Indistinguishability Obfuscation for RAM Programs and Succinct Randomized Encodings

SIAM J. COMPUT.
Vol. 47, No. 3, pp. 1123–1210

INDISTINGUISHABILITY OBFUSCATION FOR RAM PROGRAMS AND SUCCINCT RANDOMIZED ENCODINGS*

NIR BITANSKY†, RAN CANETTI‡, SANJAM GARG§, JUSTIN HOLMGREN¶, ABHISHEK JAIN‖, HUIJIA LIN#, RAFAEL PASS††, SIDHARTH TELANG††, AND VINOD VAIKUNTANATHAN¶

Abstract. We show how to construct indistinguishability obfuscation (**iO**) for RAM programs with bounded space, assuming **iO** for circuits and one-way functions, both with subexponential security. That is, given a RAM program whose computation requires space $s(n)$ in the worst case for inputs of length at most n, we generate an obfuscated RAM program that, for inputs of size at most n, runs in roughly the same time as the original program, using space roughly $s(n)$. The obfuscation process is quasi-linear in the description length of the input program and $s(n)$. At the heart of our construction are succinct randomized encodings for RAM programs. We present two very different constructions of such encodings, each with its own unique properties. Beyond their use as a tool in obfuscation for RAM programs, we show that succinct randomized encodings are interesting objects in their own right. We demonstrate the power of succinct randomized encodings in applications such as publicly verifiable delegation, functional encryption for RAMs, and key-dependent security amplification.

Key words. cryptography, randomized encodings, obfuscation, bootstrapping

AMS subject classifications. 68P25, 68Q99

DOI. 10.1137/15M1050963

*Received by the editors December 2, 2015; accepted for publication (in revised form) March 12, 2018; published electronically June 26, 2018. This paper is based on results from two works, [BGL+15] and [CHJV15], preliminary versions of which appeared in the proceedings of ACM STOC 2015. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

http://www.siam.org/journals/sicomp/47-3/M105096.html

Funding: The work of the first author was supported by the Alon Young Faculty Fellowship and by Len Blavatnik and the Blavatnik Family foundation. The work of the second author was supported by the Check Point Institute for Information Security, ISF grant 1523/14, and NSF Frontier grants CNS1413920 and 1218461. The work of the fourth and ninth authors was supported by NSF grants CNS-1350619 and CNS-1414119, a DARPA Safeware grant, a Alfred P. Sloan Research Fellowship, a Microsoft Faculty Fellowship, the NEC Corporation, and a Steven and Renee Finn Career Development Chair from MIT. The work of the sixth author was supported in part by NSF grants CNS-1528178 and CNS-1514526. The work of the seventh and eighth authors was supported in part by an Alfred P. Sloan Fellowship, a Microsoft New Faculty Fellowship, NSF award CNS-1217821, NSF CAREER award CCF-0746990, NSF award CCF-1214844, AFOSR YIP award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211.

†School of Computer Science, Tel Aviv University, Tel Aviv, 69978 Israel (nirbitan@tau.ac.il). Member of the Check Point Institute of Information Security. Part of this research was done while at IBM.

‡Department of Computer Science, Boston University, Boston, MA 02215, and School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978 Israel (canetti@bu.edu).

§Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720 (sanjamg@berkeley.edu).

¶Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02142 (holmgren@csail.mit.edu, vinodv@csail.mit.edu).

‖Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218 (abhishek@cs.jhu.edu).

#Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106 (rachel.lin@cs.ucsb.edu).

††Department of Computer Science, Cornell University, Ithaca, NY 14853 (rafael@cs.cornell.edu, sidtelang@cs.cornell.edu).

1. Introduction. The ability to cryptographically obfuscate general programs holds great prospects for securing the future digital world [BGI+01]. However, general-purpose obfuscation mechanisms prior to this work were inherently inefficient. One of the main sources of inefficiency is the fact that existing solutions consider different models of computation from those used to write modern computer programs. Indeed, the first candidate indistinguishability obfuscator (**iO**) of Garg et al. [GGH+13] and most general-purpose obfuscators that followed shortly after were designed for Boolean circuits. Using a circuit obfuscator for different models of computation such as *Turing machines* or *random access machines* (RAMs) requires first translating the given program into a circuit. This comes with clear loss in efficiency aspects: the succinct description of the program (as Turing machine or RAM) is not preserved and now becomes proportional to its running time on the worst-case input. Similarly, executing the program on any specific input now requires the same time and space resources as the worst-case input, while the *input-specific* resources required by the original program could potentially be much smaller.

Assuming the existence of succinct noninteractive arguments and circuit obfuscators that satisfy a notion of obfuscation called differing-input obfuscation, Boyle, Chung, and Pass [BCP14] and Ananth et al. [AJS17] show how to construct Turing machine obfuscators—namely, obfuscators that take a Turing machine as input and produce an obfuscated Turing machine whose runtime and space requirements per input, as well as their size, are proportional to those of the input machine (up to a polynomial factor in the security parameter). However, differing-input obfuscation is a significantly stronger security requirement that **iO**; in fact some strong variants of differing-input obfuscation are believed to be impossible [GGHW13].

Furthermore, while preferable to circuits in terms of succinct representation and input-specific usage of resources, Turing machines still do not achieve the full gains of RAM programs, which for some tasks may be much faster due to their ability to randomly access memory. To apply any of the above obfuscators to a RAM program, one has to first translate the program to a circuit or a Turing machine, which loses these advantages.

1.1. This work: iO for RAMs through succinct randomized encodings. In this work, we show how to directly obfuscate *RAM programs with bounded input and space*. That is, our obfuscator $\mathcal{O}$ takes as input a RAM program Π, input length n, and memory bound s. It outputs another RAM program $\mathcal{O}(\Pi)$ such that $\mathcal{O}(\Pi)$ has the same functionality as that of Π on all inputs x of length up to n and such that the execution of Π on x uses space at most s. The running time of $\mathcal{O}(\Pi)$ per each specific input is the same as that of Π, and its space usage is proportional to s (all up to polynomial factors in the security parameter and input bound n). The running time of the obfuscator, and in particular the size of the obfuscated $\mathcal{O}(\Pi)$, is quasi-linear in that the description Π and the space bound s.

We show security of the construction assuming one-way functions and **iO** that are subexponentially secure. These assumptions are somewhat better understood and "more straightforward" than differing-input obfuscation. In particular, it is easier to demonstrate that an obfuscator is not subexponentially secure **iO** (simply demonstrate a pair of programs along with subexponential-time distinguisher) than it is to demonstrate that it is not a differing-input obfuscator.

The central piece of our construction and conceptual heart of this paper is the notion of a *succinct randomized encoding*. We then extend these schemes to full-fledged obfuscators in a relatively straightforward way. We thus start by reviewing

randomized encodings and its history next.

Garbling and randomized encodings. Garbling [Yao86, Rog91] and randomized encoding [IK00] schemes are a pillar of cryptographic protocol design, with numerous applications such as secure two-party and multiparty computation, verifiable delegation schemes, and one-time programs, to name a few (see [Rog91, App11b] for a survey).

Such schemes partition the process of evaluating a function f on input x into two stages: a randomized *encoding* (or, *garbling*) stage, which results in an encoded version $\widehat{f(x)}$ of the computation, and a *decoding stage*, where the output $f(x)$ can be recovered from the encoding $\widehat{f(x)}$. The key property shared by all such schemes is that the encoding $\widehat{f(x)}$ should hide all information regarding the computation except for the output $f(x)$. This *privacy* requirement is naturally captured by an efficient simulator $\mathsf{Sim}(f(x))$, which, given only the output $f(x)$, produces a simulated encoding (perfectly, statistically, or computationally) indistinguishable from $\widehat{f}(x)$.

Whereas privacy alone could be trivially achieved by having the encoding be $f(x)$, garbling and randomized encoding schemes feature additional properties that (in conjunction with privacy) turn them into a powerful tool. These additional properties may vary. The seminal work of Yao on *garbled circuits* [Yao86] leverages the ability to encode (f, x) with low input-locality (each output bit of the encoding depends on at most one bit of x). Another property of Yao's solution often used in applications is that encoding the input can be done separately from encoding the function and independently of the function's complexity (this is sometimes known as separability or decomposability).[1] The work of Ishai, Kushilevitz, and Applebaum on *randomized encodings* [IK00, IK02, AIK04, AIK06] asks more generally, how much simpler can the encoding procedure be compared to direct computation of $f(x)$?

Capturing what it means to "simplify the computation of $f(x)$" may take different forms according to the complexity measure of interest. The work of Ishai, Kushilevitz, and Applebaum focuses on computing the randomized encoding $\widehat{f(x)}$ with lower parallel-time complexity than required for computing the original function f. This line of work has been extremely fruitful both in terms of constructions and applications (again, see [App11b] for a survey).

Succinct randomized encodings. In this work, we address another natural complexity measure: *the time required to compute* $\widehat{f(x)}$. Specifically, we require that, given the description of f and the input x, we can compute the encoding $\widehat{f(x)}$ in time $\widehat{T}$ that is significantly smaller than the time T required to compute $f(x)$. Decoding time, in contrast, would be as large as T, with some tolerable overhead. For this goal to be achievable, f has to be given in some succinct representation that is smaller than T, and cannot be given by, say, a size-T circuit. With the goal of obtaining obfuscation for RAM machines, we shall focus on the case that f is represented by a RAM machine Π. As a step toward this, we shall also consider the simpler case of Turing machines, which will already capture much of the challenge.

We will also be interested in the complexity of the decoding process. Ideally, this decoding should not be much more complicated than the original, nonencoded computation. In particular, decoding the result of a given RAM computation should

[1] Accordingly, the term "garbling" is usually identified with schemes that have such separate input encoding, whereas "randomized encodings" refer to schemes where the input is encoded together with the function.

amount to executing this computation (in the RAM model).

The existence of succinct randomized encodings. Under commonly believed complexity-theoretic assumptions, perfectly private randomized encodings for all of **P** are unlikely to be computable in fixed polynomial time.[2] In contrast, restricting our attention to privacy against computationally bounded adversaries, no lower bounds or barriers are known. Accordingly, we will restrict our attention to computational privacy—a simulated encoding $\mathsf{Sim}(f(x))$ is computationally indistinguishable from a real encoding $\widehat{f(x)}$.

We note that **iO** for RAMs (respectively, Turing machines) would directly imply corresponding succinct randomized encodings.

Our contributions. We first show that subexponentially secure succinct randomized encodings for RAMs imply **iO** for RAMs with similar complexity parameters.

INFORMAL THEOREM 1.1 (**iO** for RAMs from succinct randomized encodings). *Assume there exist one-way functions, (nonsuccinct) **iO** for* $\mathsf{P/poly}$*, and succinct randomized encodings for RAM programs, all subexponentially secure. Then there exists **iO** for RAM programs with inputs bounded by* n*. The time to obfuscate a program* Π *and the size of the obfuscated program are proportional to the time of encoding* (Π, x) *for inputs* x *of length* n*. The time and space requirements of the obfuscated program, per input, are proportional to those of the decoding process.*

The theorem is in a sense the succinct analog of previous bootstrapping theorems [App14, CLTV15] that show how (nonsuccinct) randomized encodings and pseudorandom functions in NC^1, together with obfuscation for NC^1 circuits, imply obfuscation for $\mathsf{P/poly}$. Here, through succinct randomized encodings, we reduce **iO** for arbitrarily long computations to **iO** for circuits of fixed polynomial size. Indeed, the proof of this theorem follows closely ideas from there. We note that this theorem is rather general; in particular it is used by [KLW15, CH16, CCC+16] to obtain fully succinct **iO** for Turing and RAM machines.

Our core technical contribution consists of two succinct randomized encoding schemes for RAM programs. Our schemes rely on (nonsuccinct) **iO** for *circuits* and incur encoding overhead that is polynomial in $s(n)$, the worst-case space requirement of the input machine, but is essentially independent of its time complexity. Decoding amounts to a RAM computation with the same complexity as the original computation, up to polynomial factors in the security parameter.

INFORMAL THEOREM 1.2 (succinct randomized encodings from circuit **iO**). *Assume the existence of **iO** for* $\mathsf{P/poly}$ *and one-way functions. Then there exists a succinct randomized encoding for all polynomial-time RAM* Π *with bounded space complexity* $s(n)$*. The time to encode is quasi-linear in the size of* Π*, input and output lengths* n, m*, and the space bound* $s(n)$*. The time to decode* $\widehat{\Pi(x)}$ *is polynomial in the size of* Π*, and quasi-linear in* $s(n)$ *and the time* T *for evaluating* $\Pi(x)$*.*

[2]Specifically, it can be shown that, for a language $\mathcal{L}$, recognized by a given $T(n)$-time machine Π, succinct randomized encodings with perfect-privacy computable in time $t(n) \ll T(n)$ would imply that $\mathcal{L}$ has 2-message interactive proofs with an $O(t(n))$-time verifier, which already suggests that $t(n)$ should at least depend on the space (or depth) of the computation. Furthermore, under commonly believed derandomization assumptions (used to show that $\mathsf{AM} \subseteq \mathsf{NP}$ [Kv99, MV99]), the above would imply that $\mathcal{L}$ can be nondeterministically decided in time $\mathrm{poly}(t(n))$ for some fixed polynomial poly. Thus, any speedup in encoding would imply related speedup by nondeterminism, whereas significant speedup is believed to be unlikely.

We give two alternative proofs of Theorem 1.2, by way of two quite different randomized encoding schemes. The first scheme starts with a relatively simple succinct randomized encoding scheme for Turing machines, based on Yao's garbled circuit method. The idea is to avoid direct garbling of the machine, and instead obfuscate the circuit that generates the garbled machine. The evaluator then runs this circuit, obtains the garbled machine, and then runs it to obtain the desired output value. Here we use the strong locality properties of Yao's garbling scheme to make sure that the obfuscated garbling circuit is small. We then show how to extend it to RAMs based on any (nonsuccinct) RAM garbling scheme [LO13, GHL$^+$14, GLOS15]. An additional feature of this construction is the following. Assuming puncturable pseudorandom functions in NC^1 (known based on various hardness assumptions, such as the hardness of the learning with errors problem [BLMR13]), and restricting our attention to any class of computations with a-priori-bounded running time $t(n)$, it is enough to assume existence of nonsuccinct **iO** for circuits in NC^1 with input size $O(\log(t(n)))$. This appears to be a qualitatively weaker assumption than **iO** for arbitrary circuits. In particular, it is a poly$(t(n))$-time falsifiable assumption in the terminology of [Nao03].

The second scheme takes a more direct approach. Rather than obfuscating the circuit that garbles the machine, we directly obfuscate the circuit that implements the basic computational step of the machine itself. That is, we obfuscate the circuit that reads the contents of the appropriate cell on the machine's tape, writes a new value to the cell, moves the head to a new tape location, and updates the machine's state according to its transition function. To make sure that the computation remains meaningful and that the evaluator does not learn the intermediate values that the machine writes on its tape, we provide a mechanism for the obfuscated machine to encrypt and authenticate the contents of the tape. Also here we first describe the scheme for the simpler case of Turing machines. We then extend the scheme to the case of RAM machines; here we also provide a mechanism for hiding the pattern of memory accesses of the machine. This construction approach is more "low level" and more technically involved than the first one. Still its directness provides some additional power; in particular it was extended in subsequent work to obtain fully succinct randomized encoding of Turing machines, RAM machines, and RAM machines with persistent memory [KLW15, CH16, CCC$^+$16, CCHR15, ACC$^+$15] (see further details in section 4.4).

We note that both constructions also satisfy the more general promise of a garbling scheme where encodings of the input and of the program can be done separately. More detailed overviews of the two constructions can be found within.

Dependence on the output length. As stated above, the size of our basic randomized encodings (or **iO**) grows with the output of the underlying computation. Such dependence can be easily shown to be inherent as long as we require simulation-based security (using a standard incompressibility argument). Nevertheless, we show that this dependence can be removed if we settle for a weaker indistinguishability-based guarantee saying that randomized encodings of two computations leading to the same output are indistinguishable.

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

Succinct functional encryption and reusable garbling. Recent advancement in the study of obfuscation has brought with it a corresponding advancement in functional encryption (FE). Today, (indistinguishability-based) functional encryption for all circuits can be constructed from **iO** [GGH+13, Wat14], or even from concrete (and efficiently falsifiable) assumptions on composite order multilinear graded-encodings [GGHZ16]. For models of computation with succinct representations, we may hope to have *succinct FE*, where a secret key sk_Π, allowing us to decrypt $\Pi(x)$ from an encryption of x, can be computed faster than the running time of Π. However, here the state of the art was similar to succinct randomized encodings, or succinct **iO**, requiring essentially the same strong (nonfalsifiable) assumptions.

One can replace **iO** for circuits in the above FE constructions, with the succinct **iO** from Theorem 1.1, and obtain FE where computing sk_Π is comparable to (succinctly) obfuscating Π. This, however, will require the same subexponential hardness of **iO** for circuits. Based on existing nonsuccinct functional encryption schemes, we show that succinct FE can be constructed without relying on subexponentially hard primitives.

As observed in previous work [GHRW14, CIJ+13, GKP+13], FE (even indistinguishability based) directly implies an enhanced version of randomized encodings known as *reusable garbling*. Here reusability means that an encoding consists of two parts: The first part, $\widehat{\Pi}$, is independent of any specific input and depends only on the machine Π. $\widehat{\Pi}$ can then be "reused" together with the second part, $\widehat{x}$, encoding any input x. We get succinct reusable garbling for space-bounded computations: encoding Π depends on the space, but is done once; subsequent input encodings depend only on the input size n and not on space.

Publicly verifiable delegation and succinct NIZKs. Succinct randomized encodings directly imply a one-round delegation scheme for polynomial-time computations with bounded space complexity. A main feature of the scheme is *public-verifiability*, meaning that given the verifier's message σ anyone can verify the proof π from the prover, without requiring any secret verification state. Previous schemes based on standard assumptions could only achieve this goal in an amortized sense (allowing an expensive preprocessing step) via reusable garbled circuits or RAMs [GKP+13, GHRW14]. Other schemes relied on strong knowledge assumptions [BCC+17, BCCT12, DFH12, BCCT13] or proven secure only in the random oracle model [Mic00].[3] Another prominent feature of the scheme is that it guarantees input privacy for the verifier. While this can generically be guaranteed with fully homomorphic encryption, the generic solution requires the prover to convert the computation into a circuit, which could incur quadratic blowup; in our solution, the complexity of the prover corresponds to decoding complexity, which could be made quasi-linear. See further discussion below.

The delegation scheme is based only on randomized encodings (and one-way functions) and thus, as explained above, can be based only on polynomial assumptions. Assuming also **iO**, we can make the verifier's message reusable; namely, the verifier can publish his message σ once and for all, and then get noninteractive proofs for multiple computations. Our transformation for reusing the verifier's message is, in fact, a generic one that can be applied to any delegation scheme, including privately verifiable schemes (e.g., [KRR14]). For privately verifiable schemes, the transformation has an additional advantage: it removes what is known as the *verifier rejection problem*; specifically, in the transformed scheme, soundness holds even against provers

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

with a verification oracle.

The delegation schemes mentioned above are restricted to **P**. Plugging our succinct **iO** into the perfect noninteractive zero-knowledge (NIZK) arguments of Sahai and Waters [SW14] directly yields a construction of perfect succinct NIZK for bounded space NP from **iO** for P/poly and one-way functions that are both sub-exponentially secure. The NIZK has a succinct common reference string whose size is independent of the time required to verify the NP statement to be proven, and depends only on the space and on the size of the input and witness (verification time depends only on the length of the statement, as in [SW14]).

Other applications. We reexamine previous applications of (nonsuccinct) randomized encodings and note the resulting succinctness features.

One application is to multiparty computation [IK00, IK02], where we can reduce the communication overhead from depending on the circuit size required to compute a multiparty function $f(x_1, \ldots, x_m)$ to depending on the space required to compute f, which can be much smaller. (In this solution, the parties simply compute $\widehat{f}$ instead of f itself.) By now, beyond its conceptual simplicity, this solution does not give any advantage over recent one-round solutions based on multikey fully homomorphic encryption [Gen09, AJL$^+$12, LTV12, GGHR14, MW16, DHRW16].

Another application is the amplification of *key-dependent message* (KDM) security. In KDM encryption schemes, semantic security needs to hold, even when the adversary obtains encryptions of functions of the secret key taken from a certain class $\mathcal{F}$. Applebaum [App11a] shows that any scheme that is KDM secure with respect to some class of functions $\mathcal{F}$ can be made resilient to a bigger class $\mathcal{F}' \supseteq \mathcal{F}$ if functions in $\mathcal{F}'$ can be randomly encoded in $\mathcal{F}$. Our succinct randomized encodings will essentially imply that KDM security for circuits of any fixed polynomial size $s(\cdot)$ (such as the scheme of [BHHI10]) can be amplified to KDM security for functions that can be computed by programs with space $S \ll s(n)$, but could potentially have larger running time.

Obfuscation with quasi-polynomial blowup and a new bootstrapping theorem. As an application of our technical approach (rather than a direct application of our succinct randomized encodings), we show a new bootstrapping theorem for **iO** for circuits. This transformation shows that obfuscation for an arbitrary circuit C can be reduced to obfuscation of $O(|C|)$ circuits whose size depends only on the security parameter and input length (the circuits themselves are also simple and are dominated by a constant number of applications of a pseudorandom function). In particular, the size of the resulting obfuscated C is $\widetilde{O}(|C|)$.

1.3. Subsequent work. In a beautiful subsequent work, Koppula, Lewko, and Waters [KLW15] construct fully succinct randomized encodings for Turing machines from **iO**. They overcome the "space barrier" by introducing a clever "selective enforcement mechanism." This was extended by Canetti and Holmgren [CH16] and independently by Chen et al. [CCC$^+$16] to produce a fully succinct randomized encoding for RAM machines. Canetti et al. [CCHR15] and independently Ananth et al. [ACC$^+$15] subsequently extended this scheme to provide security even when RAM machines to be garbled are chosen adaptively after an initial database is encoded. Ananth, Jain, and Sahai [AJS15] gave a direct construction of succinct **iO** for Turing machines that avoids invoking Theorem 1.1 and achieves a constant multiplicative size overhead.

1.4. Road map. Most of the remainder of the paper is dedicated to our constructions of succinct randomized encodings, covered in sections 3 and 4. More detailed technical overviews of the approaches can be found in the beginning of the respective sections. Section 5 covers the different applications of succinct randomized encodings. The required background and preliminaries are given in section 2.

2. Preliminaries. Let $\mathbb{N}$ denote the set of positive integers, and let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We write PPT as shorthand for probabilistic polynomial time Turing machines. The term *negligible* is used for denoting functions that are (asymptotically) smaller than any inverse polynomial. More precisely, a function $\nu : \mathbb{N} \to \mathbb{R}$ 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 $\mathsf{CIR}[D] = \{\mathcal{C}_\lambda\}$ includes all deterministic circuits of size at most $D(\lambda)$.

NC^1 circuits. For every constant c and polynomial D, the class $\mathsf{NC}^1_c[D] = \{\mathcal{C}_\lambda\}$ of polynomial-sized circuits of depth $c \log \lambda$ includes 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 $M = (M', n, m, S, T)$, where the bit length of n, m, S, T is λ and $n, m \leq S \leq T$; M takes input x of length n, 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 if M' requires more than S space, then 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 by $(M.n, M.m, M.S, M.T)$.

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, 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, we say that two Turing machines M_1 and M_2 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 RAMs. 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 replaced 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. Letting $\mathcal{X}[D]$ be any class defined above, we denote by $\mathsf{r}\mathcal{X}[D]$ the corresponding class of randomized algorithms. For example, $\mathsf{rCIR}[D]$ denotes all randomized circuits of size $D(\lambda)$, and $\mathsf{rTM}[D]$ denotes all randomized Turing machines 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. Letting $\mathcal{X}[D]$ 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 shorthand 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.
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 of the above-defined algorithm classes are well formed. Below, we denote by $T_{AL}(x)$ the running time of AL on input x, and by 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 schemes. In this section, we define garbling schemes, following in most part the definitions in [BHR12b]. As explained in the introduction, the main difference between garbling schemes and randomized encodings is that garbling schemes allow the input to be encoded separately from the program. This slight strengthening is not needed in our applications, which are described in section 5.

DEFINITION 2.1 (garbling scheme). *A garbling scheme for a class of (well-formed) deterministic algorithms* $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$ *is a tuple* $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ *satisfying the following properties:*

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

- Garb *is probabilistic and on input* $(1^\lambda, AL)$ *outputs a pair* $(\widehat{AL}, \mathbf{key})$;[4]
- Encode *is deterministic and on input* $(\mathbf{key}, x)$ *outputs* $\hat{x}$;
- Eval *is deterministic and on input* $(\widehat{AL}, \hat{x})$ *produced by* $\mathsf{Garb}, \mathsf{Encode}$ *outputs* y.

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

$$\Pr[\ \mathsf{Eval}(\widehat{AL}, \hat{x}) \neq AL_\lambda(x_\lambda)] \leq \mu(\lambda)$$

in the probability space defined by sampling

$$(\widehat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL_\lambda),$$
$$\hat{x} \xleftarrow{\$} \mathsf{Encode}(\mathbf{key}, x_\lambda).$$

DEFINITION 2.2 (security of a garbling scheme). *We say that a garbling scheme* $\mathcal{GS}$ *for a class of deterministic algorithms* $\{\mathcal{AL}_\lambda\}_{\lambda\in\mathbb{N}}$ *is secure if the following holds.*

There is a uniform machine Sim *such that for every nonuniform* PPT *distinguisher* $\mathcal{D}$, *polynomial* T', *sequence of algorithms* $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda^{T'}\}$, *and sequence* $\{x_\lambda\}$ *of inputs where* $x_\lambda \in \{0,1\}^{(AL_\lambda).n}$, *there exists a negligible function* μ, *such that, for every* $\lambda \in \mathbb{N}$ *the following holds:*

$$\left|\Pr[\mathcal{D}(\widehat{AL}, \hat{x}) = 1] - \Pr[\mathcal{D}(\widetilde{AL}, \tilde{x}) = 1]\right| \leq \mu(\lambda)$$

in the probability space defined by sampling

$$(\widehat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL_\lambda),$$
$$\hat{x} \xleftarrow{\$} \mathsf{Encode}(\mathbf{key}, x),$$
$$(\widetilde{AL}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, |x|, |AL_\lambda|, (n, m, S, T), T_{AL_\lambda}(x), AL_\lambda(x)),$$

where n, m, S, *and* T *are, respectively, shorthand for* $AL.n$, $AL.m$, $AL.S$, *and* $AL.T$, *and* Sim *runs in time* $\mathrm{poly}(\lambda, T)$. *The function* 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* subexponentially indistinguishable *if* $\mu(\lambda)$ *is bounded by* $2^{-\lambda^\varepsilon}$ *for a constant* ε.

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

We remark that in the above definition simulator Sim receives many inputs, meaning that a garbled pair $\widehat{AL}, \hat{x}$ reveals nothing but the following: the output $AL(x)$, instance running time $T_{AL}(x)$, input length $|x|$, and machine size $|AL|$, together with various parameters (n, m, S, T) of AL. We note that the leakage of the instance running time is necessary in order to achieve instance-based efficiency (see efficiency guarantees below). The leakage of $|AL|$ can be avoided by padding machines if an upper bound on their size is known. The leakage of parameters (n, m, S, T) can be avoided by setting them to 2^λ; see Remark 2.14 for more details. In particular, when the algorithms are circuits, inputs to the simulation algorithm can be simplified to $(1^\lambda, |x|, |C|, AL(x))$, since all bound parameters n, m, S, T can be set to 2^λ.

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

Efficiency guarantees. We proceed to describe the efficiency requirements for garbling schemes. When considering only circuit classes, all algorithms Garb, Encode, Eval should be polynomial-time machines; that is, the complexity of $\mathsf{Garb}, \mathsf{Eval}$ 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 2.3 (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 exist universal polynomials p_{Garb}, p_{Encode}, and p_{Eval} such that for every $\lambda \in \mathbb{N}$, $AL \in \mathcal{AL}_\lambda$, and input $x \in \{0,1\}^{AL.n}$*

- $(\widehat{AL}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, AL)$ *runs in time $p_{\mathsf{Garb}}(\lambda, |AL|, AL.m)$;*[5]
- $\hat{x} = \mathsf{Encode}(\mathbf{key}, x)$ *runs in time $p_{\mathsf{Encode}}(\lambda, |x|, AL.m)$; and*
- $y = \mathsf{Eval}(\widehat{AL}, \hat{x})$ *runs in time $p_{\mathsf{Eval}}(\lambda, |AL|, |x|, AL.m) \times T_{AL}(x)$, with overwhelming probability over the random coins of* Garb. *We note that* Eval *has instance-based efficiency.*

I/O-DEPENDENT COMPLEXITY: *A relaxed version of optimal efficiency, in which the polynomials p_{Garb}, p_{Encode}, and p_{Eval} are additionally functions of $AL.n$ and $AL.m$.*

SPACE-DEPENDENT COMPLEXITY: *A relaxed version of optimal efficiency, in which the polynomials p_{Garb}, p_{Encode}, and p_{Eval} are additionally functions of $AL.S$.*

LINEAR-TIME-DEPENDENT COMPLEXITY: *A relaxed version of optimal efficiency, in which the polynomials p_{Garb} and p_{Encode} additionally depend (quasi-)linearly on $AL.T$, and the running time of* Eval *is bounded by $AL.T \cdot p_{\mathsf{Eval}}(\lambda, |AL|, |x|)$.*

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

We note that the above four succinctness requirements are ordered from the strongest one (optimal) to the weakest one (time-dependent). We say that a garbling scheme is *succinct* if its complexity depends only polylogarithmically on the time bound. Thus a scheme with space-dependent complexity is succinct for a class of algorithms whose space usage is bounded by a fixed polynomial.

On output dependence. Note that in the optimal efficiency defined above, the complexity of the algorithms depends on the length of their respective inputs and the bound on their output lengths $AL.m$. We argue that this is necessary as long as we require simulation-based security. This follows from a standard incompressibility argument. Indeed, assume the existence of a pseudorandom generator G, and consider the encoding of G and a random input seed s. We claim that the size of the garbled function $\widehat{G}$ or encoded input $\widehat{s}$ must be as large as the output $|G(s)|$. Otherwise, the efficient simulator can "compress" random strings, as it cannot distinguish the actual output $G(s)$ from a uniformly random string.

The dependence on the output size can be eliminated if we settle for a weaker notion of indistinguishability security, meaning that the garbling of two (equal-length)

[5]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 $AL.T$.

program-input pairs $(\widehat{AL}_0, \widehat{x}_0), (\widehat{AL}_1, \widehat{x}_1)$ are computationally indistinguishable, provided that $AL_0(x_0)$ and $AL_1(x_1)$ output the same result after a similar number of steps. In section 5.2, we show how this can be achieved assuming **iO**.

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

We leave open the question of constructing succinct adaptively secure garbling schemes.

Garbling schemes for specific algorithm classes. Next we instantiate the above general definition of a garbling scheme with concrete classes.

DEFINITION 2.4 (garbling scheme for polynomial-sized circuits). *A triplet of algorithms* $\mathcal{GS}_{\mathsf{CIR}} = (\mathsf{Garb}_{\mathsf{CIR}}, \mathsf{Encode}_{\mathsf{CIR}}, \mathsf{Eval}_{\mathsf{CIR}})$ *is a garbling scheme (with linear-time-dependent complexity) for polynomial-sized circuits if it is a garbling scheme for class* CIR *(with linear-time-dependent complexity).*

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

DEFINITION 2.5 (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 a 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 I/O-*independent* complexity is impossible; in particular, the complexity of the scheme must scale with the bound on the output length.

DEFINITION 2.6 (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 [LO13, GHL+14, GLOS15] provided construction of a garbling scheme for RAM machines with linear-time-dependent complexity and succinct input encodings, assuming only one-way functions.

Garbled circuits with independent input encoding. In this work, we will make use of a garbling scheme for circuits with a special structural property. In Definition 2.4, the key **key** for garbling inputs is generated depending on the circuit (by $\mathsf{Garb}(1^\lambda, C)$); the special property of a circuit garbling scheme is that the key can be generated depending only on the length of the input $1^{|x|}$ and the security parameter, which implies that the garbled inputs $\hat{x}$ can also be generated depending only on the plain input x and the security parameter λ, independently of the circuit—we call this

independent input encoding.

DEFINITION 2.7 (garbling scheme for circuits with independent input encoding). *A garbling scheme* $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ *for a deterministic circuit class* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *has* independent input encoding *if the following holds: For every* $\lambda \in \mathbb{N}$*, and every* $C \in \mathcal{C}_\lambda$

- *the algorithm* Garb *on input* $(1^\lambda, C)$ *invokes first* $\mathbf{key} \xleftarrow{\$} \mathsf{Gen}(1^\lambda, 1^{|x|})$ *and then* $\widehat{C} \xleftarrow{\$} \mathsf{Gb}(\mathbf{key}, C)$*, where* Gen *and* Gb *are all* PPT *algorithms;*
- *the security condition holds with respect to a simulator* Sim *that on input* $(1^\lambda, 1^{|x|}, 1^{|C|}, T_C(x), C(x))$ *outputs* $(\tilde{x}, \tilde{C})$ *that are obtained by sampling*

$$(\tilde{x}, \mathbf{st}) \xleftarrow{\$} \mathsf{Sim.Gen}(1^\lambda, |x|),$$
$$\tilde{C} \xleftarrow{\$} \mathsf{Sim.Gb}((1^\lambda, |x|, |C|, C(x), \mathbf{st}),$$

where $\mathsf{Sim.Gen}$ *and* $\mathsf{Sim.Gb}$ *run in times* $\mathrm{poly}(\lambda, |x|)$ *and* $\mathrm{poly}(\lambda, |C|)$*, respectively.*

It is easy to check that many known circuit garbling schemes, in particular the construction by Yao [Yao86], have independent input encoding.

PROPOSITION 2.8. *Assume the existence of one-way functions that are hard to invert in* Γ *time. Then there exists a garbling scheme* $\mathcal{GS}_{\mathsf{CIR}}$ *for polynomial-sized circuits with independent input encoding that is* $\Gamma^{-\varepsilon}$*-indistinguishable for some constant* $\varepsilon \in (0, 1)$*.*

2.3. Indistinguishability obfuscation. We recall the definition of indistinguishability obfuscation, adapting it to arbitrary classes of polynomial-time algorithms. As before, we first define the syntax, correctness and security of **iO** and then discuss different efficiency guarantees.

