\mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,0)}, \Pi_{\epsilon,R}, T'(\lambda, 0, |\Pi|, \log(T)))$, which is $\text{poly}(\lambda, |\Pi|, \log(T))$ by the efficiency of RE.

4.4.1 Security Proof

Theorem 26. *Let* (Enc, Eval) *be a sub-exponentially-indistinguishability-secure, compact randomized encoding scheme and let* PRG *be a sub-exponentially-secure pseudorandom generator. Then the indistinguishability obfuscator defined in Construction 3 is subexponentially-secure.*

Proof. Consider any pair of ensembles of Turing machines and time bounds $\left\{\Pi^0_\lambda, \Pi^1_\lambda, T_\lambda\right\}$ where for every $\lambda \in \mathbb{N}$, $\Pi^0 = \Pi^0_\lambda$, $\Pi^1 = \Pi^1_\lambda$, $T = T_\lambda$,

$$|\Pi^0| = |\Pi^1| \leq \text{poly}(\lambda) \quad |T| \leq \text{poly}(\lambda)$$

$$\forall x, \Pi^{0,T}(x) = \Pi^{1,T}(x)$$

We first introduce some notation to describe the distributions of randomized encodings generated by $i\mathcal{O}(1^\lambda, \Pi^0_\lambda, T_\lambda)$ and $i\mathcal{O}(1^\lambda, \Pi^1_\lambda, T_\lambda)$. For $\lambda \in \mathbb{N}$, $s \in \{0,1\}^*, |s| \leq 2^\lambda$, we define the following distributions

$$D_{\lambda,0,s} = \mathsf{Enc}(1^{l(\lambda,|s|)}, \tilde{\Pi}^0_{s,R}, T')$$

$$D_{\lambda,1,s} = \mathsf{Enc}(1^{l(\lambda,|s|)}, \tilde{\Pi}^1_{s,R}, T')$$

where R is uniformly random, T' is as described in Construction 2 and $\tilde{\Pi}^b_{s,R}$ is defined for the Turing machine Π^b_λ, security parameter λ and time bound T_λ. We will show something stronger than the theorem statement. In particular, we have the following claim.

Claim 3. *There exists $\lambda_0, \epsilon \in \mathbb{N}$ such that for every $\lambda > \lambda_0$, for every $s \in \{0,1\}^*, |s| \leq 2^\lambda$ we have that the distributions $D_{\lambda,0,s}$ and $D_{\lambda,1,s}$ are $S(\lambda)$ indistinguishable where $S(\lambda) \geq 10 \cdot 2^{l(\lambda,|s|-1)^\epsilon}$.*

Using the above claim with s as the empty string and recalling $l(\lambda, 0) = \lambda$, the theorem statement follows. Therefore, in the remainder of the proof, we prove the above claim.

Proof of claim Let ϵ be the larger of the constants associated with the sub-exponential security of the pseudorandom generator PRG and the indistinguishability security of the encoding scheme $(\mathsf{Enc}, \mathsf{Eval})$ (these constants are also named ϵ in their respective security definitions). Similarly, We consider λ_0 to be

large enough so that the security of the encoding scheme $(\mathsf{Enc}, \mathsf{Eval})$ and the pseudorandom generator PRG is applicable. We will actually require a larger λ_0 so that certain asymptotic conditions (depending only on the polynomial size bounds of Π^0_λ, Π^1_λ and T_λ) hold, which we make explicit in the remainder of the proof. For every $\lambda > \lambda_0$, we prove the claim by induction on $|s|$. Our base case will be when $|s| = 2^\lambda$ and in the inductive step we show the claim holds for all s of a particular length d, if it holds for all s of length $d + 1$.

Induction statement, for a fixed $\lambda > \lambda_0$: For every $s \in \{0,1\}^{\leq 2^\lambda}$, the distributions $D_{\lambda,0,s}$ and $D_{\lambda,1,s}$ are $10 \cdot 2^{l(\lambda,|s|-1)^\epsilon}$ indistinguishable.

Base case: $|s| = 2^\lambda$.

In this case, recall that the output of $\tilde{\Pi}^b_{s,R}$ is simply $(\hat{\Pi}^{b,T}_\lambda, \hat{s})$. We first claim that, for all s, $(\hat{\Pi}^{0,T}_\lambda \hat{s})$ and $(\hat{\Pi}^{1,T}_\lambda, \hat{s})$ are $2^{\lambda'^\epsilon}$ indistinguishable where $\lambda' = l(\lambda, |s|)$, as follows.

Recall that the output of evaluating $\hat{\Pi}^{b,T}_\lambda, \hat{s}$ is simply $\Pi^{b,T}_\lambda(s)$. Since we have that $\Pi^{0,T}_\lambda(s) = \Pi^{1,T}_\lambda(s)$ for all s, we can apply the security of the randomized encoding scheme. More concretely, since the output (point) distributions are identical, they are $10 \cdot 2^{\lambda'^\epsilon}$-indistinguishable where $\lambda' = l(\lambda, |s| + 1)$. Let $B(\cdot)$ be a polynomial such that $B(\lambda')$ bounds from above $|\Pi^b|, |s|$ and T. By the security of the encoding scheme, the encodings $(\hat{\Pi}^{0,T}_\lambda \hat{s})$ and $(\hat{\Pi}^{1,T}_\lambda \hat{s})$ are S' indistinguishable where

$$S' \geq \frac{10 \cdot 2^{l(\lambda,|s|+1)^\epsilon}}{l(\lambda,|s|+1)^c} - B(l(\lambda,|s|+1))^c \geq \frac{10 \cdot 2^{l(\lambda,|s|+1)^\epsilon}}{l(\lambda,|s|+1)^d} \geq 10 \cdot 2^{l(\lambda,|s|)^\epsilon}$$

where the first inequality holds for sufficiently large λ and in the second inequality, we use the fact that $l(\lambda, |s| + 1) = l(\lambda, |s|) + \lambda^{d/\epsilon}$. Thus $(\hat{\Pi}^{0,T}_\lambda, \hat{s})$ and $(\hat{\Pi}^{1,T}_\lambda, \hat{s})$ are $10 \cdot 2^{l(\lambda,|s_\lambda|)^\epsilon}$-indistinguishable.

Now, recall that the output of $\tilde{\Pi}^b_{s,R}$ is simply $(\hat{\Pi}^{b,T}_\lambda, \hat{s})$. By the above argument, we have that, for all s, $(\hat{\Pi}^{0,T}_\lambda \hat{s})$ and $(\hat{\Pi}^{1,T}_\lambda, \hat{s})$ are $2^{\lambda'^\epsilon}$-indistinguishable where $\lambda' = l(\lambda, |s|)$. Let B' be the polynomial such that $B'(l(\lambda, |s|))$ bounds $|\tilde{\Pi}^b_{s,R}|$ and the running time of $\tilde{\Pi}^b_{s,R}$ as per Lemma 5. The encodings $D_{\lambda,0,s}$ and $D_{\lambda,1,s}$ are S' indistinguishable where

$$S' \geq \frac{10 \cdot 2^{l(\lambda,|s|)^\epsilon}}{l(\lambda,|s|)^c} - B'(l(\lambda,|s|))^c \geq \frac{10 \cdot 2^{l(\lambda,|s|+1)^\epsilon}}{l(\lambda,|s|)^d} \geq 10 \cdot 2^{l(\lambda,|s|-1)^\epsilon}$$

where, as before, the first inequality holds for sufficiently large λ and in the second inequality, we use the fact that $l(\lambda, |s|+1) = l(\lambda, |s|) + \lambda^{d/\epsilon}$. Hence the claim holds for $|s| = 2^\lambda$.

Inductive step: $|s| < 2^\lambda$. By the induction hypothesis, we assume the claim holds for all s' such that $|s'| = |s| + 1$. Recall that the output of $\tilde{\Pi}^b_{s,R}$ (where $R \xleftarrow{\$} \{0,1\}^{2l(\lambda,|s|)}$) is

$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \tilde{\Pi}^b_{s0,R_0}, T'; R_1)$$
$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \tilde{\Pi}^b_{s1,R_2}, T'; R_3)$$
$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi^b_\lambda, s, T; R_4)$$

where $(R_0, R_1, R_2, R_3, R_4) \leftarrow \mathsf{PRG}(R, 5 \cdot 2l(\lambda, |s| + 1))$. Let H^b denote the above output distribution. We will show H^0 and H^1 are indistinguishable by a hybrid argument as follows.

- Let G_1 be a hybrid distribution exactly as H^0 except that $(R_0, R_1, R_2, R_3, R_4) \xleftarrow{\$} \{0,1\}^{5 \cdot 2l(\lambda,|s|+1)}$. We claim that for both the distributions H^0 and G_1 are $5 \cdot 2^{\lambda'^\epsilon}$ indistinguishable where $\lambda' = l(\lambda, |s|)$.

  This follows from the PRG security as follows: any size $5 \cdot 2^{\lambda'^\epsilon}$ adversary A that distinguishes H^0 and G_1 can be turned into an adversary A' that can

break the PRG security with seed length $2\lambda'$ with the same advantage. A' has Π^0_λ, Π^1_λ, T_λ and s hardcoded in it. Hence, the size of A' is

$$5 \cdot 2^{\lambda'^\epsilon} + \text{poly}(\lambda) + \text{poly}(|s|) \leq 5 \cdot 2^{\lambda'^\epsilon} + \text{poly}(\lambda') \leq 2^{(2\lambda')^\epsilon}$$

where the last inequality holds when λ is sufficiently large. Hence, A' breaks the $2^{(2\lambda')^\epsilon}$-security of PRG and we have a contradiction.

Writing out the components of G_1, we have that it is identical to

$$G_1 \equiv D_{\lambda,0,s0}, D_{\lambda,0,s1}, \mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi^0_\lambda, s, T_\lambda; R)$$

- Let G_2 be a hybrid distribution obtained by modifying the first component of G_1 as follows.

  $$G_2 \equiv D_{\lambda,1,s0}, D_{\lambda,0,s1}, \mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi^0_\lambda, s, T_\lambda; R)$$

  We show that G_1 and G_2 are $5 \cdot 2^{\lambda'^\epsilon}$ indistinguishable. This follows from the induction hypothesis as follows: any size $5 \cdot 2^{\lambda'^\epsilon}$ adversary A that distinguishes G_1 and G_2 with advantage better than $1/(5 \cdot 2^{\lambda'^\epsilon})$ can be turned into an adversary A' that can distinguish $D_{\lambda,0,s0}$ and $D_{\lambda,1,s0}$ with the same advantage. As before, A' has Π^0_λ, Π^1_λ, T_λ and s hardcoded in it, and therefore the size of A' is at most $5 \cdot 2^{\lambda'^\epsilon} + \text{poly}(\lambda') \leq 10 \cdot 2^{\lambda'^\epsilon}$. Hence, A' breaks the induction hypothesis that says $D_{\lambda,0,s0}$ and $D_{\lambda,1,s0}$ are $10 \cdot 2^{\lambda'^\epsilon}$-indistinguishable.

- Similarly, let G_3 be a hybrid distribution obtained by modifying the second component of G_2 as follows.

  $$G_3 \equiv D_{\lambda,1,s0}, D_{\lambda,1,s1}, \mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi^0_\lambda, s, T_\lambda; R)$$

  Similarly as above, we have that G_2 and G_3 are $5 \cdot 2^{\lambda'^\epsilon}$-indistinguishable.

- Let G_4 be a hybrid distribution obtained by modifying the third component of G_3 as follows.

$$G_4 \equiv D_{\lambda,1,s0}, D_{\lambda,1,s1}, \mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi^1_\lambda, s, T_\lambda; R)$$

  We show G_3 and G_4 are $5 \cdot 2^{\lambda'^\epsilon}$-indistinguishable. First, since $\Pi^{0,T}_\lambda(s) = \Pi^{1,T}_\lambda(s)$, by the security of the encoding scheme, we have that the encodings that form the third component of G_3 and G_4 are S' indistinguishable where, similar to the base case, $B(l(\lambda,|s|))$ bounds from above $|\Pi^b_\lambda|, |s|$ and T

$$S' \geq \frac{10 \cdot 2^{l(\lambda,|s|)^\epsilon}}{l(\lambda,|s|)^c} - B(l(\lambda,|s|))^c \geq \frac{10 \cdot 2^{l(\lambda,|s|)^\epsilon}}{l(\lambda,|s|)^d} \geq 10 \cdot 2^{l(\lambda,|s|-1)^\epsilon}$$

  Hence by a similar argument as before, the hybrid distributions are $5 \cdot 2^{\lambda'^\epsilon}$-indistinguishable.

- Finally we observe that G_4 and H^1 are $5 \cdot 2^{\lambda'^\epsilon}$-indistinguishable just as G_1 and H^0 were. By a simple hybrid argument, we have that H^0 and H^1 are $2^{\lambda'^\epsilon}$-indistinguishable.

  Recall that H^0 and H^1 are the distributions of outputs of $\tilde{\Pi}^0_{s,R}$ and $\tilde{\Pi}^1_{s,R}$ respectively. By the security of the randomized encoding scheme, the encodings of these machines, *i.e.* $D_{\lambda,0,s}$ and $D_{\lambda,1,s}$ are $S'(\lambda)$-indistinguishable where

$$S'(\lambda) \geq \frac{2^{l(\lambda,|s|)^\epsilon}}{l(\lambda,|s|)^c} - B'(l(\lambda,|s|)^c \geq \frac{2^{l(\lambda,|s|)^\epsilon}}{l(\lambda,|s|)^d} \geq \frac{2^{l(\lambda,|s|-1)^\epsilon} \cdot 2^{(2d\lambda)}}{2^{d\lambda} \cdot (2d\lambda)^{d/\epsilon}} \geq 10 \cdot 2^{l(\lambda,|s|-1)^\epsilon}$$

  where $B'(l(\lambda,|s|))$ bounds from above $|\Pi^b_{s,R}|$ and T' as in Lemma 5. The second inequality holds for sufficiently large λ. In the third inequality, we use the fact that $l(\lambda,|s|) \leq |s|(2d\lambda)^{1/\epsilon} \leq 2^\lambda(2d\lambda)^{1/\epsilon}$ and the last inequality holds for sufficiently large λ.

□

4.4.2 Nice Distributions

Later in Section 4.7, we show that compact RE does not exist for general distributions in the plain model. However, here we observe that the above construction of unbounded input IO relies only on compact RE for certain "special purpose" distributions that is not ruled out by the impossibility result in Section 4.7. We now abstract out the structure of these special purpose distributions. Let $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Dec})$ be a randomized encoding scheme; we define "nice" distributions w.r.t. RE.

0-nice distributions: We say that a pair of distribution ensembles $\{\mathcal{D}_{0,\lambda}\}$ and $\{D_{1,\lambda}\}$ are *0-nice* if $D_{0,\lambda}$ always outputs a fixed tuple (Π_0, x, T) while $D_{1,\lambda}$ always outputs a fixed tuple (Π_1, x, T), satisfying that $\Pi_0^T(x) = \Pi_1^T(x)$.

k-nice distributions: We say that a pair of distribution ensembles $\{\mathcal{D}_{0,\lambda}\}$ and $\{\mathcal{D}_{1,\lambda}\}$ are *k-nice* if there exist some $\ell = \text{poly}(\lambda)$ pairs of distributions $(\{\mathcal{E}^i_{0,\lambda}\}, \{\mathcal{E}^i_{1,\lambda}\})_{i\in[\ell]}$, where the i^{th} pair is k^i-nice with $k^i \leq k-1$, such that, $\mathcal{D}_{b,\lambda}$ samples tuple (Π_b, x_b, T_b) satisfying the following:

- For each $i \in [\ell]$, sample $(\Lambda^i_b, z^i_b, T^i_b) \xleftarrow{\$} \mathcal{E}^i_{b,\lambda}$.
- The output of $\Pi_b(x_b)$ consists of ℓ randomized encodings, where the i^{th} encoding is in the support of $\mathsf{Enc}(1^{\lambda'}, \Lambda^i_b, z^i_b, T^i_b)$, for some $\lambda' = \text{poly}(\lambda)$.

