Succinct Randomized Encodings and their Applications[*]

Nir Bitansky[†] Sanjam Garg[‡] Sidharth Telang[§]

September 30, 2014

Abstract

A *randomized encoding* allows to represent a "complex" function $f(x)$ by a "simpler" randomized function $\hat{f}(x;r)$ whose output distribution encodes $f(x)$, while revealing nothing else regarding x. Existing randomized encodings, geared mostly to allow encoding with low parallel complexity, have proven instrumental in various strong applications such as multiparty computation and parallel cryptography.

This work focuses on another natural complexity measure: *the time required to encode*. We construct *succinct randomized encodings* where a computation given by a (Turing or random-access) machine M, and input x, requiring time t and space s, can be encoded roughly in time $\mathrm{poly}(|x|, \log t, s)$, thus inducing significant savings in time when $s \ll t$. The scheme guarantees computational input-privacy and is based on indistinguishability obfuscation for a relatively simple circuit class, which can in turn be based on a polynomial version of the subgroup elimination assumption on multilinear graded encodings.

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

- Indistinguishability obfuscation for uniform (Turing or random-access) machines, where the obfuscated machine $i\mathcal{O}(M)$ computes the same function as M for inputs x of apriori-fixed maximal size n, and is computed in time $\mathrm{poly}(n, \log t, s)$.
- Functional encryption for uniform machines, where a functional decryption key corresponding to M allows decrypting $M(x)$ from encryptions of x. As in the previous case, inputs x are of apriori-fixed maximal size n, and key derivation time is roughly $\mathrm{poly}(n, \log t, s)$.
- Publicly-verifiable 2-message delegation where verification time is roughly $\mathrm{poly}(n, \log t, s)$. We also show how to transform any 2-message delegation scheme to an essentially non-interactive system where the verifier message is reusable.

For the first application, we also require subexponentially-secure indistinguishability obfuscation for circuits, and for the second polynomial indistinguishability obfuscation, which can be replaced by more concrete polynomial hardness assumptions on multilinear graded-encodings. Previously, both applications were only known based on various non-standard knowledge assumptions.

[*]Part of this research was done at IBM T.J. Watson Research Center.

[†]MIT `nirbitan@csail.mit.edu`

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

[§]Cornell University `sidtelang@cs.cornell.edu`

1 Introduction

The notion of a *randomized encoding*, pioneered by Ishai and Kushilevitz [IK00], aims to trade the computation of a "complex" function $f(x)$ for the computation of a "simpler" randomized function $\hat{f}(x;r)$ whose output distribution encodes $f(x)$, but hides anything else regarding the input x. The "complexity" of computing f is shifted to the decoding procedure that should extract $f(x)$ from $\hat{f}(x,r)$.

The privacy of the input x is naturally captured by an efficient simulator $\mathsf{Sim}(f(x))$, who given only the output $f(x)$, produces a simulated encoding indistinguishable from $f(x;r)$; privacy can be perfect, statistical, or computational, according to the attained indistinguishability. Capturing what it means to "simplify the computation of $f(x)$" may take quite different forms according to the complexity measure of interest. Most previous work have focused on computing the randomized encoding $\hat{f}(x;r)$ with lower parallel-time complexity than required for computing the original function f, and has been quite successful. In particular, all log-space computations (and $\mathbf{NC}^1$ computations) were shown to have perfectly-private randomized encodings in $\mathbf{NC}^0$ [IK00, IK02]. When settling for privacy against computationally bounded adversaries, the latter extends to arbitrary poly-time computations [AIK06] (this is also known as Yao's garbled circuit method [Yao82]). The constructed randomized encodings were in turn shown to have various strong applications to parallel cryptography, secure computation, verifiable delegation, and more (most of these are surveyed in [App11b]).

Succinct randomized encodings. In this work, we focus on yet another natural complexity measure: *the time required to compute* $f(x)$. Specifically, given the description of f and the input x, we would like to compute the encoding $\hat{f}(x;r)$ in time $\hat{t}$ that is significantly smaller than the time t required to compute $f(x)$. Decoding, in contrast, may already be as large as t, perhaps with some tolerable overhead. For this goal to be achievable, f has to be given in some succinct representation that is smaller than t, and cannot be given by, say, a size-t circuit. Concretely, we restrict attention to the natural case that f is represented by a uniform (deterministic) machine M, e.g., a Turing machine (TM) or a random-access machine (RAM).

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

Beyond shifting computation from weak parties to strong parties, succinct randomized encodings may sometimes save in computation altogether. For instance, one of the first demonstrated applications of randomized encodings [IK00, IK02] was in multi-party computation (MPC). Indeed, most known MPC solutions explicitly utilize the circuit $C_f(x_1,\ldots,x_m)$ representing a function $f(x_1,\ldots,x_m)$, and the overhead they incur, e.g. in communication, may depend on the circuit size $|C|$. When the function f is succinctly represented by a machine M, we may have the parties compute first a succinct randomized $\widehat{M}(x_1,\ldots,x_m)$, and only decode at the end, thereby making communication overhead proportional to the smaller circuit that computes $\widehat{M}$. Furthermore, the effort of decoding (proportional to M's running time) falls only on the parties that obtain the output.

Do succinct randomized encodings exist? Under commonly believed complexity-theoretic assumptions, perfectly-private randomized encodings for say all of **P** are unlikely to be computable too fast, e.g. in fixed polynomial time.[1] In contrast, restricting attention to privacy against computationally-bounded adversaries, no lower bounds or barriers are known. Still, all the constructions introduced so far were based on considerably strong computational assumptions such as extractable witness encryption and succinct non-interactive arguments in [GKP^{+}13a], and (a strong variant of) differing-inputs obfuscation in [GHRW14]. For once, in the language of [Nao03] these assumptions are not *efficiently-falsifiable*; furthermore, in some cases they are subject to impossibilities [BCPR14, GGHW14].

In light of the above, we focus on two main questions:

Can we base succinct randomized encodings on more standard assumptions?
What are the applications of succinct randomized encodings?

1.1 Contributions

Our core contribution is a new construction of succinct randomized encodings based on assumptions that are considered more standard (and in particular are efficiently falsifiable). Concretely, we show

Theorem 1.1 (informal). *Assuming indistinguishability obfuscation (IO) and one-way functions, there exist randomized encodings where for a (Turing or random-access) machine M with running time t and space s on input x, it holds that for some fixed polynomial* $\mathrm{poly}(\cdot)$*, independent of M and t:*

- *encoding takes time* $\mathrm{polylog}(t) \cdot \mathrm{poly}(|x|, |M|, s, \lambda)$*,*
- *decoding takes time* $\widetilde{O}(t) \cdot \mathrm{poly}(|x|, |M|, s, \lambda)$ *and space* $\mathrm{polylog}(t) \cdot \mathrm{poly}(|x|, |M|, s, \lambda)$*.*

Furthermore, IO is only required for a relatively simple class of circuits. First, assuming puncturable pseudorandom functions (PRFs) in $\mathbf{NC}^1$, the obfuscated circuits are also in $\mathbf{NC}^1$. Second, and regardless of depth, the obfuscated circuits only have $O(\log t)$-size inputs. Puncturable PRFs in $\mathbf{NC}^1$ are constructed in [BLMR13] based on decision linear assumptions on multilinear maps, and in [HKW14] based on a standard variant of decision Diffie Hellman assumption. Further more Gentry et al. [GLSW14] construct indistinguishability obfuscation for $\mathbf{NC}^1$ under a subgroup elimination assumption on composite-order multilinear graded encodings [GGH13a, CLT13], where the security of the assumption scales exponentially with the circuit input length. Thus, to get randomized encodings for $\mathbf{P}$, we only need to rely on *polynomial assumptions*.

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

Succinct indistinguishability obfuscation. Typically obfuscation (and in particular IO) operates on circuits and outputs circuits of polynomially related size. Naturally, we may also want to consider obfuscation in uniform models of computation such as Turing or random-access machine. Here, given a machine M, and a bound n on input-length, we would like to produce an obfuscated machine $\mathcal{O}(M)$ that has the same functionality on all inputs of size at most n. Crucially, the obfuscation procedure should be done much

[1]Specifically, it can be shown that, for a language $\mathcal{L}$, recognized by a given $T(n)$-time Turing machine M, 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 verification time should at least depend on the space (or depth) of the computation. Furthermore, under commonly believed derandomization assumptions (used to show that $\mathbf{AM} \subseteq \mathbf{NP}$ [Kv99, MV99, SU01]), the above would imply that $\mathcal{L}$ can be non-deterministically decided in time $\mathrm{poly}(t(n))$, for some fixed polynomial poly. Thus, any speedup in encoding would imply related speedup by non-determinism, whereas significant speedup is believed to be unlikely.

faster then the machine's maximal running time $T(n)$ (and corresponding circuit size).[2] As in succinct randomized encodings, evaluating $\mathcal{O}(M)(x)$ may be proportional to the running time of $M(x)$. We call such obfuscators *succinct*.

Our first (and perhaps the most surprising) application of succinct randomized encodings is for *succinct indistinguishability obfuscation* (succinct IO in short). Indistinguishability here means that the (succinct) obfuscations of two machines that agree on all inputs $x \in \{0,1\}^{\leq n}$, will be computationally indistinguishable. Differently from (non-succinct) IO for circuits, any form of succinct IO realized so far [ABG$^+$13, BCP14] assumes differing-inputs obfuscation in conjunction with succinct non-interactive arguments (which already entail strong succinctness properties); as mentioned before, these are considered very strong up to implausible in certain settings.

