Succinct Randomized Encodings and their Applications[*]

Nir Bitansky[†] Sanjam Garg[‡] Huijia Lin[§] Rafael Pass[¶] Sidharth Telang[§]

April 21, 2015

Abstract

A *randomized encoding* allows to express a "complex" computation, given by a function f and input x, by a "simple to compute" randomized representation $\widehat{f}(x)$ whose distribution encodes $f(x)$, while revealing nothing else regarding f and x. Existing randomized encodings, geared mostly to allow encoding with low parallel-complexity, have proven instrumental in various strong applications such as multiparty computation and parallel cryptography.

This work focuses on another natural complexity measure: *the time required to encode*. We construct *succinct randomized encodings* where the time to encode a computation, given by a program Π and input x, is essentially independent of Π's time complexity, and only depends on its space complexity, as well as the size of its input, output, and description. The scheme guarantees computational privacy of (Π, x), and is based on indistinguishability obfuscation for a relatively simple circuit class, for which there exist instantiations based on polynomial hardness assumptions on multi-linear maps.

We then invoke succinct randomized encodings to obtain several strong applications, including:

- Succinct indistinguishability obfuscation, where the obfuscated program $i\mathcal{O}(\Pi)$ computes the same function as Π for inputs x of apriori-bounded size. Obfuscating Π is roughly as fast as encoding the computation of Π on any such input x. Here we also require subexponentially-secure indistinguishability obfuscation for circuits.
- Succinct functional encryption, where a functional decryption key corresponding to Π allows decrypting $\Pi(x)$ from encryptions of any plaintext x of apriori-bounded size. Key derivation is as fast as encoding the corresponding computation.
- Succinct reusable garbling, a stronger form of randomized encodings where any number of inputs x can be encoded separately of Π, independently of Π's time and space complexity.
- Publicly-verifiable 2-message delegation where verifying the result of a long computation given by Π and input x is as fast as encoding the corresponding computation. We also show how to transform any 2-message delegation scheme to an essentially non-interactive system where the verifier message is reusable.

Previously, succinct randomized encodings or any of the above applications were only known based on various non-standard knowledge assumptions.

At the heart of our techniques is a generic method of compressing a piecemeal garbled computation, without revealing anything about the secret randomness utilized for garbling.

[*]This paper unifies [BGT14] and [LP14]. An extended abstract will appear in the proceedings of STOC 15'.

[†]MIT, nirbitan@csail.mit.edu. Part of this research was done at IBM T.J. Watson Research Center.

[‡]University of California, Berkeley, sanjamg@berkeley.edu.

[§]University of California, Santa Barbara, rachel.lin@cs.ucsb.edu.

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

Contents

1 Introduction

The notion of a *randomized encoding*, coined by Ishai and Kushilevitz [IK00], 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 [IK00, IK02a, AIK04]. When settling for privacy against computationally bounded adversaries, and assuming low-depth pseudo-random generators, the latter extends to arbitrary poly-time computations [AIK06], which was already demonstrated in Yao's seminal work on garbling circuits [Yao82]. The constructed randomized encodings were in turn shown to have various strong applications to parallel cryptography, secure computation, verifiable delegation, and more (see [App11b] 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 [IK00, IK02a] was to achieve such savings in multi-party computation (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 **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 (**iO**) for any model of computation (e.g., **iO** for Turing machines) would directly imply corresponding succinct randomized encodings.[2] Still, constructions of succinct **iO** [BCP14, ABG$^+$13], or direct constructions of succinct randomized encodings [GKP$^+$13a, GHRW14b] are based on considerably strong computational assumptions such as extractable witness encryption, succinct non-interactive arguments, and differing-inputs obfuscation. In the language of [Nao03] these assumptions are not *efficiently-falsifiable*; furthermore, in some cases they have been shown implausible [BP13, BCPR14, GGHW14].

1.1 Contributions

Our core contribution is a succinct randomized encoding relying on (non-succinct) **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. The construction, in fact, satisfies the enhanced guarantee of *a succinct garbling schemes* [Yao82, AIK06, BHR12b], with the extra feature that inputs can be encoded independently of the program and its 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.*

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 [BLMR13]), and restricting attention to any class of computations with a-priori-bounded running time $t(n)$, we can settle for **iO** for circuits in NC^1 with input size $O(\log(t(n))$ (which is a $\mathrm{poly}(t(n))$-time falsifiable assumption on its own). Obtaining **iO** for this class may be done based on qualitatively weaker assumptions; indeed, for any polynomial $t(\cdot)$ the construction of Gentry et al. [GLSW14] would imply **iO** 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.

[1] Specifically, it can be shown that, for a language $\mathcal{L}$, 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 $\mathcal{L}$ 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 $\mathsf{AM} \subseteq \mathsf{NP}$ [Kv99, MV99]), the above would imply that $\mathcal{L}$ 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.

[3] More generally, one of the challenges in basing **iO** 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 $\mathrm{poly}(t(n))$.

Application 1: Succinct Indistinguishability Obfuscation. Our first (and somewhat strongest) application of succinct randomized encodings is succinct **iO** 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 x of some apriori-bounded length n are computationally indistinguishable. The construction is based on subexponential **iO**, whereas any form of succinct **iO** realized so far [ABG+13, BCP14, IPS15] 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 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 **iO** 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.*

The theorem is somewhat the succinct analog of previous bootstrapping theorems [App14, CLTV15] 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 **iO** for arbitrarily long computations to **iO** 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 [GGH+13a, Wat14], or even from concrete (and efficiently falsifiable) assumptions on composite order multilinear graded-encodings [GGHZ14]. 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 **iO**, requiring essentially the same strong (non-falsifiable) assumptions.

One can replace **iO** for circuits, in the above FE constructions, with the succinct **iO** from Theorem 2, and obtain FE where computing sk_Π is comparable to (succinctly) obfuscating Π. This, however, will require the same sub-exponential hardness of **iO** 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 3 (Informally Stated). *Assume the existence of succinct randomized encodings 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 [GGHZ14], results in full (adaptive) security.*

As observed in previous work [GHRW14b, CIJ+13, GKP+13b], 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 $\widehat{x}$ encoding any input x. 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 [GLR11, BCCT12, DFH12, BCCT13] or proven secure only in the random oracle model [Mic00].[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 4 (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 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 input size n (and the security parameter).*

Plugging in our succinct **iO** into the perfect non-interactive zero-knowledge (NIZK) arguments of Sahai-Waters [SW14a] 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 [SW14a]).