Finally, we say that a pair of distribution ensembles $\{\mathcal{D}_{0,\lambda}\}$ and $\{\mathcal{D}_{1,\lambda}\}$ are *nice* w.r.t. RE if they are *k-nice* w.r.t. RE for some integer k.

Our construction of unbounded input IO and its analysis in previous sections relies only on compact RE for nice distribution ensembles. Hence we can refine Theorem 26 to the following:

Proposition 4. *Assume the existence of a compact randomized encoding scheme* RE *which is sub-exponentially-indistinguishability-secure for every pair of distribution ensemble that are nice w.r.t.* RE*; assume further the existence of sub-exponentially secure one-way functions. Then, there is an unbounded-input indistinguishability obfuscator for Turing machines.*

We stress again that compact RE for nice distributions is not ruled out by the impossibility result in Section 4.7. Hence, we obtain unbounded input IO from a new assumption different from the extactability assumptions used in previous work [42, 2, 105].

Candidate Construction: Finally, we describe a candidate construction of compact RE for nice distributions using the KLW indistinguishability obfuscator for bounded-input Boolean Turing machines: Given input $(1^\lambda, M, x, T)$, the encoding is an obfuscation, using the KLW scheme, of the program $\Pi_{M,x}$ that on input $i \in [T]$ outputs the i^{th} bit of the output $M^T(x)$. Since $\Pi_{M,x}$ is Boolean, the KLW obfuscator can be applied, and the encoding time is $\text{poly}(\lambda, |M|, |x|, \log T)$ (hence compact). By the security of indistinguishability obfuscation, for any M_1, x_1 and M_2, x_2 with identical outputs, their encodings are indistinguishable, and thus this construction is a weak compact RE. We here consider it also a candidate construction for compact RE with distributional indistinguishability.

4.5 Bounded-Input IO from Sublinear RE

In this section, we construct a bounded-input indistinguishability obfuscator for Turing machines, using randomized encoding schemes with sublinear com-

pactness. The construction is very similar to the construction described earlier in Section 4.4. The main difference is that due to the fact that we only use sublinear compactness, the depth of the tree of encodings we construct must be bounded to some polynomial size given as a parameter, rather than being of depth 2^λ. Further, the correctness and efficiency analysis is slightly different to account for the semi-compactness of the RE scheme used, as opposed to full compactness in the previous construction.

Let $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$ be a randomized encoding scheme for Turing machines with sub-exponential indistinguishability security and sublinear efficiency. Let c be the constant associated with the security loss in the security of $(\mathsf{Enc}, \mathsf{Eval})$. We assume that $\mathsf{Enc}(1^\lambda, \cdot, \cdot)$ requires a random tape of length λ, and this is without loss of generality for two reasons: First, one can always apply a PRG to expand the λ-bit random string to a pseudo-random string of arbitrary polynomial length, and second, by Claim 4 in Section 4.3.4 it is without loss of generality to assume that we start with a RE scheme with a sufficiently small sublinear time complexity and thus even counting the time for PRG expansion, the overall time complexity is still sublinear. Let PRG be a sub-exponentially secure pseudorandom generator and let ϵ be the constant associated with the sub-exponential security of PRG.

For every $\lambda \in \mathbb{N}$, $D \leq 2^\lambda$, define

$$l(\lambda, -1) = \lambda$$

$$l(\lambda, D) = l(\lambda, D-1) + (2d\lambda)^{1/\epsilon}$$

where d is any constant strictly greater than c.

Construction 4. *Consider a Turing machine* Π*, security parameter* $\lambda \in \mathbb{N}$*, an input-*

length bound n, and time bound T on the running time of Π. For every partial input $s \in \{0,1\}^*$ with $|s| \leq n$ and $R \in \{0,1\}^{2l(\lambda,|s|)}$, we recursively define a Turing machine $\tilde{\Pi}_{s,R}$ as follows:

When $|s| < n$:

On the empty input, $\tilde{\Pi}_{s,R}$ outputs:

$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \tilde{\Pi}_{s0,R_0}, T'(\lambda, |s|+1, |\Pi|, n, T); R_1)$$

$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \tilde{\Pi}_{s1,R_2}, T'(\lambda, |s|+1, |\Pi|, n, T); R_3)$$

$$\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi, s, T; R_4)$$

where $(R_0, R_1, R_2, R_3, R_4) \leftarrow \mathsf{PRG}(R, 5 \cdot 2l(\lambda, |s|+1))$ and T' is a fixed polynomial in $\lambda, |s|+1, |\Pi|$, n and T, defined below. In the special case when $|s| = n-1$, the time bound used in the first two encodings is set to T.

On all other inputs, $\tilde{\Pi}_{s,R}$ outputs $\perp$.

When $|s| = n$:

On the empty input, $\tilde{\Pi}_{s,R}$ outputs $\mathsf{Enc}(1^{l(\lambda,|s|+1)}, \Pi, s, T; R)$. On all other inputs, $\tilde{\Pi}_{s,R}$ outputs $\perp$.

Let $p(\cdot)$ and ϵ' respectively be the polynomial and constant corresponding to the sub-linear efficiency of RE. For any λ, n, Π and T, we define the following terms (which are implicitly functions of λ, n, Π and T):

$$\lambda_n = l(\lambda, n)$$

$$\lambda_{n+1} = l(\lambda, n+1)$$

$$\tilde{\Pi}_n = \tilde{\Pi}_{0^n, 0^{2\lambda_n}}$$

$$A = 2 \cdot p(1^{\lambda_n}, |\tilde{\Pi}_n|, 0)$$

$$B = p(1^{\lambda_{n+1}}, |\Pi|, n)T^{1-\epsilon'}$$

$$C = \mathsf{Time}_{\mathsf{PRG}}(0^{2\lambda_n}, 5 \cdot 2\lambda_{n+1})$$

We note that $\tilde{\Pi}_n$ is the largest size of any $\tilde{\Pi}_{s,R}$ we ever need to consider, and further, that it has a description of polynomial size, and hence every $\tilde{\Pi}_{s,R}$ has polynomial size.

We define the polynomial $T'(\cdot,\cdot,\cdot,\cdot)$ (corresponding to the bound placed on the running time of $\tilde{\Pi}_{s,R}$), for all λ, $s \in \{0,1\}^{\leq n}$, $R \in \{0,1\}^{2l(\lambda,|s|)}$, Π and T, to be

$$T'(\lambda, |s|, |\Pi|, n, T) = (n - |s| + 1) \cdot (A^{1/\epsilon'} + B + C)$$

where each of A, B and C are defined relative to $\lambda, |\Pi|, n$ and T given as input to T'.

Given this definition of $\tilde{\Pi}_{s,R}$ and T', we define our indistinguishability obfuscator as follows:

Construction 5 (Indistinguishability Obfuscator)**.** *On input $\lambda \in \mathbb{N}$, Turing machine Π, input length bound n and time bound T, define $\tilde{\Pi}$, the indistinguishability obfuscation of Π, to be*

$$\tilde{\Pi} = i\mathcal{O}\,(1^\lambda, \Pi, n, T) = \mathsf{Enc}(1^{l(\lambda,0)}, \tilde{\Pi}_{\epsilon,R}, T'(\lambda, 0, |\Pi|, n, T))$$

Where ϵ is the empty string, and $R \xleftarrow{\$} \{0,1\}^{2l(\lambda,0)}$ and T' a fixed polynomial in $\lambda, |\Pi|$, n and T, as described above.

Evaluation: The algorithm to evaluate $\tilde{\Pi}$ on input $x \in \{0,1\}^d, d \leq n$ proceeds as follows:

1. For every $0 \leq i \leq d$, compute encodings of $\tilde{\Pi}_{x_{\leq i},R}$ successively, starting with $\tilde{\Pi}$, an encoding of $\tilde{\Pi}_{\epsilon,R}$, and subsequently, for every $0 < i \leq d$, computing the encoding of $\tilde{\Pi}_{x_{\leq i},R}$ by evaluating the encoding of $\tilde{\Pi}_{x_{<i},R}$, and selecting the encoding of $\tilde{\Pi}_{x_{\leq i},R}$ from its output.
2. Evaluate the encoding of $\tilde{\Pi}_{x,R} = \tilde{\Pi}_{x_{\leq d},R}$ and obtain from its output $(\hat{\Pi}, \hat{x}) = \mathsf{Enc}(1^{l(\lambda,|x|+1)}, \Pi, x, T)$.
3. Run $\mathsf{Eval}(\hat{\Pi}, \hat{x})$ to obtain $\Pi(x)$.

To analyze the correctness, running time, and compactness of our *iO* construction, we make use of the following lemma:

Lemma 6. *Let Π be a polynomial-time TM, $\lambda \in \mathbb{N}$ be a security parameter, n an input length bound, and $T \leq 2^\lambda$ be a running time bound. Then, for every $s \in \{0,1\}^*$ with $|s| \leq 2^\lambda$ and every $R \in \{0,1\}^{2l(\lambda,|s|)}$, the running time of $\tilde{\Pi}_{s,R}$ is bounded by $T'(\lambda, |s|, |\Pi|, n, T)$.*

Proof. We prove the lemma by fixing $\lambda \in \mathbb{N}$, and inducting on the size of s.

Base case: $|s| = n$**.** In this case, the running time T'_s of $\tilde{\Pi}_{s,R}$ is given by

$$\begin{aligned} T'_s &= \mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,|s|+1)}, \Pi, s, T) \\ &\leq p(l(\lambda, |s|+1), |\Pi|, n) T^{1-\epsilon'} \\ &\leq B \\ &\leq T'(\lambda, |s|, |\Pi|, n, T) \end{aligned}$$

This completes the base case.

Inductive step: $|s| < n$ By the induction hypothesis, we assume that the lemma holds for all s' with $|s'| = |s| + 1$, in particular, for $s' = s0$ and $s' = s1$.

Then, an execution of $\tilde{\Pi}_{s,R}$ runs a single evaluation of a PRG, and produces encodings of each of $\tilde{\Pi}_{s0,R_0}$, $\tilde{\Pi}_{s1,R_2}$ and (Π, s). The *PRG* runs in time $\mathsf{Time}_{\mathsf{PRG}}(R, 5 \cdot 2l(\lambda, |s|+1)) \leq C$, while the encoding (Π, s) takes time $\mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,|s|+1)}, \Pi, s, T) \leq B$. By the inductive hypothesis, the machines $\tilde{\Pi}_{s0,R_0}$ and $\tilde{\Pi}_{s1,R_2}$ each run in time $\leq T'(\lambda, |s|+1, |\Pi|, T)$. Then, we have that the encoding time of $\tilde{\Pi}_{s0,R_0}$ and $\tilde{\Pi}_{s1,R_2}$ is bounded by:

$$
\begin{aligned}
&2 \cdot \mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,|s|+1)}, \tilde{\Pi}_{s0,R_0}, 0, T'(\lambda, |s|+1, |\Pi|, T)) \\
&\leq 2 \cdot p(1^{l(\lambda,|s|+1)}, \tilde{\Pi}_{s0,R_0}, 0) \cdot T'(\lambda, |s|+1, |\Pi|, T)^{1-\epsilon'} \\
&\leq A \cdot T'(\lambda, |s|+1, |\Pi|, T)^{1-\epsilon'} \\
&\leq A \cdot T'(\lambda, |s|+1, |\Pi|, T)^{1} \cdot T'(\lambda, |s|+1, |\Pi|, T)^{-\epsilon'} \\
&\leq A \cdot T'(\lambda, |s|+1, |\Pi|, T) \cdot (A^{1/\epsilon'})^{-\epsilon} \\
&\leq T'(\lambda, |s|+1, |\Pi|, T)
\end{aligned}
$$

Combining these facts together, we have that the running time T'_s of $\tilde{\Pi}_{s,R}$ is given by:

$$
\begin{aligned}
T'_s &\leq T'(\lambda, |s|+1, |\Pi|, T) + B + C \\
&\leq (n - (|s|+1) + 1) \cdot (A^{1/\epsilon'} + B + C) + B + C \\
&\leq (n - |s| + 1) \cdot (A^{1/\epsilon'} + B + C) \\
&\leq T'(\lambda, |s|, |\Pi|, T)
\end{aligned}
$$

This concludes the inductive step, and the lemma follows. □

Correctness of *iO* : As in the construction in Section 4.4, the correctness of *iO* applied to any polynomial-length x follows from the correctness of evaluating encodings RE, applied at every level of the evaluation. More concretely, for each

index $i \leq |x|$ of the evaluation, except with probability $\mu(l(\lambda, i))$, the evaluation of $\tilde{\Pi}_{x_{<i},R}$ correctly produces $\tilde{\Pi}_{x_{\leq i},R}$. Further, the final evaluation of $\hat{\Pi}, \hat{x}$ produces $\Pi^T(x)$ correctly except with probability $\mu(l(\lambda, i+1))$.

Crucially, the correctness at each step relies on the fact that each encoding $\tilde{\Pi}_{x_{<i},R}$ uses time bound $T'(\lambda, i, |\Pi|, \log(T))$, which, as argued above, is sufficiently large to compute $\tilde{\Pi}_{x_{\leq i},R}$ for the next level.

Overall, the probability of incorrect evaluation is $\leq \sum_{i=1}^{|x|} \mu(\lambda, i)$, which is negligible for any polynomial-length x.

Running time of $i\mathcal{O}$: Again, considering step i, the evaluation at this step takes time $p(l(\lambda, i), |\Pi_{x_{<i},R}|, 0, T'(\lambda, i, |\Pi|, T)$, which is $\text{poly}(\lambda, i, |\Pi|, T)$. Further, the evaluation of $\hat{\Pi}, \hat{x}$ to produce the final output of the $i\mathcal{O}$ is $p(l(\lambda, |x|+1), |x|, |\Pi|, T)$, which is $\text{poly}(\lambda, |x|, |\Pi|, T)$. Therefore, overall the running time is $\text{poly}(\lambda, |x|, |\Pi|, T)$.

Efficiency of $i\mathcal{O}$: The size of $i\mathcal{O}$ $(1^\lambda, \Pi, T)$ is the same as the size of $\tilde{\Pi}_{\epsilon,R}$, which itself is bounded by $\mathsf{Time}_{\mathsf{Enc}}(1^{l(\lambda,0)}, \Pi_{\epsilon,R}, T'(\lambda, 0, |\Pi|, T))$, which is $\text{poly}(\lambda, |\Pi|, T)$ by the efficiency of RE.

Security of $i\mathcal{O}$: We note that the security proof for Construction 3 presented in Section 4.4 carries over exactly to the construction presented in this section. The only difference is that the base case for the induction starts from $|s| = n$ rather than $|s| = 2^\lambda$. Given this change, exactly the same inductive argument can be used to show that subexponential security of RE implies subexponential security of the $i\mathcal{O}$ construction given above.

4.6 Bounded-Input IO from Compact RE in the CRS Model

In this section we consider compact RE schemes for Turing machines in the *common reference string* (CRS) model. We show that (1) such encoding schemes can be constructed from compact functional encryption for circuits, and that (2) such encoding schemes suffice to get IO for circuits, which then by [112] suffices to get bounded-input IO for Turing machines.

4.6.1 Randomized Encoding Schemes in the CRS model

We first formally define a randomized encoding scheme for a class of Turing machines in the CRS model. In this model, a one-time setup is performed which takes (in addition to the security parameter) a bound on machine size, input length, running time and output length. Only computations that respect these bounds can be encoded using this setup. The setup outputs a *long* CRS (the length is polynomial in the aforementioned bounds) and a *short* public encoding key (which depends only on the security parameter). The public encoding key is used by the encoding algorithm, which produces encodings that are *compact* as before. The CRS is used by the evaluation algorithm.