Relying on succinct randomized encodings, we show a construction based on essentially different assumptions (which are typically considered more standard). Concretely, we show:

Theorem 1.2 (informal)**.** *Assuming succinct randomized encodings, one-way functions, and IO for circuits that are all $2^{\lambda^{\varepsilon}}$-secure, there exists succinct IO where for machine M and input bound $n \leq \lambda^{\varepsilon/2}$:*

- *obfuscating M takes roughly the same time as encoding $M(x)$,*
- *evaluating the obfuscated $\mathcal{O}(M)$ takes roughly the same time and space as decoding.*

As a corollary of Theorems 1.1,1.2, we obtain, from subexponentially-secure IO and one-way functions, succinct IO where the time to obfuscate depends only on the space of M, and is essentially independent of the running time of M. Here also, by assuming puncturable PRFs in $\mathbf{NC}^1$, we can rely on (subexponential) IO only for $\mathbf{NC}^1$.

We note that this theorem is somewhat analogous to a bootstrapping theorem by Applebaum [App13] who showed, for a strong notion of virtual black-box obfuscation [BGI$^+$01], how (non-succinct) randomized encodings and pseudo-random functions in $\mathbf{NC}^1$, together with obfuscation for $\mathbf{NC}^1$ circuits, imply virtual black-box obfuscation for $\mathbf{P}/\mathrm{poly}$. Using the technique used to prove Theorem 1.2, and subexponentially-secure pseudo-random functions in $\mathbf{NC}^1$, we can extend this bootstrapping theorem to also hold for the weaker notion of IO.

Succinct functional encryption. The goal of functional encryption (FE) [O'N10, BSW11] is to allow fine-grained access control to encrypted data. Concretely, in an FE scheme for a class of functions $\mathcal{F}$, the trusted holder of the (master) secret key SK can derive a special key SK_f for any function $f \in \mathcal{F}$. Given an encryption of some m, anyone that holds SK_f can learn $f(m)$, but should not learn anything else about m.

The recent leap in the study of obfuscation [GGH$^+$13b], has brought with it a corresponding leap in functional encryption. Today, functional encryption for all circuits can be constructed from IO [GGH$^+$13b, Wat14], or even from concrete (and efficiently falsifiable) assumptions on composite order multilinear graded-encodings [GGHZ14]. Considering function families in uniform models of computation, we may hope to have *succinct FE*, where the secret key SK_M corresponding to machine M can be computed faster than the running time of M. However, here the state is similar to succinct randomized encodings, or succinct IO, requiring essentially the same non-standard assumptions.

One can replace IO in the above constructions for circuits with the succinct IO from Theorem 1.2, and obtain FE where computing SK_M is comparable to (succinctly) obfuscating M. This, however, will incur the same subepxonential loss as in the construction of succinct IO. We show, however, that we can directly construct succinct FE, while circumventing this loss:

[2]One can think of a stronger model where M and accordingly $\mathcal{O}(M)$ can deal with inputs of a-priori unbounded length. We will restrict attention to the simpler bounded-input model described above.

Theorem 1.3 (informal)**.** *Assuming succinct randomized encodings, one-way functions, and IO for circuits, there exists succinct functional encryption where for machine M and input bound n:*

- *deriving a corresponding secret key SK_M takes roughly the same time as encoding $M(x)$.*
- *decrypting takes roughly the same time and space as decoding.*

Assuming puncturable PRFs in $\mathbf{NC}^1$, IO can be restricted to $\mathbf{NC}^1$, or replaced with the same (efficiently-falsifiable) assumptions on composite order multilinear graded encodings made in [GGHZ14].

As a corollary of Theorems 1.1,1.3, we obtain succinct FE where the key-derivation time depends only on the space of M, and is essentially independent of the running time of M. We note that the alternative construction based on [GGHZ14], the resulting scheme is fully secure as in [GGHZ14].

We also note that, given succinct FE, we can somewhat enhance our basic succinct randomized encodings to make them *reusable*. Such reusability was previously studied in [GKP$^+$13b, GHRW14] and means that encoding is now split into two parts: the first part $\widehat{M}$ is independent of the specific input x and only depends on the machine M. The first part can then be "reused" together with a second part $\widehat{x}$ that includes an encoding of input x. The gain is in efficiency: while encoding M depends on the space in our solution, it is only done once, and subsequent encodings of inputs depend only on the input size $|x|$ and not on space.

Publicly-verifiable delegation. In the basic setting of delegation, a weak client would like to delegate a long computation succinctly represented by a machine M, and input x to a server, and be able to later verify the result claimed by the server. Verification is required to be significantly faster than running the computation from scratch, and ideally we also want the corresponding protocol to only involve one message in each direction, a verifier message σ and a prover message including $M(x)$ and a proof π of correctness.

Recently, Kalai, Raz, and Rothblum [KRR14] gave a general solution to the problem, under the subexponential learning with errors assumption, where verification is essentially linear in the input x. Other solutions [GLR11, BCCT12, DFH12, BCCT13] rely on non-standard knowledge assumptions.

Based on the succinct randomized encodings from Theorem 1.1, we obtain a solution that has two additional desired features. The first is *public-verifiability*, meaning that given the (public) message σ anyone can verify the proof π, without requiring any secret verification state. The second is input-privacy, meaning that the server does not learn anything regarding the verifier's input.

Assuming in addition, indistinguishability obfuscation, we can turn the system into what is known as a *succinct non-interactive argument* (SNARG), where the verifier's message σ is *reusable*; namely, the verifier can send one message σ, and then get proofs for multiple different computations. (This is, in fact, a generic transformation the can be applied to any delegation scheme.)

Theorem 1.4 (informal)**.** *Assuming succinct randomized encodings and one-way functions, there exists publicly-verifiable 2-message delegation with input-privacy where for machine M and input bound n:*

- *verification takes roughly the same time as encoding $M(x)$.*
- *proving takes roughly the same time and space as decoding.*

Assuming also IO for $\mathbf{NC}^1$ and one-way functions, the verifier message can be made reusable.

Invoking the succinct randomized encodings constructed here, we obtain a system as above where verification time only depends on the space of the computation, but not its time. Also, like the succinct randomized encoding, the solution can also be based on a polynomial version of the subgroup elimination assumption (rather than subexponential hardness assumptions).

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

One application, already mentioned above, is to multiparty computation [IK00, IK02], where we can reduce the communication overhead from depending on the circuit size required to compute a multiparty $f(x_1, \ldots, x_m)$ to depending on the space required to compute f, which can be much smaller. While focusing merely on communication this problem has by now general 1-round solutions based on (multi-key) fully-homomorphic encryption [Gen09, AJL$^+$12, LTV12, GGHR14], succinct randomized encodings also allow to shift the work load to one party (the decoder), without inducing extra rounds or strong computational assumptions (such as succinct non-interactive arguments of knowledge).

Another application is to amplification of *key-dependent message security* (KDM). In KDM encryption schemes we want semantic security to hold, even when the adversary obtains encryptions of functions of the secret key taken from a certain class $\mathcal{F}$. Applebaum [App11a] essentially shows that given a 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 $\mathcal{F}$. Using succinct randomized encodings, we can roughly show that given KDM-security for circuits of some fixed polynomial size s (such as the scheme of [BHHI10]), can be amplified to KDM-security for functions that can be computed by machines with space $\ll s$, but could potentially have much larger running time.

1.2 Overview of Techniques

We now overview the main ideas behind our results, focusing on the construction of succinct randomized encodings from IO. We then sketch how the main applications are obtained.

The basic idea: obfuscating a Yao gate garbler. Our starting point is a good old non-succinct randomized encoding — Yao's garbled circuit scheme [Yao82]. The scheme takes an arbitrary circuit C and input x and computes a randomized encoding $\widehat{C}(x)$, usually referred to as the garbled circuit. While the garbled circuit is known to be computable in $\mathbf{NC}^0$ (under reasonable assumptions), the resulting $\widehat{C}(x)$ is as large as the circuit C. Nevertheless, the fact that garbling is done in $\mathbf{NC}^0$ implies that the computation of each output bit is very local, and can be done by a circuit of fixed size. At a very high-level, the basic idea is, given a uniform machine, to cram all the corresponding local computations into one small obfuscated circuit that can internally generate randomness and produce each of the small pieces of $\widehat{C}(x)$. Intuitively, strong enough obfuscation would guarantee that this circuit leaks nothing beyond $\widehat{C}(x)$ and can be published as the succinct randomized encoding. We next sketch in more detail how this is done.

Looking into how Yao garbling is done, note that the garbling of the large circuit C consists of garbling each gate separately; furthermore, this garbling is local in the sense that, for each gate, it involves a small amount of randomness that will only be used in the garbling process of the few neighbouring gates. A bit more concretely, in Yao's scheme, each wire w in the circuit is associated with two random keys $K_w^0, K_w^1 \in \{0,1\}^\lambda$, whose length depends only on the security parameter λ. The gate garbling procedure $G(\odot, \{K_w^0, K_w^1 : w \in g\})$ takes as input a binary operation $\odot$ corresponding to a gate g, and the constant number of keys corresponding to the gate's input and output wires. The procedure then generates a garbled gate $\widehat{g}$, and the garbled computation $\widehat{C}(x)$ consists of all the garbled gates $\{\widehat{g} : g \in C\}$, and the keys $\{K_i^{x_i} : i \in [n]\}$ corresponding to input wires $[n]$ and their value according to the input x.[3]

This gives rise to the following natural approach: given a uniform (say, Turing) machine M that for inputs of length n runs in time t and space s, we consider the circuit C of size $O(t \cdot s)$, with bounded fan-in

[3]The reader who is familiar with Yao may identify $\widehat{g}$ with the encryptions table corresponding to g. Here, we will not need to dive in to how $\widehat{g}$ is computed, and will be more interested the guaranteed security properties. Details may be found in [LP09].

and fan-out, that represents M's computation (for simplicity of exposition, we assume that M has a constant size description). The circuit C, in turn, can be represented by a small circuit C_{sc} of size $O(\log t)$, which given a label $i \in [O(t \cdot s)]$ for any gate g_i in the circuit, outputs the corresponding gate operation $\odot$ and the labels of its corresponding wires $\{w \in g_i\}$. We can then obfuscate an augmented gate garbling circuit $G_{M,K}$ that will garble any given gate g_i, using (pseudo) random keys internally generated by a pseudorandom function F_K. That is, $G_{M,K}$ is given a label $i \in [O(t \cdot s)]$, invokes $C_{\mathsf{sc}}(i)$ to learn the corresponding operation $\odot$ and wires $\{w \in g_i\}$, internally derive the required keys $(K_w^0, K_w^1) = F_K(w)$, and invoke G to create the garbled $\widehat{g}_i$. We then publish the obfuscated $\mathcal{O}(G_{M,K})$ together with the input keys $\{K_i^{x_i}\}$, derived consistently using F_K.

