A Simple Construction of iO for Turing Machines*

Sanjam Garg and Akshayaram Srinivasan

University of California, Berkeley
{sanjamg,akshayaram}@berkeley.edu

Abstract. We give a simple construction of indistinguishability obfuscation for Turing machines where the time to obfuscate grows only with the description size of the machine and otherwise, independent of the running time and the space used. While this result is already known [Koppula, Lewko, and Waters, STOC 2015] from $i\mathcal{O}$ for circuits and injective pseudorandom generators, our construction and its analysis are conceptually much simpler. In particular, the main technical component in the proof of our construction is a simple combinatorial pebbling argument [Garg and Srinivasan, EUROCRYPT 2018]. Our construction makes use of indistinguishability obfuscation for circuits and somewhere statistically binding hash functions.

1 Introduction

Indistinguishability Obfuscation ($i\mathcal{O}$) [BGI+12, GGH+13] is a central primitive in cryptography giving rise to new and powerful cryptographic applications [SW14, GGHR14]. $i\mathcal{O}$ requires that for any two circuits C_0 and C_1 computing the exact same functionality, obfuscation of C_0 is computationally indistinguishable from the obfuscation of C_1. While circuits are powerful enough to simulate other models of computation such as Turing machines or RAM programs [PF79], a drawback of using them is that size of the circuit (and hence the size of obfuscation) grows with both the running time and the space of the computation. In a beautiful work Koppula, Lewko and Waters [KLW15] (building on prior work [BGL+15, CHJV15]) showed a method for removing this limitation by giving a construction of succinct $i\mathcal{O}$ for Turing machines from $i\mathcal{O}$ for circuits and injective pseudorandom generators. By succinct, we mean that the time to obfuscate a machine grows only with its description size and is otherwise independent of its running time and its space complexity.

Our Contribution. In this paper, we give a *simple* construction of succinct indistinguishability obfuscation for Turing machines from sub-exponentially secure

* Research supported in part from 2017 AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

$i\mathcal{O}$ for circuits and sub-exponentially secure somewhere statistically binding hash functions [HW15, KLW15]. Our new construction is simple to describe and its analysis is much simpler than the previous works. Inspired by [GS18a], the main technical component in our security proof is a simple combinatorial pebbling argument.

In a bit more detail, we achieve the above new result by first giving a new construction of *succinct randomized encoding* [AIK04, CHJV15, BGL$^+$15, App17] from polynomially hard indistinguishability obfuscation for circuits and laconic oblivious transfer [CDG$^+$17, DG17, BLSV18, DGHM18].[1] A randomized encoding allows to encode a Turing machine M, an input x and a time bound t to $\widehat{M}_{x,t}$. Given $\widehat{M}_{x,t}$, the decoding procedure recovers $M(x)$ which is the output of M on input x obtained in time t. The security property requires that the distribution of $\widehat{M}_{x,t}$ does not leak anything about x except $M(x)$. A randomized encoding is said to be *succinct* if the encoding procedure runs in time that is polynomial in the security parameter, the machine description size and the input size and is otherwise independent of the time and space complexity of M. Next, to construct succinct $i\mathcal{O}$ for Turing machines, we use a transformation from any succinct randomized encoding (with sub-exponential security) to succinct $i\mathcal{O}$ for Turing machines given in the works of [CHJV15, BGL$^+$15]. This yields the desired result.

1.1 Overview

In this section, we give a high level overview of our construction of succinct randomized encodings and the security proof.

Starting point. The starting point of our work is the construction of *semi-succinct* randomized encodings for Turing machines in [CHJV15, BGL$^+$15] based on $i\mathcal{O}$ for circuits and Yao's garbling scheme. Semi-succinct randomized encodings require that the time to encode a machine to be independent of its running time but could depend on the space complexity of the computation. In particular, it is a weaker requirement when compared to full succinctness wherein we also require the time to encode a machine to be independent of the space complexity. Below we start by recalling this construction and explain why it achieves only semi-succinctness when compared to full succinctness.

The encoding procedure is given as input a Turing machine M, an input x and a time bound t and it has to output a randomized encoding $\widehat{M}_{x,t}$. The first step in the above works is to reduce the machine M to a "succinctly describable" circuit C that computes the same function as that of M. We say that a circuit is succinctly describable if there exists a "small" circuit C_{sc} that on

[1] Note that [CDG$^+$17] also described a construction of laconic oblivious transfer from witness encryption [GGSW13] and somewhere statistically binding hash functions. Since witness encryption can be instantiated from $i\mathcal{O}$ for circuits and one-way functions (which is implied by somewhere statistically binding hash functions), we obtain our main result from $i\mathcal{O}$ for circuits and somewhere statistically binding hash functions.

input any gate index, outputs the binary function computed by that gate along with the description of its input and output wires. Next, these works observed that Yao's garbling procedure is highly "local", meaning that given only the local information about a gate (which includes its input, output wires and the functionality computed by it), Yao's garbling procedure can output the garbled encryption table corresponding to that gate. Now, these two ideas are combined in an elegant way to obtain a randomized encoding of a Turing machine. To give more details, the encoding consists of an obfuscated circuit that on input any gate index, outputs the garbled encryption table corresponding to that gate. Specifically, this circuit uses the succinct description to obtain the binary logic computed by the gate along with the description of the input and output wires. It uses a (puncturable) PRF key to obtain the labels corresponding to the input and the output wires and outputs the Yao's garbled table corresponding to that gate (using randomness derived from the puncturable PRF key). The encoding procedure outputs this obfuscation along with the labels corresponding to the input x. The decoding procedure evaluates this obfuscation on every gate index to obtain the garbled tables corresponding to every gate and then evaluates the garbled circuit to obtain the output.

Let us now describe the simulator for the above construction. Recall that the simulator on input $M(x)$ must output a randomized encoding such that the distribution of the simulator's output is computationally indistinguishable to the distribution of an honestly generated encoding. The simulator in these works obfuscates a circuit that on input any gate number, outputs the simulated Yao's garbled table. Intuitively, it should follow from the security of Yao's garbled circuit construction that the real garbled tables are computationally indistinguishable to the simulated garbled tables. However, for the proof to go through, these works cannot change the distribution of all the garbled gates from the real to simulated in one shot. Rather, they use a careful hybrid argument wherein they change the distribution of the garbled tables from the real to simulated for one gate at a time and this where the succinctness takes a hit. Let us now explain this in more detail.

Recall from the proof of Yao's garbled circuit construction [LP09], that each hybrid corresponds to a particular distribution of garbled encryption tables (also called as configurations in [HJO$^+$16]). In a particular configuration, a garbled gate can either be in three modes: the real mode, or the input dependent simulation mode, or the simulated mode. The real mode is one where in the garbled encryption tables are distributed exactly as in the construction. In the input dependent simulation mode, all the entries of the garbled encryption table encrypt a single label and this label corresponds to the output of that gate. In the simulated mode, every entry of the garbled encryption table encrypts a single label and this label corresponds to the bit 0. The real world distribution corresponds to a configuration wherein each garbled gate is in the real mode and the simulated configuration is one in which each garbled gate is in the simulated mode. In order to go from the real world distribution to the simulated distribution, we need to go over a sequence of hybrids. Each hybrid change corresponds

to changing the configuration of a particular gate. These changes can be made according to the following two rules:

- **Rule A:** A garbled gate can be changed from the real mode to input dependent simulation mode if all its fan-in gates are in input dependent simulation mode.
- **Rule B:** A garbled gate can be changed from an input dependent simulation mode to the simulated mode if all its fan-out gates are in input dependent simulation mode.

A direct consequence of such a hybrid argument is that the obfuscated circuit (in the construction of succinct randomized encoding) in a particular hybrid must somehow encode the outputs of all the gates that are in the input dependent simulation mode. Notice that in general, the fan-out of a gate could be as large as the space of the computation (denoted by s). Thus, to change one garbled gate from input dependent simulation mode to the simulated mode, we must encode the outputs of at most s gates in the obfuscated circuit. Thus, the size of the obfuscated circuit in this intermediate hybrids grows with s. Thus, to use $i\mathcal{O}$ security, the real world obfuscation must also be padded to the size of the circuit in the intermediate hybrid and hence, these works could only achieve semi-succinctness. Because of the above-mentioned challenges, this approach seemed insufficient for realizing full succinctness. Thus, Koppula, Lewko and Waters [KLW15] gave a very different approach for realizing full succinctness. However, unfortunately, their realization is rather involved.

Our Approach. In this work, we start with the above-mentioned approach followed in the realization of semi-succinct iO constructions but employ a crucial technique to achieve full succinctness. Specifically, to achieve full succinctness, we use a *linearized garbling scheme* (introduced in the work of Garg and Srinivasan [GS18a]) in place of Yao's garbling scheme. Informally, a linearized garbled circuit helps in "flattening" the underlying circuit which may have large width into a circuit with width 1. Intuitively, such a flattening would be helpful as the size of intermediate obfuscations may not have to grow with the width of the circuit (which is proportional to the space complexity). In the rest of the overview, we give an informal description of the linearlized garbled circuit, state its properties and explain the combinatorial pebbling game that forms the main crux of the proof. This approach allows us to achieve a simpler construction than Koppula, Lewko and Waters [KLW15].

Linearized Garbled Circuits. To understand the concept of a linearized garbled circuits[2], it is best to view the circuit C as a *sequence of step circuits*. In more details, we will consider C as a sequence of step circuits along with a database/memory D. The i-th step circuit implements the i-th gate (with some topological ordering of the gates) in the circuit C. The database D is initially loaded with the input x and contents of the database represent the state of the

[2] This paragraph is taken verbatim from [GS18a].

computation. That is, the snapshot of the database before the evaluation of the i-th step circuit contains the output of every gate $g < i$ in the execution of C on input x. The i-th step circuit reads contents from two pre-determined locations in the database and writes a bit to location i. The bits that are read correspond to the values in the input wires for the i-th gate. The output of the circuit is easily derived from the contents of the database at the end of the computation.

To garble a circuit C, we must garble each of the step circuits and the database D. To draw a parallel with the Yao's garbling scheme, the garbled encryption tables are now replaced with garbled step circuits. As in the of Yao's garbling procedure, the task of garbling the step circuits has the desired locality property, meaning that given only the locations accessed by the step circuit and the functionality computed by it, we can computed the garbled version of that particular step circuit. Furthermore, we can think of the distributions wherein a step circuit is in real mode, or in input dependent simulation mode, or in simulated mode as natural extensions of the same notions for a garbled gate. For the sake of keeping things simple in the introduction, we wouldn't be going into the exact details of the actual distributions in these three modes.

Now we are ready to state the properties of a linearized garbled circuit. We say a garbling scheme to be linearized if it satisfies the following two properties:

1. **Rule A:** A step circuit can be changed from the real mode to an input dependent simulation mode (or, vice-versa) if the previous step circuit is in input dependent simulation mode. This restriction however, does not apply to the first step circuit i.e., it can always be changed from real to input dependent simulation mode (or, vice-versa).
2. **Rule B:** A step circuit can be changed from input dependent simulation mode to the simulated mode if the previous step circuit is in input dependent simulation mode and all the subsequent step circuits are in simulated mode. This rule must be contrasted with the corresponding rule for Yao's garbled circuits wherein we must maintain all the gates which fan-out from this particular gate in input dependent simulation mode.