Definition 32 (Randomized Encoding Schemes in the CRS Model)**.** *A Randomized Encoding scheme* RE *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *in the CRS model consists of the following algorithms:*

- $(\mathsf{crs}, pk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^m, 1^n, 1^T, 1^l)$*:* Setup *gets as input (in unary) the security parameter* λ*, a machine size bound* m*, input length bound* n*, time bound* T *and output length bound* l*.*

- $\hat{\Pi}_x \xleftarrow{\$} \mathsf{Enc}(pk, \Pi, x)$*:* Enc *is probabilistic and gets as input a public key pk generated by* Setup*, Turing machine* $\Pi \in \mathcal{M}_\lambda$ *and input x. It outputs an encoding* $\hat{\Pi}_x$[7]*.*
- $y \leftarrow \mathsf{Eval}(\hat{\Pi}_x, \mathsf{crs})$*: On input* $\hat{\Pi}_x$ *produced by* Enc *and* crs *produced by* Setup*,* Eval *outputs y.*

Correctness: *For every security parameters* $\lambda \in \mathbb{N}$*,* $m, n, T, l \in \mathbb{N}$*, Turing machine* $\Pi \in \mathcal{M}_\lambda$ *and input x, such that,* $|\Pi| \leq m$*,* $|x| \leq n$*, and* $|\Pi^T(x)| \leq l$*, we have that*

$$\Pr\left[\begin{array}{c}(\mathsf{crs}, pk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^m, 1^n, 1^T, 1^l) \\ \hat{\Pi}_x \xleftarrow{\$} \mathsf{Enc}(pk, \Pi, x)\end{array} : \mathsf{Eval}(\hat{\Pi}_x, \mathsf{crs}) = \Pi^T(x)\right] = 1$$

The simulation security in the CRS model is essentially the same as that in the plain model (Definition 27), except that simulator can additionally simulate the CRS.

Definition 33. *A randomized encoding scheme* RE *for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *in the CRS model satisfies* $(\lambda_0, S(\cdot))$**-simulation security**, *if there exists a PPT algorithm* Sim *and a constant c, such that, for every ensemble* $\{\Pi_\lambda, x_\lambda, m_\lambda, n_\lambda, l_\lambda, T_\lambda\}$ *where* $\Pi_\lambda \in \mathcal{M}_\lambda$ *and* $|\Pi_\lambda|, |x_\lambda|, m_\lambda, n_\lambda, l_\lambda, T_\lambda \leq B(\lambda)$ *for some polynomial B, the following ensembles are* $(\lambda_0, S'(\lambda))$ *indistinguishable, with* $S'(\lambda) = S(\lambda) - B(\lambda)^c$ *for all* $\lambda \in N$*.*

$$\left\{(\mathsf{crs}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^m, 1^n, 1^T, 1^l),\ \hat{\Pi}_x \xleftarrow{\$} \mathsf{Enc}(pk, \Pi, x) \ :\ (\mathsf{crs}, pk, \hat{\Pi}_x)\right\}$$
$$\left\{(\mathsf{crs}, pk, \hat{\Pi}_x) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \Pi^T(x), 1^{|\Pi|}, 1^{|x|}, 1^m, 1^n, 1^T, 1^l) \ :\ (\mathsf{crs}, pk, \hat{\Pi}_x)\right\}$$

where subscripts of security parameter are suppressed.

[7]Encoding $\hat{\Pi}_x$ can be viewed as the combination of the program encoding $\hat{\Pi}$ and the input encoding $\hat{x}$ of Definition 26

Indistinguishability-security in the CRS model can be defined similar to that in the plain model, except now we need to work with distributions $D_{b,\lambda}$ that samples $(\Pi_b, x_b, T_b, m_b, n_b, l_b)$.

Definition 34 (Distributional $(\lambda_0, S(\cdot))$-Indistinguishability Security)**.** *A randomized encoding scheme* RE *for a class of Turing machines $\{\mathcal{M}_\lambda\}$ in the CRS model satisfies* $(\lambda_0, S(\cdot))$-ind-security*, if the following is true w.r.t. some constant $c > 0$: For every ensembles of distributions $\{D_{0,\lambda}\}$ and $\{D_{1,\lambda}\}$ with the following property:*

1. *there exists a polynomial B, such that, for every $b \in \{0,1\}$, $D_{b,\lambda}$ is a distribution over tuples of the form $(\Pi_b, x_b, T_b, m_b, n_b, l_b)$ with polynomial size and $\lambda, |\Pi_b|, |x_b|, T_b, m_b, n_b, l_b \leq B(\lambda)$.*
2. *there exist an integer $\lambda_0' \geq \lambda_0$, and a function S' with $\leq S'(\lambda) \leq S(\lambda)$ for all λ, such that, the ensembles of output distributions $\{O_{0,\lambda}\}$ and $\{O_{1,\lambda}\}$ are $(\lambda_0', S'(\cdot))$-indistinguishable,*

$$O_{b,\lambda} = \left((\Pi_b, x_b, T_b, m_b, n_b, l_b) \xleftarrow{\$} \mathcal{D}_{b,\lambda} : \Pi_b^{T_b}(x_b), |\Pi_b|, |x_b|, T_b, m_b, n_b, l_b\right)$$

the ensembles of encoding $\{\mathcal{E}_{0,\lambda}\}$ and $\{\mathcal{E}_{1,\lambda}\}$ defined below is $(\lambda_0', S''(\cdot))$ indistinguishable, where $S''(\lambda) = \frac{S'(\lambda)}{\lambda^c} - B(\lambda)^c$.

$$\mathcal{E}_{b,\lambda} = \Big((\Pi_b, x_b, T_b, m_b, n_b, l_b) \xleftarrow{\$} \mathcal{D}_{0,\lambda},$$
$$(\mathsf{crs}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^{m_b}, 1^{n_b}, 1^{T_b}, 1^{l_b}),\ \hat{\Pi}_x \xleftarrow{\$} \mathsf{Enc}(pk, \Pi_b, x_b)\ :\ (\mathsf{crs}, pk, \hat{\Pi}_x)\Big)$$

(We note that [78] previously defined a notion of distributional indistinguishability security for reusable garbled circuits. In the CRS model, reusable garbled circuits are equivalent to a *secret key* variant of RE (where one needs a secret key related to the CRS in order to encode). Thus our definition of distri-

butional indistinguishability of RE in the CRS model (modified to have secret key-based encoding) can be viewed as similar to their definition.)

In the CRS model, it is possible to have a compact RE for all Turing machines with simulation security. Therefore, we define compactness in the CRS model independent of security notion.

Definition 35 (Compactness and Sublinear Compactness in the CRS model)**.** *A randomized encoding scheme* $\mathsf{RE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval})$ *for Turing machines in the CRS model is* compact *(or sublinear compact) if* Setup *is PPT, and* Enc *and* Eval *have the same efficiency as their counterparts in a compact (or sublinear compact) randomized encoding scheme for Turing machines in the plain model.*

Remark 8. *As mentioned in Section 4.3.5, in the plain model* (λ_0, S)*-simulation security implies* (λ_0, S)*-indistinguishability security. We remark that the same holds in the CRS model and the proof is essentially the same. We omit the details here.*

4.6.2 Succinctness and Weak Sublinear Compactness

We also consider a different weakening of compactness, called succinctness [31], where encoding time can depend linearly on the length of the output (but only polylogarithmically on the time bound T).

Definition 36 (Succinct Randomized Encoding for Turing machines [31])**.** *A* succinct randomized encoding *scheme for Turing machines in the CRS model is succinct if it has the following efficiency:*

- *For every security parameters* $\lambda \in \mathbb{N}$*,* $m, n, T, l \in \mathbb{N}$*, Turing machine* $\Pi \in \mathcal{M}_\lambda$ *and input* x*, such that,* $|\Pi| \leq m$*,* $|x| \leq n$*, and* $|\Pi^T(x)| \leq l$*, every* $(pk, \mathsf{crs}) \leftarrow$

$\mathsf{Setup}(1^\lambda, 1^m, 1^n, 1^T, 1^l)$ *and every encoding* $\hat{\Pi}_x \leftarrow \mathsf{Enc}(1^\lambda, \Pi, x, T)$*, it holds*

$$\mathsf{Time}_{\mathsf{Setup}}(1^\lambda, 1^m, 1^n, 1^T, 1^l) = \mathrm{poly}(\lambda, m, n, T, l)$$

$$\mathsf{Time}_{\mathsf{Enc}}(pk, \Pi, x) = \ell \cdot \mathrm{poly}(\lambda, |\Pi|, |x|, \log T)$$

$$\mathsf{Time}_{\mathsf{Eval}}(\hat{\Pi}, \hat{x}, \mathsf{crs}) = \mathrm{poly}(\lambda, m, n, T)$$

We finally consider another notion of RE that is weaker than sublinear-compactness, where we allow the encoding time to be polynomially dependent on the time bound T, but still require the encoding size be sub-linear in T. We call such RE schemes *weakly sublinear compact*.

Definition 37 (Weakly Sublinear Compact Randomized Encoding scheme)**.** *We say a randomized encoding scheme* $\mathsf{RE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval})$ *in the CRS model for a class of Turing machines* $\{\mathcal{M}_\lambda\}$ *is* weakly sublinear compact *if the efficiency requirement on* Enc *in Definition 36 is changed to: For some constant* $\varepsilon \in (0, 1)$*,*

$$\mathsf{Time}_{\mathsf{Enc}}(pk, \Pi, x) = \mathrm{poly}(\lambda, |\Pi|, |x|, T)$$

$$\mathsf{outlen}_{\mathsf{Enc}}(pk, \Pi, x) = T^{1-\epsilon} \cdot \mathrm{poly}(\lambda, |\Pi|, |x|)$$

Next, we observe that RE schemes satisfying the notions defined above (*i.e.* succinctness and weak sublinear compactness) can be composed to get a RE scheme satisfying sub-linear compactness. In particular, by composing a succinct RE scheme with a weakly compact RE scheme, one can obtain a sub-linearly compact RE scheme.

Theorem 27. *Assume the existence of pseudorandom generators. If there is a* succinct *RE scheme and a* weakly sublinear compact *RE scheme for Turing machines, then there is a* sub-linearly compact randomized encoding *scheme for Turing machines.*

Proof. Let $\mathsf{RE}_1 = (\mathsf{Setup}_1, \mathsf{Enc}_1, \mathsf{Eval}_1)$ be a succinct RE scheme, and $\mathsf{RE}_2 = (\mathsf{Setup}_2, \mathsf{Enc}_2, \mathsf{Eval}_2)$ be a weakly sublinear compact RE scheme. Our sub-linearly compact scheme $\mathsf{RE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval})$ is as follows: the encoding of a machine Π is a succinct encoding of a machine G that itself computes a weakly compact encoding of Π. Weak compactness ensures the output length of G is sub-linear in ℓ (*i.e.* the output length of Π) and hence the time to encode G is also sub-linear in ℓ. Details follow.

$$
\begin{aligned}
\mathsf{Setup}(1^\lambda, 1^m, 1^n, 1^T, 1^l) \quad &: \quad \text{Output } (pk_1, \mathsf{crs}_1, pk_2, \mathsf{crs}_2) \text{ where} \\
&\qquad (pk_1, \mathsf{crs}_1) \leftarrow \mathsf{Setup}_1(1^\lambda, 1^{m_G}, 1^n, 1^{T_G}, 1^{l_G}) \\
&\qquad (pk_2, \mathsf{crs}_2) \leftarrow \mathsf{Setup}_2(1^\lambda, 1^m, 1^n, 1^T, 1^l) \\
\mathsf{Enc}(pk_1, pk_2, \Pi, x) \quad &: \quad s \xleftarrow{\$} \{0,1\}^\lambda \\
&\qquad \hat{G}_x \xleftarrow{\$} \mathsf{Enc}_1(pk_1, G, x) \\
&\qquad \text{where } G[\lambda, \Pi, s](x) = \mathsf{Enc}_2(pk_2, \Pi, x, ; \ \mathsf{PRG}(s)) \\
\mathsf{Eval}(\mathsf{crs}_1, \mathsf{crs}_2, \hat{G}_x) \quad &: \quad \hat{\Pi}_x = \mathsf{Eval}_1(\mathsf{crs}_1, \hat{G}_x), \ y = \mathsf{Eval}_2(\mathsf{crs}_2, \hat{\Pi}_x)
\end{aligned}
$$

where T_G, m_G, l_G are upper bounds on the run-time, description size and output length of G, which can be efficiently calculated using similar bounds on Enc_2 and PRG.

It was shown in [103, 10] that randomized encoding schemes with simulation security are composable, *i.e.* the composed scheme defined above is also simulation secure. We note that their proof essentially goes through in the CRS model too. For concreteness, the simulator is as follows, where Sim_1 and Sim_2

are the simulators for RE_1 and RE_2 respectively:

$$\mathsf{Sim}(1^\lambda, \Pi^T(x), 1^{|\Pi|}, 1^{|x|}, 1^m, 1^n, 1^T, 1^l): \text{ Output } (pk_1, \mathsf{crs}_1, pk_2, \mathsf{crs}_2, \hat{G}_x) \text{ where}$$
$$(pk_2, \mathsf{crs}_2, \hat{\Pi}_x) \leftarrow \mathsf{Sim}_2(1^\lambda, \Pi^T(x), 1^{|\Pi|}, 1^{|x|}, 1^m, 1^n, 1^T, 1^l)$$
$$(pk_1, \mathsf{crs}_1, \hat{G}_x) \leftarrow \mathsf{Sim}_1(1^\lambda, \hat{\Pi}_x, 1^{|G|}, 1^{|x|}, 1^{m_G}, 1^n, 1^{T_G}, 1^{l_G})$$

The output of Sim is indistinguishable from the real encodings, by a hybrid argument. We sketch the order of the hybrids as follows: starting with the real encoding, we first replace $(pk_1, \mathsf{crs}_1, \hat{G}_x)$ with a simulation using Sim_1 on desired output $\hat{\Pi}_x$, where $\hat{\Pi}_x$ is honestly generated using Enc_2 and randomness from $\mathsf{PRG}(s)$. Next, we replace $\hat{\Pi}_x$ so that it is generated using Enc_2, but using fresh randomness instead of that from $\mathsf{PRG}(s)$. Finally, we replace $\hat{\Pi}_x$ (and also (pk_2, crs_2)) with a simulation using Sim_2 on desired output $\Pi^T(x)$. By the security of Sim_1, PRG and Sim_2, these hybrids are indistinguishable.

It remains to analyze the efficiency. By construction, the time complexity of Enc is:

$$\mathsf{Time}_{\mathsf{Enc}}(pk_1, pk_2, \Pi, x) = \mathsf{Time}_{\mathsf{Enc}_1}(pk_1, G, x) = \ell_G \cdot \mathrm{poly}(\lambda, m_G, n, \log T_G)$$

Note that by the definition of G,

$$\begin{aligned}
\ell_G &= \mathsf{outlen}_{\mathsf{Enc}_2}(pk_2, \Pi, x) = T^{1-\epsilon} \cdot \mathrm{poly}(\lambda, |\Pi|, |x|) \\
T_G &= \mathsf{Time}_{\mathsf{Enc}_2}(pk_2, \Pi, x) = \mathrm{poly}(\lambda, |\Pi|, |x|, T) \\
m_G &= |G[\lambda, \Pi]| = \mathrm{poly}(\lambda, |\Pi|)
\end{aligned}$$

Hence, we obtain that

$$\begin{aligned}
\mathsf{Time}_{\mathsf{Enc}}(pk_1, pk_2, \Pi, x) &= ł_G \cdot \mathrm{poly}(\lambda, m_G, n, \log T_G) \\
&= T^{1-\epsilon} \cdot \mathrm{poly}(\lambda, |\Pi|, |x|) \cdot \mathrm{poly}(\lambda, m_G, n, \log T_G).
\end{aligned}$$

Bringing all this together, using that m_G is polynomial is $\text{poly}(\lambda, |\Pi|)$, and choosing appropriate ϵ' with $0 < \epsilon' < \epsilon$, we have that:

$$\mathsf{Time}_{\mathsf{Enc}}(pk_1, pk_2, \Pi, x) \quad = \quad \text{poly}(\lambda, |\Pi|, |x|) \cdot T^{1-\epsilon'}.$$

This concludes the theorem. □

4.6.3 Randomized encodings with CRS from Compact Functional Encryption

In this section we construct RE schemes in the CRS model from Compact Functional encryption schemes and pseudorandom generators.