__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 pseudo-random 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

[4]Notably, in the setting of private-verification Kalai, Raz, and Rothblum give a solution based on the subexponential learning with errors assumption [KRR14].

[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., [KRR14]). 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.

can be composed [AIK06]. Concretely, we can consider an outer layer of a non-succinct shallow randomized encoding (like Yao [Yao82]) 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 [IK00, IK02a], where we can reduce the communication overhead from depending on the circuit size required to compute a multiparty function $f(x_1, \dots, x_m)$ 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 [Gen09, AJL+12, LTV12, GGHR14]. 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 $\mathcal{F}$. Applebaum [App11a] shows that any scheme that is KDM-secure with respect to some class of functions $\mathcal{F}$ can be made resilient to a bigger class $\mathcal{F}' \supseteq \mathcal{F}$, if functions in $\mathcal{F}'$ can be randomly encoded in $\mathcal{F}$. Our succinct randomized encodings will essentially imply that KDM-security for circuits of any fixed polynomial size $s(\cdot)$ (such as the scheme of [BHHI10]) 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 2) 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* ***iO*** *for* 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 **iO** 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 [CHJV15], who constructed succinct **iO** 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.

1.2 Techniques

We next overview our construction of succinct randomized encodings for bounded space programs. Beyond **iO**, 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** [Yao82, BHR12b]. 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 **iO** to "compress" the size of the garbled program, by providing 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 [Yao82] 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 $\widetilde{\Pi}_{t^*}$.

We then iteratively in descending order simulate the j^{th} garbled circuits $\widetilde{\Pi}_j$ with the output being set to $\widetilde{\text{conf}}_{j+1}$ generated in the previously simulated garbled circuit. We finally simulate the remaining $j > t^*$ garbled circuits $\widetilde{\Pi}_j$ 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 $(\widetilde{\Pi}_1, \ldots \widetilde{\Pi}_{t(n)})$.[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 **iO** 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.

[6]This "layered" simulation strategy resembles that of Applebaum, Ishai, and Kushilevitz in the context of arithmetic garbling [AIK11].

Now, the new succinct garbled program is the obfuscated program $\mathbf{\Lambda} \overset{\$}{\leftarrow} i\mathcal{O}(\mathbf{\Pi}^{s,s'})$, 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 $\widetilde{\text{conf}}_j$ 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 **iO** 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 [SW14a] 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 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 $\widetilde{\mathbf{\Pi}}_{j+1}^{s,s'}$, 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. [CLTV15] show that **iO** for NC^1 can be bootstrapped to obtain **iO** for all circuits, assuming puncturable PRFs in NC^1 [BLMR13], and incurring a security loss that is exponential in the size of the input. Accordingly, for polynomial

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

$t(n)$, 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 [LO13, GHL+14, GLOS15]. 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 $\text{poly}_{\mathbf{iO}}(s(n))$, where $\text{poly}_{\mathbf{iO}}(\cdot)$ is the overhead due to obfuscation, and the complexity of decoding for a T-time computation $\Pi(x)$ is at least $T \cdot \text{poly}_{\mathbf{iO}}(s(n))$, 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 $\text{poly}_{\mathbf{iO}}(\cdot)$, so that eventually decoding may take time $\text{poly}_{\mathbf{iO}}(s(n)) \cdot t(n)/s(n)$ 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 $O(T + s(n))$.

1.2.1 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 [App14]. 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 x, applies the PRF to x to derive randomness, and then computes a succinct randomized encoding of $\widehat{\Pi}(x)$. The obfuscated $i\mathcal{O}(\Pi)$, given input x computes the encoding, decodes it, and returns the result.

The analysis in [App14] establishes security in case that the circuit obfuscator $i\mathcal{O}$ is virtually black-box secure. We show that if $i\mathcal{O}$ 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. [CLTV15].

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. [GHRW14b], we can replace the non-succinct randomized encodings for RAM in their construction with our succinct randomized encodings, and obtain selectively-secure FE.[8] Alternatively, starting from the scheme of Garg et al. [GGHZ14], 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) [AIK10, GGP10, PRV12, GKP⁺13b]. To delegate a computation, given by Π and x, the verifier simply sends the prover a randomized encoding $\widehat{\Pi'}(x, r)$, where Π' is a machine that returns r if and only if it accepts x, 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, x), applies the PRF to derive randomness, and generates a first message for the delegation 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 [KRR14] and maintains soundness, even if the prover has a verification oracle.

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

[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.

the underlying garbled RAMs, which are indeed implemented in [LO13, GHL$^+$14, GLOS15] 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 indistinguishable. However, assuming **iO** it is not clear how to achieve any advantage over the previous solution. Intuitively, whenever we invoke **iO** 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 [KLW15] construct fully-succinct randomized encodings from **iO**. Their solution takes a similar route to that of Canetti et al. [CHJV15] 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 "**iO**-friendly" in their terminology.

Organization In Section 2, we provide preliminaries, including: different models of computation considered in the paper, definitions of garbling schemes and **iO** with different efficiency levels. In Section 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 4. Finally, in Section 5, we present applications of succinct randomized encodings to succint **iO** and delegation; we omit details for other applications that are achieved by directly plugging in randomized encodings in previous works. In Appendix A, we show how to bootstrap any circuit **iO** to one with quasi-linear blowup.

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 $\nu(n) < n^{-c}$.

2.1 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 $\mathsf{CIR}[D] = \{\mathcal{C}_\lambda\}$ of include all deterministic circuits of size at most $D(\lambda)$.

NC^1 **Circuits.** For every constant c and polynomial D, the class $\mathsf{NC}_c[D] = \{\mathcal{C}_\lambda\}$ 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 [BCP14].

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 $\mathsf{ALG}^T = \left\{\mathcal{AL}^T_\lambda\right\}$ 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 Garbling Schemes

In this section, we define garbling schemes, following in most part the definitions in [BHR12b]. 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 5, and is a direct projection of the definition of garbling schemes.

Definition 1 (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 $(\widehat{AL}, \mathbf{key})$.*[9]

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

- 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 $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda^T\}$ 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 2 (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 $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda^{T'}\}$, 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 simulation 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 $\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval}$ should be polynomial time machines, that is, the complexity of $\mathsf{Garb}, \mathsf{Eval}$ 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 3 (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* $p_{\mathsf{Garb}}(\lambda, |AL|, AL.m)$,[10]
- $\hat{x} = \mathsf{Encode}(\mathbf{key}, x)$ *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.*

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

Space-dependent complexity: *The above efficiency conditions hold with $p_{\mathsf{Garb}}, p_{\mathsf{Encode}}, p_{\mathsf{Eval}}$ 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 on $AL.T$, and the running time of* Eval *is bounded by $AL.T \cdot p_{\mathsf{Eval}}(\lambda, |AL|, |x|)$.*

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 Output Dependence. 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 as long as we require simulation-based security. This follows from a standard incompressibility argument. Indeed, assume the existence of a pseudorandom generator G, and consider the encoding of G and a random input seed s. We claim that the size of the garbled function $\widehat{G}$ and encoded input $\widehat{s}$ must be as large as the output $|G(s)|$. Otherwise, the efficient simulator can "compress" random strings, as it cannot distinguish the actual output $G(s)$ from a truly uniform one.