In terms of functionality and succinctness this seems like a promising solution. The size of the resulting randomized encoding, and more generally the time it takes to compute it, is polynomial in the size of $G_{M,K}$, which is in turn polynomial in the circuit size of G and F_K, which are both $\mathrm{poly}(\lambda)$, and the size of C_{sc}, which is $O(\log t)$; thus, the overall time of encoding is $\mathrm{poly}(\lambda, \log t)$. Decoding involves running $\mathcal{O}(G_{M,K})$ on each $i \in [O(t \cdot s)]$, thus deriving the garbled $\widehat{C}(x)$ and then running the decoding procedure for garbled circuits; in total this takes time $t \cdot s \cdot O(\log t + \mathrm{poly}(\lambda))$.

Intuitively, to prove that the above succinct randomized encoding is secure, we would like to rely on the security of the underlying circuit garbling and that of the obfuscator $\mathcal{O}$. The question is

what kind of security do we have to require from $\mathcal{O}$?

To answer this question, let us first recall the security guarantee of Yao, which says that an efficient simulator $\mathsf{Sim}(C, y)$, given only the output $y = C(x)$, but nothing about input x, can generate a simulated garbled circuit $\widehat{C}^{\mathsf{sim}}$ that is computationally indistinguishable from $\widehat{C}(x)$. Furthermore, similarly to the real garbling process, also the simulated one is local; namely, there exists a small circuit $G^{\mathsf{sim}}(\odot, \left\{K_w^0, K_w^1 : w \in g\right\}, y)$ that is also given the output y, and creates simulated garbled gates. In addition, the real garbled keys $\{K_i^{x_i}\}$ are replaced with simulated keys $\left\{K_i^0 : i \in [n]\right\}$ that can correspond to the all zero input 0^n (or any other string for that matter). Thus, a natural simulation strategy is to produce an obfuscation $\mathcal{O}(G^{\mathsf{sim}}_{M,K,y})$ of a simulated gate garbler $G^{\mathsf{sim}}_{M,K,y}$ that is defined just like $G_{M,K}$, only that it invokes G^{sim} rather than G.

We would now like to argue that the real encoding $\{K_i^{x_i}\}, \mathcal{O}(G_{M,K})$ is computationally indistinguishable from the simulated $\left\{K_i^0\right\}, \mathcal{O}(G^{\mathsf{sim}}_{M,K,y})$. If the obfuscation was "ideal" and equivalent to a black-box computing the underlying function, this would follow easily. Specifically, looking just at the output distributions of the real and simulated obfuscations, we know that:

$$\{K_i^{x_i}\}, \{G_{M,K}(i)\}_{i \in [O(t \cdot s)]} \approx_c \widehat{C}(x) \approx_c \widehat{C}^{\mathsf{sim}} \approx_c \left\{K_i^0\right\}, \left\{G^{\mathsf{sim}}_{M,K,y}(i)\right\}_{i \in [O(t \cdot s)]} \quad ;$$

indeed, the two inner distributions are indistinguishable just by the garbled circuit security guarantee, and are each identical to the corresponding outer distribution, except for the fact that the outer distributions use pseudorandom keys rather than truly random keys.

This intuition can be quite directly transformed into a proof relying on the virtual black-box definition of Barak et al. [BGI$^+$01], which says that whatever predicate can be learned from an obfuscation can also be learned by a simulator, given only black-box access to the circuit. However, the virtual black-box definition is rather strong, and in some settings impossible altogether [BGI$^+$01, GK05, BCC$^+$14]; moreover, as argued above it seems that we only need the obfuscation to satisfy some sort of indistinguishability requirement, rather than a strong simulation requirement. Instead, we attempt to base the scheme on the much weaker notion of indistinguishability obfuscation (IO), for which we have candidate constructions, under different assumptions on multilinear graded encodings [GGH$^+$13b, PST14, GLSW14].

The notion of IO only guarantees that for any two circuits C_0, C_1 of the same size and which compute the same function, their two obfuscation $i\mathcal{O}(C_0)$ and $i\mathcal{O}(C_1)$ are computationally indistinguishable. Is IO enough for recovering the above argument? For once, had we taken the circuit $G_{M,K}$ (and accordingly $G_{M,K,y}$) and had simply padded it to be large enough, then we could simply replace it with a circuit that has $O(t \cdot s)$ hardcoded values $G_{M,K}(i)$, each returned given the corresponding input i. Clearly, this results in the same functionality, and thus we can rely on IO and the same argument as above. However, this beats the purpose of achieving a succinct randomized encoding.

The question is whether this padding is necessary. We typically think of IO as the best-possible obfuscator [BGI$^+$01, GR07]; however, what we really mean is that $i\mathcal{O}(C)$ is as good as the best possible obfuscation $b\mathcal{O}(C)$, when the circuit C is padded to be of size $b\mathcal{O}(C)$. In contrast, virtual black-box obfuscation is truly best-possible, without having to pad, because when the circuit is regarded as an oracle, the size doesn't matter. One can naturally try to formulate a notion of best-possible obfuscation that does not require padding, but only satisfies an indistinguishability guarantee and is not as strong as virtual black-box. In Section 3.5, we explore such a notion which we call *padding-free indistinguishability obfuscation*.

Relying on (plain) IO, and its cost. Aiming to avoid additional (arguably strong) assumptions, we show that, by replacing arbitrary pseudo-random functions (PRFs) with what is known as *puncturable PRFs* [SW14], we can rely on plain IO, while padding proportionally to the space s of the computation rather than the time t. We now explain the rough ideas behind this, as well as where the space comes up.

In puncturable PRFs, it is possible to efficiently puncture a given key K at any polynomial set of points S in the domain of the function. The punctured function F_{K_S}, with punctured key K_S, preserves functionality at any other point, but hides any information on $\{F_K(x) : x \in S\}$; namely, these values are pseudo-random, even given (S, K_S). As shown in several recent works [BW13, BGI14, KPTZ13], such puncturable PRFs follow from the GGM construction [GGM84].

Recall, that we would like to prove that the real $\{K_i^{x_i}\}, \mathcal{O}(G_{M,K} \circ 0^{\ell})$ is computationally indistinguishable from the simulated $\{K_i^0\}, \mathcal{O}(G^{\mathsf{sim}}_{M,K,y} \circ 0^{\ell_{\mathsf{sim}}})$, where this time the two circuits are padded with an appropriate number of zeros, which is bounded by $s \cdot \mathrm{poly}(\lambda)$. To show this, we will closely follow the way that Garbled circuit security is usually proven (described in detail in [LP09]). Specifically, we will show how to move from the real randomized encoding to the simulated one by a sequence of $O(t \cdot s)$ hybrids, each time making only local changes to the underlying obfuscated program. In hybrid i, we shall obfuscate a hybrid circuit $G^i_{M,K} \circ 0^{\ell_i}$ that for all gates $j < i$ generates simulated garbled gates using G^{sim}, and for gates i and beyond creates real garbled gates using G. In the first n hybrids, we shall also gradually change the choice of keys from $\{K_i^{x_i}\}$ to $\{K_i^0\}$.

The standard garbled simulation argument shows that in step i we can change the real garbled gate $\widehat{g}_i$ to a simulated one $\widehat{g}_i^{\mathsf{sim}}$, provided that: (a) all the gates below i were already simulated, and (b) we know the real value corresponding to the wire going out from g_i. The first condition, can be guaranteed by looking at a topological order of the gates. For simplicity, we consider a $O(t)$-layered circuit, where each layer is of size $O(s)$ is connected only to the one below and above. The second condition is also not a problem in the standard garbled circuit setting; indeed, the process generating the current hybrid distribution may "remember" the intermediate wire values for the current layer.

Trying to carry a similar hybrid argument to our setting, requires a more careful reasoning. Indeed, the major different in our setting is that the adversary doesn't see only the resulting garbled circuit, but actually sees the circuit that computes it, which internally includes a the PRF key used to generate the randomness. Not surprisingly, if this PRF key is leaked in full, the scheme is completely broken. We show, however, that IO in conjunction with a puncturable PRF guarantee sufficient hiding. Specifically, the place where the above difference kicks in is exactly when replacing a real garbled gate with a simulated one. In the standard garbled setting both are generated using true (rather than pseudo) randomness, which is completely hidden from the adversary. This is exactly where we use the puncturing property: when dealing with gate g_i in the i^{th} hybrid, we will puncture the PRF in the constant number of corresponding wires, and instead directly hardcode the garbled gate $\widehat{g}_i$ (or $\widehat{g}_i^{\mathsf{sim}}$) into the circuit. This change is indistinguishable, because we did not change the functionality of the circuit. Furthermore, now we can invoke the guaranteed pseudorandomness in punctured points, and generate the garbled gate using true randomness, just as in the standard setting, allowing us to replace the real gate by a simulated gate. Importantly, before moving to the next hybrid we unpuncture the PRF, to guarantee that the key doesn't grow as the number of punctured points increases.[4]

The reason for blowing up the obfuscated circuit proportionally to the space is that we still need to remember the intermediate wires values for the currently simulated layer. Unlike the standard garbling setting where this wasn't an issue, as the adversary anyhow sees only the resulting garbled circuit, here it is since the adversary actually sees the (obfuscated) circuit that generates the garbled circuit. In our setting, the intermediate values for the current layer will be hardwired into the circuit, and each one of them can only be "forgotten" once all the connected gates in the layer above are simulated (inducing new values to remember). As long as we simulate layer by layer, we can guarantee that the overall blowup does not exceed $s \cdot \mathrm{poly}(\lambda)$.