Let $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ be a public key, compact functional encryption scheme for $\mathcal{P}/poly$, and let PRG be a pseudorandom generator. We define a randomized encoding scheme in the CRS model $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval})$ as follows.

The setup algorithm $\mathsf{Setup}(1^\lambda, 1^m, 1^n, 1^T, 1^l)$:

- Setup first generates keys for the functional encryption scheme $(mpk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda)$ and samples a uniformly random string $s \leftarrow \{0,1\}^\lambda$.
- Next, it generates the string $c \leftarrow 0^l \oplus \mathsf{PRG}(s, l)$. That is, it encrypts 0^l using a one-time pad with the key coming from $\mathsf{PRG}(s, l)$
- Let U be the universal circuit that on input (Π, x) where $|\Pi| \leq m$ and $|x| \leq n$ runs machine Π on x for at most T steps and outputs the first l bits of the tape as output. We define a circuit $C_{U,c}$, that has the string c and circuit U hardcoded in it, as follows.

1. $C_{U,c}$ takes as input (Π, x, s', b) where (Π, x) satisfies the size constraints as described above, $s' \in \{0,1\}^\lambda$ and $b \in \{0,1\}$.
2. If $b = 0$ then $C_{U,c}$ outputs $U(\Pi, x)$.
3. Otherwise $C_{U,c}$ outputs $c \oplus \mathsf{PRG}(s')$.

- Setup runs $sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C_{U,c})$ and outputs sk_C as the common reference string crs and mpk as the public encoding key pk

The encoding algorithm $\mathsf{Enc}(pk, \Pi, x)$**:** Enc parses pk as the functional public key mpk and runs $ct \leftarrow \mathsf{FE.Enc}(mpk, (\Pi, x, 0^\lambda, 0))$. Enc outputs the functional ciphertext ct as the encoding $\hat{\Pi}_x$.

The evaluation algorithm $\mathsf{Eval}(\hat{\Pi}_x, \mathsf{crs})$: Eval parses $\hat{\Pi}_x$ as a functional ciphertext ct and crs as the functional secret key $sk_{C_{U,c}}$. Eval runs $y \leftarrow \mathsf{FE.Dec}(sk_{C_{U,c}}, ct)$ and outputs y.

The correctness of the above encoding scheme follows directly from that of the underlying functional encryption scheme. When a randomized encoding of (Π, x) is evaluated, it outputs the result of running the universal circuit U on (Π, x) that is $\Pi^T(x)$. Also the efficiency properties of the above scheme follow directly from the compactness properties of the functional encryption scheme. For example, if the functional encryption scheme we start from has sub-linear compactness (the ciphertext size is sub-linear in the circuit size of the function for which the functional secret keys are generated) then we get an encoding scheme with sub-linear compactness.

We have the following theorem.

Theorem 28. *Let* $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *be a public key functional encryption scheme for* $\mathcal{P}/poly$ *with* $(\lambda_0, S(\cdot))$ *selective security, and let* PRG *be a pseudorandom*

generator with $(\lambda_0, S(\cdot))$ *security. The randomized encoding scheme defined above is* $(\lambda_0, \frac{S(\cdot)}{4})$*-simulation secure.*

Corollary 1. *If there exists a public key, compact (resp. succinct, weakly sublinear compact) functional encryption for* P/poly *scheme with selective security, and a secure PRG, then there exists a compact (resp. succinct*[8]*, weakly sublinear compact) randomized encoding scheme for Turing machines in the CRS model that is simulation secure.*

Proof. Let (Setup, Enc, Eval) be the randomized encoding scheme as defined above. We need to show there exists a PPT algorithm Sim that, when given the output of a machine on some input (together with the bound parameters of the machine and input), simulates an encoding that is indistinguishable from the real encoding of the machine and input. We define Sim as follows. **The simulator** $\mathsf{Sim}(1^\lambda, 1^m, 1^n, 1^T, y, 1^{|\Pi|}, 1^{|x|})$**:** Here we denote

- Sim first generates functional keys $(mpk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda)$ and a random string $s \leftarrow \{0,1\}^\lambda$.
- Next, Sim generates the string $c \leftarrow y \oplus \mathsf{PRG}(s, |y|)$. That is, it encrypts the output y under a one-time pad with the key coming from $\mathsf{PRG}(s, |y|)$.
- Sim runs $sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C_{U,c})$ where $C_{U,c}$ is same as in the construction of the encoding scheme, except with c being the string generated in the above step.
- Sim generates $ct \leftarrow \mathsf{FE.Enc}(mpk, (0^{|\Pi|}, 0^{|x|}, s, 1))$.
- Sim outputs sk_C as the simulated crs, mpk as the public encoding key pk and ct as simulated machine encoding $\hat{\Pi}_x$.

[8]We note that for succinct RE, we first apply the transformation from succinct FE to get succinct RE with 1-bit output, and to encode Turing Machines with multi-bit outputs, we generate one such RE for each output bit

Now we formally prove that the above simulator is secure. Consider any ensemble $\{\Pi_\lambda, x_\lambda, m_\lambda, n_\lambda, l_\lambda, T_\lambda\}$ where $\Pi_\lambda \in \mathcal{M}_\lambda$ and $|\Pi_\lambda|, |x_\lambda|, m_\lambda, n_\lambda, l_\lambda, T_\lambda \leq B(\lambda)$ for some polynomial B. Subsequently, we suppress the security parameter in the subscript.

We need to show that for every $\lambda > \lambda_0$,

$$D_0 = \left(\begin{array}{c} (pk, \mathsf{crs}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^{m(\lambda)}, 1^{n(\lambda)}, 1^{T(\lambda)}) \\ (\hat{\Pi}_x) \xleftarrow{\$} \mathsf{Enc}(pk, \Pi, x) \end{array} : \mathsf{crs}, pk, \hat{\Pi}_x \right)$$

$$D_1 = \left((pk, \mathsf{crs}, \hat{\Pi}_x) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, 1^{m(\lambda)}, 1^{n(\lambda)}, 1^{T(\lambda)}, \Pi^T(x), 1^{|\Pi|}, 1^{|x|}) : \mathsf{crs}, pk, \hat{\Pi}_x \right)$$

are $\frac{S(\lambda)}{4} - B(\lambda)^d$ indistinguishable, where $\lambda_0, S(\cdot)$ is the security of functional encryption scheme, and d is some constant. We show this by a hybrid argument as follows.

- Let H_0 be the distribution of the real encoding D_0. Rewriting H_0 in terms of the underlying primitives we have

$$H_0 = \left(\begin{array}{c} (mpk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda) \\ s \leftarrow \{0,1\}^\lambda \\ c \leftarrow 0^l \oplus \mathsf{PRG}(s, l) \\ sk_{C_{U,c}} \leftarrow \mathsf{FE.KeyGen}(msk, C_{U,c}) \\ ct \leftarrow \mathsf{FE.Enc}(mpk, (\Pi, x, 0^\lambda, 0) \end{array} : mpk, sk_{C_{U,c}}, ct \right)$$

- Let H_1 be a hybrid distribution exactly as above, except that c is generated as $c \leftarrow 0^l \oplus R$ where R is uniformly random from $\{0,1\}^l$. We claim H_0 and H_1 are $S(\lambda) - B(\lambda)^{d'}$ indistinguishable, where d' is some constant. This follows from the security of the pseudorandom generator. Any adversary A that distinguishes H_0 and H_1 can be turned into an adversary A' that breaks the security of the pseudorandom generator with the same advantage. A' has

Π, x, T hard-coded and needs to run the setup and generation algorithms of the functional encryption schemes. Therefore the size of A' is

$$\mathsf{size}(A) + \mathrm{poly}(B(\lambda)) = (S(\lambda) - B(\lambda)^{d'}) + B(\lambda)^{d'} = S(\lambda)$$

Hence A' breaks the $(\lambda_0, S(\cdot))$ security of the pseudorandom generator and we have a contradiction.

- Let H_2 be a hybrid distribution just as above except that c is generated from the output y as $c \leftarrow y \oplus R$ where, as before, R is uniformly random from $\{0,1\}^l$. Since R is uniformly random, the distributions H_2 and H_1 are identical.
- Let H_3 be a hybrid distribution just as above except that the one-time pad key is generated using the pseudorandom generator. That is, $c \leftarrow y \oplus \mathsf{PRG}(s, l)$. As before, the distributions H_3 and H_2 are $S(\lambda) - B(\lambda)^{d'}$ indistinguishable, where d' is some constant.
- Let H_4 be a hybrid distribution just as above except that ct is generated as

$$ct \leftarrow \mathsf{FE.Enc}(mpk, (0^{m(\lambda)}, 0^{n(\lambda)}, s, 1))$$

This is exactly the distribution as generated by the simulator Sim. We claim H_3 and H_4 are $S(\lambda) - B(\lambda)^{d''}$ indistinguishable where d' is some constant. This follows from the security of the functional encryption scheme. Any adversary A that distinguishes H_3 and H_4 can be turned into an adversary A' that breaks the security of the functional encryption scheme with the same advantage. A' selects the challenge messages as $(0^{m(\lambda)}, 0^{n(\lambda)}, s, 1)$ and $(\Pi, x, 0^\lambda, 0)$ and secret key query $C_{U,c}$ which has the same output y on both messages. A' needs to additionally run the pseudorandom generator. Hence the size of A' is $\mathsf{size}(A) + B(\lambda)^{d'}$ for some constant d', which as before

results in size $S(\lambda)$. Hence A' breaks the $(\lambda_0, S(\cdot))$ security of the functional encryption scheme and we have a contradiction.

Each of the hybrid distribution pairs above are $S(\lambda) - B(\lambda)^{d'}$ indistinguishable. By a simple hybrid argument we have that the distributions $D_0 = H_0$ and $D_1 = H_4$ are $\frac{S(\lambda)-B(\lambda)^{d'}}{4} \geq \frac{S(\lambda)}{4} - B(\lambda)^{d}$ indistinguishable, for some constant d, hence completing the proof.

□

The above theorem and corollary also work in the regime of sub-exponential security. That is, starting with a functional encryption scheme and pseudorandom generator that are sub-exponentially secure we obtain a RE scheme with sub-exponential security.

The following corollary is obtained by combining Corollary 1 with Theorem 22 and Theorem 23. While we use this corollary in our results, we believe it is of independent interest too. Succinct RE schemes for Turing machines were shown by [31] to have a variety of applications. However the only known construction of it ([112]) relies on $i\mathcal{O}$ for circuits. We observe that in the CRS model, succinct RE schemes can be based simply on LWE.

Corollary 2. *Assuming LWE (resp. with sub-exponential hardness), there exists a succinct RE scheme for Turing machines in the CRS model with (resp. sub-exponential) simulation security.*

Finally, the following corollary shows that, assuming LWE, weakly sublinear compact FE is sufficient to construct sublinearly-compact RE in the CRS model.

This corollary follows by combining Corollary 1, which shows that weakly sublinear compact FE implies weakly sublinear compact RE in the CRS model, Corollary 2, which constructs succinct RE in the CRS model from LWE, and finally Theorem 27, which shows that weakly sublinear compact RE and succinct RE can be combined to produce sublinearly-compact RE in the CRS model.

Corollary 3. *Assuming LWE (resp. with sub-exponential hardness), if there exists a weakly sublinear compact FE scheme for* P/poly *(resp. with sub-exponential security), then there exists a sublinearly-compact RE scheme for Turing machines in the CRS model with (resp. sub-exponential) simulation security.*

4.6.4 IO for Circuits from RE in the CRS model

In this section we show that compact RE schemes for Turing machines in the CRS model implies *iO* for circuits; combining with the result of [112] that *iO* for circuits implies *iO* for (bounded-input) Turing machines, we obtain the following theorem:

Theorem 29. *Assume the existence of sub-exponentially secure one-way functions. If there exists a sublinearly compact randomized encoding scheme in the CRS model with sub-exponential simulation security, then there exists an bounded-input indistinguishability obfuscator for Turning machines.*

We note that the theorem also holds w.r.t. sublinearly compact randomized encoding scheme in the CRS model, satisfying, weaker, distributional indistinguishability security, *with auxiliary inputs* (i.e., Definition 34 w.r.t. distributions $\{D_{b,\lambda}\}$ that additionally samples an auxiliary input z_b, and the security requirement is that if the output distributions together with the auxiliary inputs are in-

distinguishable, then the encodings together with the auxiliary inputs are also indistinguishable, with appropriate security loss). Since the distributional indistinguishability security is implied by simulation security, and in the CRS model, we can construct sublinearly compact RE with simulation security from sublinearly compact FE schemes, for simplicity, we directly state and prove the theorem w.r.t. simulation security.

The construction and proof is very similar to that of unbounded-input *iO* from compact RE schemes in the plain model presented in Section 4.4.

Proof sketch for Theorem 29. We first briefly recall the main ideas behind the unbounded-input *iO* construction. The unbounded-input *iO* generates an encoding of a recursively defined Turing machine $\Pi_{\epsilon,R}$ (where ϵ is the empty string and R is uniformly random). $\Pi_{s,R}$ generates encodings of Π_{s0,R_0}, Π_{s1,R_1} and of the machine to be obfuscated with input s, where R_0 and R_1 are pseudorandom strings derived from R. For every such machine $\Pi_{s,R}$, we refer to $|s|$ as the *level* of the machine. Evaluating the obfuscation on an input of length n involves evaluating an encoding of a machine at every level $i \in \{0, \ldots, n\} = [n]$.

Construction: Our circuit obfuscator *iO* gets as input the security parameter 1^λ and the circuit to obfuscate C. Let n be the input length of C. To use RE schemes in the CRS model, *iO* first generates a RE setup (pk_i, crs_i) for every level $i \in [n]$. The obfuscation consists of $\vec{\mathsf{crs}} = \{\mathsf{crs}_i\}_{i=0}^n$, and an encoding of the machine $\Pi_{\vec{pk}_1,C,\epsilon,R}$ which has public keys $\vec{pk}_1 = \{pk_i\}_{i=1}^n$ hardcoded (more generally $\vec{pk}_i$ denotes $\left\{pk_j\right\}_{j=i}^n$). For every level $i < n$ and $s \in \{0,1\}^i$, the machine $\Pi_{\vec{pk}_{i+1},C,s,R}$ uses pk_{i+1} to generate encodings of $\Pi_{\vec{pk}_{i+2},C,s0,R_0}$ and $\Pi_{\vec{pk}_{i+2},C,s1,R_1}$. When $i = n$, the machine $\Pi_{\vec{pk}_{i+1},C,s,R}$ simply outputs $C(s)$. To evaluate the obfuscation on an input x, one evaluates an encoding of the machine $\Pi_{\vec{pk}_{i+1},C,x[1\ldots i],R}$ at every level $i \in [n]$,

using $\mathsf{crs}_i \in \vec{\mathsf{crs}}$.

Note that, just as in the unbounded-input *iO* construction (Lemma 5), the compactness of the RE ensures that the size of the encodings at each level is some fixed polynomial in the security parameter and the circuit size. The obfuscation additionally contains a crs_i for every level $i \in [n]$ where the length of crs_i is polynomial in the running time of machines at that level (*i.e.* the time taken to encode machines at the next level), which by the same argument (Lemma 5) is some fixed polynomial in the security parameter and the circuit size. All in all, the size of the obfuscated circuit is polynomial in the security parameter and the circuit size.

