Succinct Garbling Schemes and Applications

Huijia Lin[*] Rafael Pass[†]

September 29, 2014

Abstract

Assuming the existence of **iO** for $P/poly$ and one-way functions, we show how to *succinctly* garble bounded-space computations (BSC) M: the size of the garbled program (as well as the time needed to generate the garbling) only depends on the size and space (including the input and output) complexity of M, but not its running time. The key conceptual insight behind this construction is a method for using **iO** to "compress" a computation that can be performed piecemeal, without revealing anything about it.

As corollaries of our succinct garbling scheme, we demonstrate the following:

- functional encryption for BSC from **iO** for $P/poly$ and one-way functions;
- *reusable* succinct garbling schemes for BSC from **iO** for $P/poly$ and one-way functions;
- succinct **iO** for BSC from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially secure one-way functions;
- (Perfect NIZK) SNARGS for bounded space and witness NP from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions.

Previously such primitives were only know to exists based on "knowledge-based" assumptions (such as SNARKs and/or differing-input obfuscation).

We finally demonstrate the first (non-succinct) **iO** for RAM programs with bounded input and output lengths, that has poly-logarithmic overhead, based on the existence of sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions.

[*]University of California at Santa Barbara, Email: `huijia@cs.ucsb.edu`.
[†]Cornell University, Email: `rafael@cs.cornell.edu`.

1 Introduction

Since the seminal work of Yao [Yao86] in the 1980s, *garbled circuits* and more generally *garbled programs* (such as e.g., garbled RAMs [LO13]) have been extensively studied. A garbling scheme consists of three procedures: a) *program garbling* method, b) an *input encoding* method, and c) an evaluation procedure. Given a program Π, the garbling method produces a "garbled" version $\widetilde{\Pi}$ of Π as well as some key *key*. Next, given the key *key* and some input x, the input encoding method produces an encoding $\tilde{x}$ of x; this encoding method is "efficient"—the time needed to encode x (and as a consequence also the length of $\tilde{x}$) does does not depend on the complexity of Π. Finally, given a garbled program Π and an encoded input $\tilde{x}$, the evaluation procedure evaluates $\Pi(x)$; additionally, given only the encoded program and inputs, $\widetilde{\Pi}$, $\tilde{x}$, an attacker cannot learn more than just $\Pi(x)$. As a direct application, a garbling scheme enables a client to, in an "off-line" stage, garble a program Π, and next provide the garbled program $\widetilde{\Pi}$ to a server. Later, in an on-line stage, when the input x on which the client wants to evaluate Π becomes known, it can *efficiently* and *without revealing neither* Π *nor* x ask the server to evaluate Π simply by sending it the encoding $\tilde{x}$ of x—in essence, garbling schemes provide a method for privately delegating computation (with an expensive off-line stage). Furthermore, on top of this direct application of garbling schemes, they have found numerous other applications (e.g., secure two-party computation [Yao86], multi-party computations [BMR90], reusable delegation of computation [GGP10], and so on and so forth.)

But a major deficiency of the all known garbling schemes (both of circuit and RAM garbling schemes) is that the size of the garbled program $\widetilde{\Pi}$ (and thus also the time needed to generate it) grows linearly with (and in particular is lower-bounded by) the circuit-size/time-complexity of the underlying program Π. Thus, even if get a succinct description of Π (e.g., as a Turing-machine or RAM program), the size of $\widetilde{\Pi}$ will be proportional to the running-time of Π.

1.1 Succinct Garbling Schemes

In this work we study *succinct garbling schemes* where the size, as well as the time required to generate, the garbled program $\widetilde{\Pi}$ only grows *logarithmically* with the running-time of the underlying program Π (but polynomially in the size of it). We provide constructions of such succinct garbling schemes for general classes of computation (assuming the existence of *indistinguishability obfuscators* [BGI+01, GGH+13b] for appropriate classes of computation), and next demonstrate several new applications of succinct garbling schemes.

As an initial observation, we show that assuming the existence of "succinct" **iO** for some "nice" class $\mathcal{C}$ of computations [BGI+01, BCP14, ABG+13], there exists succinct garbling the same class, with the same complexity.

Theorem 1 (Initial observations—Informally stated)**.** *Assume the existence of succinct indistinguishability obfuscators for a "nice" class of computations $\mathcal{C}$ and one-way function. Then, there exists a succinct garbling scheme for $\mathcal{C}$. (In particular, the size of the garbled program depends polynomially on the size of the program, the length of the input and outputs of the program, but only poly-logarithmically on its running-time and space.)*

Roughly speaking, the garbling of a program Π is an obfuscation a slightly modified program Π'_{key} that takes as input an authenticated encrypted input, decrypts and verifies the authenticity of the input (w.r.t the key *key*) and if it the input is valid simply runs Π on this input; the encoding of an input is simply an authenticated encryption of the input.[1]

[1]The overview oversimplies. To prove this construction secure assuming only that the underlying obfuscation satisfies **iO** we need to rely on a particular method for authenticated encryption.

Let us remark here that the reason the above-mentioned construction is succinct is that we rely on the succinctness of the underlying **iO** (as e.g., in the definition of **iO** for Turing machine [BGI$^+$01, BCP14, ABG$^+$13].) As such it provides little into into how to achieve succinctness "from-scratch". Additionally, to date, the only constructions of succinct **iO** for Turing machines rely on "knowledge-based" assumptions—more specifically, the existence of *differing-input obfuscation* [BGI$^+$01, BCP14, ABG$^+$13]. Rather, we are here interested in basing succinct garbling schemes on some *hardness assumption.* (As we shall see shortly, doing this will also provide insight into constructions of succinct **iO** for Turing machines based on some hardness assumption.)

Our main result shows how to obtain succinct garbling schemes relying only on (non-succinct) **iO** for *circuits*—by now there are several constructions of **iO** [GGH$^+$13b, PST14, GLSW14] for *P/poly* based on specific hardness assumptions regarding graded encodings [GGH13a], and some of them are even falsifiable [PST14, GLSW14]. Our succinct garbling construction from **iO** for *P/poly*, however, only works for any *a-priori bounded-space* computation—in other words, the size of the garbled circuit depends polynomially on the space complexity of the computation, but is independent of its running-time.

Theorem 2 (Main Theorem—Informally stated)**.** *Assume the existence of indistinguishability obfuscators for P/poly and one-way functions. Then, for every polynomial p, there exists a succinct garbling scheme for all polynomial-time programs* Π *with space-complexity* $p(\cdot)$*. (In particular, the size of the garbled program depends polynomially on the size of* Π*, the length of the input and output of* Π *and the , and the space complexity* $s(\cdot)$*, but only logarithmically on* Π*'s running-time.)*

The key conceptual insight behind our construction is a method for using **iO** to "compress" a computation that can be performed piecemeal, without revealing anything about it. More precisely, in a first step, we show how to obtain a "non-succinct" garbling of bounded-space Turing machines, and next, in a second step, we show how to use **iO** to compress this non-succinct garbling into a succinct one. (We believe that this compression technique may be of independent interest.)

Next, we use this main theorem to enable, among other things, bounded space (polynomial-time) computations in the context of a) functional encryption (FE), b) *resusable* succinct garbling schemes, c) **iO**, and d) succinct non-interactive arguments (SNARGs), assuming **iO** for *P/poly* and one-way function (for some of these results with sub exponential security); prior to this paper, these primitives could only be constructed based on "knowledge-based" assumptions [GKP$^+$13a, BCP14, ABG$^+$13] or in the Random Oracle model [Mic00].

1.2 Succinct Garbling Schemes for Bounded-Space Computations

We turn to providing an overview of our construction of succinct garbling schemes for bounded space Turing machines. Our construction proceeds in two steps: we first construct a *non-succinct* garbling scheme, with the property that the garbled program consists of many "small pieces" that can be independently generated. Next, in a second step, we use indistinguishability obfuscation to "compress" the size of the garbled program, by releasing an obfuscated program that takes an index as input and generates the "piece" corresponding to that index. As a result, the final garbled program (namely the obfuscated program) is small and can be efficiently computed, and it is only at the evaluation time that the underlying non-succinct garbled program gets "decompressed" (by running the obfuscated program on all possible indexes to recover it).

A Non-succinct Garbling Scheme: Construction Overview Let us first outline a *non succinct garbling scheme* for Turing machines[2] based on any one-way function. Note that a "trivial" approach for achieving this is to simply transform any polynomial-time Turing machine into a polynomial-size circuits and then garble the circuit. While our construction in essence relies on this principle, we provide a construction that uses as a *black-box* a garbling scheme for "small" fixed-sized circuits (and thus this construction may be of independent interest). More precisely, we will rely on the existence of a garbling scheme for circuits (as in [Yao86]) satisfying an additional useful property: the key key can be generated independently of the circuit to be garbled (more precisely, we now have a key generation algorithm Gen that outputs the key key; next, both the encoding and garbling methods receive this key as input); furthermore, we additionally require that encoded inputs can be simulated without knowledge of the circuit to be garbled. We refer to such schemes as *garbling schemes with independent key generation* and note that Yao's original scheme (which can be based on one-way functions) satisfies this property. Our non-succinct garbling scheme now proceeds as follows for a Turing machine Π with bounded space complexity $s(\cdot)$ and 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 $\mathit{key}_1, \ldots, \mathit{key}_{T(n)}$ for the $T(n)$ garbled circuits. The i^{th} garbled circuit (which is computed using key key_i) 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 outputs the outputs generated by Π, otherwise its outputs an *encoding* of this new state using key key_{i+1}. (Note that after $T(n)$ steps we are guaranteed to get to a final state and thus this process is well-defined.)

The input encoding method simply encodes the initial state of Π with input x using the key_1, and to evaluate the garbled program we simply sequentially evaluate each garbled circuit, using the encodings generated in the previous one as inputs to the next one, and finally outputting the output generated.

A Non-succinct Garbling Scheme: Proof 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 $\widetilde{\Pi}_{t^*}$ (if the simulation is valid, $\widetilde{\text{conf}}_{t^*-1}$ is supposed to be an encoding of the configuration conf_t of the TM after t steps of computation). We then iteratively in descending order simulate the i^{th} ($i < t^*$) garbled circuits $\widetilde{\Pi}_i$ with the output being set to $\widetilde{\text{conf}}_{i+1}$ generated in the previously simulated garbled circuit. We finally simulate the remaining $i > t^*$ garbled circuits $\tilde{\Pi}_i$ with the output being set to some arbitrary output in the range of the circuit (e.g., simply y), and release $\widetilde{\text{conf}}_1$ and $(\widetilde{\Pi}_1, \ldots \widetilde{\Pi}_{T(n)})$. (Note that the fact that we simulate the i^{th} ($i > t^*$) garbled circuit with the output being set to some arbitrary value is fine since encoded inputs to those circuits are not released.)