IO for a simple class of circuits, and randomized encodings in NC^1. Finally, we note that the obfuscated circuits in our constructions are of a special kind. First, input size of the obfuscated circuits is $\log ts = O(\log t)$. Second, assuming puncturable PRFs in **NC**^{1} (constructed in [BLMR13, HKW14]), the obfuscated circuits can also be computed in **NC**^{1}. Another thing to note is that the constructed succinct randomized encodings can be made to be in **NC**^{1}. This is similar to the simple observation that randomized encodings can be composed [AIK06]; that is we can consider an outer later of an **NC**^{1} randomized encoding (like Yao), that computes the inner succinct randomized encoding.

1.2.1 Main Ideas behind the Applications

We now briefly sketch the main ideas behind our three main applications.

Succinct IO. We start by the construction of succinct IO from succinct randomized encodings. The basic scheme is a natural instantiation of the bootstrapping approach of Applebaum [App13]. There the goal is to reduce obfuscation of general circuits to obfuscation of **NC**^{1} circuits; our goal is to reduce obfuscation of uniform machines with large running time to obfuscation of significantly smaller circuits. To obfuscate a machine M with respect to inputs of size at most n, the idea is to obfuscate a small circuit $C_{M,K}$ that has a hardwired key for a PRF, and given input x, applies the PRF to x to derive randomness, and then computes a succinct randomized encoding of $U_x(M)$, where U_x is a universal machine that runs the input machine M on x. The obfuscated $\mathcal{O}(M)$, given input x uses the obfuscated circuit $\mathcal{O}(C_{M,K})$ to compute the encoding $\widehat{U}_x(M)$, and then decodes the result.

The analysis in [App13] would imply security in case that $\mathcal{O}$ has virtual black-box security, for arbitrary PRFs. We show that if $\mathcal{O}$ has $2^{\lambda^{\varepsilon}}$-security for security parameter $\lambda > n^{2/\varepsilon}$, and the PRF is a puncturable with a similar level of security, then a similar result holds for IO, rather than virtual black-box. The idea is simple, given two machines M_0, M_1 that compute the same function on all inputs $x \in \{0,1\}^{\leq n}$, we show how to move from an obfuscation of M_0 to an obfuscation of M_1 with a sequence of $O(2^n)$ hybrids. In the i^{th} hybrid we obfuscate a threshold circuit $C^i_{M_0,M_1,K}$ that for inputs $x < i$ computes the randomized encodings using M_0, and for larger inputs using M_1 (here we naturally think of the x as the corresponding

[4]In the body, and for a simpler technical exposition, we handle an entire layer consisting of $O(s)$ gates each time.

number according to its lexicographic place in $\{0,1\}^{\leq n}$). Each circuit $C^i_{M_0,M_1,K}$ is larger than the original $C_{M_0,K}$ but only on a factor of $O(n)$; to invoke IO, we will need to pad them accordingly.

It is easy to see that the first and last circuits $C^i_{M,K}$, compute the exact same function as $C^i_{M_0,K}$ and $C^i_{M_1,K}$, respectively. To show indistinguishability of two consecutive hybrids, we simply puncture at the current point i, hardwire the succinct randomized encodings, and rely on the fact that $\widehat{U}_i(M_0) \approx_c \widehat{U}_i(M_0)$, since $U_i(M_0) = M_0(i) = M_1(i) = U_i(M_1)$. Subexponential security of the circuit IO is required due to the subexponential number of hybrids. (We note that using a puncturable PRF in $\mathbf{NC}^1$ and the fact that the succinct randomized encoding can be computed in $\mathbf{NC}^1$, we can rely on IO for $\mathbf{NC}^1$.)

Succinct FE. We now move on to discuss the construction of succinct functional encryption, which follows rather directly from succinct randomized encodings and previous work. Roughly speaking, the idea is to start from an FE scheme for randomized circuits, rather than deterministic ones, and then in order to derive a key for a machine M, use the randomized circuit FE to derive a key for the circuit that given x, computes a randomized encoding of $M(x)$. One construction of such FE schemes was shown by [GJKS13] based on IO. Another construction was shown by [GGHZ14] only for circuits in $\mathbf{NC}^1$, under concrete assumptions on composite-order multilinear graded encodings; this is sufficient for us since the succinct randomized encoding can be computed in $\mathbf{NC}^1$. The latter construction, in fact, satisfies a strong notion of adaptively-secure FE (for randomized circuits).[5].

Publicly-verifiable delegation. Finally, we sketch the ideas behind our delegation scheme and SNARGs. The delegation scheme is pretty simple and similar in spirit to previous delegation schemes (in a weaker processing model) [AIK10, GGP10, GKP+13b]. To delegate a computation, given by M and x, the verifier simply sends the prover a randomized encoding $\widehat{M'}(x,r)$, where M' is a machine that returns r if and only if it accepts x, and r is a 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 pucturable PRF, and given a computation (M,x), applies the PRF to derive randomness and generates a first message for the delegation scheme. Thus for each new computation, a first message is effectively sampled afresh. Relying on IO and the security of the puncturable PRF, we can show that (non-adaptive) soundness is guaranteed. (Again, using puncturable PRFs in $\mathbf{NC}^1$, we can rely on IO for $\mathbf{NC}^1$.) 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.

Organization

In Section 2 we go over preliminiaries and primitives we will be using. In Section 3 we define succinct randomized encoding schemes (Section 3.1) and present our main construction (Section 3.3). We present our fully-succinct construction in Section 3.4. In Section 4 we present some applications of our randomized encoding schemes, particularly those of succinct indisitnguishability obfuscation (Section 4.1), succinct functional encryption (Section 4.2), publicly verifiable delegation and SNARGS for **P** (Section 4.3).

[5]Eventually, we rely on a restricted version of FE for randomized circuits, restricted to the setting of randomized encodings.

2 Preliminaries

We review basic concepts and definitions used throughout the paper.

2.1 Standard Computational Concepts

We rely on the standard notions of Turing machines and Boolean circuits. We say that a (uniform) Turing machine is PPT if is probabilistic and runs in polynomial time. A polynomial-size (or just polysize) circuit family $\mathcal{C}$ is a sequence of circuit $\mathcal{C} = \{C_\lambda\}_{\lambda \in \mathbb{N}}$, such that each circuit C_λ is of polynomial size $\lambda^{O(1)}$ and has $\lambda^{O(1)}$ inputs and outputs bits.

We follow the standard habit of modeling any efficient adversary strategy as a family of polynomial-sized circuits. For an adversary $\mathcal{A}$ corresponding to a family of polysize circuits $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, we often omit the subscript λ, when it is clear from the context.

2.2 Indistinguishability Obfuscation

We recall the notion of indistinguishability obfuscation (IO) recently realized in [GGH$^+$13b] using candidate multilinear maps[GGH13a].

Definition 2.1 (Indistinguishability Obfuscator (IO))**.** *A PPT machine $i\mathcal{O}$ is an* indistinguishability obfuscator *for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if the following conditions are satisfied:*

- **Functionality:** *for all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that*

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

- **Indistinguishability:** *for any polysize distinguisher $\mathcal{D}$, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$ of the same size, we have that if $C_0(x) = C_1(x)$ for all inputs x, then*

$$\Big| \Pr\big[\mathcal{D}(i\mathcal{O}(C_0)) = 1\big] - \Pr\big[\mathcal{D}(i\mathcal{O}(C_1)) = 1\big] \Big| \leq \alpha(\lambda) \ .$$

2.3 Puncturable Pseudo-Random Functions

Puncturable PRFs defined by Sahai and Waters [SW14], are PRFs for which a key can be given out that allows evaluation of the PRF on all inputs, except for a designated polynomial-size set of inputs.

Definition 2.2 (Puncturable PRFs)**.** *A puncturable pseudo-random function $\mathcal{F}$ is given by a triple of efficient algorithms (*$\mathsf{Key}_\mathcal{F}$,$\mathsf{Puncture}_\mathcal{F}$*, and* $\mathsf{Eval}_\mathcal{F}$*), and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:*

- ***Functionality preserved under puncturing****: For every polynomial size set $S \subseteq \{0,1\}^{n(\lambda)}$ and for every $x \in \{0,1\}^{n(\lambda)} \backslash S$, we have that:*

$$\Pr[\mathsf{Eval}_\mathcal{F}(K, x) = \mathsf{Eval}_\mathcal{F}(K_S, x) : K \leftarrow \mathsf{Key}_\mathcal{F}(1^\lambda), K_S = \mathsf{Puncture}_\mathcal{F}(K, S)] = 1$$

- ***Pseudorandom at punctured points**: For every polynomial size set $S \subseteq \{0,1\}^{n(\lambda)}$ we have that for every polysize adversary $\mathcal{A}$ we have that:*

$$|\Pr[\mathcal{A}(K_S, \mathsf{Eval}_{\mathcal{F}}(K,S)) = 1] - \Pr[\mathcal{A}(K_S, \mathsf{Eval}_{\mathcal{F}}(K, U_{m(\lambda)\cdot|S|})) = 1]| = \mathrm{negl}(\lambda)$$

where $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$ and $K_S = \mathsf{Puncture}_{\mathcal{F}}(K,S)$ and $\mathsf{Eval}_{\mathcal{F}}(K,S)$ denotes the concatenation of $\mathsf{Eval}_{\mathcal{F}}(K,x_1)), \dots, \mathsf{Eval}_{\mathcal{F}}(K,x_k)$ where $S = \{x_1, ..., x_k\}$ is the enumeration of the elements of S in lexicographic order, U_ℓ denotes the uniform distribution over ℓ bits.