Security: We need to show that, for any pair of functionally equivalent circuits C_0 and C_1, the joint distribution $(\vec{\mathsf{crs}}, \tilde{\Pi}_{\vec{pk}_1,\epsilon,C_0,R})$ is indistinguishable from $(\vec{\mathsf{crs}}, \tilde{\Pi}_{\vec{pk}_1,\epsilon,C_1,R})$. Just as in the proof of Theorem 26, we prove a stronger statement by induction. We claim that for every level $i \in [n]$, and every $s \in \{0,1\}^i$, the joint distribution $(\tilde{\Pi}_{\vec{pk}_{i+1},s,C_0,R}, \vec{\mathsf{crs}}_i, \vec{pk}_i)$ is indistinguishable from $(\tilde{\Pi}_{\vec{pk}_{i+1},s,C_1,R}, \vec{\mathsf{crs}}_i, \vec{pk}_i)$.

When $i = n$ (the base case), the above distributions are indistinguishable since the output of $\Pi_{\vec{pk}_{i+1},s,C_0,R}$ (in this case $C_0(s)$) is identical to the output of $\Pi_{\vec{pk}_{i+1},s,C_1,R}$. Hence by the indistinguishability security of the RE scheme (which is implied by simulation security, see Remark 8), $(\tilde{\Pi}_{\vec{pk}_{n+1},s,C_0,R}, pk_n, \mathsf{crs}_n)$ is indistinguishable from $(\tilde{\Pi}_{\vec{pk}_{n+1},s,C_1,R}, pk_n, \mathsf{crs}_n)$.

For the inductive step, we show the distributions are indistinguishable at any level $i < n$ assuming they are indistinguishable at level $i+1$. We do this by a hybrid argument as follows. Let H_0 be the joint distribution of the encoding at level i with C_0 hardcoded, with $\vec{\mathsf{crs}}_i$ and $\vec{pk}_i$. Writing this a bit differently, we

have,

$$H_0 = ((\tilde{\Pi}_{\vec{pk}_{i+1},s,C_0,R}, \mathsf{crs}_i, pk_i), \vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1})$$

Next, we define hybrid distribution H_1 as follows.

$$H_1 = (\mathsf{Sim}(out(\tilde{\Pi}_{\vec{pk}_{i+1},s,C_0,R})), \vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1})$$

where Sim is the simulator for the RE scheme (for the sake of brevity in this proof sketch, we omit the other inputs to the simulator). By simulation security, H_0 and H_1 are indistinguishable.

Next, we define hybrid distribution H_2 by changing the underlying circuit to C_1.

$$H_2 = (\mathsf{Sim}(out(\tilde{\Pi}_{\vec{pk}_{i+1},s,C_1,R})), \vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1})$$

To show H_2 and H_1 are indistinguishable, we show that the following distributions are indistinguishable.

$$(out(\Pi_{\vec{pk}_{i+1},s,C_0,R}), \vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1}) \approx (out(\Pi_{\vec{pk}_{i+1},s,C_1,R}), \vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1})$$

This follows from the induction hypothesis as follows. Recall that the output of $\Pi_{\vec{pk}_{i+1},s,C_0,R}$ is a pair of level $i+1$ encodings $(\tilde{\Pi}_{\vec{pk}_{i+2},s0,C_0,R_0}, \tilde{\Pi}_{\vec{pk}_{i+2},s1,C_0,R_1})$. By the induction hypothesis, each of these encodings is indistinguishable from one with C_1 hardcoded, in the presence of $(\vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1})$. By a simple hybrid argument, the above indistinguishability holds, and hence H_2 is indistinguishable from H_1.

Finally, we define hybrid distribution H_3 which contains the encoding at level i with C_1 hardcoded.

$$H_3 = ((\tilde{\Pi}_{\vec{pk}_{i+1},s,C_1,R}, \mathsf{crs}_i, pk_i), \vec{\mathsf{crs}}_{i+1}, \vec{pk}_{i+1})$$

By the simulation security of the RE scheme H_3 is indistinguishable from H_2. Hence, by a hybrid argument H_0 is indistinguishable from H_3 hence completing the inductive step. □

4.6.5 Summary of Results using RE in the CRS model

We observe that by combining Theorem 29 with Corollary 1, we reprove the results of [4, 33]

Theorem 30. *Assuming the existence of compact functional encryption with subexponential security, there exists a bounded-input indistinguishability obfuscator for Turing Machines.*

Further, we get the following new result, as a consequence of Corollary 3 and Theorem 29:

Theorem 31. *Assuming the existence of weakly sublinear compact functional encryption with subexponential security and LWE with subexponential security, there exists a bounded-input indistinguishability obfuscator for Turing Machines.*

4.7 Impossibility of Compact RE

In this section, we show several impossibility results related to sublinear (and hence compact) RE with different security:

Theorem 32. *The following impossibility results hold in the* plain *model:*

1. *Sublinear randomized encoding schemes with (polynomial) simulation security do not exist, assuming one-way functions.*
2. *Sublinear randomized encoding schemes with* sub-exponential *indistinguishability security do not exist, assuming sub-exponentially secure one-way functions.*
3. *Sublinear randomized encoding schemes with (polynomial) indistinguishability security do not exist, assuming bounded-input* $i\mathcal{O}$ *for Turing machines and one-way functions.*

Next we proceed to prove Theorem 32.

Impossibility 1. The impossibility of sublinear RE with simulation security follows from standard techniques that leverages the sublinear-size of the encoding to derive a contradiction to the imcompressibility of pseudo-random strings; below, we provide a proof sketch.

Proof Sketch. We argue that assuming one-way functions, compact RE with simulation security does not exist in the plain model. Suppose not and there is a compact RE that admits a simulator Sim which on input the output $y = \Pi(x)$ can simulate an encoding $(\tilde{\Pi}, \tilde{x})$ that is indistinguishable from an honestly generated encoding $(\hat{\Pi}, \hat{x})$. By the indistinguishability, it follows that evaluating $(\tilde{\Pi}, \tilde{x})$ yields the output y. Now, consider the specific computation of evaluating a PRG G on a short random seed s, that is, $\Pi = G$ and $x = s$. The simulator Sim on the pseudo-random output $y = G(s)$, produces $(\tilde{G}, \tilde{s})$, that can be evaluated to generate y. By the pseudo-randomness of PRG, it follows that Sim on input a truly random string y', can also output a tuple $(\tilde{G}', \tilde{s}')$ that evaluates to y', and

the length of the tuple is sublinear in the length of y'. However, this contradicts the imcompressibility of random strings. □

Impossibility 2. The impossibility of sub-exponentially (ind-)secure sublinear RE is implied by impossibility 3, and the fact that bounded-input *iO* for Turing machines can be constructed from sub-exponentially (ind-)secure sublinear RE. More precisely, by our construction in Section 4.5, sub-exp secure sublinear-RE (and one-way functions) imply sub-exp secure *iO* for circuits; combined with the result of [112] that sub-exp secure *iO* for circuits (and one-way functions) imply (polynomially secure) bounded-input *iO* for Turing machines, we have that assuming sub-exp secure one-way functions,

$$\text{Sub-exp (ind-)secure Sublinear RE} \Longrightarrow \text{Bounded-input } iO \text{ for TMs}$$

On the other hand, impossibility 3 states that

$$\text{Bounded-input } iO \text{ for TMs} \Longrightarrow \textbf{NO} \text{ (ind-)secure Sublinear RE}$$

Therefore, if impossibility 3 is true, sub-exponentially secure sublinear RE is impossible assuming sub-exp secure one-way functions.

Impossibility 3. We show that (poly-secure) sublinear RE does not exist assuming the following two primitives.

- A bounded-input *iO* for Turing machines iO, which on input $(1^\lambda, 1^n, R, T)$ runs in time

$$\mathsf{Time}_{iO}(1^\lambda, 1^n, R, T) \leq \mathrm{poly}(\lambda, n, |R|, \log T) .$$

- A pseudo-random generator PRG that on input a seed s of length λ and a length k outputs a string of length k in time $\text{poly}(\lambda, k)$. This is implied by the existence of one-way functions.

The efficiency of iO and PRG means that there is a constant d, such that, the following holds:

- For every $\lambda \in N$, $\ell = \ell(\lambda)$, program R with size $|R| \leq \lambda + \ell$, input length $n = \ell/2$, and $T = 2^\lambda$, the run-time of iO is

$$\mathsf{Time}_{iO}(1^\lambda, 1^{\ell/2}, R, 2^\lambda) \leq (\lambda\ell)^d \ , \tag{4.1}$$

- and for every string $s \in \{0,1\}^\lambda$, and $k \in N$, the run-time of the PRG is

$$\mathsf{Time}_{\mathsf{PRG}}(s, k) \leq (\lambda k)^d \ . \tag{4.2}$$

Assuming such iO and PRG, we first show that there does not exist sublinear RE that are sufficiently compact in the following sense:

Claim 4. *Let d be a constant defined as above w.r.t. iO and* PRG*. There is no sublinear RE with time complexity satisfying the following:*

$$\mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi, x, T) \leq \text{poly}(\lambda, |\Pi|, |x|)T^\alpha, \quad \textit{for } \alpha \leq 1/2d \ . \tag{4.3}$$

Before proving the claim, we argue that the above claim suffices for ruling out the existence of any sublinear RE schemes. By Claim 4 in Section 4.3.4, given any sublinear RE with complexity scaling with T^β for an arbitrary constant $\beta > 0$, one can reduce the time complexity to scaling with T^α for an arbitrary smaller constant α, by recursively composing it for some constant times. Combining this fact with the above claim, we conclude that assuming bounded-input iO for TMs and one-way function, sublinear RE do not exist.

Proof of Claim 4. Assume for contradiction that there is a sublinear RE scheme $\mathsf{RE} = (\mathsf{Enc}, \mathsf{Eval})$ whose time complexity depends on T^α with a sufficiently small $\alpha \leq 1/2d$ (and some multiplicative polynomial factor $\mathrm{poly}(\lambda, |\Pi|, |x|)$). Below, we derive a contradiction by constructing two ensembles of distributions $\{D_{0,\lambda}\}$, $\{D_{1,\lambda}\}$ that both sample triplets of form (Π, x, T); we show that (1) the outputs of the program and input sampled from these two distributions are indistinguishable, yet (2) the encoding of the program and input are distinguishable with probability close to 1. This violates the ind-security of RE, and gives a contradiction.

Let $\ell = \ell(\lambda)$ be a sufficiently large polynomial in λ, whose magnitude will become clear in the description below.

Distribution $D_{b,\lambda}$ samples triplet $(\Pi_b, 0, T)$ as follows:

- Sample two seeds $s \xleftarrow{\$} \{0,1\}^\lambda$, and $u \xleftarrow{\$} \{0,1\}^\lambda$.
- Turing machine $\Pi_b[\lambda, s, u]$ (with (λ, s, u) hardwired in), on input 0, proceeds in two steps
  1. Compute $\mathsf{PRG}(s, \ell) = y$, and $\mathsf{PRG}(u, \Gamma) = r$ where $\Gamma = (\lambda\ell)^d$.
  2. Obfuscate the program $R_b[b, y]$ described in Figure 4.1 to obtain
  $$\hat{R}_b = iO(1^\lambda, 1^{\ell/2}, R_b, 2^\lambda \; ; \; r) \; .$$
  (The input length of R_b is bounded by $\ell/2$ and its run time is bounded by 2^λ.)
  3. Output $(y, \hat{R}_b)$.
- Set $T = 2(\lambda\Gamma)^d$.

  Note that T is an upper bound on the run-time of Π_b, as the first step of Π_b

takes at most $(\lambda\ell)^d + (\lambda\Gamma)^d$ *(according to condition* (4.2)*), and the second step takes at most* $(\lambda\ell)^d$ *steps (according to condition* (4.1)*).*

The program $R_b[b, y]$

Hardwired Values [b, y]: b is in $\{0, 1, \perp\}$, and y an ℓ-bit length string.
Input ($\hat{\Pi}$, $\hat{x}$): encoding of a program and an input, s.t. $|\hat{\Pi}| + |\hat{x}| \leq \ell/2$.
The program proceeds as follow:

- Compute $z \leftarrow \mathsf{Eval}(\hat{\Pi}, \hat{x})$.
- Output b if the first ℓ bits of z equals to y, otherwise output $\perp$.

Figure 4.1: The program R_b used in the proof of impossibility of sublinear RE in the plain model.

We first show that the distributions, $out_{0,\lambda}$ and $out_{1,\lambda}$, of outputs of the program and input sampled from $D_{0,\lambda}$ and $D_{1,\lambda}$ are indistinguishable

$$\begin{aligned} out_{b,\lambda} &= \left((\Pi_b, 0, T) \xleftarrow{\$} D_{b,\lambda} : \Pi_b^T(0), T, |\Pi_b|, |0|\right) \\ &= \Big(s, u \xleftarrow{\$} \{0,1\}^\lambda, y = \mathsf{PRG}(s, \ell), r = \mathsf{PRG}(u, \Gamma), \\ &\qquad \hat{R}_b = iO(1^\lambda, 1^{\ell/2}, R_b[b, y], 2^\lambda \; ; \; r) \; : \; (y, \hat{R}_b), T, |\Pi_b|, |0|\Big) \end{aligned}$$

Towards this, consider the following hybrid distributions.

Distribution $H_{b,\lambda}$ samples output tuple in the same way as $out_{b,\lambda}$ does, except that the pseudo-random strings y and r are replaced with truly random strings $\tilde{y} \xleftarrow{\$} \{0,1\}^\ell$ and $\tilde{r} \xleftarrow{\$} \{0,1\}^\Gamma$. More precisely,

$$\begin{aligned} H_{b,\lambda} = \Big(&\underline{\tilde{y} \xleftarrow{\$} \{0,1\}^\ell, \tilde{r} \xleftarrow{\$} \{0,1\}^\Gamma}, \\ &\tilde{R}_b = iO(1^\lambda, 1^{\ell/2}, R_b[b, \underline{\tilde{y}}], 2^\lambda \; ; \; \underline{\tilde{r}}) \; : \; (\underline{\tilde{y}}, \tilde{R}_b), T, |\Pi_b|, |0|\Big) \end{aligned}$$

By the security of PRG, the above distribution is indistinguishable from $out_{b,\lambda}$.

Distribution $G_{b,\lambda}$ samples output tuple in the same way as $H_{b,\lambda}$ does, except that, instead of obfuscating the program R_b, it obfuscate the program $R_\perp$, which always outputs $\perp$ (and has the same run time as R_b). More precisely,

$$G_{b,\lambda} = \Big(\tilde{y} \xleftarrow{\$} \{0,1\}^\ell, \tilde{r} \xleftarrow{\$} \{0,1\}^\Gamma, \\ \underline{\tilde{R}_\perp = iO(1^\lambda, 1^{\ell/2}, R_\perp[\perp, \tilde{y}], 2^\lambda \; ; \; \tilde{r})} \; : \; (\tilde{y}, \underline{\tilde{R}_\perp}), T, |\Pi_b|, |0|\Big)$$

We claim that it follows from the security of iO that $G_{b,\lambda}$ and $H_{b,\lambda}$ are indistinguishable. Towards this, it suffices to show that with probability $1-2^{-\ell/2}$ over the random choice of $\tilde{y}$, the program $R_b[b, \tilde{y}]$ (obfuscated in $H_{b,\lambda}$) always outputs $\perp$, in which case $R_b[b, \tilde{y}]$ and $R_\perp[\perp, \tilde{y}]$ agree on all inputs and their obfuscation is indistinguishable. By construction, R_b outputs a value that is not $\perp$ only when the input $(\hat{\Pi}, \hat{x})$ satisfies that the first ℓ bits of the string z evaluated from it matches $\tilde{y}$. However, the encoding length is bounded by $\ell/2$, there are at most $2^{\ell/2}$ possible string z; when $\tilde{y}$ is chosen at random, the probability that z agrees with $\tilde{y}$ is at most $2^{-\ell/2}$. Except with this probability, R_b always outputs $\perp$.