Garg and Srinivasan [GS18a] constructed such a linearized garbling scheme from laconic oblivious transfer [CDG$^+$17].[3] We will now show that how this linearized garbling structure is helpful in obtaining a fully succinct randomized encoding scheme.

Pebbling Game. Now, let us explain how the concept of linearized garbled circuit helps us in achieving full succinctness. The simulator for our construction of succinct randomized encoding is exactly the same as in the previous constructions [CHJV15, BGL$^+$15]. In particular, it obfuscates a circuit that on input any step circuit index, outputs the garbled version of that step circuit in the simulated mode. In the real world distribution, all the step circuits are garbled in the real mode whereas in the simulated distribution all the step circuits are

[3] As mentioned in the introduction, a laconic oblivious transfer can be constructed from $i\mathcal{O}$ for circuits and somewhere statistically binding hash functions.

garbled in the simulated mode. The goal is to change all the step circuits from the real mode to the simulated mode where in each step/hybrid, we can use either one of the above two rules to change the configuration of a particular gate. In order to keep the size of the intermediate obfuscations small, we need to minimize the number of step circuits that are present in the input dependent simulation mode. This is because for every step circuit that is present in the input dependent simulation mode, we must hardcode the output of the gate in the obfuscation and hence the size of the obfuscation grows with this number. These requirements can be abstractly modeled as the following pebbling game whose description is taken verbatim from [GS18a].

Consider the positive integer line $1, 2, \ldots, N$. We are given pebbles of two colors: gray and black . A black pebble corresponds to a step circuit in the simulated mode and a gray pebble corresponds to a step circuit in the input dependent simulation mode. A position without any pebble corresponds to real garbling. We can place the pebbles on this positive integer line according to the following two rules:

Rule A: We can place or remove a gray pebble in position i if and only if there is a gray pebble in position $i-1$. This restriction does not apply to position 1: we can always place or remove a gray pebble at position 1. This rule captures the first requirement of a linearized garbling scheme.

Rule B: We can replace a gray pebble in position i with a black pebble as long as all the positions $> i$ have black pebbles and there is a gray pebble in position $i-1$ or if $i = 1$. This rule captures the second requirement of a linearized garbling scheme.

Optimization goal of the pebbling game. The goal is to pebble the line $[1, N]$ such that every position has a black pebble while minimizing the number of gray pebbles that are present on the line at any point in time.

Any strategy for the above pebbling game that uses a maximum of ℓ gray pebbles gives a randomized encoding scheme where the time to encode grows with ℓ. We note that the same pebbling game was considered in the work of [GS18a] in the context of constructing adaptive garbled circuits with optimal online complexity. Using the pebbling strategy considered in their work (that uses $\log N$ gray pebbles), we give a construction of randomized encoding scheme where the time to encode grows only with $\mathsf{poly}(|M|, |x|, \lambda, \log T)$ where T is the running time of the computation. This gives us the desired succinctness.

1.2 Concurrent Work

In a concurrent and independent work, Ananth and Lombardi [AL18] gave a construction of succinct randomized encoding from polynomially hard compact functional encryption and laconic oblivious transfer. They defined an abstraction called as strong locally simulatable garbling schemes and then used it to construct a succinct randomized encoding. At a conceptual level, the notion of strong locally simulatable garbling scheme is similar to our notion of linearized

garbling schemes and hence the underlying techniques used in both these papers are similar. We remark that even our construction can be instantiated from polynomially hard compact functional encryption using the works of [AJ15, BV15] as the size of the input to the obfuscation scheme is $O(\log \lambda)$ where λ is the security parameter.

2 Preliminaries

Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$ there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$ we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. For a probabilistic algorithm A, we denote $A(x; r)$ to be the output of A on input x with the content of the random tape being r. When r is omitted, $A(x)$ denotes a distribution. For a finite set S, we denote $x \leftarrow S$ as the process of sampling x uniformly from the set S. We will use PPT to denote Probabilistic Polynomial Time. We denote $[a]$ to be the set $\{1, \ldots, a\}$ and $[a, b]$ to be the set $\{a, a+1, \ldots, b\}$ for $a \leq b$ and $a, b \in \mathbb{Z}$. For a binary string $x \in \{0,1\}^n$, we will denote the i^{th} bit of x by x_i. We assume without loss of generality that the length of the random tape used by all cryptographic algorithms is λ. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function.

2.1 Succinct Circuits

We now recall the definition of succinct circuits. Most of this subsection is taken verbatim from [BGT14].

Definition 1 (Succinct Circuits). *Let $C : \{0,1\}^n \to \{0,1\}$ be a circuit with $N - n$ binary gates. The gates of the circuit are numbered as follows. The input gates are given the numbers $\{1, \ldots, n\}$. The intermediate gates are numbered $\{n+1, n+2, \ldots, N-1\}$ such that a gate that receives its input from gates i and j is given a number greater than i and j. The output gate is numbered N. Each gate $g \in [n+1, N]$ is* described by a tuple $(i, j, f_g) \in [g-1]^2 \times \mathsf{GType}$ *where outputs of gates i and j serves as inputs to gate g and f_g denotes the binary functionality computed by the gate. Here,* GType *denotes the set of all binary functions.*

We say that C is succinctly represented by a circuit C_{sc}, if C_{sc} given a gate label $g \in [n+1, N]$ gives out its description (i, j, f_g). Furthermore, $|C_{\mathsf{sc}}| < |C|$.

We now recall the lemma from [PF79] that converts any uniform Turing machine to a succinct circuit.

Lemma 1 ([PF79]). *Any Turing machine M, which for inputs of size n, requires a maximal running time $t(n)$ and space $s(n)$, can be converted in time $O(|M| + \log(t(n)))$ to a circuit C_{sc} that succinctly represents $C : \{0,1\}^n \to \{0,1\}$ where C computes the same function as M (for inputs of size n), and is of size $\widetilde{O}(t(n) \cdot s(n))$.*

2.2 Succinct Randomized Encoding

We now recall the definition of succinct randomized encoding.

Definition 2 ([BGT14]). *A succinct randomized encoding (SRE) consists of two algorithms* $(\mathsf{sRE.Enc}, \mathsf{sRE.Dec})$ *with the following syntax:*

- $\widehat{M}_{x,t} \leftarrow \mathsf{sRE.Enc}(1^\lambda, M, x, t)$: *takes as input the security parameter* λ*, a machine* M*, input* x*, time bound (encoded in binary)* t *and outputs the randomized encoding* $\widehat{M}_{x,t}$*.*
- $y \leftarrow \mathsf{sRE.Dec}(M, \widehat{M}_{x,t})$: *takes as input the machine* M *and the randomized encoding* $\widehat{M}_{x,t}$ *and deterministically computes the output* y*.*

We require the scheme to satisfy the following three properties.

- ***Correctness:*** *For every* x *and* M *such that* M *halts on input* x *within* t *steps, it holds that* $y = M(x)$ *with probability* 1 *over the random coins of* $\mathsf{sRE.Enc}$*.*
- ***Security:*** *there exists a PPT simulator* Sim *such that for any poly size adversary* $\mathcal{A}$ *there exists a negligible* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, machine* M*, input* x*, and time bound* t*:*

$$\left|\Pr[\mathcal{A}(\widehat{M}_{x,t}) = 1] - \Pr[\mathcal{A}(\mathsf{Sim}(1^\lambda, y, M, t, 1^{|x|})) = 1]\right| \leq \mathsf{negl}(\lambda) \cdot p(t)$$

  where $\widehat{M}_{x,t} \leftarrow \mathsf{sRE.Enc}(1^\lambda, M, x, t)$*,* y *is the output of* $M(x)$ *after* t *steps and* $p(\cdot)$ *is a fixed polynomial that does not depend on* (M, x, t)*.*[4]
- ***Succinctness:*** *The running time of* $\mathsf{sRE.Enc}$ *and the size of the encoding* $\widehat{M}_{x,t}$ *are* $\mathsf{poly}(|M|, |x|, \log t, \lambda)$*. The running time of* $\mathsf{sRE.Dec}$ *is* $\mathsf{poly}(t, \lambda)$*.*

Remark 1. We note that our definition of succinct randomized encoding differs from the original definition given in [BGT14] as the procedure $\mathsf{sRE.Dec}$ additionally takes in M as input. We note that this is without loss of generality as we can always set M to be the universal Turing machine and include the description of the machine that has to be encoded as part of the input.

2.3 Indistinguishability Obfuscation

We now define indistinguishability obfuscator from [BGI+12, GGH+13].

Definition 3. *A PPT algorithm* $i\mathcal{O}$ *is an indistinguishability obfuscator for a family of circuits* $\{C_\lambda\}_\lambda$ *that satisfies the following properties:*

- ***Correctness:*** *For all* λ *and for all* $C \in C_\lambda$ *and for all* x*,*

$$\Pr[i\mathcal{O}(C)(x) = C(x)] = 1$$

  where the probability is over the random choices of $i\mathcal{O}$*.*

[4] When t bounded by a polynomial then RHS can just be $\mathsf{negl}(\lambda)$.

– ***Security:*** *For all $C_0, C_1 \in C_\lambda$ such that for all x, $C_0(x) = C_1(x)$ and for all poly sized adversaries $\mathcal{A}$,*

$$|\Pr[\mathcal{A}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(C_1)) = 1]| \leq \mathsf{negl}(\lambda)$$

We now give the definition of a succinct indistinguishability obfuscation.

Definition 4 (Succinct Indistinguishability Obfuscator [BGL+15]). *A* succinct indistinguishability obfuscator *for a machine class $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of a uniform PPT machine $i\mathcal{O}\mathsf{M}$ that works as follows:*

– *$i\mathcal{O}\mathsf{M}$ takes as input the security parameter 1^λ, the machine M to obfuscate, and an input length n and time bound t for M.*
– *$i\mathcal{O}\mathsf{M}$ outputs a machine $\mathsf{ob}M$ which is an obfuscation of M corresponding to input length n and time bound t. $\mathsf{ob}M$ takes as input $x \in \{0,1\}^n$ and $t' \leq t$.*

The scheme should satisfy the following three requirements.