The GGM tree-based construction of PRFs [GGM84] from one-way functions are easily seen to yield puncturable PRFs where the size of the punctured key grows polynomial in the size of the set S being punctured, as recently observed by [BW13, BGI14, KPTZ13]. Thus we have:

Lemma 2.1 ([GGM84, BW13, BGI14, KPTZ13])**.** *If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.*

3 Succinct Randomized Encodings

In this section, we define fully succinct and semi-succinct randomized encodings. We then present two constructions: the first obtains semi-succinct randomized encodings based on IO for **P**/poly; the second obtains fully succinct randomized encodings from a new notion called padding-free indistinguishability obfuscation.

3.1 Definitions

We now define the basic concepts of succinct and semi-succinct randomized encodings. The definitions naturally generalize the standard definition of randomized encodings for circuits [IK00, AIK06]; the essential difference is that the computations to be encoded are succinctly represented by a Turing machine or a RAM M and input x, and the goal is that the running time of the encoding procedure would be proportional to its succinct description, i.e. $|M| + |x|$ rather than depend on the running time of M. In a semi-succinct randomized encoding the encoding time is allowed to depend on the space s of M.

Definition 3.1 (Succinct Randomized Encoding)**.** *A succinct randomized encoding (SRE) scheme* sRE *consists of two algorithms* ($\mathsf{sRE.enc}, \mathsf{sRE.dec}$) *that work as follows:*

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

The scheme should satisfy the following three requirements:

1. **Functionality:** *for every input x and machine M such that M halts on x within t steps,it holds that $y = M(x)$ with overwhelming probability over the coins of* $\mathsf{sRE.enc}$.

2. **Security:** *there exists a PPT simulator* Sim *satisfying: for any poly-size distinguisher $\mathcal{D}$, there exists a negligible $\alpha(\cdot)$, such that for any $\lambda \in \mathbb{N}$, machine M, input x, and time bound t:*

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

where $\widehat{M} \leftarrow \mathsf{sRE.enc}(M, x, t, 1^{|x|}, 1^{\lambda})$, y is the output of $M(x)$ after t steps, and $p(\cdot)$ is a fixed polynomial that does not depend on (M, x, t). [6]

3. **Full-succinctness:** *The running time of* sRE.enc *and the size of the encoding $\widehat{M}_{x,t}$ are* $\mathrm{poly}(|M|, |x|, \log t, \lambda)$, *depending only polylogarithmically on the computation time t. The running time of* sRE.dec *is* $\mathrm{poly}(t, \lambda)$.

3'. **Semi-succinctness:** *The randomized encoding is semi-succinct if encoding complexity may also depend on space; concretely: the running time of* sRE.enc *and the size of the encoding $\widehat{M}_{x,t}$ are* $\mathrm{poly}(|M|, |x|, \log t, s, \lambda)$, *where s is the maximum space used by M in t steps. (The running time of* sRE.dec *is* $\mathrm{poly}(t, \lambda)$ *as above.)*

Remark 3.1 (Reusability). Reusable randomized encodings are such where encoding is split into two parts: the first part $\widehat{M}$ is independent of the specific input x and only depends on the machine M. This first part can then be "reused" together with a second part $\widehat{x}$ that includes an encoding of input x. Such reusable randomized encodings were previously studied in [GKP$^+$13b, GHRW14], in the context of reusable garbled circuits or RAMs. There, first encoding part is as large as the time if the computation, which can be much larger than the input; accordingly, the goal of reusability is to reuse and thus amortize the first expensive part, for many encoded inputs, where the each encoding takes time proportional only to the input size and not the time of the computation.

For fully succinct randomized encodings this is typically insignificant, because encoding is in any case independent of the time of the computation; however, for semi-succinctness this may be meaningful as now encoding depends on the space, which may be much larger than the input size. We can obtain also reusable semi-succinct randomized encodings, as a direct corollary of the succinct FE constructed in Section 4.2.

Remark 3.2 (Sub-exponential security). Currently the security definition of (semi) succinct randomized encodings addresses polynomial security. We can naturally strengthen the definition to require sub-exponential security, e.g. for constant $\varepsilon < 1$, we may require $2^{-\Omega(\lambda^{\varepsilon})} \cdot p(t)$-indistinguishability holds against $2^{O(\lambda^{\varepsilon})}$-size distinguishers.

Remark 3.3 (Relaxed security). In our applications, we in fact settle for schemes with a relaxed security, where the simulator Sim is allowed to be inefficient.

Remark 3.4 (Multi-bit output). The above description only allows for randomized encodings of machines that output a single bit. In some our applications, we will require a randomized encoding for machines with multiple output bits. This is easily achievable by providing a separate randomized encoding for each output bit. Accordingly, the running time of both sRE.enc and sRE.dec increases by a multiplicative factor of the output length.

[6] We shall mostly be interested in settings where t is also polynomially bounded by λ, in which case the above probability is simply negligible in λ.

3.2 Succinct Circuits and Circuit Garbling

We now describe two main tools that will be utilized by our construction. The first is the known reduction between Turing or random-access machines to succinctly represented circuits. The second is Yao's Garbled circuit method.

Succinct circuits. We consider two models of succinctly represented circuits. In the first, a possibly large circuit C is represented by a succinct circuit C_{sc} that can generate each of the gates in the circuit C along with pointer to their neighbours. In the second model, we focus on layered circuits C, where each layer is connected only to its predecessor and successor. Here C will be represented by a succinct circuit C_{sc} that outputs a layer (rather than a gate).

Definition 3.2 (Succinct Circuit). *Let $C : \{0,1\}^n \to \{0,1\}$ be a circuit with ℓ binary gates. The gates are topologically ordered and labeled by $1, \dots, \ell$ so that if an output of gate $i \in [\ell]$ is an input to gate $j \in [\ell]$ then $i < j$. We assume that the gates corresponding to $\{1, \dots, n\}$ are input gates with fan-in one, the gate corresponding to ℓ is the output gate with fan-out one, and $\{n+1, \dots, \ell-1\}$ are intermediate gates with fan-in and fan-out at most two.*

We say that C is succinctly represented by a circuit C_{sc}, if C_{sc} given a gate label $g \in [\ell]$ as input, outputs the labels of the neighbouring gates of gate g in C, and the corresponding binary operation $\odot$. Furthermore, $|C_{\mathsf{sc}}| < |C|$.

Definition 3.3 (Layered Succinct Circuit). *Let $C : \{0,1\}^n \to \{0,1\}$ be a circuit composed of d circuit layers $C_1, \dots C_d$, where the output of each layer $C_1, \dots, C_{d-1}$ consists of w output wires that are the input for the next, and C_d outputs the output bit.*

We say that C is succinctly represented by a layered circuit C_{sc}, if C_{sc} given a layer label $i \in [d]$, outputs C_i. Furthermore, $|C_{\mathsf{sc}}| < |C|$.

It is known that Turing and random-access machines can be reduced to either a succinct circuit or to a layered succinct circuit (paying in efficiency). Here we state a simple case of such a reduction.

Lemma 3.1 (From uniform machines to succinct circuits [PF79]). *Any (Turing or Ram) machine M, which for inputs of size n, requires maximal running time $t(n)$ and space $s(n)$,*

1. *can be reduced in time $O(|M| + \log t(n))$ to a circuit C^M_{sc} that succinctly represents $C^M : \{0,1\}^n \to \{0,1\}$, where C^M computes the same function as M (for inputs of length n), and is of size $\widetilde{O}(t(n)s)$.*
2. *can be reduced in time $O(|M| + \log t(n) + s(n))$ to a layered circuit C^M_{sc} that succinctly represents a layered $C^M : \{0,1\}^n \to \{0,1\}$, where C^M computes the same function as M (for inputs of length n), and is of size $O(t(n) \cdot s(n))$.*

Yao's garbled circuit. We rely on Yao's garbled circuit method [Yao82]. A circuit garbling scheme takes as input a circuit C and a pair of keys for every input wire and every output wire, and encodes the circuit into a garbled circuit $\widehat{C}$. Given the keys corresponding to some input x it is possible to decode, and obtain the keys corresponding to the output $y = C(x)$, without revealing any other information about the input x.

Definition 3.4 (Garbled Circuit). *A circuit garbling scheme consists of two algorithms* ($\mathsf{Garb.enc}$, $\mathsf{Garb.dec}$) *that work as follows:*

- $\widehat{C} \leftarrow \mathsf{Garb.enc}(C, \mathcal{K}_{\mathsf{i}}, \mathcal{K}_{\mathsf{o}} 1^\lambda)$: *takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^n$, a security parameter 1^λ, input keys $\mathcal{K}_{\mathsf{i}} = \left\{K^0_{\mathsf{i},j}, K^1_{\mathsf{i},j}\right\}_{j \in [n]}$, and output keys $\mathcal{K}_{\mathsf{o}} = \left\{K^0_{\mathsf{o},j}, K^1_{\mathsf{o},j}\right\}_{j \in [n]}$, where each key is of length λ. The algorithm outputs a garbled circuit $\widehat{C}$.*

- $\mathcal{K}^y_{\mathsf{out}} \leftarrow \mathsf{Garb.dec}(\widehat{C}, \mathcal{K}^x_{\mathsf{in}})$: *takes as input a garbled gate $\widehat{C}$, and input keys $\mathcal{K}^x_{\mathsf{in}} = \left\{K^{x_j}_{\mathsf{in},j}\right\}_{j\in[n]}$, corresponding to an input $x \in \{0,1\}^n$. The algorithm produces output keys $\mathcal{K}^y_{\mathsf{out}} = \left\{K^{y_j}_{\mathsf{out}}\right\}_{j\in[n]}$.*

The scheme should satisfy the following two requirements:

1. **Functionality:** *for every input $x \in \{0,1\}^n$, the decoding procedure* $\mathsf{Garb.dec}$ *produces the keys corresponding to the output, namely $y = C(x)$.*
2. **Security:** *there exists a PPT simulator* $\mathsf{Garb.Sim}$ *satisfying: for any poly-size distinguisher $\mathcal{D}$, there exists a negligible $\alpha(\cdot)$, such that for any $\lambda \in \mathbb{N}$, circuit C, inputs $x, x' \in \{0,1\}^n$, and keys $\mathcal{K}_{\mathsf{out}} = \left\{K^0_{\mathsf{out},j}, K^1_{\mathsf{out},j}\right\}_{j\in[n]}$:*

$$\left|\Pr[\mathcal{D}\left(\mathcal{K}^x_{\mathsf{in}}, \widehat{C}\right) = 1] - \Pr[\mathcal{D}\left(\mathcal{K}^{x'}_{\mathsf{in}}, \widehat{C}^{\mathsf{sim}}\right) = 1]\right| \leq \alpha(\lambda) \ ,$$

where $\mathcal{K}_{\mathsf{in}} = \left\{K^0_{\mathsf{in},j}, K^1_{\mathsf{in},j}\right\}_{j\in[n]} \leftarrow \{0,1\}^{\lambda\times 2n}$, $\mathcal{K}^x_{\mathsf{in}} = \left\{K^{x_j}_{\mathsf{in},j}\right\}_{j\in[n]}$, $\widehat{C} \leftarrow \mathsf{Garb.enc}(C, \mathcal{K}_{\mathsf{in}}, 1^\lambda)$, $\widehat{C}^{\mathsf{sim}} \leftarrow \mathsf{Garb.Sim}(C, \mathcal{K}_{\mathsf{in}}, \mathcal{K}^y_{\mathsf{out}}, 1^\lambda)$, $y = C(x)$, and $\mathcal{K}^y_{\mathsf{out}} = \left\{K^{y_j}_{\mathsf{out},j}\right\}_{j\in[n]}$.

Circuit garbling schemes are known to exist assuming one-way functions.

Theorem 3.1 (Garbled circuits from OWFs [Yao82, LP09])**.** *Assuming the existence of one-way functions, there exist circuit garbling schemes.*

3.3 Semi-Succinct Randomized Encodings from Indistinguishability Obfuscation

We now describe a semi-succinct randomized encoding scheme based on IO. An overview is given in Section 1.2

Notation: We introduce notation that will be used in the description of the construction and its proof of security. We shall consider a layered circuit C^M (corresponding to some machine M) with d layers and at most w input wires per layer. We label every wire in C^M with a pair (i,j) that identifies it as the j^{th} input wire to the i^{th} layer of C^M. The output wire is labelled as $(d+1,1)$, and the input wires by $\{(1,j)\}_{j\in n}$; to be more explicit, we shall often denote them instead by $(\mathsf{out},1)$ and $\{(\mathsf{in},j)\}_{j\in n}$. In the construction below, there will be a pair of keys $(K^0_{i,j}, K^1_{i,j})$ associated with every wire (i,j). Consistently with the notation in Definition 3.4, we shall denote by $\mathcal{K}_i$ all the keys $\left\{K^0_{i,j}, K^1_{i,j}\right\}_{j\in[w]}$ corresponding to the input wires to layer i, where w is the number of such wires. Also, for a possible assignment $x_i \in \{0,1\}^w$ to these values, we denote by $\mathcal{K}^{x_i}_i$ the keys $\left\{K^{x_{i,j}}_{i,j}\right\}_{j\in[w]}$ corresponding to the bits of x_i.

3.3.1 The Scheme

Let $i\mathcal{O}$ be an indistinguishability obfuscator for $\mathbf{P}/\mathrm{poly}$ and $(\mathsf{Garb.enc}, \mathsf{Garb.dec})$ be a circuit garbling scheme. Let $(\mathsf{Key}_{\mathcal{F}}, \mathsf{Puncture}_{\mathcal{F}}, \mathsf{Eval}_{\mathcal{F}})$ be a puncturable PRF with input length λ and output length 2λ. We describe the encoding and decoding algorithms.

The encoder $\mathsf{sRE.enc}(M, x, t, 1^\lambda)$**:**

1. sRE.enc first reduces the machine M, input length $|x|$ and time bound t to a layered circuit C^M_{sc} which is a succinct representation of circuit C^M as per Lemma 3.1. Let d be the number of layers in C^M and w be the maximum number of input wires in any layer.
2. Next, sRE.enc samples a puncturable PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$, and derives input keys $\mathcal{K}_{\mathsf{in}}$ and output keys $\mathcal{K}_{\mathsf{out}}$, by running $\mathsf{Eval}_{\mathcal{F}}(K, (\mathsf{in}, j))$ for every wire (in, j), and $\mathsf{Eval}_{\mathcal{F}}(K, (\mathsf{out}, 1))$.[1]
3. Let $\mathsf{pad}_\ell(L[C^M_{\mathsf{sc}}, K])$ be the layer garbling circuit from Figure 1 padded to size $\ell \leq |C^M_{\mathsf{sc}}| \cdot \mathrm{poly}(\lambda)$, which will be specified exactly in the analysis. sRE.enc outputs $(i\mathcal{O}(\mathsf{pad}_\ell(L[C^M_{\mathsf{sc}}, K])), \mathcal{K}^x_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}})$, where $\mathcal{K}^x_{\mathsf{in}}$ are the keys corresponding to the input layer and concrete input x.

Layer garbling circuit $L[C^M_{\mathsf{sc}}, K]$

Hardwired: a succinct representation C^M_{sc} of the circuit C^M which has d layers and at most w input wires per layer, and a puncturable PRF key $K \in \{0,1\}^\lambda$.

Input: a label $i \in [d]$ for a layer in the circuit C^M.

1. Compute $C_i \leftarrow C^M_{\mathsf{sc}}(i)$, the i^{th} layer of C^M.
2. Generate wire keys $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ for the input and output wires for C_i, by running $(K^0_{i,j}, K^1_{i,j}) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, (i, j))$ and $(K^0_{i+1,j}, K^1_{i+1,j}) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, (i+1, j))$, for every $j \in [w]$.[1]
3. Output a garbled C_i:
$$\widehat{C}_i \leftarrow \mathsf{Garb.enc}(C_i, \mathcal{K}_i, \mathcal{K}_{i+1}, 1^\lambda) \ .$$

Figure 1: Circuit $L[C^M_{\mathsf{sc}}, K]$

The decoder $\mathsf{sRE.dec}(\widehat{M}_{x,t})$:

1. Parse $\widehat{M}_{x,t}$ as a circuit $\tilde{L}$ and wire keys $\mathcal{K}^x_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}}$.
2. For each layer $i = 1 \ldots d$,
   - Compute the garbled i^{th} layer $\widehat{C}_i = \tilde{L}(i)$.
   - Using the input keys $\mathcal{K}^{x_i}_i$ for the i^{th} layer, generate input keys $\mathcal{K}^{x_{i+1}}_{i+1}$ for the $i+1^{st}$ layer. That is, compute $\mathcal{K}^{x_{i+1}}_{i+1} = \mathsf{Garb.dec}(\widehat{C}_i, \mathcal{K}^{x_i}_i)$, where x_{i+1} is a string of length at most w, representing the values of the wires at layer i (the first input x_1, also denoted as x_{in}, is x).
3. Obtain a key for the output wire $K^b_{\mathsf{out},1}$, and according to $\mathcal{K}_{\mathsf{out}} = (K^0_{\mathsf{out},1}, K^1_{\mathsf{out},1})$, output the bit b.

We show

Theorem 3.2. *The randomized encoding scheme described above is a (semi-succinct) secure randomized encoding.*

[1]Throughout, we assume WLOG that $w \cdot d \leq 2^\lambda$, and thus wires can be represented as inputs to the puncturable PRF in $\{0,1\}^\lambda$.

Remark 3.5 (On the IO assumption). We note here that in the our construction we rely on indistinguishability obfuscation for circuits that take short inputs *i.e.* the circuit we obfuscate takes inputs of length $\log t$ when encoding with time bound t. Indistinguishability obfuscation for such circuits is a falsifiable assumption, and using the result of [GLSW14], can be based on *polynomial* hardness of multilinear group based assumptions. We also note that assuming puncturable PRFs in $\mathbf{NC}^1$ (in particular, $\mathsf{Eval}_{\mathcal{F}}$ is in $\mathbf{NC}^1$), the class of circuits we require IO for is in $\mathbf{NC}^1$.

Remark 3.6 (Encoding in $\mathbf{NC}^1$). We can modify our randomized encoding scheme so that the encoder $\mathsf{sRE.enc}$ is computed by a $\mathbf{NC}^1$ circuit. We simply compose the above construction with a randomized encoding scheme for circuits with encoding in $\mathbf{NC}^1$ (e.g. Yao's circuit garbling). That is, the encoder for the composed scheme computes a randomized encoding for the circuit $\mathsf{sRE.enc}$ (as described in our construction) on input (M, x, t). This will be useful for some of our applications.

Correctness and Succinctness: Correctness follows directly from the correctness of the circuit garbling scheme. Furthermore, the running time of $\mathsf{sRE.enc}$ and the size of the encoding $\widehat{M}_{x,t}$ are $\mathrm{poly}(|C^M_{\mathsf{sc}}|, \lambda)$ which by Lemma 3.1 is bounded by $\mathrm{poly}(|M|, |x|, \log t, s, \lambda)$, where s is the space used by M; hence, the encoding scheme is semi-succinct.

3.3.2 Proof of Security

In this section, we prove Theorem 3.2.

Remark 3.7. We note that the following proof also works for achieving sub-exponential security for our randomized encoding scheme, assuming sub-exponential security for the indistinguishability obfuscator, puncturable PRF and garbled circuits.