The dependence on the output size could possibly be eliminated if we settle for indistinguishability-based security, meaning that the garbling of two (equal-length) program-input pairs $(\widehat{AL}_0, \widehat{x}_0), (\widehat{AL}_1, \widehat{x}_1)$ are computationally indistinguishable, provided that $AL_0(x_0)$ and $AL_1(x_1)$ output the same result after a similar number of steps. In Section 5.1, we show how this can be achieved assuming **iO**.

Static v.s. Adaptive Security Throughout this work, we consider statically secure garbling schemes; that is, the privacy guarantees only hold when the entire computation (AL, x) to be garbled is chosen statically. In the literature, stronger privacy guarantees have been considered [BHR12a, BHK13], allowing the input x to be chosen maliciously and adaptively depending on the garbled $\widehat{AL}$.

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.

[10]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 $AL.T$.

Definition 4 (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 5 (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 6 (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 [LO13, GHL+14] 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 4, 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 7 (Garbling Scheme for Circuits with Independent Input Encoding)**.** *A Garbling scheme* $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ *for a deterministic circuit class* $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *has* **independent input encoding** *if the following holds: For every* $\lambda \in \mathbb{N}$*, and every* $C \in \mathcal{C}_\lambda$*,*

- *The algorithm* Garb *on input* $(1^\lambda, C)$ *invokes first* $\mathbf{key} \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda, 1^{|x|})$ *and then* $\widehat{C} \overset{\$}{\leftarrow} \mathsf{Gb}(\mathbf{key}, C)$*, 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* $(\tilde{x}, \mathbf{st}) \overset{\$}{\leftarrow} \mathsf{Sim.Gen}(1^\lambda, |x|)$ *and then* $\tilde{C} \overset{\$}{\leftarrow} \mathsf{Sim.Gb}((1^\lambda, |x|, |C|, C(x), \mathbf{st})$*, where* $\mathsf{Sim.Gen}$ *and* $\mathsf{Sim.Gb}$ *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 [Yao82], 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* $\varepsilon \in (0, 1)$*.*

2.3 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 **iO**, and then discuss about different efficiency guarantees.

Definition 8 (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 $\left\{\mathcal{AL}^T_\lambda \times \mathcal{AL}^T_\lambda \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$, 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,

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

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 **iO** 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 9 (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* $\mathrm{poly}(\lambda, |AL|, AL.n)$ / $\mathrm{poly}(\lambda, |AL|, AL.S)$ / $\mathrm{poly}(\lambda, |AL|)AL.T$.

We note that unlike the case of garbling schemes, the optimal efficiency of an **iO** 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 **iO** for polynomial-sized circuits, NC^1 [BGI+01]; and give definitions of **iO** for polynomial time Turing machines [BCP14] and RAM machines with different efficiency guarantees.

Definition 10 (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 11 (**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 [BCP14, ABG+13] give constructions of **iO** for Turing machines[11] with input-dependent complexity assuming FHE, differing-input obfuscation for circuits, and P-certificates [CLP13]; furthermore, the dependency on input lengths can be removed—leading to a scheme with optimal efficiency—if assuming SNARK instead of P-certificates.

Definition 12 (**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 3 and 5.1 (from **iO** and sub-exp **iO** for circuits respectively) has complexity* $\mathrm{poly}(|AL|, AL.S, \log(AL.T))$*. (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 [LO13, GHL+14] 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 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*

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

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 2 and 8 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. $AL.S = 2^\lambda$).

2.4 Puncturable Pseudo-Random Functions

We recall the definition of puncturable pseudo-random functions (PRF) from [SW14a]. Since in this work, we only uses puncturing at one point, the definition 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* ($\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F}$)*, and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:*

Correctness. *For all outputs K of $\mathsf{PRF.Gen}(1^\lambda)$, all points $i \in \{0,1\}^{n(\lambda)}$, and $K(-i) = \mathsf{PRF.Punc}(K, i)$, 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, $K \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K(i) = \mathsf{PRF.Punc}(K, i)$, 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 [BW13, BGI14, KPTZ13], the GGM tree-based construction of PRFs [GGM86] 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 pseudo-random.

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 5. *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 1 and 2, 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.

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 $|M| \leq D, |\,\mathrm{conf}_t\,| \leq S$, 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 $T^* = T_M(x)$. $\mathrm{conf}_1, \cdots, \mathrm{conf}_{T^*-1}, \mathrm{conf}_{T^*}$ are the sequence of configurations of $M(x)$ until it halts (conf_t is the configuration before the t^{th} step starts). $\mathrm{conf}_{T^*}, \cdots, \mathrm{conf}_{T+1}$ are set for simplicity 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 $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim.Gen}_{\mathsf{CIR}}, \mathsf{Sim.Gb}_{\mathsf{CIR}})$ as described in Definition 7.

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 $\mathbf{key} = \mathrm{key}_1 \| 1^S$.

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

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

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.

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

$$\left\{\mathsf{simu}_{ns}(1^\lambda, M, x)\right\} = \left\{(\tilde{M}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}_{ns}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T_M(x), M(x))\ :\ (\tilde{M}, \tilde{x})\right\}_\lambda \quad (2)$$

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 $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim.Gen}_{\mathsf{CIR}}, \mathsf{Sim.Gb}_{\mathsf{CIR}})$ 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 $(T^*)^{\mathrm{th}}$ 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 $(\widetilde{\mathrm{conf}}_{t+1}, \mathbf{st}_{t+1}) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_{t+1})$ 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 $(\widetilde{\text{conf}}_t, \mathbf{st}_t) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_t)$ and then $\widetilde{\mathbf{C}}_t = \mathsf{Sim.Gb}_{\mathsf{CIR}}(1^\lambda, S, q, out_t, \mathbf{st}_t\ ;\ \beta_t)$, 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 $(\widetilde{\text{conf}}_1, \mathbf{st}_1) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_1)$ using randomness α_1, and setting $\tilde{x} = \widetilde{\text{conf}}_1$.