On the other hand, we show that the distributions, $enc_{0,\lambda}$ and $enc_{1,\lambda}$, of the encoding of the program and input sampled from $D_{0,\lambda}$ and $D_{1,\lambda}$ are efficiently *distinguishable*.

$$enc_{b,\lambda} = \Big((\Pi_b, 0, T) \xleftarrow{\$} D_{b,\lambda} : (\hat{\Pi}_b, \hat{0}) \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \Pi_b, 0, T)\Big)$$

Towards this, we first show that the length of the encoding $(\hat{\Pi}_b, \hat{0})$ is bounded by $\ell/2$. This is because the run-time of Enc satisfies

$$\begin{aligned} \mathsf{Time}_{\mathsf{Enc}}(1^\lambda, \Pi_b, 0, T) &\leq \mathrm{poly}(\lambda, |\Pi_b|, |0|)T^\alpha \text{ for } \alpha \leq 1/2d \\ &\leq \mathrm{poly}(\lambda)(2(\lambda\Gamma)^d)^{1/2d^2} = \mathrm{poly}(\lambda)\Gamma^{1/2d} \\ &= \mathrm{poly}(\lambda)((\lambda\ell)^d)^{1/2d} \\ &\leq \lambda^c \sqrt{\ell} \leq \ell/2 \end{aligned}$$

where the second line follows from that $|\Pi_b| = O(\lambda)$ and $T = 2(\lambda\Gamma)^d$, the third line follows from that $\Gamma = (\lambda\ell)^d$, and in the last line, c is a constant independent of d and the last inequality holds for sufficiently large $\ell = \ell(\lambda)$.

□

Acknowledgment: We are extremely grateful to Nir Bitansky and Omer Paneth for informing us of their impossibility result for compact RE assuming differing-input obfuscation, SNARKs and collision-resistant hash functions; this results was the inspiration behind our main impossibility result. We are also very grateful to them for many delightful and insightful discussions.

CHAPTER 5

INDISTINGUISHABILITY OBFUSCATION WITH EXPONENTIAL EFFICIENCY

This chapter contains joint work with Huijia Lin (UCSB), Rafael Pass (Cornell University) and Karn Seth (Cornell University).

5.1 Introduction

The goal of *program obfuscation* is to "scramble" a computer program, hiding its implementation details (making it hard to "reverse-engineer"), while preserving the functionality (i.e, input/output behavior) of the program. In recent years, the notion of *indistinguishability obfuscation (iO)* [17, 70] has emerged as the central notion of obfuscation: Roughly speaking, this notion requires that obfuscations $iO(C_1)$, $iO(C_2)$ of any two *functionally equivalent* circuits C_1 and C_2 (i.e., whose outputs agree on all inputs) from some class $\mathcal{C}$ (of circuits of some bounded size) are computationally indistinguishable.

On the one hand, this notion of obfuscation is strong enough for a plethora of amazing applications (see e.g., [134, 42, 40, 67, 31, 54, 112]); on the other hand, it may plausibly exist [70, 16, 129, 85], whereas stronger notion of obfuscations have run into strong impossibility results, even in idealized models (see e.g., [17, 89, 55, 131, 119, 114])

However, despite all these amazing progress, to date, all candidate constructions of iO rely on candidate constructions of *multi-linear maps* [66, 61, 81, 62], all of which have non-trivial attacks [59, 122], and it is not clear to what extent

the security of the obfuscators that rely on them are affected.

In this paper, rather than studying new candidate construction of $i\mathcal{O}$, we are interested in studying *weaker* notions of indistinguishability obfuscation that can amplified/bootstrapped into the standard notion of $i\mathcal{O}$:

> *Identify quantitatively weaker notions of $i\mathcal{O}$ that can be amplified into the "standard" notion of $i\mathcal{O}$.*

Our hope that that doing so will simplify future constructions of $i\mathcal{O}$. This approach is orthogonal to the approach of [70], which studies whether $i\mathcal{O}$ for "weak" *classes of circuits* (e.g., NC^1 circuits) can be bootstrapped to $i\mathcal{O}$ for all polynomial-size circuits, but is similar to e.g., the hardness amplification approach of Yao [137] of amplifying a *weak one-way function* into a (strong) one.)

One initial weakening of $i\mathcal{O}$ appeared (implicitly) in [31], where the authors consider $i\mathcal{O}$ for polynomial-size circuits with $O(\log \lambda)$ *length inputs*, where λ is the security parameter; we refer to this class of circuits as $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ and refer to $i\mathcal{O}$ for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ as *short-input $i\mathcal{O}$*. Short-input $i\mathcal{O}$ is more appealing than standard $i\mathcal{O}$ (for P/poly) in the sense that it can be efficiently checked whether an attack on a candidate scheme succeeds [124] (an attacker needs to come up with two circuits C_1, C_2 that are functionally equivalent for which it can distinguish obfuscations; checking whether two circuits are functionally equivalent may be hard in general, but becomes efficient if the circuits are restricted to inputs of length $O(\log \lambda)$ by simply enumerating all inputs). Additionally, [31] show that for *some* (but far from all) applications of $i\mathcal{O}$, this weaker notion actually suffices. But it is not known whether this weaker notion of $i\mathcal{O}$ implies "standard $i\mathcal{O}$"; we shall return to this question shortly.

Inefficient *iO* : We here consider a further weakening of short-input *iO* . Recall that indistinguishability obfuscators with running time

$$T_0(|C|, \lambda, n) = \text{poly}(|C|, \lambda) \cdot 2^n,$$

and size

$$\mathsf{Size}_0(|C|, \lambda, n) = \text{poly}(|C|, \lambda) \cdot 2^n,$$

where C is the circuit to be obfuscated, λ is the security parameter, and n is the input length of C, exists *unconditionally*—simply output the function table of C (i.e., the output of C on all possible inputs). Such inefficient *iO* , however, are not useful for applications.

We here consider *iO* with just "slightly non-trivial" running-time; namely, we allow the running time to be

$$T_0(|C|, \lambda, n) = \text{poly}(|C|, \lambda) \cdot 2^n,$$

but require the *size* of the obfuscation to be

$$\mathsf{Size}_\epsilon(|C|, \lambda, n) = \text{poly}(|C|, \lambda) \cdot 2^{n(1-\epsilon)}$$

where $\epsilon > 0$. We refer to this notion as *iO with exponential efficiency*, or simply *exponentially-efficient iO (**XiO**)* (Recall that, in contrast, for "standard" *iO* , the running time and size of the obfuscator is required to be $\text{poly}(|C|, \lambda)$). In essence, **XiO** requires the obfuscator to be just slightly smaller than a brute-force canonicalization of the circuit.

Note that **XiO** obfuscators are only efficiently computable for circuits that take short inputs; we thus here restrict our attention to **XiO** for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ (or "short-input" **XiO**).

Main Theorem: Perhaps surprisingly, we show that in the regime of subexponential security, under the LWE assumption, **XiO** for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ implies (standard) *iO* for P/poly.

Theorem 33. *Assume subexponential security of the LWE assumption, and the existence of subexponentially-secure* ***XiO*** *for* $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$. *Then there exists subexponentially-secure iO for* P/poly.

As a corollary of Theorem 33, we get that subexponentially-secure short-input *iO* implies subexponentially-secure "standard" *iO* (since *iO* trivially implies **XiO**).

Techniques [114], improving on results of Ananth and Jain [4] and Bitansky and Vaikuntanathan [33], show that the existence of subexponentially-secure *functional encryption with weakly sublinearly compact ciphertexts (a.k.a. weakly sublinear compact FE)* for P/poly implies *iO* for P/poly. Roughly speaking, a (single-key) functional encryption scheme is a public-key encryption scheme for which it is possible to release a (single) functional secret-key sk_C (for circuit C of some a-priori bounded size S) such that knowledge of sk_C enables efficiently computing $C(m)$ given any encryption of the message m, (but nothing more); sublinear compactness means that the ciphertext *size* is sublinear in the upper bound S on the circuit-size (though the encryption *time* is allowed to depend polynomially on S).[1]

Our main technical contribution will be showing that **XiO** for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ implies sublinear compact FE for P/poly, which by the above-mentioned result implies our main theorem.

[1] More precisely, in a functional encryption scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, Setup samples a public-key, secret-key pair (pk, msk), $\mathsf{KeyGen}(msk, C)$ generates the functional secret key sk_C; $\mathsf{Enc}(pk, m)$ outputs an encryption c of m, and $\mathsf{Dec}(sk_C, c)$ outputs $C(m)$ if c is an encryption of m.

Theorem 34. *Assume the LWE assumption (resp. subexponential security of the LWE assumption) holds, and the existence of* ***XiO*** *for* $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ *(resp. subexponentially-secure* ***XiO*** *for* $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$*). Then there exists weakly sublinear compact FE for* P/poly *(resp. subexponentially-secure sublinear compact FE for* NC^1*).*

Note that Theorem 34 is interesting in its own right as it applies also in the regime of polynomial security.[2]

The proof of Theorem 34 proceeds as follows. Following a proof template from [4] (we discuss this result in more detail below), we start off with the result of Goldwasser et al [93] which shows that under the LWE assumption, there exists a functional encryption scheme for *boolean* functions (i.e., functions with 1-bit outputs) in NC^1 that has *logarithmic* compactness. Combined with [3], this can be used to construct a functional encryption scheme for *boolean* functions in P/poly that still has logarithmic compactness. We next show how to use **XiO** for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ to extend any such compact FE scheme for boolean functions to one that handles arbitrary polynomial-sized circuits (with potentially long outputs). ([4] provided a similar transformation assuming so-called *compact randomized encoding* instead of **XiO**.)

We now turn to describe our transformation from "single-bit compact FE" to "multi-bit weakly sublinear compact FE". As an intial approach, instead of simply encrypting a message m, encrypt the sequence $(m; 1), (m; 2), \ldots (m; \ell)$, where ℓ is the maximum output length of the class of functions we want to be able to evalute. Then, instead on simply releasing a functional secret key for a circuit C, release a secret key for the function $C'(m; i) = C_i(m)$, where $C_i(m)$ denotes the ith

[2]Furthemore, as we remark later on, sublinear compact FE trivially implies a variant of **XiO** and this variant of **XiO** is also sufficient for our theorems. As such, by our results, **XiO** may be viewed as a new way to characterize the complexity of sublinear compact FE.

output bit of $C(m)$. This approach clearly enables evaluating circuits with multi-bit outputs; but the encryption scheme is no longer compact! The length of the ciphertext grows *linearly* with the number of output bits. To retain compactness (or at least sublinear compactness), we have the encryption algorithm release an obfuscation of a program Π that generates all the ℓ encryptions—more precisely, given an index i, it applies a PRF (with a hard-coded seed) to the index i to generate randomness r_i and then outputs an encryption of $(m; i)$. As long as obfuscation size is "just-slightly-compressing", the functional encryption will have weak sublinear compactness; furthermore, the program we obfuscate only needs to take inputs of length $O(\log \lambda)$. Thus, it suffices to assume the obfuscator satisfies **XiO** for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$.

To prove security of the construction, we use the "one-input-at-a-time" technique from [42, 86, 130, 85, 57], and the punctured program technique of Sahai and Waters [134]; the crucial point that enables us to keep the obfuscation small is that the output of the program Π on different inputs uses independent randomness (since they are independent encryptions) and thus in the hybrid arguments it suffices to puncture the PRF on a single point.

Let us end this section by briefly comparing our transformation to that of Ananth and Jain [4]; as mentioned above [4] shows how to use "compact randomized encoding" to transform single-bit compact FE for NC^1 into multi-bit compact FE for NC^1. As we explain in more detail in Remark 11, compact randomized encoding can be viewed as a special case of **XiO** for the class of *Turing machines* (as opposed to circuits) with short input. Turing machine obfuscation is a significantly more challenging task than circuit obfuscation. We provide a brief description of their transformation in Section 5.5 and explain why the

transformation fails when using **XiO**.

5.2 Preliminaries

Let $\mathcal{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \ldots, n\}$. We denote by PPT probabilistic polynomial time Turing machines, and by nuPPT non-uniform probabilistic polynomial time Turing machines. The term $\mathsf{negligible}$ is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from non-negative integers to reals is called $\mathsf{negligible}$ if for every constant $c > 0$ and all sufficiently large n, it holds that $\nu(n) < n^{-c}$ For any algorithm A and input x we denote by $\mathsf{outlen}_A(x)$, the output length of A when run with input x.

Definition 38. *We denote by* $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ *the class of circuits* $\{C_\lambda\}$ *where* C_λ *are* $\mathrm{poly}(\lambda)$*-size circuits that have input length* $c \log \lambda$ *for some constant* c*.*

5.2.1 Puncturable PRF

Definition 39 (Puncturable PRF)**.** *A puncturable pseudo-random function* F *is given by a triple of efficient algorithms* ($\mathsf{F.Key}$, $\mathsf{F.Punc}$, $\mathsf{F.Eval}$)*, and a pair of computable functions* $n(\cdot)$ *and* $m(\cdot)$*, satisfying the following conditions:*

- **Functionality preserved under puncturing***: For every polynomial size set* $S \subseteq \{0,1\}^{n(\lambda)}$ *and for every* $x \in \{0,1\}^{n(\lambda)} \backslash S$*, we have that:*

$$\Pr[\mathsf{F.Eval}(K, x) = \mathsf{F.Eval}(K_S, x) : K \leftarrow \mathsf{F.Key}(1^\lambda), K_S = \mathsf{F.Punc}(K, S)] = 1$$

- **Pseudorandom at punctured points**: *For every polynomial size set* $S \subseteq \{0,1\}^{n(\lambda)}$ *we have that for every nuPPT adversary A we have that:*

$$|\Pr[A(K_S, \mathsf{F.Eval}(K, S)) = 1] - \Pr[A(K_S, U_{m(\lambda)\cdot|S|}) = 1]| = \mathrm{negl}(\lambda)$$

where $K \leftarrow \mathsf{F.Key}(1^\lambda)$ *and* $K_S = \mathsf{F.Punc}(K, S)$ *and* $\mathsf{F.Eval}(K, S)$ *denotes the concatenation of* $\mathsf{F.Eval}(K, x_1), \ldots, \mathsf{F.Eval}(K, x_k)$ *where* $S = \{x_1, ..., x_k\}$ *is the enumeration of the elements of* S *in lexicographic order,* U_ℓ *denotes the uniform distribution over* ℓ *bits.*

5.2.2 Functional Encryption

We note that in this work, we only need the security of the functional encryption scheme to hold with respect to statically chosen challenge messages and functions. We further consider FE schemes that only produce a single functional secret key for each public key.

Definition 40 (Functional Encryption)**.** *A public key functional encryption scheme for a class of circuits* $\{C_\lambda\}$ *is a tuple of PPT algorithms* $(\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *that behave as follows:*

- $(msk, pk) \leftarrow \mathsf{FE.Setup}(1^\lambda)$*:* $\mathsf{FE.Setup}$ *takes as input the security parameter* λ *and outputs the master secret key msk and public key pk.*
- $sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C)$*:* $\mathsf{FE.KeyGen}$ *takes as input the master secret key and a circuit* $C \in C_\lambda$ *and outputs the functional secret key* sk_C*.*
- $c \leftarrow \mathsf{FE.Enc}(pk, m)$*:* $\mathsf{FE.Enc}$ *takes as input the public key and message* $m \in \{0,1\}^*$ *and outputs the ciphertext c.*

- $y \leftarrow \mathsf{FE.Dec}(sk_C, c)$*:* $\mathsf{FE.Dec}$ *takes as input the functional secret key and ciphertext and outputs* $y \in \{0,1\}^*$.