– ***Correctness****: For all security parameters $\lambda \in \mathbb{N}$, for all $M \in \mathcal{M}_\lambda$, for all inputs $x \in \{0,1\}^n$, time bounds t and $t' \leq t$, let y be the output of M on t' steps, then we have that:*

$$\Pr[\mathsf{ob}M(x, t') = y : \mathsf{ob}M \leftarrow i\mathcal{O}\mathsf{M}(1^\lambda, 1^n, 1^{\log t}, M)] = 1$$

– ***Security****: For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, time bounds t, and pairs of machines $M_0, M_1 \in \mathcal{M}_\lambda$ of the same size such that for all running times $t' \leq t$ and for all inputs x, $M_0(x) = M_1(x)$ when M_0 and M_1 are executed for time t', we have that:*

$$\left|\Pr\left[D(i\mathcal{O}\mathsf{M}(1^\lambda, 1^n, 1^{\log t}, M_0)) = 1\right] - \Pr\left[D(i\mathcal{O}\mathsf{M}(1^\lambda, 1^n, 1^{\log t}, M_1)) = 1\right]\right| \leq \alpha(\lambda)$$

– ***Efficiency and Succinctness****: We require that the running time of $i\mathcal{O}\mathsf{M}$ and the length of its output, namely the obfuscated machine $\mathsf{ob}M$, is $\mathsf{poly}(|M|, \log t, n, \lambda)$. We also require that the obfuscated machine on input x and t' runs in time $\mathsf{poly}(|M|, t', n, \log t, \lambda)$ (or $\mathsf{poly}(t', \lambda)$ for short).*

2.4 Garbled Circuits

Below we recall the definition of garbling scheme for circuits [Yao82, Yao86, AIK04] with selective security (see Lindell and Pinkas [LP09] and Bellare et al. [BHR12] for a detailed proof and further discussion). A garbling scheme for circuits is a tuple of PPT algorithms $(\mathsf{GarbleCkt}, \mathsf{EvalCkt})$. Very roughly, $\mathsf{GarbleCkt}$ is the circuit garbling procedure and $\mathsf{EvalCkt}$ is the corresponding evaluation procedure. We use a formulation where input labels for a garbled circuit are provided as input to the garbling procedure rather than generated as output. (This simplifies the presentation of our construction.) More formally:

- $\widetilde{\mathsf{C}} \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C, \{\mathsf{lab}_{w,b}\}_{w\in x, b\in\{0,1\}}\right)$: $\mathsf{GarbleCkt}$ takes as input a security parameter λ, a circuit C, and input labels $\mathsf{lab}_{w,b}$ where $w \in x$ (x is the set of input wires to the circuit C) and $b \in \{0,1\}$. This procedure outputs a *garbled circuit* $\widetilde{\mathsf{C}}$. We assume that for each w, b, $\mathsf{lab}_{w,b}$ is chosen uniformly from $\{0,1\}^\lambda$.
- $y \leftarrow \mathsf{EvalCkt}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in x}\right)$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a sequence of input labels $\{\mathsf{lab}_{w,x_w}\}_{w\in x}$ (referred to as the garbled input), $\mathsf{EvalCkt}$ outputs a string y.

Correctness. For correctness, we require that for any circuit C, input $x \in \{0,1\}^{|x|}$ and input labels $\{\mathsf{lab}_{w,b}\}_{w\in x, b\in\{0,1\}}$ we have that:

$$\Pr\left[C(x) = \mathsf{EvalCkt}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in x}\right)\right] = 1$$

where $\widetilde{\mathsf{C}} \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C, \{\mathsf{lab}_{w,b}\}_{w\in x, b\in\{0,1\}}\right)$.

Selective Security. For security, we require that there exists a PPT simulator $\mathsf{Sim}_{\mathsf{Ckt}}$ such that for any circuit C and input $x \in \{0,1\}^{|x|}$, we have that

$$\left\{\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in x}\right\} \overset{c}{\approx} \left\{\mathsf{Sim}_{\mathsf{Ckt}}\left(1^\lambda, 1^{|C|}, C(x), \{\mathsf{lab}_{w,x_w}\}_{w\in x}\right), \{\mathsf{lab}_{w,x_w}\}_{w\in x}\right\}$$

where $\widetilde{\mathsf{C}} \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C, \{\mathsf{lab}_{w,b}\}_{w\in x, b\in\{0,1\}}\right)$ and for each $w \in x$ and $b \in \{0,1\}$ we have $\mathsf{lab}_{w,b} \leftarrow \{0,1\}^\lambda$. Here $\overset{c}{\approx}$ denotes that the two distributions are computationally indistinguishable.

Theorem 1 ([Yao86, LP09]). *Assuming the existence of one-way functions, there exists a construction of garbling scheme for circuits.*

2.5 Updatable Laconic Oblivious Transfer

In this subsection, we recall the definition of updatable laconic oblivious transfer from [CDG+17].

Definition 5 ([CDG+17]). *An updatable laconic oblivious transfer consists of the following algorithms:*

- $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$: *It takes as input the security parameter 1^λ (encoded in unary) and outputs a common reference string* crs.
- $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$: *It takes as input the common reference string* crs *and database $D \in \{0,1\}^*$ as input and outputs a digest* d *and a state $\widehat{D}$. We assume that the state $\widehat{D}$ also includes the database D.*
- $\mathsf{d}^* \leftarrow \mathsf{HashUpdate}(\mathsf{crs}, \mathsf{d}, (L, b), \mathsf{aux})$: *It takes as input the common reference string* crs*, a digest* d*, position $L \in N$, a bit b and some auxiliary information of size* $\mathsf{poly}(\log|D|, \lambda)$ *and outputs* d^*.

- $e \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{d}, L, m_0, m_1)$: *It takes as input the common reference string* crs*, a digest* d*, a location* $L \in \mathbb{N}$ *and two messages* $m_0, m_1 \in \{0,1\}^{p(\lambda)}$ *and outputs a ciphertext* e*.*
- $m \leftarrow \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, e, L)$: *This is a RAM algorithm with random read access to* $\widehat{D}$*. It takes as input a common reference string* crs*, a ciphertext* e*, and a database location* $L \in \mathbb{N}$ *and outputs a message* m*.*
- $e_w \leftarrow \mathsf{SendWrite}(\mathsf{crs}, \mathsf{d}, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{|\mathsf{d}|})$: *It takes as input the common reference string* crs*, a digest* d*, a location* $L \in \mathbb{N}$*, a bit* $b \in \{0,1\}$ *to be written, and* $|\mathsf{d}|$ *pairs of messages* $\{m_{j,0}, m_{j,1}\}_{j=1}^{|\mathsf{d}|}$*, where each* $m_{j,c}$ *is of length* $p(\lambda)$ *and outputs a ciphertext* e_w*.*
- $\{m_j\}_{j=1}^{|\mathsf{d}|} \leftarrow \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, L, b, e_w)$: *This is a RAM algorithm with random read/write access to* $\widehat{D}$*. It takes as input the common reference string* crs*, a location* L*, a bit* $b \in \{0,1\}$ *and a ciphertext* e_w*. It updates the state* $\widehat{D}$ *(such that* $D[L] = b$*) and outputs messages* $\{m_j\}_{j=1}^{|\mathsf{d}|}$*.*

We require an updatable laconic oblivious transfer to satisfy the following properties.

Correctness: *We require that for any database* D *of size at most* $M = \mathsf{poly}(\lambda)$*, any memory location* $L \in [M]$*, any pair of messages* $(m_0, m_1) \in \{0,1\}^{p(\lambda)}$ *where* $p(\cdot)$ *is a polynomial that*

$$\Pr\left[m = m_{D[L]} \;\middle|\; \begin{array}{rl} \mathsf{crs} & \leftarrow \mathsf{crsGen}(1^\lambda) \\ (\mathsf{d}, \widehat{D}) & \leftarrow \mathsf{Hash}(\mathsf{crs}, D) \\ e & \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{d}, L, m_0, m_1) \\ m & \leftarrow \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, e, L) \end{array} \right] = 1,$$

Correctness of Hash updates: *We require that for any database* D *of size* $M = \mathsf{poly}(\lambda)$*, any memory location* $L \in [M]$*, any bit* $b \in \{0,1\}$*, we require* $\mathsf{HashUpdate}(\mathsf{crs}, \mathsf{d}, (L, i), \mathsf{aux})$ *to be same as* $\mathsf{Hash}(\mathsf{crs}, D^*)$ *where* D^* *is same as* D *except that* $D^*[L] = b$*. Here,* aux *corresponds to an auxiliary information that is specific to position* L*.*

Correctness of Writes: *Let database* D *be of size at most* $M = \mathsf{poly}(\lambda)$ *and let* $L \in [M]$ *be any memory location. Let* D^* *be a database that is identical to* D *except that* $D^*[L] = b$*. For any sequence of messages* $\{m_{j,0}, m_{j,1}\}_{j \in [\lambda]} \in \{0,1\}^{p(\lambda)}$ *we require that*

$$\Pr\left[\begin{array}{l} m'_j = m_{j,\mathsf{d}^*_j} \\ \forall j \in [|\mathsf{d}|] \end{array} \;\middle|\; \begin{array}{rl} \mathsf{crs} & \leftarrow \mathsf{crsGen}(1^\lambda) \\ (\mathsf{d}, \widehat{D}) & \leftarrow \mathsf{Hash}(\mathsf{crs}, D) \\ (\mathsf{d}^*, \widehat{D}^*) & \leftarrow \mathsf{Hash}(\mathsf{crs}, D^*) \\ e_w & \leftarrow \mathsf{SendWrite}\left(\mathsf{crs}, \mathsf{d}, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{|\mathsf{d}|}\right) \\ \{m'_j\}_{j=1}^{|\mathsf{d}|} & \leftarrow \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, L, b, e_w) \end{array} \right] = 1,$$

Sender Privacy: *There exists a PPT simulator* $\mathsf{Sim}_{\ell\mathsf{OT}}$ *such that the for any non-uniform PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function*

$\mathsf{negl}(\cdot)$ *s.t.,*

$$\left|\Pr[\mathsf{SenPrivExpt}^{\mathsf{real}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{SenPrivExpt}^{\mathsf{ideal}}(1^\lambda, \mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{SenPrivExpt}^{\mathsf{real}}$ *and* $\mathsf{SenPrivExpt}^{\mathsf{ideal}}$ *are described in Figure 1.*

$\mathsf{SenPrivExpt}^{\mathsf{real}}[1^\lambda, \mathcal{A}]$	$\mathsf{SenPrivExpt}^{\mathsf{ideal}}[1^\lambda, \mathcal{A}]$
1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$. 2. $(D, L, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$. 3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$. 4. Output $\mathcal{A}_2(\mathsf{st}, \mathsf{Send}(\mathsf{crs}, \mathsf{d}, L, m_0, m_1))$.	1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$. 2. $(D, L, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$. 3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$. 4. Output $\mathcal{A}_2(\mathsf{st}, \mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D, L, m_{D[L]}))$.