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 $\mathbf{Q}$, generates the last $T - \gamma$ garbled circuits honestly using the program $\mathbf{P}$, and simulates the γ^{th} garbled circuits using the program $\mathbf{R}$ 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 $\text{conf}_{\gamma+1}$ 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 $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ 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 $t \le \gamma - 1$, 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 $t \ge \gamma + 1$, 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 $\widetilde{\mathbf{C}}_t = \mathbf{R}^{\lambda, S, \text{conf}_{\gamma+1}}(\gamma\ ;\ (\alpha_\gamma, \alpha_{\gamma+1}, \beta_\gamma))$ 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 $\text{conf}_{\gamma+1}$ is a final configuration. Otherwise, the output should be the garbled input corresponding to $\text{conf}_{\gamma+1}$ for the next garbled circuit; since the $\gamma + 1^{\text{st}}$ circuit is generated honestly, we compute $out_\gamma = \widehat{\text{conf}}_{\gamma+1}$ by first computing $\text{key}_{\gamma+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_{\gamma+1})$, and then encoding $\widehat{\text{conf}}_{\gamma+1} = \mathsf{Encode}_{\mathsf{CIR}}(\text{key}_{\gamma+1}, \text{conf}_{\gamma+1})$.
     (Note that the difference between program $\mathbf{Q}$ and $\mathbf{R}$ is that the former prepares the output out_γ using simulated garbled input $\widetilde{\text{conf}}_{t+1}$, whereas the latter using honestly generated garbled input $\widehat{\text{conf}}_{\gamma+1}$.)
  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 $(\widetilde{\text{conf}}_\gamma, \mathbf{st}_\gamma) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_\gamma)$ and $\widetilde{\mathbf{C}}_t = \mathsf{Sim.Gb}_{\mathsf{CIR}}(1^\lambda, S, q, out_\gamma, \mathbf{st}_\gamma\ ;\ \beta_\gamma)$, 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 (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)). 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 $\mathrm{key}_{\gamma+1}$.

  – $\underline{\text{Receive externally honest garbling } (\widehat{\mathbf{C}}_\gamma, \widehat{\mathrm{conf}}_\gamma) \text{ of } (\mathbf{C}_\gamma, \mathrm{conf}_\gamma).}$

  – Simulate the first $\gamma - 1$ circuits using $\mathbf{R}$ and $\mathbf{Q}$, with $\widehat{\mathrm{conf}}_\gamma$ hardwired in $\mathbf{R}$.

- In hyb^γ_{ns}, 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 $\mathrm{conf}_{\gamma+1}$ is a final configuration) or the honestly generated gabled input $\widehat{\mathrm{conf}}_{\gamma+1}$. In other words, $out_\gamma = \mathbf{C}_\gamma(\mathrm{conf}_\gamma)$, 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 $\widetilde{\mathrm{conf}}_\gamma$ of $\widetilde{\mathbf{C}}_\gamma$.

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

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

Then it follows from the security of the circuit garbling scheme $\mathcal{GS}_{\mathsf{CIR}}$ that the distributions of $(\widehat{\mathbf{C}}_\gamma, \widehat{\mathrm{conf}_\gamma})$ and $(\widetilde{\mathbf{C}}_\gamma, \widetilde{\mathrm{conf}_\gamma})$ 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.

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 $i\mathcal{O}$ 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 $\overline{\mathbf{P}} = i\mathcal{O}(\mathbf{P})$, 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 $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim.Gen}_{\mathsf{CIR}}, \mathsf{Sim.Gb}_{\mathsf{CIR}})$.
- An indistinguishability obfuscator $i\mathcal{O}_{\mathsf{CIR}}(\cdot, \cdot)$ for polynomial-sized circuits.
- A puncturable PRF $(\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F})$ 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 1, 2 and 3 are padded to their maximum size.

Figure 1: Circuits used in the construction and simulation of $\mathcal{GS}$

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

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF.Gen}(1^{\lambda})$ and $K_\beta \xleftarrow{\$} \mathsf{PRF.Gen}(1^{\lambda})$.
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 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 $z = \mathsf{Eval}_{\mathsf{CIR}}(\widehat{\mathbf{C}}_t, \widehat{\mathrm{conf}}_t)$; 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 $\mathsf{Garb}, \mathsf{Encode}$ 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 $\mathsf{Garb}_{ns}, \mathsf{Encode}_{ns}$ of the non-succinct scheme, *using pseudo-random coins*, formalized below.

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

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.

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

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 $\mathrm{poly}(\lambda, S)$. Overall, Garb runs in time $\mathrm{poly}(\lambda, |M|, S)$ as claimed.

- Encode run in time the same as the Encode_{ns} algorithm which is $\mathrm{poly}(\lambda, |M|, S)$.

- 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 $\mathrm{poly}(\lambda, |M|, S) \times T^*$, $T^* = T_M(x)$, 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 $\mathrm{poly}(\lambda, |M|, S)$ time, overall Eval runs in time $T_M(x) \times \mathrm{poly}(\lambda, |M|, S)$ 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:

$$\left\{\mathsf{real}(1^\lambda, M, x)\right\} = \left\{(\overline{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, M),\ \hat{x} = \mathsf{Encode}(\mathbf{key}, x) \ :\ (\overline{\mathbb{P}}, \hat{x})\right\}_\lambda \tag{3}$$

$$\left\{\mathsf{simu}(1^\lambda, M, x)\right\} = \left\{(\overline{\mathbb{Q}}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, |x|, |M|, S, T, T_M(x), M(x)) \ :\ (\overline{\mathbb{Q}}, \tilde{x})\right\}_\lambda \tag{4}$$

As discussed in the overview, the simulation will obfuscate the program $\mathbf{Q}$ used for simulating the non-succinct garbled machine $\widetilde{M} = (\widetilde{\mathbf{C}}_1, \cdots, \widetilde{\mathbf{C}}_T)$. More precisely,

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

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$.
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 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 $(\widetilde{\mathrm{conf}}_1, \mathbf{st}_1) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_1)$, where $\alpha_1 = \mathsf{F}(K_\alpha, 1)$; set $\tilde{x} = \widetilde{\mathrm{conf}}_1$.
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 equation (3) and (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 $\mathrm{conf}_{\gamma+1}$ as defined by $\mathsf{CONFIG}(M, x)$.