DEFINITION 2.9 (indistinguishability obfuscator ($i\mathcal{O}$)). *A uniform machine* $i\mathcal{O}$ *is an 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* $AL \in \mathcal{AL}_\lambda$*, and 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* nonuniform PPT *sampleable distribution ensemble* $\mathcal{D} = \{\mathcal{D}_\lambda\}$ *over* $\left\{\mathcal{AL}_\lambda^T \times \mathcal{AL}_\lambda^T \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$*, and adversary* $\mathcal{A}$*, the following hold:*

If when sampling $(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda)$*, it holds with probability* 1 *that*

$$\forall x, AL_1(x) = AL_2(x),$$
$$T_{AL_1}(x) = T_{AL_2}(x),$$

and

$$\big(|AL_1|, (AL_1).n, (AL_1).m, (AL_1).S\big) = \big(|AL_2|, (AL_2).n, (AL_2).m, (AL_2).S\big),$$

then

$$\Big|\Pr\left[\mathcal{A}(AL_1', z) = 1\right] - \Pr\left[\mathcal{A}(AL_2', z) = 1\right]\Big| \le \mu(\lambda)$$

in the probability space defined by sampling

$$(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda),$$
$$AL_1' \xleftarrow{\$} i\mathcal{O}(1^\lambda, AL_1),$$
$$AL_2' \xleftarrow{\$} i\mathcal{O}(1^\lambda, AL_2),$$

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. We say that $i\mathcal{O}$ is* subexponentially 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 and AL_2 that have the same functionality and matching bound parameters. This means that 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 must ensure that $AL.S$ or $AL.T$ are not revealed, one can simply pad them to 2^λ; then the above security definition automatically ensures privacy of that parameter. See Remark 2.14 for more details.

DEFINITION 2.10 (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 $i\mathcal{O}$ 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, this stems from the fact that indistinguishability-based security does not require "programming" outputs, which is the case in simulation-based security for garbling.

iO for specific algorithm classes. We recall the definition of **iO** for polynomial-sized circuits, NC^1 [BGI+01], and give definitions of **iO** for polynomial-time Turing machines [BCP14] and RAM machines with different efficiency guarantees.

DEFINITION 2.11 (indistinguishability obfuscator for polynomial-sized circuits and for 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 $c \in \mathbb{N}$, $i\mathcal{O}_{\mathsf{NC}^1}(c,\cdot,\cdot)$ is an indistinguishability obfuscator for NC_c with optimal efficiency.

DEFINITION 2.12 (**iO** for Turing machines). *A uniform machine $i\mathcal{O}_{\mathsf{TM}}(\cdot,\cdot)$ is an 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 [BCP14, AJS17] gave constructions of **iO** for Turing machines[6] with input-dependent complexity assuming fully homomorphic encryption

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

(FHE), differing-input obfuscation for circuits, and P-certificates [CLP13]; furthermore, the dependency on input length can be removed—leading to a scheme with optimal efficiency—if assuming succinct noninteractive argument of knowledge ($\mathcal{SNARK}$) instead of P-certificates.

DEFINITION 2.13 (**iO** for RAM machines). *A uniform machine $i\mathcal{O}_{\mathsf{TM}}(\cdot,\cdot)$ is an indistinguishability obfuscator for polynomial-time RAM machines, with optimal efficiency or linear-time-dependent complexity, if it is an indistinguishability obfuscator for the class* RAM *with the same efficiency.*

Remark 2.14 (explicit vs. implicit bound parameters). In the above definitions of garbling schemes 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. The following are some examples of such schemes:

- The garbling and **iO** schemes for TM that we construct in section 3 and subsection 5.2 (from **iO** and subexp **iO** for circuits, respectively) have complexity $\mathrm{poly}(|AL|, AL.S, \log(AL.T))$. (In particular, the size of the garbled Turing machine and obfuscated Turing machine is of this order.)
- The garbling scheme for TM (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, GLOS15] has complexity that scales polynomially in $(|AL|, AL.n, AL.m)$ and quasi-linearly in $AL.T$. This construction leads to an **iO** for RAM (from subexp **iO** for circuits) of the same complexity as in 5.2.

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 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, Definitions 2.2 and 2.9, capture this by allowing leakage of all parameters $|AL|, AL.n, AL.m, AL.S, AL.T$. However, this seems to "overshoot," since 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 sets that parameter to 2^λ (e.g. $AL.S = 2^\lambda$).

2.4. Puncturable pseudorandom functions. We recall the definition of puncturable pseudorandom functions (PRF) from [SW14]. Since in this work we only use puncturing at one point, the definition below is restricted to puncturing only at one point instead of at polynomially many points.

DEFINITION 2.15 (puncturable PRFs). *A puncturable family of PRFs is given by a triple of uniform PPT machines* $(\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F})$ *and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:*

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

PSEUDORANDOM AT PUNCTURED POINT. *For every* PPT *adversary $(\mathcal{A}_1, \mathcal{A}_2)$, there is a negligible function μ, such that in an experiment where $\mathcal{A}_1(1^\lambda)$ outputs a point $i \in \{0,1\}^{n(\lambda)}$ and a state σ, $K \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K(i) =$*

$\mathsf{PRF.Punc}(K, i)$*, the following holds:*

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

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

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

As observed by [BW13, BGI14, KPTZ13], the GGM construction of PRFs [GGM86] from pseudorandom generators (PRGs) yields puncturable PRFs for any (efficiently computable) $n(\cdot)$ and $m(\cdot)$. Furthermore, it is easy to see that if the PRG underlying the GGM construction is subexponentially hard (and this can in turn be built from subexponentially hard one-way funstions (OWFs)), then the resulting puncturable PRF is subexponentially pseudorandom.

2.5. Injective noninteractive commitment. An injective noninteractive bit commitment scheme is a pair of polynomials $n(\cdot)$ and $m(\cdot)$ and a polynomial-sized ensemble of injective functions $\mathsf{Commit}_\lambda : \{0,1\} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$ such that for all polynomial time adversaries $\mathcal{A}$,

$$\Pr\left[\mathcal{A}(1^\lambda, \mathsf{Commit}_\lambda(b; r)) = b \middle| \begin{array}{l} b \leftarrow \{0,1\} \\ r \leftarrow \{0,1\}^{n(\lambda)} \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

The scheme is said to be subexponentially secure if the $\mathrm{negl}(\lambda)$ above can in fact be replaced by $2^{-\lambda^{\epsilon}}$ for some constant $\epsilon > 0$.

We can construct a (subexponentially secure) injective noninteractive commitment scheme given a (subexponentially secure) injective one-way function

$$f : \{0,1\}^{n'(\lambda)} \to \{0,1\}^{m'(\lambda)},$$

and we give the construction here without proof. Let h be a hard-core bit of f (note that the standard construction of a one-way function with a hard-core bit preserves injectivity). Then define $n(\lambda) = n'(\lambda)$, $m(\lambda) = m'(\lambda) + 1$, and

$$\mathsf{Commit}_\lambda(b; r) = f(r) \| b \oplus h(r).$$

3. Succinct garbling via garbled circuits and garbled RAMs. In this section (based on [BGL+15]) we first show a succinct garbling scheme for Turing machines based on **iO** for circuits and Yao's garbled circuit method. We then show how to extend it to RAMs based on circuit **iO** and any (nonsuccinct) RAM garbling scheme.

3.1. Overview. We first provide an overview of our garbling scheme for the class of Turing machines TM with space-dependent complexity. In particular, we show the following theorem (formally proven in section 3.2).

THEOREM 3.1. *Assuming the existence of* ***iO*** *for circuits and one-way functions, there exists a garbling scheme for* TM *with space-dependent complexity.*

Toward this goal, we proceed in two steps: In the first step, we construct a *nonsuccinct* garbling scheme for TM, which satisfies the correctness and security requirements of Definitions 2.1 and 2.2, 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 Turing machine being garbled, M (as well as the simulation algorithm); the produced garbled Turing machine is of size in the same order. In contrast, the time to compute the encoded input $\widehat{x}$ depends on the length of the input x, but not on the typically larger running time of M. This feature of "independent input encoding" is crucial for our construction. Another crucial property of the nonsuccinct garbling scheme is that the garbled machine consists of many "small garbled pieces" that can be generated separately. In the second step, we use **iO** to "compress" the size of the garbled program by providing an obfuscated program that takes an index as input and generates the "garbled piece" corresponding to that index. As a result, the final garbled program (namely, the obfuscated program) is small (i.e., depends on the space complexity $M.S$). It is only at evaluation time that the underlying nonsuccinct garbled program is unraveled, by running the obfuscated program on every index, and decoded.

Nonsuccinct garbling scheme. We outline the *nonsuccinct garbling scheme* for Turing machines, based on any one-way function. A "trivial" approach toward such garbling is to simply transform any polynomial-time Turing machine into a circuit and then garble the circuit. While our construction in essence relies on this principle, it will in fact invoke garbling for "small" fixed-size circuits. Concretely, we rely on the existence of a circuit garbling scheme satisfying two additional properties. First, we require that the shared string **key**, and thus also the input encoding, be generated independently of the circuit being garbled (e.g., **key** is sampled at random and given to both the input encoding and circuit garbling procedures). Second, we require that encoded inputs can be simulated, given only the input size, whereas the garbled program is simulated using the result $M(x)$ of the computation (and the randomness used to simulate the encoded input). We refer to such schemes as *garbling schemes with independent input encoding* (Definition 2.7) and note that Yao's basic scheme [Yao86] satisfies the two properties.

Our nonsuccinct garbling scheme now proceeds as follows. Let M be a Turing machine with bounded space complexity $M.S$, running-time $M.T$, and inputs of length $M.n$. We construct a "chain" of $M.T$ garbled circuits that evaluate M step by step. More precisely, we first generate keys $\mathbf{key}_1, \ldots, \mathbf{key}_{M.T}$ for the $M.T$ garbled circuits. The jth garbled circuit (which is computed using key $\mathbf{key}_j$) takes as input some state of M and computes the next state (i.e., the state after one computation step); if the next state is a final state, it returns the output generated by M; otherwise it outputs an *encoding* of this new state using the next key $\mathbf{key}_{j+1}$. (Note that after $M.T$ steps we are guaranteed to get to a final state, and thus this process is well-defined.)

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

Security of the nonsuccinct scheme. To show that this construction is a secure (nonsuccinct) garbling scheme we need to exhibit a simulator that, given just the output $y = M(x)$ of the program M on input x and *the number of steps* t^* *taken by* $M(x)$, 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.) Toward this goal, we start by simulating the t^*th garbled circuit with the output being set to y; this simulation generates a garbled program $\tilde{M}_{t^*}$ and an encoded input $\widetilde{\mathrm{conf}}_{t^*-1}$ (which are the simulated input keys).

We then iteratively, in descending order, simulate the jth garbled circuits $\tilde{M}_j$ with the output being set to $\widetilde{\mathrm{conf}}_{j+1}$ generated in the previously simulated garbled circuit. We finally simulate the remaining $j > t^*$ garbled circuits $\tilde{M}_j$ with the output being set to some arbitrary output in the range of the circuit (e.g., the output y). The simulated encoded input is then $\widetilde{\mathrm{conf}}_1$ and the simulated garbled program is $(\tilde{M}_1, \ldots, \tilde{M}_{M.T})$.[7]

To prove indistinguishability of the simulated garbling and the real garbling, we consider a sequence of hybrid experiments $H_0, \ldots, H_{M.T}$, where in H_j the first j garbled circuits are simulated, and the remaining $M.T - j$ garbled circuits are honestly generated. To "stitch together" the simulated circuits with the honestly generated ones, the jth garbled circuit is simulated using as output an honest encoding $\widehat{\mathrm{conf}}_j$ of the actual configuration conf_j of the Turing machine M after j steps.

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

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

Succinct garbling scheme. We now show how to make this garbling scheme succinct. The idea is simple: instead of providing the actual garbled circuits in the clear, we provide an obfuscation of the *randomized* program that generates these garbled circuits. More precisely, we provide an $\mathbf{iO}$ of a program $\mathbf{M}^{s,s'}(\cdot)$, where s and s' are seeds for a PRF F: $\mathbf{M}^{s,s'}(j)$, given a "time step" $j \in [M.T]$, which generates the jth garbled circuit in the nonsuccinct garbling of M using pseudorandom coins generated by the PRF with seed s and s'. Specifically, it uses $\mathsf{F}(s, j)$ and $\mathsf{F}(s, j+1)$ as randomness to generate $\mathbf{key}_j$ and $\mathbf{key}_{j+1}$ (recall that the functionality of the jth circuit depends on $\mathbf{key}_{j+1}$) and uses $\mathsf{F}(s', j)$ as randomness for garbling the jth circuit.

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

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

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

More precisely, given an output y and the running time t^* of $M(x)$, the simulator outputs the obfuscation $\widetilde{\mathbf{\Lambda}}$ of a program $\widetilde{\mathbf{M}}^{y,t^*,s,s'}$ that, given input j, outputs a simulated jth garbled circuit, using randomness $\mathsf{F}(s, j+1)$ to generate $\widetilde{\mathrm{conf}}_{j+1}$ as the output, and $\mathsf{F}(s, j)$ and $\mathsf{F}(s', j)$ as the extra randomness needed to simulate the input $\widetilde{\mathrm{conf}}_j$ and the garbled M_j.[8] The encoded input $\tilde{x}$ is simulated as in the nonsuccinct garbling scheme, but using pseudorandom coins $\mathsf{F}(s, 1)$.

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

To prove indistinguishability of any two consecutive hybrids H'_j and H'_{j+1}, we rely on the punctured program technique of Sahai and Waters [SW14] to replace pseudorandom coins $\mathsf{F}(s, j+1)$, $\mathsf{F}(s', j+1)$ for generating the $j+1$st simulated garbled circuit with truly random coins, and then rely on the indistinguishability of the simulation of the $j+1$st garbled circuit. A bit more concretely, at each step we puncture the seeds s, s' only on the (three) points corresponding to the $j+1$st step, and hardwire instead the corresponding outputs generated by $\widetilde{\mathbf{M}}_j^{s,s'}$; next, relying on the puncturing guarantee, we can sample these outputs using true independent randomness. At this point, we can already replace the real hardwired garbling with a simulated one. Finally, we go back to generating the hardwired value pseudorandomly as part of the circuit's logic, now identical to $\widetilde{\mathbf{M}}_{j+1}^{s,s'}$, and "unpuncture" the seeds s, s'. We note that each such step requires hardwiring a new (real) intermediate configuration conf_{j+1} (used to simulate the $j+1$st garbling), but now the previous hardwired configuration conf_j can be "forgotten" and blowup is avoided.

iO for a simple class of circuits is enough. The obfuscated circuits in the construction are of a special kind—their input size is $O(\log M.T)$. Canetti et al. [CLTV15] show that **iO** for NC^1 can be bootstrapped to obtain **iO** for all circuits, assuming puncturable PRFs in NC^1 [BLMR13], and incurring a security loss that is exponential in the size of the input. Accordingly, for polynomial $M.T$, it suffices to assume (polynomially secure) **iO** for classes in NC^1 with logarithmic-size inputs.

A garbing scheme for any bounded space computation. We observe that the above approach for constructing a succinct garbling scheme for bounded space Turing machines applies generally to any *bounded space computation* (e.g., bounded space RAM). In particular, we show the following theorem (formally proven in section 3.3).

THEOREM 3.2. *Assume the existence of **iO** for circuits and one-way functions. There exists a garbling scheme for any abstract model of sequential computation, such*

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

as TM *and* RAM*, with space-dependent complexity.*

The succinct garbling scheme described above for Turing machines does not apply uniquely to Turing machines, but rather to any model of computation that can be divided into sequential steps using one memory, for instance, RAMs. Given an underlying circuit garbling scheme $\mathcal{GS} = (\mathsf{Garb}, \mathsf{Encode}, \mathsf{Eval})$ with independent input encoding, to construct a garbling scheme $\mathcal{GS}^A$ for $\{\mathcal{AL}_\lambda\}$, proceed with the following two steps.

Step 1. *Construct a nonsuccinct garbling scheme.* Observe that the computation of a machine AL of $AL.T$ steps can be divided into $AL.T$ 1-*step "blocks"* that transforms the current configuration to the next; therefore, to garble AL, it suffices to produce a sequence of "*garbled blocks*," one for each 1-step block. The actual programs being garbled is an "*augmented block*," whose execution consists of a 1-step block followed by the encoding algorithm of $\mathcal{GS}$ that encodes the output configuration for the next garbled block (when an output is produced, it is output directly without encoding). The final garbling then consists of a sequence of T garbled blocks.

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

Improved construction and analysis. We also note that in the case of RAM, our construction can be made more efficient, relying on previous garbled RAM solutions [LO13, GHL+14, GLOS15]. In the solution described above, we can in fact replace the underlying circuit garbling scheme with any garbling scheme, as long as it admits independent input encoding. For instance, in the case where the program AL is a RAM, we may use previous *garbled RAM* solutions [LO13, GHL+14, GLOS15]. The benefit is that this allows optimizing the efficiency of our scheme. Indeed, in the solution described above, each step of the machine is translated to a garbled circuit of size $O(AL.S)$ (up to polynomial factors in the security parameter), which means that the complexity of encoding is $\mathrm{poly}_{\mathbf{iO}}(AL.S)$, where $\mathrm{poly}_{\mathbf{iO}}(\cdot)$ is the overhead due to obfuscation, and the complexity of decoding for a T-time computation $AL(x)$ is at least $T \cdot \mathrm{poly}_{\mathbf{iO}}(AL.S)$, which may be significantly larger than the original computation.

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

This modification on its own may still be insufficient; indeed, obfuscating the circuit that produces the garbled RAM may incur nonlinear overhead $\mathrm{poly}_{\mathbf{iO}}(\cdot)$, so that eventually decoding may take time $\mathrm{poly}_{\mathbf{iO}}(AL.S) \cdot AL.T/AL.S$, which may be again as large as $AL.T \cdot AL.S$. To circumvent this blowup, and as a result of independent interest, we show how to bootstrap any **iO** for circuits to one that has quasi-linear blowup. Overall, in the new solution, for a T-time computation $AL(x)$, encoding

takes time $\tilde{O}(AL.S)$ and decoding $\widehat{AL}(x)$ takes time $O(T + AL.S)$).

3.2. Succinct garbling for Turing machines via garbled circuits. In this section we present our succinct garbling scheme for Turing machines. As explained in the overview, we first construct a nonsuccinct garbling scheme based on Yao's garbled circuit construction. Next, we show how to rely on this nonsuccinct scheme and **iO** to construct a succinct garbling scheme.

3.2.1. A nonsuccinct garbling scheme. We now describe formally our nonsuccinct garbling scheme $\mathcal{GS}_{ns} = (\mathsf{Garb}_{ns}, \mathsf{Encode}_{ns}, \mathsf{Eval}_{ns})$. We rely on a garbling scheme for polynomial-sized circuits with independent input encoding.

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

$$\mathsf{Garb}_{\mathsf{CIR}} = (\mathsf{Gen}_{\mathsf{CIR}}, \mathsf{Gb}_{\mathsf{CIR}}),$$
$$\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim.Gen}_{\mathsf{CIR}}, \mathsf{Sim.Gb}_{\mathsf{CIR}}),$$

as described in Definition 2.7.

The execution of a Turing machine M consists of a sequence of steps in which 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 $M.T$ evaluations of $\mathsf{Next}^{D,S}(M, \cdot)$, producing a chain of configurations denoted by

$$\mathsf{CONFIG}(M, x) = (T^*, \mathrm{conf}_1, \ldots, \mathrm{conf}_{M.T}, \mathrm{conf}_{M.T+1}),$$

where $T^* = T_M(x)$ and $\mathrm{conf}_1, \ldots, \mathrm{conf}_{T^*-1}, \mathrm{conf}_{T^*}$ are the sequence of configurations of $M(x)$ until it halts (conf_t is the configuration before the tth step starts). $\mathrm{conf}_{T^*}, \ldots, \mathrm{conf}_{M.T+1}$ are set for simplicity to the output $y = M(x)$.

We note that the initial configuration conf_1 can be derived efficiently from x; conf_{T^*} is called the final configuration, which can be efficiently recognized and from which an output y can be extracted efficiently.

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 $\alpha_1, \ldots, \alpha_t$ and $\beta_1, \ldots, \beta_t$; 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 $t+1$st garbled circuit $\mathrm{key}_{t+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_{t+1})$ using randomness α_{t+1}. (Note that key_t is generated for inputs of length S.)
2. *Prepare the step-circuit* $\mathbf{C}_t$*:*
   $\mathbf{C}_t$ on an S-bit input conf_t (i) computes $\mathrm{conf}_{t+1} = \mathsf{Next}^{D,S}(M, \mathrm{conf}_t)$; (ii) if conf_{t+1} is a final configuration, simply outputs the output y con-

tained in it;[9] (iii) otherwise, translate conf_{t+1} to the garbled inputs of the $t+1$st garbled circuit by computing $\widehat{\text{conf}}_{t+1} = \mathsf{Encode}_{\mathsf{CIR}}(\text{key}_{t+1}, \text{conf}_{t+1})$.

3. *Garble the step-circuit* $\mathbf{C}_t$*:*
   Compute the key using randomness α_t, $\text{key}_t = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S; \alpha_t)$, and garble $\mathbf{C}_t$ using randomness β_t, $\widehat{\mathbf{C}}_t = \mathsf{Gb}_{\mathsf{CIR}}(\text{key}_t, \mathbf{C}_t; \beta_t)$.
4. *Output* $\widehat{\mathbf{C}}_t$.

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

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

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

THE EVALUATION ALGORITHM $\mathsf{Eval}_{ns}(\hat{M}, \hat{x})$: Evaluate the chain of garbled circuits $\hat{M} = (\widehat{\mathbf{C}}_1, \ldots, \widehat{\mathbf{C}}_T)$ in sequence in T iterations: At iteration t, compute $z = \mathsf{Eval}_{\mathsf{CIR}}(\widehat{\mathbf{C}}_t, \widehat{\text{conf}}_t)$; if z is the garbled input $\widehat{\text{conf}}_{t+1}$ 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 nonsuccinct garbling scheme for TM.

Efficiency. We summarize the complexity of different algorithms of the nonsuccinct 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 $\text{poly}(\lambda, D, S) \times T$ and produces a garbling machine whose size is of the same order. Thus the garbling scheme is nonsuccinct. On the other hand, the encoding and evaluation algorithms Encode_{ns} and Eval_{ns} are all deterministic polynomial-time algorithms. Finally, the simulation runs in time $\text{poly}(\lambda, D, S) \times T$, as does the garbling algorithm.

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

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

Let $\mathsf{CONFIG}(M, x) = (T^*, \text{conf}_1, \ldots, \text{conf}_T, \text{conf}_{T+1})$ 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 $\widehat{\text{conf}}_t$ corresponding to conf_t, computes the correct garbled inputs $\widehat{\text{conf}}_{t+1}$ corresponding to conf_{t+1}, and (2) for $t = T^*$, the garbled circuit $\widehat{\mathbf{C}}_{T^*}$, if given the garbled input $\widehat{\text{conf}}_{T^*-1}$ 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 the garbled initial configuration $\widehat{\text{conf}}_1$, 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 $x_\lambda \in \{0,1\}^{M.n}$. To show 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

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

the appearance of $M.n$ and $M.m$ as input to Sim:

(3.1)
$$\left\{\mathsf{real}_{ns}(1^\lambda, M, x)\right\} = \left\{(\hat{M}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}_{ns}(1^\lambda, M),\ \hat{x} = \mathsf{Encode}_{ns}(\mathbf{key}, x) : (\hat{M}, \hat{x})\right\}_\lambda,$$

(3.2)
$$\left\{\mathsf{simu}_{ns}(1^\lambda, M, x)\right\} = \left\{(\tilde{M}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}_{ns}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T_M(x), M(x)) : (\tilde{M}, \tilde{x})\right\}_\lambda.$$

Below we describe the simulation algorithm. Observe that the garbled machine $\hat{M}$ consists of T garbled circuits $(\widehat{\mathbf{C}}_1, \ldots, \widehat{\mathbf{C}}_T)$, and the garbled input $\hat{x}$ is simply the garbled input of the initial configuration conf_0 (corresponding to x) for the first garbled circuit $\widehat{\mathbf{C}}_1$. Naturally, to simulate them, the algorithm Sim_{ns} needs to utilize the simulation algorithm $\mathsf{Sim}_{\mathsf{CIR}} = (\mathsf{Sim.Gen}_{\mathsf{CIR}}, \mathsf{Sim.Gb}_{\mathsf{CIR}})$ of the circuit garbling scheme, which requires knowing the output of each garbled circuit. In a real evaluation with $\hat{M}, \hat{x}$, the output of the (T^*)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 $t = T^*, \ldots, T$ can be simulated using output y, whereas garbled circuits $t = 1, \ldots, T^* - 1$ will be simulated using the *simulated garbled inputs* for circuit $t+1$. More precisely, the simulation algorithm is as follows.

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 $\alpha_1, \ldots, \alpha_T, \beta_1, \ldots, \beta_T$. 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 tth simulated circuit $\widetilde{\mathbf{C}}_t$:*
   If $t \geq T^*$, $out_t = y$. Otherwise, if $t < T^*$, set the output as the garbled input for the next garbled circuits, that is, $out_t = \widetilde{\mathrm{conf}}_{t+1}$ computed from $(\widetilde{\mathrm{conf}}_{t+1}, \mathbf{st}_{t+1}) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_{t+1})$ using randomness α_{t+1}.
2. *Simulate the tth step-circuit $\widetilde{\mathbf{C}}_t$:*
   Given the output out_t, simulate the tth garbled circuit $\widetilde{\mathbf{C}}_t$ by computing first $(\widetilde{\mathrm{conf}}_t, \mathbf{st}_t) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_t)$ and then $\widetilde{\mathbf{C}}_t = \mathsf{Sim.Gb}_{\mathsf{CIR}}(1^\lambda, S, q, out_t, \mathbf{st}_t\ ;\ \beta_t)$, using randomness α_t, β_t, where $q = q(\lambda, S)$ is the size of the circuit $\mathbf{C}_t$.
3. *Output $\widetilde{\mathbf{C}}_t$.*

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

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

Toward 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 $\mathsf{hyb}^0_{ns}, \ldots, \mathsf{hyb}^T_{ns}$, 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 γth hybrid simulator on input $(1^\lambda, M, x)$ simulates 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 γth 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 $S_1, \ldots, S_\ell$ is a partition of the domain X.

THE HYBRID SIMULATION ALGORITHM $\mathsf{HSim}^{\gamma}_{ns}(1^\lambda, M, x)$ FOR $\gamma = 0, \ldots, T$:

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

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

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

- If $t \le \gamma - 1$, compute $\widetilde{\mathbf{C}}_t = \mathbf{Q}^{\lambda, S, |M|, T^*, y}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$; output $\widetilde{\mathbf{C}}_t$.
- If $t \ge \gamma + 1$, compute $\widehat{\mathbf{C}}_t = \mathbf{P}^{\lambda, S, M}(t\ ;\ (\alpha_t, \alpha_{t+1}, \beta_t))$; output $\widehat{\mathbf{C}}_t$.
- If $t = \gamma$, compute $\widetilde{\mathbf{C}}_t = \mathbf{R}^{\lambda, S, \mathrm{conf}_{\gamma+1}}(\gamma\ ;\ (\alpha_\gamma, \alpha_{\gamma+1}, \beta_\gamma))$ using program $\mathbf{R}$ defined below; output $\widetilde{\mathbf{C}}_t$.

  Program $\mathbf{R}^{\lambda, S, \mathrm{conf}_{\gamma+1}}(\gamma\ ;\ (\alpha_\gamma, \alpha_{\gamma+1}, \beta_\gamma))$:

  1. *Prepare the output out_γ of the simulated γth circuit $\widetilde{\mathbf{C}}_t$:*
     Set the output out_γ to y if $\mathrm{conf}_{\gamma+1}$ is a final configuration. Otherwise, the output should be the garbled input corresponding to $\mathrm{conf}_{\gamma+1}$ for the next garbled circuit; since the $\gamma+1$st circuit is generated honestly, we compute $out_\gamma = \widehat{\mathrm{conf}}_{\gamma+1}$ by first computing $\mathrm{key}_{\gamma+1} = \mathsf{Gen}_{\mathsf{CIR}}(1^\lambda, 1^S\ ;\ \alpha_{\gamma+1})$, and then encoding $\widehat{\mathrm{conf}}_{\gamma+1} = \mathsf{Encode}_{\mathsf{CIR}}(\mathrm{key}_{\gamma+1}, \mathrm{conf}_{\gamma+1})$.
     (Note that the difference between program $\mathbf{Q}$ and $\mathbf{R}$ is that the former prepares the output out_γ using simulated garbled input $\widetilde{\mathrm{conf}}_{t+1}$, whereas the latter uses honestly generated garbled input $\widehat{\mathrm{conf}}_{\gamma+1}$.)
  2. *Simulate the γth circuit $\widetilde{\mathbf{C}}_t$:*
     Given the output out_γ, simulate the γth garbled circuit $\widetilde{\mathbf{C}}_\gamma$ by computing $(\widetilde{\mathrm{conf}}_\gamma, \mathbf{st}_\gamma) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_\gamma)$ and $\widetilde{\mathbf{C}}_t = \mathsf{Sim.Gb}_{\mathsf{CIR}}(1^\lambda, S, q, out_\gamma, \mathbf{st}_\gamma\ ;\ \beta_\gamma)$, where $q = q(\lambda, S)$ is the size of the circuit $\mathbf{C}_t$.
  3. *Output $\widetilde{\mathbf{C}}_t$.*

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

We use notation $\mathsf{hyb}^\gamma_{ns}(1^\lambda, M, x)$ to denote 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, that is, $\{\mathsf{hyb}^0_{ns}(1^\lambda, M, x)\} = \{\mathsf{real}_{ns}(1^\lambda, M, x)\}$ (where real_{ns} is the distribution of honestly generated garbling; see (3.1)); furthermore, when $\gamma = T$, $\mathbf{M}^T = \mathbf{Q}$ and the garbled input $\tilde{x}_\gamma$ is simulated, that is, $\left\{\mathsf{hyb}^\gamma_{ns}(1^\lambda, M, x)\right\} = \left\{\mathsf{simu}_{ns}(1^\lambda, M, x)\right\}$ (where simu_{ns} is the distribution of simulated garbling; see (3.2)). 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 3.3. *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 $\mathsf{hyb}_{ns}^{\gamma}(1^\lambda, M, x)$ is the following: Let conf_γ be the intermediate configuration at the beginning of step γ.

- In $\mathsf{hyb}_{ns}^{\gamma-1}$, 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 $\tilde{x}_\gamma$ depend 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 $\mathsf{hyb}_{ns}^{\gamma-1}$ can be generated by the following alternative sampling algorithm:
  - Generate garbled circuits $\gamma+1,\ldots,T$ honestly using program $\mathbf{P}$; prepare the γth circuit $\mathbf{C}_\gamma$ using $\mathrm{key}_{\gamma+1}$.
  - Receive externally honest garbling $(\widehat{\mathbf{C}}_\gamma, \widehat{\mathrm{conf}}_\gamma)$ of $(\mathbf{C}_\gamma, \mathrm{conf}_\gamma)$.
  - Simulate the first $\gamma-1$ circuits using $\mathbf{R}$ and $\mathbf{Q}$, with $\widehat{\mathrm{conf}}_\gamma$ hardwired in $\mathbf{R}$.
- In $\mathsf{hyb}_{ns}^{\gamma}$, the 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 $\tilde{x}_{\gamma+1}$ depend potentially on the corresponding simulated garbled input $\widetilde{\mathrm{conf}}_\gamma$ of $\widetilde{\mathbf{C}}_\gamma$.
  In other words, the output of $\mathsf{hyb}_{ns}^{\gamma}$ can be generated by the same alternative sampling algorithm above, except that the second step is modified as follows:
  - Receive externally simulated garbling $(\widetilde{\mathbf{C}}_\gamma, \widetilde{\mathrm{conf}}_\gamma)$ generated using output $\mathbf{C}_\gamma(\mathrm{conf}_\gamma)$.

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

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

3.2.2. 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. Our construction relies on the following building blocks:

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

Circuit $\mathbb{P} = \mathbb{P}^{\lambda,S,M,K_\alpha,K_\beta}$: On input $t \in [T]$, does:
Generates pseudorandom 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 pseudorandom 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 Figures 3.1, 3.2, and 3.3 are padded to their maximum size.

FIG. 3.1. *Circuits used in the construction and simulation of* $\mathcal{GS}$.

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

THE GARBLING ALGORITHM $\mathsf{Garb}(1^\lambda, M)$:

1. *Sample the PRF keys* $K_\alpha \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{P}$:
   Obfuscate the circuit $\mathbb{P}(t) = \mathbb{P}^{\lambda,S,M,K_\alpha,K_\beta}(t)$ as described in Figure 3.1, which is essentially a wrapper program that evaluates $\mathbf{P}$ on t using pseudorandom 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 pseudorandom 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 nonsuccinct 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 $t \in [T]$, the previous $t-1$ garbled circuits have 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 used later by the encoding algorithm Encode to obtain garbled input $\hat{x}$, while the former is used later by the evaluation algorithm Eval to create the nonsuccinct garbling $\hat{M} = \{\widehat{\mathbf{C}}_t = \overline{\mathbb{P}}(t)\}_{t\in[T]}$; the nonsuccinct garbling $\hat{M}$ is then evaluated with $\hat{x}$ using algorithm Eval_{ns}. The distribution of the garbled input and the nonsuccinct 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 nonsuccinct scheme, *using pseudorandom coins*, formalized below:

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

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

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

The correctness of the nonsuccinct 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 nonsuccinct 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 derandomized version of P) which garbles circuits $\mathbf{C}_t$ using pseudorandom 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 nonsuccinct garbling scheme, so do $\mathbf{P}$ and $\mathbb{P}$ (note that the input range T of these two programs is contained as part of the description of M, and hence $|M| > \log T$). Therefore, Garb takes time $\mathrm{poly}(\lambda, |M|, S)$ to produce the obfuscation of $\mathbb{P}$. Additionally, notice that Garb generates the $\mathbf{key}$ 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 the same time as the Encode_{ns} algorithm, which is $\mathrm{poly}(\lambda, |M|, S)$.
- 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 nonsuccinct 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 nonsuccinct garbling $\hat{M}$ in one shot, but rather generates and evaluates the garbled circuits in $\hat{M}$ one by one, it thus 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 $x_\lambda \in \{0,1\}^{M.n}$. Toward show-

ing the security of $\mathcal{GS}$, we construct a simulator Sim satisfying that the following two ensembles are indistinguishable in λ:

$$\{\mathsf{real}(1^\lambda, M, x)\} = \left\{(\overline{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, M),\ \hat{x} = \mathsf{Encode}(\mathbf{key}, x)\ :\ (\overline{\mathbb{P}}, \hat{x})\right\}_\lambda, \tag{3.3}$$

$$\{\mathsf{simu}(1^\lambda, M, x)\} = \left\{(\overline{\mathbb{Q}}, \tilde{x}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, |x|, |M|, S, T, T_M(x), M(x))\ :\ (\overline{\mathbb{Q}}, \tilde{x})\right\}_\lambda. \tag{3.4}$$

As discussed in the overview, the simulation will obfuscate the program $\mathbf{Q}$ used for simulating the nonsuccinct garbled machine $\tilde{M} = (\widetilde{\mathbf{C}}_1, \ldots, \widetilde{\mathbf{C}}_T)$, shown more precisely as follows.