We require the following conditions hold:

- **Correctness***: For every* $\lambda \in \mathbb{N}$, $C \in C_\lambda$ *with input length* n *and message* $m \in \{0,1\}^n$, *we have that*

$$\Pr\left[\begin{array}{l|l} & (pk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda) \\ sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C) & : C(m) = \mathsf{FE.Dec}(sk_C, c) \\ & c \leftarrow \mathsf{FE.Enc}(pk, m) \end{array}\right] = 1$$

- **Selective Security***: For every nuPPT A there exists a negligible function* μ *such that for every* $\lambda \in \mathbb{N}$, *every circuit* $C \in C_\lambda$ *with input length* n *and pair of messages* $m_0, m_1 \in \{0,1\}^n$ *such that* $C(m_0) = C(m_1)$ *we have that* $|\Pr[A(\mathcal{D}_0) = 1] - \Pr[A(\mathcal{D}_1) = 1]| \leq \mu(\lambda)$ *where*

$$\mathcal{D}_b = Pr\left[\begin{array}{l|l} & (pk, msk) \leftarrow \mathsf{FE.Setup}(1^\lambda) \\ sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C) & : (pk, sk_C, c_b) \\ & c_b \leftarrow \mathsf{FE.Enc}(pk, m_b) \end{array}\right]$$

We say the scheme has sub-exponential security *if there exists a constant* ϵ *such that for every* λ, *every* 2^{λ^ϵ}*-size adversary* A, $|\Pr[A(\mathcal{D}_0) = 1] - \Pr[A(\mathcal{D}_1) = 1]| \leq 1/2^{\lambda^\epsilon}$ *where* $\mathcal{D}_b$ *is defined above.*

We recall the definition of compactness and succinctness for functional encryption schemes, as defined in [33, 4].

Definition 41 (Compact Functional Encryption)**.** *We say a functional encryption scheme for a class of circuits* $\{C_\lambda\}$ *is* compact *if for every* $\lambda \in \mathbb{N}$, $pk \leftarrow \mathsf{FE.Setup}(1^\lambda)$ *and* $m \in \{0,1\}^*$ *we have that* $\mathsf{Time}(\mathsf{FE.Enc}(pk, m)) = \mathrm{poly}(\lambda, |m|, \log s)$ *where* $s =$

$\max_{C \in \mathcal{C}_\lambda} |C|$.

We say the scheme has sub-linear compactness *if the running time of* $\mathsf{FE.Enc}$ *is bounded as* $\mathsf{Time}(\mathsf{FE.Enc}(pk, m)) = \mathrm{poly}(\lambda, |m|) \cdot s^{1-\epsilon}$ *where* $\epsilon > 0$.

Definition 42 (Succinct Functional Encryption)**.** *A compact functional encryption scheme for a class of circuits that output only a* single bit *is called a* succinct *functional encryption scheme.*

Theorem 35 ([93])**.** *Assuming (sub-exponentially secure) LWE, there exists a (sub-exponentially secure) succinct functional encryption scheme for* NC^1.

We note that [93] do not explicitly consider sub-exponentially secure succinct functional encryption, but their construction satisfies it (assuming sub-exponentially secure LWE). Additionally, we have the following bootstrapping theorem:

Theorem 36 ([78, 3, 4])**.** *Assuming the existence of symmetric-key encryption with decryption in* NC^1 *(resp. sub-exponentially secure) and succinct FE for* NC^1 *(resp. sub-exponentially secure), there exists succinct FE for* $\mathsf{P/poly}$ *(resp. sub-exponentially secure).*

In this paper, we will consider a weaker compactness notion, where only the ciphertext size (but not the encryption time) is sub-linear in the output length of the function being evaluated.

Definition 43 (Weakly Sublinear Compact Functional Encryption)**.** *We say a functional encryption scheme for a class of circuits* $\{C_\lambda\}$ *is* weakly sublinear compact *if there exists* $\epsilon > 0$ *such that for every* $\lambda \in \mathbb{N}$, $pk \leftarrow \mathsf{FE.Setup}(1^\lambda)$ *and* $m \in \{0,1\}^*$

we have that

$$\mathsf{Time}_{\mathsf{FE.Enc}}(pk, m) = \mathrm{poly}(\lambda, |m|, s)$$

$$\mathsf{outlen}_{\mathsf{FE.Enc}}(pk, m) = s^{1-\epsilon} \cdot \mathrm{poly}(\lambda, |m|)$$

where $s = \max_{C \in \mathcal{C}_\lambda} |C|$.

5.2.3 Indistinguishability Obfuscation

We recall the notion of indistinguishability obfuscation ($i\mathcal{O}$).

Definition 44 (Indistinguishability Obfuscator)**.** *A PPT machine* $i\mathcal{O}$ *is an* indistinguishability obfuscator *for a circuit class* $\{\mathcal{C}_\lambda\}_{\lambda \in \mathcal{N}}$ *if the following conditions are satisfied:*

- **Functionality:** *for all security parameters* $\lambda \in \mathbb{N}$*, for all* $C \in \mathcal{C}_\lambda$*, for all inputs* x*, we have that*

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(C)] = 1 \ .$$

- **Indistinguishability:** *for any polysize distinguisher* $\mathcal{D}$*, there exists a negligible function* μ *such that the following holds: For all security parameters* $\lambda \in \mathbb{N}$*, for all pairs of circuits* $C_0, C_1 \in \mathcal{C}_\lambda$ *of the same size, we have that if* $C_0(x) = C_1(x)$ *for all inputs* x*, then*

$$\left| \Pr\left[\mathcal{D}(i\mathcal{O}(C_0)) = 1\right] - \Pr\left[\mathcal{D}(i\mathcal{O}(C_1)) = 1\right] \right| \leq \mu(\lambda) \ .$$

We say the scheme has sub-exponential security *if there exists a constant* ϵ *such that for every* λ*, every* 2^{λ^ϵ}*-size adversary* $\mathcal{D}$*,* $|\Pr[\mathcal{D}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{D}(i\mathcal{O}(C_1)) = 1]| \leq 1/2^{\lambda^\epsilon}$.

We recall the following result from [114].

Theorem 37. *([114]) Assume the existence of sub-exponentially secure LWE. If there exists a weakly sublinear compact functional encryption scheme for* P/poly *with sub-exponential security, then there exists a sub-exponentially secure indistinguishability obfuscator for* P/poly.

5.3 Exponentially-Efficient *iO* (XiO)

In this section, we define our new notion of "inefficient" *iO* , which allows the obfuscator to to have running time as long as a brute-force canonicalizer that outputs the entire input-output table of the function, but requires the obfuscated program to be slightly smaller in size than a brute-force canonicalization.

Definition 45 (Exponentially-Efficient Indistinguishability Obfuscation (*iO*))**.** *A machine* ***XiO*** *is an* weak indistinguishability obfuscator *for a circuit class* $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ *if the following conditions are satisfied:*

- **Functionality:** *for all security parameters* $\lambda \in \mathbb{N}$*, for all* $C \in C_\lambda$*, for all inputs* x*, we have that*

$$\Pr[C' \leftarrow \boldsymbol{XiO}(1^\lambda, C) : C'(x) = C(x)] = 1 \ .$$

- **Indistinguishability:** *for any nuPPT distinguisher* A*, there exists a negligible function* μ *such that the following holds: For all security parameters* $\lambda \in \mathbb{N}$*, for all pairs of circuits* $C_0, C_1 \in C_\lambda$ *of the same size, we have that if* $C_0(x) = C_1(x)$ *for all inputs* x*, then*

$$|\Pr[A(\boldsymbol{XiO}(1^\lambda, C_0)) = 1] - \Pr[A(\boldsymbol{XiO}(1^\lambda, C_1)) = 1]| \leq \mu(\lambda)$$

We say the scheme has sub-exponential security *if there exists a constant ϵ such that for every λ, every 2^{λ^ϵ}-size adversary A,*

$$|\Pr[A(\boldsymbol{XiO}(1^\lambda, C_0)) = 1] - \Pr[A(\boldsymbol{XiO}(1^\lambda, C_1)) = 1]| \leq 1/2^{\lambda^\epsilon}$$

- **Non-trivial Efficiency:** *There exists a constant $\epsilon > 0$ such that for any security parameter $\lambda \in \mathbb{N}$, circuit $C \in C_\lambda$ with input length n and $C' \in \boldsymbol{XiO}(1^\lambda, C)$, we have that*

$$\mathsf{Time}_{\boldsymbol{XiO}}(1^\lambda, C) = \mathrm{poly}(\lambda, |C| \cdot 2^n)$$
$$\mathsf{outlen}_{\boldsymbol{XiO}}(1^\lambda, C) = \mathrm{poly}(\lambda, |C|) \cdot 2^{n(1-\epsilon)}$$

Remark 9. *(Circuits with logarithmic input length) Note that if we want the obfuscation to be efficient (i.e., polynomial-time in λ and the size of the circuit to be obfuscated), then the above definition is only meaningful when the class of circuits C_λ has input length $O(\log \lambda)$. Our results in this paper hold assuming Exponentially-Efficient $i\mathcal{O}$ for such classes.*

Remark 10. *(Exponentially-Efficient $i\mathcal{O}$ in the preprocessing model and comparison with Compact Functional Encryption) We can consider further a relaxation of the running-time requirement of the obfuscator. The obfuscator may first perform a* long *"pre-processing" step (without having seen the program to be obfuscated), taking time* $\mathrm{poly}(\lambda, s, 2^n)$ *(where s is the size bound on circuits to be obfuscated), and outputting a (potentially long) pre-processing public-key O_{pk}. The actual obfuscation then takes O_{pk}, and the circuit C as inputs, runs in time* $\mathrm{poly}(\lambda, s, 2^n)$ *and outputs an obfuscated program of size* $\mathrm{poly}(\lambda, s) \cdot 2^{n(1-\epsilon)}$*, and then the evaluation of the obfuscated program may finally also access the public-key O_{pk}. All our results also apply to this relaxed notion of exponentially efficient $i\mathcal{O}$.*

Additionally, we note that weakly sublinear compact FE directly implies this notion as follows: pre-processing public key O_{pk} (generated in the pre-processing step) is the public key pk for the FE and the functional secret key sk_{FT} corresponding to a function table generator *program that takes as input a circuit and outputs the function table of it; the obfuscation of a circuit C is an encryption of the circuit C (w.r.t., the FE public key pk), and evaluation of the obfuscated code uses the functional secret key sk_{FT} inside O_{pk} to compute the function table of C and selects the appropriate output. Sub-linear compactness of the functional encryption scheme implies the obfuscator has exponential efficiency.*

Remark 11. *(Comparison with Compact Randomized Encoding for Turing machines) [4] and [114] study a notion of compact randomized encodings [102, 8]. Roughly speaking, a* randomized encoding (RE) *is a method for* encoding *a Turing Machine* Π*, an input x and a running-time bound T, into a randomized encoding* $\widehat{\Pi(x)}$ *from which* $\Pi(x)$ *can be efficiently* decoded*; furthermore the encodings does not leak anything more about* Π *and x than what can be (inefficiently) deduced from just the output* $\Pi(x)$ *(truncated at T steps).*[3] *A randomized encodings is compact (resp. sublinearly compact) if the encoding time is poly-logarithmic (resp sublinear) in T (and polynomial in the size of* Π *and x). We note that sublinear compact RE directly implies* ***XiO*** *as follows: to obfuscate a circuit C, compute an encoding* $\widehat{FT_C}$ *of the* function table generator *Turing machine* FT_C *that has the circuit C hardcoded (i.e.,* FT_C *takes no inputs and simply computes the function table of C); evaluation of the obfuscation on an input i simply decodes the encoding* $\widehat{FT_C}$ *picks out the ith output. Sublinear compactness of the RE implies that the obfuscator is exponentially-efficient.*

In fact, the above methods extend to show that (sublinearly) compact RE implies

[3] Or equivalently, for any two programs Π_1, Π_2 and inputs x_1, x_2 such that $\Pi_1(x_1) = \Pi_2(x_1)$, a randomized encoding of Π_1, x is indistinguishable from an encoding of Π_2, x_2.

*a notion of **XiO** for* Turing machines. *We note that Turing machine obfuscation is a significantly harder task than circuit obfuscation (indeed, all known construction of Turing machine obfuscators first go through circuit obfuscation). We also point out that whereas (subexponentially-secure) iO for circuits is known to imply iO for Turing machine [31, 54, 112], these techniques do not apply in the regime of programs with short input.*

5.4 *iO* from XiO

In this section, we show how to achieve (full-fledged) *iO* from **XiO**.

5.4.1 Weakly Sublinear Compact FE from Succinct FE and XiO

We first give our construction of weakly sublinear compact FE from succinct FE and **XiO** for circuits with input-size $O(\log(\lambda))$. At a high-level, our idea is to have the ciphertext for the FE scheme be **XiO** of a circuit that, on input i, generates a succinct FE encryption of (m, i). The secret key corresponding to C consists of a single key for the succinct FE scheme, that, given a ciphertext encrypting (m, i), computes the ith output bit of $C(m)$.

Let F be a puncturable pseudorandom function, **XiO** be a exponentially efficient indistinguishability obfuscator for $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ and sFE be a succinct functional encryption scheme (resp. with sub-exponential security) for an appropriate class of circuits that includes C' defined below.. We define a compact functional encryption scheme FE for a class of poly-size circuits $\{C_\lambda\}$ as follows:

$(msk, pk) \leftarrow \mathsf{FE.Setup}(1^\lambda)$**:** FE.Setup is identical to sFE.Setup and has the same output.

$c \leftarrow \mathsf{FE.Enc}(pk, m)$**:** FE.Enc samples a PRF key $K \leftarrow \mathsf{F.Key}(1^\lambda)$ and outputs $\mathbf{XiO}(1^\lambda, G[pk, K])$ where $G[pk, K]$ is a circuit with input length $n = \log s$ where $s = \max_{C \in C_\lambda} \mathsf{outlen}(C)$, defined as follows:

$$G[pk, K](i) = \mathsf{sFE.Enc}(pk, (m, i); \mathsf{F.Eval}(K, i))$$

G is padded to be the same size as another circuit G'', which we will define later in the security proof. Both G and G'' will ultimately have size $S = \mathrm{poly}(\lambda, |m|, \log s)$ where $s = \max_{C \in C_\lambda} |C|$.

$sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C)$**:** FE.KeyGen outputs $\mathsf{sFE.KeyGen}(msk, C')$ where C' on input (m, i) outputs the i^{th} bit of $C(m)$, or outputs $\perp$ if i is greater than the output length of C.

$y \leftarrow \mathsf{FE.Dec}(sk_C, c)$**:** FE.Dec runs $c_i \leftarrow G[pk, K](i)$ and $y_i \leftarrow \mathsf{sFE.Dec}(sk_C, c_i)$ for every i and outputs $y_1, \ldots y_{2^n}$.

Let $\{C'_\lambda\}$ be a class of circuits that includes C' as defined above for every $C \in C_\lambda$.

Theorem 38. *Assuming* F *is a pseudorandom function (resp. with subexponential security),* ***XiO*** *is an exponentially efficient indistinguishability obfuscator for* $\mathsf{P}^{\mathsf{O}(\log \lambda)}/\mathsf{poly}$ *(resp. with subexponential security) and* sFE *is a succinct functional encryption scheme for* $\{C'_\lambda\}$ *(resp. with subexponential security), we have that* FE *as defined above is a functional encryption scheme for* $\{C_\lambda\}$ *with weakly sub-linear compactness (resp. and with subexponential security).*

Proof. We first show weak sublinear compactness of FE. Consider any λ, $C \in C_\lambda$, message m, $pk \in \mathsf{FE.Setup}(1^\lambda)$ and PRF key $K \in \{0,1\}^\lambda$. $\mathsf{Time}(\mathsf{FE.Enc}(pk, m))$

is the time **XiO** takes to obfuscate the circuit $G[pk, K]$, which is of size $S = \text{poly}(\lambda, |m|, \log s)$ where $s = \max_{C \in \mathcal{C}_\lambda} |C|$. Hence we have that

$$\mathsf{Time}_{\mathbf{XiO}}(1^\lambda, G[pk, K]) = \text{poly}(\lambda, |m|, \log s, \cdot 2^n) \leq \text{poly}(\lambda, |m|, s)$$

$$\mathsf{outlen}_{\mathbf{XiO}}(1^\lambda, G[pk, K]) = \text{poly}(\lambda, |m|, \log s) \cdot 2^{n(1-\epsilon)} \leq \text{poly}(\lambda, |m|) \cdot s^{1-\epsilon'}$$

where ϵ' is a constant with $0 < \epsilon' < \epsilon$.