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{M}^\gamma$*:*
   Obfuscate the circuit $\mathbb{M}^\gamma(t) = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \mathrm{conf}_{\gamma+1}, K_\alpha, K_\beta}(t)$ as described in Figure 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 $\overline{\mathbb{M}}^\gamma \xleftarrow{\$} i\mathcal{O}(1^\lambda, \mathbb{M}^\gamma)$.
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 $\mathbb{M}^\gamma = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \mathrm{conf}_{\gamma+1}, K_\alpha, K_\beta}$: 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:

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

The circuits in Figure 1, 2 and 3 are padded to their maximum size.

Figure 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, We describe circuits $\mathbb{M}_1^\gamma$ to $\mathbb{M}_6^\gamma$. They all have parameters $\lambda, S, M, T^*, y, \mathrm{conf}_{\gamma+1}$ 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 $\mathbb{M}_3^\gamma = (\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widehat{\mathbf{C}}_{\gamma+1}, \widehat{\mathrm{conf}}_{\gamma+1}}$**:** On input $t \in [T]$, does:

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

If $t = \gamma$, set out_γ using $\widehat{\mathrm{conf}}_{\gamma+1}$ 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 $\mathbb{M}_4^\gamma = (\mathbb{M}_4^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\mathrm{conf}}_{\gamma+1}}$**:**

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

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 $\widetilde{\mathrm{conf}}_{\gamma+1}$ 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 1, 2 and 3 are padded to their maximum size.

Figure 3: Circuits used in the security analysis of $\mathcal{GS}$, continued

$\{\mathsf{hyb}^0(1^\lambda, M, x)\} = \{\mathsf{real}(1^\lambda, M, x)\}$; 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 $K_\alpha(\gamma + 1) = \mathsf{PRF.Punc}(K_\alpha, \gamma + 1)$ and $K_\beta(\gamma + 1) = \mathsf{PRF.Punc}(K_\beta, \gamma + 1)$. 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 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 $i\mathcal{O}$ 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 pseudo-randomness 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 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 $i\mathcal{O}$ 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:

$$(\widetilde{\mathrm{conf}}_{\gamma+1}, \mathbf{st}_{\gamma+1}) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S; \alpha'_{\gamma+1}) \tag{5}$$

$$\widetilde{\mathbf{C}}_{\gamma+1} = \mathsf{Sim.Gb}_{\mathsf{CIR}}\left(1^\lambda, S, 1^q, out_{\gamma+1}, \mathbf{st}_{\gamma+1}; \beta'_{\gamma+1}\right) \tag{6}$$

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 (5) and (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 (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 $\widetilde{\mathrm{conf}}_1$ as described in equation (5), and then output $\tilde{x}_4 = \widetilde{\mathrm{conf}}_1$.

It follows from the fact that $\mathbb{M}_5$ computes $(\widetilde{\mathrm{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ 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 $i\mathcal{O}$ 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.

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 6. *Assuming the existence of* $\mathbf{iO}$ *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 $\mathbf{iO}$ 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 6.

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 [LO13, GHL+14], and modifying the construction of Theorem 6 slightly, we can improve the efficiency of $\mathcal{GS}^A$ when the algorithm class is RAM. More precisely, we show the following theorem.

Theorem 7. *Assuming the existence of* $\mathbf{iO}$ *for circuits and one-way functions. There exists a garbling scheme* $\mathcal{GS}^{\mathsf{RAM}}$ *for* RAM *with* linear-*space-dependent complexity. Furthermore, 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 [LO13, GHL+14].

The Basic RAM Garbling Scheme $\mathcal{GS}'$: Theorem 7 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 7 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 7, 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 $\mathbf{key}$ produced is bounded by $\mathrm{poly}(\lambda)$.

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

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

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 $\mathrm{poly}(\lambda, |R|) \times T$. 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 $\mathrm{poly}(\lambda)|x|$.

Instantiation of the Basic Garbling Scheme. We observe that the construction of [LO13, GHL+14] satisfies the above three properties, with some small modifications.

- *independent input encoding:* The construction of [LO13, GHL+14] 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 [LO13, GHL+14] 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 $\mathrm{poly}(\lambda)T_{\mathsf{Gen}}(\lambda, |x|)$, 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 [LO13, GHL+14] is exactly as required above, namely $\mathrm{poly}(\lambda, |R|) \times T$ for the first two and $\mathrm{poly}(\lambda)|x|$ for the last one.[14] Furthermore, its key generation algorithm runs in time linear in the input length $\mathrm{poly}(\lambda)|x|$.

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

[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.

[14]In [LO13, GHL+14], the overhead of garbling is $\mathrm{poly}(\lambda) \times |R| \times \mathrm{poly}\log(n)$, 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.

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 $\mathsf{Sim}' = (\mathsf{Sim.Gen}', \mathsf{Sim.Gb}')$. 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 $\mathrm{poly}(\lambda, |R|)$ overhead (that is, evaluation of $\widehat{R}, \hat{x}$ takes $\mathrm{poly}(\lambda, |R|)T_R(x)$ steps). In comparison, the previous general construction has *polynomial* space dependent complexity and $\mathrm{poly}(\lambda, |R|, S)$ 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". 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 of size at least $\mathrm{poly}(\lambda, \Phi) = \mathrm{poly}(\lambda, |R|, S)$. 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 $\mathrm{poly}(\lambda, |R|, S)$.

Better efficiency: To avoid the polynomial overhead due to obfuscation, we instead use an $i\mathcal{O}$ for circuits with quasi-linear complexity $|C|\,\mathrm{poly}(\lambda, n)$, where $|C|$ is the size of the circuit obfuscated and n is the length of the input. As shown in Appendix A, such an scheme can be constructed generically from any $i\mathcal{O}$ (for circuits), puncturable PRF, and randomized

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

encoding that is local (as defined in Appendix A 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 computed in time $\mathrm{poly}(\lambda, |R|)S$ (roughly the same as Φ). Therefore, the size of the new garbled RAM (and the complexity for generating it) is,

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

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

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

This concludes Theorem 7.

Remark 2 (RAM Garbling Scheme with Complexity Linear in the Program Size)**.** *The RAM garbling scheme of Theorem 7 produces garbled RAM of size* $\mathrm{poly}(\lambda, |R|) \times S$ *and run-time* $\mathrm{poly}(\lambda, |R|)T^*$ *(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 [LO13, GHL+14] 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 7 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 7, we omit the details here.*

5 Applications

In this section, we address two of our main applications of succinct garbling schemes: succinct **iO** 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 $\mathcal{RE} = (\mathsf{REnc}, \mathsf{Dec})$ for $\{\mathcal{AL}_\lambda\}$ consists of a randomized encoding algorithm REnc and a decoding algorithm Dec. $\mathsf{REnc}(1^\lambda, AL, x)$, 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 $\mathsf{REnc} = \mathsf{Garb} \circ \mathsf{Encode}$ is given by

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

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

5.1 From Randomized Encodings to iO

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 **iO**, $i\mathcal{O}^C$, 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 3 and 4, we directly obtain **iO** for TM and RAM with space-dependent complexity.