THE SIMULATION ALGORITHM $\mathsf{Sim}(1^\lambda, |x|, |M|, S, T, T^* = T_M(x), y = M(x))$:

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{Q}$*:*
   Obfuscate the circuit $\mathbb{Q}(t) = \mathbb{Q}^{\lambda, S, |M|, T^*, y, K_\alpha, K_\beta}(t)$ as described in Figure 3.1, which is essentially a wrapper program that evaluates $\mathbf{Q}$ on t, using pseudorandom 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 that simulator Sim_{ns} does, but using pseudorandom coins. That is, compute $(\widetilde{\mathrm{conf}}_1, \mathbf{st}_1) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S\ ;\ \alpha_1)$, where $\alpha_1 = \mathsf{F}(K_\alpha, 1)$; set $\tilde{x} = \widetilde{\mathrm{conf}}_1$.
4. *Finally, output* $(\overline{\mathbb{Q}}, \tilde{x})$.

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

To show 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 (3.3) and (3.4) for formal definitions of real and simu), we will consider a sequence of hybrids $\mathsf{hyb}^0, \ldots, \mathsf{hyb}^T$, 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 derandomized wrapper of) a merged program $\mathbf{M}^\gamma$ that produces a hybrid chain of garbled circuits as in the security proof of the nonsuccinct garbling scheme, where the first γ garbled circuits are simulated and the rest are generated honestly. This hybrid simulation algorithm is given more precisely as follows.

THE HYBRID SIMULATION ALGORITHM $\mathsf{HSim}^\gamma(1^\lambda, M, x)$ FOR $\gamma = 0, \ldots, T$:

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 the PRF keys* $K_\alpha \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$.
2. *Obfuscate the circuit* $\mathbb{M}^\gamma$*:*
   Obfuscate the circuit $\mathbb{M}^\gamma(t) = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \mathrm{conf}_{\gamma+1}, K_\alpha, K_\beta}(t)$ as described in Figure 3.2, 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 pseudorandom 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 pseudorandom 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 Figures 3.1, 3.2, and 3.3 are padded to their maximum size.

FIG. 3.2. *Circuits used in the security analysis of* $\mathcal{GS}$.

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

CLAIM 3.4. *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 in their implementation by at most two inputs, namely, $\gamma, \gamma+1$.
- 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 $\mathsf{H}_0^\gamma, \ldots, \mathsf{H}_7^\gamma$ 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 $i = 0, \ldots, 7$. For convenience, below we suppress the superscript γ and simply use the notation $\mathsf{H}_i = \mathsf{H}_i^\gamma$, $\overline{\mathbb{M}}_i = \overline{\mathbb{M}}_i^\gamma$, $\mathbb{M}_i = \mathbb{M}_i^\gamma$, 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 with the following modifications:

- In the first step, puncture the two PRF keys K_α, K_β at input $\gamma+1$ and obtain $K_\alpha(\gamma+1) = \mathsf{PRF.Punc}(K_\alpha, \gamma+1)$ and $K_\beta(\gamma+1) = \mathsf{PRF.Punc}(K_\beta, \gamma+1)$. Furthermore, compute $\alpha_{\gamma+1} = \mathsf{F}(K_\alpha, \gamma+1)$ and $\beta_{\gamma+1} = \mathsf{F}(K_\beta, \gamma+1)$.
- In the second step, obfuscate a circuit $\mathbb{M}_1$ slightly modified from $\mathbb{M}^\gamma$: Instead of having the full PRF keys K_α, K_β hardwired in, $\mathbb{M}_1$ has the

punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$ and the PRF values $\alpha_{\gamma+1}, \beta_{\gamma+1}$ hardwired in; $\mathbb{M}_1$ proceeds identically to $\mathbb{M}_1$, except that it uses the punctured PRF keys to generate pseudorandom coins corresponding to input $t \neq \gamma+1$ and directly uses $\alpha_{\gamma+1}, \beta_{\gamma+1}$ as the coins for input $t = \gamma+1$. See Figure 3.3 for a description of $\mathbb{M}_1 = \mathbb{M}_1^\gamma$.

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

Circuit $\mathbb{M}_1^\gamma = (\mathbb{M}_1^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$: On input $t \in [T]$, does:
If $t \neq \gamma$, generate pseudorandom string $\alpha_{t+1} = \mathsf{F}(K_\alpha(\gamma+1), t+1)$.
If $t \neq \gamma+1$, generate pseudorandom 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 $\mathbb{M}_3^\gamma = (\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$: On input $t \in [T]$, does:
If $t = \gamma+1$, output $\widehat{\mathbf{C}}_{\gamma+1}$.
If $t = \gamma$, set out_γ using $\widetilde{\text{conf}}_{\gamma+1}$ as in step 1 of program $\mathbf{R}$; simulate and output $\widetilde{\mathbf{C}}_\gamma$ as in step 2 of $\mathbf{R}$.
Otherwise, compute as $\mathbb{M}_2^\gamma$ does using the punctured keys $K_\alpha(\gamma+1)$, $K_\beta(\gamma+1)$.

Circuit $\mathbb{M}_4^\gamma = (\mathbb{M}_4^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$:
Identical to $(\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$, with simulated garbling pair $\widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}$.

Circuit $\mathbb{M}_5^\gamma = (\mathbb{M}_5^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$: On input $t \in [T]$, does:
If $t = \gamma+1$, compute $\widetilde{\mathbf{C}}_{\gamma+1}$ using program $\mathbf{R}$ with randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$.
If $t = \gamma$, compute $\widetilde{\mathbf{C}}_\gamma$ using program $\mathbf{Q}$, which internally computes $\widetilde{\text{conf}}_{\gamma+1}$ for setting the output out_γ using randomness $\alpha'_{\gamma+1}$.
Otherwise, compute as $\mathbb{M}_4^\gamma$ does using the punctured keys $K_\alpha(\gamma+1)$, $K_\beta(\gamma+1)$.

Circuit $\mathbb{M}_6^\gamma = (\mathbb{M}_6^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$:
Identical to $(\mathbb{M}_5^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$, with $\alpha_{\gamma+1} = \mathsf{F}(K_\alpha, \gamma+1)$, $\beta_{\gamma+1} = \mathsf{F}(K_\beta, \gamma+1)$.

The circuits in Figures 3.1, 3.2, and 3.3 are padded to their maximum size.

FIG. 3.3. *Circuits used in the security analysis of $\mathcal{GS}$, continued.*

By construction, H_1 only differs from hyb^γ in the underlying program that is obfuscated; this 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 with the following modifications: Instead of using

pseudorandom 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 replaces $\alpha_{\gamma+1}, \beta_{\gamma+1}$ with these truly random strings. More specifically, H_2 obfuscates a program $\mathbb{M}_2$ that is identical to $\mathbb{M}_1$, but with $(K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1})$ hardwired in; furthermore, if $\gamma = 0$, α'_1 (as opposed to α_1) is used to generate the garbled input $\tilde{x}_2$. Since only the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$ are used in the whole simulation procedure, it follows from the pseudorandomness of the punctured PRF that the output $(\overline{\mathbb{M}}_2, \tilde{x}_2)$ of H_2 is indistinguishable from $(\overline{\mathbb{M}}_1 \tilde{x}_1)$ 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 3.1 for a description of $\mathbb{M}_3$.) It is easy to see that when the correct values $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\mathrm{conf}}_{\gamma+1}$ are hardwired, the program $\mathbb{M}_3$ has the same functionality as $\mathbb{M}_2$.
- In H_2, if $\gamma = 0$, α'_1 is used for garbling the input,

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

  where conf_1 is the initial state corresponding to x.
  Second modification: Instead, if $\gamma = 0$, hybrid H_3 receives $\widehat{\mathrm{conf}}_1$ externally and directly outputs it as the garbled inputs $\hat{x}_3 = \widehat{\mathrm{conf}}_1$.

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

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

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

$$(\widetilde{\mathrm{conf}}_{\gamma+1}, \mathbf{st}_{\gamma+1}) = \mathsf{Sim.Gen}_{\mathsf{CIR}}(1^\lambda, S; \alpha'_{\gamma+1}), \tag{3.5}$$

$$\widetilde{\mathbf{C}}_{\gamma+1} = \mathsf{Sim.Gb}_{\mathsf{CIR}}\left(1^\lambda, S, 1^q, out_{\gamma+1}, \mathbf{st}_{\gamma+1}; \beta'_{\gamma+1}\right), \tag{3.6}$$

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

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

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

- It obfuscates a program $\mathbb{M}_5$ that has $K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}$ 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 (3.5) and (3.6)).
  On input γ, it computes $\widetilde{\mathbf{C}}_\gamma$ using the program $\mathbf{Q}$ with randomness $\alpha_\gamma, \alpha'_{\gamma+2}, \beta_\gamma$ (which computes internally $\widetilde{\mathrm{conf}}_{\gamma+1}$ as described in (3.5)).
  On other inputs $t \neq \gamma, \gamma+1$, it computes as $\mathbb{M}_4$ does.
- If $\gamma = 0$, α'_1 is used for computing $\widetilde{\mathrm{conf}}_1$ as described in (3.5), and then outputs $\tilde{x}_4 = \widetilde{\mathrm{conf}}_1$.

It follows from the fact that $\mathbb{M}_5$ computes $(\widetilde{\mathrm{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ correctly internally that it has the same functionality as $\mathbb{M}_4$; thus, the obfuscations 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 pseudorandom 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 pseudorandomness 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 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 $\mathbb{M}_6^{\gamma}$ have 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.

3.3. Succinct garbling for RAMs via garbled RAMs. Note that our construction of succinct garbling in the previous section uses the underlying circuit garbling scheme $\mathcal{GS}$ in a black-box way. In fact, the scheme does not even require the underlying garbling scheme to be for circuits—*any garbling scheme for any class of algorithms that is "complete," in particular that can be used to implement the augmented blocks, suffices.* Below we show that by plugging in the one-time garbled RAM of [LO13, GHL+14] and modifying the construction of Theorem 3.2 slightly, we can improve the efficiency of our garbling scheme when the algorithm class is RAM. More precisely, we show the following theorem.

THEOREM 3.5. *Assume the existence of* $\boldsymbol{i\mathcal{O}}$ *for circuits and one-way functions. There exists a garbling scheme* $\mathcal{GS}^{\mathsf{RAM}}$ *for* RAM *with* linear*-space-dependent complexity. Furthermore, for any RAM* R *and input* x*, evaluation of a garbled pair* $(\widehat{R}, \hat{x})$ *produced by* $\mathcal{GS}^{\mathsf{RAM}}$ *takes time* $\mathrm{poly}(\lambda, |R|) \times (T_R(x) + S)$.

Proof. Our garbling scheme $\mathcal{GS}$ supports garbling RAM computations, given by a RAM program R and an input x. The RAM program R on input x has access to a memory (initially empty) that R can read from and write to at various locations throughout its execution. If a RAM program R has space complexity S, its memory size is bounded by S.

Below, we start with a brief review of schemes in [LO13, GHL+14].

The RAM garbling schemes of [LO13, GHL+14]. These schemes support garbling RAM computations of a more general form, given by a RAM program R, an input x, and additional initial data D contained in the memory; denote this evaluation by $R^D(x)$. This formulation is able to capture evaluation of multiple RAM programs with persistent memory, denoted by $(R_1(x_1), \ldots, R_k(x_k))^D$, where the memory D is updated by each RAM evaluation and the updates persist to the next evaluation. Though we do not show that our RAM garbling scheme $\mathcal{GS}$ supports persistent memory, as we will see later, the fact that schemes in [LO13, GHL+14] can garble initial data D in linear time in its size $|D|$ is the key toward showing the linear-space-dependent complexity of $\mathcal{GS}$.

Below, we recall the syntax and efficiency of schemes in [LO13, GHL+14] and note several special properties that will be useful for our construction later. Let R be a RAM machine with parameters n, m, S, T.

ALGORITHMS: The schemes contain the following algorithms.

- $k^d \xleftarrow{\$} \mathsf{Gen}^d(1^\lambda)$ samples a data garbling key k^d in $\mathrm{poly}(\lambda)$ steps.
- $\widehat{D} \xleftarrow{\$} \mathsf{GData}^D(k^d)$ takes D as initial data and a data garbling key $k^d \in \{0,1\}^\lambda$ as input and outputs garbled data $\widehat{D}$ in linear time in the data size $\mathrm{poly}(\lambda)|D|$. The RAM program GData has size $\mathrm{poly}(\lambda)$.
- $(\widehat{R}, k^i) \xleftarrow{\$} \mathsf{GProg}(k^d, R)$ takes the description of a RAM machine R and a data garbling key k^d and outputs a garbled program $\widehat{R}$ and an input garbling key k^i in linear time in the time bound $\mathrm{poly}(\lambda, n, |R|)T$.

Independent input/data encoding: Note that the data garbling algorithm GData does not depend on the program or the input. Moreover, the program garbling algorithm GProg can be decomposed into two subroutines, $k^i \xleftarrow{\$} \mathsf{Gen}^i(1^\lambda, 1^n)$ and $\widehat{R} \xleftarrow{\$} \mathsf{GProg}'(k^d, k^i, R)$, which runs, respectively, in time $\mathrm{poly}(\lambda, n)$ and $\mathrm{poly}(\lambda, n, |R|)T$. (The garbled RAM schemes of [LO13, GHL+14] are based on Yao's garbled circuits and inherit the property that the input garbling key depends only on the input length and security parameter.) Thus, the input garbling algorithm GInput below also does not depend on the program.

- $\hat{x} \xleftarrow{\$} \mathsf{GInput}(k^i, x)$ takes input $x \in \{0,1\}^n$ and the input garbling key k^i, and outputs a garbled input $\hat{x}$ in time $\mathrm{poly}(\lambda, n)$.
- $y = \mathsf{GEval}^{\widehat{D}}(\widehat{R}, \hat{x})$ takes a garbled program $\widehat{R}$ and a garbled input $\hat{x}$, with access to the garbled data $\widehat{D}$, and computes the output $y = R^D(x)$ in linear time in the time bound $\mathrm{poly}(\lambda, n, |R|)T$.[10]

SIMULATION: $(\tilde{R}, \tilde{x}, \widetilde{D}) \xleftarrow{\$} \mathsf{GSim}(1^\lambda, (|x|, |D|, |R|, n, m, S, T), R(x))$ simulates the garbled program, input, and data.[11]

Independent garbled input/data simulation: The simulation algorithm can be decomposed into the following two PPT procedures:

$$(\tilde{x}, \widetilde{D}, \mathbf{st}) \xleftarrow{\$} \mathsf{GSim.Gen}(1^\lambda, |x|, |D|),$$
$$\tilde{R} \xleftarrow{\$} \mathsf{GSim.Gb}(1^\lambda, (|x|, |D|, |R|, n, m, S, T), R(x), \mathbf{st}).$$

Finally, we note that the schemes of [LO13, GHL+14] assumes the existence of an identity-based encryption (IBE), which is implied by **iO** and one-way functions.

The basic RAM garbling scheme $\mathcal{GS}'$. From the nonsuccinct RAM garbling schemes of [LO13, GHL+14] described above, we derive a basic RAM garbling scheme $\mathcal{GS}' = (\mathsf{Garb}',\ \mathsf{Encode}',\ \mathsf{Eval}')$ with similar syntax to a circuit garbling scheme with independent input encoding. Then it becomes easier to see why we can use $\mathcal{GS}'$ to replace a circuit garbling scheme in the construction of Theorem 3.2. Furthermore, the efficiency guarantee of $\mathcal{GS}'$ makes the construction more efficient, leading to linear-space-dependent complexity.

ALGORITHMS: The scheme $\mathcal{GS}'$ consists of the following algorithms:

- The garbling algorithm $(\mathbf{key}, \widehat{R}) \xleftarrow{\$} \mathsf{Garb}'(1^\lambda, R)$ consists of two subroutines, $\mathbf{key} \xleftarrow{\$} \mathsf{Gen}'(1^\lambda)$ and $\widehat{R} \xleftarrow{\$} \mathsf{Gb}'(\mathbf{key}, R)$, defined below:

$$\mathbf{key} \xleftarrow{\$} \mathsf{Gen}'(1^\lambda, 1^n) : \mathbf{key} = (k^d, k^i),\ k^d \xleftarrow{\$} \mathsf{Gen}^d(1^\lambda),\ k^i \xleftarrow{\$} \mathsf{Gen}^i(1^\lambda, 1^n)\ ,$$
$$\widehat{R} \xleftarrow{\$} \mathsf{Gb}'(\mathbf{key}, R) : \widehat{R} \xleftarrow{\$} \mathsf{GProg}'(k^d, k^i, R)\ .$$

  The runtime of Garb' is bounded by $\mathrm{poly}(\lambda, n, |R|)T$.
- The input/data encoding algorithm Encode' runs the GInput and GData algorithms:

$$(\hat{x}, \widehat{D}) \xleftarrow{\$} \mathsf{Encode}'^D(\mathbf{key} = (k^d, k^i), x) : \hat{x} \xleftarrow{\$} \mathsf{GInput}(k^i, x),\ \widehat{D} \xleftarrow{\$} \mathsf{GData}^D(k^d)\ .$$

[10] Note that the evaluation runs in linear time in the time bound T, as opposed to the instance running time $T_R(x)$. The weaker efficiency guarantee suffices.

[11] Note that the simulation procedure does not receive the instance running time $T_R(x)$. This matches the fact that evaluation procedure does not have instance-based efficiency.

The runtime of Encode' is bounded by $\mathrm{poly}(\lambda, n)|D|$, and the size of Encode' is bounded by $\mathrm{poly}(\lambda, n)$.

- The evaluation algorithm Eval' runs GEval and outputs $y = R^D(x)$:

$$y = {\mathsf{Eval}'}^{\widehat{D}}(\widehat{R}, \hat{x}) \ : \ y = \mathsf{GEval}^{\widehat{D}}(\widehat{R}, \hat{x}) \, .$$

The runtime of Eval' is bounded by $\mathrm{poly}(\lambda, n, |R|)T$.

SIMULATION: The simulation procedure Sim' runs $\mathsf{Sim.Gen}' := \mathsf{GSim.Gen}$ and $\mathsf{Sim.Gb}' := \mathsf{GSim.Gb}$:

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

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

MODIFICATION TO STEP 1: As before, the first step is constructing a nonsuccinct garbling scheme by dividing a RAM computation into small blocks and garbling all of them using $\mathcal{GS}'$.

The only, and key, difference is, instead of dividing a T-step RAM computation into T 1-step "blocks," divide it into $\lceil T/S \rceil$ S-step "blocks." Each block, say the tth block W_t, is a RAM algorithm that (i) takes as input a state $\mathbf{st}_t$ and has access to a memory with content D_t, (ii) runs R for S steps, and (iii) outputs a state $\mathbf{st}_{t+1}$ for the next block and updates the memory content to D_{t+1}. (When $t = 1$, the memory is initializes with the input $D_1 = x$, and $\mathbf{st}_1$ is set to the initial state.) That is, $\mathbf{st}_{t+1} = W_t^{D_t}(\mathbf{st}_t)$. The states have size at most $|R|$,[12] and the memory has size at most $|S|$.

As before, each block W_t is augmented with the encoding algorithm Encode' that garbles the state $\mathbf{st}_{t+1}$ and data D_{t+1} produced by W_t, implementing a $\mathbf{key}_{t+1}$ used for garbling the next augmented data, that is, the tth augmented block $B^{D_t}(t, \mathbf{key}_{t+1}, \mathbf{st}_t)$ does the following:

1. Run $W_t^{D_t}(\mathbf{st}_t)$ to obtain output $\mathbf{st}_{t+1}$ and updated data D_{t+1}.
2. Run ${\mathsf{Encode}'}^{D_{t+1}}(\mathbf{key}_{t+1}, \mathbf{st}_{t+1})$ and obtain garbled state $\widehat{\mathbf{st}}_{t+1}$ and data $\widehat{D}_{t+1}$.
3. Output $\widehat{\mathbf{st}}_{t+1}, \widehat{D}_{t+1}$. In addition, if $\mathbf{st}_{t+1}$ is a final state, output y.

Then the program $B^\star(t, \mathbf{key}_{t+1}, \star)$ for each augmented block is garbled using Garb', producing garbled blocks.

Efficiency. We now analyze various efficiency parameters.

[12] The states are part of the inputs and outputs of the circuit C_{CPU} implementing the next step function of R. The description size of R is at least $|C_{\mathsf{CPU}}|$. Therefore, the sizes of states are bounded by $|R|$.

- Each augmented block basically runs W_t followed by Encode'. Since $\mathbf{key}_{t+1}$ has size $\mathrm{poly}(\lambda)$, we have

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

  The latter follows since Encode' runs in linear time in the data size and polynomially in the size of the states (which are inputs to Encode').
- By the efficiency of Gb', each garbled block has size

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

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

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

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

Modification to step 2: As before, the second step is using obfuscation to "compress" the size of the nonsuccinct garbling scheme constructed in step 1. However, using any obfuscator to obfuscate the program that generates each of $\lceil T/S \rceil$ garbled blocks leads to an obfuscated program of size at least $\mathrm{poly}(\lambda, \Phi) = \mathrm{poly}(\lambda, |R|, S)$. In this case, the complexity of the new garbling scheme is not linear in S, and the overhead of the produced garbled RAM is at least $\mathrm{poly}(\lambda, |R|, S)$.

Better efficiency. To avoid the polynomial overhead due to obfuscation, we instead use an $i\mathcal{O}$ for circuits with quasi-linear complexity $|C| \mathrm{poly}(\lambda, n)$, where $|C|$ is the size of the circuit obfuscated and n is the length of the input. As shown in section 5.4, such a scheme can be constructed generically from any $i\mathcal{O}$ (for circuits), puncturable PRF, and randomized encoding that is local (as defined in section 5.4 and satisfied, for instance, by Yao's garbled circuits), all with $2^{-(n+\omega(\log \lambda))}$ security.

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

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

which is linear in the space complexity of R.

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

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

This concludes the proof of Theorem 3.5. ∎

3.3.1. RAM garbling scheme with complexity linear in the program size. The RAM garbling scheme of Theorem 3.5 produces garbled RAM of size $\mathrm{poly}(\lambda, |R|) \times S$ and running time $\mathrm{poly}(\lambda, |R|)T^*$ (for an input of instance complexity T^*), both depending polynomially in the description size of the underlying RAM $|R|$. We show that the complexity can be improved to depending linearly on $|R|$, that is, the garbled RAM has size $\mathrm{poly}(\lambda) \times (|R| + S)$ and running time $\mathrm{poly}(\lambda) \times (|R| + S + T^*)$.

To achieve this, we need to rely on a basic RAM garbling scheme that satisfies the properties—independent input encoding and linear complexity—described above, and the following strengthening: The complexity of the garbling algorithm Gb' depends linearly on $|R|$, that is, $\mathrm{poly}(\lambda)(|R|+T)$ (as opposed to $\mathrm{poly}(\lambda,|R|)T$). To obtain such a basic RAM garbling scheme, we observe that there is a universal RAM M, such that any RAM computation $R(x)$ can be transformed into computing $M^R(x,|R|)$, where the description of R is provided as a part of the initial memory. The universal machine M has constant size and $M^R(x,|R|)$ takes at most $cT_R(x)$ steps for some constant c (since each step of R depends on at most a constant number of bits of the description of R). Then applying the construction of [LO13, GHL$^+$14] to M with a persistent database R yields a garbled RAM of size $\mathrm{poly}(\lambda)(T+|R|)$ (where $\mathrm{poly}(\lambda)T$ corresponds to the size of garbling of M and $\mathrm{poly}(\lambda)|R|$ corresponds to the garbling of the persistent database R).

Now, instantiate the construction of Theorem 3.5 with such a basic RAM garbling scheme, with an additional modification: In step 1, instead of dividing a T-step RAM computation into $\lceil T/S\rceil$ S-step blocks, divide it into $\lceil T/(S+|R|)\rceil$ $(S+|R|)$-step blocks; the rest of the construction follows identically. We now argue that this construction indeed has complexity linear in $|R|$. Each augmented block has the same size as before $\mathrm{poly}(\lambda)+|R|$, but a longer running time of $\mathrm{poly}(\lambda)(S+|R|)$. By the complexity of the (new) basic RAM garbling scheme, each of the garbled blocks has size $\mathrm{poly}(\lambda)(S+|R|)$. Therefore, when obfuscating using an $i\mathcal{O}$ with quasi-linear complexity, the program that produces the garbled blocks, it leads to a new garbled RAM of size $\mathrm{poly}(\lambda)(S+|R|)$. The evaluation of such a garbled RAM with an input x of instance complexity T^* takes time $\lceil T^*/(S+|R|)\rceil \times \mathrm{poly}(\lambda)(S+|R|) = \mathrm{poly}(\lambda)(S+|R|+T^*)$. Since the construction and analysis is essentially the same as in Theorem 3.5, we omit the details here.

4. Succinct garbling via reusable obfuscation. In this section (based on [CHJV15]) we provide alternative proofs for Theorems 3.1 and 3.2. As mentioned in the introduction, the approach here is more direct: We obfuscate the circuit that implements a single computational step of the machine itself. More specifically, this is the circuit that reads the contents of the appropriate cell on the machine's tape, writes a new value to the cell, moves the head to a new tape location, and updates the machine's state according to its transition function. To make sure that the computation remains meaningful and that the evaluator does not learn the intermediate values that the machine writes on its tape, we provide a mechanism for the obfuscated machine to encrypt and authenticate the contents of the tape.

We proceed in three steps. First, in section 4.1, we present a special, **iO**-friendly authenticated encryption scheme, ACE. This scheme, which is a central piece in our solution, will be used to encrypt and authenticate data written to the tape.

Second, in section 4.2 we present and analyze the scheme for garbling Turing machines using **iO** and an ACE scheme.

Finally, in section 4.3 we present and analyze our scheme for garbling RAM machines. The main additional component here is an "**iO**-friendly oblivious RAM" mechanism.

4.1. Special authenticated encryption (ACE). In the following sections we give an alternate method for garbling Turing machines and RAM machines from **iO**. These methods rely heavily on a special authenticated encryption scheme, which we define here and call asymmetrically constrained encryption (ACE). An ACE scheme has the following special properties:

1. Ciphertexts are unique: for each message m and key K, there is at most a

single string that decrypts to m under key K.

2. Both encryption and decryption capabilities under the key K are *puncturable*. That is, let S be a subset of the message space which is decidable by a bounded-size circuit C (i.e., $S = \{m : C(m) = 1\}$). One can generate a "punctured" encryption key EK_C which is only able to encrypt messages m for which $m \notin S$.
   Similarly, one can generate a constrained *decryption* key DK_C which, given an encryption of m, outputs m only if $m \notin S$ (and otherwise outputs $\perp$). We also require that DK_C be indistinguishable from DK, even given certain "allowed" auxiliary information. In particular, they remain indistinguishable even given an *encryption* key which is constrained on a superset of S.

These properties are central in the analysis of our garbling schemes.

4.1.1. Definition.

DEFINITION 4.1. *An asymmetrically constrained encryption (ACE) scheme is a* 5*-tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{GenEK}, \mathsf{GenDK}, \mathsf{Enc}, \mathsf{Dec})$ *satisfying syntax, correctness, security of constrained decryption, and selective indistinguishability of ciphertexts as described below.*

Syntax. The algorithms $(\mathsf{Setup}, \mathsf{GenEK}, \mathsf{GenDK}, \mathsf{Enc}, \mathsf{Dec})$ have the following syntax.

- *Setup.* $\mathsf{Setup}(1^\lambda, 1^n, 1^s)$ is a randomized algorithm that takes as input the security parameter λ, the message length n, and a "circuit succinctness" parameter s, all in unary. Setup then outputs a secret key SK. We think of secret keys as consisting of two parts: an encryption key EK and a decryption key DK.
  Let $\mathcal{M} = \{0,1\}^n$ denote the message space.
- *(Constrained) key generation.* Let $S \subset \mathcal{M}$ be any set whose membership is decidable by a circuit C_S. We say that S is *admissible* if $|C_S| \leq s$. Intuitively, the set size parameter s denotes the upper bound on the size of circuit description of sets to which encryption and decryption keys can be constrained.
  - $\mathsf{GenEK}(SK, C_S)$ takes as input the secret key SK of the scheme and the description of circuit C_S for an admissible set S. It outputs an encryption key $EK\{S\}$. We write EK to denote $EK\{\emptyset\}$.
  - $\mathsf{GenDK}(SK, C_S)$ also takes as input the secret key SK of the scheme and the description of circuit C_S for an admissible set S. It outputs a decryption key $DK\{S\}$. We write DK to denote $DK\{\emptyset\}$.

  Unless mentioned otherwise, we will only consider admissible sets $S \subset \mathcal{M}$.
- *Encryption.* $\mathsf{Enc}(EK', m)$ is a deterministic algorithm that takes as input an encryption key EK' (that may be constrained) and a message $m \in \mathcal{M}$ and outputs a ciphertext c or the reject symbol $\perp$.
- *Decryption.* $\mathsf{Dec}(DK', c)$ is a deterministic algorithm that takes as input a decryption key DK' (that may be constrained) and a ciphertext c and outputs a message $m \in \mathcal{M}$ or the reject symbol $\perp$.

Correctness. An ACE scheme is correct if the following properties hold.

1. *Correctness of decryption.* For all n, all $m \in \mathcal{M}$, and all sets $S, S' \subset \mathcal{M}$ such that $m \notin S \cup S'$,

$$\Pr\left[\mathsf{Dec}(DK, \mathsf{Enc}(EK, m)) = m \;\middle|\; \begin{array}{l} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ EK \leftarrow \mathsf{GenEK}(SK, C_{S'}), \\ DK \leftarrow \mathsf{GenDK}(SK, C_S) \end{array}\right] = 1.$$

   Informally, this says that $\mathsf{Dec} \circ \mathsf{Enc}$ is the identity on messages which are in neither of the punctured sets.
2. *Equivalence of constrained encryption.* Let $SK \leftarrow \mathsf{Setup}(1^\lambda)$. For any message $m \in \mathcal{M}$ and any sets $S, S' \subset \mathcal{M}$ with m not in the symmetric difference $S \Delta S'$ (i.e., we require that m is in both S and S' or m is in neither S nor S'),

$$\Pr\left[\mathsf{Enc}(EK, m) = \mathsf{Enc}(EK', m) \;\middle|\; \begin{array}{l} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ EK \leftarrow \mathsf{GenEK}(SK, C_S), \\ EK' \leftarrow \mathsf{GenEK}(SK, C_{S'}) \end{array}\right] = 1.$$

3. *Unique ciphertexts.* With high probability over $SK \leftarrow \mathsf{Setup}(1^\lambda)$, it holds for any c and c' that if $\mathsf{Dec}(DK, c) = \mathsf{Dec}(DK, c') \neq \perp$, then $c = c'$.
4. *Safety of constrained decryption.* For all strings c and all $S \subset \mathcal{M}$,

$$\Pr\left[\mathsf{Dec}(DK, c) \in S \;\middle|\; SK \leftarrow \mathsf{Setup}(1^\lambda), DK \leftarrow \mathsf{GenDK}(SK, C_S)\right] = 0.$$

   This says that a punctured key $DK\{S\}$ will never decrypt a string c to a message in S.
5. *Equivalence of constrained decryption.* If $\mathsf{Dec}(DK\{S\}, c) = m \neq \perp$ and $m \notin S'$, then $\mathsf{Dec}(DK\{S'\}, c) = m$.

Security of constrained decryption. Intuitively, this property says that for any two sets S_0, S_1, no adversary can distinguish between the constrained key $DK\{S_0\}$ and $DK\{S_1\}$, even given additional auxiliary information in the form of a constrained encryption key EK' and ciphertexts $c_1, \ldots, c_t$. To rule out trivial attacks, EK' is constrained at least on $S_0 \Delta S_1$. Similarly, each c_i is an encryption of a message $m \notin S_0 \Delta S_1$.

Formally, we describe security of constrained decryption as a multistage game between an adversary $\mathcal{A}$ and a challenger.

- *Setup.* $\mathcal{A}$ chooses sets S_0, S_1, U such that $S_0 \Delta S_1 \subseteq U \subseteq \mathcal{M}$ and sends its circuit descriptions (C_{S_0}, C_{S_1}, C_U) to the challenger. $\mathcal{A}$ also sends arbitrary polynomially many messages $m_1, \ldots, m_t$ such that $m_i \notin S_0 \Delta S_1$.
  The challenger chooses a bit $b \in \{0, 1\}$ and computes the following:
  1. $SK \leftarrow \mathsf{Setup}(1^\lambda)$.
  2. $DK\{S_b\} \leftarrow \mathsf{GenDK}(SK, C_{S_b})$.
  3. $EK \leftarrow \mathsf{GenEK}(SK, \emptyset)$.
  4. $c_i \leftarrow \mathsf{Enc}(EK, m_i)$ for every $i \in [t]$.
  5. $EK\{U\} \leftarrow \mathsf{GenEK}(SK, C_U)$.

  Finally, it sends the tuple $(EK\{U\}, DK\{S_b\}, \{c_i\})$ to $\mathcal{A}$.
- *Guess.* $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ in this game (on security parameter λ) is defined as $\mathsf{adv}_\mathcal{A} = \left|\Pr[b' = b] - \frac{1}{2}\right|$. We require that for all PPT $\mathcal{A}$, $\mathsf{adv}_\mathcal{A}(\lambda)$ is negligible in λ.

Selective indistinguishability of ciphertexts. Intuitively, this property says that no adversary can distinguish encryptions of m_0 from encryptions of m_1, even given certain auxiliary information. The auxiliary information corresponds to constrained encryption and decryption keys EK', DK', as well as some ciphertexts $c_1, \ldots, c_t$. In order to rule out trivial attacks, EK' and DK' should both be punctured on at least $\{m_0, m_1\}$, and none of $c_1, \ldots, c_t$ should be an encryption of m_0 or m_1. Let both $\mathcal{F}_1$ and $\mathcal{F}_2$ be subexponentially secure.