To prove indistinguishability of this simulation, 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 $T(n) - j$

[2]The choice of a Turing machine as the model of computation is arbitrary and the solution work no matter what the model of computation is.

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 $\widehat{\mathrm{conf}}_j$ of the actual configuration conf_j of the TM after t steps. It follows from the security of the garbling scheme (and the fact that a only single encoded input is released for circuit $j+1$) that hybrids H_j and H_{j+1} are indistinguishable and thus also H_0 (i.e., the real experiment) and $H_{T(n)}$ (i.e., the simulation).

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

The Succinct Garbling Scheme: Construction Overview Let us now turn to making this garbling scheme succinct. The key idea is to, instead of releasing the actual garbled circuits, release an obfuscation of the *randomized* program that generates the garbled circuits. More precisely, we release an indistinguishability obfuscation of a program $\mathbf{\Pi}^{x,s,s'}(t)$ where $x \in \{0,1\}^n$ is an input to Π, $t \in [T(n)]$ is a "time-step" of Π and s is the seed for a PRF F: $\mathbf{\Pi}^{x,s,s'}(t)$ generates and outputs the t^{th} garbled circuit in the non-succinct garbling of Π using pseudo-random coins generated by the PRF with seed s and s'. More specifically, it uses $\mathsf{F}(s,t)$ and $\mathsf{F}(s,t+1)$ as randomness to generate key_t and key_{t+1} (recall that the functionality of the t^{th} garbled circuit may depend on key_{t+1}), and uses $\mathsf{F}(s',t)$ as randomness for generating the t^{th} garbled circuit.

Now, the new succinct garbled program is the obfuscated program $\mathbf{\Lambda} \overset{\$}{\leftarrow} i\mathcal{O}(\mathbf{\Pi}^{x,s,s'})$, and the encoding $\hat{x}$ of x remains the same as before, except that now 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 first generating the entire non-succinct garbled program by computing $\mathbf{\Lambda}$ on every time step t, and evaluating the non-succinct garbling with $\hat{x}$.

The Succinct Garbling Scheme: Proof 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 garbled program enable exactly this.

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 on input t:

- outputs the simulation of the t^{th} garbled circuit (as described in the simulation of the non-sucking garbling scheme, and using y as the output of the whole program) using $\mathsf{F}(s,t)$ and $\mathsf{F}(s,t+1)$ as randomness to generate $\widetilde{\mathrm{conf}}_t$ and $\widetilde{\mathrm{conf}}_{t+1}$, and $\mathsf{F}(s',t)$ as randomness to generate the simulated garbled circuit;

The encoding of input $\tilde{x}$ is simulated as the non-succinct garbling scheme does, but using pseudo-random coins $\mathsf{F}(s,1)$. (Note that we here strongly rely on the independent simulation property of the non-succinct garbled program constructed above, which in turn relies on the independent key generation property of the underlying garbled circuit.)

It is not hard to see that this simulation works if the obfuscation is virtually black-box secure, as the entire truth tables of the two programs $\mathbf{\Pi}^{x,s,s'}$ and $\widetilde{\mathbf{\Pi}}^{y,t^*,s,s'}$ are indistinguishable when the hardwired PRF keys s, s' are chosen at random. Our goal, however, is to show that assuming indistinguishability obfuscation suffices. Towards doing this, as above, we consider a sequence of hybrid experiments $H'_0, \ldots, H'_{T(n)}$ that obfuscates a sequence of programs that "morph" gradually from $\mathbf{\Pi}$ to $\widetilde{\mathbf{\Pi}}$. In particular, the program $\widetilde{\mathbf{\Pi}}_j^{x,s,s'}$ 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 generated using seeds s, s' are used instead of truly random coins. (More specifically, for the first j inputs, $\widetilde{\mathbf{\Pi}}_j$ produces simulated garbled circuits (as in H_j), and for the rest inputs, it produces honestly generated garbled circuits.)

To prove indistinguishability of any two consecutive hybrids H'_j and H'_{j+1}, we finally rely on the punctured program technique of Sahai and Waters [SW14], to replace pseudo-random coins $\mathsf{F}(s, j+1)$, $\mathsf{F}(s', j+1)$ for generating the $j+1^{\text{th}}$ simulated garbled circuit with truly random coins, and can then rely on the indistinguishability of the simulation of the $j+1^{\text{th}}$ garbled circuit to conclude the indistinguishability of neighboring hybrids.

A note on the efficiency of the Garbling Scheme Note that the time (and size) of the garbled program depends polynomially on the space bound and the length of the inputs and outputs and only poly-logarithmically in the upper bound on the running-time of the program. The evaluation time, on the other hand, is linear in the time complexity of the program Π—*no matter what model of computation we rely on (e.g., TM or RAM or PRAM)*, and again polynomial in the space and input/output lengths.

1.3 Applications of (Succinct) Garbling Schemes

We now turn to presenting applications of garbling schemes. While our focus here is applications of *succinct* garbling schemes, as we shall explain shortly, our results are general and lead to interesting corollaries also when relying on non succinct schemes.

Application 1: Succinct Randomized Encoding Recall that a *randomized encoding* [IK02, AIK04] is a method to, given an input x and a function f, (randomly) encode $f(x)$ in a way that leaks nothing beyond just $f(x)$. As is well known, garbling schemes imply randomized encoding: the randomized encoding of f,x is simply the garbling of f and the encoding of x. Whereas previous works on randomized encoding have focused on encoding methods that can be performed by low-depth computations [AIK04], when relying on succinct garbling schemes, we obtain a new type of a *succinct randomized encoding* where the encoding can be produced much more *efficiently* than computing f—in particular, the time needed to produce the encoding only grows poly-logarithmically with the time-complexity of f. We next show how such succinct randomized encodings are useful for applications. For notational convenience, we describe these applications using garbling scheme, but it should be appreciated that for these application succinct randomized encodings actually suffice.

Applications 2: FE, reusable garbling schemes, secure computations We observe that in contexts such as secure computation [GMW87] and functional encryption [SW05, O'N10, BSW12], to evaluate a function f on an input x, it suffices to evaluate the *randomized* function that computes a garbled program of f and an encoding of the input (recall that by the security of the garbling scheme this reveal no more than the output of the function).[3] Thus, by plugging-in

[3]That is, we are computing a randomized encoding of $f(x)$.

our construction of succinct garbling schemes for bounded-space computations (BSC) into earlier constructions of secure computation or *randomized* functional encryption [GJKS][4], we directly obtain, assuming **iO** for *P/poly* and one-way functions, a randomized functional encryption for BSC, and secure computation protocols for bounded-space programs, where the communication complexity is grows logarithmically with the the running-time of the program to be evaluated. We additionally observe that by combing our construction of functional encryption for BSC with previous results [CIJ$^+$13, GKP$^+$13b]—[CIJ$^+$13] showed that function encryption schemes with indistinguishability-based security implies ones with simulation-based security, which further implies reusable garbling schemes by [GKP$^+$13b]—directly yields a construction of *reusable* succinct garbling schemes for BSC from **iO** for *P/poly* and one-way functions. Summarizing,

Theorem 3 (Informally stated). *Assume the existence of indistinguishability obfuscators for P/poly and one-way function. Then, for every polynomial p, the exists secure constructions of the following primitives which handle computations all polynomial-time computations with space-complexity $s(\cdot)$:*

- reusable *succinct garbling;*
- *functional encryption where the size of the secret key for a function is independent of its running time;*
- *secure computation where the communication complexity of the protocol is independent of the running-time of the program.*

Let us mentioned that prior to this paper, the second of these primitives could only be constructed based on "knowledge-based" assumptions (such as SNARKs and extractable witness encryption [GKP$^+$13a] or differing-input obfuscation [BCP14, ABG$^+$13]), and the third one based on incomparable assumptions (namely, FHE—it is unknown whether **iO** and one-way functions imply FHE).

Applications 3: Succinct iO and SNARGs Recall that in Theorem 1 we demonstrated how to use succinct **iO** to get succinct garbling schemes—in fact, the construction works for any "nice". We now show a converse of this result: assuming sub-exponentially-secure **iO** for *P/poly*, the existence of a sub-exponentially-secure succinct garbling scheme for a "nice"'[5] class of algorithms yields an **iO** for the same class of algorithms *with the same complexity.* We first observe that if we had an appropriate notion of **iO** for randomized functionalities, then we could rely on the same argument as in the context of secure computation and functional encryption—instead of evaluating a function f, simply compute the randomized functionality that computes the garbled version of f and the encoded input. We observe that a notion recently considered by Canetti, Lin, Tessaro and Vaikuntanathan [CLTV14] (which can be achieved based on sub-exponentially-secure **iO** and one-way functions) suffices for our purposes as long as the garbling scheme is sub-exponentially secure.

Combined with Theorem 2, this yields succinct **iO** for BSC from sub-exponentially-secure **iO** for *P/poly* and sub-exponentially secure one-way functions. Plugging in this result into the Perfect NIZK construction of Sahai-Waters [SW14] directly yields a construction of (Perfect NIZK)

[4]The reason we need randomized functional encryption is to be able to compute the randomized garbling function.

[5]Here by "nice", we mean that $\mathcal{C}$ (1) contains algorithms with a-priori polynomially-bounded input and output lengths, (2) is closed under composition with polynomial-sized circuits, and (3) algorithms contained in $\mathcal{C}_\lambda$ is also contained in $\mathcal{C}_{\lambda'}$, with $\lambda' \geq \lambda$. The last requirement is a technicality in order to enable applying a cryptographic algorithm on an algorithm from $\mathcal{C}_\lambda$ with a bigger security parameter λ'.

SNARGS for bounded-space NP from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions. Summarizing,

Theorem 4 (Informally state)**.** *Assume the existence of sub-exponentially-secure indistinguishability obfuscators for P/poly and sub-exponentially-secure one-way function. Then, for every polynomial p, there exists*

- *iO for all polynomial-time computations with space-complexity* $s\cdot)$*;*
- *(Perfect NIZK) SNARGS (with adaptive soundness)*[6] *for all languages in NP that can be decided by a non-deterministic polynomial-time Turing machines with space-complexity* $s(\cdot)$*.*

These primitives were only know to exists based on "knowledge-based" assumptions [BCP14, ABG^{+}13, BP13] or in the Random Oracle Model [Mic00].