Theorem 8. *Assume a sub-exponentially indistinguishable **iO** 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 [CLTV15].

Probabilistic **iO***.* Let $i\mathcal{O}$ and F be 2^{λ^ε}-indistinguishable **iO** 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 **iO** obfuscation of $\Pi^{C,k}$ for a randomly sampled key C, that is,

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

where $\lambda' = (\lambda + n)^{1/\varepsilon}$ for $n = C.n$, so that **iO** and F are $\text{negl}(\lambda)2^n$-indistinguishable. The work of [CLTV15] 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 $\text{negl}(\lambda)2^{-n}$ indistinguishable (given z). Summarizing,

Lemma 1 (**piO** for Circuits [CLTV15])**.** *Assume sub-exponentially indistinguishable* ***iO*** *for circuits $i\mathcal{O}^C$, 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 $\{C_\lambda \times C_\lambda \times \{0,1\}^{\text{poly}(\lambda)}\}$, if it holds that for every non-uniform* PPT *adversary $\mathcal{R}$, and input x,*

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

the following ensembles are computationally indistinguishable:

$$\Big\{C_1, C_2, pi\mathcal{O}(1^\lambda, C_1), z\Big\}_\lambda \approx \Big\{C_1, C_2, pi\mathcal{O}(1^\lambda, C_2), z\Big\}_\lambda$$

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

Proof Sketch of Lemma 1. 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, $\hat{C}_b \xleftarrow{\$} pi\mathcal{O}(1^\lambda, C_1)$ is indistinguishable to the obfuscation of $C'_b \xleftarrow{\$} i\mathcal{O}(C'_b)$ 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 $i\mathcal{O}(C'_0)$ and $i\mathcal{O}(C'_1)$ 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 $\text{negl}(\lambda)2^{-n}$-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 $x \le i$ 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 $O(\text{negl}(\lambda)2^{-n})$. Thus, by a hybrid argument, obfuscations of C_1 and C_2 are indistinguishable. This concludes the lemma. □

*Construction of **iO** for General Algorithms.* Using Lemma 1, we now prove Proposition 3.

Given $2^{-\lambda^{\varepsilon}}$-indistinguishable **iO** $i\mathcal{O}^C$ 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 **iO** for the a general algorithm class $\{\mathcal{AL}_\lambda\}$ is defined as follows,

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

where the security parameter $\lambda' = (\lambda+n)^{1/\varepsilon}$ for $n = AL.n$ so that REnc is $\text{negl}(\lambda)2^n$-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 $\{\mathcal{AL}^T_\lambda \times \mathcal{AL}^T_\lambda \times \{0,1\}^{\text{poly}(\lambda)}\}$, 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

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

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 $C'_b = \mathsf{REnc}(1^{\lambda'}, AL_b, \cdot)$. 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 $\text{negl}(\lambda)2^n$-indistinguishable, given x, z, C'_1, C'_2. Thus, it follows from Lemma 1 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 **iO** and PRF as well as the parameters of their sub-exponential security; moreover, if the underlying **iO** has linear-time-dependent complexity, p_{pIO} also depends linearly in $|C|$ (still polynomially in λ and $C.n$). Let $T_{\mathsf{REnc}}(\lambda', |AL|, n, m, S, T)$ be the running time of $\mathsf{REnc}(1^{\lambda'}, AL, x)$. Overall, the running time of $i\mathcal{O}^A(1^\lambda, AL)$ is,

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

Therefore,

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

- If $\mathcal{RE}$ and the underlying **iO** has linear-time-dependent complexity (that is, T_{REnc} depends linearly on T and T_{pIO} depends linearly on $|C|$), so does $i\mathcal{O}^A$.

This concludes the proof of Proposition 3.

A corollary: output-independence. The size of the randomized encodings (or garbling schemes) described in previous sections depends (linearly) on the output of the encoded computation, accordingly so does the succinct **iO** construction described in this section. We start by noting that in the succinct **iO** construction this dependence can be easily removed. Concretely, rather than considering the machine $AL(x)$ that for any input x might have am m-bit output y, we can consider a new single-bit machine $AL'(x, i)$ that given additional input $i \in \{0,1\}^{\log m}$, outputs y_i. Observe that if AL_0 and AL_1 compute the same function then clearly so do their single-bit versions AL'_0 and AL'_1. The overhead is only polylogarithmic in the output-size m. Thus, we directly obtain succinct **iO** that is output-independent. We note that this does not involve making any additional computational assumptions. (Note that we only increase the input size logarithmically, and thus the exponential loss in the input incurred by the transformation given by Theorem 3 is only polynomial.)

Next, we observe that this directly implies indistinguishability-based succinct randomized encodings that are output-independent. The encoding of AL, x simply consists of an (output-independent) obfuscation of a machine that has no input and output $AL(x)$. (Note that this only requires polynomial **iO**, since the exponential blowup in the input size of the transformation given by Theorem 3 is completely avoided.)

More Efficient Construction. Evaluating the **iO** for TM and RAM obtained in Theorem 8 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 $T_{AL}(x) \times \mathrm{poly}(\lambda, |AL|, S)$. 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 7.

Theorem 9. *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* $\mathrm{poly}(\lambda, n, |R|) \times S$*, and evaluation of the obfuscated program on input* x *takes time* $\mathrm{poly}(\lambda, n, |R|) \times (T_R(x) + S)$*, with* $n = R.n$ *and* $S = R.S$*.*

Towards the above theorem, consider instantiating the Proposition 3 using the RAM garbling scheme of Theorem 7 and a sub-exponentially secure $i\mathcal{O}$ scheme with quasi-linear complexity (implied by sub-exponentially secure $i\mathcal{O}$ and OWF as shown in Appendix A). Recall that the RAM garbling scheme of Theorem 7 has linear-space-dependent complexity $\mathrm{poly}(\lambda, |R|) \times S$ and evaluation time $\mathrm{poly}(\lambda, |R|) \times (T^* + S)$ with $T^* = T_R(x)$; such a garbling scheme leads to a randomized encoding algorithm REnc 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 $\mathrm{poly}(\lambda, |R|, n)S$. Therefore, its evaluation time is now $\mathrm{poly}(\lambda, |R|, n) \times S + \mathrm{poly}(\lambda, |R|) \times (T^* + S)$, which is $\mathrm{poly}(\lambda, |R|, n) \times (S + T^*)$.