Formally, we require that for all sets $S, U \subset \mathcal{M}$, for all $m_0^*, m_1^* \in S \cap U$, and for all $m_1, \ldots, m_t \in \mathcal{M} \setminus \{m_0^*, m_1^*\}$, the distribution

$$EK\{S\}, DK\{U\}, c_0^*, c_1^*, c_1, \ldots, c_t$$

is computationally indistinguishable from

$$EK\{S\}, DK\{U\}, c_1^*, c_0^*, c_1, \ldots, c_t$$

in the probability space defined by sampling $SK \leftarrow \mathsf{Setup}(1^\lambda)$, $EK \leftarrow \mathsf{GenEK}(SK, \emptyset)$, $EK\{S\} \leftarrow \mathsf{GenEK}(SK, C_S)$, $DK\{U\} \leftarrow \mathsf{GenDK}(SK, C_U)$, $c_b^* \leftarrow \mathsf{Enc}(EK, m_b^*)$, and $c_i \leftarrow \mathsf{Enc}(EK, m_i)$.

4.1.2. Construction. We now present a construction of an asymmetrically constrainable encryption scheme $ACE = (\mathsf{Setup}, \mathsf{GenEK}, \mathsf{GenDK}, \mathsf{Enc}, \mathsf{Dec})$. Our scheme is based on **iO** and one-way functions that are subexponentially secure and is based on the "hidden triggers" mechanism in the deniable encryption scheme of [SW14].

$\mathsf{Setup}(1^\lambda, 1^n, 1^s)$: Let $\mathcal{F}_1 = (\mathsf{PRF.Gen}_1, \mathsf{PRF.Punc}_1, \mathsf{F}_1)$ be a subexponentially secure puncturable PRF (PPRF) family with domain $\{0,1\}^n$ and codomain $\{0,1\}^{3n+\lambda}$ such that $\mathcal{F}_1$ is *(statistically) injective* with failure probability $2^{-n-\lambda}$, as defined and constructed in [SW14, Theorem 2]. Let $\mathcal{F}_2 = (\mathsf{PRF.Gen}_2, \mathsf{PRF.Punc}_2, \mathsf{F}_2)$ be a (noninjective) subexponentially secure PPRF family with domain $\{0,1\}^{3n+\lambda}$ and codomain $\{0,1\}^n$.

Let λ' denote a security parameter chosen as a function of λ and n to ensure that any nonuniform $\mathsf{poly}(\lambda)$-size adversary $\mathcal{A}$ has $2^{-n} \cdot \mathrm{negl}(\lambda)$ advantage in the PRF games associated to $\mathcal{F}_1$ and $\mathcal{F}_2$, as well as in the **iO** game associated with $i\mathcal{O}(1^{\lambda'}, \cdot)$. Since we assumed that $\mathcal{F}_1$, $\mathcal{F}_2$, and $i\mathcal{O}$ were subexponentially secure, this is achievable (indeed with *subexponential* negl) with some $\lambda' = \mathsf{poly}(\lambda)$.

Setup samples keys $K_1 \leftarrow \mathsf{PRF.Gen}_1(1^{\lambda'})$ and $K_2 \leftarrow \mathsf{PRF.Gen}_2(1^{\lambda'})$, and outputs (K_1, K_2, p), where $p = \mathsf{poly}(\lambda', n, s)$ is a fixed polynomial "padding" chosen to be larger than $\mathcal{G}_{\mathsf{enc}}$ and $\mathcal{G}_{\mathsf{dec}}$ in each hybrid distribution of the security proof (see Remarks 4.4 and 4.6). We will write F_i to denote the function $\mathsf{F}_i(K_i, \cdot)$.

$\mathsf{GenEK}((K_1, K_2, p), C_S)$: The encryption key generation algorithm takes as input keys K_1, K_2 and the circuit description C_S of an admissible set S. It prepares a circuit representation of $\mathcal{G}_{\mathsf{enc}}$ (Figure 4.1), padded to be of size p. Next, it computes the encryption key $EK\{S\} \leftarrow i\mathcal{O}(\mathcal{G}_{\mathsf{enc}})$ and outputs the result.

$\mathsf{GenDK}((K_1, K_2, p), C_S)$: The decryption key generation algorithm takes as input keys K_1, K_2 and the circuit description C_S of an admissible set S. It prepares a circuit representation of $\mathcal{G}_{\mathsf{dec}}$ (Figure 4.2), padded to be of size p. It then computes the decryption key $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}_{\mathsf{dec}})$ and outputs the result.

$\mathsf{Enc}(EK', m)$: The encryption algorithm simply runs the encryption key program EK' on message m to compute the ciphertext $c \leftarrow EK'(m)$.

Input: Message $m \in \{0,1\}^n$.
Constants: K_1, K_2, circuit C_S.
1. If $m \in S$, output $\perp$.
2. Otherwise, output $\alpha \| \beta$, where $\alpha = F_1(m)$ and $\beta = F_2(\alpha) \oplus m$.

FIG. 4.1. *(Constrained) encryption* $\mathcal{G}_{\mathsf{enc}}$.

Input: Ciphertext $c \in \{0,1\}^{3n+\lambda}$.
Constants: K_1, K_2, circuit C_S.
1. Parse c as $\alpha \| \beta$ with $\beta \in \{0,1\}^n$, and compute $m = F_2(\alpha) \oplus \beta$.
2. If $m \in S$, or if $\alpha \neq F_1(m)$, then output $\perp$.
3. Otherwise, output m.

FIG. 4.2. *(Constrained) decryption* $\mathcal{G}_{\mathsf{dec}}$.

$\mathsf{Dec}(DK', c)$: The decryption algorithm simply runs the decryption key program DK' on the input ciphertext c and returns the output $DK'(c)$.

Remark 4.2. Subexponential security of **iO** and one-way functions was required only for $2^{-n} \cdot \mathrm{negl}(\lambda)$ security against $\mathsf{poly}(\lambda)$-time adversaries with relatively small λ' $(= \mathsf{poly}(\lambda))$. If we only had polynomially secure **iO** and one-way functions, we would need $\lambda' = \lambda \cdot 2^{\epsilon n}$ for any $\epsilon > 0$, which in general could be extremely large.

However, in our applications to garbling polynomial-time Turing machine and RAM computations on inputs of length $\mathsf{poly}(\lambda)$, we use ACE with $n = O(\log \lambda)$. Thus in this case we can achieve the requisite security with $\lambda' = \mathsf{poly}(\lambda)$, assuming only polynomially secure **iO** and one-way functions.

This completes the description of our construction of ACE.

4.1.3. Proof of correctness and security.

Correctness. We first argue correctness:
1. *Correctness of decryption.* This follow directly from the definitions of $\mathcal{G}_{\mathsf{enc}}$ and $\mathcal{G}_{\mathsf{dec}}$ given in Figures 4.1 and 4.2, along with the perfect correctness of $i\mathcal{O}$.
2. *Equivalence of constrained encryption.* This follows directly from the definition of $\mathcal{G}_{\mathsf{enc}}$ and the perfect correctness of $i\mathcal{O}$.
3. *Uniqueness of encryptions.* A ciphertext $c = \alpha \| \beta$ decrypts to $m \neq \perp$ only if $\alpha = F_1(m)$ and $\beta = F_2(\alpha)$. Since F_1 is injective with probability at least $1 - 2^{-\lambda}$, there can be only one ciphertext which decrypts to m.
4. *Safety of constrained decryption.* This follows directly from the definition of $\mathcal{G}_{\mathsf{dec}}$ and the perfect correctness of $i\mathcal{O}$.
5. *Equivalence of constrained decryption.* This follows directly from the definition of $\mathcal{G}_{\mathsf{dec}}$ and the perfect correctness of $i\mathcal{O}$.

Security of constrained decryption. We now prove that ACE satisfies security of constrained decryption.

LEMMA 4.3. *The proposed scheme ACE satisfies security of constrained decryption.*

Proof. Let S_0, S_1, U be arbitrary subsets of $\{0,1\}^n$ such that $S_0 \Delta S_1 \subseteq U$, and let C_{S_0}, C_{S_1}, C_U be their circuit descriptions. Let $m_1, \ldots, m_t$ be arbitrary messages such

that every $m_i \in M \setminus (S_0 \Delta S_1)$. We argue that no PPT distinguisher can distinguish between

$$(EK\{U\}, DK\{S_0\}, \{c_i\})$$

and

$$(EK\{U\}, DK\{S_1\}, \{c_i\})$$

in the probability space defined by sampling the following:

$$\begin{array}{l} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ EK \leftarrow \mathsf{GenEK}(SK, \emptyset), \\ EK\{U\} \leftarrow \mathsf{GenEK}(SK, C_U). \\ \text{For every } i \in [t] \\ \quad c_i := \mathsf{Enc}(EK, m_i). \\ \text{For } b \in \{0,1\} \\ \quad DK\{S_b\} := \mathsf{GenDK}(SK, C_{S_b}). \end{array}$$

Without loss of generality, we will suppose that $S_0 \subseteq S_1$. The general case follows because then

$$\begin{aligned} (EK\{U\}, DK\{S_0\}, \{c_i\}) &\approx_c (EK\{U\}, DK\{S_0 \cap S_1\}, \{c_i\}) \\ &\approx_c (EK\{U\}, DK\{S_1\}, \{c_i\}). \end{aligned}$$

We now prove the lemma by exhibiting a sequence of $|S_1 \setminus S_0| + 1$ hybrid experiments $\mathsf{H}_0, \ldots, \mathsf{H}_{|S_1 \setminus S_0|+1}$, each of which is $2^{-n} \cdot \mathrm{negl}(\lambda)$-indistinguishable from the previous, such that

$$\mathsf{H}_0 \approx_c (EK\{U\}, DK\{S_0\}, \{c_i\})$$

and

$$\mathsf{H}_{|S_1 \setminus S_0|} \approx_c (EK\{U\}, DK\{S_1\}, \{c_i\}).$$

We now proceed to give details. Let u_i denote the lexicographically ith element of $S_1 \setminus S_0$. Throughout the experiments, we will refer to the encryption key and decryption key programs given to the distinguisher as EK' and DK', respectively. Similarly, (unless stated otherwise) we will refer to the unobfuscated algorithms underlying EK' and DK' as $\mathcal{G}'_{\mathsf{enc}}$ and $\mathcal{G}'_{\mathsf{dec}}$, respectively.

HYBRID H_i (FOR $1 \le i \le |S_1 \setminus S_0|$): In the ith hybrid, the decryption key program $\mathcal{G}'_{\mathsf{dec}}$ first checks whether $m \in S_1$ *and* $m \le u_i$. If this is the case, then it simply outputs $\perp$. Otherwise, it behaves in the same manner as $DK\{S_0\}$. The underlying unobfuscated program $\mathcal{G}'_{\mathsf{dec}}$ is described in Figure 4.3.

Therefore, in hybrid H_0, $\mathcal{G}'_{\mathsf{dec}}$ has the same functionality as $DK\{S_0\}$, and in $\mathsf{H}_{|S_1 \setminus S_0|}$, $\mathcal{G}'_{\mathsf{dec}}$ has the same functionality as $DK\{S_1\}$.

Input: Ciphertext $c \in \{0,1\}^{4n+\lambda}$.
Constants: PPRF keys K_1, K_2, circuits C_{S_0}, C_{S_1}.

1. Parse c as $\alpha \| \beta$ with $\beta \in \{0,1\}^n$, and compute $m = F_2(\alpha) \oplus \beta$.
2. If $m \le u_i$ and $m \in S_1$, or $m > u_i$ and $m \in S_0$, or $\alpha \ne F_1(m)$, output $\perp$.
3. Otherwise, output m.

FIG. 4.3. *(Constrained) decryption $\mathcal{G}'_{\mathsf{dec}}$ in hybrid i.*

We now construct a series of intermediate hybrid experiments $\mathsf{H}_{i,0}, \ldots, \mathsf{H}_{i,7}$, where $\mathsf{H}_{i,0}$ is the same as H_i and $\mathsf{H}_{i,7}$ is the same as H_{i+1}. For every j, we will prove that $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ are $2^{-n} \cdot \mathrm{negl}(\lambda)$-indistinguishable, which establishes that H_i and H_{i+1} are also $2^{-n} \cdot \mathrm{negl}(\lambda)$-indistinguishable.

HYBRID $\mathsf{H}_{i,0}$: This is the same as experiment H_i.

HYBRID $\mathsf{H}_{i,1}$: This is the same as experiment $\mathsf{H}_{i,0}$ except that we modify $\mathcal{G}'_{\mathsf{dec}}$ as in Figure 4.4. If the decrypted message m is u_{i+1}, then instead of checking whether $\alpha \neq F_1(m)$ in line 5, $\mathcal{G}'_{\mathsf{dec}}$ now checks whether $\mathsf{Commit}(0;\alpha) \neq \mathsf{Commit}(0; F_1(u_{i+1}))$, where Commit is an injective noninteractive commitment.

Input: Ciphertext $c \in \{0,1\}^{4n+\lambda}$.
Constants: PPRF keys K_1, K_2, circuits C_{S_0}, C_{S_1}, message u_{i+1}, $z = \mathsf{Commit}(0; F_1(u_{i+1}))$.

1. Parse c as $\alpha \| \beta$ with $\beta \in \{0,1\}^n$, and compute $m = F_2(\alpha) \oplus \beta$.
2. If
   - $m \leq u_i$ and $m \in S_1$, or
   - $m > u_i$ and $m \in S_0$, or
   - $m = u_{i+1}$ and $\mathsf{Commit}(0;\alpha) \neq z$, or
   - $\alpha \neq F_1(m)$,

   then output $\perp$.
3. Otherwise, output m.

FIG. 4.4. $\mathcal{G}'_{\mathsf{dec}}$ *in hybrid* $\mathsf{H}_{i,1}$.

HYBRID $\mathsf{H}_{i,2}$: This is the same as experiment $\mathsf{H}_{i,1}$ except that we modify $\mathcal{G}'_{\mathsf{dec}}$ such that the PRF key K_1 in $\mathcal{G}'_{\mathsf{dec}}$ is punctured at u_{i+1}; i.e., K_1 is replaced with $K_1\{u_{i+1}\} \leftarrow \mathsf{Puncture}_{\mathsf{PRF}}(K_1, u_{i+1})$.

HYBRID $\mathsf{H}_{i,3}$: This is the same as experiment $\mathsf{H}_{i,2}$ except that the PRF key K_1 hardwired in $\mathcal{G}'_{\mathsf{enc}}$ is replaced with the same punctured key $K_1\{u_{i+1}\} \leftarrow \mathsf{Puncture}_{\mathsf{PRF}}(K_1, u_{i+1})$ as is used in $\mathcal{G}'_{\mathsf{dec}}$.

HYBRID $\mathsf{H}_{i,4}$: This is the same as experiment $\mathsf{H}_{i,3}$ except that the hardwired value z in $\mathcal{G}'_{\mathsf{dec}}$ is now computed as $\mathsf{Commit}(0; r)$, where r is a randomly chosen string in $\{0,1\}^{3n+\lambda}$.

HYBRID $\mathsf{H}_{i,5}$: This is the same as experiment $\mathsf{H}_{i,4}$ except that the hardwired value z in $\mathcal{G}'_{\mathsf{dec}}$ is now set to $\mathsf{Commit}(1; r)$, where r is a randomly chosen string in $\{0,1\}^{3n+\lambda}$.

HYBRID $\mathsf{H}_{i,6}$: This is the same as experiment $\mathsf{H}_{i,5}$ except that we now modify $\mathcal{G}'_{\mathsf{dec}}$ such that it outputs $\perp$ when the decrypted message m is u_{i+1}. An equivalent description of $\mathcal{G}'_{\mathsf{dec}}$ is that in line 3 it now checks whether $m \leq u_{i+1}$ instead of $m \leq u_i$.

HYBRID $\mathsf{H}_{i,7}$: This is the same as experiment $\mathsf{H}_{i,6}$ except that the PRF key corresponding to F_1 is unpunctured in both $\mathcal{G}'_{\mathsf{enc}}$ and $\mathcal{G}'_{\mathsf{dec}}$. That is, we replace $K_1\{u_{i+1}\}$ with K_1 in both $\mathcal{G}'_{\mathsf{enc}}$ and $\mathcal{G}'_{\mathsf{dec}}$. Note that experiment $\mathsf{H}_{i,7}$ is the same as experiment H_{i+1}.

This completes the description of the hybrid experiments.

Remark 4.4. There is a fixed polynomial poly such that in each hybrid experiment $\mathsf{H}_{i,j}$, $|\mathcal{G}_{\mathsf{enc}}|$ and $|\mathcal{G}_{\mathsf{dec}}|$ are bounded by $\mathsf{poly}(\lambda', n, s)$.

We now argue their $2^{-n} \cdot \mathrm{negl}(\lambda)$-indistinguishability.

Indistinguishability of $\mathsf{H}_{i,0}$ *and* $\mathsf{H}_{i,1}$. Since Commit is injective, the following two checks are equivalent: $\alpha \neq F_1(m)$ and $\mathsf{Commit}(0;\alpha) \neq \mathsf{Commit}(0; F_1(m))$. Thus, the algorithms $\mathcal{G}'_{\mathsf{dec}}$ in $\mathsf{H}_{i,0}$ and $\mathsf{H}_{i,1}$ are functionally equivalent. Therefore, the indistinguishability of $\mathsf{H}_{i,0}$ and $\mathsf{H}_{i,1}$ follows from the security of $i\mathcal{O}$.

Indistinguishability of $\mathsf{H}_{i,1}$ *and* $\mathsf{H}_{i,2}$. Let $\mathcal{G}'_{\mathsf{dec}}$ (resp., $\mathcal{G}''_{\mathsf{dec}}$) denote the unobfuscated algorithms underlying the decryption key program DK' in experiments $\mathsf{H}_{i,1}$ (resp., $\mathsf{H}_{i,2}$). We will argue that $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ are functionally equivalent. The indistinguishability of $\mathsf{H}_{i,0}$ and $\mathsf{H}_{i,1}$ then follows from the security of the indistinguishability obfuscator $i\mathcal{O}$.

Let $c_{i+1} = \alpha_{i+1} \| \beta_{i+1}$ denote the *unique* ciphertext such that $\mathsf{Dec}(DK, c_{i+1}) = u_{i+1}$ (where DK denotes the unconstrained decryption key program). First note that on any input $c \neq c_{i+1}$, both $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ have identical behavior, except that $\mathcal{G}'_{\mathsf{dec}}$ uses the PRF key K_1, while $\mathcal{G}''_{\mathsf{dec}}$ uses the punctured PRF key $K_1\{u_{i+1}\}$. Since the punctured PRF scheme preserves functionality under puncturing, we have that $\mathcal{G}'_{\mathsf{dec}}(c) = \mathcal{G}''_{\mathsf{dec}}(c)$. Now, on input c_{i+1}, after decrypting to obtain u_{i+1}, $\mathcal{G}'_{\mathsf{dec}}$ computes $\mathsf{Commit}(0; F_1(u_{i+1}))$ and then checks whether $\mathsf{Commit}(0;\alpha_{i+1}) \neq \mathsf{Commit}(0; F_1(u_{i+1}))$, whereas $\mathcal{G}''_{\mathsf{dec}}$ simply checks whether $\mathsf{Commit}(0;\alpha_i) \neq z$. But since the value z hardwired in $\mathcal{G}''_{\mathsf{dec}}$ is equal to $\mathsf{Commit}(0; F_1(u_{i+1}))$, we have that $\mathcal{G}'_{\mathsf{dec}}(c_i) = \mathcal{G}''_{\mathsf{dec}}(c_i)$.

Thus we have that $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ are functionally equivalent.

Indistinguishability of $\mathsf{H}_{i,2}$ *and* $\mathsf{H}_{i,3}$. Let $\mathcal{G}'_{\mathsf{enc}}$ denote the circuit such that in $\mathsf{H}_{i,2}$, the encryption key EK' is $i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{enc}})$. $\mathsf{H}_{i,3}$ is obtained from $\mathsf{H}_{i,2}$ by replacing the hard-coded PPRF key K_1 with the punctured key $K_1\{u_{i+1}\}$. By the security of $i\mathcal{O}$, it suffices to show that this is a functionality-preserving change. Since the punctured PRF preserves functionality at unpunctured points, it suffices to show that $\mathcal{G}'_{\mathsf{enc}}$ does not depend on the value $F_1(u_{i+1})$ (this notion of dependence is well-defined because $\mathcal{G}'_{\mathsf{enc}}$ uses F_1 as a black box). This is true simply because $\mathcal{G}'_{\mathsf{enc}}$ evaluates F_1 on u_{i+1} only when its input $m = u_{i+1}$. But when this happens, $\mathcal{G}'_{\mathsf{enc}}$ outputs $\perp$ unconditionally because $u_{i+1} \in S_1 \subseteq U$.

Indistinguishability of $\mathsf{H}_{i,3}$ *and* $\mathsf{H}_{i,4}$. From the security of the punctured PRF, it follows immediately that $\mathsf{H}_{i,3}$ and $\mathsf{H}_{i,4}$ are computationally indistinguishable.

Indistinguishability of $\mathsf{H}_{i,4}$ *and* $\mathsf{H}_{i,5}$. $\mathsf{H}_{i,4}$ and $\mathsf{H}_{i,5}$ are computationally indistinguishable because of the hiding properties of Commit.

Indistinguishability of $\mathsf{H}_{i,5}$ *and* $\mathsf{H}_{i,6}$. Let $\mathcal{G}'_{\mathsf{dec}}$ (resp., $\mathcal{G}''_{\mathsf{dec}}$) denote the unobfuscated algorithms underlying the decryption key program DK' in experiments $\mathsf{H}_{i,5}$ and $\mathsf{H}_{i,6}$. We will argue that with all but negligible probability, $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ are functionally equivalent. The indistinguishability of $\mathsf{H}_{i,5}$ and $\mathsf{H}_{i,6}$ then follows from the security of the indistinguishability obfuscator $i\mathcal{O}$.

Let c_i denote the *unique* ciphertext corresponding to the message u_i. We note that with overwhelming probability, the random string z (hardwired in both $\mathsf{H}_{i,5}$ and $\mathsf{H}_{i,6}$) is not in the image of the PRG. Thus, except with negligible probability, there does not exist an α_i such that $PRG(\alpha_i) = z$. This implies that except with negligible probability, $\mathcal{G}'_{\mathsf{dec}}(c_i) = \perp$. Since $\mathcal{G}''_{\mathsf{dec}}$ also outputs $\perp$ on input c_i and $\mathcal{G}'_{\mathsf{dec}}, \mathcal{G}''_{\mathsf{dec}}$ behave identically on all other input ciphertexts, we have that $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ are functionally equivalent.

Indistinguishability of $\mathsf{H}_{i,6}$ *and* $\mathsf{H}_{i,7}$. This follows in the same manner as the indistinguishability of experiments $\mathsf{H}_{i,2}$ and $\mathsf{H}_{i,3}$. We omit the details. ∎

Selective indistinguishability of ciphertexts. We now prove that *ACE* satisfies indistinguishability of ciphertexts.

LEMMA 4.5. *The proposed scheme ACE satisfies selective indistinguishability of ciphertexts.*

Proof. The proof of the lemma proceeds in a sequence of hybrid distributions where we make indistinguishable changes to $EK\{U\}$, $DK\{S\}$, and the challenge ciphertexts (c_0^*, c_1^*). The "extra" ciphertexts $c_1, \ldots, c_t$ remain unchanged throughout the hybrids.

HYBRID H_0: This is the real-world distribution. For completeness (and to ease the presentation of the subsequent hybrid distributions), we describe the sampling process here. Let $S, U \subset \mathcal{M} = \{0,1\}^n$ be the sets chosen by the adversary, and let C_S, C_U be their corresponding circuit descriptions. Let m_0^*, m_1^* be the challenge messages in $S \cap U$, and let $(m_1, \ldots, m_t)$ be the extra messages in $\{0,1\}^n$. Then the following hold:

1. Sample the PRF keys $K_1 \leftarrow \mathsf{Gen}(1^\lambda)$, $K_2 \leftarrow \mathsf{Gen}(1^\lambda)$, $b \leftarrow \{0,1\}$.
2. For $b \in \{0,1\}$, compute $\alpha_b^* \leftarrow F_1(m_b^*)$, $\gamma_b^* \leftarrow F_2(\alpha_b^*)$, and $\beta_b^* = \gamma_b^* \oplus m_b^*$. Let $c_b^* = \alpha_b^* \| \beta_b^*$.
3. For every $j \in [t]$, compute $\alpha_j \leftarrow F_1(m_j)$, $\gamma_j \leftarrow F_2(\alpha_j)$, and $\beta_j = \gamma_j \oplus m_j$. Let $c_j = \alpha_j \| \beta_j$.
4. Compute $EK\{U\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{enc}})$, where $\mathcal{G}'_{\mathsf{enc}}$ is described in Figure 4.5, but padded to size p.
5. Compute $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{dec}})$, where $\mathcal{G}'_{\mathsf{dec}}$ is described in Figure 4.6, but padded to size p.
6. Output the following tuple:

$$\left(EK\{S\}, DK\{U\}, c_b^*, c_{1-b}^*, c_1, \ldots, c_t\right).$$

Input: Message $m \in \{0,1\}^n$.
Constants: K_1, K_2, circuit C_U.
1. If $m \in U$, output $\perp$.
2. Otherwise, output $\alpha \| \beta$ where $\alpha = F_1(m)$ and $\beta = F_2(\alpha) \oplus m$.

FIG. 4.5. $\mathcal{G}'_{\mathsf{enc}}$ *in hybrid* H_0.

Input: Ciphertext $c \in \{0,1\}^{3n+\lambda}$.
Constants: K_1, K_2, circuit C_S.
1. Parse c as $\alpha \| \beta$ with $\beta \in \{0,1\}^n$, and compute $m = F_2(\alpha) \oplus \beta$.
2. If $m \in S$, or if $\alpha \neq F_1(m)$, then output $\perp$.
3. Otherwise, output m.

FIG. 4.6. $\mathcal{G}'_{\mathsf{dec}}$ *in hybrid* H_0.

HYBRID H_1: Modify $\mathcal{G}'_{\mathsf{enc}}$: the hardwired PRF key K_1 is replaced with a punctured key $K_1\{m_0^*, m_1^*\} \leftarrow \mathsf{Puncture}_{\mathsf{PRF}}(K_1, \{m_0^*, m_1^*\})$.

Input: Ciphertext $c \in \{0,1\}^{3n+\lambda}$.
Constants: $K_1\{m_0^*, m_1^*\}$, K_2, circuit C_S, c_0^*, c_1^*.
1. If $c = c_b^*$ for any $b \in \{0,1\}$, then output $\perp$.
2. Parse c as $\alpha \| \beta$ with $\beta \in \{0,1\}^n$, and compute $m = F_2(\alpha) \oplus \beta$.
3. If $m \in S$, or if $\alpha \neq F_1(m)$, then output $\perp$.
4. Otherwise, output m.

FIG. 4.7. $\mathcal{G}'_{\mathsf{dec}}$ *in hybrid* 3.

HYBRID H_2: Modify $\mathcal{G}'_{\mathsf{dec}}$: the hardwired PRF key K_1 is replaced with a punctured key $K_1\{m_0^*, m_1^*\} \leftarrow \mathsf{Puncture}_{\mathsf{PRF}}(K_1, \{m_0^*, m_1^*\})$.

HYBRID H_3: Modify $\mathcal{G}'_{\mathsf{dec}}$: Perform the following check in the beginning. If input ciphertext $c = c_b^*$ for $b \in \{0,1\}$, then output $\perp$. The modified $\mathcal{G}'_{\mathsf{dec}}$ is described in Figure 4.7.

HYBRID H_4: Modify challenge ciphertexts $c_b^* = \alpha_b^* \| \beta_b^*$: Generate each α_b^* as a truly random string.

HYBRID H_5: Modify $\mathcal{G}'_{\mathsf{enc}}$: the hardwired PRF key K_2 is replaced with a punctured key $K_2\{\alpha_0^*, \alpha_1^*\} \leftarrow \mathsf{Puncture}_{\mathsf{PRF}}(K_2, \{\alpha_0^*, \alpha_1^*\})$.

HYBRID H_6: Modify $\mathcal{G}'_{\mathsf{dec}}$: we change the check performed in line 1 of Figure 4.7. For any input ciphertext $c = \alpha \| \beta$, if $\alpha = \alpha_b^*$ for $b \in \{0,1\}$, then output $\perp$. Note that $\mathcal{G}'_{\mathsf{dec}}$ only has α_b^* hardwired, as opposed to c_b^*. The modified $\mathcal{G}'_{\mathsf{dec}}$ is described in Figure 4.8.

Input: Ciphertext $c \in \{0,1\}^{3n+\lambda}$.
Constants: PPRF keys $K_1\{m_0^*, m_1^*\}$, K_2, circuit C_S, ciphertext prefixes α_0^*, α_1^*.
1. Parse c as $\alpha \| \beta$ with $\beta \in \{0,1\}^n$.
2. If $\alpha = \alpha_b^*$ for any $b \in \{0,1\}$, then output $\perp$.
3. Compute $m = F_2(\alpha) \oplus \beta$.
4. If $m \in S$, or if $\alpha \neq F_1(m)$, then output $\perp$.
5. Otherwise, output m.

FIG. 4.8. $\mathcal{G}'_{\mathsf{dec}}$ *in hybrid* 6.

HYBRID H_7: Modify $\mathcal{G}'_{\mathsf{dec}}$: the hardwired PRF key K_2 is replaced with the same punctured key $K_2\{\alpha_0^*, \alpha_1^*\} \leftarrow \mathsf{Puncture}_{\mathsf{PRF}}(K_2, \{\alpha_0^*, \alpha_1^*\})$ as was used in $\mathcal{G}'_{\mathsf{enc}}$.

HYBRID H_8: Modify challenge ciphertexts $c_b^* = \alpha_b^* \| \beta_b^*$: For $b \in \{0,1\}$, generate β_b^* as a truly random string.

This completes the description of the hybrid experiments.

Remark 4.6. There is a fixed polynomial poly such that in each hybrid experiment $\mathsf{H}_0, \ldots, \mathsf{H}_8$, $|\mathcal{G}_{\mathsf{enc}}|$ and $|\mathcal{G}_{\mathsf{dec}}|$ are bounded by $\mathsf{poly}(\lambda', n, s)$.

We will now first prove indistinguishability of experiments H_i and H_{i+1} for every i. We will then observe that no adversary can guess bit b in the final hybrid H_8 with probability better than $\frac{1}{2}$. This suffices to prove the claim.

Indistinguishability of H_0 *and* H_1. Let $\mathcal{G}'_{\mathsf{enc}}$ denote the circuit such that in H_0,

$EK\{U\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{enc}})$. H_1 is obtained from H_0 by replacing the PPRF key K_1 in $\mathcal{G}'_{\mathsf{enc}}$ by the punctured key $K_1\{m_0^*, m_1^*\}$. By the security of $i\mathcal{O}$, it suffices to show that this change is functionality-preserving. Since the punctured PRF scheme preserves functionality at unpunctured points, it suffices to show that $\mathcal{G}'_{\mathsf{enc}}$'s functionality does not depend on $F_1(m_0^*)$ or $F_1(m_1^*)$. This is true because $\mathcal{G}'_{\mathsf{enc}}$ uses F_1 as a black box and evaluates it at m_0^* or m_1^* only when its input m is m_0^* or m_1^*. But when this happens, $\mathcal{G}'_{\mathsf{enc}}$ outputs $\perp$ (by the check at line 1) no matter what the value of F_1 is.

Indistinguishability of H_1 *and* H_2. Let $\mathcal{G}'_{\mathsf{dec}}$ denote the circuit such that in H_1, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{dec}})$. H_2 is obtained from H_1 by replacing the PPRF key K_1 in $\mathcal{G}'_{\mathsf{dec}}$ with a punctured key $K_1\{m_0^*, m_1^*\}$. Note that due to the check performed in line 2 of Figure 4.6, $\mathcal{G}'_{\mathsf{dec}}$ does not depend on the value of $F_1(m_0^*)$ or $F_1(m_1^*)$: if $m = m_b^*$, then $m \in S$, so $\mathcal{G}_{\mathsf{dec}}$ outputs $\perp$ unconditionally. Then, since the punctured PRF scheme preserves functionality under puncturing, replacing K_1 by $K_1\{m_0^*, m_1^*\}$) is a functionality-preserving change. The indistinguishability of H_1 and H_2 then follows from the security of the indistinguishability obfuscator $i\mathcal{O}$.

Indistinguishability of H_2 *and* H_3. Let $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ denote the circuits such that in H_2, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{dec}})$, and in H_3, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}''_{\mathsf{dec}})$. The only difference between $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ is that $\mathcal{G}''_{\mathsf{dec}}$ outputs $\perp$ if the input ciphertext c is equal to c_0^* or c_1^*. However, due to line 2 of Figure 4.6, $\mathcal{G}'_{\mathsf{dec}}$ already outputs $\perp$ on c_0^* and c_1^*. Thus, $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ are functionally equivalent and the indistinguishability of H_2 and H_3 follows from the security of $i\mathcal{O}$.

Indistinguishability of H_3 *and* H_4. This follows immediately from the security of the punctured PRF family $\mathcal{F}_1$. (Note that each ciphertext $c_1, \ldots, c_t$ can be generated using the punctured PRF key, because they are not encryptions of m_0^* or m_1^*.)

Indistinguishability of H_4 *and* H_5. Note that with overwhelming probability, the random strings α_b^* are not in the range of F_1. Therefore, except with negligible probability, there does not exist a message m such that $F_1(m) = \alpha_b^*$ for $b \in \{0, 1\}$. Since the punctured PRF scheme preserves functionality under puncturing, $\mathcal{G}'_{\mathsf{enc}}$ (using K_2) and $\mathcal{G}''_{\mathsf{enc}}$ (using $K_2\{\alpha_0^*, \alpha_1^*\}$) behave identically on all input messages, except with negligible probability. The indistinguishability of H_4 and H_5 follows from the security of $i\mathcal{O}$.

Indistinguishability of H_5 *and* H_6. Let $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ denote the circuits such that in H_5, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{dec}})$, and in H_6, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}''_{\mathsf{dec}})$. The only difference between $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ is that $\mathcal{G}''_{\mathsf{dec}}$ explicitly outputs $\perp$ whenever the input ciphertext c has α_0^* or α_1^* as a prefix, while $\mathcal{G}'_{\mathsf{dec}}$ does so only when c is exactly $\alpha_0^* \| \beta_0^*$ or $\alpha_1^* \| \beta_1^*$. However, with overwhelming probability, each of the random strings α_b^* is not in the image of F_1. Thus (because of the check that $\alpha = F_1(m)$) $\mathcal{G}'_{\mathsf{dec}}$ also (except with negligible probability) outputs $\perp$ on *every* c with α_0^* or α_1^* as a prefix. As a consequence, $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ are functionally equivalent except with negligible probability, and the indistinguishability of H_5 and H_6 follows from the security of $i\mathcal{O}$.