Application 4: iO for RAM with poly-logarithmic overhead We finally observe that the above observation that in the context of **iO** it suffices to evaluate the garbling of the function instead of directly evaluating the functions is useful also if relying on non-succinct garbling schemes. In particular, by relying on the garbled RAM constructions of [LO13, GHL^{+}14] we directly obtain as a corollary,

- **iO** for RAM programs with bounded input and output lengths, where the size of the obfuscated program only grows quasi-linearly with the RAM complexity of the program, assuming **iO** for $P/poly$ and one-way functions, both with sub-exponential security.

There is one subtle detail that needs to be dealt with to obtain the above corollary. If simply relying on any **iO** in the above construction, then the use of this underlying primitive could blow-up the running-time. (If we rely on an **iO** for circuits with quasi-linear overhead then we are fine, but this seems to require stronger assumptions.) Rather, we rely on an observation from Gentry, Halevi, Raykova and Wichs [GHRW14]: the garbled RAM constructions of [LO13, GHL^{+}14] satisfy a nice "bit-wise compactness" property, where each bit of the garbled circuit can be independently generated by a "small" circuit of size depending quasi-linearly in the input and output lengths and only poly-logarithmically in the running time.[7] Thus, (inspired by [GHRW14],) instead of obfuscating the program that generates the whole garbled circuit in one shot, we simply obfuscate *many* "small" programs, each of which generates one bit of the garbled circuit. (Note that the resulting **iO** is not succinct: the size of the obfucated program depends on the size of the garbled RAM).

Let us remark that a similar construction was recently provided in [GHRW14]; the difference between our construction and theirs is that they propose to obfuscate only a single "small" program that will generate all bits in the garbled RAM, whereas our **iO** consists of the obfuscation of many "small" programs, each of which generates only a single bit in the garbled RAM. But, the authors of [GHRW14] simply conjecture the security of their construction; in contrast, we prove it secure assuming that the underlying obfuscator satisfies sub-exponentially secure **iO**.

[6]The Perfect NIZK construction of [SW14] 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.

[7]More precisely, the size of the small circuit is $\tilde{O}(|R| + n + m) \times \mathrm{poly}(\lambda, \log T)$, where R is the RAM machine under consideration, n and m are its input and output lengths, and T is its running time.

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 $\mathsf{ALG}^T = \left\{\mathcal{AL}_\lambda^T\right\}$ to denote the class of algorithms in $\mathsf{ALG} = \{\mathcal{AL}_\lambda\}$ that run in time $T(\lambda)$ (in particular, these with $AL_\lambda.T < T(\lambda)$).

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

2.2 Garbling Scheme

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 $(\hat{AL}, \mathbf{key})$.*[8]

[8](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* $(\hat{AL}, \hat{x})$ *produced by* $\mathsf{Garb}, \mathsf{Encode}$ *outputs* y.

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

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

Definition 2 (Security of a Garrbling 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:*

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

where $(n, m, S, T) = (AL.n, AL.m, AL.S, AL.T)$ and Sim *runs in time* $\mathrm{poly}(\lambda, T'(\lambda))$*. Moreover, 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 $\hat{AL}, \hat{x}$ 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 $(1^\lambda, 1^{|x|}, 1^{|C|}, AL(x))$, 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* $p_{\mathsf{Garb}}(\lambda, |AL|, AL.m)$,[9]
- $\hat{x} = \mathsf{Encode}(\mathbf{key}, x)$ *runs in time* $p_{\mathsf{Encode}}(\lambda, |x|, AL.m)$*, and*
- $y = \mathsf{Eval}(\hat{AL}, \hat{x})$ *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 $AL.n$ 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.*

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

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

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

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

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

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 $|C|$.

[9]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 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 Key Generation. 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 key generation.*

Definition 7 (Garbling Scheme for Circuits with Independent Key Generation)**.** *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 key generation** *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* $\hat{C} \overset{\$}{\leftarrow} \mathsf{Gb}(\mathbf{key}, C)$*, 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* $(\tilde{x}, \mathbf{st}) \overset{\$}{\leftarrow} \mathsf{Sim{\cdot}Gen}(1^\lambda, 1^{|x|})$ *and then* $\tilde{C} \overset{\$}{\leftarrow} \mathsf{Sim{\cdot}Gb}((1^\lambda, 1^{|x|}, 1^{|C|}, C(x), \mathbf{st})$*, where* $\mathsf{Sim{\cdot}Gen}$ *and* $\mathsf{Sim{\cdot}Gb}$ *are all uniform* PPT *algorithms.*

It is easy to check that many known circuit garbling schemes, in particular the construction by Yao [Yao86], has independent key generation.

Proposition 1. *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 key generation 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 $\left\{\mathcal{AL}_\lambda^T \times \mathcal{AL}_\lambda^T \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$, and adversary $\mathcal{A}$, there is a negligible function μ, such that, for sufficiently large $\lambda \in \mathbb{N}$, if*

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

Then,

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

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 "programing" 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[10] 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 6 (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 6.*

By using the canonical representation, our general definition allows the garbling or ***iO*** *scheme to depend on any subset of parameters flexibly. Naturally, if a scheme depends on a subset of parameters, the resulting garbled or obfuscated machines may "leak" these parameters (in the above three examples above, the size of the garbled or obfuscated machines leaks the parameters they depend on); thus, the security definitions must reflect this "leakage" correspondingly. The general security definitions 2 and 8 captures this by allowing leakage of all parameters* $|AL|, AL.n, AL.m, AL.S, AL.T$*.*

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

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 [SW14]. 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* $(\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F})$*, and a pair of computable functions* $n(\cdot)$ *and* $m(\cdot)$*, satisfying the following conditions:*

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

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

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

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

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

As observed by [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 A Succinct Garbling Scheme for BSTM

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 $|M| \le D, |\,\mathrm{conf}_t\,| \le S$, outputs the next configuration conf_{t+1}—we call this circuit the "universal next-step circuit". The size of the circuit is a fixed polynomial p_{Next} in the size of the machine and the configuration, that is, $p_{\mathsf{Next}}(D, S)$. The whole execution of $M(x)$ can be carried out by performing at most T evaluations of $\mathsf{Next}^{D,S}(M, \cdot)$, producing a chain of configurations denoted by,

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

When succinctness is not required, the natural idea to garble a T-step Turing machine computation of $M(x)$ is to produce a chain of T garbled circuits $(\widehat{\mathbf{C}}_1, \cdots, \widehat{\mathbf{C}}_T)$, for evaluating the next step circuit $\mathsf{Next}^{D,S}(M, \cdot)$ for M. The t^{th} circuit $\mathbf{C}_t$ is designated to compute from the t^{th} configuration conf_t (as input) to the next conf_{t+1}; if the produced conf_{t+1} is a final configuration, then it simply outputs the output y; otherwise, to enable the evaluation of the next garbled circuit $\widehat{\mathbf{C}}_{t+1}$, it translates conf_{t+1} into the corresponding garbled inputs $\widehat{\mathrm{conf}}_{t+1}$ for $\widehat{\mathbf{C}}_{t+1}$—we call $\mathbf{C}_t$ the t^{th} *step-circuit*. Then evaluation propagates and the intermediate configurations of the execution of M on x is implicitly computed one by one, until it reaches the final configuration, in which case, an output is produced explicitly (without translating into the garbled inputs of the next garbled circuit). Since each computation step is garbled, and all intermediate configurations, except from the final output y, are "encrypted" as garbled inputs, the entire chain of garbled circuits can be simulated given only the output y.

Finally, we note that each step-circuit $\mathbf{C}_t$ evaluates $\mathsf{Next}^{D,S}(M, \cdot)$ and has the capability of garbling an input for the next garbled circuit $\widehat{\mathbf{C}}_t$; this can only be achieved if the circuit garbling scheme has independent key generation, 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 key generation.

- 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 key generation, that is, $\mathsf{Garb}_{\mathsf{CIR}} = (\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Gb}_{\mathsf{CIR}})$, and $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim}\text{·}\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Sim}\text{·}\mathsf{Gb}_{\mathsf{CIR}})$ 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 $t+1^{\text{th}}$ 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[11]; (iii) otherwise, translate conf_{t+1} to the garbled inputs of the $t+1^{\text{th}}$ 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 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.

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

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.

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

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

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 $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}, \mathsf{Sim{\cdot}Gb}_{\mathsf{CIR}})$ of the circuit garbling scheme, which requires knowing the output of each garbled circuit. In a real evaluation with $\hat{M}, \hat{x}$, the output of the $(T^*)^{\text{th}}$ garbled circuit is $y = M(x)$, the output of the garbled circuits $t < T^*$ is the garbled input $\widehat{\mathrm{conf}}_{t+1}$ for next garbled circuit $t+1$, and the garbled circuits $t > T^*$ are not evaluated, but for which y is a valid output. Thus, in the simulation, garbled circuits $t = T^*, \cdots, T$ can be simulated using output y; whereas garbled circuits $t = 1, \cdots, T^* - 1$ will be simulated using the *simulated garbled inputs* for circuit $t+1$. More precisely,

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

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

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

1. *Prepare the output out_t for the t^{th} simulated circuit $\widetilde{\mathbf{C}}_t$:*
   If $t \geq T^*$, $out_t = y$. Otherwise, if $t < T^*$, set the output as the garbled input for the next garbled circuits, that is, $out_t = \widetilde{\mathrm{conf}}_{t+1}$ computed from $(\widetilde{\mathrm{conf}}_{t+1}, \mathbf{st}_{t+1}) = \mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_{t+1})$ 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 $(\widetilde{\text{conf}}_t, \mathbf{st}_t) = \mathsf{Sim}\cdot\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_t)$ and then $\widetilde{\mathbf{C}}_t = \mathsf{Sim}\cdot\mathsf{Gb}_{\mathsf{CIR}}(1^\lambda, 1^S, 1^q, out_t, \mathbf{st}_t\ ;\ \beta_t)$, 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 $(\widetilde{\text{conf}}_1, \mathbf{st}_1) = \mathsf{Sim}\cdot\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_1)$ 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 $t \leq \gamma - 1$, compute $\widetilde{\mathbf{C}}_t = \mathbf{Q}^{\lambda, S, |M|, T^*, y}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$; output $\widetilde{\mathbf{C}}_t$.
- If $t \geq \gamma + 1$, compute $\widehat{\mathbf{C}}_t = \mathbf{P}^{\lambda, S, M}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$; output $\widehat{\mathbf{C}}_t$.
- If $t = \gamma$, compute $\widetilde{\mathbf{C}}_t = \mathbf{R}^{\lambda, S, \text{conf}_{\gamma+1}}(\gamma\ ;\ (\alpha_\gamma, \alpha_{\gamma+1}, \beta_\gamma))$ 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 $\gamma + 1^{\text{th}}$ 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 $(\widetilde{\text{conf}}_\gamma, \mathbf{st}_\gamma) = \mathsf{Sim}\cdot\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_\gamma)$ and $\widetilde{\mathbf{C}}_t = \mathsf{Sim}\cdot\mathsf{Gb}_{\mathsf{CIR}}(1^\lambda, 1^S, 1^q, out_\gamma, \mathbf{st}_\gamma\ ;\ \beta_\gamma)$, 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 γth 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 γth 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 γth 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 $\square$