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* $\mathrm{poly}(\lambda)(|R| + S)$ *and evaluation complexity of* $\mathrm{poly}(\lambda)(|R| + S + T^*)$*. When using such a RAM garbling scheme as* *the underlying scheme in our construction of IO scheme for RAM, we obtain an* ***iO*** *scheme with complexity* $\mathrm{poly}(\lambda, n)(|R| + S)$ *and evaluation time* $\mathrm{poly}(\lambda, n)(|R| + S + T^*)$*.*

5.2 Publicly-Verifiable Delegation, $\mathcal{SNARG}$s for 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 **iO** and the work of [SW14b].

5.2.1 P-delegation

A delegation system for **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 $z = (AL, x, \lambda)$, $\mathcal{G}$ generates a message σ. The prover, given (z, σ), produces a proof π attesting that AL accepts x 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 (**P**-Delegation)**.** *A prover and verifier* $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ *constitute a delegation scheme for* **P** *if it satisfies:*

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

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

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

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

3. **Optimal verification and instance-based prover efficiency:** *There exists a (universal) polynomial* p *such that for every* $z = (AL, x, \lambda)$*:*
   - *the verifier algorithms* $(\mathcal{G}, \mathcal{V})$ *run in time* $p(\lambda, |AL|, |x|, \log AL.T)$*;*
   - *the prover* $\mathcal{P}$ *runs in time* $p(\lambda, |AL|, |x|)T_{AL}(x)$*.*

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* $z = (AL, x, \lambda)$*:*
   - *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 $(\mathsf{REnc}, \mathsf{Dec})$ be a randomized encoding scheme. We describe $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ as follows. Let $z = (AL, x, \lambda)$ be a tuple consisting of an algorithm, input, and security parameter.

Generator $\mathcal{G}(z)$:
For $r \leftarrow \{0,1\}^\lambda$, let $AL'(x,r)$ be the machine that returns r if $AL(x) = 1$ and $\perp$ otherwise. $\mathcal{G}$ generates and outputs $\sigma \leftarrow \mathsf{REnc}(1^\lambda, \Pi', (x,r))$ 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 10. *If* $(\mathsf{REnc}, \mathsf{Dec})$ *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 $(\mathsf{REnc}, \mathsf{Dec})$. Also, note that the running time of the verifier algorithms $(\mathcal{G}, \mathcal{V})$ is related to the running time of REnc. Therefore, it also follows directly that if $(\mathsf{REnc}, \mathsf{Dec})$ 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 $(\mathsf{REnc}, \mathsf{Dec})$ 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 $z = (AL, x, \lambda)$ where AL does not accept x and $AL.T \leq p(\lambda)$, we have that

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

Let $\mathcal{Z}$ be the sequence of such $z = (AL, x, \lambda)$ and consider any $z \in \mathcal{Z}$. Recall that $\mathcal{G}(z)$ samples $r \leftarrow \{0,1\}^\lambda$ and outputs $\sigma \leftarrow \mathsf{REnc}(1^\lambda, AL', (x,r))$ and $\tau \leftarrow f(r)$. Since AL does not accept x, we have that $AL'(x,r)$ outputs $\perp$. By the security of $(\mathsf{REnc}, \mathsf{Dec})$, there exists a PPT simulator Sim such that the ensembles $\{\mathsf{REnc}(1^\lambda, AL', (x,r)\}_{r \in \{0,1\}^\lambda, z \in \mathcal{Z}}$ and $\{\mathsf{Sim}(1^\lambda, \perp, AL', 1^{|x|+|r|})\}_{r \in \{0,1\}^\lambda, z \in \mathcal{Z}}$ are indistinguishable. Therefore, given a simulated $\sigma \leftarrow \mathsf{Sim}(1^\lambda, \perp, AL', 1^{|x|+|r|})$ 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

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

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. □

5.2.2 $\mathcal{SNARG}$s for **P**

A succinct non-interactive argument system ($\mathcal{SNARG}$) for **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,x as before. We require that*

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

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

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

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

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 $z=(AL,x,\lambda)$, where $AL.T\leq T(\lambda)$, $|AL,x|\leq n(\lambda)$, and AL does* **not** *accept x:*

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

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

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

- *the prover $\mathcal{P}$ runs in time $p(\lambda, |AL|, |x|)T_{AL}(x))$.*

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 $\mathbf{P}$. 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 [KRR14]).

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

Let $z = (AL, x, \lambda)$ be a tuple consisting of an algorithm, input and security parameter such that $|AL, x| \leq n$ 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 $K \leftarrow \mathsf{PRF.Gen}(1^\lambda)$.

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 11. *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 **iO**. The running time of $\mathcal{G}(\lambda, T, n)$ is polynomial in λ and the maximum running time of $\mathcal{G}_d$ on inputs $z = (AL, x, \lambda)$ where $|AL, x| \le n$ and $AL.T \le T$. 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 **iO** and the punctured PRF $(\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F})$. 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 $z = (AL, x, \lambda)$, where $AL.T \le T(\lambda)$, $|AL, x| \le n(\lambda)$, and AL does not accept x:

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

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 $K \leftarrow \mathsf{PRF.Gen}(1^\lambda)$ and punctures it at the input z to obtain a punctured key $K_z \leftarrow \mathsf{PRF.Punc}(K, z)$.

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 $z = (AL, x, \lambda)$ where AL does not accept x the following holds:

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

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

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 $z \in \mathcal{Z}$, $\mathcal{A}_d(z, \mathcal{P}_d^*) = \mathcal{A}_d(z, \mathcal{P}_d^{\mathsf{Hyb}})$. 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 $\mathcal{P}_d^{\mathsf{Hyb}}$, 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 x, 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 $z \in \mathcal{Z}$,

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

. We first observe that in the experiment corresponding to $\mathcal{A}_d(z, \mathcal{P}_d^{\mathsf{Hyb}})$, 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. *The following ensembles are computationally indistinguishable.*

$$\{(\sigma, \tau) : (\sigma, \tau, \pi) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z, \sigma_d, \tau_d), (\sigma_d, \tau_d) \leftarrow \mathcal{G}_d(z)\}_{z \in \mathcal{Z}} \tag{7}$$

$$\approx_c \{(\sigma, \tau) : (\sigma, \tau) \leftarrow \mathcal{G}(\lambda, T(\lambda), n(\lambda))\}_{z \in \mathcal{Z}} \tag{8}$$

Proof. Recall that in ensemble (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 (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 (7) and (8) is the same. Indistinguishability follows from the security of **iO** and that of $(\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F})$ in the standard way. We provide a brief overview.