Indistinguishability of H_6 *and* H_7. Let $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ denote the circuits such that in H_6, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}'_{\mathsf{dec}})$, and in H_7, $DK\{S\} \leftarrow i\mathcal{O}(1^{\lambda'}, \mathcal{G}''_{\mathsf{dec}})$. Note that due to the check performed in line 2 (see Algorithm 4.8), line 3 is not executed in both $\mathcal{G}'_{\mathsf{dec}}$ and $\mathcal{G}''_{\mathsf{dec}}$ whenever the input ciphertext c is of the form $\alpha^* \| \star$. Then, since the punctured PRF scheme preserves functionality under puncturing, $\mathcal{G}'_{\mathsf{dec}}$ (using K_2) and $\mathcal{G}''_{\mathsf{dec}}$ (using $K_2\{\alpha_0^*, \alpha_1^*\}$) are functionally equivalent, and the indistinguishability of H_6 and H_7 follows from the security of $i\mathcal{O}$.

Indistinguishability of H_7 *and* H_8. This follows immediately from the security of the punctured PRF family $\mathcal{F}_2$ (note that with overwhelming probability, each ciphertext $c_1, \ldots, c_t$ can be generated with the punctured F_2, because the $\tilde{\alpha}_b^*$ are chosen randomly).

Finishing the proof. Observe that in experiment H_8, every challenge ciphertext c_i^b consists of independent uniformly random strings $\alpha_b^* \| \beta_b^*$ that information-theoretically hide the bit b. Further, $EK\{U\}$ and $DK\{S\}$ are also independent of bit b. Therefore, the adversary cannot guess the bit b with probability better than $\frac{1}{2}$. ∎

4.2. Garbling Turing machines. In this section, we describe a direct approach to randomized encodings for Turing machines. In section 4.3, we extend this approach to arbitrary RAM machines. We use the following natural construction.

To compute a randomized encoding of a Turing machine M and an input x which runs in space at most S and time at most T, our scheme first samples two authenticated encryption key pairs $(EK_{\mathsf{mem}}, DK_{\mathsf{mem}})$ and $(EK_{\mathsf{state}}, DK_{\mathsf{state}})$. It then generates the randomized encoding in two parts: a garbled machine $\tilde{M}$ and a garbled initial tape $\widetilde{\mathcal{T}}_0$, which we describe below. We assume that M has an *oblivious* access pattern—that is, there is some universal function τ (independent of M and x) so that $\tau(i)$ is the ith address accessed by M on x. Furthermore,

- $\tau(i)$ is computable in time $\mathsf{poly}(\log i)$;
- the function β, where $\beta(i) = \min\{i' : i' < i \wedge \tau(i') = \tau(i)\}$, is also computable in time $\mathsf{poly}(\log i)$.

This assumption is without loss of generality due to a transformation of Pippenger and Fischer [PF79], which shows how to compile any Turing machine into one with such an oblivious access pattern.

The garbled machine. Suppose that M has initial state q_0 and transition function δ. We define the garbled machine $\tilde{M}$ by defining its initial state $\tilde{q}_0$ and its transition function $\tilde{\delta}$, which crucially we give as an *obfuscated* circuit. We define $\tilde{q}_0 = \mathsf{Enc}(EK_{\mathsf{state}}, (0, q_0))$, and define $\tilde{\delta} = i\mathcal{O}(1^\lambda, \hat{\delta})$, where $\hat{\delta}$ is described in Figure 4.9, but padded to a larger size p.

The padding parameter p is $\mathsf{poly}(\lambda) \cdot T^\epsilon$ for arbitrary $\epsilon > 0$, or $\mathsf{poly}(\lambda, \log T)$ if we assume subexponentially secure **iO** and one-way functions.

Input: State ciphertext $\tilde{q}$, memory ciphertext $\tilde{s}$.
Constants: Keys $(EK_{\mathsf{state}}, DK_{\mathsf{state}})$, $(EK_{\mathsf{mem}}, DK_{\mathsf{mem}})$, transition function δ.

1. Decrypt $(t, q) \leftarrow \mathsf{Dec}(DK_{\mathsf{state}}, \tilde{q})$ and decrypt $(t_{\mathsf{mem}}, \mathsf{addr}, s) \leftarrow \mathsf{Dec}(DK_{\mathsf{mem}}, \tilde{s})$.
2. Check that $t_{\mathsf{mem}} - 1 = \beta(t)$ and $\mathsf{addr} = \tau(t)$ and $t \leq T$. If not, output $\perp$.
3. If $\delta(q, s) \in \{\mathsf{accept}, \mathsf{reject}\}$, output it. Otherwise, compute $(q', s') \leftarrow \delta(q, s)$.
4. Encrypt $\tilde{q}' \leftarrow \mathsf{Enc}(EK_{\mathsf{state}}, (t+1, q'))$ and $\tilde{s}' \leftarrow \mathsf{Enc}(EK_{\mathsf{mem}}, (t+1, \mathsf{addr}, s'))$.
5. Output $(\tilde{q}', \tilde{s}')$.

FIG. 4.9. *Transition function $\hat{\delta}$ to be obfuscated.*

The garbled initial tape. We generate a garbled tape $\widetilde{\mathcal{T}}_0$ consisting of S ciphertexts. The ith ciphertext $\widetilde{\mathcal{T}}_0[i]$ is defined as

$$\widetilde{\mathcal{T}}_0[i] = \begin{cases} \mathsf{Enc}(EK_{\mathsf{mem}}, (0, i, x_i)) & \text{if } i < |x|, \\ \mathsf{Enc}(EK_{\mathsf{mem}}, (0, i, \perp)) & \text{otherwise.} \end{cases}$$

Evaluation. On input $\tilde{M} = (\tilde{q}_0, \tilde{\delta})$ and $\widetilde{\mathcal{T}}_0$, $\mathsf{Eval}(\tilde{M}, \widetilde{\mathcal{T}}_0)$ is computed by running $\tilde{M}$ as a Turing machine with initial tape contents $\widetilde{\mathcal{T}}_0$. More precisely, one runs the procedure described in Figure 4.10.

Input: Garbled Turing machine $\tilde{M} = (\tilde{q}_0, \tilde{\delta})$, garbled initial tape $\widetilde{\mathcal{T}}_0 = (\tilde{x}_1, \ldots, \tilde{x}_S)$.

1. Initialize a mutable tape $\widetilde{\mathcal{T}} := \widetilde{\mathcal{T}}_0$.
2. Initialize a variable $\tilde{q} := \tilde{q}_0$.
3. For $t = 1, 2, \ldots, T$:
   (a) Compute $\mathsf{out} \leftarrow \tilde{\delta}(\tilde{q}, \tilde{\mathcal{T}}[\tau(t)])$.
   (b) If $\mathsf{out} \in \{\mathsf{accept}, \mathsf{reject}\}$, output out.
   (c) Otherwise, parse out as $(\tilde{q}', \tilde{s}')$.
   Set $\tilde{q} := \tilde{q}'$ and set $\tilde{\mathcal{T}}[\tau(t)] := \tilde{s}'$.
4. Output $\perp$.

FIG. 4.10. *Procedure for evaluating a garbled Turing machine.*

Remark 4.7. In this section and the next, we achieve a stronger notion of garbling than the one presented in section 2.2. Specifically, we allow the garbling key **key** to be generated ahead of time, so that inputs and machines may syntactically be garbled separately.

4.2.1. Security with virtual black box (VBB) obfuscation: A warmup and sanity check. In this section we argue that the above garbling scheme is secure when instantiated with a VBB obfuscator and an authenticated encryption scheme satisfying existential unforgeability and semantic security. Precisely, we give a sketch of the following proposition.

PROPOSITION 4.8. *For any two machine-input pairs* $(M^{(0)}, x^{(0)})$ *and* $(M^{(1)}, x^{(1)})$ *satisfying* $M^{(0)}(x^{(0)}) = M^{(1)}(x^{(1)})$ *and* $\textsc{Time}(M^{(0)}, x^{(0)}) = \textsc{Time}(M^{(1)}, x^{(1)})$*, it holds that for all PPT oracle algorithms* $\mathcal{A}$*,*

$$\Pr\left[\mathcal{A}^{\tilde{\delta}}(\widetilde{\mathcal{T}}_0, \tilde{q}_0) = b \;\middle|\; \begin{array}{l} b \leftarrow \{0,1\} \\ M \leftarrow M^{(b)}, x \leftarrow x^{(b)} \\ \mathbf{key} \leftarrow \mathsf{Gen}(1^\lambda) \\ (\tilde{\delta}, \tilde{q}_0) \leftarrow \mathsf{Gb}(\mathbf{key}, M) \\ \widetilde{\mathcal{T}}_0 \leftarrow \mathsf{Encode}(\mathbf{key}, x) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Proof sketch. Let $(\tilde{q}_0, \tilde{s}_0), \ldots, (\tilde{q}_{t^*}, \tilde{s}_{t^*})$ denote the queries made by $\mathsf{Eval}(\tilde{M}^{(b)}, \tilde{x})$ to $\tilde{\delta}$. First, we claim that any $\mathcal{A}$ can be simulated by a $\mathcal{B}$ which also queries only $(\tilde{q}_0, \tilde{s}_0), \ldots, (\tilde{q}_{t^*}, \tilde{s}_{t^*})$ but has nearly the same advantage. That is, in the above probability space,

$$\left|\Pr\left[\mathcal{A}^{\tilde{\delta}}(\widetilde{\mathcal{T}}_0, \tilde{q}_0) = b\right] - \Pr\left[\mathcal{B}^{\tilde{\delta}}(\widetilde{\mathcal{T}}_0, \tilde{q}_0) = b\right]\right| \leq \mathrm{negl}(\lambda).$$

This follows because after having queried $(\tilde{q}_0, \tilde{s}_0), \ldots, (\tilde{q}_{i-1}, \tilde{s}_{i-1})$, the set of ciphertexts seen by the adversary only contains one new pair which, when given to $\tilde{\delta}$ as input, causes a non-$\perp$ output (by the checks on line 2 of $\hat{\delta}$). That pair is $(\tilde{q}_i, \tilde{s}_i)$. The existential unforgeability of the authenticated encryption means that with high probability, $\mathcal{A}$ cannot make any other queries for which $\tilde{\delta}$ outputs a non-$\perp$ value.

Let $q_0, \ldots, q_{t^*-1}$ be the internal states of M in an honest execution on x, and let $\mathcal{T}_0, \ldots, \mathcal{T}_{t^*-1}$ be the tape configurations after every step of M. Then the view of $\mathcal{B}$ is simply y $(= M(x))$ along with an oracle which is simulatable given the following ciphertexts:

- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, i, x_i))$ for each $i \in \{0, \ldots, |x| - 1\}$.
- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, i, \perp))$ for each $i \in \{|x|, \ldots, S\}$.
- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{state}}, (i, q_i))$ for each $i \in \{0, \ldots, t^* - 1\}$.
- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (i+1, \tau(i), (\mathcal{T}_{i+1})_{\tau(i)}))$ for each $i \in \{0, \ldots, t^* - 1\}$.

By the semantic security of the authenticated encryption scheme, $\mathcal{B}$ cannot distinguish the case when these ciphertexts are generated from $M = M^{(0)}$ and $x = x^{(0)}$ from the case when $M = M^{(1)}$ and $x = x^{(1)}$. ∎

4.2.2. Security with iO: Technical overview. We now undertake to prove the security of this construction assuming only indistinguishability obfuscation, following the same ideas as the proof of VBB security.

Recall our reasoning for a VBB obfuscator. We first argued (by existential unforgeability) that a computationally bounded adversary could query $\hat{\delta}$ only using the set of ciphertexts it has already accumulated from the outputs of $\hat{\delta}$. In other words, there is a certain "reachable" set of inputs to δ, and functionality differences outside that set are indistinguishable.

We then argued that the information available to an adversary is limited to the computation's output, plus polynomially many ciphertexts of statically chosen messages, which computationally reveal no information.

There are two problems with applying this reasoning for an **iO** obfuscator. First, in contrast to oracle access, the obfuscated code of $\hat{\delta}$ might leak information about the authenticated encryption secret key. An additional difficulty is that two obfuscated programs may be distinguishable even if it is computationally difficult to find an input on which they differ. We will see, nevertheless, that ACE is a particularly "**iO**-friendly" authenticated encryption scheme. We begin with some (much) simpler examples.

iO-friendly authentication. Let us first see how ACE and **iO** can hide functionality differences between two transition functions δ_0 and δ_1 outside of a "reachable" set of inputs. Transition functions are somewhat cumbersome because they take multiple inputs, but we can illustrate the symbiosis between ACE and IO with a couple of toy examples. The first toy example demonstrates how ACE is used together with IO to garble a circuit and an input, for the simple, non-iterated case where the circuit is applied only a single time. The second toy example demonstrates how ACE and IO are used for the extended case where the circuit is applied iteratively a fixed number of times, but the computation performed in all steps but the last one is trivial. Once these examples are in place, we move on to the actual construction and proof.

Toy Example 1.

Informal Claim 4.9. *For any circuit $\delta : \{0,1\}^n \to \{0,1\}$ and input $x^* \in \{0,1\}^n$, $(\tilde{x}^*, \tilde{\delta})$ reveals no more than $(x^*, \delta(x^*))$, where $\tilde{x}^*$ and $\tilde{\delta}$ are sampled according*

to the following procedure:

$$SK \leftarrow ACE.\mathsf{Setup}(1^\lambda, 1^n, 1^{O(n)}),$$
$$EK \leftarrow ACE.\mathsf{GenEK}(SK, \emptyset),$$
$$DK \leftarrow ACE.\mathsf{GenDK}(SK, \emptyset),$$
$$\tilde{x}^* := ACE.\mathsf{Enc}(EK, x^*),$$
$$\hat{\delta} := \textit{the circuit described in Figure } 4.11,$$
$$\tilde{\delta} \leftarrow i\mathcal{O}(1^\lambda, \hat{\delta}).$$

Input: Ciphertext $\tilde{x}$.
Constants: ACE decryption key DK
1. Decrypt $x := \mathsf{Dec}(DK, \tilde{x})$ or output $\perp$ if the output of Dec is $\perp$.
2. Output $\delta(x)$.

FIG. 4.11. *The circuit $\hat{\delta}$ to be obfuscated.*

Proof sketch. Let $\mathcal{A}$ be any PPT algorithm. We give a sequence of changes to the distribution of $\tilde{x}, \tilde{\delta}$ which (assuming the security of ACE and $i\mathcal{O}$) only negligibly affect the probability that $\mathcal{A}$ outputs 1.

1. Instead of sampling
$$DK \leftarrow ACE.\mathsf{GenDK}(SK, \emptyset),$$
sample
$$DK \leftarrow ACE.\mathsf{GenDK}(SK, \{0,1\}^n \setminus \{x^*\}).$$
Here we have written $\{0,1\}^n \setminus \{x^*\}$ as shorthand for the circuit which on input $x \in \{0,1\}^n$ outputs 1 if and only if $x \neq x^*$. The change to DK (and hence to $(\tilde{x}, \tilde{\delta})$) is computationally indistinguishable by the security of constrained decryption.
2. Modify $\hat{\delta}$ so that instead of *computing* $\delta(x)$ when $x = \perp$, it outputs the hard-coded value $\delta(x^*)$. Pad $\hat{\delta}$ so that its size is unchanged. This is a functionality-preserving change by the safety of constrained decryption, and so the change to $(\tilde{x}, \tilde{\delta})$ is computationally indistinguishable by the security of $i\mathcal{O}$.

After applying the last change, the distribution of $(\tilde{x}, \tilde{\delta})$ depends only on x and $\delta(x)$, and thus is a (computationally indistinguishable, efficiently sampleable) simulation of the "real-world" distribution of $(\tilde{x}, \tilde{\delta})$. ∎

Our next example illustrates how to handle circuits which need to decrypt (or in this case simply verify) their own encrypted (authenticated) outputs.

Toy Example 2.

INFORMAL CLAIM 4.10. *For any circuit $f : \{0,1\}^n \to \{0,1\}$ and input $x^* \in \{0,1\}^n$, let $\hat{\delta}$ be the circuit, described formally in Figure 4.12, which "counts to T" before computing f. $(\tilde{x}^*, \tilde{\delta})$ reveals no more than $(x^*, \delta(x^*))$, where $\tilde{x}^*$ and $\tilde{\delta}$ are*

sampled according to the following procedure:

$$SK \leftarrow ACE.\mathsf{Setup}(1^\lambda, 1^n, 1^{O(n+\log(T))}),$$
$$EK \leftarrow ACE.\mathsf{GenEK}(SK, \emptyset),$$
$$DK \leftarrow ACE.\mathsf{GenDK}(SK, \emptyset),$$
$$\tilde{x}^* := ACE.\mathsf{Enc}(EK, (0, x^*)),$$
$$\hat{\delta} := \textit{the circuit described in Figure } 4.12,$$
$$\tilde{\delta} \leftarrow i\mathcal{O}(1^\lambda, \hat{\delta}_b).$$

Input: Ciphertext $\tilde{x}$.
Constants: ACE encryption, decryption key (EK, DK).
1. Decrypt $(i, x) := \mathsf{Dec}(DK, \tilde{x})$, or output $\perp$ if the output of Dec is $\perp$.
2. If $i = T$, output $f(x)$. Otherwise, output $\mathsf{Enc}(EK, (i + 1, x))$.

FIG. 4.12. *The circuit $\hat{\delta}$ to be obfuscated.*

Proof sketch. The set of ciphertexts that should be "reachable" by an adversary given $\tilde{x}^*$ and $\tilde{\delta}$ is just the encryptions of (i, x^*) for $0 \le i < T$. We would like, therefore, to imitate the steps of the proof in our first toy example:

1. Replace DK by the punctured key $DK\{(i, x) : i \ge T \vee x \ne x^*\}$.
2. Modify $\hat{\delta}$ so that on an encryption of (T, x^*) it outputs the *hard-coded* value $f(x^*)$.

However, there is a difficulty with step 1. $\hat{\delta}$ contains both a decryption key *and* an encryption key—thus we cannot indistinguishably puncture the decryption key without first puncturing the encryption key, but neither can we puncture the encryption key without first puncturing the decryption key.

We solve this problem by stratifying the message space by timestamp, and then alternately puncturing the encryption and decryption keys a little at a time. That is, instead of immediately replacing DK by $DK\{(i, x) : i \ge T \vee x \ne x^*\}$, we use a sequence of $O(T)$ different indistinguishable changes to the distribution of $(\tilde{x}^*, \tilde{\delta})$, as follows:

(1a) Change the sampling of EK to $EK \leftarrow ACE.\mathsf{GenEK}(SK, \{(i, x) : i \ge T \vee (i = 0 \wedge x \ne x^*)\})$. This change does not affect the functionality of $\hat{\delta}$ and hence is computationally indistinguishable.

(1b) Change the sampling of DK to $DK \leftarrow ACE.\mathsf{GenDK}(SK, \{(i, x) : i \ge T \vee (i = 0 \wedge x \ne x^*)\})$. By the security of constrained decryption, this is a computationally indistinguishable change to $(\tilde{x}^*, DK, EK)$ and hence to $(\tilde{x}^*, \tilde{\delta})$.

(1c) Repeat steps (a) and (b) alternately, first with "$i = 0$" replaced by "$i \le 1$", then by "$i \le 2$", and so on, up until "$i < T$". Crucially (for ACE's succinctness requirement on the punctured set), the punctured sets are decidable by a circuit of size $O(n + \log T)$. ∎

iO-friendly encryption. We now see how the ciphertext-indistinguishability of ACE allows the garbling of much more general "iterated functions."

Not-So-Toy Example 3.

INFORMAL CLAIM 4.11. *For any circuit $f : \{0,1\}^n \to \{0,1\}^n$ and input $x^* \in \{0,1\}^n$, $(\tilde{x}^*, \tilde{\delta})$ reveals no more than $f^T(x)$, where $\tilde{x}^*$ and $\tilde{\delta}$ are sampled according to*

the following procedure:

$$SK \leftarrow ACE.\mathsf{Setup}(1^\lambda, 1^n, 1^{O(n)}),$$
$$EK \leftarrow ACE.\mathsf{GenEK}(SK, \emptyset),$$
$$DK \leftarrow ACE.\mathsf{GenDK}(SK, \emptyset),$$
$$\tilde{x}^* := ACE.\mathsf{Enc}(EK, (0, x^*)),$$
$$\hat{\delta} := \textit{the circuit described in Figure } 4.13,$$
$$\tilde{\delta} \leftarrow i\mathcal{O}(1^\lambda, \hat{\delta}_b).$$

Input: Ciphertext $\tilde{x}$.
Constants: ACE encryption, decryption key EK, DK, total number of iterations T.

1. Decrypt $(t, x) := \mathsf{Dec}(DK, \tilde{x})$, or output $\perp$ if the output of Dec is $\perp$.
2. If $t = T$, output x. Otherwise output $\mathsf{Enc}(EK, (t+1, f(x)))$.

FIG. 4.13. *The circuit $\hat{\delta}$ to be obfuscated.*

Proof sketch. Suppose that $\hat{\delta}$ were padded to be of size $\Omega(T)$. Then we could show indistinguishability from a simulated distribution by building on the techniques of Toy Example 2:

1. Change the sampling of DK to $DK \leftarrow ACE.\mathsf{GenDK}(SK, \{(i, x) : i \geq T \vee x \neq f^i(x^*)\})$. This change is indistinguishable by the technique of Toy Example 2.
2. Change $\hat{\delta}$ to a circuit that has $f^T(x^*)$ and $c_0, \ldots, c_T$ hard-coded (but not EK or DK), where $c_i = \mathsf{Enc}(EK, (i, f^i(x^*)))$. $\hat{\delta}$ now acts as follows: If the input is c_i for $0 \leq i < T$, output c_{i+1}. If the input is c_T, output $f^T(x^*)$. Otherwise, output $\perp$.
   This change does not affect the functionality of $\hat{\delta}$, so indistinguishability follows directly from the security of $i\mathcal{O}$.
3. Change each ciphertext c_i to be $\mathsf{Enc}(EK, (i, 0))$, and also change $\tilde{x}^*$ to be $\mathsf{Enc}(EK, (i, 0))$. The indistinguishability of this change follows from ACE's ciphertext indistinguishability.

Unfortunately, padding $\hat{\delta}$ to size $\Omega(T)$ would make this construction completely nonsuccinct, and therefore no better than a Yao garbled circuit.

We show a succinct version of this argument which step by step changes $\hat{\delta}$ to a circuit which only does "dummy steps," yielding T intermediate hybrid distributions $\mathsf{H}_1, \ldots, \mathsf{H}_T$. In H_i, $\tilde{x}^*$ is defined as $\mathsf{Enc}(EK, (0, f^i(x^*)))$ and $\hat{\delta}$ is defined as in Figure 4.14.

Input: Ciphertext $\tilde{x}$.
Constants: ACE encryption, decryption key EK, DK, total number of iterations T, hybrid number i.

1. Decrypt $(t, x) := \mathsf{Dec}(DK, \tilde{x})$, or output $\perp$ if the output of Dec is $\perp$.
2. If $t < i$, output $\mathsf{Enc}(EK, (t+1, x))$.
3. Otherwise, if $i \leq t < T$, output $\mathsf{Enc}(EK, (t+1, f(x)))$.
4. Otherwise (if $i = T$), output x.

FIG. 4.14. *The circuit $\hat{\delta}$ to be obfuscated in hybrid H_i.*

To show that $\mathsf{H}_i \approx \mathsf{H}_{i+1}$, we give a sequence of indistinguishable changes, proceeding through i intermediate hybrid distributions $\mathsf{H}_{i,i}, \ldots, \mathsf{H}_{i,0}$. $\mathsf{H}_{i,j}$ is obtained from H_i by modifying $\hat{\delta}$ to output $\mathsf{Enc}(EK, (j+1, f^{i+1}(x^*)))$ when $t = j$, as in Figure 4.15. Now we can indistinguishably change $\mathsf{H}_{i,j}$ to $\mathsf{H}_{i,j-1}$ as follows:

Input: Ciphertext $\tilde{x}$.
Constants: ACE encryption, decryption key EK, DK, total number of iterations T, hybrid number i, j, the value $f^{i+1}(x^*)$.
1. Decrypt $(t, x) := \mathsf{Dec}(DK, \tilde{x})$, or output $\perp$ if the output of Dec is $\perp$.
2. If $t = j$, output $\mathsf{Enc}(EK, (j+1, f^{i+1}(x^*)))$.
3. Otherwise, if $t < i$, output $\mathsf{Enc}(EK, (t+1, x))$.
4. Otherwise, if $i \le t < T$, output $\mathsf{Enc}(EK, (t+1, f(x)))$.
5. Otherwise (if $i = T$), output x.

FIG. 4.15. *The circuit $\hat{\delta}$ to be obfuscated in hybrid $\mathsf{H}_{i,j}$.*

1. Change the sampling of DK to $DK \leftarrow ACE.\mathsf{GenDK}(SK, G^c_{i,j})$, where $G_{i,j}$ is the set of "good" messages, namely, (t, x) for which the following hold:
   - If $t \le j$, then $x = f^i(x^*)$.
   - If $j < t \le i+1$, then $x = f^{i+1}(x^*)$.

   $G^c_{i,j}$ is just the set complement of $G_{i,j}$. The indistinguishability of this change follows from the techniques of Toy Example 2.
2. Change $\tilde{\delta}$ so that if $t = j-1$, it outputs a hard-coded ciphertext $c = \mathsf{Enc}(EK, (j, f^i(x^*)))$, and replace the "if $t = j$" check with "if $\tilde{x} = c$" (to be executed *before* decryption). Also, expand the set on which EK and DK are punctured to include all (t, x) for which $t = j$. These changes do not affect the functionality of $\tilde{\delta}$, so indistinguishability follows from $i\mathcal{O}$.
3. Change c to $\mathsf{Enc}(EK, (j, f^{i+1}(x^*)))$. This is indistinguishable by ACE's ciphertext indistinguishability.
4. Change $\tilde{\delta}$ by removing the "if $\tilde{x} = c$" special case. Change the special-case behavior when $t = j-1$ to computing and outputting $\mathsf{Enc}(EK, (j, f(x)))$ rather than using the hard-coded ciphertext c. Remove the hard-coded ciphertext c altogether. Unpuncture EK and reduce the puncturing of DK to the set $G^c_{i,j-1}$. These changes do not affect the functionality of $\hat{\delta}$, so indistinguishability follows from $i\mathcal{O}$.
5. Apply the changes of step 1 in reverse to indistinguishably unpuncture DK. ∎

We note that when f is the function which maps one Turing machine configuration to the next, Not-So-Toy Example 3 already is a meaningfully succinct garbling for Turing machines. In particular, the size of the garbling is $\mathsf{poly}(S, \lambda)$, and the evaluation time is $T \cdot \mathsf{poly}(S, \lambda)$. Our main construction improves on this by reducing the evaluation time to $T \cdot \mathsf{poly}(\lambda)$ and the garbling size to $S \cdot \mathsf{poly}(\lambda)$. The proof of security for the construction builds on our proof of security for Not-So-Toy Example 3, but with a finer-grained "per-cell" puncturing strategy.

4.2.3. Security with iO: Proof.

THEOREM 4.12. *There is a PPT algorithm* Sim *such that for every Turing ma-*

chine M and input x, running in time T and space S, the distribution

$$\tilde{\delta}, \tilde{q}_0, \widetilde{\mathcal{T}}_0 \left| \begin{array}{l} \mathbf{key} \leftarrow \mathsf{Gen}(1^\lambda, T, S), \\ \tilde{\delta}, \tilde{q}_0 \leftarrow \mathsf{Gb}(\mathbf{key}, M), \\ \widetilde{\mathcal{T}}_0 \leftarrow \mathsf{Encode}(\mathbf{key}, x) \end{array} \right. \tag{4.1}$$

is computationally indistinguishable from

$$\mathsf{Sim}(1^\lambda, M(x), 1^T, 1^S).$$

Proof. We start with a generalization of the techniques given in the technical overview (specifically the technique used to show that $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ were indistinguishable in Not-So-Toy Example 3). Let $\delta : Q \times \Sigma \to Q \times \Sigma \cup \{\mathsf{accept}, \mathsf{reject}, \perp\}$ be a circuit.

DEFINITION 4.13. *Say that δ is* timed *if for every $q \in Q$ and $s \in \Sigma$ there are associated timestamps $q.t$ and $s.t$ such that* (i) *if $(q', s') = \delta(q, s)$, then $q'.t = s'.t = q.t+1$, and* (ii) *if $s.t > q.t$, then $\delta(q, s) = \perp$.*

DEFINITION 4.14. *Say that a timed circuit δ is* T-bounded *if for all q and s with $q.t > T$ it holds that $\delta(q, s) = \perp$.*

DEFINITION 4.15. *A pair of sets $G_{\mathsf{state}} \subseteq Q$ and $G_{\mathsf{mem}} \subseteq \Sigma$ is said to be δ-invariant if*

$$\delta(G_{\mathsf{state}} \times G_{\mathsf{mem}}) \cap (Q \times \Sigma) \subseteq G_{\mathsf{state}} \times G_{\mathsf{mem}}.$$

LEMMA 4.16. *Let $\delta_0, \delta_1 : Q \times \Sigma \to Q \times \Sigma \cup \{\mathsf{accept}, \mathsf{reject}, \perp\}$ be T-bounded timed circuits, and let $(G_{\mathsf{state}}, G_{\mathsf{mem}})$ be δ_0-invariant, with G_{state} and G_{mem} decidable by circuits of size, respectively, $\mathsf{poly}(\log |Q|)$ and $\mathsf{poly}(\log |\Sigma|)$.*

Suppose there exist $q_0^* \in G_{\mathsf{state}}$, $q_1^* \in Q \setminus G_{\mathsf{state}}$ and $s_0^* \in G_{\mathsf{mem}}$, $s_1^* \in \Sigma \setminus G_{\mathsf{mem}}$ such that for all $q \in G_{\mathsf{state}}$ and $s \in G_{\mathsf{mem}}$, $(q', s') = \delta_0(q, s)$ if and only if

$$(\tau_{\mathsf{state}}(q'), \tau_{\mathsf{mem}}(s')) = \delta_1(\tau_{\mathsf{state}}(q), \tau_{\mathsf{mem}}(s)),$$

where $\tau_{\mathsf{state}} : Q \to Q$ denotes the transposition interchanging q_0^* and q_1^*, and $\tau_{\mathsf{mem}} : \Sigma \to \Sigma$ denotes the transposition interchanging s_0^* and s_1^*.

Then for any $q_0 \in G_{\mathsf{state}}$ and any $s_0^{(0)}, \ldots, s_0^{(S-1)} \in G_{\mathsf{mem}}$, it holds that $\left(\vec{c}_0, \tilde{\delta}_0\right)$ is computationally indistinguishable from $\left(\vec{c}_1, \tilde{\delta}_1\right)$ in the probability space defined by sampling

$$\begin{array}{l} SK_{\mathsf{state}} \leftarrow ACE.\mathsf{Setup}(1^\lambda, 1^{\log |Q|}, 1^{\mathsf{poly}(\log |Q|)}), \\ EK_{\mathsf{state}} \leftarrow ACE.\mathsf{GenEK}(SK_{\mathsf{state}}, \emptyset), \\ DK_{\mathsf{state}} \leftarrow ACE.\mathsf{GenDK}(SK_{\mathsf{state}}, \emptyset), \\ SK_{\mathsf{mem}} \leftarrow ACE.\mathsf{Setup}(1^\lambda, 1^{\log |\Sigma|}, 1^{\mathsf{poly}(\log |\Sigma|)}), \\ EK_{\mathsf{mem}} \leftarrow ACE.\mathsf{GenEK}(SK_{\mathsf{mem}}, \emptyset), \\ DK_{\mathsf{mem}} \leftarrow ACE.\mathsf{GenDK}(SK_{\mathsf{mem}}, \emptyset), \\ \vec{c}_0 := \left(\mathsf{Enc}(EK_{\mathsf{state}}, q_0), \left(\mathsf{Enc}(EK_{\mathsf{mem}}, s_0^{(i)})\right)_{i=0}^{S-1}\right), \\ \vec{c}_1 := \left(\mathsf{Enc}(EK_{\mathsf{state}}, \tau_{\mathsf{state}}(q_0)), \left(\mathsf{Enc}(EK_{\mathsf{mem}}, \tau_{\mathsf{mem}}(s_0^{(i)}))\right)_{i=0}^{S-1}\right), \\ \textit{for each } b \in \{0,1\}\textit{:} \\ \quad \tilde{\delta}_b \leftarrow i\mathcal{O}(1^\lambda, \hat{\delta}_b), \end{array}$$

where $\hat{\delta}_b$ is the circuit described in Figure 4.16*, padded to size $\mathsf{poly}(\lambda, \log |Q|, \log |\Sigma|)$.*

Input: Ciphertexts $\tilde{q}$, $\tilde{s}$.
Constants: ACE encryption, decryption keys EK_{state}, DK_{state}, EK_{mem}, DK_{mem}.

1. Decrypt $q := \mathsf{Dec}(DK_{\mathsf{state}}, \tilde{q})$ and $s := \mathsf{Dec}(DK_{\mathsf{mem}}, \tilde{s})$, or output $\perp$ if the output of Dec is $\perp$.
2. Compute $\mathsf{out} := \delta_b(q, s)$. If $\mathsf{out} \in \{\mathsf{accept}, \mathsf{reject}, \perp\}$, output out. Otherwise parse out as (q', s').
3. Output $(\tilde{q}', \tilde{s}')$, where $\tilde{q}' = \mathsf{Enc}(EK_{\mathsf{state}}, q')$ and $\tilde{s}' = \mathsf{Enc}(EK_{\mathsf{mem}}, s')$.

FIG. 4.16. *The circuit* $\hat{\delta}_b$ *in Lemma* 4.16.

Proof. We show this by a sequence of $O(T)$ computationally indistinguishable changes which change $(\vec{c}_0, \tilde{\delta}_0)$ to $(\vec{c}_1, \tilde{\delta}_1)$. First, we iteratively restrict $EK_{\mathsf{state}}, DK_{\mathsf{state}}$ and $EK_{\mathsf{mem}}, DK_{\mathsf{mem}}$ as follows:

1. Restrict EK_{state} (i.e., change the set at which it is punctured) so that

$$\mathsf{Enc}(EK_{\mathsf{state}}, q) \neq \perp \text{ if and only if } q.t > 0 \text{ or } q \in G_{\mathsf{state}}.$$

   Similarly, restrict EK_{mem} so that

$$\mathsf{Enc}(EK_{\mathsf{mem}}, s) \neq \perp \text{ if and only if } s.t > 0 \text{ or } s \in G_{\mathsf{mem}}.$$

   Because $\hat{\delta}_0$ never encrypts messages with timestamp 0, these changes to EK_{state} and EK_{mem} are functionality-preserving, and thus by the security of $i\mathcal{O}$ the overall change to $\tilde{\delta}$ is computationally indistinguishable.
2. For $j = 1, \ldots, T$, do the following:
   (a) Modify DK_{state} and DK_{mem}, such that
      - if q is in the image of $\mathsf{Dec}(DK_{\mathsf{state}}, \cdot)$, then $q.t > j - 1$ or $q \in G_{\mathsf{state}}$;
      - if s is in the image of $\mathsf{Dec}(DK_{\mathsf{mem}}, \cdot)$, then $s.t > j - 1$ or $s \in G_{\mathsf{mem}}$.