Figure 1: Sender Privacy Security Game

Sender Privacy for Writes: *There exists a PPT simulator* $\mathsf{Sim}_{\ell\mathsf{OTW}}$ *such that the for any non-uniform PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.,*

$$\left|\Pr[\mathsf{WriSenPrivExpt}^{\mathsf{real}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{WriSenPrivExpt}^{\mathsf{ideal}}(1^\lambda, \mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{WriSenPrivExpt}^{\mathsf{real}}$ *and* $\mathsf{WriSenPrivExpt}^{\mathsf{ideal}}$ *are described in Figure 2.*

$\mathsf{WriSenPrivExpt}^{\mathsf{real}}[1^\lambda, \mathcal{A}]$	$\mathsf{WriSenPrivExpt}^{\mathsf{ideal}}[1^\lambda, \mathcal{A}]$
1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$. 2. $(D, L, b, \{m_{j,0}, m_{j,1}\}_{j\in[\lambda]}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$. 3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$. 4. $e_w \leftarrow \mathsf{SendWrite}(\mathsf{crs}, \mathsf{d}, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{\lvert\mathsf{d}\rvert})$ 5. Output $\mathcal{A}_2(\mathsf{st}, e_w)$.	1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$. 2. $(D, L, b, \{m_{j,0}, m_{j,1}\}_{j\in[\lambda]}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$. 3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$. 4. $(\mathsf{d}^*, \widehat{D}^*) \leftarrow \mathsf{Hash}(\mathsf{crs}, D^*)$ where D^* be a database that is identical to D except that $D^*[L] = b$. 5. $e_w \leftarrow \mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D, L, b, \{m_{j,\mathsf{d}_j^*}\}_{j\in[\lambda]})$ 6. Output $\mathcal{A}_2(\mathsf{st}, e_w)$.

Figure 2: Sender Privacy for Writes Security Game

Efficiency: *The algorithm Hash runs in time* $|D|\mathsf{poly}(\log|D|, \lambda)$*. The algorithms* HashUpdate*,* Send*,* SendWrite*,* Receive*,* ReceiveWrite *run in time* $\mathsf{poly}(\log|D|, \lambda)$.

Theorem 2 ([CDG$^+$17]). *Assuming $i\mathcal{O}$ for circuits and* somewhere statistically binding hash functions*, there exists a construction of updatable laconic oblivious transfer.*

Remark 2. We note that the security requirements given in Definition 5 is stronger than the one in [CDG$^+$17] as we require the crs to be generated before the adversary provides the database D and the location L. However, the constructions given in [CDG$^+$17] already satisfies this stronger definition and this was noted in [GS18a].

A Note on Hash Updates. The construction of updatable Laconic Oblivious Transfer given in [CDG$^+$17] uses a Merkle Hash to hash the database. Thus, to compute the hash we need the contents of the entire database to be specified. But in our construction of succinct randomized encodings, we need a methodology to compute the Merkle tree "on the fly." More specifically, let us consider a scenario wherein we are not initially specified the entire database $D \in \{0,1\}^M$ but are only given the contents of the first n locations. We give a methodology to compute the Merkle hash which "binds" the first n locations, keeps the other locations to be unspecified and runs in time $\mathsf{poly}(n, \lambda, \log M)$. A similar trick has been used in [OPWW15].

Let us assume that we are given a hash function $H : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$. To store a database of size M, the Merkle tree consists of M leaves where each leaf stores a λ bit string which either corresponds to the bit 0, or the bit 1 or a special symbol $\perp$ (using some canonical encoding). We construct the Merkle tree in a bottom-up fashion by labeling all the internal nodes. The label of the root node gives the hash value. We label each internal node of the Merkle tree with children given labels lab_ℓ and lab_r as follows:

- If both lab_ℓ and lab_r are given labels $\perp$, then node is given $\perp$ as its label.
- Otherwise, the node is given $H(\mathsf{lab}_\ell \| \mathsf{lab}_r)$ as the label where $\|$ denotes concatenation.

Note that if all the locations are unspecified then the label of the root corresponds to $\perp$. For each additional location L that is specified, we just fix the auxiliary information aux to be labels of the all the nodes in the root to the leaf given by L along with their siblings. Note we only need to maintain the state of all labels which are not equal $\perp$ when performing an hash update. Given this information, we can easily recompute the label of the root. This gives the required methodology to update the hash value in time $\mathsf{poly}(n, \lambda, \log M)$ where n is the number of specified locations.

2.6 Puncturable Pseudorandom Function

We recall the notion of puncturable pseudorandom function from [SW14]. The construction of pseudorandom function given in [GGM86] satisfies the following definition [BW13, KPTZ13, BGI14].

Definition 6. *A puncturable pseudorandom function* PPRF *is a tuple of PPT algorithms* ($\mathsf{KeyGen}_{\mathsf{PPRF}}$, $\mathsf{PRF}, \mathsf{Punc}$) *with the following properties:*

- ***Efficiently Computable:*** *For all* λ *and for all* $S \leftarrow \mathsf{KeyGen}_{\mathsf{PPRF}}(1^\lambda)$, $\mathsf{PRF}_S : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ *is polynomial time computable.*
- ***Functionality is preserved under puncturing:*** *For all* λ, *for all* $y \in \{0,1\}^\lambda$ *and* $\forall x \neq y$,

$$\Pr[\mathsf{PRF}_{S\{y\}}(x) = \mathsf{PRF}_S(x)] = 1$$

  where $S \leftarrow \mathsf{KeyGen}_{\mathsf{PPRF}}(1^\lambda)$ *and* $S\{y\} \leftarrow \mathsf{Punc}(S, y)$.
- ***Pseudorandomness at punctured points:*** *For all* λ, *for all* $y \in \{0,1\}^\lambda$, *and for all poly sized adversaries* $\mathcal{A}$

$$|\Pr[\mathcal{A}(\mathsf{PRF}_S(y), S\{y\}) = 1] - \Pr[\mathcal{A}(U_\lambda, S\{y\}) = 1]| \leq \mathsf{negl}(\lambda)$$

  where $S \leftarrow \mathsf{KeyGen}_{\mathsf{PPRF}}(1^\lambda)$, $S\{y\} \leftarrow \mathsf{Punc}(S, y)$ *and* U_λ *denotes the uniform distribution over* $\{0,1\}^\lambda$.

Remark 3. We can generalize the puncturing procedure to puncture at multiple points $y_1, \ldots, y_m$. The security requirement now is that even given the punctured key $S\{y_1, \ldots, y_m\}$, the PRF evaluations on inputs $y_1, \ldots, y_m$ are computationally indistinguishable to random. We note that in the case of multiple puncturings, the size of the punctured key $S\{y_1, \ldots, y_m\}$ grows polynomially in m and λ.

3 Construction of Succinct Randomized Encoding

In this section, we give a construction of succinct randomized encoding for succinctly describable Turing machines. More formally, we show that:

Theorem 3. *Assuming the existence of indistinguishability obfuscation and updatable laconic oblivious transfer, there exists a construction of succinct randomized encoding.*

As shown in [BGL+15], a succinct randomized encoding with sub-exponential security gives a construction of succinct $i\mathcal{O}$ for Turing machines. For completeness, we sketch the details of this transformation in the full version of our paper [GS18b]. We give the formal description of our construction of succinct randomized encodings in Figure 3 and give an overview below.

Overview. Let us start with an overview of the encoding scheme. The encoding procedure takes as input a description of the Turing machine M and an input x on which the machine has to be evaluated. The procedure first reduces M to a circuit C_{sc} (as given in Lemma 1) that succinctly represents the circuit C which computes the same function as that of M. Let C consist of $N - n$ binary

gates with N being the output gate. Each gate $g \in [n+1, N]$ is described by a tuple $(i, j, f_g) \in [g-1]^2 \times \mathsf{GType}$ where outputs of gates i and j serves as inputs to gate g and f_g is the binary function computed by gate g. Given an input $g \in [n+1, N]$, the succinct circuit C_{sc} outputs (i, j, f_g).

For our construction, we consider an alternate view of the circuit C. We view the circuit C as a sequence of step circuits $\mathsf{SC}_{n+1}, \ldots, \mathsf{SC}_N$ along with a database D. The database is initially loaded with the input x and each step circuit writes a single bit to the database. More precisely, for each $g \in [n+1, N]$, the step circuit SC_g implements the functionality of the gate g and writes the output of that gate to position g in the database. Further, the step circuits access the database via an updatable laconic OT. Specifically, the step circuit SC_g takes as input the digest of the database where the first $g-1$ cells are filled appropriately and the rest of the positions being $\perp$. Using the digest, it reads the contents of the database in positions i and j (where (i, j) are the inputs to gate g) using the Send function of laconic OT. Once it has read the contents of those two locations, it applies the function f_g on those two bits and writes the output to the location g using the $\mathsf{SendWrite}$ function. It passes on the updated digest to the next circuit SC_{g+1}. Thus, each of the step circuits faithfully model the computation of the corresponding gate and the contents in location N of the database gives the output of the circuit C.

Let us now explain how the encoding procedure uses the above view of the circuit. The encoding procedure obfuscates the function Gate (formally described in Figure 4). The function Gate on input $g \in [n+1, N]$, uses the succinct circuit C_{sc} to get the description of gate g. Next, it constructs the step circuit SC_g (formally described in Figure 5) and garbles the circuit (the randomness and the labels are derived using a puncturable pseudorandom function). The Gate function finally outputs the garbled step circuit $\widetilde{\mathsf{SC}}_g$. The output of the encoding function is this obfuscation along with the labels corresponding to the initial digest of the database (where the input is loaded).

Given an obfuscation of the function Gate, a decoder can run this obfuscation on every gate $g \in [n+1, N]$ to obtain the garbled step circuit $\widetilde{\mathsf{SC}}_g$. Given the labels corresponding to the initial digest, the decoder evaluates each of the garbled step circuits from $n+1$ to N (labels corresponding to the g^{th} step circuit are output by the $(g-1)^{th}$ circuit). At the end of the computation, the content of the database at location N gives the output.

However, there is one technical issue. Recall that the laconic OT is not guaranteed to hide the contents of the database. In order to hide the contents of the database, we use a one-time pad to mask each bit that is written. This one time pad is succinctly derived using a puncturable pseudorandom function.

Correctness. This argument is based on the correctness proof in [GS18a]. Let D_{g^*} be the contents of the database at the beginning of g^*-th iteration of the **for** loop in $\mathsf{sRE.Dec}$. We first argue via an inductive argument that for each gate $g^* \in [1, N]$, $D_{g^*+1,g}$ is the output of gate g masked with r_g for every $g \in [1, g^*]$.

$\mathsf{sRE.Enc}(1^\lambda, M, x, t)$**:** On input a Turing machine M, an input $x \in \{0,1\}^n$ and a time bound t do:

1. Reduce M to a succinct circuit C_{sc} from Lemma 1 that describes the circuit $C : \{0,1\}^n \to \{0,1\}$ computing the same function as that of M. Let $N - n$ be the number of binary gates in C.
2. Sample $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$ and three PRF keys $S, R, K \leftarrow \mathsf{KeyGen}_{\mathsf{PPRF}}(1^\lambda)$. We will truncate the output length of $\mathsf{PRF}_R(\cdot)$ to one bit.
3. For each $k \in [\lambda]$ and $b \in \{0,1\}$, compute $\mathsf{lab}^1_{k,b} = \mathsf{PRF}_K((1,k,b))$.
4. Compute $i\mathcal{O}(\mathsf{pad}_\ell(\mathsf{Gate}[C_{\mathsf{sc}}, \mathsf{crs}, S, R, K]))$ where the circuit Gate is described in Figure 4 and $\mathsf{pad}_\ell(\cdot)$ pads the circuit to size ℓ which will be specified in the proof.
5. For each $i \in [n]$, set $y_i = x_i \oplus \mathsf{PRF}_R(i)$.
6. Set $\mathsf{d} = \perp$ and for each $i \in [n]$,
   (a) Recompute $\mathsf{d} = \mathsf{HashUpdate}(\mathsf{crs}, \mathsf{d}, (i, y_i), \mathsf{aux})$ where aux is the auxiliary information for updating position i.
7. Compute $r_N = \mathsf{PRF}_R(N)$.
8. Output $\big(i\mathcal{O}(\mathsf{pad}_\ell(\mathsf{Gate}[C_{\mathsf{sc}}, \mathsf{crs}, S, R, K])), \{\mathsf{lab}^1_{k,\mathsf{d}_k}\}_{k\in[\lambda]}, \{y_i\}_{i\in[n]}, r_N\big)$.