Proof. Our goal is to construct a PPT simulator Sim satisfying, for any poly-size distinguisher $\mathcal{D}$, machine M, input x, and time bound t:

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

for some negligible $\alpha(\cdot)$, where $\widehat{M}_{x,t} \leftarrow \mathsf{sRE.enc}(M, x, t, 1^{|x|}, 1^\lambda)$, y is the output of $M(x)$ after t steps, and $p(\cdot)$ is a fixed polynomial that does not depend on (M, x, t).

Recall that $\widehat{M}_{x,t} = (\tilde{L}, \mathcal{K}^x_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}})$ contains an obfuscation $\tilde{L}$ of a layer garbling circuit, input wire keys $\mathcal{K}^x_{\mathsf{in}}$, and output wire keys $\mathcal{K}_{\mathsf{out}}$. At high-level, Sim simulates $\widehat{M}_{x,t}$ by producing an obfuscation of a circuit that applies the garbled circuit simulator to simulate fake garbled layers, and fake input keys.

Simulator $\mathsf{Sim}(y, M, t, 1^{|x|}, 1^\lambda)$ **(differs from** $\mathsf{sRE.enc}$ **in the third of three steps):**

1. Sim first computes the succinct representation C^M_{sc} of the layered circuit C^M.

2. Next, Sim samples a puncturable PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$, and derives input keys $\mathcal{K}_{\mathsf{in}}$ and output keys $\mathcal{K}_{\mathsf{out}}$, by running $\mathsf{Eval}_{\mathcal{F}}(K, (\mathsf{in}, j))$ for every wire (in, j), and $\mathsf{Eval}_{\mathcal{F}}(K, (\mathsf{out}, 1))$.

3. Let $\mathsf{pad}_\ell(L^{\mathsf{sim}}[y, C^M_{\mathsf{sc}}, K])$ be the layer simulating circuit from Figure 2 padded to size $\ell \leq |C^M_{\mathsf{sc}}| \cdot \mathrm{poly}(\lambda)$, which will be specified later on. Sim outputs $(i\mathcal{O}(\mathsf{pad}_\ell(L^{\mathsf{sim}}[y, C^M_{\mathsf{sc}}, K])), \mathcal{K}^{0^n}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}})$, where $\mathcal{K}^{0^n}_{\mathsf{in}}$ are the keys corresponding to the input layer and input 0^n.

Layer simulating circuit $L^{\mathsf{sim}}[y, C^M_{\mathsf{sc}}, K]$

Hardwired: a succinct representation C^M_{sc} of the circuit C^M which has d layers and at most w input wires per layer, a puncturable PRF key $K \in \{0,1\}^\lambda$, and an output bit $y \in \{0,1\}$.

Input: a label $i \in [d]$ for a layer in the circuit C^M.

1. Compute $C_i \leftarrow C^M_{\mathsf{sc}}(i)$, the i^{th} layer of C^M.
2. Generate wire keys $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ for the input and output wires for C_i, by running $\mathsf{Eval}_{\mathcal{F}}(K, (i, j))$ and $\mathsf{Eval}_{\mathcal{F}}(K, (i+1, j))$, for every $j \in [w]$.
3. Simulate a garbled C_i as follows:
   - for $i < d$, $\widehat{C}^{\mathsf{sim}}_i \leftarrow \mathsf{Garb.Sim}(C_i, \mathcal{K}_i, \mathcal{K}^{0^w}_{i+1}, 1^\lambda)$,
   - for $i = d$, $\widehat{C}^{\mathsf{sim}}_i \leftarrow \mathsf{Garb.Sim}(C_i, \mathcal{K}_d, \mathcal{K}^{y}_{\mathsf{out}}, 1^\lambda)$.
4. Output $\widehat{C}^{\mathsf{sim}}_i$.

Figure 2: Circuit $L^{\mathsf{sim}}[y, C^M_{\mathsf{sc}}, K]$

We show that the (real) randomized encoding is indistinguishable from the simulated one by a hybrid argument. Informally, we will consider d hybrid distributions, where in the k^{th} distribution we shall obfuscate a hybrid layer garbling circuit that simulates all the garbled layers up to layer k, and truly garbles the layers above layer k.

Formally, for $k \in [d+1]$ and $x_k \in \{0,1\}^{\leq w}$, the hybrid simulated distribution $\mathsf{Hyb}(k, x_k, C^M_{\mathsf{sc}}, 1^\lambda)$ is defined as follows.

Hybrid simulator $\mathsf{Hyb}(k, x_k, C^M_{\mathsf{sc}}, 1^\lambda)$ **(differs from** $\mathsf{sRE.enc}$ **and** Sim **in the third of three steps):**

1. Hyb first computes the succinct representation C^M_{sc} of the layered circuit C^M.
2. Next, Hyb samples a puncturable PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$, and derives input keys $\mathcal{K}_{\mathsf{in}}$ and output keys $\mathcal{K}_{\mathsf{out}}$, by running $\mathsf{Eval}_{\mathcal{F}}(K, (\mathsf{in}, j))$ for every wire (in, j), and $\mathsf{Eval}_{\mathcal{F}}(K, (\mathsf{out}, 1))$.
3. Let $\mathsf{pad}_\ell(L^{\mathsf{Hyb}}[k, x_k, C^M_{\mathsf{sc}}, K])$ be the hybrid layer garbling circuit from Figure 3 padded to size $\ell \leq |C^M_{\mathsf{sc}}| \cdot \mathrm{poly}(\lambda)$, which will be specified later on. Hyb outputs:
   - for $k > 1$: $(i\mathcal{O}(\mathsf{pad}_\ell(L^{\mathsf{Hyb}}[k, x_k, C^M_{\mathsf{sc}}, K])), \mathcal{K}^{0^n}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}})$
   - for $k = 1$: $(i\mathcal{O}(\mathsf{pad}_\ell(L^{\mathsf{Hyb}}[1, x_1, C^M_{\mathsf{sc}}, K])), \boxed{\mathcal{K}^{x_1}_{\mathsf{in}}}, \mathcal{K}_{\mathsf{out}})$.

First, note that for any machine M, input x and time bound t, $\widehat{M_{x,t}}$ has the same distribution as the one generated by $\mathsf{Hyb}(1, x, C^M_{\mathsf{sc}}, 1^\lambda)$, and $\mathsf{Sim}(y, M, t, 1^{|x|}, 1^\lambda)$ has the same distribution as the one generated by $\mathsf{Hyb}(d+1, y, C^M_{\mathsf{sc}}, 1^\lambda)$ where y is the output of M on x in t steps.

From hereon, for some fixed (C^M, x) we denote by $x_k \in \{0,1\}^{\leq w}$ the bit string corresponding to the values of the input wires to the k^{th} layer of C^M when computed with input x. That is, $x_1 = x$ and $x_{k+1} = C_k(x_k)$, where $C_k = C^M_{\mathsf{sc}}(k)$. In particular, $x_{d+1} = y = C^M(x)$.

Hybrid layer garbling circuit $L^{\mathsf{Hyb}}[k, x_k, C^M_{\mathsf{sc}}, K]$

Hardwired: a succinct representation C^M_{sc} of the circuit C^M which has d layers and at most w input wires per layer, a puncturable PRF key $K \in \{0,1\}^\lambda$, and $x_k \in \{0,1\}^{\leq w}$, representing the values of the wires going into layer k.

Input: a label $i \in [d]$ for a layer in the circuit C^M.

1. Compute $C_i \leftarrow C^M_{\mathsf{sc}}(i)$, the i^{th} layer of C^M.
2. Generate wire keys $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ for the input and output wires for C_i, by running $\mathsf{Eval}_{\mathcal{F}}(K,(i,j))$ and $\mathsf{Eval}_{\mathcal{F}}(K,(i+1,j))$, for every $j \in [w]$.
3. Simulate a garbled C_i as follows:
   - for $i < k-1$, $\widehat{C}^{\mathsf{sim}}_i \leftarrow \mathsf{Garb.Sim}(C_i, \mathcal{K}_i, \mathcal{K}^{0^w}_{i+1}, 1^\lambda)$,
   - for $i = k-1$, $\widehat{C}^{\mathsf{sim}}_i \leftarrow \mathsf{Garb.Sim}(C_{k-1}, \mathcal{K}_{k-1}, \mathcal{K}^{x_k}_{k}, 1^\lambda)$,
   - for $i \geq k$, $\widehat{C}_i \leftarrow \mathsf{Garb.enc}(C_i, \mathcal{K}_i, \mathcal{K}_{i+1}, 1^\lambda)$.
4. Output the resulting garbled result.

Figure 3: Circuit $L^{\mathsf{Hyb}}[k, x_k, C^M_{\mathsf{sc}}, K]$

Since the number of hybrids is $d+1$, which is in turn bounded by $O(t)$ the running time of $M(x)$, it suffices to prove the following lemma, in order to complete the proof of the theorem.

Lemma 3.2. *For any poly-size distinguisher $\mathcal{D}$, there exists a negligible $\alpha(\cdot)$, such that for any $\lambda \in \mathbb{N}$, machine M, input x, and time bound t, the following is satisfied: let C^M_{sc} be the layered circuit that succinctly represents C^M corresponding to (M, x, t) steps. Let d be the number of layers in C^M. Then, for every $k \in [d]$,*

$$\left|\Pr[\mathcal{D}(\mathsf{Hyb}(k, x_k, C^M_{\mathsf{sc}}, 1^\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}(k+1, x_{k+1}, C^M_{\mathsf{sc}}, 1^\lambda)) = 1]\right| \leq \alpha(\lambda) \ .$$

Proof. The lemma is proved via yet another hybrid argument. Concretely, we consider the following hybrid distributions.