      The indistinguishability of this change follows from ACE's security of constrained decryption.
   (b) Restrict EK_{state} and EK_{mem} analogously to step 1, namely,
      - $\mathsf{Enc}(EK_{\mathsf{state}}, q) \neq \perp$ if and only if $q.t > j$ or $q \in G_{\mathsf{state}}$;
      - $\mathsf{Enc}(EK_{\mathsf{mem}}, s) \neq \perp$ if and only if $s.t > j$ or $s \in G_{\mathsf{mem}}$.

      The functional equivalence (and hence, by **iO**, indistinguishability) of this change follows from the δ_0-invariance of $(G_{\mathsf{state}}, G_{\mathsf{mem}})$.

By the T-boundedness of δ_0, EK_{state} and DK_{state} are now punctured at $Q \setminus G_{\mathsf{state}}$, while EK_{mem} and DK_{mem} are now punctured at $\Sigma \setminus G_{\mathsf{mem}}$. Let $\tilde{q}_b^*$ denote $\mathsf{Enc}(EK_{\mathsf{state}}, q_b^*)$, and let $\tilde{s}_b^*$ denote $\mathsf{Enc}(EK_{\mathsf{mem}}, s_b^*)$.

1. Modify $\hat{\delta}_0$ so that instead of encrypting q_0^* (resp., decrypting $\tilde{q}_0^*$), it uses the hard-coded correspondence $q_0^* \leftrightarrow \tilde{q}_0^*$. Similarly instead of encrypting s_0^* (resp. decrypting $\tilde{s}_0^*$), it uses the hard-coded correspondence $s_0^* \leftrightarrow \tilde{s}_0^*$. Let EK_{state} and DK_{state} be punctured now on $(Q \setminus G_{\mathsf{state}}) \cup \{q_0^*\}$, and let EK_{mem} and DK_{mem} be punctured on $(\Sigma \setminus G_{\mathsf{mem}}) \cup \{s_0^*\}$. These changes are clearly functionality-preserving and hence indistinguishable by the security of $i\mathcal{O}$.
2. Change $\tilde{q}_0^*$ to $\mathsf{Enc}(EK_{\mathsf{state}}, q_1^*)$, and similarly change $\tilde{s}_0^*$ to $\mathsf{Enc}(EK_{\mathsf{mem}}, s_1^*)$. Similarly modify all ciphertexts in $\vec{c}$. This is indistinguishable by ACE's ciphertext indistinguishability.
3. Replace δ_0 by δ_1 and remove the hard-coded correspondences $q_0^* \leftrightarrow \tilde{q}_0^*$ and $s_0^* \leftrightarrow \tilde{s}_0^*$. Reduce the puncturing of EK_{state} and DK_{state} to $Q \setminus \tau_{\mathsf{state}}(G_{\mathsf{state}})$,

and reduce the puncturing of EK_{mem} and DK_{mem} to $\Sigma \setminus \tau_{\mathsf{mem}}(G_{\mathsf{mem}})$. By assumption on the relation of δ_1 to δ_0, this change is functionality-preserving and hence indistinguishable by the security of $i\mathcal{O}$.

Finally, remove all of the key restrictions and hard-codings. This change is indistinguishable by reversing the above hybrids. ∎

We now return to the proof of Theorem 4.12. We present a number of indistinguishable hybrid distributions $\mathsf{H}_0, \ldots, \mathsf{H}_T$, each defining a different way of sampling $\tilde{\delta}$, $\tilde{q}_0$, and $\widetilde{\mathcal{T}_0}$, where $\tilde{\delta}$ is sampled as $\tilde{\delta} \leftarrow i\mathcal{O}(1^\lambda, \hat{\delta})$ for some $\hat{\delta}$. In hybrid H_0, $(\hat{\delta}, \tilde{q}_0, \widetilde{\mathcal{T}_0})$ are sampled as in (4.1), while in hybrid H_T they are sampled given only $M(x)$, T, and S. Thus, if we show that for all $i \in \{0, \ldots, T-1\}$, $\mathsf{H}_i \approx \mathsf{H}_{i+1}$, then hybrid H_T defines the desired Sim.

All of our hybrids are designed around the transition function $\hat{\delta}$. We first describe the sorts of transition functions we will define. $\hat{\delta}$ always takes the following as input:

- $\tilde{q}$, expected to be an encryption of (t, q) for some timestamp t and state q.
- $\tilde{s}$, expected to be an encryption of $(t_{\mathsf{mem}}, \mathsf{addr}, s)$ for some different timestamp t_{mem}, an address tag addr, and an underlying memory symbol s.

On an input $(\tilde{q}, \tilde{s})$, $\hat{\delta}$ decrypts

$$(t, q) \leftarrow \mathsf{Dec}(DK_{\mathsf{state}}, \tilde{q}),$$
$$(t_{\mathsf{mem}}, \mathsf{addr}, s') \leftarrow \mathsf{Dec}(DK_{\mathsf{mem}}, \tilde{s})$$

and checks that $t < T$, $t_{\mathsf{mem}} = \beta(t)$, and $\mathsf{addr} = \tau(t)$. If not, then $\hat{\delta}$ outputs $\perp$. If so, $\hat{\delta}$ does one of three things, depending on the hybrid distribution as well as on the timestamp t:

1. *Dummy step*: Only update the timestamps. That is, output $(\tilde{q}', \tilde{s}')$, where

$$\tilde{q}' = \mathsf{Enc}(EK_{\mathsf{state}}, (t+1, q)),$$
$$\tilde{s}' = \mathsf{Enc}(EK_{\mathsf{mem}}, (t+1, \mathsf{addr}, s)).$$

2. *Real step*: Perform one step of M's computation. That is, compute $\mathsf{out} \leftarrow \delta(q, s)$. If out is a "final answer," i.e., $\mathsf{out} \in \{\mathsf{accept}, \mathsf{reject}\}$, then output out in the clear. Otherwise, parse out as (q', s') and output $(\tilde{q}', \tilde{s}')$, where

$$\tilde{q}' = \mathsf{Enc}(EK_{\mathsf{state}}, (t+1, q')),$$
$$\tilde{s}' = \mathsf{Enc}(EK_{\mathsf{mem}}, (t+1, \mathsf{addr}, s')).$$

3. *Hard-coded output*: Output some hard-coded constant y in the clear.

With this terminology, we can crisply describe our hybrids. Let t^* denote the running time of M on x. Let q_t denote the tth internal state of M when executed on x for t steps. Similarly, let $\mathcal{T}_t$ denote the tape of M when executed on x for t steps.

HYBRID H_i (IF $i < t^*$):

- If $t < i$, $\hat{\delta}$ does a dummy step. If $i \le t \le T$, $\hat{\delta}$ does a real step.
- $\tilde{q}_0$ is defined as $\mathsf{Enc}(EK_{\mathsf{state}}, (0, q_i))$.
- $\widetilde{\mathcal{T}_0}$ is defined as $(c_1, \ldots, c_S)$, where $c_j = \mathsf{Enc}(EK_{\mathsf{mem}}, (0, j, \mathcal{T}_i[j]))$.

HYBRID H_i (IF $t^* \le i \le T$):

- If $t < t^*$, $\hat{\delta}$ does a dummy step. If $t = t^*$, $\hat{\delta}$ outputs the hard-coded $M(x)$. If $t^* < t \le i$, $\hat{\delta}$ does a dummy step. If $i < t \le T$, $\hat{\delta}$ does a real step.
- $\tilde{q}_0$ is defined as $\mathsf{Enc}(EK_{\mathsf{state}}, (0, \perp))$.

- $\widetilde{\mathcal{T}}_0$ is defined as $(c_1, \ldots, c_S)$, where $c_j = \mathsf{Enc}(EK_{\mathsf{mem}}, (0, j, \mathcal{T}_{t^*}[j]))$.

HYBRID H_{T+1}:

- If $t < t^*$, $\hat{\delta}$ does a dummy step. If $t = t^*$, $\hat{\delta}$ outputs the hard-coded $M(x)$. If $t^* < t$, $\hat{\delta}$ does a dummy step.
- $\tilde{q}_0$ is defined as $\mathsf{Enc}(EK_{\mathsf{state}}, (0, \perp))$.
- $\widetilde{\mathcal{T}}_0$ is defined as $(c_1, \ldots, c_S)$, where $c_j = \mathsf{Enc}(EK_{\mathsf{mem}}, (0, j, \perp))$.

Indistinguishability of hybrids H_i and H_{i+1} (for $i < t^*$). We will show that H_i and H_{i+1} are indistinguishable by following our earlier proof template using ACE. We give a sequence of subhybrids.

HYBRID $\mathsf{H}_{i,1}$: This is the same as hybrid H_i, but EK_{mem}, DK_{mem} and EK_{state},DK_{state} are, respectively, punctured at $\Sigma \setminus G^i_{\mathsf{mem}}$ and $Q \setminus G^i_{\mathsf{state}}$, where

$$G^i_{\mathsf{mem}} = \{(t, a, s) : \text{If } t \leq i \text{ and } a = \tau(i), \text{ then } s = \mathcal{T}_i[\tau(i)]\}$$

and

$$G^i_{\mathsf{state}} = \{(t, q) : \text{If } t \leq i \text{ then } q = q_i\}.$$

HYBRID $\mathsf{H}_{i,2}$: Add the following special-case branch to $\hat{\delta}$: If $t = i$, output $(\tilde{q}^*, \tilde{s}^*)$, where these are hard-coded constants

$$\tilde{q}^* = \mathsf{Enc}(EK_{\mathsf{state}}, (i + 1, q_{i+1}))$$

and

$$\tilde{s}^* = \mathsf{Enc}(EK_{\mathsf{mem}}, (i + 1, \tau(i), \mathcal{T}_{i+1}[\tau(i)])).$$

$\hat{\delta}$ is now described by Figure 4.17.

HYBRID $\mathsf{H}_{i,3}$: In this step, we change the time at which the special-case branch above is activated. That is, for j ranging from i to 0, we have a hybrid $\mathsf{H}_{i,j}$ in which the transition function $\hat{\delta}$ instead has the following special-case branches. Let $\gamma(j)$ denote $\min\{j' : j' \geq j \wedge \tau(j') = \tau(i)\}$:

(a) If $t = j$, set $\tilde{q}' = \tilde{q}^*$, where $\tilde{q}^*$ is the hard-coded value $\mathsf{Enc}(EK_{\mathsf{state}}, (j + 1, q_{i+1}))$. Otherwise, define $\tilde{q}'$ as in a dummy step.

(b) If $t = \gamma(j)$, then set $\tilde{s}' = \tilde{s}^*$, where $\tilde{s}^*$ is the hard-coded value

$$\mathsf{Enc}(EK_{\mathsf{mem}}, (j + 1, \tau(i), (\mathcal{T}_{i+1})[\tau(i)])).$$

Otherwise, define $\tilde{s}'$ as in a dummy step.

CLAIM 4.17. *The distribution of $(\hat{\delta}, \tilde{q}_0, \widetilde{\mathcal{T}}_0)$ induced by hybrid $\mathsf{H}_{i,2}$ is indistinguishable from that induced by $\mathsf{H}_{i,3}$.*

Proof. There are really two changes here: changing the time at which branch (a) is activated (together with the ciphertext c_{state}), and changing the time at which branch (b) is activated (together with the ciphertext c_{mem}). Although the two are treated analogously, we only illustrate with branch (a) for simplicity because its hard-coded time always changes from j to $j - 1$.

We first modify branch (a) to have two parts:

(a.1) If $t = j - 1$, set $\tilde{q}' = c_{\mathsf{state}}$, where $c_{\mathsf{state}} = \mathsf{Enc}(EK_{\mathsf{state}}, (j, q_i))$.

(a.2) If $\tilde{q} = c_{\mathsf{state}}$, set $\tilde{q}' = \tilde{q}^*$, where $\tilde{q}^* = \mathsf{Enc}(EK_{\mathsf{state}}, (j + 1, q_{i+1}))$.

This change is indistinguishable by **iO**. Next, we restrict EK_{state} and DK_{state} to be unable to encrypt or decrypt anything with timestamp j. This is also indistinguishable by **iO** because

Input: State ciphertext $\tilde{q}$, memory ciphertext $\tilde{s}$.
Constants: Keys $EK'_{\mathsf{state}} = EK_{\mathsf{state}}\{Q \setminus G_{\mathsf{state}}\}$, $DK'_{\mathsf{state}} = DK_{\mathsf{state}}\{Q \setminus G_{\mathsf{state}}\}$, $EK'_{\mathsf{mem}} = EK_{\mathsf{mem}}\{\Sigma \setminus G_{\mathsf{mem}}\}$, $DK'_{\mathsf{mem}} = DK_{\mathsf{mem}}\{\Sigma \setminus G_{\mathsf{mem}}\}$, ciphertexts $\tilde{q}^*$, and $\tilde{s}^*$.

1. Decrypt $(t, q) \leftarrow \mathsf{Dec}(DK'_{\mathsf{state}}, \tilde{q})$ and decrypt $(t_{\mathsf{mem}}, \mathsf{addr}, s) \leftarrow \mathsf{Dec}(\mathsf{sk}'_{\mathsf{mem}}, \tilde{s})$.
2. Check that $t_{\mathsf{mem}} = \beta(t)$ and $\mathsf{addr} = \tau(t)$ and $t \le T$. If not, output $\perp$.
3. If $t = i$, output $(\tilde{q}^*, \tilde{s}^*)$.
4. If $\delta(q, s) \in \{\mathsf{accept}, \mathsf{reject}\}$, output it. Otherwise, compute $(q', s') \leftarrow \delta(q, s)$.
5. Encrypt $\tilde{q}' \leftarrow \mathsf{Enc}(EK'_{\mathsf{state}}, (t + 1, q'))$ and $\tilde{s}' \leftarrow \mathsf{Enc}(\mathsf{sk}'_{\mathsf{mem}}, (t + 1, \mathsf{addr}, s'))$.
6. Output $(\tilde{q}', \tilde{s}')$.

FIG. 4.17. *Obfuscated transition function $\hat{\delta}$ with punctured keys and hard-coded ciphertexts to output.*

- $\hat{\delta}$ never encrypts something with timestamp j; it outputs c_{state} instead;
- since DK_{state} was already restricted to G^i_{state}, $\mathsf{Dec}(DK_{\mathsf{state}}, c)$ would be of the form $(j, \star)$ if and only if $c = c_{\mathsf{state}}$, but $\hat{\delta}$ doesn't ever decrypt c_{state}.

Next, we change the hard-coded value c_{state} to be equal to $\mathsf{Enc}(EK_{\mathsf{state}}, (j, q_{i+1}))$. This is indistinguishable due to ciphertext indistinguishability of ACE.

We now undo the hard-coding and key restriction we did to facilitate this change:

1. $\hat{\delta}$ is modified so that when $t = j$, it computes $\tilde{q}'$ as in a dummy step instead of hard-coding $\tilde{q}^*$. This is indistinguishable because, by the puncturing of DK_{state}, the only reachable case (mapping c_{state} to $\tilde{q}^*$) is functionally equivalent to a dummy step.
2. We unrestrict EK_{state} and $DK_{\mathsf{state}}$to allow encryption and decryption of (j, q_{i+1}) and remove special-case branch (a.2). This is functionally equivalent, and thus indistinguishable by **iO**.
3. We then unrestrict DK_{state} to allow decryption of any message with timestamp j. This is indistinguishable by ACE's security of constrained decryption.
4. We then unrestrict EK_{state} to allow encryption of any message with timestamp j. This is indistinguishable by **iO** because $\hat{\delta}$ only tries to encrypt (j, q) when $q = q_{i+1}$.

The result of these changes is hybrid $\mathsf{H}_{i,j-1}$, so we have shown that for $j > 1$, $\mathsf{H}_{i,j} \approx \mathsf{H}_{i,j-1}$. We must also show that $\mathsf{H}_{i,1} \approx \mathsf{H}_{i+1}$. The proof is exactly analogous, and hybrid H_{i+1} should be thought of as a degenerate hybrid $\mathsf{H}_{i,0}$. Instead of modifying $\hat{\delta}$ to have a hard-coded output when $t = 0$ ($\hat{\delta}$ is never run with $t = 0$—its first timestamp is $t = 1$), one modifies the initial state $\tilde{q}_0$ and the initial tape $\widetilde{\mathcal{T}}_0$. ∎

Indistinguishability of hybrids H_{t^*-1} and H_{t^*}. This proceeds analogously to the proof that $\mathsf{H}_{i-1} \approx \mathsf{H}_i$ for $i < t^*$. The only difference is that the hard-coded behavior at time t^* is to output some hard-coded $y \in \{\mathsf{accept}, \mathsf{reject}\}$ rather than to output two hard-coded ciphertexts $(c_{\mathsf{state}}, c_{\mathsf{mem}})$. There is also no need to go through hybrids $\mathsf{H}_{i,i}$ to $\mathsf{H}_{i,0}$ to change $\tilde{q}_0$ (resp., $\widetilde{\mathcal{T}}_0$) from $\mathsf{Enc}(EK_{\mathsf{state}}, (0, q_{t^*}))$ to $\mathsf{Enc}(EK_{\mathsf{state}}, (0, q_{t^*+1}))$; since the running time of M on x is only t^*, there is no q_{t^*+1}.

Indistinguishability of hybrids H_{t^*} *and* H_T. We use the fact that in H_{t^*}'s definition of $\hat{\delta}$, the procedure $\mathsf{Enc}(EK_{\mathsf{state}}, \cdot)$ is never invoked on a message of the form $(t^* + 1, \star)$. We then iteratively restrict EK_{state} and DK_{state} so that for any $t \in \{t^* + 1, \ldots, T\}$, any message of the form $(t, \star)$ is not in the image of $\mathsf{Dec}(DK_{\mathsf{state}}, \cdot)$. Then modifying $\hat{\delta}$ to run a dummy step on all such t preserves functionality and is therefore indistinguishable.

Indistinguishability of hybrids H_T *and* H_{T+1}. We need to indistinguishably replace $\tilde{q}_0$ and $\widetilde{\mathcal{T}}_0$ with null ciphertexts. This will require $T \cdot S$ more hybrids. Due to its similarity to the lengthy argument described above, we only sketch these hybrids here.

We change the initial ciphertexts one at a time. For example, we might change $\tilde{q}_0$ first, and then change $\widetilde{\mathcal{T}}_0$, changing addresses from lowest to highest. But the exact order isn't important. Suppose that we want to change $\widetilde{\mathcal{T}}_0[0]$ to be $\mathsf{Enc}(EK_{\mathsf{mem}}, (0, 0, \perp))$. We go through hybrids $\mathsf{H}_{T,1}, \ldots, \mathsf{H}_{T,t^*}$. In each of these hybrids,

$$\widetilde{\mathcal{T}}_0[0] = \mathsf{Enc}(EK_{\mathsf{mem}}, (0, 0, \perp)),$$

but only in $\mathsf{H}_{T,T}$ will $\hat{\delta}$ be independent of the old value of $\widetilde{\mathcal{T}}_0$.

In hybrid $\mathsf{H}_{T,i}$, $\hat{\delta}$ will do dummy steps everywhere, except for at $t' = \min\{t' : t^* > t' > i \text{ and } \tau(t') = 0\}$, if that set is nonempty. At this t', $\hat{\delta}$ sets $\tilde{s}' = \mathsf{Enc}(EK_{\mathsf{mem}}, (t' + 1, 0, \mathcal{T}_{t^*+1}[0]))$. ∎

4.3. Garbling RAM machines. In this section we describe how to modify the construction of section 4.2 to work for RAM machines. For our purposes, a RAM machine consists of two components: an initial internal state q_0, and a transition function δ which takes as input an internal state q and a memory symbol s, and outputs

1. an internal state q', a memory symbol s', and an address addr; or
2. a "final output": either accept or reject.

An input to a RAM machine is a tape $\mathcal{T}_0$. To execute a RAM machine $M = (q_0, \delta)$ on a tape $\mathcal{T}_0$, one does the following.

Define $\mathsf{addr}_0 = 0$. For each $i > 0$, define $(q_i, \mathsf{addr}_i, s_i) = \delta(q_{i-1}, \mathcal{T}_{i-1}[\mathsf{addr}_{i-1}])$ and define the ith memory configuration $\mathcal{T}_i$ as

$$\mathcal{T}_i[\mathsf{addr}] = \begin{cases} s_i & \text{if } s_i \neq \perp \text{ and } \mathsf{addr} = \mathsf{addr}_i, \\ \mathcal{T}_{i-1}[\mathsf{addr}] & \text{otherwise.} \end{cases}$$

If $\delta(q_{t-1}, \mathcal{T}_{i-1}[\mathsf{addr}_{i-1}]) = y \in \{\mathsf{accept}, \mathsf{reject}\}$ for some t, then we say that $M(\mathcal{T}_0) = y$. If there is no such t, we say that $M(\mathcal{T}_0) = \perp$.

Technical overview. The difficulty with garbling RAM machines is the lack of an exact analog to the Pippenger–Fischer theorem. Although there is a vast literature on oblivious RAM (ORAM) transformations, these transformations (unlike the Pippenger–Fischer theorem) inherently produce randomized RAM machines, with access patterns from a fixed *distribution* on sequences of addresses. With *nonsuccinct* garbling schemes, this distinction is unimportant, because typically it is easier to construct a "weak" garbling scheme in which a garbling of (M, x) is simulatable given $M(x)$ and the sequence of addresses accessed by M. Such a garbling scheme can be composed with any ORAM to yield a scheme with full security: a garbling of (M, x) is simulatable given only $M(x)$.

Weak garbling schemes typically preserve or nearly preserve the access pattern of M (indeed otherwise they would themselves be an ORAM, which runs counter to the goal of modularity). For *succinct* garbling schemes, such a weak garbling scheme is impossible: a simulator given an arbitrary access pattern cannot hope to compress it into a succinct $(\tilde{M}, \tilde{x})$, which reproduces that access pattern. Instead, one could hope for the natural weakening to an indistinguishability-based notion of security: if two RAM machines have *identical* access patterns, then their weak garblings should be indistinguishable. It is not clear how such a weak garbling scheme can be composed with an ORAM, but a subsequent work of Canetti and Holmgren [CH16] shows how to do it (for any ORAM satisfying a certain "localized randomness" property).

In this work, due to limitations of our techniques (which were later removed in [KLW15]), we don't know a simpler way of obtaining such a weak garbling scheme. Indeed what we show is how to adapt the bounded space Turing machine garbling scheme of section 4.2 (and its security proof) to a garbling scheme for bounded space RAM machines.

Adapting the construction itself is quite straightforward. Starting with a RAM machine $M = (q_0, \delta)$, we transform it to obtain a different RAM machine $\hat{M}$ as follows. First, we transform M to have an oblivious access pattern with a *predictably timed writes* property; i.e., just prior to reading any memory address, the last-written time for that address is efficiently computable from M's local state. Such a transformation is given in [GHRW14]. Then we construct a new RAM machine $\hat{M}$ which simulates M underneath a basic memory authentication mechanism (starting with an initial tape $\widehat{\mathcal{T}}_0$): whenever M would perform a write "$\mathcal{T}[a] := v$,', $\hat{M}$ instead performs the write "$\widehat{\mathcal{T}}[a] := \mathsf{Enc}(t, a, v)$.'. Whenever M would read $\mathcal{T}[a]$, $\hat{M}$ reads a ciphertext at $\widehat{\mathcal{T}}[a]$, decrypts it to obtain (t, a', v), checks that $a = a'$ and t is the last time that M wrote to $\mathcal{T}[a]$, and if so returns v to M. Additionally, $\hat{M}$ operates on an encrypted local state. Let $(\hat{q}_0, \hat{\delta})$ denote the initial state and transition function of $\hat{M}$. Finally, the garbled RAM machine is $(\hat{q}_0, \tilde{\delta}, \widehat{\mathcal{T}}_0)$, where $\tilde{\delta} \leftarrow i\mathcal{O}(1^\lambda, \hat{\delta})$.

The proof of security follows the same high-level structure as for the Turing machine garbling scheme, although several new technical difficulties arise. We give a sequence of computationally indistinguishable changes to $(\tilde{q}_0, \tilde{\delta}, \widetilde{\mathcal{T}}_0)$, proceeding through several hybrid distributions $\mathsf{H}_1, \ldots, \mathsf{H}_T$, and culminating in a distribution which depends only on $M(\mathcal{T}_0)$. In hybrid H_i, $\hat{M}$ first performs i "dummy steps" in which it accesses memory at a sequence of addresses which simulate those accessed by the ORAM. At each of these addresses, $\hat{M}$ only reencrypts the contents of each memory location it touches. After completing all of the i dummy steps, $\hat{M}$ performs $T - i$ steps of computation as before.

In showing that H_{i-1} is indistinguishable from H_i, we take two steps, each with their own challenges. First, we show that H_{i-1} and H_i are indistinguishable from distributions in which $\hat{\delta}$ has hard-coded mappings for the input/output pairs that it would encounter in an honest execution at time i (and otherwise $\hat{\delta}$ outputs $\perp$ for inputs with timestamp i). This is not easy (see Claim 4.23), but it leaves the hard-coded input/output mappings in $\hat{\delta}$ as the only remaining difference between H_{i-1} and H_i. Within this, the most notable difference is that $\hat{\delta}$'s hard-coded outputs contain (in the clear) different memory addresses. In H_{i-1}, these are real addresses accessed by the ORAM, while in H_i these are simulated addresses (part of a dummy step).

Next, we show that these hard-coded addresses are indistinguishable, even given the other auxiliary information which is present. In particular, we emphasize that in H_i, the initial state $\hat{q}_0$ and $\widehat{\mathcal{T}}_0$ contain the entire ORAM internal state following the

first i accesses. We require the ith oblivious access to remain oblivious even given this information—this is a forward security property that to the best of our knowledge has not been previously studied for ORAMs. We show that an existing ORAM (namely, the ORAM of [CP13]) has this property. It seems likely that other tree-based ORAMs also have this property, but we have not verified this.

4.3.1. Oblivious RAM. We describe the ORAM of Chung and Pass [CP13], which is a simplification of [SCSL11], and we highlight the security property needed for our garbled RAM construction.

Construction. Starting with a RAM machine Π that uses N memory words, the construction transforms it into a machine Π' that uses $N' = N \cdot \mathsf{poly}(\log N, \lambda)$ memory words. While the eventual goal is to store $O(1)$ words in the local state of Π', Chung and Pass start with a "basic" construction in which the local state of Π' consists of a "position map" $\mathsf{pos} : [N/\alpha] \to [N/\alpha]$ for some constant $\alpha > 1$.

The N underlying memory locations are divided into N/α "blocks," each storing α underlying memory words. The external memory is organized as a complete binary tree of N/α leaves. The semantics of the position map is that the ith block of memory maps to the leaf labeled $\mathsf{pos}(i)$. Let $d = \log(N/\alpha)$. The CP/SCSL invariant is that

> *"Block i is stored in some node on the path from the root to the leaf labeled with* $\mathsf{pos}(i)$*."*

Each internal node of the tree stores a few memory blocks. In particular, each internal node, labeled by a string $\gamma \in \{0,1\}^{\leq d}$, is associated with a "bucket" of β blocks for some $\beta = \mathsf{polylog}(N)$.

The reads and writes to a location $r \in [N]$ in the CP/SCSL ORAM proceed as follows:

- Fetch: Let $b = \lfloor r/\alpha \rfloor$ be the block containing the memory location r, and let $i = r \bmod \alpha$ be the component within block b containing the location r. We first look up the leaf corresponding to block b using the (locally stored) position map. Let $p = \mathsf{Pos}(b)$.
  Next, we traverse the tree from the roof to the leaf p, reading and writing the bucket associated to each internal node exactly once. In particular, we read the content once, and then we either write it back, or we erase a block once it is found, and write back the rest of the blocks.
- Update Position Map: Pick a uniformly random leaf $p' \leftarrow [N/\alpha]$ and set (in the local memory) $\mathsf{Pos}(b) = p'$.
- Write Back: In the case of a READ, add the tuple (b, p', v) to the root of the tree. In the case of a WRITE, add the tuple (b, p', v'), where v' is the new value to be written. If there is not enough space in the bucket associated with the root, output overflow and abort. (We note that [CP13, SCSL11] show that, setting the parameters appropriately, the probability that the overflow event happens is negligible.)
- Flush the Block: Pick a uniformly random leaf $p^* \leftarrow [N/\alpha]$ and traverse the tree from the roof to the leaf p^*, making exactly one read and one write operation for every memory cell associated with the nodes along the path so as to implement the following task: "push down" each tuple $(\tilde{b}, \tilde{p}, \tilde{v})$ read in the nodes traversed as far as possible along the path to p^* while ensuring that the tuple is still on the path to its associated leaf $\tilde{p}$ (i.e., maintaining the CP/SCSL invariant). In other words, the tuple ends up in the node $\gamma =$ the longest common prefix of p^* and $\widetilde{p}$. If at any point some bucket is about to

overflow, abort outputting overflow.

The following observation is central to the correctness and security of the CP/SCSL ORAM.

Observation 1. Each oblivious READ and WRITE operation traverses the tree along two randomly chosen paths, independent of the history of operations so far.

This key observation follows from the facts that (1) each position in the position map is used exactly once in a traversal (and before this traversal, this position is not used in determining what nodes to traverse), and (2) the flushing, by definition, traverses a random path, independent of the history.

Security property. Suppose an underlying access pattern is given; we will consider the randomized procedure of executing this sequence of accesses via this ORAM. We want a randomized "dummy" access algorithm $\mathsf{OSample}$ which on input j outputs a list of locations. This list should be distributed according to the real distribution of accesses corresponding to the jth underlying access.

We can now describe our desired security property. Let Q_s be a random variable for the entire ORAM state (both private registers and memory configuration) after the sth virtual step, and let I_j be a random variable for the physical addresses accessed on the jth virtual step.

LEMMA 4.18. *There exist PPT algorithms* Sim *and* $\mathsf{OSample}$ *such that for any* s*, any initial virtual memory configuration* $\mathcal{T}_0$*, any virtual access pattern* $\vec{a} = (a_1, \ldots, a_t)$ *(including both addresses and values written), and any possible (nonzero probability) values* $i_1, \ldots, i_{s-1}$ *of* $I_1, \ldots, I_{s-1}$*, it holds that for all PPT adversaries* $\mathcal{A}$*,*

$$\Pr\left[\mathcal{A}(q_s^b, i_s^b) = b \;\middle|\; \begin{array}{l} q_{s-1}^0 \leftarrow \mathsf{Sim}(\mathcal{T}_0, \vec{a}, i_1, \ldots, i_{s-1}) \\ q_s^0, i_s^0 \leftarrow \mathsf{OAccess}(a_s; q_{s-1}) \\ i_s^1 \leftarrow \mathsf{OSample}(s) \\ q_s^1 \leftarrow \mathsf{Sim}(\mathcal{T}_0, \vec{a}, i_1, \ldots, i_{s-1}, i_s^1) \\ b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

This is a consequence of the following two claims.

CLAIM 4.19. *I_s is independent of $I_1, \ldots, I_{s-1}$ and $a_1, \ldots, a_s$, and is efficiently sampleable.*

Proof. This follows immediately from Observation 1. ∎

This lets us define $\mathsf{OSample}(s)$ as the efficient algorithm which samples I_s. Specifically, in the CP/SCSL ORAM, $\mathsf{OSample}$ samples and outputs two uniformly random paths in each tree.

CLAIM 4.20. *The conditional distribution $Q_s | I_1, \ldots, I_s$ is efficiently sampleable for all values of $I_1, \ldots, I_s$ (that jointly occur with nonzero probability).*

Proof. Recall that Q_s has two parts: a position map Pos and memory contents $\tilde{D}_s$, which are structured as a tree. We first give the sampling procedure for the basic construction, and then extend it to the recursive case. It is easy to verify that this procedure samples the correct distribution.

To sample Q_s given a sequence of physical memory accesses $(i_1, \ldots, i_s)$, do the following. For every memory block $b \in [N/\alpha]$, let $\tau_b \leq s$ be the last time when block b was accessed. Let $I_j = (I_j^{\mathsf{read}}, I_j^{\mathsf{flsh}})$ be the pair of paths that comprise each I_j.

- For each block b, pick a uniformly random leaf $p_b \leftarrow \lfloor N/\alpha \rfloor$. Compute the unique internal node γ_b such that γ_b is the largest common prefix between p_b and any of $I_{\tau_b}^{\mathsf{flsh}}, \ldots, I_s^{\mathsf{flsh}}$.

- Construct Pos by letting $\mathsf{Pos}(b) = p_b$.
- Construct $\tilde{D}_s$ by writing each memory block b together with its value at time s to the internal node γ_b.

To sample Q_s for the recursive construction, we note that this basic sampler doesn't need to know $\vec{x}$ and the entire access pattern; it only needs to know each τ_b described above, as well as the memory contents at time s. This information $(\{\tau'_{b'}\}, \vec{x}'_s)$ for the next smaller recursive case is readily computable. $\vec{x}'_s$ for the next level ORAM is just Pos, which we have already computed. $\tau'_{b'}$ is the maximum of τ_b over all b corresponding to b'. So we can run the basic sampler repeatedly until we have sampled Q_s for the whole recursive construction. ∎

This allows us to define Sim as the above procedure for efficiently sampling Q_s conditioned on $I_1 = i_1, \ldots, I_s = i_s$.

We can now prove a stronger (statistical) version of Lemma 4.18, knowing that Sim and $\mathsf{OSample}$ output the correct conditional distributions.

CLAIM 4.21. $\Pr[q_s^0, i_s^0 = q_s, i_s] \approx \Pr[q_s^1, i_s^1 = q_s, i_s]$.

Proof.

$$\begin{aligned}
\Pr[q_s^0, i_s^0 = q_s, i_s] &\approx \mathbb{E}_{q_{s-1}}[\Pr[q_s, i_s | q_{s-1}, i_1, \ldots, i_{s-1}] | i_1, \ldots, i_{s-1}] \\
&= \Pr[q_s, i_s | i_1, \ldots, i_{s-1}] \\
&= \Pr[q_s | i_1, \ldots, i_s] \Pr[i_s | i_1, \ldots, i_{s-1}] \\
&= \Pr[q_s | i_1, \ldots, i_s] \Pr[i_s] \\
&\approx \Pr[q_s^1, i_s^1 = q_s, i_s].
\end{aligned}$$

1. The first approximate equality follows from Sim approximately sampling q_{s-1} given $i_1, \ldots, i_{s-1}$ and from $\mathsf{OAccess}$ (exactly) sampling q_s, i_s given q_{s-1}.
2. The second equality is just marginalization over q_{s-1}.
3. The third equality is the chain rule for probabilities.
4. The fourth equality is Claim 4.19, namely, that the locations accessed at time s are independent of previously accessed locations.
5. The fifth approximate equality follows from $\mathsf{OSample}$ approximately sampling i_s. ∎

4.3.2. Garbling construction. As for Turing machines, our garbling scheme for RAM machines consists of three parts:

- an initial state $\tilde{q}_0$;
- an obfuscated transition function $i\mathcal{O}(\hat{\delta})$;
- an initial tape $\widetilde{\mathcal{T}}_0$.