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 $\overline{\mathbb{P}} \stackrel{\$}{\leftarrow} i\mathcal{O}(\mathbb{P})$. 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 programing 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 key generation: $\mathcal{GS}_{\mathsf{CIR}} = (\mathsf{Garb}_{\mathsf{CIR}}, \mathsf{Encode}_{\mathsf{CIR}}, \mathsf{Eval}_{\mathsf{CIR}})$, where $\mathsf{Garb}_{\mathsf{CIR}} = (\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Gb}_{\mathsf{CIR}})$ and its the simulation algorithm is $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}, \mathsf{Sim{\cdot}Gb}_{\mathsf{CIR}})$.
- An indistinguishability obfuscator $i\mathcal{O}_{\mathsf{CIR}}(\cdot, \cdot)$ for polynomial-sized circuits.
- A puncturable PRF ($\mathsf{PRF{\cdot}Gen}, \mathsf{PRF{\cdot}Punc}, \mathsf{F}$) 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:* $K_\alpha \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{P}$*:*
   Obfuscate the circuit $\mathbb{P}(t) = \mathbb{P}^{\lambda,S,M,K_\alpha,K_\beta}(t)$ as described in Figure 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 $\mathbf{key}$ 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.

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

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:

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

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

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

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

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{Q}$*:*
   Obfuscate the circuit $\mathbb{Q}(t) = \mathbb{Q}^{\lambda, S, |M|, T^*, y, K_\alpha, K_\beta}(t)$ as described in Figure 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 $(\widetilde{\mathrm{conf}}_1, \mathbf{st}_1) = \mathsf{Sim{\cdot}Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_1)$, 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 $\mathsf{Sim}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T^*, y = M(x))$ 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 $\mathsf{Sim}_{\mathsf{CIR}}(1^\lambda, 1^S)$ 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:* $K_\alpha \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{M}^\gamma$*:*
   Obfuscate the circuit $\mathbb{M}^\gamma(t) = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \mathrm{conf}_{\gamma+1}, K_\alpha, K_\beta}(t)$ 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 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

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

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

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

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

- For every γ, the underlying obfuscated programs $\mathbb{M}^\gamma, \mathbb{M}^{\gamma+1}$ differ on their implementation for at most two inputs, namely $\gamma, \gamma+1$, and,

- when $\gamma = 0$, the garbled input $\tilde{x}^0$ is generated honestly in hyb^0, whereas $\tilde{x}^1$ is simulated in hyb^1.

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

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

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

$$\begin{aligned}
(\widetilde{\mathrm{conf}}_{\gamma+1}, \mathbf{st}_{\gamma+1}) &= \mathsf{Sim}\text{-}\mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha'_{\gamma+1}) && (5)\\
\widetilde{\mathbf{C}}_{\gamma+1} &= \mathsf{Sim}\text{-}\mathsf{Gb}_{\mathsf{CIR}}\left(1^\lambda, 1^S, 1^q, out_{\gamma+1}, \mathbf{st}_{\gamma+1}; \beta'_{\gamma+1}\right) && (6)
\end{aligned}$$

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 $(\widetilde{\text{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ 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 $\widetilde{\text{conf}}_{\gamma+1}$ 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 $\widetilde{\text{conf}}_1$ as described in equation (5), and then output $\tilde{x}_4 = \widetilde{\text{conf}}_1$.

It follows from the fact that $\mathbb{M}_5$ computes $(\widetilde{\text{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ 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 of Bounded Space 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 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 $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$, 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 garble each block, augmented with the encoding algorithm of $\mathcal{GS}$ that translates the next configuration to the garbled input for the next garbled block (when an output is produced, it is output directly without translation)—call this an "augmented block". The final garbling then consists of a sequence of T garbled blocks.

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

The efficiency and security analysis remains the same as before. This concludes Theorem 6.

5 From Garbling to FE to Reusable Garbling

We observe that in contexts such as secure computation [GMW87] and functional encryption [SW05, O'N10, BSW12], to evaluate a function f on an input x, it suffices to evaluate the *randomized* function that computes a garbled program of f and an encoding of the input (recall that by the security of the garbling scheme this reveal no more than the output of the function). Thus, by plugging-in our construction of garbling schemes with space-dependent complexity into earlier constructions of secure computation or *randomized* functional encryption [GJKS], we directly obtain, assuming **iO** for $P/poly$ and one-way functions, a randomized functional encryption with space-dependent complexity, and secure computation protocols whose the communication complexity grows polynomially with the space complexity of the program to be evaluated, but only logarithmically with the the running-time.

We additionally observe that by combing our construction of functional encryption with previous results [CIJ$^+$13, GKP$^+$13b]—[CIJ$^+$13] showed that function encryption schemes with indistinguishability based security implies ones with simulation-based security, which further implies reusable garbling schemes by [GKP$^+$13b]—directly yields a construction of *reusable* succinct garbling schemes with space-dependent complexity from **iO** for $P/poly$ and one-way functions.

We emphasize that the above applications work generally for any "nice" class of algorithms; (conditions on the algorithm classes are specified in the theorem statements below). Therefore, below we show the connections between Garbling, randomized functional encryption and reusable garbling w.r.t. general classes of algorithms.[12] Applications to specific models of computation can then be derived as special cases.

Below, we start with the definitions of function encryption and reusable garbling scheme, and then move to showing the connections.

5.1 Functional Encryption

We first recall definitions of public key functional encryption schemes, both the indistinguishability-based definitions and simulation-based definitions, adapting to general algorithm classes. For indistinguishability-based security, we recall the definition for randomized algorithms in [GJKS], whereas for simulation-based security, we recall the definition of [BSW12, O'N10] for deterministic Boolean algorithms. Below we first introduce the syntax, then various security notions and finally the efficiency guarantees.

Syntax. We introduce the syntax and correctness of functional encryption scheme w.r.t. classes of potentially randomized algorithms, which immediately imply the syntax and correctness for deterministic algorithms.

Definition (Functional Encryption.). *A functional encryptions scheme $\mathcal{FE}$ for a class of (well-formed) (potentially randomized) algorithms $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of algorithms* ($\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec}$) *where the first three are probabilistic algorithms while the last is deterministic; furthermore, they satisfy the following syntax and correctness:*

- $\mathsf{FE.Setup}(1^\lambda)$ *outputs a public key* MPK *and a master secret key* MSK.
- $\mathsf{FE.Enc}(x, \mathsf{MPK})$ *outputs a ciphertext* CT.
- $\mathsf{FE.KeyGen}(AL, \mathsf{MSK})$ *on input an algorithm $AL \in \mathcal{AL}_\lambda$, outputs a secret key* sk_{AL}.
- $\mathsf{FE.Dec}(\mathsf{CT}, \mathsf{sk}_{AL})$ *outputs a string y.*

Correctness: *For every polynomial T and polynomial $n = n(\lambda)$, every sequence of algorithm tuples $\{\vec{AL} = \vec{AL}_\lambda\} \in \{\mathcal{AL}^n_\lambda\}$ and every sequence of input tuples $\{\vec{x} = \vec{x}_\lambda\}$ where $x_i \in \{0,1\}^{\mathrm{poly}(\lambda)}$, the following two distributions are computationally indistinguishable:*

- **Real:** $\left\{\mathsf{FE.Dec}(\mathsf{CT}_i, SK_{AL_j})\right\}_{i\in[n],j\in[n]}$ *where,*
  $(\mathsf{MPK}, \mathsf{MSK}) \xleftarrow{\$} \mathsf{FE.Setup}(1^\lambda)$
  $\mathsf{CT}_i \xleftarrow{\$} \mathsf{FE.Enc}(\mathsf{MPK}, x_i)$ *for* $i \in [n]$
  $\mathsf{sk}_{AL_j} \xleftarrow{\$} \mathsf{FE.KeyGen}(AL_j, \mathsf{MSK})$ *for* $j \in [n]$
- **Ideal:** $\{AL_j(x_i; r_{ij})\}_{i\in[n],j\in[n]}$ *where,*
  for every $j \in [n]$, $\{r_{ij}\}_{i\in[n]}$ are sufficiently long random strings if AL_j is a randomized algorithm and are empty if AL_j is a deterministic algorithm. (In the case that x_i exceeds the input length bound of the algorithm AL_j, $|x_i| > AL_j.n$, the output is set to $\perp$).

[12]Since the application to MPC is straightforward, we omit the details of the proof below.

Indistinguishability based security. We provide the definition of indistinguishability based security for classes of (potentially randomized) algorithms. Our definition is a generalization of that in [GJKS] to the case of full security.

Definition 13 ((Full) $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND-security)**.** *A functional encryption scheme $\mathcal{FE}$ is* (ful-)$\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND *secure if for every polynomial T, the advantage of any* PPT *adversary $\mathcal{A}$ in the following game is negligible.*

Experiment (ful-)$\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-$\mathsf{IND}_{\mathcal{A}}^{\mathcal{FE}}(1^\lambda, T = T(\lambda))$**:**

1. *Setup:* $(\mathsf{MPK}, \mathsf{MSK}) \xleftarrow{\$} \mathsf{FE.Setup}(1^\lambda)$
2. *Non-Adaptive Key Queries:* $(\vec{m}_0, \vec{m}_1, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}^{\mathsf{FE.KeyGen}(\mathsf{MSK},\cdot),\mathcal{O}_1(\cdot)}(\mathsf{MPK})$.
3. *Generate Challenge Ciphertext:* $\forall i$, $\mathsf{CT}_i \xleftarrow{\$} \mathsf{FE.Enc}(\mathsf{MPK}, m_{b,i})$*, where* $b \xleftarrow{\$} \{0,1\}$.
4. *Adaptive Key Queries:* $\alpha \xleftarrow{\$} \mathcal{A}^{\mathsf{FE.KeyGen}(\mathsf{MSK},\cdot),\mathcal{O}_2(\cdot)}(\mathbf{st}_1, \vec{\mathsf{CT}})$.