$\mathsf{sRE.Dec}(M, \widehat{M}_{x,t})$: On input the machine M and the randomized encoding $\widehat{M}_{x,t}$ do:

1. Initialize the Merkle tree $\widehat{D}$ with the leaf node i storing bit y_i for every $i \in [n]$. Initialize all other leaves with special symbol $\perp$.
2. For each $g \in [n+1, N]$ do:
   (a) $\widetilde{\mathsf{SC}}_g := i\mathcal{O}(\mathsf{pad}_\ell(\mathsf{Gate}[C_{\mathsf{sc}}, \mathsf{crs}, S, R, K]))(g)$.
3. Set $\overline{\mathsf{lab}} = \{\mathsf{lab}^1_{k,\mathsf{d}_k}\}_{k\in[\lambda]}$.
4. **for** each g from $n+1$ to N **do**:
   (a) Let (i, j, f_g) be the description of gate g.
   (b) Compute $(\gamma, e) := \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{EvalCkt}(\widetilde{\mathsf{SC}}_g, \overline{\mathsf{lab}}), i), j)$.
   (c) Set $\overline{\mathsf{lab}} := \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, g, \gamma, e)$.
5. Recover the contents of the leaves D from the final state $\widehat{D}$.
6. Output $D_N \oplus r_N$.

Figure 3: Succinct Randomized Encoding

Given this, the correctness follows by setting $g^* := N$ and observing that the $D_{N+1,N}$ is unmasked using r_N in Step 7 of $\mathsf{sRE.Dec}$.

The base case is $g^* = n$ which is clearly true since in the beginning D_{n+1} is set as $(r_{[1,n]} \oplus x || \perp^{N-n})$. In order to prove the inductive step for a gate g^* (with description (i, j, f_{g^*})), we now argue that that the γ recovered in Step 4.(b) of $\mathsf{sRE.Dec}$ corresponds to $f_{g^*}(D_{g^*,i} \oplus r_i D_{g^*,j} \oplus r_j) \oplus r_{g^*}$ which by inductive hypothesis corresponds to output of the gate g^* masked with r_{g^*}. This is shown as follows.

Gate

Input: A gate $g \in [n+1, N]$.
Hardcoded: The circuit C_{sc}, common reference string crs, a triplet of PRF keys (S, R, K).

1. Run C_{sc} on input g to obtain (i, j, f_g).
2. Set $r_i = \mathsf{PRF}_R(i)$, $r_j = \mathsf{PRF}_R(j)$ and $r_g = \mathsf{PRF}_R(g)$.
3. Compute $\mathsf{lab}^g_{k,b} = \mathsf{PRF}_K(g, k, b)$ and $\mathsf{lab}^{g+1}_{k,b} = \mathsf{PRF}_K(g{+}1, k, b)$ for each $k \in [\lambda]$ and $b \in \{0,1\}$. (We use $\{\mathsf{lab}^g_{k,b}\}$ to denote $\{\mathsf{lab}^g_{k,b}\}_{k\in[\lambda], b\in\{0,1\}}$.)
4. Compute (where the step-circuit SC is described in Figure 5)

$$\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, \mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i, j), f_g, \{\mathsf{lab}^{g+1}_{k,b}\}, 0], \{\mathsf{lab}^g_{k,b}\}; \mathsf{PRF}_S(g)\right).$$

5. Output $\widetilde{\mathsf{SC}}_g$.

Figure 4: Description of Gate

Step Circuit SC

Input: A digest d.
Hardcoded: The common reference string crs, a triplet of masking bits (r_i, r_j, r_g), a description (i, j) of gate g, a binary function $f_g : \{0,1\}^2 \to \{0,1\}$, a set of labels $\{\mathsf{lab}_{k,b}\}$ and a bit τ ($\tau = 1$ case is only relevant for the proof).

1. Compute $e_b \leftarrow \mathsf{SendWrite}(\mathsf{crs}, \mathsf{d}, g, b, \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[\lambda]})$ for $b \in \{0,1\}$.
2. Define for all $\alpha, \beta \in \{0,1\}$, $\gamma(\alpha,\beta) := \begin{cases} f_g(\alpha \oplus r_i, \beta \oplus r_j) \oplus r_g & \text{if } \tau = 0 \\ r_g & \text{if } \tau = 1 \end{cases}$
3. Generate

$$c_0 \leftarrow \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, (\gamma(0,0), e_{\gamma(0,0)}), (\gamma(0,1), e_{\gamma(0,1)})\right),$$

$$c_1 \leftarrow \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, (\gamma(1,0), e_{\gamma(1,0)}), (\gamma(1,1), e_{\gamma(1,1)})\right).$$

4. Output $\mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, i, c_0, c_1\right)$

Figure 5: Description of the Step Circuit

$$\begin{aligned}
(\gamma, e) &:= \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{EvalCkt}(\widetilde{\mathsf{SC}}_g, \overline{\mathsf{lab}}), i), j) \\
&= \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, i, c_0, c_1\right), i), j) \\
&= \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, c_{D_{g^*,i}}, j) \\
&= \mathsf{Receive}^{\widehat{D}}\left(\mathsf{crs}, \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, (\gamma(D_{g^*,i}, 0), e_{\gamma(D_{g^*,i},0)}), (\gamma(D_{g^*,i}, 1), e_{\gamma(D_{g^*,i},1)})\right), j\right) \\
&= \left(\gamma(D_{g^*,i}, D_{g^*,j}), e_{\gamma(D_{g^*,i}, D_{g^*,j})}\right) \\
&= \left(f_{g^*}(D_{g^*,i} \oplus r_i D_{g^*,j} \oplus r_j) \oplus r_{g^*}, e_{f_{g^*} D_{g^*,i} \oplus r_i D_{g^*,j} \oplus r_j \oplus r_{g^*}}\right)
\end{aligned}$$

4 Security Proof

In this section, we prove that the construction presented in the Section 3 satisfies security property given in Definition 2. In Subsection 4.1, we start by defining circuit configurations. Next, in Subsection 4.2 we show that both the real world garbling procedure and the simulated distributions are special cases of this circuit configuration. Finally, in the rest of the subsection we show that the real garbling and the simulated distributions are indistinguishable.

4.1 Circuit Configuration

Our proof of security proceeds via a hybrid argument over different *circuit configurations* which we describe in this section. A circuit configuration denoted by $\mathsf{conf} = (I, i)$ consists of a set $I \subseteq [n+1, N]$ and an index $i \in [n+1, N]$. Intuitively, each circuit configuration defines a distribution of the randomized encoding $\widetilde{M}^{\mathsf{conf}}_{x,t}$. Let us now explain the semantics of the set I and the index i.

Recall that from our construction described in Figure 3, $i\mathcal{O}(\mathsf{pad}_\ell(\mathsf{Gate}))$ outputs $\widetilde{\mathsf{SC}}_g$ when given a gate $g \in [n+1, N]$ as input. Intuitively, a configuration of a circuit defines a particular distribution of $\widetilde{\mathsf{SC}}_g$ for each $g \in [n+1, N]$. In particular, for each gate g, the distribution of $\widetilde{\mathsf{SC}}_g$ can be in one of the three modes: White mode, Gray mode and the Black mode. We say that $\widetilde{\mathsf{SC}}_g$ is said to be in White mode if for the distribution of $\widetilde{\mathsf{SC}}_g$ is same as the honest garbling procedure given in Figure 4. We say that $\widetilde{\mathsf{SC}}_g$ is in Gray mode if its distribution depends only on the output of the gate g when the circuit C is evaluated with input x. We say that $\widetilde{\mathsf{SC}}_g$ is in Black mode if its distribution is independent of the input x. Looking ahead, initially all the step circuits will be in White mode and the goal will be to convert all of them to Black in the simulation. We will achieve this in the reverse order i.e., we first change SC_N to Black mode and then change SC_{N-1} and so on. The index i (given as part of defining the circuit configuration) is such that for all $g > i$ the distribution of the garbled step circuit $\widetilde{\mathsf{SC}}_g$ is in Black mode. We can also extend the notion of Black mode to input gates $[1, n]$. So i can be any element in the set $[0, N]$. The subset I indicates the set of gates g such that the distribution of the garbled step circuit $\widetilde{\mathsf{SC}}_g$ is in Gray mode. The rest of the garbled step circuits $\widetilde{\mathsf{SC}}_g$ where $g \notin I$ and $g \leq i$ are generated in White mode. We say a configuration is valid if $I \cap [i+1, N] = \emptyset$.

Simulation in a valid configuration. In Figure 6, we describe the simulated encoding procedure $\mathsf{SimsRE.Enc}$ for any given configuration conf. Note that these simulated encoding function also takes x as input whereas the ideal world simulation does not. We describe our simulator functions with these additional inputs so that it captures simulation in all of our intermediate hybrids. We note that final ideal world simulation does not use these values.