We generate $(\tilde{q}_0, \tilde{\delta}, \widetilde{\mathcal{T}}_0)$ by transforming $(q_0, \hat{\delta}, \mathcal{T}_0)$ into three steps:

1. We apply the ORAM transformation of [CP13]. Additionally, we give the transformed machine a "predictably timed writes" property [GHRW14]. Informally, this states that whenever it accesses a memory address, it already knows the last time at which that address was written to. To supply the randomness needed by the ORAM, we give the transition function a hard-coded puncturable PRF F_{ORAM}. The "randomness" used at time t is given by $F_{\mathsf{ORAM}}(t)$.
2. We augment the internal states to contain a timestamp which starts at 0 and is incremented on every step. We also augment the contents of each memory cell to store its address, as well the timestamp at which it was last written.

Finally, we modify δ to check, given its internal state, that the accessed memory value is labeled with the expected timestamp and address (checking the expected timestamp uses the predictably timed writes property).

3. We modify the transition function so that its inputs and outputs are all ciphertexts, with the exception of the address addr, which it outputs in the clear. We hard-code two secret keys, $\mathsf{sk}_{\mathsf{state}}$ and $\mathsf{sk}_{\mathsf{mem}}$, which it uses to decrypt its inputs and encrypt its outputs using our special authenticated encryption scheme (ACE).

We will let $M' = (q_0', \delta')$ and $\mathcal{T}_0'$ refer to the result of step 1 above. Internal states q' of M' will be tuples (q, q_{ORAM}), where q is a state of M and q_{ORAM} is an ORAM state. Our hybrids below will primarily modify M', and then will apply the same steps 2 and 3 above.

4.3.3. Security proof. Given a RAM machine $M = (q_0, \delta)$ and an input $\mathcal{T}_0$, we give a sequence of hybrid distributions $\mathsf{H}_0, \ldots, \mathsf{H}_T$ such that H_0 is distributed as our garbling of $(M, \mathcal{T}_0)$ constructed above, and H_T is efficiently sampleable given $M(\mathcal{T}_0)$.

Definition of hybrids. To better describe our hybrids, we will introduce a bit of terminology. Suppose the ORAM has an overhead of η—so one step of executing M on $\mathcal{T}_0$ naturally corresponds to a chunk of η steps of $\hat{M}$ on $\widehat{\mathcal{T}}_0$. We refer to one such chunk as a *virtual step*. When we wish to emphasize that we are referencing a single step in the execution of $\hat{M}$ on $\widehat{\mathcal{T}}_0$, we use the term *physical step*.

In the description of our hybrids, the greatest complexity arises in describing the transition function $\hat{\delta}$. For simplicity, we describe transition functions by how they act at the level of virtual steps. Precisely, we will consider transition functions $\hat{\delta}$ which on input $(\tilde{q}, \tilde{s})$ first decrypt $(t, q) \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{state}}, \tilde{q})$ and decompose t as $\eta \cdot t_{\mathsf{vrt}} + t_{\mathsf{phys}}$. $\hat{\delta}$ then does one of four things:

1. *Real virtual step:*
   (a) Decrypt $(t_{\mathsf{mem}}, \mathsf{addr}, s) \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{mem}}, \tilde{s})$.
   (b) Check that t_{mem} and addr are as expected given (t, q) (i.e., t_{mem} matches the "expected timestamp" given by predictably timed writes). Otherwise output $\perp$.
   (c) Compute $\mathsf{out} \leftarrow \delta'(q, s)$.
   (d) If $\mathsf{out} \in \{\mathsf{accept}, \mathsf{reject}\}$, output it. Otherwise, parse out as (q', s', addr').
   (e) Let $\tilde{q} = \mathsf{Enc}(\mathsf{sk}_{\mathsf{state}}, (t+1, q'))$ and let $\tilde{s} = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (t+1, \mathsf{addr}, s'))$.
   (f) Output $(\tilde{q}, \tilde{s}, \mathsf{addr}')$.
2. *Dummy virtual step:*
   (a) Decrypt $(t_{\mathsf{mem}}, \mathsf{addr}, s) \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{mem}}, \tilde{s})$.
   (b) Check that t_{mem} and addr are as expected, given (t, q). Otherwise output $\perp$.
   (c) Compute $(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta) \leftarrow \mathsf{OSample}(i; F_{\mathsf{dummy}}(i))$.
   (d) Let $\tilde{q} = \mathsf{Enc}(\mathsf{sk}_{\mathsf{state}}, (t+1, q))$ and let $\tilde{s} = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (t+1, \mathsf{addr}, s))$.
   (e) Output $(\tilde{q}, \tilde{s}, \mathsf{addr}_{t_{\mathsf{phys}}})$.
3. *Half-dummy virtual step:* Do the same thing as in a dummy virtual step, except use hard-coded values for $(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta)$ instead of sampling them from $\mathsf{OSample}(i)$.
4. *Hard-coded virtual step:* There are η different hard-coded output values $w_1, \ldots, w_\eta$, where each w_i is a triple $(c_i^{\mathsf{state}}, c_i^{\mathsf{mem}}, \mathsf{addr}_i)$. Output $w_{t_{\mathsf{phys}}}$.

Definition of hybrid H_i. In the ith hybrid, $\tilde{M}$ is a RAM program M'' which performs i dummy virtual steps, and then continues for up to $T - i$ real virtual steps.

$\mathcal{T}_0''$ and q_0'' are generated to be consistent with M'''s i dummy steps by using the ORAM's Sim procedure. Namely, let $\vec{a} = \mathsf{addr}_1, \ldots, \mathsf{addr}_{t^*}$ be the virtual addresses accessed by M on x. Let $\vec{a}'' = \mathsf{OSample}(0; F_{\mathsf{dummy}}(0)), \ldots, \mathsf{OSample}(i; F_{\mathsf{dummy}}(i))$ be the physical addresses accessed in the first i virtual steps of M''. We sample

$$\mathcal{T}_0'', q_{\mathsf{ORAM}}'' \leftarrow \mathsf{Sim}_{\mathsf{ORAM}}(\mathcal{T}_0, \vec{a}, \vec{a}'').$$

$\widetilde{\mathcal{T}}_0$ is defined such that

$$\widetilde{\mathcal{T}}_0[\mathsf{addr}] = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, \mathsf{addr}, \mathcal{T}_0''[\mathsf{addr}])),$$

and $\tilde{q}_0$ is defined such that

$$\tilde{q}_0 = \mathsf{Enc}(\mathsf{sk}_{\mathsf{state}}, (0, (q_i, q_{\mathsf{ORAM}}''))).$$

Indistinguishability of H_{i-1} *and* H_i. To show that hybrid $i-1$ is indistinguishable from hybrid i, we introduce additional hybrids $\mathsf{H}_{i-1,1}$ through $\mathsf{H}_{i-1,3}$. These hybrids' transition functions differ from that of hybrid H_{i-1} only in how $\hat{\delta}$ behaves on the ith virtual step.

HYBRID $\mathsf{H}_{i-1,1}$: The ith virtual step is a hard-coded step, with hard-coded values chosen so that the transcript of execution is identical to that of hybrid H_{i-1}.

HYBRID $\mathsf{H}_{i-1,2}$: The ith virtual step is a half-dummy step, with the addresses hard-coded as in hybrid $\mathsf{H}_{i-1,1}$.

In this hybrid, the initial state $\tilde{q}_0$ is defined as $\mathsf{Enc}(\mathsf{sk}_{\mathsf{state}}, (0, q_i))$, and the initial tape $\widetilde{\mathcal{T}}_0$ is defined such that

$$\widetilde{\mathcal{T}}_0[\mathsf{addr}] = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, \mathsf{addr}, \mathcal{T}_i[\mathsf{addr}])).$$

HYBRID $\mathsf{H}_{i-1,3}$: The ith virtual step is again a half-dummy step, but the hard-coded addresses $(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta)$ are now obtained by sampling

$$(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta) \leftarrow \mathsf{OSample}(i).$$

Additionally, $\mathcal{T}_0''$ and q_{ORAM}'' are now sampled as

$$\mathcal{T}_0'', q_{\mathsf{ORAM}}'' \leftarrow \mathsf{Sim}_{\mathsf{ORAM}}(\mathcal{T}_0, \vec{a}, \vec{a}''),$$

where $\vec{a}''$ denotes the physical addresses accessed by the first i virtual steps, rather than the first $i-1$.

CLAIM 4.22. *Hybrid* $\mathsf{H}_{i-1} \approx \mathsf{H}_{i-1,1}$.

Proof. We apply the template from section 4.2 of iteratively restricting the encryption/decryption keys to ensure that for each $t_{\mathsf{phys}} \in \{0, \ldots, \eta-1\}$, conditioned on $t = \eta i + t_{\mathsf{phys}}$, there is only one possible output of $\hat{\delta}$. Once this unique output condition is satisfied, hard-coding the output values when $t_{\mathsf{vrt}} = i$ preserves the functionality of $\hat{\delta}$ and is hence indistinguishable by the security of $i\mathcal{O}$.

In order to guarantee the above uniqueness of possible outputs, we define the good set G_{state} as all (t, q) satisfying the following:

- If $t < \eta i$, then q differs from $q_{\eta i}$ only in the parts of the state required for predictably timed writes. In other words, q essentially doesn't change during the dummy steps.

- If $\eta i \le t < (\eta+1)i$, then q is "correct," i.e., its value is the same as in an honest execution. Note that this only requires hard-coding η values into the description of G_{state}.

And we define the good set G_{mem} as all $(t_{\mathsf{mem}}, \mathsf{addr}, s)$ satisfying the following:

- If addr is one of the physical addresses accessed in the ith virtual step, and if $t_{\mathsf{mem}} < (\eta+1)i$, then s is "correct"; i.e., its value is the same as in an honest execution. Since $\mathcal{T}[\mathsf{addr}]$ is modified at most η times before virtual step $i+1$, and there are at most η such addresses, the description size of this constraint is $O(\eta^2)$.
- If addr is the parent of a physical address addr' accessed in the ith virtual step (with memory viewed as a tree according to the predictably timed writes structure), let $\tau_{\mathsf{addr}'}$ denote the last time that a dummy step accesses addr. If $\tau_{\mathsf{addr}'} \le t_{\mathsf{mem}} < \eta i$, then s encodes that the expected timestamp for addr' is exactly $\tau_{\mathsf{addr}'}$.

EK_{state}, DK_{state} and $EK_{\mathsf{mem}}, DK_{\mathsf{mem}}$ can be restricted and unrestricted to G_{state} and G_{mem}, respectively, because $(G_{\mathsf{state}}, G_{\mathsf{mem}})$ is δ-invariant. ∎

CLAIM 4.23. *Hybrid* $\mathsf{H}_{i-1,1} \approx \mathsf{H}_{i-1,2}$.

Proof. This claim is analogous to Claim 4.17 in our Turing machine construction, but with additional difficulties.

The challenges. The heart of the proof of Claim 4.17 replaced a conditional "if $t = t^*$" with a different condition—"if $\tilde{q} = c_{\mathsf{state}}$" or "if $\tilde{s} = c_{\mathsf{mem}}$"—and then argued that these conditions were functionally equivalent with suitably (indistinguishably) punctured keys. In the case of $\tilde{s}$, it suffices to show that $t = t^*$ if and only if $t_{\mathsf{mem}} = \beta(t^*)$, where $\beta(t^*)$ is the time before t^* that the same address was accessed. For Turing machines, these checks were equivalent, because $\hat{\delta}$ directly computed and checked that $t_{\mathsf{mem}} = \beta(t)$. $\beta(t)$ was the most recent time the tth address of the Pippenger–Fischer access pattern was accessed.

However, for an oblivious RAM, there is no such fixed access pattern, and β is therefore not computable by the real-world $\hat{\delta}$. $\hat{\delta}$ computes β instead via the predictably timed write transformation of [GHRW14]. In this transformation, each access to memory is transformed into several $(\log S)$ accesses, such that if all previous accesses were performed correctly, then the RAM machine's local state will know what timestamp to expect in the address presently being accessed.

The approach. We will define more elaborate sets on which $\hat{\delta}$'s ACE keys are punctured. These sets must be restrictive enough to ensure that the predictably timed write steps yield the correct answer at time t^*, but not so restrictive that they have too large of a description size. For example, we cannot define the sets to be "all of the values in a correct execution." To construct the appropriate sets, we will need to look at the structure of the predictably timed write transformation.

We want to emulate virtual random access to some database D, such that on every physical memory access to an address addr, we know the timestamp at which addr was last written to (and therefore the timestamp we should expect to see upon reading addr). We structure physical memory as a complete binary tree whose leaves are the cells of D. This tree obeys the invariant that, at the beginning of any virtual access, every node stores the times at which each of its children were most recently accessed. To virtually access $D[i]$, we first access the root, and then access each successive child on the path to $D[i]$. When accessing a node, we first read it, to see when the next child was last written. We then write back the node with an updated

timestamp for that child, knowing that the child is about to be accessed.

The proof. To show that $\mathsf{H}_{i-1,1} \approx \mathsf{H}_{i-1,2}$, we use a hybrid argument, which we describe as a sequence of indistinguishable changes.

1. First, we separate the hard-coded state ciphertexts from the hard-coded memory ciphertexts. That is, instead of
   If $t = \eta i + j$, output w_j
   for some hard-coded tuple of outputs w_j, $\hat{\delta}$ will instead have three conditions:
   If $t = \eta i + j$, then set $\tilde{q}' = c_j^{\mathsf{state}}$
   and
   If $t = \eta i + j$, then set $\tilde{s}' = c_j^{\mathsf{mem}}$
   and
   If $t = \eta i + j$, then set $\mathsf{addr}' = \mathsf{addr}_j$.
   Clearly, this change preserves the functionality of $\hat{\delta}$.
2. Next, for every address, we remove all but the last time at which the address is written. So, $\hat{\delta}$ contains a line of code saying
   If $t = \eta i + \eta - 1$, then set $\tilde{q}' = c_{\eta-1}^{\mathsf{state}}$
   and also, for every address addr_j, a line which says
   If $t = \eta i + j$, then set $\tilde{s}' = c_j^{\mathsf{mem}}$.
   We can do such a thing because all but the last hard-coded value will never be decrypted, so the others may as well be dummy steps. The good sets required to enforce this are similar to good sets we have seen already.
3. Finally, for each of these lines, we go through a sequence of hybrids whose result is to (i) remove the line and (ii) modify $\tilde{\mathcal{T}}_0$ or $\tilde{q}_0$ instead to incorporate the hard-coded c_j^{mem} or $c_{\eta-1}^{\mathsf{state}}$, respectively. This is the focus of the paragraph below. For brevity, we include only the (more involved) case of removing the lines which give a hard-coded value of $\tilde{s}'$.

Let addr^* be the address to which $\tilde{s}'$ is being written, let $c^* = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, m^*)$ be the hard-coded value of $\tilde{s}'$, and let t^* be the time at which it is written. We will replace t^* by $\beta(t^*)$ until $\beta(t^*) = 0$—that is, until t^* is the first time at which addr^* is written. Recall that $\beta(t^*)$ is the last time before t^* that addr^* was written. We similarly change the timestamp of m^* from t^* to $\beta(t^*)$. We write this as $m^*[t_{\mathsf{mem}} \mapsto \beta(t^*)]$.

If $\beta(t^*)$ is 0, then we completely remove the line of code and replace $\tilde{\mathcal{T}}_0[\mathsf{addr}]$ by $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, m^*)$ (but with the timestamp of m^* set to 0).

We now must show how t^* can be replaced by $\beta(t^*)$. Again, this is a sequence of indistinguishable changes.

1. We first add another branch to $\hat{\delta}$, saying
   If $t = \beta(t^*)$, then set $\tilde{s}' = c'^*$
   with hard-coded $\beta(t^*)$ and hard-coded c'^* chosen to be the "correct" value. This change is indistinguishable for the same reason that H_{i-1} was indistinguishable from $\mathsf{H}_{i-1,1}$.
2. Next, we change the condition "if $t = t^*$" into "if $\tilde{s} = c'^*$." With appropriately defined sets G_{state} and G_{mem}, this change preserves the functionality of $\hat{\delta}$ and is hence indistinguishable.
3. We replace c'^*_{mem} by $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, m^*[t_{\mathsf{mem}} \mapsto \beta(t^*)])$. With appropriately restricted ACE keys (no memory ciphertext with a timestamp of $\beta(t^*)$ needs to be encrypted or decrypted by $\hat{\delta}$), this is indistinguishable by semantic security.
4. We remove the "if $\tilde{s} = c'^*$" branch. With the newly modified value of c'^*, its action of setting $\tilde{s}' = c^*$ is the same as what a dummy step would do, so

removing the branch has no effect on the functionality of $\hat{\delta}$.

The main thing now is to show how to make the conditions "if $t = t^*$" and "if $\tilde{s} = c'^*$" functionally equivalent. It suffices to define the good sets such that the following hold:

- G_{state} is the set of (t, q) satisfying the following:
  - If $t \leq t^*$, then q differs from q_{t^*} only in the substate of the predictably timed write mechanism.
  - If $t = t^*$, then $q = q_{t^*}$.
- G_{mem} is the set of $(t_{\mathsf{mem}}, \mathsf{addr}, s)$ satisfying the following:
  - If addr is an internal node of the predictably timed write tree on the path from the root to addr^*, and if $t^*_{\mathsf{mem}} \leq t_{\mathsf{mem}} < t^*$, then s encodes that the next node on the path to addr^* should also have a timestamp which is exactly t^*_{mem}. ∎

CLAIM 4.24. *Hybrid* $\mathsf{H}_{i-1,2} \approx \mathsf{H}_{i-1,3}$.

Proof. The indistinguishability of this change is exactly our ORAM security property, which is stated and proved in Lemma 4.18 of section 4.3.1. ∎

CLAIM 4.25. *Hybrid* $\mathsf{H}_{i-1,3} \approx \mathsf{H}_i$.

Proof. The difference between $\mathsf{H}_{i-1,3}$ and H_i is that $\mathsf{H}_{i-1,3}$ has a hard-coded list of addresses $(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta)$ to access on virtual step i, while H_i computes the addresses as $(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta) = \mathsf{OSample}(i; F_{\mathsf{dummy}}(i))$.

We give a sequence of indistinguishable changes which transform $\mathsf{H}_{i-1,3}$ into H_i:

1. Replace the PRF key F_{dummy} in $\hat{\delta}$ with a punctured PRF key $F_{\mathsf{dummy}}\{i\}$. This preserves the functionality of $\hat{\delta}$ and is thus indistinguishable.
2. Instead of hard-coding the addresses as
$$(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta) \leftarrow \mathsf{OSample}(i)$$
(i.e., with true randomness), hard-code them as
$$(\mathsf{addr}_1, \ldots, \mathsf{addr}_\eta) = \mathsf{OSample}(i; F_{\mathsf{dummy}}(i)).$$
This change is indistinguishable because $F_{\mathsf{dummy}}(i)$ is pseudorandom *even to a distinguisher which is given the punctured key* $F_{\mathsf{dummy}}\{i\}$.
3. Change $\hat{\delta}$ to be as in H_i. This preserves the functionality of $\hat{\delta}$ and is hence indistinguishable. ∎

CLAIM 4.26. *There is a PPT simulator* Sim *such that hybrid* H_T *is computationally indistinguishable from* $\mathsf{Sim}(M(\mathcal{T}_0), 1^S, 1^T)$.

Proof. This is not immediately clear, since in our definition of H_T, $\tilde{\mathcal{T}}_0$ still contains an encryption of $\mathcal{T}_T$, so we need to argue that $\mathcal{T}_T$ is not revealed. Before describing how to do this, we first remark that one can get around this in a black-box way with no asymptotic overhead if restricted to RAM machines whose running time is longer than their database size. Instead of garbling a RAM machine M directly, one simply garbles an M' which runs M, and then erases its memory.

However, since the RAM model of computation is arguably most useful when applied to sublinear computations, we would like to achieve something more efficient.

We use a sequence of $S+1$ hybrids $\mathsf{H}_{T,0}, \ldots, \mathsf{H}_{T,S}$, where hybrid $\mathsf{H}_{T,j}$ differs from H_T only in that, for all $0 \leq k < j$,
$$\tilde{\mathcal{T}}_0[k] = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, k, \perp)).$$

It therefore suffices to show how to change $\tilde{\mathcal{T}}_0[j]$ from $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, j, \mathcal{T}_T[j]))$ to $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, j, \perp))$.

For this, we give another sequence of hybrids $\mathsf{H}_{T,j-1,0}, \ldots, \mathsf{H}_{T,j-1,N_j}$, where the number of hybrids N_j is equal to the number of times that $\hat{\delta}$ accesses address j. The structure of these hybrids resembles those used in Claim 4.23, but in reverse.

Let t_k denote the kth time that $\hat{\delta}$ accesses address j. Hybrid $\mathsf{H}_{T,j-1,k}$ is defined to differ from $\mathsf{H}_{T,j}$ only in that $\hat{\delta}$ hard-codes $\tilde{s}' = \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (t_k + 1, j, \mathcal{T}_T[j]))$ at time t_k. Or, if $k = 0$, $\tilde{\mathcal{T}}[j]$ has the hard-coded value $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, j, \mathcal{T}_T[j]))$ instead.

Just as in Claim 4.23, these hybrids are shown to be indistinguishable by introducing another hard-coded branch at t_{k+1}, and then switching the hard-coded value of $\tilde{s}'$ so that the "if $t = t_k$" branch is functionally equivalent to a dummy step. All of this is done with analogously defined good sets. ∎

4.4. The bounded space requirement and subsequent work. In this section we describe why the size of our randomized encodings grows proportional to the space complexity of the underlying computation, and give a taste of the techniques used by [KLW15] to remove this dependence.

The dependence on the space complexity appears to come from our proof technique rather than being inherent in the construction. Consider for simplicity our Turing machine construction. Recall that only one part of the construction grows with S—the initial garbled tape $\widetilde{\mathcal{T}}_0$, defined as

$$\widetilde{\mathcal{T}}_0[i] = \begin{cases} \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, i, x_i)) & \text{if } i < |x|, \\ \mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}, (0, i, \perp)) & \text{otherwise} \end{cases}$$

for $i = 0, \ldots, S - 1$. In other words, $\widetilde{\mathcal{T}}_0$ is a bit-by-bit encryption of the input x, padded with $S - |x|$ dummy ciphertexts. One natural approach to making the randomized encoding fully succinct is to give out the dummy ciphertexts implicitly (and succinctly) instead of explicitly, e.g., via a small obfuscated circuit that on input i (for $|x| \leq i < S$) outputs $\mathsf{Enc}(\mathsf{sk}_{\mathsf{mem}}(0, i, \perp))$. One may conjecture the security of this variant, and we do not know of any attacks. However, major difficulties arise in reducing the security of this variant to the security of the underlying primitives (**iO** for circuits and ACE).

Indeed, recall that we prove security via a hybrid argument, where in the ith hybrid $\widetilde{\mathcal{T}}_0$ encrypts the (padded) ith memory configuration $\mathcal{T}_i$ of M when executed on x. In order for these hybrids to be indistinguishable from one another, $\widetilde{\mathcal{T}}_0$ (in all hybrids) needs to be as large as S, the size of the largest memory configuration of M when executed on x. This hybrid structure was no accident: due to the succinctness requirements on the sets to which ACE keys are constrainable, we were only able to change the behavior of $\hat{\delta}$ on the $i + 1$st step once $\hat{\delta}$ already performs dummy actions on the first i steps. This behavior of $\hat{\delta}$ necessitated hard-coding the ith configuration in $\widetilde{\mathcal{T}}_0$ for correctness.

[KLW15] get around this difficulty by developing a new way of authenticating the inputs to $\hat{\delta}$: Rather than using an authenticated encryption scheme directly on encrypted memory words, they maintain a signed Merkle tree of the entire tape contents. This hash function is designed so that the Merkle tree is "somewhere statistically binding" as in [HW15]. Namely, the "local opening" property of the Merkle tree can be strengthened: for any index $j \in [S]$, the hash function can be indistinguishably sampled so that for any Merkle root, there is only one local opening at j. Thus the problem of authenticating the memory inputs to $\hat{\delta}$ on step i is reduced to maintaining

a small authenticated local state (the TM state and Merkle tree root).

To authenticate this state, [KLW15] uses a special "splittable signature" scheme with constrainability properties resembling those of ACE. They use a puncturable PRF to generate one signing/verification key pair per time step. As in our security proof, the signing and verification keys are alternately constrained until the verification key used on the ith time step will only accept signatures of the *correct* ith local state. The main difficulty now is how to "forget" (for the sake of succinctness) that keys of the first $i-1$ time steps are constrained—otherwise the number of hard-coded keys or messages in $\hat{\delta}$ will grow linearly with T. [KLW15] show how this can be accomplished by augmenting each local state q_i with a special hash of q_{i-1}.

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

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

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

and the evaluation algorithm Eval is the decoding algorithm Dec.

In accordance, the correctness, security, and efficiency properties are all defined similarly to garbling schemes, as defined in section 2.2 (in particular, it will be convenient to consider randomized encodings that, like garbling schemes, also guarantee the privacy of the program and not just the input). When projecting a garbling scheme to a randomized encoding scheme as above, the randomized encoding inherits the corresponding efficiency properties of the garbling scheme.

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

Succinct iO. The construction of succinct **iO** from randomized encoding and exponential **iO** for circuits is a natural instantiation of the bootstrapping approach suggested by Applebaum [App14]. There, the goal is to reduce obfuscation of general circuits to obfuscation of NC^1 circuits; our goal is to reduce obfuscation of programs with large running time (but bounded space) to obfuscation of significantly smaller circuits. To obfuscate a succinct program Π with respect to inputs of size at most n, we obfuscate a small circuit $C^{\Pi,K}$ that has a hardwired seed K for a PRF and, given input x, applies the PRF to x to derive randomness and then computes a succinct randomized encoding of $\widehat{\Pi}(x)$. The obfuscated $i\mathcal{O}(\Pi)$, given input x, computes the encoding, decodes it, and returns the result.

The analysis in [App14] establishes security in case the circuit obfuscator $i\mathcal{O}$ is virtually black-box secure. We show that if $i\mathcal{O}$ has $2^{-\lambda^\varepsilon}$-security for security parameter $\lambda \gg n^\varepsilon$, and the PRF is puncturable and is also $2^{-\lambda^\varepsilon}$-secure, then a similar

result holds for **iO** (rather than virtual black box). The proof is based on a general *probabilistic* ***iO*** argument, an abstraction recently made by Canetti et al. [CLTV15].

Succinct FE. The construction of succinct functional encryption (FE) follows rather directly by plugging our randomized encodings into previous constructions of nonsuccinct functional encryption. Concretely, starting with the scheme of Gentry et al. [GHRW14], we can replace the nonsuccinct randomized encodings for RAM in their construction with our succinct randomized encodings and obtain selectively secure FE.[13] Alternatively, starting from the scheme of Garg et al. [GGHZ16], we can replace randomized encodings for circuits in their construction with our succinct randomized encodings and get an adaptively secure succinct FE scheme. (Here we also need to rely on the fact that succinct randomized encodings can be computed in low depth, which is required in their construction.) We note that in both cases, our succinct randomized encodings already satisfy the required security for their security proof to go through, and only the succinctness features change.

Publicly verifiable delegation. Finally, we sketch the basic ideas behind the delegation scheme. The delegation scheme is pretty simple and similar in spirit to previous delegation schemes (in a weaker processing model) [AIK10, GGP10, PRV12, GKP+13]. To delegate a computation, given by Π and x, the verifier simply sends the prover a randomized encoding $\widehat{\Pi'}(x, r)$, where Π' is a machine that returns r if and only if it accepts x, and r is a sufficiently long random string. The security of the randomized encoding implies that the prover learns nothing of r, unless the computation is accepting. The scheme can be easily made publicly verifiable by publishing $f(r)$ for some one-way function f. Furthermore, the scheme ensures input-privacy for the verifier.

We then propose a simple transformation that can be applied to any delegation scheme in order to make the first verifier message reusable. The idea is natural: we let the verifier's first message be an obfuscation of a circuit C^K that has a hardwired key K for a puncturable PRF and, given a computation (Π, x), applies the PRF to derive randomness and generates a first message for the delegation scheme. Thus, for each computation, a first message is effectively sampled afresh. Relying on **iO** and the security of the puncturable PRF, we can show that (nonadaptive) soundness is guaranteed. The transformation can also be applied to privately verifiable delegation schemes, such as the one of [KRR14], and maintains soundness, even if the prover has a verification oracle.

5.2. From randomized encodings to iO. We present a generic transformation from a garbling scheme for an algorithm class $\{\mathcal{AL}_\lambda\}$ to an indistinguishability obfuscator for $\{\mathcal{AL}_\lambda\}$, assuming subexponentially indistinguishability obfuscators for circuits. We require that the algorithm class 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 5.1. *Let $\{\mathcal{AL}_\lambda\}$ be any monotonically increasing class of deterministic algorithms. It holds that if there are*

(i) *a subexponentially indistinguishable* ***iO****, $i\mathcal{O}^C$, for circuits, and*

(ii) *a subexponentially indistinguishable randomized encoding $\mathcal{RE}$ for $\{\mathcal{AL}_\lambda\}$,*

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

then there is an indistinguishability obfuscator $i\mathcal{O}^A$ for $\{\mathcal{AL}_\lambda\}$.

Furthermore, the following efficiency preservation holds:

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

Before moving to the proof of the proposition, we first note that combining Proposition 5.1 with constructions of garbling schemes for TM and RAM in section 3, we directly obtain **iO** for TM and RAM with space-dependent complexity.

THEOREM 5.2. *Assume a subexponentially indistinguishable **iO** for circuits and subexponentially secure OWF. There is an indistinguishability obfuscator for* TM *and* RAM *with space-dependent complexity.*

Proof of Proposition 5.1. This result relies on the following natural way of obfuscating probabilistic circuits, abstracted in [CLTV15].

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

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

where $\lambda' = (\lambda + n)^{1/\varepsilon}$ for $n = C.n$, so that **iO** and F are $\mathrm{negl}(\lambda)2^n$-indistinguishable. The work of [CLTV15] showed that the above obfuscations are indistinguishable for circuits whose output distributions are strongly indistinguishable for every input. More specifically, circuits C_1 and C_2 with the same input length n are strongly indistinguishable (with respect to auxiliary input z) if for every input $x \in \{0,1\}^n$ the outputs $C_1(x)$ and $C_2(x)$ are $\mathrm{negl}(\lambda)2^{-n}$-indistinguishable (given z). The following lemma summarizes this.

LEMMA 5.3 ($pi\mathcal{O}$ for circuits [CLTV15]). *Assume subexponentially indistinguishable **iO** for circuits $i\mathcal{O}^C$, and subexponentially indistinguishable OWF. Then, for every class $\{C_\lambda\}$ of polynomial-sized circuits, and every nonuniform* PPT *sampleable distribution $\mathcal{D}$ over the support of $\left\{C_\lambda \times C_\lambda \times \{0,1\}^{\mathrm{poly}(\lambda)}\right\}$, if it holds that for every nonuniform* PPT *adversary $\mathcal{R}$, and input x, that*

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

then the following ensembles are computationally indistinguishable:

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

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

Proof sketch of Lemma 5.3. The lemma essentially follows from complexity leveling. To see the proof, first consider a simpler case, where the two circuits C_1 and C_2 have identical implementation on all but one input x^*, and the outputs on x^*, $C_1(x^*)$ and $C_2(x^*)$, are indistinguishable. In this case, it follows directly from the security

iO that obfuscation of C_b, $\hat{C}_b \xleftarrow{\$} pi\mathcal{O}(1^\lambda, C_1)$ is indistinguishable to the obfuscation of $C'_b \xleftarrow{\$} i\mathcal{O}(C'_b)$, where C'_b has a punctured key $k(x^*)$ and $C_b(x^*; \mathsf{F}(k, x^*))$ hardwired in; then it follows from the pseudorandomness of puncturable PRF and the indistinguishability of $C_1(x^*)$ and $C_2(x^*)$ that $i\mathcal{O}(C'_0)$ and $i\mathcal{O}(C'_1)$ are indistinguishable. Therefore, overall obfuscation of C_1 and C_2 are indistinguishable.

Now consider the case where C_1 and C_2 are sampled from $\mathcal{D}(1^\lambda)$, and their output distributions for every input are $\mathrm{negl}(\lambda)2^{-n}$-indistinguishable. To show that their $\mathsf{p}IO$ obfuscation are indistinguishable, consider an exponential, 2^n, number of hybrids where, in each hybrid, a circuit C_i is obfuscated, which outputs $C_2(x)$ for every input $x \le i$ and outputs $C_1(x)$ for every input $x > i$. Since in every two neighboring hybrids, C_i and C_{i+1} are the same except on one input, $x^* = i + 1$. By the argument above, neighboring hybrids have a distinguishing gap $O(\mathrm{negl}(\lambda)2^{-n})$. Thus, by a hybrid argument, obfuscations of C_1 and C_2 are indistinguishable. This concludes the lemma. ∎

Construction of iO for general algorithms. Using Lemma 5.3, we now prove Proposition 5.1.

Given $2^{-\lambda^\varepsilon}$-indistinguishable **iO** $i\mathcal{O}^C$ and $2^{-\lambda^\varepsilon}$-indistinguishable randomized encoding $\mathcal{RE}$, let $pi\mathcal{O}$ be the obfuscator for probabilistic circuits constructed from $i\mathcal{O}^C$. Our **iO** for the a general algorithm class $\{\mathcal{AL}_\lambda\}$ is defined as follows:

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

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

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

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

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

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

Consider the sampler $\mathcal{D}'(1^\lambda)$ that outputs C'_1, C'_2, z by sampling $(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and setting $C'_b = \mathsf{REnc}(1^{\lambda'}, AL_b, \cdot)$. It follows from the security of $\mathcal{RE}$ that for every nonuniform adversary $\mathcal{R}$ and every input x, the output distributions of $C'_1(x)$ and $C'_2(x)$ are $\mathrm{negl}(\lambda)2^n$-indistinguishable, given x, z, C'_1, C'_2. Thus, it follows from Lemma 5.3 that the above two ensembles are indistinguishable, as are the obfuscations of AL_1 and AL_2.