Consider a hybrid ensemble that is identical to ensemble (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 (7). Furthermore, the circuits obfuscated as σ and τ in this hybrid ensemble and in ensemble (8) are functionally equivalent. Hence, by the security of **iO**, ensemble (8) is indistinguishable from the hybrid ensemble. A hybrid argument completes the proof of the claim. $\square$

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 $\mathcal{P}^{\mathsf{Hyb}}_d$ 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 $\mathcal{P}^*_d$. $\square$

5.2.3 Succinct Perfect NIZK for NP

In this section we briefly note that using the succinct indistinguishability obfuscator from Section 5.1 in the construction of [SW14a] 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 [SW14a] construction and how it can be made succinct using succinct indistinguishability obfuscation.

In [SW14a], 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. [SW14a] 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 [SW14a] construction is based on).

Theorem 12. *(Follows from [SW14a]) 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].
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 $\text{poly}(\lambda)(|AL| + AL.S + |x|)$ and the prover complexity for a T-time computation $AL(x)$ is $\text{poly}(\lambda)(|AL| + AL.S + T)$. Furthermore, combining this delegation scheme with Theorem 11, while applying the trick of "obfuscating in a piecemeal fashion" as in Section 4.1 and 5.1, we obtain a $\mathcal{SNARG}$ for **P** with the same verification and prover efficiency.*

*Finally, using the more efficiency succinct **iO** of Remark 3 in Theorem 12, the size of the CRS can be improved to $\text{poly}(\lambda, n, \tau(n)) \cdot s(n)$, and the prover efficiency can be improved to $\text{poly}(\lambda, n, \tau(n))(s(n) + T)$.*

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 [GHRW14b]. 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.

References

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

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th Annual Symposium on Foundations of Computer Science*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[16] The perfect NIZK construction of [SW14a] 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.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010: 37th International Colloquium on Automata, Languages and Programming, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163, Bordeaux, France, July 6–10, 2010. Springer, Berlin, Germany.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, California, USA, October 22–25, 2011. IEEE Computer Society Press.

[AJL$^+$12] 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. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

[App11a] Benny Applebaum. Key-dependent message security: Generic amplification and completeness. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 527–546, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.

[App11b] 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.

[App14] 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.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 326–349, Cambridge, Massachusetts, USA, January 8–10, 2012. Association for Computing Machinery.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 52–73, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 505–514, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[BGI$^+$01] 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.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.

[BGT14] Nir Bitansky, Sanjam Garg, and Sidharth Telang. Succinct randomized encodings and their applications. *IACR Cryptology ePrint Archive*, 2014:771, 2014.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 423–444, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

[BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 398–415, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 134–153, Beijing, China, December 2–6, 2012. Springer, Berlin, Germany.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796, 2012.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

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

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT (2)*, pages 280–300, 2013.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for ram programs. In *47th Annual ACM Symposium on Theory of Computing*. ACM Press, 2015.

[CIJ$^+$13] 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.

[CLP13] 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.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 468–497, 2015.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 54–74, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Berlin, Germany.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, Maryland, USA, May 31 – June 2, 2009. ACM Press.

[GGH$^+$13a] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGH$^+$13b] 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.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 518–535, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014.

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

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Germany.

[GHL$^+$14] 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.

[GHRW14a] Gentry, Halevi, Raykova, and Wichs. Outsourcing private ram computation. *Proc. of FOCS 2014*, 2014.

[GHRW14b] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private ram computation. In *55th Annual Symposium on Foundations of Computer Science*, 2014.

[GKP$^+$13a] 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.

[GKP$^+$13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In *47th Annual ACM Symposium on Theory of Computing.* ACM Press, 2015.

[GLR11] 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.

[GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014.

[IK00] 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*, pages 294–304, Redondo Beach, California, USA, November 12–14, 2000. IEEE Computer Society Press.

[IK02a] 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.

[IK02b] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[IPS15] 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.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *47th Annual ACM Symposium on Theory of Computing*. ACM Press, 2015.

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

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 485–494, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[Kv99] Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *31st Annual ACM Symposium on Theory of Computing*, pages 659–667, Atlanta, Georgia, USA, May 1–4, 1999. ACM Press.

[LO13] 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.

[LP14] Huijia Lin and Rafael Pass. Succinct garbling schemes and applications. *IACR Cryptology ePrint Archive*, 2014:766, 2014.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 1219–1234, New York, NY, USA, May 19–22, 2012. ACM Press.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MV99] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *40th Annual Symposium on Foundations of Computer Science*, pages 71–80, New York, New York, USA, October 17–19, 1999. IEEE Computer Society Press.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Berlin, Germany.

[SW14a] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *Proc. of STOC 2014*, 2014.

[SW14b] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

A Obfuscating Circuits with Quasi-Linear Blowup

In general, when considering **iO** for circuits, the size of an obfuscation $|i\mathcal{O}(C)|$ (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., [GGH+13b]) the blow-up is quadratic (see discussion in [GHRW14a]). 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 5.1 to construct succinct **iO** from succinct randomized encodings, which in turn go back to the bootstrapping technique of Applebaum [App14]. Concretely, we rely on plain randomized encodings [IK02b, AIK06] 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 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 $s \cdot \mathrm{poly}(\lambda, n)$ as required. The proof of security relies on a variant of the probabilistic **iO** argument invoked in Section 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 $\mathcal{RE} = (\mathsf{REnc}, \mathsf{Dec})$ for circuits is said to be local if*

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

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

Such randomized encodings can be constructed based on any one-way function [Yao82, AIK06].

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

- A randomized encoding $\mathcal{RE} = (\mathsf{REnc}, \mathsf{Dec})$ 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 $(\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F})$.

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 $K_1, \ldots, K_r \leftarrow \mathsf{PRF.Gen}(1^\lambda)$.
3. For each $i \in [s]$, obfuscate using $i\mathcal{O}$ the circuit $\mathbb{C}_i$ that has hardwired $\{K_j : j \in S_i\}$ 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

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

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

We use a probabilistic **iO** argument similar to the one used in Section 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 $\mathbb{C}_1^j, \dots, \mathbb{C}_s^j$ 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 5.1, we puncture the underlying PRFs at this point j, and hardwire the values $\widehat{C}_1(x; R|_{S_1}), \dots, \widehat{C}_s(x; R|_{S_s})$ (or $\widehat{C'}_1(x; R|_{S_1}), \dots, \widehat{C'}_s(x; R|_{S_s})$, 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). $\square$