$\mathcal{O}_1$ is a stateful oracle that on input $(AL, null)$, generates a secret key $\mathsf{sk}_{AL} \xleftarrow{\$} \mathsf{FE.KeyGen}(\mathsf{MSK}, AL)$ and records it internally, and on input (AL, C) checks if a secret key has been generated for AL, and returns $\mathsf{FE.Dec}(\mathsf{sk}_{AL}, C)$ if it is the case ($\perp$ otherwise). $\mathcal{O}_2$ is identical to $\mathcal{O}_1$ except that it only decrypts ciphertext $C \neq \mathsf{CT}_i$ for all i. The advantage of $\mathcal{A}$ is $\Pr[\alpha = b] - 1/2$.

Restriction on $\mathcal{A}$: *Let S_1 and S_2 represent the sets of non-adaptive and adaptive key queries made by $\mathcal{A}$ in Step 2 and 4 respectively; let Q_1, Q_2 be the sets of key queries $\mathcal{A}$ submits to $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. $\mathcal{A}$ must follow the restriction that 0) $S_1, S_2, Q_1, Q_2 \subseteq \mathcal{AL}_\lambda^T$, 1) $|S_1| \leq q_1(\lambda)$, 2) $|S_2| \leq q_2(\lambda)$, 3) $|\vec{m}_0| = |\vec{m}_1| \leq \ell$ and for every $i \in [|\vec{m}_0|]$ $|m_{0,i}| = |m_{1,i}|$, and 4) for every i, j, let AL_j be the j^{th} query in $S_1 \cup S_2$, the following distributions are indistinguishable:*

$$\{r \xleftarrow{\$} \{0,1\}^{\text{poly}(\lambda)} \ : \ (AL_j(m_{0,i}; r), T_{AL}(m_{0,i}; r))\}_\lambda$$
$$\{r \xleftarrow{\$} \{0,1\}^{\text{poly}(\lambda)} \ : \ (AL_j(m_{1,i}; r), T_{AL}(m_{1,i}; r))\}_\lambda$$

Additionally, we consider the following weaker notions.

Definition 14 (Selective $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND-security)**.** *A functional encryption scheme $\mathcal{FE}$ is* **selective** $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND *secure if for every polynomial T, the advantage of any* PPT *adversary $\mathcal{A}$ is negligible in the* sel-$\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-$\mathsf{IND}(1^\lambda, T)$ *game, which is the same as the above* $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND *game, except that $\mathcal{A}$ is required to select the challenge messages $\vec{m}_0, \vec{m}_1$ at the beginning of experiment before* $\mathsf{MPK}, \mathsf{MSK}$ *are chosen.*

Note that in the selective game sel-$\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND, it is without loss of generality to remove Step 2 in the experiment; thus, we can assume w.l.o.g. that $q_1 = 0$. Furthermore, it follows from standard hybrid argument that for any polynomials q_1, ℓ, q_2, $\mathsf{q_1}$-$\mathsf{1}$-$\mathsf{q_2}$-IND security implies $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND security, both in the selective and full game. Thus in the rest of the paper we use interchangeably $\mathsf{q_1}$-$\mathsf{1}$-$\mathsf{q_2}$-IND security and $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND security.

Definition 15 (Honest-Sender (full or selective) $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND-Security)**.** *We say that a functional encryption scheme is honest-sender (full or selective)* $\mathsf{q_1}$-ℓ-$\mathsf{q_2}$-IND *secure, if it satisfies the same security condition as in Definition 13, except that, in the experiments the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ are empty.*

When the class of algorithms are deterministic, the standard definition in the literature considers only honest-sender security. We follow this convention; when an algorithm class is deterministic only security against malicious receiver is considered.

Definition ((Honest-sender) Full or Selective IND-security). *A functional encryption scheme $\mathcal{FE}$ is (honest-sender)* $\mathsf{ful\text{-}IND}$*-secure or* $\mathsf{sel\text{-}IND}$*-secure, if it is (honest-sender)* $\mathsf{ful\text{-}poly\text{-}1\text{-}poly\text{-}IND}$*-secure or* $\mathsf{sel\text{-}0\text{-}1\text{-}poly\text{-}IND}$*-secure.*

Simulation based security. Next we proceed to define simulation based security notions. In this case, we consider only classes of algorithms $\{\mathcal{AL}_\lambda\}$ that are *deterministic*. Furthermore, we note that it is without loss of generality to consider only Boolean algorithms. This is because a functional encryption scheme for Boolean algorithms can be easily turned into a scheme for algorithms with m-bit outputs, by running the Boolean scheme for m times in parallel. Such parallel repetition leads to a scheme where the ciphertext length is linear in $n \times m$ (as well as in λ), where n is the input length. On the other hand, it was shown that simulation-based secure functional encryption must have the size of ciphertexts growing linearly with the output length (if there is any non-adaptive queries made before the challenge ciphertexts are generated). Thus, the parallel repetition is essentially optimal.

Definition. *A functional encryption scheme $\mathcal{FE}$ for a class $\{\mathcal{AL}_\lambda\}$ of deterministic* Boolean *algorithms with is* $(\mathsf{ful\text{-}})\mathsf{q}_1\text{-}\ell\text{-}\mathsf{q}_2\text{-}\mathsf{SIM}$ *secure if for every* PPT *adversary $\mathcal{A}$, there exists a simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that the output of the following two experiments are indistinguishable for every polynomial T.*

Experiment $(\mathsf{ful}\text{-})\mathsf{q}_1\text{-}\ell\text{-}\mathsf{q}_2\text{-}\mathsf{RealExp}_{\mathcal{A}}^{\mathcal{FE}}(1^\lambda, T = T(\lambda))$**:**

1. *Setup:* $(\mathsf{MPK}, \mathsf{MSK}) \xleftarrow{\$} \mathsf{FE.Setup}(1^\lambda)$
2. *Non-Adaptive Key Queries:* $(\vec{m}, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}^{\mathsf{FE.KeyGen}(\mathsf{MSK},\cdot)}(\mathsf{MPK})$*.*
3. *Generate Challenge Ciphertext:* $\forall i$, $\mathsf{CT}_i \xleftarrow{\$} \mathsf{FE.Enc}(\mathsf{MPK}, m_i)$*, where* $b \xleftarrow{\$} \{0,1\}$*.*
4. *Adaptive Key Queries:* $\alpha \xleftarrow{\$} \mathcal{A}^{\mathsf{FE.KeyGen}(\mathsf{MSK},\cdot)}(\mathbf{st}_1, \vec{\mathsf{CT}})$*.*
5. *Output* $(\mathsf{MPK}, \vec{m}, S_1, S_2, \alpha)$*, where* S_1 *and* S_2 *are the sets of non-adaptive and adaptive key queries made by* $\mathcal{A}$ *in Step 2 and 4 respectively.*

Experiment $(\mathsf{ful}\text{-})\mathsf{q}_1\text{-}\ell\text{-}\mathsf{q}_2\text{-}\mathsf{IdealExp}_{\mathcal{A}}^{\mathcal{FE}}(1^\lambda, T = T(\lambda))$**:**

1. *Setup:* $(\mathsf{MPK}, \mathsf{MSK}) \xleftarrow{\$} \mathsf{FE.Setup}(1^\lambda)$
2. *Non-Adaptive Key Queries:* $(\vec{m}, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}^{\mathsf{FE.KeyGen}(\mathsf{MSK},\cdot)}(\mathsf{MPK})$*; let* $\ell' = |\vec{m}|$
3. *Generate Challenge Ciphertext:* $\underline{(\vec{\mathsf{CT}}, \mathbf{st}') \xleftarrow{\$} \mathsf{Sim}_1(\mathsf{MPK}, \ell', \{|m_i|\}, \mathcal{V})}$
4. *Adaptive Key Queries:* $\underline{\alpha \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}(\mathbf{st}_1, \vec{\mathsf{CT}})}$
5. *Output* $(\mathsf{MPK}, \vec{m}, S_1, S_2, \alpha)$*.*

In the above experiment $\mathcal{V}$ *in Step 3 contains the outputs of the algorithms in* S_1 *applied to* $\vec{m}$*, that is,* $\mathcal{V} = \cup_{Q \in S_1} \mathcal{V}_Q$ *with* $\mathcal{V}_Q = \{Q, \mathsf{sk}_Q, T_Q(m_i), Q(m_i)\}_{i \in [\ell']}$*. Additionally, the oracle* $\mathcal{O}(\cdot)$ *in Step 4 is the second stage simulator, namely* $\mathsf{Sim}_2(\mathbf{st}', \mathsf{MSK}, \cdot, \cdot)$*, where the third argument is a query circuit* Q *and the fourth argument is* $\mathcal{V}_Q$*.*

Restriction on $\mathcal{A}$ in the above two experiments: *0)* $S_1, S_2 \subseteq \mathcal{AL}_\lambda^T$ *1)* $|S_1| \le q_1(\lambda)$*, 2)* $|S_2| \le q_2(\lambda)$*, 3)* $|\vec{m}| \le \ell$*.*

Additionally, a functional encryption scheme $\mathcal{FE}$ *is (honest-sender)* **selective** $\mathsf{q}_1\text{-}\ell\text{-}\mathsf{q}_2\text{-}\mathsf{SIM}$ *secure if for every* PPT *adversary* $\mathcal{A}$*, the outputs of the experiments* $\mathsf{sel}\text{-}\mathsf{q}_1\text{-}\ell\text{-}\mathsf{q}_2\text{-}\mathsf{RealExp}$ *and* $\mathsf{sel}\text{-}\mathsf{q}_1\text{-}\ell\text{-}\mathsf{q}_2\text{-}\mathsf{IdealExp}$ *are indistinguishable for every polynomial* T*, where the two games are the same as the above, except that* $\mathcal{A}$ *is required to select the challenge messages* $\vec{m}_0, \vec{m}_1$ *at the beginning of experiment before* $\mathsf{MPK}, \mathsf{MSK}$ *are chosen (and without access to the* $\mathsf{FE.KeyGen}$ *oracle).*

We note that in the above definition, the second state simulator Sim_2 receive as input the instance running time $T_Q(m_i)$ for every Q and m_i. This is necessary since the efficiency guarantees below require the decryption algorithm to have instance-based efficiency.

Efficiency. Finally, We now move to define different efficiency requirement for functional encryption.