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

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

Therefore,

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

This concludes the proof of Proposition 5.1.

A corollary: Output-independence. The size of the randomized encodings (or garbling schemes) described in previous sections depends (linearly) on the output of the encoded computation; accordingly, so does the succinct **iO** construction described in this section. We start by noting that in the succinct **iO** construction this dependence can be easily removed. Concretely, rather than considering the machine $AL(x)$ that for any input x might have an m-bit output y, we can consider a new single-bit machine $AL'(x, i)$ that, given additional input $i \in \{0,1\}^{\log m}$, outputs y_i. Observe that if AL_0 and AL_1 compute the same function, then clearly so do their single-bit versions, AL'_0 and AL'_1. The overhead is only polylogarithmic in the output size m. Thus, we directly obtain succinct **iO** that is output-independent. We note that this does not involve making any additional computational assumptions. (Note that we only increase the input size logarithmically, and thus the exponential loss in the input incurred by the transformation given by Theorem 5.1 is only polynomial.)

Next, we observe that this directly implies indistinguishability-based succinct randomized encodings that are output-independent. The encoding of AL, x simply consists of an (output-independent) obfuscation of a machine that has no input and output $AL(x)$. (Note that this only requires polynomial **iO**, since the exponential blowup in the input size of the transformation given by Theorem 5.1 is completely avoided.)

More efficient construction. Evaluating the **iO** for TM and RAM obtained in Theorem 5.2 on input x involves evaluating the obfuscated program on x once to obtain a randomized encoding $\widehat{AL}(x)$, and then decoding it. When relying on an arbitrary randomized encoding with space-dependent complexity, the overall evaluation takes time $T_{AL}(x) \times \mathrm{poly}(\lambda, |AL|, S)$. When the space is large, the overhead on the running time is large.

We now improve the evaluation efficiency by combining Proposition 5.1 with the specific RAM garbling scheme of Theorem 3.5.

THEOREM 5.4. *Assume a subexponentially indistinguishable **iO** for circuits and subexponentially secure OWFs. There is an indistinguishability obfuscator for* TM *and* RAM*, where obfuscation of a machine R takes linear time in the space complexity*

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

Toward the above theorem, consider instantiating Proposition 5.1 using the RAM garbling scheme of Theorem 3.5 and a subexponentially secure **iO** scheme with quasi-linear complexity (implied by subexponentially secure **iO** and OWF as shown in section 5.4). Recall that the RAM garbling scheme of Theorem 3.5 has linear-space-dependent complexity $\mathrm{poly}(\lambda, |R|) \times S$ and evaluation time $\mathrm{poly}(\lambda, |R|) \times (T^* + S)$ with $T^* = T_R(x)$; such a garbling scheme leads to a randomized encoding algorithm REnc with the same encoding and decoding complexity. By the same efficiency analysis as in Proposition 5.1, this instantiation yields an **iO** for RAM with linear-space-dependent complexity, namely, $\mathrm{poly}(\lambda, |R|, n)S$. Therefore, its evaluation time is now $\mathrm{poly}(\lambda, |R|, n) \times S + \mathrm{poly}(\lambda, |R|) \times (T^* + S)$, which is $\mathrm{poly}(\lambda, |R|, n) \times (S + T^*)$.

Remark 5.5 (indistinguishability obfuscation with linear complexity in the program size). In section 3.3.1, we showed that the efficiency of our RAM garbling scheme can be improved to depend only linearly in program description size $|R|$, namely, it has garbling complexity of $\mathrm{poly}(\lambda)(|R| + S)$ and evaluation complexity of $\mathrm{poly}(\lambda)(|R| + S + T^*)$. When using such a RAM garbling scheme as the underlying scheme in our construction of **iO** scheme for RAM, we obtain an **iO** scheme with complexity $\mathrm{poly}(\lambda, n)(|R| + S)$ and evaluation time $\mathrm{poly}(\lambda, n)(|R| + S + T^*)$.

5.3. Publicly verifiable delegation, $\mathcal{SNARG}$s for P, and succinct NIZKs for NP. We now present the publicly verifiable delegation scheme for bounded space computations, following from our succinct randomized encodings, as well as a general transformation from delegation schemes to succinct noninteractive arguments. We also note the implications to succinct NIZKs as a corollary of our succinct **iO** and the work of [SW14].

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

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

Definition 5.6 (**P**-delegation). *A prover and verifier* $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ *constitute a delegation scheme for* **P** *if it satisfies the following:*

1. Completeness: *For any* $z = (AL, x, \lambda)$, *such that* AL *accepts* x *within* $AL.T$ *steps,*
$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathcal{G}(z) \\ \pi \leftarrow \mathcal{P}(z, \sigma)\end{array}\right] = 1\,.$$
2. Soundness: *For any polynomial-sized prover* $\mathcal{P}^*$, *polynomial* $T(\cdot)$, *there exists a negligible* $\alpha(\cdot)$ *such that for any* $z = (AL, x, \lambda)$, *such that* $AL.T \leq T(\lambda)$, *and* AL *does* not *accept* x *within* $AL.T$ *steps,*
$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathcal{G}(z) \\ \pi \leftarrow \mathcal{P}^*(z, \sigma)\end{array}\right] \leq \alpha(\lambda)\,.$$

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

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

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

Remark 5.7 (input privacy). Our construction achieves an additional property of input privacy which states that the first message of the delegation scheme σ leaks no information about the input x on which the computation of AL is being delegated, beyond the output $AL(x)$. This ensures that, in the outsourcing computation application, the server performing the computation learns no more than is necessary about the input to the computation.

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

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

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

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

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

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

THEOREM 5.8. *If $(\mathsf{REnc}, \mathsf{Dec})$ is a randomized encoding scheme with optimal complexity (resp., space-dependent complexity), then $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ as described above is a publicly verifiable $\mathbf{P}$-delegation scheme with optimal verification (resp., space-dependent verification).*

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

To show soundness, we will rely on the security of $(\mathsf{REnc}, \mathsf{Dec})$ and the one-wayness of f. Assume for contradiction there exist polynomial-size prover $\mathcal{P}^*$ and polynomial $p(\cdot)$ such that for infinitely many $z = (AL, x, \lambda)$, where AL does not accept x and $AL.T \leq p(\lambda)$, we have that

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

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

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

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

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

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

DEFINITION 5.9 ($\mathcal{SNARG}$). *A $\mathcal{SNARG}$ $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ is defined as a delegation scheme, with the following change to the syntax of $\mathcal{G}$: the generator $\mathcal{G}$ now gets as input a security parameter, time bound, and input bound $\lambda, T, n \in \mathbb{N}$, and does not get AL, x as before. We require the following:*

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

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

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

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

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

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

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

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

Remark 5.10 (nonadaptive soundness). Note that in the definition above and in our construction, we will consider only *nonadaptive* soundness, as opposed to *adaptive* soundness, where the malicious prover $\mathcal{P}^*$ can pick the statement z after seeing the first message σ.

We now show a simple transformation, based on **iO**, that takes any 2-message delegation scheme (e.g., the one constructed above) and turns it into a $\mathcal{SNARG}$ for **P**. The transformation works in either the public or private verification setting. Furthermore, it always results in a $\mathcal{SNARG}$ with strong soundness; even the delegation we start with does not have strong soundness (such as the scheme of [KRR14]).

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

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

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

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

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

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

THEOREM 5.11. *Assume the existence of an indistinguishability obfuscator $i\mathcal{O}$. If $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ is a privately verifiable (resp., publicly verifiable) **P**-delegation scheme, then $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ as described above is a privately verifiable (resp., publicly verifiable) $\mathcal{SNARG}$ with strong soundness. Moreover, if the delegation scheme has optimal or space-dependent verification and relative prover efficiency, then so does the $\mathcal{SNARG}$.*

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

To show strong soundness of $(\mathcal{P},(\mathcal{G},\mathcal{V}))$, we will rely on the soundness of $(\mathcal{P}_d,(\mathcal{G}_d,\mathcal{V}_d))$ and the security of **iO** and the punctured PRF $(\mathsf{PRF.Gen},\mathsf{PRF.Punc},\mathsf{F})$. We will first consider the privately verifiable setting. Assume for contradiction that there exist polynomial-sized oracle-aided prover $\mathcal{P}^*$, polynomials $T(\cdot)$,$n(\cdot)$,$p(\cdot)$ such that for infinitely many $z=(AL,x,\lambda)$, where $AL.T\leq T(\lambda)$, $|AL,x|\leq n(\lambda)$, and AL does not accept x,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We first observe that in the experiment corresponding to $\mathcal{A}_d(z, \mathcal{P}_d^{\mathsf{Hyb}})$, the event $\mathcal{V}_d(z, \sigma_d, \tau_d, \pi) = 1$ is equivalent to the event $\mathcal{V}(z, \sigma, \tau, \pi) = 1$, where $(\sigma, \tau) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z, \sigma_d, \tau_d)$. This follows directly from the construction of $\mathcal{V}$ and the fact that σ and τ are hardwired to output σ_d and τ_d on input z. Hence we have that

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

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

CLAIM 5.12. *The following ensembles are computationally indistinguishable:*

$$\{(\sigma, \tau) : (\sigma, \tau, \pi) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z, \sigma_d, \tau_d), (\sigma_d, \tau_d) \leftarrow \mathcal{G}_d(z)\}_{z \in \mathcal{Z}} \tag{5.1}$$

$$\approx_c \{(\sigma, \tau) : (\sigma, \tau) \leftarrow \mathcal{G}(\lambda, T(\lambda), n(\lambda))\}_{z \in \mathcal{Z}}. \tag{5.2}$$

Proof. Recall that in ensemble (5.1), $\sigma \leftarrow i\mathcal{O}(C^*_\sigma[K_z, \sigma_d])$, where K_z is a PRF key punctured at input z and C^*_σ on all input z outputs σ_d and on all other inputs z^* outputs $\mathcal{G}_d(z^*; \mathsf{F}(K_z, z^*))$. However, in ensemble (5.2), $\sigma \leftarrow i\mathcal{O}(C_\sigma[K])$, where C on input z^* outputs $\mathcal{G}_d(z^*; \mathsf{F}(K, z^*))$. The difference between τ in ensembles (5.1) and (5.2) is the same. Indistinguishability follows from the security of **iO** and that of $(\mathsf{PRF.Gen}, \mathsf{PRF.Punc}, \mathsf{F})$ in the standard way. We provide a brief overview.

Consider a hybrid ensemble that is identical to ensemble (5.1) except that instead of uniform randomness, $\mathcal{G}_d$ uses randomness from $\mathsf{F}(K, z)$, where K is a PRF key. K is then punctured at input z and given to $\mathcal{D}^{\mathsf{Hyb}}$ to use as K_z. By the security of the punctured PRF, this hybrid ensemble is indistinguishable from ensemble (5.1). Furthermore, the circuits obfuscated as σ and τ in this hybrid ensemble and in ensemble (5.2) are functionally equivalent. Hence, by the security of **iO**, ensemble (5.2) is indistinguishable from the hybrid ensemble. A hybrid argument completes the proof of the claim. ∎

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

5.3.3. Succinct perfect NIZK for NP. In this section we briefly note that by using the succinct indistinguishability obfuscator from section 5.2 in the construction of [SW14], we can obtain an NIZK argument scheme for any NP language $\mathcal{R}_L$ that is perfect zero-knowledge and additionally *succinct* in the following sense: Let $\Pi^{\mathcal{R}}$ be a uniform program that computes the NP relation $\mathcal{R}(x, w)$, and let $\tau(n)$ and $s(n)$ be, respectively, bounds on the length of witness and space needed by $\Pi^{\mathcal{R}}$ for n-bit statements. The length of the common reference string (CRS) of the scheme for proving n-bit statements grows polynomially with n, $\tau(n)$, and $s(n)$ (and is essentially independent of the verification time of the language). Below we provide a brief overview of the [SW14] construction and how it can be made succinct using succinct **iO**.

In [SW14], the NIZK scheme relies on **iO** for circuits as follows: The CRS contains an obfuscation of two circuits that contain the same PRF key. The first obfuscation is used by the Prover to generate proofs: the circuit takes as input a statement and witness (x, w) of lengths n and $\tau(n)$, and outputs the image of the input under the PRF as the proof if the witness is valid, that is, $\Pi^{\mathcal{R}}(x, w) = 1$. The second obfuscation is used by the Verifier to check if this proof is valid. [SW14] shows how to use this idea relying on **iO** and puncturable PRFs. In their construction, the length of the proof is succinct: it depends only on the security parameter. However, the length of the CRS is related to the size of the circuits obfuscated in the CRS, which is related to the verification time. We note that by obfuscating the pair of Turing machines that perform the above functionality, and using our succinct indistinguishability obfuscator instead, the length of the CRS can be made to depend on the statement and witness lengths, as well as the space complexity of the verification program, independent of the verification time.

Note that this succinct construction relies on our succinct indistinguishability obfuscator, which in turn relies on subexponentially secure **iO** for circuits (as opposed to standard **iO** for circuits that the [SW14] construction is based on).

THEOREM 5.13 (follows from [SW14]). *Assuming subexponentially secure **iO** for circuits and subexponentially secure OWFs, there exists an NIZK argument scheme for every NP language determined by a uniform polynomial-time program $\Pi^{\mathcal{R}}$ with the following properties:*

1. *The scheme is perfectly zero-knowledge.*
2. *The scheme has adaptive soundness.*[14]
3. *There are universal polynomials p, p', and p'', such that the length of the CRS of the scheme for verifying statements x of length n is $p(\lambda, n, \tau(n), s(n))$, where λ is the security parameter, $\tau(n)$ is a bound on the length of witness, and $s(n)$ is the space complexity of $\Pi^{\mathcal{R}}$ for verifying n-bit statements. The length of the proof is $p'(\lambda)$. The running time of the prover for statement and witness (x, w) is $p''(\lambda, n, \tau(n), s(n))T$, where T is the running time of $\Pi^{\mathcal{R}}(x, w)$.*

Remark 5.14 (improved efficiency of delegation and $\mathcal{SNARG}$s). When plugging in the more efficient garbling scheme of section 3.3.1, we directly obtain a publicly verifiable delegation scheme that the verification complexity is $\text{poly}(\lambda)(|AL| + AL.S + |x|)$ and the prover complexity for a T-time computation $AL(x)$ is $\text{poly}(\lambda)(|AL| + AL.S + T)$. Furthermore, combining this delegation scheme with Theorem 5.11, while applying the trick of "obfuscating in a piecemeal fashion" as in sections 3.3 and 5.2, we obtain a $\mathcal{SNARG}$ for **P** with the same verification and prover efficiency.

Finally, using the more efficiency succinct **iO** of Remark 5.5 in Theorem 5.13, the size of the CRS can be improved to $\text{poly}(\lambda, n, \tau(n)) \cdot s(n)$, and the prover efficiency can be improved to $\text{poly}(\lambda, n, \tau(n))(s(n) + T)$.

5.4. A new bootstrapping theorem. In general, when considering **iO** for circuits, the size of an obfuscation $|i\mathcal{O}(C)|$ (or more generally the time required to obfuscate) is allowed to be an arbitrary polynomial in the original circuit size $|C|$. In known candidate constructions (e.g., [GGH+13]) the blowup is quadratic (see the discussion in [GHRW14]). In this section, we show how to bootstrap **iO** for circuits (with arbitrary polynomial blowup) to obtain **iO** for circuits where the blowup is quasi-linear. The transformation has the additional feature that the underlying obfuscation scheme is only used for (simple) circuits whose size depends only on the security parameter and input.

The high-level idea. The transformation relies on ideas similar to those used in section 5.2 to construct succinct **iO** from succinct randomized encodings, which in turn go back to the bootstrapping technique of Applebaum [App14]. Concretely, we rely on plain randomized encodings [IK02, AIK06] for circuits known to have the following basic locality property: Given a circuit C with s gates and n-bit input x, computing a randomized encoding $\widehat{C}(x; R)$ can be decomposed into s computations $\widehat{C}_1(x; R), \ldots, \widehat{C}_s(x; R)$, each of fixed size ℓ independent of the circuit size $|C|$. In particular, each such computation $\widehat{C}_i(x; R)$ involves at most ℓ bits of the shared randomness R.

Similarly to the transformation in section 5.2, the transformation here is based on the basic idea of obfuscating the circuit that computes the randomized encoding $\widehat{C}(x; r)$ for any input x, while deriving the randomness R, by applying a puncturable PRF to the input x. The only difference is that, rather than obfuscating this circuit

[14] The perfect NIZK construction of [SW14] only satisfies nonadaptive soundness, but by a standard complexity leveraging trick it can be made to satisfy adaptive soundness. Since we assume subexponential security of the **iO** anyway, this comes at no cost to us.

as a whole, we separately obfuscate s smaller circuits computing the corresponding $\widehat{C}_i(x;R)$. To make sure that deriving the randomness is also local, we associate $r = |R|$ PRF keys $K_1, \ldots, K_r$ with each of the bits of the shared randomness R. Each one of the s obfuscated circuits only includes the PRF keys required for its local computation. The corresponding bits of randomness are again derived by applying the corresponding PRFs to the input x.

The gain is that the size of the resulting obfuscated circuit is thus $s \cdot \mathrm{poly}(\lambda, n)$, as required. The proof of security relies on a variant of the probabilistic **iO** argument invoked in section 5.2, with the difference that puncturing is performed simultaneously across all r PRF keys. Accordingly, it incurs a 2^n security loss in the input length n (which is polynomial when considering circuits with logarithmic-sized inputs, as is often the case in this work).

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

DEFINITION 5.15 (locality of randomized encodings). *A randomized encoding* $\mathcal{RE} = (\mathsf{REnc}, \mathsf{Dec})$ *for circuits is said to be local if*

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

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

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

Such randomized encodings can be constructed based on any one-way function [Yao86, AIK06].

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

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

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

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

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

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

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

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

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

PROPOSITION 5.16. *$i\mathcal{O}^*$ is a circuit obfuscator with quasi-linear blowup.*

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

We use a probabilistic **iO** argument similar to the one used in section 5.2. Concretely, given two circuits C, C' of the same size and functionality, we consider $2^n + 1$ hybrids that transition from $i\mathcal{O}^*(1^\lambda, C)$ to $i\mathcal{O}^*(1^\lambda, C')$. In the jth hybrid, the s obfuscations are with respect to hybrid circuits $\mathbb{C}_1^j, \ldots, \mathbb{C}_s^j$, where $\mathbb{C}_i^j$ uses $\widehat{C}_i$ for all inputs $x < j$ and $\widehat{C'}_i$ for all inputs $x \geq j$. Each two consecutive hybrids differ on only a single point, j. Similarly to section 5.2, we puncture the underlying PRFs at this point j and hardwire the values $\widehat{C}_1(x; R|_{S_1}), \ldots, \widehat{C}_s(x; R|_{S_s})$ (or $\widehat{C'}_1(x; R|_{S_1}), \ldots, \widehat{C'}_s(x; R|_{S_s})$, respectively), using true randomness instead of pseudorandomness. Then we can rely on the security of the randomized encodings to switch between the two.

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

Acknowledgments. We thank Boaz Barak and Guy Rothblum for their input regarding the plausibility of interactive proofs with fast verification (relevant to the plausibility of perfectly private succinct randomized encodings). We thank Daniel Wichs for discussing several aspects of [GHRW14]. We thank Stefano Tessaro for many delightful discussions at the early stage of the project. Finally, we thank the anonymous reviewers of STOC and SICOMP for their valuable comments.

REFERENCES

[ACC+15] P. ANANTH, Y.-C. CHEN, K.-M. CHUNG, H. LIN, AND W.-K. LIN, *Delegating RAM computations with adaptive soundness and privacy*, in Theory of Cryptography – TCC 2016, M. Hirt and A. Smith, eds., Lecture Notes in Comput. Sci. 9986, Springer, 2016, pp. 3–30.

[AIK04] B. APPLEBAUM, Y. ISHAI, AND E. KUSHILEVITZ, *Cryptography in NC^0*, in 45th Annual Symposium on Foundations of Computer Science (Rome, Italy, 2004), IEEE Computer Society Press, 2004, pp. 166–175.

[AIK06] B. APPLEBAUM, Y. ISHAI, AND E. KUSHILEVITZ, *Computationally private randomizing polynomials and their applications*, Comput. Complexity, 15 (2006), pp. 115–162.

[AIK10] B. APPLEBAUM, Y. ISHAI, AND E. KUSHILEVITZ, *From secrecy to soundness: Efficient verification via secure computation*, in ICALP 2010: 37th International Colloquium on Automata, Languages and Programming, Part I (Bordeaux, France, 2010), Lecture Notes in Comput. Sci. 6198, S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, eds., Springer, 2010, pp. 152–163.

[AIK11] B. APPLEBAUM, Y. ISHAI, AND E. KUSHILEVITZ, *How to garble arithmetic circuits*, in 52nd Annual Symposium on Foundations of Computer Science (Palm Springs, CA, 2011), R. Ostrovsky, ed., IEEE Computer Society Press, 2011, pp. 120–129.

[AJL+12] G. ASHAROV, A. JAIN, A. LÓPEZ-ALT, E. TROMER, V. VAIKUNTANATHAN, AND D. WICHS, *Multiparty computation with low communication, computation and interaction via threshold FHE*, in Advances in Cryptology – EUROCRYPT 2012 (Cambridge, UK, 2012), Lecture Notes in Comput. Sci. 7237, D. Pointcheval and T. Johansson, eds., Springer, 2012, pp. 483–501.

[AJS15] P. ANANTH, A. JAIN, AND A. SAHAI, *Indistinguishability Obfuscation with Constant Size Overhead*, IACR Cryptology ePrint Archive, 2015:1023, 2015.

[AJS17] P. ANANTH, A. JAIN, AND A. SAHAI, *Indistinguishability Obfuscation for Turing Machines: Constant Overhead and Amortization*, in Advances in Cryptology – CRYPTO 2017, J. Katz and H. Shacham, eds., Lecture Notes in Comput. Sci. 10402, Springer, 2017, pp. 252–279.

[App11a] B. APPLEBAUM, *Key-dependent message security: Generic amplification and completeness*, in Advances in Cryptology – EUROCRYPT 2011 (Tallinn, Estonia, 2011),

Lecture Notes in Comput. Sci. 6632, K. G. Paterson, ed., Springer, 2011, pp. 527–546.

[App11b] B. Applebaum, *Randomly encoding functions: A new cryptographic paradigm (invited talk)*, in Proceedings of the 5th International Conference on Information Theoretic Security, ICITS 2011 (Amsterdam, The Netherlands), 2011, pp. 25–31.

[App14] B. Applebaum, *Bootstrapping obfuscators via fast pseudorandom functions*, in Advances in Cryptology - ASIACRYPT 2014 - Proceedings of the 20th International Conference on the Theory and Application of Cryptology and Information Security (Kaoshiung, Taiwan, R.O.C.), 2014, pp. 162–172.

[BCCT12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, *From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again*, in ITCS 2012: 3rd Innovations in Theoretical Computer Science (Cambridge, MA, 2012), S. Goldwasser, ed., Association for Computing Machinery, 2012, pp. 326–349.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, *Recursive composition and bootstrapping for SNARKS and proof-carrying data*, in 45th Annual ACM Symposium on Theory of Computing (Palo Alto, CA, 2013), D. Boneh, T. Roughgarden, and J. Feigenbaum, eds., ACM Press, 2013, pp. 111–120.

[BCC+17] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, and E. Tromer, *The hunting of the SNARK*, J. Cryptology, 30 (2017), pp. 989–1066.

[BCP14] E. Boyle, K.-M. Chung, and R. Pass, *On extractability obfuscation*, in TCC 2014: 11th Theory of Cryptography Conference (San Diego, CA, 2014), Lecture Notes in Comput. Sci. 8349, Y. Lindell, ed., Springer, 2014, pp. 52–73.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, *On the (im)possibility of obfuscating programs*, in Advances in Cryptology CRYPTO 2001, Springer, 2001, pp. 1–18.

[BGI14] E. Boyle, S. Goldwasser, and I. Ivan, *Functional signatures and pseudorandom functions*, in PKC, 2014, pp. 501–519.

[BGL+15] N. Bitansky, S. Garg, H. Lin, R. Pass, and S. Telang, *Succinct randomized encodings and their applications*, in Proceedings of the 47th Annual ACM on Symposium on Theory of Computing, STOC 2015 (Portland, OR), ACM, 2015, pp. 439–448.

[BHHI10] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai, *Bounded key-dependent message security*, in Advances in Cryptology – EUROCRYPT 2010 (French Riviera, 2010), Lecture Notes in Comput. Sci. 6110, H. Gilbert, ed., Springer, 2010, pp. 423–444.

[BHK13] M. Bellare, V. T. Hoang, and S. Keelveedhi, *Instantiating random oracles via UCEs*, in Advances in Cryptology – CRYPTO 2013, Part II (Santa Barbara, CA, 2013), Lecture Notes in Comput. Sci. 8043, R. Canetti and J. A. Garay, eds., Springer, pp. 398–415.

[BHR12a] M. Bellare, V. T. Hoang, and P. Rogaway, *Adaptively secure garbling with applications to one-time programs and secure outsourcing*, in Advances in Cryptology – ASIACRYPT 2012 (Beijing, China, 2012), Lecture Notes in Comput. Sci. 7658, X. Wang and K. Sako, eds., Springer, 2012, pp. 134–153.

[BHR12b] M. Bellare, V. T. Hoang, and P. Rogaway, *Foundations of garbled circuits*, in The ACM Conference on Computer and Communications Security, CCS'12 (Raleigh, NC), 2012, pp. 784–796.

[BLMR13] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan, *Key homomorphic PRFs and their applications*, in Advances in Cryptology – CRYPTO 2013, Part I (Santa Barbara, CA, 2013), Lecture Notes in Comput. Sci. 8042, R. Canetti and J. A. Garay, eds., Springer, 2013, pp. 410–428.

[BW13] D. Boneh and B. Waters, *Constrained pseudorandom functions and their applications*, in ASIACRYPT (2), 2013, pp. 280–300.

[CCC+16] Y.-C. Chen, S. S. M. Chow, K.-M. Chung, R. W. F. Lai, W.-K. Lin, and H.-S. Zhou, *Cryptography for Parallel RAM from Indistinguishability Obfuscation*, IACR Cryptology ePrint Archive, Report 2015/406, 2015, https://eprint.iacr.org/2015/406.

[CCHR15] R. Canetti, Y. Chen, J. Holmgren, and M. Raykova, *Succinct Adaptive Garbled RAM*, Cryptology ePrint Archive, Report 2015/1074, 2015, https://eprint.iacr.org/2015/1074.

[CH16] R. Canetti and J. Holmgren, *Fully succinct garbled RAM*, in Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (Cambridge, MA, 2016), MIT, 2016, pp. 169–178, http://doi.acm.org/10.1145/2840728.2840765.

[CHJV15] R. Canetti, J. Holmgren, A. Jain, and V. Vaikuntanathan, *Succinct garbling and*

indistinguishability obfuscation for RAM programs, in Proceedings of the 47th Annual ACM Symposium on Theory of Computing, STOC 2015 (Portland, OR), 2015, pp. 429–437.

[CIJ+13] A. De Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano, *On the achievability of simulation-based security for functional encryption*, in Advances in Cryptology - CRYPTO 2013, Part II (Santa Barbara, CA, 2013), Springer, 2013, pp. 519–535.

[CLP13] K.-M. Chung, H. Lin, and R. Pass, *Constant-round concurrent zero knowledge from p-certificates*, in 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013 (Berkeley, CA), 2013, pp. 50–59.

[CLTV15] R. Canetti, H. Lin, S. Tessaro, and V. Vaikuntanathan, *Obfuscation of probabilistic circuits and applications*, in Theory of Cryptography - 12th Theory of Cryptography Conference, Part II, TCC 2015 (Warsaw, Poland), 2015, pp. 468–497.

[CP13] K.-M. Chung and R. Pass, *A Simple ORAM*, IACR Cryptology ePrint Archive, 2013:243, 2013.

[DFH12] I. Damgård, S. Faust, and C. Hazay, *Secure two-party computation with low communication*, in TCC 2012: 9th Theory of Cryptography Conference (Sicily, Italy, 2012), Lecture Notes in Comput. Sci. 7194, R. Cramer, ed., Springer, 2012, pp. 54–74.

[DHRW16] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs, *Spooky encryption and its applications*, in Advances in Cryptology - CRYPTO 2016, Part III (Santa Barbara, CA), Springer, 2016, pp. 93–122.

[Gen09] C. Gentry, *Fully homomorphic encryption using ideal lattices*, in 41st Annual ACM Symposium on Theory of Computing (Bethesda, MD, 2009), M. Mitzenmacher, ed., ACM Press, pp. 169–178.

[GGH+13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, *Candidate indistinguishability obfuscation and functional encryption for all circuits*, in 54th Annual Symposium on Foundations of Computer Science (Berkeley, CA, 2013), IEEE Computer Society Press, 2013, pp. 40–49.

[GGHR14] S. Garg, C. Gentry, S. Halevi, and M. Raykova, *Two-round secure MPC from indistinguishability obfuscation*, in TCC 2014: 11th Theory of Cryptography Conference (San Diego, CA, 2014), Lecture Notes in Comput. Sci. 8349, Y. Lindell, ed., Springer, 2014, pp. 74–94.

[GGHW13] S. Garg, C. Gentry, S. Halevi, and D. Wichs, *On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input*, in Advances in Cryptology - CRYPTO 2014 (Santa Barbara, CA, 2014), Springer, 2014, pp. 518–535.

[GGHZ16] S. Garg, C. Gentry, S. Halevi, and M. Zhandry, *Functional encryption without obfuscation*, in Theory of Cryptography - 13th Theory of Cryptography Conference, Part II, Lecture Notes in Comput. Sci. 9563, Springer, 2016, pp. 480–511.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.

[GGP10] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers*, in Advances in Cryptology – CRYPTO 2010 (Santa Barbara, CA, 2010), Lecture Notes in Comput. Sci. 6223, T. Rabin, ed., Springer, 2010, pp. 465–482.

[GHL+14] C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs, *Garbled RAM revisited*, in Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques (Copenhagen, Denmark), 2014, pp. 405–422.

[GHRW14] C. Gentry, S. Halevi, M. Raykova, and D. Wichs, *Outsourcing private RAM computation*, in 55th Annual Symposium on Foundations of Computer Science, 2014.

[GKP+13] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, *Reusable garbled circuits and succinct functional encryption*, in 45th Annual ACM Symposium on Theory of Computing (Palo Alto, CA, 2013), D. Boneh, T. Roughgarden, and J. Feigenbaum, eds., ACM Press, 2013, pp. 555–564.

[GLOS15] S. Garg, S. Lu, R. Ostrovsky, and A. Scafuro, *Garbled RAM from one-way functions*, in 47th Annual ACM Symposium on Theory of Computing, ACM Press, 2015.

[HW15] P. Hubácek and D. Wichs, *On the communication complexity of secure function evaluation with long output*, in ITCS, ACM Press, 2015, pp. 163–172.

[IK00] Y. Ishai and E. Kushilevitz, *Randomizing polynomials: A new representation with applications to round-efficient secure computation*, in 41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000), IEEE Computer Society Press, 2000, pp. 294–304.

[IK02] Y. Ishai and E. Kushilevitz, *Perfect constant-round secure computation via perfect randomizing polynomials*, in Automata, Languages and Programming, 29th International Colloquium, ICALP 2002 (Malaga, Spain), 2002, pp. 244–256.

[KLW15] V. Koppula, A. B. Lewko, and B. Waters, *Indistinguishability obfuscation for Turing machines with unbounded memory*, in 47th Annual ACM Symposium on Theory of Computing, ACM Press, 2015.

[KPTZ13] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, *Delegatable pseudorandom functions and applications*, in CCS, 2013, pp. 669–684.

[KRR14] Y. T. Kalai, R. Raz, and R. D. Rothblum, *How to delegate computations: The power of no-signaling proofs*, in 46th Annual ACM Symposium on Theory of Computing, D. B. Shmoys, ed., ACM Press, 2014, pp. 485–494.

[Kv99] A. Klivans and D. van Melkebeek, *Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses*, in 31st Annual ACM Symposium on Theory of Computing (Atlanta, GA, 1999), ACM Press, 1999, pp. 659–667.

[LO13] S. Lu and R. 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), 2013, pp. 719–734.

[LTV12] A. López-Alt, E. Tromer, and V. Vaikuntanathan, *On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption*, in 44th Annual ACM Symposium on Theory of Computing, H. J. Karloff and T. Pitassi, eds., ACM Press, 2012, pp. 1219–1234.

[Mic00] S. Micali, *Computationally sound proofs*, SIAM J. Comput., 30 (2000), pp. 1253–1298, https://doi.org/10.1137/S0097539795284959.

[MV99] P. B. Miltersen and N. V. Vinodchandran, *Derandomizing Arthur-Merlin games using hitting sets*, in 40th Annual Symposium on Foundations of Computer Science (New York, 1999), IEEE Computer Society Press, 1999, pp. 71–80.

[MW16] P. Mukherjee and D. Wichs, *Two round multiparty computation via multi-key FHE*, in Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part II (Vienna, Austria), 2016, pp. 735–763.

[Nao03] M. Naor, *On cryptographic assumptions and challenges (invited talk)*, in Advances in Cryptology – CRYPTO 2003 (Santa Barbara, CA, 2003), Lecture Notes in Comput. Sci. 2729, D. Boneh, ed., Springer, 2003, pp. 96–109.

[PF79] N. Pippenger and M. J. Fischer, *Relations among complexity measures*, J. ACM, 26 (1979), pp. 361–381.

[PRV12] B. Parno, M. Raykova, and V. Vaikuntanathan, *How to delegate and verify in public: Verifiable computation from attribute-based encryption*, in TCC 2012: 9th Theory of Cryptography Conference (Sicily, Italy, 2012), Lecture Notes in Comput. Sci. 7194, R. Cramer, ed., Springer, 2012, pp. 422–439.

[Rog91] P. Rogaway, *The Round Complexity of Secure Protocols*, Ph.D. thesis, Massachusetts Institute of Technology, 1991.

[SCSL11] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, *Oblivious RAM with $O((\log N)^3)$ worst-case cost*, in ASIACRYPT, 2011, pp. 197–214.

[SW14] A. Sahai and B. Waters, *How to use indistinguishability obfuscation: Deniable encryption, and more*, in Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, D. B. Shmoys, ed., ACM Press, 2014, pp. 475–484.

[Wat14] B. Waters, *A Punctured Programming Approach to Adaptively Secure Functional Encryption*, Cryptology ePrint Archive, Report 2014/588, 2014.

[Yao86] A. C.-C. Yao, *How to generate and exchange secrets (extended abstract)*, in 27th Annual Symposium on Foundations of Computer Science (Toronto, Ontario, Canada, 1986), IEEE Computer Society Press, 1986, pp. 162–167.