$\mathsf{SimsRE.Enc}(1^\lambda, M, x, t)$**:** On input a Turing machine M, an input $x \in \{0,1\}^n$ and a time bound t do:

1. Reduce M to a succinct circuit C_{sc} from Lemma 1 that describes the circuit $C : \{0,1\}^n \to \{0,1\}$. Let $N - n$ be the number of binary gates in C.
2. Sample $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$ and three PRF keys $S, R, K \leftarrow \mathsf{KeyGen}_{\mathsf{PPRF}}(1^\lambda)$. We will truncate the output length of $\mathsf{PRF}_R(\cdot)$ to one bit.
3. **Notation:** For $g \in [n+1, N+1]$, we let D_g be such that

$$D_{g,w} = \begin{cases} x_w \oplus \mathsf{PRF}_R(w) & w \le n, \\ E_w \oplus \mathsf{PRF}_R(w) & n+1 \le w < g, \\ \bot & \text{otherwise}, \end{cases}$$

   where E_w is the bit assigned to wire w of the circuit C computed on input x. Finally, we let d_g be the digest of D_g (i.e., $(\mathsf{d}_g, \cdot) := \mathsf{Hash}(\mathsf{crs}, D_g)$) and $\mathsf{d}_{g,k}$ be the k^{th} bit of d_g.
4. For each $k \in [\lambda]$ and $b \in \{0,1\}$, compute $\mathsf{lab}^1_{k,b} = \mathsf{PRF}_K((1,k,b))$.
5. **for** each g from N down to $n+1$ such that $g \in I$:
   (a) Set $e \leftarrow \mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D_g, g, D_{g+1,g}, \{\mathsf{lab}^{g+1}_{k,\mathsf{d}_{g+1,k}}\}_{k\in[\lambda]})$.
   (b) Set $\mathsf{out}_g \leftarrow \mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D_g, i, \mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D_g, j, e))$
6. Compute $i\mathcal{O}(\mathsf{pad}_\ell(\mathsf{SimGate}[C_{\mathsf{sc}}, \mathsf{crs}, S, R, K, (I,i), \{\mathsf{out}_g, \mathsf{d}_g\}_{g\in I}]))$ where the circuit $\mathsf{SimGate}$ is described in Figure 7 and $\mathsf{pad}_\ell(\cdot)$ pads the circuit to size ℓ which will be specified later.
7. For each $w \in [n]$, set $y_w = \mathsf{PRF}_R(w)$ if $w > i$ and $y_w = x_w \oplus \mathsf{PRF}_R(w)$ otherwise.
8. Set $\mathsf{d} = \bot$ and for each $w \in [n]$,
   (a) Recompute $\mathsf{d} = \mathsf{HashUpdate}(\mathsf{d}, \mathsf{aux}, w, y_w)$ where aux is the auxiliary information for updating position w.
9. If $i < N$ then compute $r'_N = \mathsf{PRF}_R(N) \oplus M(x)$; else, compute $r'_N = \mathsf{PRF}_R(N)$.
10. Output $\big(i\mathcal{O}(\mathsf{pad}_\ell(\mathsf{Gate}[C_{\mathsf{sc}}, S, R, K])), \{\mathsf{lab}_{k,\mathsf{d}_k}\}_{k\in[\lambda]}, \{y_i\}_{i\in[n]}, r'_N\big)$.

Figure 6: Succinct Randomized Encoding in configuration $\mathsf{conf} = (I, i)$.

4.2 Our Hybrids

For every circuit configuration $\mathsf{conf} = (I, i)$, we define $\mathsf{Hybrid}_{\mathsf{conf}}$ to be a distribution of $\widehat{M}_{x,t}$ as given in Figure 6. We start by observing that both real world and ideal distribution from Definition 2 can be seen as instance of $\mathsf{Hybrid}_{\mathsf{conf}}$ where $\mathsf{conf} = (\emptyset, N)$ and $\mathsf{conf} = (\emptyset, 0)$, respectively. In other words, the real world distribution corresponds to having all gates in White mode and the ideal world distribution corresponds to having all gates in Black mode. The goal is to move from the real world distribution to the ideal world distribution while

SimGate

Input: A gate $g \in [n+1, N]$.
Hardcoded: The circuit C_{sc}, common reference string crs, a triplet of PRF keys (S, R, K), the configuration (I, i), $\{\mathsf{out}_g\}_{g\in I}$ and $\{\mathsf{d}_g\}_{g\in I}$.

1. Run C_{sc} on input g to obtain $f_g, (i, j)$.
2. Set $r_i = \mathsf{PRF}_R(i)$, $r_j = \mathsf{PRF}_R(j)$ and $r_g = \mathsf{PRF}_R(g)$.
3. Compute $\mathsf{lab}^g_{k,b} = \mathsf{PRF}_K(g, k, b)$ and $\mathsf{lab}^{g+1}_{k,b} = \mathsf{PRF}_K(g+1, k, b)$ for each $k \in [\lambda]$ and $b \in \{0, 1\}$. (We use $\{\mathsf{lab}^g_{k,b}\}$ to denote $\{\mathsf{lab}^g_{k,b}\}_{k\in[\lambda],b\in\{0,1\}}$.)
4. If $g \leq i$ and $g \notin I$ then compute (where the step-circuit SC is described in Figure 5)

$$\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, \mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i, j), f_g, \{\mathsf{lab}^{g+1}_{k,b}\}, 0], \{\mathsf{lab}^g_{k,b}\}; \mathsf{PRF}_S(g)\right).$$

5. Else if $g > i$, compute

$$\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, \mathsf{SC}[\mathsf{crs}, (0, 0, r_g), (i, j), \{\mathsf{lab}^{g+1}_{k,b}\}, 1], \{\mathsf{lab}^g_{k,b}\}; \mathsf{PRF}_S(g)\right).$$

6. Else, compute $\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{Sim}_{\mathsf{Ckt}}\left(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}_g, \{\mathsf{lab}^g_{k,\mathsf{d}_{g,k}}\}_{k\in[\lambda]}; \mathsf{PRF}_S(g)\right)$.
7. Output $\widetilde{\mathsf{SC}}_g$.

Figure 7: Description of SimGate

minimizing the maximum number of gates in the Gray mode in any intermediate hybrid.[5]

4.2.1 Rules of Indistinguishability We will now describe the two rules (we call these rule A and rule B) to move from one valid circuit configuration conf to another valid configuration conf' such that $\mathsf{Hybrid}_{\mathsf{conf}}$ is computationally indistinguishable from $\mathsf{Hybrid}_{\mathsf{conf}'}$.

Rule A: Rule A says that for any valid configuration conf we can indistinguishably change gate g^* in White mode to Gray mode if it is the first gate or if its predecessor is also in Gray mode. More formally, let $\mathsf{conf} = (I, i)$ and $\mathsf{conf}' = (I', i')$ be two valid circuit configurations and $g^* \in [n+1, N]$ be a gate such that:
- $i = i'$.
- $g^* \notin I$, $I' = I \cup \{g^*\}$ and $g^* \leq i$.
- Either $g^* = n+1$ or $g^* - 1 \in I$.

[5] This is because the number of gates in the Gray mode increases the circuit size of SimGate by a proportional factor.

In Lemma 4, we will show that for two valid configurations $\mathsf{conf}, \mathsf{conf}'$ satisfying the above constraints we have that $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$. Note that we can also use this rule to move a gate g^* from Gray mode to White mode. We refer to those invocations of the rule as *inverse A rule*. Rule A is illustrated in Figure 8.

Rule B: Rule B says that for any configuration for any valid configuration conf we can indistinguishably change gate g^* in Gray mode to Black mode if all gates subsequent to g^* is in Black mode and the predecessor is in Gray mode. More formally, let $\mathsf{conf} = (I, g^*)$ and $\mathsf{conf}' = (I', g')$ be two valid circuit configurations such that:

- $g^* = g' + 1$.
- $g^* \in I$, $I' = I \setminus \{g^*\}$.
- Either $g^* = n + 1$ or $g^* - 1 \in I$.

In Lemma 5, we will show that for an valid configurations $\mathsf{conf}, \mathsf{conf}'$ satisfying the above constraints we have that $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$. Rule B is illustrated in Figure 9.

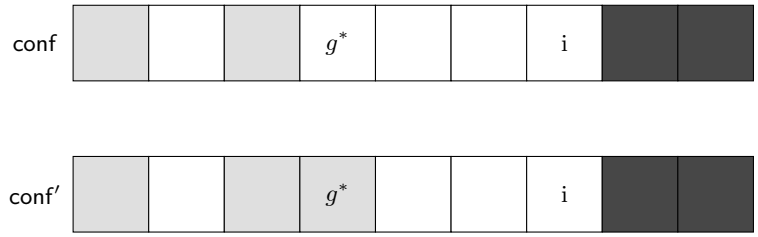

Figure 8: Example of Rule A

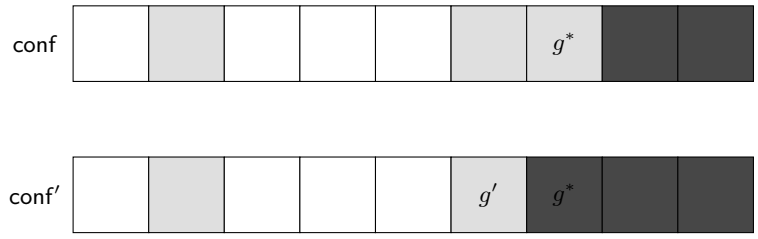

Figure 9: Example of Rule B

4.2.2 Interpreting the rules of indistinguishability as a pebbling game Sections 4.2.2 and 4.2.3 are taken verbatim from [GS18a]. Our sequence of hybrids from the real to the ideal world follow an optimal strategy for the following pebbling game. The two rules described above correspond to the rules of our pebbling game below.

Consider the positive integer line $n+1, n+2, \ldots N$. We are given pebbles of two colors: gray and black . A black pebble corresponds to a gate in the Black (i.e., input independent simulation) mode and a gray pebble corresponds to a gate in the Gray (i.e., input dependent simulation) mode. A position without any pebble corresponds to real garbling or in the White mode. We can place the pebbles on this positive integer line according to the following two rules:

Rule A: We can place or remove a gray pebble in position i if and only if there is a gray pebble in position $i-1$. This restriction does not apply to position $n+1$: we can always place or remove a gray pebble at position $n+1$.

Rule B: We can replace a gray pebble in position i with a black pebble as long as all the positions $> i$ have black pebbles and there is a gray pebble in position $i-1$ or if $i = n+1$.

Optimization goal of the pebbling game. The goal is to pebble the line $[n+1, N]$ such that every position has a black pebble while minimizing the number of gray pebbles that are present on the line at any point in time.

4.2.3 Optimal Pebbling Strategy To provide some intuition, we start with the naïve pebbling strategy. The naïve pebbling strategy involves starting from position $n+1$ and placing a gray pebble at every position in $[n+1, N]$ and then replacing them with black pebbles from N to $n+1$. However, this strategy uses a total of $N-n$ gray pebbles. Using a more clever strategy, it is actually possible to do the same using only $\log(N-n)$ gray pebbles. We first recall the following lemma from [GPSZ17].

Lemma 2 ([GPSZ17]). *For any integer $n+1 \leq p \leq n+2^k-1$, it is possible to make $O((p-n)^{\log_2 3}) \approx O((p-n)^{1.585})$ moves and get a gray pebble at position p using k gray pebbles.*

Proof. For completeness we give the proof. This proof is taken verbatim from [GPSZ17].

First we observe to get a gray pebble placed at p, for each $i \in [n+1, p-1]$ there must have been at some point a gray pebble placed at location i.

Next, we observe that it suffices to show we can get a gray pebble at position $p = n + 2^k - 1$ for every k using $O(3^k) = O((p-n)^{\log_2 3})$ steps. Indeed, for more general p, we run the protocol for $p' = n + 2^k - 1$ where $k = \lceil \log_2(p-n-1) \rceil$, but stop the first time we get a gray pebble at position p. Since $p'/p \leq 3$, the running time is at most $O((p-n)^{\log_2 3})$.