Definition (Different Levels of Efficiency of Functional Encryption Scheme)**.** *We say that a functional encryption scheme* $\mathcal{FE}$ *has optimal efficiency, or I/O- / space- / linear-time dependent complexity if the following conditions hold.*

Optimal efficiency: *Algorithms* $\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}$ *run in time polynomial in their input lengths, that is* $p_{\mathsf{Setup}}(\lambda)$, $p_{\mathsf{Enc}}(\lambda, |x|)$, $p_{\mathsf{KeyGen}}(\lambda, |AL|)$, *and* $\mathsf{FE.Dec}$ *runs in time* $p_{\mathsf{Dec}}(\lambda, |AL|, |x|)T_{AL}(x)$ *for a polynomial* p_{Dec}.

I/O-dependent complexity: *The above condition holds, except that the running time of* $\mathsf{FE.KeyGen}$ *and* $\mathsf{FE.Dec}$ *additionally depends on* $AL.n, AL.m$.

Space-dependent complexity: *The above condition holds, except that the running time of* $\mathsf{FE.KeyGen}$ *and* $\mathsf{FE.Dec}$ *additionally depends on* $AL.S$.

linear-time-dependent complexity: *The above condition holds, except that the running time of* $\mathsf{FE.KeyGen}$ *and* $\mathsf{FE.Dec}$ *depends quasi-linearly on* $AL.T$.

We note that the $\mathsf{FE.Setup}$ and $\mathsf{FE.Enc}$ always runs polynomially in their input length, independent of the parameters of the algorithms that are potentially to be evaluated. Furthermore, in the case of SIM-secure functional encryptions, we only consider Boolean algorithms; thus, it is possible to have schemes with optimal efficiency, where the encryption and key generation algorithms runs in time independent of the output length.

5.2 Reusable Garbling Scheme

We recall the definition of reusable garbled circuits in [GKP+13b], adapted for general algorithm classes. In [GKP+13b], their definition considers adaptive input selection, and their construction achieves so. An alternative definition with static input selection is considered in [GHRW14] for reusable garbled RAM. The two variants corresponds tightly with fully or selectively SIM-secure functional encryption schemes. Below we define both variants.

Definition 16 (Reusable Security [GKP+13b].)**.** *We say that a garbling scheme* $\mathcal{GS}$ *for a class of deterministic Boolean algorithms* $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$ *has reusable security with adaptive input selection, if for all* PPT *adversary* $\mathcal{A}$ *there is a simulator* $\mathsf{Sim} = (\mathsf{Sim{\cdot}Gb}, \mathsf{Sim{\cdot}Inp})$ *the following two games are indistinguishable for all polynomial* T.

Experiment $\mathsf{RealExp}^{\mathcal{GS}}_{\mathcal{A}}(1^\lambda, T = T(\lambda))$**:**

1. $(AL, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)$
2. $(\hat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL)$
3. $\alpha \xleftarrow{\$} \mathcal{A}^{\mathsf{Encode}(\mathbf{key},\cdot)}(\mathbf{st}_1, \hat{AL})$

Experiment $\mathsf{IdealExp}^{\mathcal{GS}}_{\mathcal{A}}(1^\lambda, T = T(\lambda))$**:**

1. $(AL, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)$
2. $(\tilde{AL}, \mathbf{st}) \xleftarrow{\$} \mathsf{Sim{\cdot}Gb}(1^\lambda, |AL|, (AL.n, AL.m, AL.S, AL.T))$
3. $\alpha \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}(\mathbf{st}_1, \tilde{AL})$

where the oracle in the third step is the second stage simulator $\mathsf{Sim{\cdot}Inp}(\mathbf{st}, \cdot, \cdot, \cdot)$ *that receives as the second parameter the length of the input* $1^{|x|}$*, as the third parameter the instance running time* $T_{AL}(x)$ *and the fourth parameter the output* $AL(x)$*.*
Restriction on $\mathcal{A}$ **in the above two experiments:** *the algorithm* AL *chosen by* $\mathcal{A}$ *is in* $\mathcal{AL}^T_\lambda$*.*

Furthermore, we say that $\mathcal{GS}$ *has reusable security with selective input selection, if the above indistinguishability is true for all* PPT *adversary* $\mathcal{A}$ *that selects its oracle queries in the third step at the beginning of the games.*

5.3 IND-secure FE for General Classes of Randomized Algorithm

We show a general method for constructing (full or selective) IND-secure functional encryption for a general class $\{\mathcal{AL}_\lambda\}$ of (potentially randomized) algorithms, from a garbling scheme $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ for the corresponding class of "de-randomized" algorithms, and a (full or selective respectively) IND-secure functional encryption for circuits.

Towards this, consider a randomized algorithm AL, let $\mathsf{deRand}[AL, k]$ be the following de-randomized algorithm corresponding to AL, where $k \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda)$ is a PRF key.

$$\mathsf{deRand}^{\mathsf{F}}[AL, k](x) : \text{ Run } AL(x) \text{ with pseudo-random coins } r_t = \mathsf{F}(k, t) \text{ for step } t$$

Then the high-level idea for constructing a IND-secure FE $\mathcal{FE}^A$ for $\{\mathcal{AL}_\lambda\}$ is the following: The secret key corresponding to algorithm AL will a the secret key generated using the IND-secure circuit-FE $\mathcal{FE}^C$ for the (randomized) algorithm that garbles the de-randomized C_{AL} as described above. More specifically, the circuit C_{AL} is constructed as follows:

$$(\hat{\Gamma}, \hat{x}) \xleftarrow{\$} C_{AL}(x) \ : \ k \xleftarrow{\$} \mathsf{PRF{\cdot}Gen}(1^\lambda),\ (\hat{\Gamma}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, \mathsf{deRand}^{\mathsf{F}}[AL, k]),\ \hat{x} \xleftarrow{\$} \mathsf{Encode}(\mathbf{key}, x)$$

Below we state our result formally. Note that since the garbling algorithm must work with algorithms of the form $\mathsf{deRand}^{\mathsf{F}}[AL, k])$, we require that it is for an algorithms class that is *closed under composition with polynomial-sized circuits,* which implies that for every $AL \in \mathcal{AL}_\lambda$, $\mathsf{deRand}^{\mathsf{F}}[AL, k])$ is in $\mathcal{AL}_\lambda$.

Proposition 2 (IND-secure FE for General Classes of Randomized Algorithms)**.** *Let* $\{\mathcal{AL}_\lambda\}$ *be any class of (potentially randomized) algorithms that is closed under composition with polynomial-sized circuits. It holds that if there are*

- i) *a garbling scheme* $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ *for deterministic algorithms in* $\{\mathcal{AL}_\lambda\}$.

  ii) *a* (ful-)IND*-secure (or* sel-IND*-secure) functional encryption scheme* IND-$\mathcal{FE}^C$ *for the class* rCIR *of polynomial-sized randomized circuits.*

- **then**, *there is a* (ful-)IND*-secure (or* sel-IND*-secure respectively) functional encryption scheme* IND-$\mathcal{FE}^A$ *for* $\{\mathcal{AL}_\lambda\}$.

Furthermore, if $\mathcal{GS}$ *has optimal efficiency or I/O-dependent complexity,* IND-$\mathcal{FE}^A$ *has I/O-dependent complexity. If* $\mathcal{GS}$ *has space- / linear-time- dependent complexity, so does* IND-$\mathcal{FE}^A$.

Proof. Below we give the construction and proof. Given an IND-secure circuit FE $\mathcal{FE}^C = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$, our IND-secure FE is constructed as follows: It has the same setup and encryption algorithm. The key generation and decryption algorithms invokes the algorithms FE.KeyGen and FE.Dec as sub-routines. To generate a key for algorithm AL, $\mathcal{FE}^A$ generates a key $\mathsf{sk}_{C_{AL}}$ using $\mathcal{FE}^C$ for the circuit C_{AL} as described above and returns $\mathsf{sk}_{AL} = \mathsf{sk}_{C_{AL}}$. To decrypt a ciphertext c of value x, it invokes the decryption algorithm $\mathsf{FE.Dec}(\mathsf{sk}_{C_{AL}}, c)$ to obtain a pair $(\hat{\Gamma}, \hat{x})$, and then evaluates the garbled pair to obtain an output y.

The IND-security of $\mathcal{FE}^A$ reduces to the IND-security of $\mathcal{FE}^C$. This follows as an adversary $\mathcal{A}$ attacking the former can be transformed into an adversary $\mathcal{A}'$ attacking the latter. More specifically, in an $\mathsf{IND}^{\mathcal{FE}^C}_{\mathcal{A}'}$ game, the adversary $\mathcal{A}'$ can internally emulate a game $\mathsf{IND}^{\mathcal{FE}^A}_{\mathcal{A}}$ for $\mathcal{A}$: For every key query AL from $\mathcal{A}$, $\mathcal{A}'$ translates AL to a key query C_{AL} to its own key generation oracle, and for every decryption query c from $\mathcal{A}$, $\mathcal{A}'$ simply relays c to its own decryption oracles, and upon receiving an output $(\hat{\Gamma}, \hat{x})$, it evaluates the garbled pair to obtain y and feeds $\mathcal{A}$ with y.

It is easy to see that $\mathcal{A}$ emulates the view of $\mathcal{A}'$ perfectly. The only thing we need to establish is that whenever $\mathcal{A}$ sends a legitimate key query AL—that is, for the two challenge messages x_1 and x_2, the outputs and running time of AL on x_1 and x_2 are indistinguishable—the translated key queries C_{AL} is also legitimate—that is, its outputs on x_1 and x_2 are indistinguishable. Below we argue this.