Next we show the selective security of FE. The proof proceeds by a hybrid argument where in each hybrid distribution, the circuit being obfuscated, on input i, produces ciphertexts of m_1 when i is less than a "threshold", and ciphertexts of m_0 otherwise. Indistinguishability of neighboring hybrids is shown using the "punctured programming" technique" of [134], as was done in [57] for constructing $i\mathcal{O}$ for probabilistic functions. This technique is also used extensively in other applications of $i\mathcal{O}$, eg., [31], [54], [112] and more.

Assume for contradiction there exists a nuPPT A and polynomial p such that for sufficiently large λ, circuit $C \in \mathcal{C}_\lambda$ and messages m_0, m_1 such that $C(m_0) = C(m_1)$, A distinguishes $\mathcal{D}_0$ and $\mathcal{D}_1$ as defined in Definition 40 with advantage $1/p(\lambda)$. For $j \in [l]$, we define the j^{th} hybrid distribution H_j as follows:

$$H_j = \left(\begin{array}{ll} (msk, pk) \leftarrow \mathsf{FE.Setup}(1^\lambda) & \\ K \leftarrow \{0,1\}^\lambda & : pk, sk_C, \mathbf{XiO}(G'[pk, K, j, m_0, m_1]) \\ sk_C \leftarrow \mathsf{FE.KeyGen}(msk, C) & \end{array} \right)$$

where $G'[pk, K, j, m_0, m_1]$, where G' is defined as follows

$$G'[pk, K, j, m_0, m_1](i) = \begin{cases} \mathsf{sFE.Enc}(pk, (m_0, i); \mathsf{F}(K, i)) & \text{if } i > j \\ \mathsf{sFE.Enc}(pk, (m_1, i); \mathsf{F}(K, i)) & \text{if } i \leq j \end{cases}$$

We also require G' to be padded to be of the same size S as $G[pk, K, m]$.

We consider the hybrid sequence $\mathcal{D}_0, H_1, \ldots, H_l, \mathcal{D}_1$. By a hybrid argument, there exists a pair of neighboring hybrids in this sequence such that A distinguishes the pair with probability $\frac{1}{p(\lambda)\cdot(l+2)} = \frac{1}{\text{poly}(\lambda)}$. We show a contradiction by proving that each pair of neighboring hybrids is computationally indistinguishable.

We first note that $\mathcal{D}_0$ is indistinguishable from H_0. This follows by observing that $G'[pk, K, 0, m_0, m_1]$ is functionally identical to $G[pk, K, m_0]$, and applying the security of **XiO**. The same argument also shows that H_l is indistinguishable from $\mathcal{D}_1$.

Next, we show H_{j^*} and H_{j^*+1} are indistinguishable for each $j^* \in [l]$. Define hybrid distribution H'_0 which is identical to H_{j^*} except that **XiO** obfuscates a different circuit $G''[pk, K_{j^*}, j^*, m_0, m_1, c]$ where $K_{j^*} \leftarrow \mathsf{Punc}(\lambda, j^*)$ and $c \leftarrow \mathsf{sFE.Enc}(pk, (m_0, j^*); R)$ using uniformly sampled randomness R. G'' on input i has the same behavior as G' except $i = j^*$, where it outputs the hardcoded ciphertext c. The padding parameter S is defined as the size of G'', which is $\text{poly}(\lambda, |m|, \log s)$. By the "punctured programming" technique of Sahai-Waters [134], which relies on the security of the obfuscator **XiO** and puncturable PRF F, it follows that for sufficiently large λ, A distinguishes between H_{j^*} and H'_0 with negligible probability.

The puncturing programming technique itself works in two hybrid steps:

- First the circuit G' is modified on input j^* to output a hardcoded value $\mathsf{sFE.Enc}(pk, (m_0, j^*); \mathsf{F}(K, j^*))$, which is the same ciphertext G' previously computed. Also, the PRF key in G' is modified to be punctured on input j^*. Since this doesn't change the functionality of the circuit, indistinguisha-

bility follows from the security of **XiO**.

- Second, the hardcoded ciphertext is modified to be generated from real randomness R, and indistinguishability follows from the security of the puncturable PRF.

Next, we define hybrid distribution H'_1 which is identical to H'_0 except that the hardcoded ciphertext c is generated as $\mathsf{sFE.Enc}(pk, (m_1, j^*); R)$ for uniformly sampled randomness R. Since $C(m_0)$ is identical to $C(m_1)$, from the security of sFE, A distinguishes H'_0 and H'_1 with negligible probability.

Finally, note that H'_1 and H_{j^*+1} differ in the same way H'_0 and H_{j^*} do, and are hence indistinguishable by a similar argument. Hence A distinguishes H_{j^*} and H_{j^*+1} with negligible probability and we have a contradiction. This completes the proof.

We note that the proof above is described in terms of computational indistinguishability, but in fact also can be applied to show that FE is subexponentially-secure, if both **XiO** and sFE are subexponentially secure. □

Theorem 39. *Assuming sub-exponentially hard LWE, if there exists a sub-exponentially-secure exponentially efficient indistinguishability obfuscator for* $\mathsf{P/poly}$ *then there exists an indistinguishability obfuscator for* $\mathsf{P/poly}$ *with sub-exponential security.*

Proof. By Theorem 35 and Theorem 23, assuming subexponentially secure LWE, there exists a succinct functional encryption scheme for $\mathsf{P/poly}$ that is subexponentially-secure. Using this with a subexponentially-secure exponentially efficient indistinguishability obfuscator, by Theorem 38, we get weakly

sublinear compact function encryption for $\mathsf{P/poly}$ with sub-exponential selective security. Together with Theorem 37, this gives us iO for $\mathsf{P/poly}$. □

Remark 12. *(**XiO** for NC^1 suffices) Note that we can also achieve the above results assuming **XiO** for only NC^1 instead of $\mathsf{P/poly}$, with the caveat that we must additionally assume the existence of puncturable PRFs in NC^1. To do so, we modify our above construction, so that instead of obfuscating the circuit G, we instead obfuscate a different circuit H that generates a "garbling" [138] of G. Since H lies in NC^1, we only need **XiO** for NC^1.*

5.5 Comparison with [4]

In this section we briefly describe the related result by [4] and compare it with our result. [4] show how to construct a Compact Functional Encryption scheme from a Succinct Functional Encryption scheme and Compact Randomized Encodings for Turing machines. The rough idea is as follows: the compact functional secret key for a function f is a sequence of ℓ independent succinct functional secret keys where ℓ is the output length of f. The i^{th} succinct functional secret key corresponds to the function that outputs the i^{th} bit of f. The compact functional ciphertext for a message m is the randomized encoding of a machine Π that takes no input and when run, outputs $\{Enc(pk_i, m)\}_{i\in[\ell]}$ where pk_i is the public key corresponding to the i^{th} instance of the succinct functional scheme (these instances are generated using a PRF, hence the description size of Π is independent of ℓ). The compactness of the functional encryption scheme follows from the compactness of the randomized encoding scheme.

Note that the above result necessarily requires the computation being en-

coded be represented as a Turing machine, since the description size is required to be independent of the output length. In contrast we are able to rely on **XiO** for *circuits*, which is significantly weaker (see Remark 11).

BIBLIOGRAPHY

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

[2] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.

[3] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. The trojan method in functional encryption: From selective to adaptive security. Technical report, generically. Cryptology ePrint Archive, Report 2014/917, 2014.

[4] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:173, 2015.

[5] Benny Applebaum. Key-dependent message security: Generic amplification and completeness. pages 527–546, 2011.

[6] Benny Applebaum. Randomly encoding functions: A new cryptographic paradigm - (invited talk). In *Information Theoretic Security - 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings*, pages 25–31, 2011.

[7] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 162–172, 2014.

[8] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *FOCS*, pages 166–175, 2004.

[9] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. pages 166–175, 2004.

[10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[12] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. pages 152–163, 2010.

[13] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. pages 120–129, 2011.

[14] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. pages 483–501, 2012.

[15] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, volume 0, pages 106–115, 2001.

[16] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EuroCrypt'14*, 2013.

[17] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology CRYPTO 2001*, pages 1–18. Springer, 2001.

[18] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. pages 423–444, 2010.

[19] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. In *STOC*, pages 1–5, 1986.

[20] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. pages 398–415, 2013.

[21] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. pages 134–153, 2012.

[22] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012.

[23] Mihir Bellare, Igors Stepanovs, and Stefano Tessaro. Poly-many hardcore bits for any one-way function. Cryptology ePrint Archive, Report 2013/873, 2013. `http://eprint.iacr.org/`.

[24] Nir Bitansky and Ran Canetti. On strong simulation and composable point obfuscation. In *CRYPTO*, pages 520–537, 2010.

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

[26] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. pages 111–120, 2013.

[27] Nir Bitansky, Ran Canetti, Yael Kalai, and Omer Paneth. Virtual-grey-box obfuscation from general circuits. In *Advances in Cryptology CRYPTO 2014*, 2014.

[28] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. Indistinguishability obfuscation vs. auxiliary-input extractable functions: One must fall. *IACR Cryptology ePrint Archive*, 2013:641, 2013.

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

[30] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 439–448, 2015.

[31] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. *IACR Cryptology ePrint Archive*, 2015:356, 2015.

[32] Nir Bitansky and Omer Paneth. Zaps and non-interactive witness indistinguishability from indistinguishability obfuscation. In *Theory of Cryptog-*

raphy - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II, pages 401–427, 2015.

[33] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *IACR Cryptology ePrint Archive*, 2015:163, 2015.

[34] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.

[35] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology EUROCRYPT 2005*, pages 440–456. 2005.

[36] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. pages 410–428, 2013.

[37] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[38] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.

[39] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT (2)*, pages 280–300, 2013.

[40] 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, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 480–499, 2014.

[41] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology CRYPTO 2014*, 2014.

[42] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73, 2014.

[43] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. pages 52–73, 2014.

[44] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.

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

[46] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. 2013.

[47] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. To appear in TCC 2010, 2011.

[48] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[49] Zvika Brakerski, Ilan Komargodski, and Gil Segev. From single-input to multi-input functional encryption in the private-key setting. *IACR Cryptology ePrint Archive*, 2015:158, 2015.

[50] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[51] Christina Brzuska and Arno Mittelbach. Indistinguishability obfuscation versus point obfuscation with auxiliary input. Cryptology ePrint Archive, Report 2014/405, 2014. `http://eprint.iacr.org/`.

[52] Christina Brzuska and Arno Mittelbach. Using indistinguishability obfuscation via uces. Cryptology ePrint Archive, Report 2014/381, 2014. `http://eprint.iacr.org/`.

[53] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology CRYPTO 1997*, pages 455–469, 1997.

[54] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. *IACR Cryptology ePrint Archive*, 2014:769, 2014.

[55] Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On obfuscation with random oracles. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 456–467, 2015.

[56] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic functionalities and applications. Manuscript, 2014.

[57] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and JesperBuus Nielsen, editors, *Theory of Cryptography*, volume 9015 of *Lecture Notes in Computer Science*, pages 468–497. Springer Berlin Heidelberg, 2015.

[58] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 519–535, 2013.

[59] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 3–12, 2015.

[60] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero knowledge from p-certificates. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 50–59, 2013.

[61] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[62] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2015*

- 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, pages 267–286, 2015.

[63] Ivan Damgrd, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. pages 54–74, 2012.

[64] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[65] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.

[66] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology–EUROCRYPT 2013*, pages 1–17. Springer, 2013.

[67] 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 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 74–94, 2014.

[68] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. pages 74–94, 2014.

[69] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

[70] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *Proc. of FOCS 2013*, 2013.

[71] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. pages 40–49, 2013.

[72] 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'14*, 2013.

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

[74] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014. `http://eprint.iacr.org/2014/666`.

[75] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the 45th Annual ACM Symposium on Symposium on Theory of Computing*, STOC '13, pages 467–476, 2013.

[76] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. 2015.

[77] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. pages 465–482, 2010.

[78] Gentry, Halevi, Raykova, and Wichs. Outsourcing private ram computation. *Proc. of FOCS 2014*, 2014.

[79] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[80] Craig Gentry. Fully homomorphic encryption using ideal lattices. pages 169–178, 2009.

[81] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 498–527, 2015.

[82] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 405–422, 2014.

[83] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private ram computation. 2014.

[84] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. `http://eprint.iacr.org/2014/309`.

[85] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014.

[86] Craig Gentry, Allison Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *Advances in Cryptology–CRYPTO 2014*, pages 426–443. Springer, 2014.

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

[88] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, pages 578–602, 2014.

[89] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 553–562, 2005.

[90] 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.

[91] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. pages 555–564, 2013.

[92] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564, 2013.

[93] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564, 2013.

[94] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-Proofs. Cryptology ePrint Archive, Report 2011/456, 2011.

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

[96] Shafi Goldwasser and Guy Rothblum. On best-possible obfuscation. In *Theory of Cryptography*, volume 4392, pages 194–213. 2007.

[97] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 545–554, 2013.

[98] Satoshi Hada. Zero-knowledge and code obfuscation. In *Advances in Cryptology–ASIACRYPT 2000*, pages 443–457. Springer, 2000.

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

[100] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304, 2000.

[101] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. pages 294–304, 2000.

[102] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[103] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 244–256, 2002.

[104] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *Theory of Cryptography - 12th Theory*

of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II, pages 668–697, 2015.

[105] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 668–697, 2015.

[106] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. pages 485–494, 2014.

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

[108] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[109] Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. pages 659–667, 1999.

[110] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. 2014.

[111] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for np. Cryptology ePrint Archive, Report 2014/213, 2014. `http://eprint.iacr.org/`.

[112] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. Technical report, Cryptology ePrint Archive, Report 2014/925, 201 4. http://eprint. iacr. org, 2014.

[113] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. 2015.

[114] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. 2015.

[115] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Non-trivially

efficient indistinguishability obfuscation. *Proceedings of the Nineteenth International Conference on the Theory and Practice of Public-Key Cryptography, PKC 2016, Taipei, Taiwan, March 6-9, 2016*, To appear.

[116] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. *Proceedings of the Thirteen Theory of Cryptography Conference, TCC 2016, Tel-Aviv, Israel, January 10-13, 2016*, To appear.

[117] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. pages 1219–1234, 2012.

[118] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 719–734, 2013.

[119] Mohammad Mahmoody, Ameer Mohammed, and Soheil Nematihaji. More on impossibility of virtual black-box obfuscation in idealized models. *IACR Cryptology ePrint Archive*, 2015:632, 2015.

[120] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[121] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. pages 71–80, 1999.

[122] Brice Minaud and Pierre-Alain Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015. `http://eprint.iacr.org/`.

[123] Moni Naor. On cryptographic assumptions and challenges. In Boneh [34], pages 96–109.

[124] Moni Naor. On cryptographic assumptions and challenges. In Boneh [34], pages 96–109.

[125] Moni Naor. On cryptographic assumptions and challenges (invited talk). pages 96–109, 2003.

[126] Moni Naor and Moti Yung. Public-key cryptosystems provably secure

against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990.

[127] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. pages 422–439, 2012.

[128] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 500–517, 2014.

[129] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO'14*, 2014.

[130] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology–CRYPTO 2014*, pages 500–517. Springer, 2014.

[131] Rafael Pass and Abhi Shelat. Impossibility of VBB obfuscation with ideal constant-degree graded encodings. *IACR Cryptology ePrint Archive*, 2015:383, 2015.

[132] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[133] Ron D Rothblum. On the circular security of bit-encryption. In *Theory of Cryptography*, pages 579–598. Springer, 2013.

[134] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *Proc. of STOC 2014*, 2014.

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

[136] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014. `http://eprint.iacr.org/`.

[137] Andrew C Yao. Theory and application of trapdoor functions. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 80–91. IEEE, 1982.

[138] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.