Now for the algorithm. The sequence of steps will create a fractal pattern, and we describe the steps recursively. We assume an algorithm A_{k-1} using $k-1$ gray pebbles that can get a gray pebble at position $n + 2^{k-1} - 1$. The steps are as follows:

- Run A_{k-1}. There is now a gray pebble at position $n + 2^{k-1} - 1$ on the line.
- Place the remaining gray pebble at position $n + 2^{k-1}$, which is allowed since there is a gray pebble at position $n + 2^{k-1} - 1$.
- Run A_{k-1} in reverse, recovering all of the $k-1$ gray pebbles used by A. The result is that there is a single gray pebble on the line at position $n + 2^{k-1}$.
- Now associate the portion of the number line starting at $n + 2^{k-1} + 1$ with a new number line. That is, associate $n + 2^{k-1} + a$ on the original number line with $n' + a$ (where $n' = n + 2^{k-1}$) on the new number line. We now have $k-1$ gray pebbles, and on this new number line, all of the same rules apply. In particular, we can always add or remove a gray pebble from the first position $n'+1 = n+2^{k-1}+1$ since we have left a gray pebble at $n+2^{k-1}$. Therefore, we can run A_{k+1} once more on the new number line starting at $n' + 1$. The end result is a pebble at position $n'+2^{k-1} - 1 = n+2^{k-1}+(2^{k-1}-1) = n+2^k-1$.

It remains to analyze the running time. The algorithm makes 3 recursive calls to A_{k-1}, so by induction the overall running time is $O(3^k)$, as desired.

Using the above lemma, we now give an optimal strategy for our pebbling game.

Lemma 3 ([GS18a]). *For any $N \in \mathbb{N}$, there exists a strategy for pebbling the line graph $[n+1, N]$ according to rules A and B by using at most $\log N$ gray pebbles and making $\mathsf{poly}(N)$ moves.*

Proof. The proof is taken verbatim from [GS18a].

The strategy is given below. For each g from N down to $n+1$ **do**:

1. Use the strategy in Lemma 2 to place a gray pebble in position g. Note that there exists a gray pebble in position $g-1$ as well.
2. Replace the gray pebble in position g with a black pebble. This replacement is allowed since all positions $> g$ have black pebbles and there is a gray pebble in position $g-1$.
3. Recover all the gray pebbles by reversing the moves.

The correctness of this strategy follows by inspection and the number of moves is polynomial in N.

4.3 Proof of Indistinguishability for the Rules

In this subsection, we will use the security of underlying primitives to implement the two rules.

4.3.1 Implementing Rule A

Lemma 4 (Rule A). *Let conf and conf' be two valid circuit configurations satisfying the constraints of rule A, then assuming the security of garbling scheme for circuits, updatable laconic oblivious transfer, indistinguishability obfuscation and puncturable PRFs we have that $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$.*

Proof. We prove this via a hybrid argument.

- $\underline{\mathsf{Hybrid}_{\mathsf{conf}}}$: This is our starting hybrid and is distributed as $\mathsf{Hybrid}_{(I,i)}$.
- $\underline{\mathsf{Hybrid}_1}$: In this hybrid, instead of hardwiring the PPRF keys K and S in the circuit $\mathsf{SimGate}$, we hardwire the key K that is punctured at (g^*, k, b) for every $k \in [\lambda], b \in \{0,1\}$ and S punctured at g^*. We additionally hardwire $\{\mathsf{lab}_{k,b}^{g^*}\}_{k\in[\lambda],b\in\{0,1\}}$ and $\mathsf{PRF}_S(g^*)$. This blows up the size of the circuit by a factor $\mathsf{poly}(\lambda)$. On input g^*-1 and g^*, the circuit now uses the hardwired labels/randomness instead of computing them using the PPRF.
  It can be noted that the $\mathsf{SimGate}$ circuits in both $\mathsf{Hybrid}_{\mathsf{conf}}$ and Hybrid_1 computes the exact same functionality and hence the indistinguishability between $\mathsf{Hybrid}_{\mathsf{conf}}$ and Hybrid_1 follows from the security of $i\mathcal{O}$.
- $\underline{\mathsf{Hybrid}_2}$: We make three changes to the $\mathsf{SimGate}$.
  - By conditions of Rule A, we have that $g^*-1 \in I$ (if $g^* \neq n+1$). Therefore, we note that all the input labels $\{\mathsf{lab}_{k,b}^{g^*}\}$ are not used in $\mathsf{SimGate}$ but only the labels corresponding to d_{g^*} i.e., $\{\mathsf{lab}_{k,\mathsf{d}_{g^*},k}^{g^*}\}_{k\in[\lambda]}$. We just hardwire these labels in $\mathsf{SimGate}$.
  - We also hardwire $\widetilde{\mathsf{SC}}_{g^*}$ (that is computed using randomness $\mathsf{PRF}_S(g^*)$) in $\mathsf{SimGate}$ instead of generating it inside $\mathsf{SimGate}$.
  - We remove the hardwired randomness $\mathsf{PRF}_S(g^*)$.

  The computational indistinguishability between Hybrid_2 from Hybrid_1 follows from the security of $i\mathcal{O}$ since the function computed by $\mathsf{SimGate}$ in Hybrid_1 and Hybrid_2 is exactly the same.
- $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we sample the labels $\{\mathsf{lab}_{k,\mathsf{d}_{g^*},k}\}_{k\in[\lambda]}$ and the randomness used in generating $\widetilde{\mathsf{SC}}_{g^*}$ uniformly at random instead of generating them as outputs of the puncturable PRF. The computational indistinguishability between Hybrid_2 and Hybrid_3 follows from the security of puncturable PRF.
- $\underline{\mathsf{Hybrid}_4}$: In this hybrid, we generate $\widetilde{\mathsf{SC}}_{g^*}$ (that is hardwired inside $\mathsf{SimGate}$) from the simulated distribution. More formally, we generate

$$\widetilde{\mathsf{SC}}_{g^*} \leftarrow \mathsf{Sim}_{ckt}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}_{k,\mathsf{d}_{g^*},k}^{g^*}\}_{k\in[\lambda]})$$

  where $\mathsf{out} \leftarrow \mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i, j, f_g), \{\mathsf{lab}_{k,b}^{g^*+1}\}, 0](\mathsf{d}_{g^*})$.
  The only change in hybrid Hybrid_3 from Hybrid_2 is in the generation of the garbled circuit $\widetilde{\mathsf{SC}}_{g^*}$ and the security follows directly from the selective security of the garbling scheme.
- $\underline{\mathsf{Hybrid}_5}$: In this hybrid, we change how the output value out hardwired in $\widetilde{\mathsf{SC}}_{g^*}$ is generated. Recall that in Hybrid_4 this value is generated by first computing c_0 and c_1 as in Figure 5 and then generating out as $\mathsf{Send}(\mathsf{crs}, \mathsf{d}, i, c_0, c_1)$. In this hybrid, we just generate $c_{D_{g^*},i}$ and use the laconic OT simulator to generate out. More formally, out is generated as

$$\mathsf{out} \leftarrow \mathsf{Sim}_{\ell\mathsf{OT}}\left(\mathsf{crs}, D_{g^*}, i, c_{D_{g^*},i}\right).$$

  Computational indistinguishability between hybrids Hybrid_4 and Hybrid_5 follows directly from the sender privacy of the laconic OT scheme.

- $\underline{\mathsf{Hybrid}_6}$: In this hybrid, we change how the value $c_{D_{g^*,i}}$ is generated. Recall from Figure 5 that $c_{D_{g^*,i}}$ is set as $\mathsf{Send}(\mathsf{crs}, \mathsf{d}, j, (\gamma(D_{g^*,i}, 0), e_{\gamma(D_{g^*,i},0)}), (\gamma(D_{g^*,i}, 1), e_{\gamma(D_{g^*,i},1)}))$. We change the distribution of $c_{D_{g^*,i}}$ to $\mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D_{g^*}, j, e_{D_{g^*+1,g^*}})$, where $e_{D_{g^*+1,g^*}}$ is sampled as in Figure 5.
  Computational indistinguishability between hybrids Hybrid_6 and Hybrid_5 follows directly from the sender privacy of the laconic OT scheme. The argument is analogous to the argument of indistinguishability between Hybrid_4 and Hybrid_5.
- $\underline{\mathsf{Hybrid}_7}$: In this hybrid, we change how $e_{D_{g^*+1,g^*}}$ is generated. More specifically, we generate it using the simulator $\mathsf{Sim}_{\ell\mathsf{OTW}}$. In other words, $e_{D_{g^*+1,g}}$ is generated as

$$\mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D_{g^*}, g^*, D_{g^*+1,g^*}, \{\mathsf{lab}^{g^*+1}_{k,\mathsf{d}_{g^*+1,k}}\}_{k\in[\lambda]}).$$

  Computational indistinguishability between hybrids Hybrid_6 and Hybrid_7 follows directly from the sender privacy for writes of the laconic OT scheme.
- $\underline{\mathsf{Hybrid}_8 - \mathsf{Hybrid}_{10}}$: In this hybrid, we reverse the changes made in Hybrid_1 to Hybrid_3 except that we hardwire $\{\mathsf{out}_{g^*}, \mathsf{d}_{g^*}\}$ in $\mathsf{SimGate}$ and use it to generate $\widetilde{\mathsf{SC}}_{g^*}$. The indistinguishability between Hybrid_7 to Hybrid_{10} follows in analogous manner to the indistinguishability between $\mathsf{Hybrid}_{\mathsf{conf}}$ to Hybrid_3. Finally, observe that hybrid Hybrid_{10} is the same as $\mathsf{Hybrid}_{\mathsf{conf}'}$.

This completes the proof of the lemma. We additionally note that the above sequence of hybrids is reversible. This implies the inverse rule A.

4.3.2 Implementing Rule B

Lemma 5 (Rule B). *Let* conf *and* conf' *be two valid circuit configurations satisfying the constraints of rule B, then assuming the security of somewhere equivocal encryption, garbling scheme for circuits and updatable laconic oblivious transfer, we have that* $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$.

Proof. We prove this via a hybrid argument starting with $\mathsf{Hybrid}_{\mathsf{conf}'}$ and ending in hybrid $\mathsf{Hybrid}_{\mathsf{conf}}$. We follow this ordering of the hybrids as this keeps the proof very close to the proof of Lemma 4.

- $\underline{\mathsf{Hybrid}_{\mathsf{conf}'}}$: This is our starting hybrid and is distributed as $\mathsf{Hybrid}_{(I',g')}$.
- $\underline{\mathsf{Hybrid}_1}$: In this hybrid, instead of hardwiring the PPRF keys K, R and S in the circuit $\mathsf{SimGate}$, we hardwire the key K that is punctured at (g^*, k, b) for every $k \in [\lambda], b \in \{0,1\}$, R and S are punctured at g^*. We additionally hardwire $\{\mathsf{lab}^{g^*}_{k,b}\}_{k\in[\lambda],b\in\{0,1\}}$, $(r_i, r_{j,g})$, $\mathsf{PRF}_R(g^*)$ and $\mathsf{PRF}_S(g^*)$. This blows up the size of the circuit by a factor $\mathsf{poly}(\lambda)$. On input $g^* - 1$ and g^*, the circuit now uses the hardwired labels/randomness instead of computing them using the PPRF. Note that by constraints on conf and conf', $\mathsf{PRF}_R(g^*)$ is only needed on input g^*. This is because all gates $g > g^*$ are in Black mode. It can be noted that the $\mathsf{SimGate}$ circuits in both $\mathsf{Hybrid}_{\mathsf{conf}}$ and Hybrid_1 computes the exact same functionality and hence the indistinguishability between $\mathsf{Hybrid}_{\mathsf{conf}}$ and Hybrid_1 follows from the security of $i\mathcal{O}$.