It follows from the pseudo-randomness of PRF and the security of the garbling scheme that for any polynomial T, any T-time randomized algorithm AL and any two inputs x_1, x_2 such that $AL(x_1; r), T_{AL}(x_1, r)$ for random r is indistinguishable from $AL(x_2; r), T_{AL}(x_2, r)$ for random r, we have that the outputs of C_{AL} on input x_1 and x_2 are also indistinguishable. More precisely, for every polynomial T,

$$\begin{array}{ll} \forall & \{AL\}_\lambda \in \{\mathcal{AL}^T_\lambda\}, \{x_1\}_\lambda, \{x_2\}_\lambda, \\ \text{if} & \left\{r \xleftarrow{\$} \{0,1\}^{AL.T} \ : \ AL(x_1; r), T_{AL}(x_1, r)\right\}_\lambda \approx \left\{r \xleftarrow{\$} \{0,1\}^{AL.T} \ : \ AL(x_2; r), T_{AL}(x_2, r)\right\}_\lambda, \\ \text{then,} & \{C_{AL}(x_1)\}_\lambda \approx \{C_{AL}(x_2)\}_\lambda \end{array}$$

This follows since for each $i \in \{1, 2\}$, it follows from the pseudo-randomness of the PRF that the output and running time of $\mathsf{deRand}^{\mathsf{F}}[AL, k]$ on x_1 and x_2 are indistinguishable, when the PRF key is chosen at random, that is,

$$\begin{aligned} &\left\{k \xleftarrow{\$} \mathsf{PRF}\text{·}\mathsf{Gen}(1^\lambda) \ : \ \mathsf{deRand}^{\mathsf{F}}[AL, k](x_1), T_{\mathsf{deRand}[AL,k]}(x_1)\right\}_\lambda \\ \approx &\left\{k \xleftarrow{\$} \mathsf{PRF}\text{·}\mathsf{Gen}(1^\lambda) \ : \ \mathsf{deRand}^{\mathsf{F}}[AL, k](x_2), T_{\mathsf{deRand}[AL,k]}(x_2)\right\}_\lambda, \end{aligned}$$

Then it follows from the security of the garbling scheme that the output of $C_{AL}(x_1)$, which is a freshly generated garbled pair $\{\hat{\Gamma}, \hat{x}_1\}$, can be simulated by random variables in the first ensemble above, and that of $C_{AL}(x_2)$ can be simulated by variables in the second ensembles. Therefore, the indistinguishability of $C_{AL}(x_1)$ and $C_{AL}(x_2)$ holds.

This concludes the security of $\mathcal{FE}^A$. $\square$

Combining with known garbling schemes for circuits [Yao86], and TM and RAM with space-dependent complexity (our construction), the above lemma immediately implies the following theorem:

Theorem 7. *The following statements hold:*

- *Assume the existence of randomized functional encryption for* NC^1*, there is a randomized functional encryption for* $\mathsf{P/poly}$*.*
- *Assume the existence of randomized functional encryption for* $\mathsf{P/poly}$*, there is a randomized functional encryption scheme for* TM *and* RAM *with space-dependent complexity.*

5.4 From IND security to SIM security.

The work by De Caro et. al. [CIJ+13] showed a tight connection between functional encryption with indistinguishability-based security and that with simulation-based security. More precisely, they provide a generic transformation that turns any (full or selective) q-ℓ-poly-IND secure functional encryption scheme IND-$\mathcal{FE}$ for polynomial-sized circuits to a (full or selective) q-ℓ-poly-SIM functional encryption scheme SIM-$\mathcal{FE}$ for polynomial-sized circuits, assuming one-way functions.

With a closer examination of their transformation, it is easy to see that their transformation works for general classes of algorithms (beyond circuits). Below we state the generic transformation theorem implicit in their security proof.

Proposition 3 (Generic Transformation from FE with IND-Security to SIM-Security (Implicit in [CIJ+13])). *For every class* $\{\mathcal{AL}_\lambda\}$ *of deterministic Boolean algorithms* that are closed under composition with polynomial-sized circuits*, the following holds: Assuming the existence of one-way functions, for every polynomial* q *and* ℓ*,*

- *if there is a* $(\mathsf{ful}$-$)\mathsf{q}$-ℓ-poly-IND*-secure (or* sel-q-ℓ-poly-IND*-secure) functional encryption scheme* IND-$\mathcal{FE}$ *for the class of algorithms* $\{\mathcal{AL}_\lambda\}$*.*
- *then there is a* $(\mathsf{ful}$-$)\mathsf{q}$-ℓ-poly-SIM*-secure (or* sel-q-ℓ-poly-SIM*-secure respectively) functional encryption scheme* SIM-$\mathcal{FE}$ *for* $\{\mathcal{AL}_\lambda\}$*,*

Furthermore, the following efficiency preservation holds.

- *if* IND-$\mathcal{FE}$ *has optimal efficiency, or space- / linear-time-dependent complexity, so does* SIM-$\mathcal{FE}$*, and*
- *if* IND-$\mathcal{FE}$ *has succinct ciphertexts, so does* SIM-$\mathcal{FE}$*.*

Note that the fact that the resulting SIM-secure functional encryption scheme can have optimal efficiency (if the underlying IND-secure functional encryption has) does not contradict with the lower bound that the ciphertext and key sizes must scale with the output length, because we consider only Boolean algorithms. SIM-secure functional encryption scheme for multi-bit algorithms can be constructed via parallel repetition of a scheme for Boolean algorithms; in this case, the ciphertext and key sizes do scale with the output length.

5.5 From SIM-secure FE to Reusable Garbling Scheme

The work by Goldwasser et. al. [GKP$^+$13b] give a transformation from a (ful-)1-1-0-SIM-secure functional encryption scheme for circuits to a fully secure reusable garbling scheme. It is straightforward to see that their construction works for general classes of algorithms and when considering reusable garbling schemes with only static input selection, only sel-1-1-0-SIM-secure functional encryption is needed.

Proposition 4 (Generic Transformation from 1-1-0-SIM-Secure FE to Reusable Garbling Scheme (Implicit in [GKP$^+$13b]))**.** *For every class $\{\mathcal{AL}_\lambda\}$ of well-formed deterministic Boolean algorithms* that are closed under composition with polynomial-sized circuits*, the following holds: Assuming the existence of one-way functions,*

- *if there is a* (ful-)1-1-0-SIM*-secure (or* sel-1-1-0-SIM*-secure) functional encryption scheme* SIM-$\mathcal{FE}$ *for the class of algorithms $\{\mathcal{AL}_\lambda\}$,*
- *then there is a reusable garbling scheme $\mathcal{GS}$ for $\{\mathcal{AL}_\lambda\}$ with adaptive (or static) input selection.*

Furthermore, the following efficiency preservation holds.

- *if* SIM-$\mathcal{FE}$ *has optimal efficiency or I/O- /space- / linear-time-dependent complexity, so does $\mathcal{GS}$, and*
- *if* SIM-$\mathcal{FE}$ *has succinct ciphertexts, $\mathcal{GS}$ has succinct input encodings.*

Combining Proposition 2, 3 and 4, with our construction of garbling schemes with space-dependent complexity, we obtain the following.

Theorem 8. *Assume the existence of* ***iO*** *for* P/poly *and one-way function, there is re-usable garbling scheme for* TM *and* RAM *with space dependent complexity.*

6 Garbling v.s. iO

In this section we show a connection between (succinct) **iO** for classes of algorithms $\mathcal{C}$ and (succinct) garbling schemes for the same classes $\mathcal{C}$. Roughly speaking,

- assuming the existence of (succinct) **iO** for any "nice" class $\mathcal{C}$ of algorithms and one-way functions, there exists a (succinct) garbling scheme for $\mathcal{C}$;
- assuming the existence of *iO* for *P/poly* and a (succinct) garbling scheme for any "nice" class $\mathcal{C}$ of algorithms, both with sub-exponential security, there exists a (succinct) **iO** for $\mathcal{C}$.

6.1 From Garbling 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 5. *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 garbling scheme $\mathcal{GS}$ 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 $\mathcal{GS}$ has optimal efficiency or I/O-dependent complexity, $i\mathcal{O}^A$ has I/O-dependent complexity.*
- *If $\mathcal{GS}$ has space-dependent complexity, so does $i\mathcal{O}^A$.*
- *If $\mathcal{GS}$ and $i\mathcal{O}^C$ have linear-time-dependent complexity, so does $i\mathcal{O}^A$.*

Proof of Proposition 5. This result relies on a notion from the recent work by Canetti, Lin, Tessaro and Vaikuntanathan [CLTV14] which they refer to as *probabilistic* ***iO*** . In [CLTV14], they show that assuming *sub-exponentially indistinguishable* IO for circuits and *sub-exponentially secure* puncturable PRF, the following natural way for obfuscating probabilistic circuits does achieve a limited notion of indistinguishability-based security. Below, we first recall their result and show how to apply it to obtain **iO** from garbling.

Probabilistic ***iO****.* Consider the following natural way of obfuscating a *probabilistic* circuit C.

$\hat{C} \xleftarrow{\$} pi\mathcal{O}(1^\lambda, C) : \;\; k \xleftarrow{\$}$

program C'_b that evaluates C_b with pseudo-random coins generated using a hardwired PRF key k. Thus, it follows from the security of **iO** that $\hat{C}_b$ is indistinguishable to $i\mathcal{O}(C''_b)$ 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, by a hybrid argument, we have that $pi\mathcal{O}(1^\lambda, C_1) \approx pi\mathcal{O}(1^\lambda, C_2)$.

Now consider the case where C_1 and C_2 are completely different, but their output distributions are negl$(\lambda)2^{-n}$-indistinguishable. To show that their pIO obfuscation are indistinguishable, we just need to consider an exponential, 2^n, number of hybrids, where in each hybrid, the implementation for one input x^* is changed from using C_1 to C_2. By the same argument as before, neighboring hybrids have a distinguishing gap $O(\text{negl}(\lambda)2^{-n})$; thus by an exponential hybrid argument, the overall distinguishing gap is bounded by negl(λ). This concludes the lemma.

*Construction of **iO** for General Algorithms.* Using the lemma from [CLTV14], we now prove Proposition 5.

Given $2^{-\lambda^\varepsilon}$-indistinguishable **iO** $i\mathcal{O}^C$ and garbling scheme $\mathcal{GS}$, let $pi\mathcal{O}$ be the obfuscator for probabilistic circuits from $i\mathcal{O}^C$ as described above. Let RE be the following "randomized encoding" algorithm:

$$(\hat{AL}, \hat{x}) \xleftarrow{\$} \mathsf{RE}(1^\lambda, AL, x), \text{ where } (\hat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^{\lambda'}, AL, x),\ \hat{x} = \mathsf{Encode}(\mathbf{key}, x),\ \lambda' = (\lambda + AL.n)^{1/\varepsilon}$$

Note that it is due to the monotonically increasing property of the algorithm class that we can invoke Garb with a bigger security parameter λ' for $AL \in \mathcal{AL}_\lambda$. Then our **iO** for the general algorithm class proceeds as,

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

It follows from the correctness of $\mathcal{GS}$ and $i\mathcal{O}^C$ underlying $pi\mathcal{O}$ that $i\mathcal{O}^A$ has correctness.