– $\underline{\mathsf{Hybrid}_2}$: We make three changes to the $\mathsf{SimGate}$.
  - By conditions of Rule A, we have that $g^*-1 \in I$ (if $g^* \neq n+1$). Therefore, we note that all the input labels $\{\mathsf{lab}_{k,b}^{g^*}\}$ are not used in $\mathsf{SimGate}$ but only the labels corresponding to d_{g^*} i.e., $\{\mathsf{lab}_{k,\mathsf{d}_{g^*,k}}^{g^*}\}_{k\in[\lambda]}$. We just hardwire these labels in $\mathsf{SimGate}$.
  - We also hardwire $\widetilde{\mathsf{SC}}_{g^*}$ (where SC_{g^*} has r_{g^*} hardwired and $\widetilde{\mathsf{SC}}_{g^*}$ is computed using randomness $\mathsf{PRF}_S(g^*)$) in $\mathsf{SimGate}$ instead of generating it inside $\mathsf{SimGate}$.
  - We remove the hardwired randomness $\mathsf{PRF}_S(g^*)$ and $\mathsf{PRF}_R(g^*)$.

  The computational indistinguishability between Hybrid_2 from Hybrid_1 follows from the security of $i\mathcal{O}$ since the function computed by $\mathsf{SimGate}$ in Hybrid_1 and Hybrid_2 is exactly the same.
– $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we sample the labels $\{\mathsf{lab}_{k,\mathsf{d}_{g^*,k}}\}_{k\in[\lambda]}$, $\mathsf{PRF}_R(g^*)$ and the randomness used in generating $\widetilde{\mathsf{SC}}_{g^*}$ uniformly at random instead of generating them as outputs of the puncturable PRF. The computational indistinguishability between Hybrid_2 and Hybrid_3 follows from the security of puncturable PRF.
– $\underline{\mathsf{Hybrid}_4}$: In this hybrid, we generate $\widetilde{\mathsf{SC}}_{g^*}$ (that is hardwired inside $\mathsf{SimGate}$) from the simulated distribution. More formally, we generate

$$\widetilde{\mathsf{SC}}_{g^*} \leftarrow \mathsf{Sim}_{ckt}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}_{k,\mathsf{d}_{g^*,k}}^{g^*}\}_{k\in[\lambda]})$$

  where $\mathsf{out} \leftarrow \mathsf{SC}[\mathsf{crs}, (0,0,r_g), (i,j,f_g), \{\mathsf{lab}_{k,b}^{g^*+1}\}, 1](\mathsf{d}_{g^*})$.
  The only change in hybrid Hybrid_3 from Hybrid_4 is in the generation of the garbled circuit $\widetilde{\mathsf{SC}}_{g^*}$ and the security follows directly from the selective security of the garbling scheme.
– $\underline{\mathsf{Hybrid}_5}$: In this hybrid, we set change how the output value out hardwired in $\widetilde{\mathsf{SC}}_{g^*}$ is generated. Recall that in hybrid Hybrid_4 this value is generated by first computing c_0 and c_1 as in Figure 5 and then generating out as $\mathsf{Send}(\mathsf{crs}, \mathsf{d}, i, c_0, c_1)$. In this hybrid, we just generate $c_{D_{g^*,i}}$ and use the laconic OT simulator to generate out. More formally, out is generated as

$$\mathsf{out} \leftarrow \mathsf{Sim}_{\ell\mathsf{OT}}\left(\mathsf{crs}, D_{g^*}, i, c_{D_{g^*,i}}\right).$$

  Computational indistinguishability between hybrids Hybrid_4 and Hybrid_5 follows directly from the sender privacy of the laconic OT scheme.
– $\underline{\mathsf{Hybrid}_6}$: In this hybrid, we change how the how the value $c_{D_{g^*,i}}$ is generated in hybrid Hybrid_5. Recall from Figure 5 that $c_{D_{g^*,i}}$ is set as $\mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, e_{r_{g^*}}, e_{r_{g^*}}\right)$. We change the distribution of $c_{D_{g^*,i}}$ to $\mathsf{Sim}_{\ell\mathsf{OT}}\left(\mathsf{crs}, D_g, j, e_{r_{g^*}}\right)$, where $e_{r_{g^*}}$ is sampled as in Figure 5.
  Computational indistinguishability between hybrids Hybrid_5 and Hybrid_6 follows directly from the sender privacy of the laconic OT scheme. The argument is analogous to the argument of indistinguishability between Hybrid_4 and Hybrid_5.

– $\underline{\mathsf{Hybrid}_7}$: In this hybrid, we change how $e_{r_{g^*}}$ is generated. More specifically, we generate it using the simulator $\mathsf{Sim}_{\ell\mathsf{OTW}}$. In other words, $e_{r_{g^*}}$ is generated as

$$\mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D_{g^*}, g^*, r_{g^*}, \{\mathsf{lab}^{g^*+1}_{k,\mathsf{d}_{g^*+1,k}}\}_{k\in[\lambda]}).$$

Computational indistinguishability between hybrids Hybrid_6 and Hybrid_7 follows directly from the sender privacy for writes of the laconic OT scheme.

– $\underline{\mathsf{Hybrid}_8}$: The only difference between Hybrid_7 and Hybrid_8 is how D_{g^*+1,g^*} is set. Namely, in Hybrid_7 this value is set to be r_{g^*} while in Hybrid_8 this value is set as $r_{g^*} \oplus f_{g^*}(D_{g^*,i} \oplus r_i, D_{g^*,j} \oplus r_j)$. We argue that the distributions Hybrid_7 and Hybrid_8 are identical. Two cases arise:
  - $g^* \leq N-1$: In this case, note that since r_{g^*} is not hardwired anywhere else, we have that the distribution r_{g^*} and $r_{g^*} \oplus f_{g^*}(D_{g^*,i} \oplus r_i D_{g^*,j} \oplus r_j)$ are both uniform and identical.
  - $g^* = N$: In this case, we have that $r_{g^*} = M(x) \oplus r'_{g^*}$ which is again identical to the distribution of r_{g^*} in Hybrid_8.

– $\underline{\mathsf{Hybrid}_9 - \mathsf{Hybrid}_{11}}$: In this hybrid, we reverse the changes made in Hybrid_1 to Hybrid_3 except that we hardwire $\{\mathsf{out}_{g^*}, \mathsf{d}_{g^*}\}$ in $\mathsf{SimGate}$ and use it to generate $\widetilde{\mathsf{SC}}_{g^*}$.. The indistinguishability between Hybrid_8 to Hybrid_{11} follows in analogous manner to the indistinguishability between $\mathsf{Hybrid}_{\mathsf{conf}'}$ to Hybrid_3. Observe that Hybrid_{11} is distributed identically to $\mathsf{Hybrid}_{\mathsf{conf}}$.

This completes the proof of the lemma.

4.3.3 Completing the Hybrids The strategy given in Lemma 3 yields a sequence of configurations $\mathsf{conf}_0 \ldots \mathsf{conf}_m$ for an appropriate polynomial m with $\mathsf{conf}_0 = (\emptyset, N)$ and $\mathsf{conf}_m = (\emptyset, n)$, where $\mathsf{Hybrid}_{\mathsf{conf}_{i-1}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}_i}$ either using rule A (i.e., Lemma 4) or using rule B (i.e., Lemma 5). We now show that $\mathsf{Hybrid}_{\mathsf{conf}_m}$ is computationally indistinguishable to the ideal world distribution given by $\mathsf{Hybrid}_{(\emptyset,0)}$. This is argued using the security property of puncturable PRF using the key R and the security of $i\mathcal{O}$ as follows.

– Hybrid_1 : In this hybrid, we puncture the PRF key R at points $\{1, \ldots, n\}$ and hardwire it in $\mathsf{SimGate}$. Note that in $\mathsf{Hybrid}_{(\emptyset,n)}$, the function $\mathsf{SimGate}$ never uses the PRF key on inputs $\{1, \ldots, n\}$ and hence the functionality computed by the $\mathsf{SimGate}$ is exactly the same in this hybrid and $\mathsf{Hybrid}_{(\emptyset,n)}$. The computational indistinguishability follows from the security of $i\mathcal{O}$.
– Hybrid_2 : In this hybrid, we replace y_w with a random bit r_w for each $w \in [n]$. The computational indistinguishability between Hybrid_1 and Hybrid_2 follows from the security of puncturable PRF.
– Hybrid_3 : In this hybrid, we replace y_w with $\mathsf{PRF}_R(w)$ for every $w \in [n]$. The computational indistinguishability between Hybrid_2 and Hybrid_3 follows from the security of puncturable PRF.
– Hybrid_4 : In this hybrid, we reverse the change made in Hybrid_1 and the indistinguishability follows from the security of $i\mathcal{O}$. Notice that Hybrid_4 is distributed identically to $\mathsf{Hybrid}_{(\emptyset,\phi)}$.

Finally, the padding size ℓ is set to be maximum over the sizes of $\mathsf{SimGate}$ in every intermediate hybrid in the proof of Lemma 4, Lemma 5 and in the proof of indistinguishability between $\mathsf{Hybrid}_{(\emptyset,n)}$ and $\mathsf{Hybrid}_{(\emptyset,0)}$. This is observed to be $\mathsf{poly}(|M|, \log N, \lambda, n)$. This completes the proof of security.

References

[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.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. To appear in TCC, 2018. `https://eprint.iacr.org/2018/759`.

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. Cryptology ePrint Archive, Report 2017/385, 2017. `http://eprint.iacr.org/2017/385`.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 501–519, 2014.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press.

[BGT14] Nir Bitansky, Sanjam Garg, and Sidharth Telang. Succinct randomized encodings and their applications. Cryptology ePrint Archive, Report 2014/771, 2014. `http://eprint.iacr.org/2014/771`.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. To appear in Eurocrypt, 2018. `https://eprint.iacr.org/2017/967`.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th Annual*

Symposium on Foundations of Computer Science, pages 171–190, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 280–300, 2013.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic receiver oblivious transfer and applications. *To appear in Crypto*, 2017.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 429–437, Portland, OR, USA, June 14–17, 2015. ACM Press.

[DG17] Nico Döttling and Sanjam Garg. Identity based encryption from diffie-hellman assumptions. *To appear in Crypto*, 2017.

[DGHM18] Nico Dttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. To appear in PKC, 2018. `https://eprint.iacr.org/2017/978`.

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

[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, Heidelberg, Germany.

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

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 156–181, 2017.

[GS18a] Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. *IACR Cryptology ePrint Archive*, 2018:151, 2018.

[GS18b] Sanjam Garg and Akshayaram Srinivasan. A simple construction of io for turing machines. Cryptology ePrint Archive, Report 2018/771, 2018. `https://eprint.iacr.org/2018/771`.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in*

Cryptology – CRYPTO 2016, Part III, volume 9816 of *Lecture Notes in Computer Science*, pages 149–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684, 2013.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 121–145, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 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.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.