Security. Next, we argue about the security of $i\mathcal{O}^A$ by contra-position. Fix a polynomial T, a *non-uniform* PPT samplable distribution $\mathcal{D}$ over the support $\left\{\mathcal{AL}^T_\lambda \times \mathcal{AL}^T_\lambda \times \{0,1\}^{\text{poly}(\lambda)}\right\}$, such that, with overwhelming probability, $(AL_1, AL_2, z) \leftarrow \mathcal{D}(1^\lambda)$ satisfies that AL_1 and AL_2 are functionally equivalent and has matching parameters. Suppose that $i\mathcal{O}^A$ is insecure, that is, there is a non-uniform PPT $\mathcal{A}$ that distinguishes the following ensembles with inverse polynomial probability $1/p(\lambda)$.

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

Since $\mathcal{D}$ satisfies that with overwhelming probability, AL_1, AL_2 sampled from it have the same functionality and matching parameters, there exists one such sequence $\{(AL_{1,\lambda}, AL_{2,\lambda}, z_\lambda)\}$ w.r.t. which $\mathcal{A}$ distinguishes obfuscation of $AL_{1,\lambda}$ and $AL_{2,\lambda}$ given z_λ with probability $1/2p(\lambda)$. By construction of $i\mathcal{O}^A$, this implies that $\mathcal{A}$ distinguishes the following ensembles with probability $1/2p(\lambda)$.

$$\left\{(pi\mathcal{O}(1^\lambda, \mathsf{RE}(1^\lambda, AL_{1,\lambda}, \cdot)), z_\lambda)\right\}_\lambda \tag{7}$$
$$\left\{(pi\mathcal{O}(1^\lambda, \mathsf{RE}(1^\lambda, AL_{2,\lambda}, \cdot)), z_\lambda)\right\}_\lambda \tag{8}$$

We show this $\mathcal{A}$ contradicts with Lemma 1. It follows from $2^{-\lambda^{\varepsilon}}$-indistinguishability of $\mathcal{GS}$ and the fact that algorithm RE invokes the garbling algorithm with security parameter $\lambda' = (\lambda+n)^{1/\varepsilon}$, for every $\lambda \in \mathbb{N}$, and $x_\lambda \in \{0,1\}^{AL_1.n}$, the following ensembles are $\text{negl}(\lambda)2^{-n}$-indistinguishable.

$$\left\{C_1, C_2, x_\lambda, \mathsf{RE}(1^\lambda, AL_{1,\lambda}, x_\lambda)), z_\lambda)\right\}_\lambda$$
$$\left\{C_1, C_2, x_\lambda, \mathsf{RE}(1^\lambda, AL_{2,\lambda}, x_\lambda)), z_\lambda)\right\}_\lambda$$

where $C_b = \mathsf{RE}(1^\lambda, AL_{b,\lambda}, \cdot)$. Thus it follows from Lemma 1 that ensembles (7) and (8) are indistinguishable. This gives a contradiction with the fact that $\mathcal{A}$ distinguishes them and hence, the security of $i\mathcal{O}^A$ holds.

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 polynomial time $p_{pIO}(\lambda', |C|)$ where the polynomial p_{pIO} depends on the running time of the underlying **iO** and PRF; moreover, if the underlying **iO** has linear-time-dependent complexity, p_{pIO} also depends quasi-linearly in $|C|$. Let $p_{\mathsf{RE}}(\lambda', |AL|, n, m, S, T)$ be the running time of $\mathsf{RE}(1^\lambda, AL, x)$ (depending on the efficiency of $\mathcal{GS}$, the polynomial p_{RE} depends on a subset of the parameters). Overall, the running time of $i\mathcal{O}^A(1^\lambda, AL)$ is

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

Therefore,

- If $\mathcal{GS}$ has optimal efficiency (that is, p_{RE} depends only on m) or I/O-dependent complexity (that is, p_{RE} does not depend on S, T), $i\mathcal{O}^A$ has I/O-dependent complexity.
- If $\mathcal{GS}$ has space-dependent complexity (that is, p_{RE} depend on T), so does $i\mathcal{O}^A$.
- If $\mathcal{GS}$ and the underlying **iO** has linear-time-dependent complexity (that is, p_{RE} depends quasi-linearly on T and so does p_{pIO} on $|C|$), so does $i\mathcal{O}^A$.

Finally, we note that combining the proposition 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 9. *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.*

Regarding Evaluation Efficiency. Evaluating the **iO** for TM and RAM obtained in Theorem 9 on input x, involves evaluating the obfuscated program on x once to obtain a garbled pair $(\hat{AL}, \hat{x})$, and then evaluate them. Therefore, overall, evaluation takes time $T_{AL}(x) \times \text{poly}(\lambda, |AL|, AL.S)$. When the space is large, the overhead on computation time is large. We can then improve the evaluation efficiency (at the price of losing succinctness of the garbled RAM) by combining proposition with the construction of garbling RAM by [LO13, GHL+14]. Since their garbled RAM has size and evaluation time quasi-linear in the time complexity (i.e., $\text{poly}(\lambda, |AL|, AL.m)T$), we can obtain the following:

Theorem 10. *Assume a sub-exponentially indistinguishable **iO** for circuits and sub-exponentially secure OWF. There is an indistinguishability obfuscator for* RAM *with input and output lengths bounded by a-priori fixed polynomials, and the indistinguishability obfuscator has linear-time dependent complexity.*

We note that when combining the theorem with the garbling scheme for RAM of [LO13, GHL$^+$14] in a straightforward way, it actually yields only an **iO** for RAM with complexity (in terms of both size and evaluation time) polynomial in the time complexity T of the RAM machine being obfuscated. However, we can apply the same trick as described in [GHRW14] to improve the efficiency of the **iO** for RAM to depend only quasi-linearly on T. In the work of [GHRW14], they noticed that in the construction of [LO13, GHL$^+$14], each bit in a garbled RAM and an input encoding can be generated using a small circuit of size $\tilde{O}(|R| + n + m) \times \text{poly}(\lambda, \log T)$, where R is the RAM machine under consideration, $n = R.n$, $m = R.m$ and $T = R.T$. Thus, to achieve linear-time-dependent complexity, we can simply modify our construction above as follows. Instead of directly obfuscating the whole randomized encryption algorithm RE, which has size $S = \tilde{O}(|R| + n + m) \times \text{poly}(\lambda, \log T) \times T$ and leads to a poly(S) size obfuscated circuit, do the following: Decompose RE into a set of S small circuits $\{\mathsf{RE}_i\}$ each of which computes the i^{th} bit in the randomized encoding, and then obfuscate each RE_i separately. Since each RE_i is small, the final obfuscation only depends on T quasi-linearly; more precisely, the complexity is $\text{poly}(\lambda, |R|, n, m, \log T) \times T$.

6.2 From iO to Garbling

We here show how to transform **iO** for a class C into garbling scheme for the same class with the same efficiency.

Proposition 6. *Let $\{\mathcal{AL}_\lambda\}$ be any class of deterministic algorithms. It holds that if there is an **iO** $i\mathcal{O}$ for $\{\mathcal{AL}_\lambda\}$, then there is a garbling scheme $\mathcal{GS}$ for this class. Furthermore, the following efficiency preservation holds.*

- *if $i\mathcal{O}$ has optimal efficiency or input- / I/O- dependent complexity, $\mathcal{GS}$ has I/O-dependent complexity.*
- *If $i\mathcal{O}$ has space- / linear-time- dependent complexity, so does $\mathcal{GS}$.*

We observe that combining the construction of **iO** for Turing machines with input-dependent complexity by [BCP14, ABG$^+$13] with the theorem, we obtain a Garbling scheme for Turing machine with only I/O dependent complexity.

Corollary 1. *Assume the existence of **iO** for Turing machines with input-dependent complexity. There is a garbling scheme for Turing machines with I/O-dependent complexity.*

Proof of Proposition 6. The construction is quite straight-forward and illustrates the difference between obfuscation and garbling schemes. The key generation is simply the key generation algorithm for Lamport's one-time signature scheme [Lam79] with the tweak that instead of using a one-way function (as in Lamport's construction), we use a length doubling PRG—the public key for signing messages of length n consist of $2n$ images $\{c_i^b = f(r_i^b)\}_{i\in[n],b\in\{0,1\}}$ of a PRG f, and the secret key is the set of pre-images $\{r_i^b\}_{i\in[n],b\in\{0,1\}}$; both the public and secret keys are output as part of the garbling key. The encoding of an input x is a signature of x (i.e., $(r_1^{x_1}, \ldots, r_n^{x_n})$, and the encoding of an algorithm AL is the obfuscation of a program $\Pi[AL]$ that on input n (presumed) pre-images $\vec{r}$ determines whether there exist an input x such that $f(r_i) = c_i^{x_i}$ and if so runs $AL(x)$; otherwise it simply outputs $\perp$; it is important that the computation performed to determine the input x takes a number of steps t_0 that is independent of the input.

To show that this construction is a secure garbling we need to exhibit a simulator that given just the output $y = AL(x)$ of the program AL on input x and the number of steps T^* taken by $AL(x)$ (as well as other parameters (n, m, S, T) of AL) can simulate the encoded input and program.

To simulate the encoded input, we simply output n random pre-images $\vec{r}$, and to simulate the the encoded program, we simply obfuscate the program $\widetilde{\Pi}$ that on input $\vec{r}$ outputs y while taking T^* steps, and on any other input outputs $\perp$ while taking t_0 steps. (The bound parameters of $\widetilde{\Pi}$ are set to (n, m, S, T).)

To show indistinguishability of the simulation, we consider a hybrid experiment which proceeds just as the real one expect that the key generation algorithm is modified so that (with overwhelming probability) there only exists a signature for the message x—we simply let $c_i^{1-x_i}$ be chosen as a uniform random string (instead of picking it in the image of f. By the security of the PRG this experiment is indistinguishable from the real one; furthermore (with overwhelming probability) the program being obfuscated in this experiment is functionally equivalent and has the same running-time as $\widetilde{\Pi}$ for all inputs, and thus by security of the TM indistinguishability obfuscator this experiment is indistinguishable from the simulated one. □

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. 2013.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *FOCS*, pages 166–175, 2004.

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

[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.

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EuroCrypt'14*, 2013.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[BP13] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. 2013.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BSW12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

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

[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.

[CLTV14] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic functionalities and applications. Manuscript, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology–EUROCRYPT 2013*, pages 1–17. Springer, 2013.

[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.

[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 *CRYPTO*, pages 465–482, 2010.

[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.

[GHRW14] Gentry, Halevi, Raykova, and Wichs. Outsourcing private ram computation. *Proc. of FOCS 2014*, 2014.

[GJKS] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities.

[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 *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564, 2013.

[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.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

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

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. SRI Technical Report, 1979.

[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.

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

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO'14*, 2014.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *Proc. of STOC 2014*, 2014.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.