- $\mathcal{H}_0$: This distribution is identical to $\mathsf{Hyb}(k, x_k, C^M_{\mathsf{sc}}, 1^\lambda)$.
- $\mathcal{H}_1$: This distribution is obtained changing $\mathsf{Hyb}(k, x_k, C^M_{\mathsf{sc}}, 1^\lambda)$ as follows. Instead of obfuscating the circuit $\mathsf{pad}_\ell(L^{\mathsf{Hyb}}[k, x_k, C^M_{\mathsf{sc}}, K])$, Hyb obfuscates a circuit $\mathsf{pad}_\ell(L^{\mathsf{Hyb}}_1[k, x_k, C^M_{\mathsf{sc}}, K_{k\times[w]}, \mathcal{K}^{x_k}_k, \widehat{C}_k])$, that differs from the latter in the following ways:
  1. L^{Hyb}_1 uses a PRF key $K_{k\times[w]}$ that is punctured at the set of inputs $\{(k,j)\}_{j\in[w]}$ corresponding to the input wires to the k^{th} layer of C^M; that is, $K_{k\times[w]} \leftarrow \mathsf{Puncture}_{\mathcal{F}}(K, \{k\} \times \{1, \dots, w\})$.
  2. for inputs $i \notin \{k-1, k\}$, L^{Hyb}_1 proceeds as L^{Hyb}, only that it uses the punctured key $K_{k\times[w]}$ instead of K.

3. for $i = k-1$, L_1^{Hyb} does not explicitly derive $\mathcal{K}_k$ using K, but rather has only the keys $\mathcal{K}_k^{x_k}$ hardwired to it, and uses them as before to compute $\widehat{C}_{k-1}^{\mathsf{sim}}$.
4. for $i = k$, L_1^{Hyb} does not explicitly compute $\widehat{C}_k$ using the keys $\mathcal{K}_k$ derived from K, as in L^{Hyb}, but rather L_1^{Hyb} has $\widehat{C}_k$ hardwired, and outputs it given input k.

- $\mathcal{H}_2$: This distribution is identical to $\mathcal{H}_1$ except that the keys $\mathcal{K}_k$ are not derived pseudorandomly from K, but rather are sampled truly at random $\mathcal{K}_k \leftarrow U_{2\lambda\times w}$. Accordingly, the hardwired values $\mathcal{K}_k^{x_k}$ and $\widehat{C}_k$ are now with respect to this (random) choice of $\mathcal{K}_k$.
- $\mathcal{H}_3$: This distribution is identical to $\mathcal{H}_1$ with the following two changes:
  1. The hardwired $\widehat{C}_k$ in L_1^{Hyb} is now replaced with a simulated one:
  $$\widehat{C}_k^{\mathsf{sim}} \leftarrow \mathsf{Garb.Sim}(C_k, \mathcal{K}_k, \mathcal{K}_{k+1}^{x_{k+1}}), 1^\lambda) \ ,$$
  where $C_k \leftarrow C_{\mathsf{sc}}^M(k)$, $\mathcal{K}_k$ is sampled uniformly as in the previous hybrid, and $\mathcal{K}_{k+1}^{x_{k+1}}$ is obtained by computing $\{\mathsf{Eval}_{\mathcal{F}}(K_{k\times[w]}, (k+1, j)) : j \in [w]\}$ and $x_{k+1} = C_k(x_k)$.
  2. Instead of having $\mathcal{K}_k^{x_k}$ hardwired in it, L_1^{Hyb} has $\mathcal{K}_k^{0^w}$ hardwired.
- $\mathcal{H}_4$: This distribution is identical to $\mathsf{Hyb}(k+1, x_{k+1}, C_{\mathsf{sc}}^M, 1^\lambda)$.

Let ℓ be the maximal of $L, L^{\mathsf{sim}}, L^{\mathsf{Hyb}}, L_1^{\mathsf{Hyb}}$, for any possible $k \in [d+1]$, and note that
$$\ell \leq (|C_k| + |x_k|) \cdot \mathrm{poly}(\lambda) \leq 2|C_k| \cdot \mathrm{poly}(\lambda) \ .$$

To prove the lemma, we show that the above hybrids are computationally indistinguishable; that is, for any poly-size distinguisher $\mathcal{D}$, there exists a negligible $\alpha(\cdot)$, such that for any $\lambda \in \mathbb{N}$, $i \in [4]$,
$$|\Pr[\mathcal{D}(\mathcal{H}_{i-1}) = 1] - \Pr[\mathcal{D}(\mathcal{H}_i) = 1]| \leq \alpha(\lambda)$$

.

- $\mathcal{H}_0$ and $\mathcal{H}_1$: follows from the IO guarantee. Specifically, the only difference between the two is in the the circuit they obfuscate $\mathsf{pad}_\ell(L^{\mathsf{Hyb}})$, or $\mathsf{pad}_\ell(L_1^{\mathsf{Hyb}})$; however, by the definition of L_1^{Hyb} these two circuits compute exactly the same function, and we can invoke the IO guarantee.
- $\mathcal{H}_1$ and $\mathcal{H}_2$: follows from pseudorandomness at punctured points. Specifically, any distinguisher between the two hybrids directly translates to a distinguisher between
$$K_{k\times[w]}, (\mathsf{Eval}_{\mathcal{F}}(K, (k, j)) : j \in [w]) \text{ and } K_{k\times[w]}, U_{2\lambda\times w} \ .$$
Indeed, the two distributions $\mathcal{H}_1$,$\mathcal{H}_2$ can be perfectly simulated from $K_{k\times[w]}$ and $\mathcal{K}_k$, which results in $\mathcal{H}_1$ if $\mathcal{K}_k$ are chosen according to K, and in $\mathcal{H}_2$, if $\mathcal{K}_k$ are chosen uniformly at random.
- $\mathcal{H}_2$ and $\mathcal{H}_3$: follows from the security of the circuit garbling scheme. Indeed, we are guaranteed that
$$\mathcal{K}_k^{x_k}, \widehat{C}_k \approx_c \mathcal{K}_k^{0^w}, \widehat{C}_k^{\mathsf{sim}} \ ,$$
for $\mathcal{K}_k \leftarrow U_{2\lambda\times w}$, $\widehat{C}_k \leftarrow \mathsf{Garb.enc}(C_k, \mathcal{K}_k, \mathcal{K}_{k+1}, 1^\lambda)$, and $\widehat{C}_k^{\mathsf{sim}} \leftarrow \mathsf{Garb.Sim}(C_k, \mathcal{K}_k, \mathcal{K}_{k+1}^{x_{k+1}}, 1^\lambda)$, assuming $x_{k+1} = C_k(x_k)$, and any choice of $\mathcal{K}_{k+1}$, in particular $\mathcal{K}_{k+1} = \mathsf{Eval}_{\mathcal{F}}(K_{k\times w}, (k+1, j))_{j\in[w]}$. Moreover, given $K_{k\times w}$, it is possible to simulate the rest of $\mathcal{H}_2$ or $\mathcal{H}_3$.

- $\mathcal{H}_3$ and $\mathcal{H}_4$: this follows as in the indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_2$, but in reverse order. That is, we first invoke pseudorandomness at punctured points, to go from sampler $\mathcal{K}_k$ at random, to sampling it according to K, and then "unpuncture" K at $k \times [w]$, and invoke IO.

This completes the proof of the lemma. □

This completes the proof of the theorem. □

3.4 Fully-Succinct Randomized Encodings from Padding-Free IO

In this section, we introduce *padding-free indistinguishability obfuscation* (padding-free IO). Then we construct (fully) succinct randomized encodings from padding-free IO.

In what follows, we shall say that a PPT algorithm $\mathcal{O}$ is function-preserving if for any circuit C, $\mathcal{O}(C)$ computes the same function as $\mathcal{O}$. (In particular, any obfuscator $\mathcal{O}$ is function-preserving).

Definition 3.5 (padding-free IO)**.** *Let $\mathcal{O}$ be a function-preserving PPT algorithm. We say that $\mathcal{O}$ is a padding-free indistinguishability obfuscation with respect to two distribution ensembles $(\mathcal{C}_0, \mathcal{C}_1) = \{(\mathcal{C}_{\lambda,0}, \mathcal{C}_{\lambda,1})\}$, if for any function-preserving PPT $\mathcal{O}'$, if $\mathcal{O}'(\mathcal{C}_0) \approx_c \mathcal{O}'(\mathcal{C}_1)$, then $\mathcal{O}(\mathcal{C}_0) \approx_c \mathcal{O}(\mathcal{C}_1)$.*

Before showing how to construct full-succinct randomized encodings from padding-free IO, a couple of remarks are in place. First, we note that crucially, the $padding - freeindistinguishabilityobfuscation$ obfuscator $\mathcal{O}$ has some fixed polynmoial blowup, whereas the obfuscator $\mathcal{O}'$ may blowup the circuit to an arbitrary poly-size. Second, we note that while this definition is implied by VBB, and may seem weaker, it is also seems unlikely to hold for general distributions.[7] Nevertheless, the notion may be satisfied for certain distributions, such as the one we are interested in here.

3.4.1 The Scheme

Let $\mathcal{O}$ be a padding-free indistinguishability obfuscation for $\mathbf{P}/\mathrm{poly}$ and $(\mathsf{Garb.enc}, \mathsf{Garb.dec})$ be a circuit garbling scheme. Let $(\mathsf{Key}_{\mathcal{F}}, \mathsf{Eval}_{\mathcal{F}})$ be a PRF with input length λ and output length 2λ. We describe the encoding and decoding algorithms.

Notation: We introduce notation that will be used in the description of the construction and its proof of security. As before, we shall consider a circuit C^M corresponding to some machine M. Let s be the size of the circuit and w be the number of wires in the circuit. We label every gate with $i \in [s]$ and every wire with $j \in [w]$. Let $\mathsf{in}(i)$ and $\mathsf{out}(i)$ denote the set of input wires and output wires to gate i respectively. In the construction below, there will be a pair of keys (K_j^0, K_j^1) associated with every wire $j \in [w]$. Adding to the notation in Definition 3.4, we shall denote by $\mathcal{K}_{\mathsf{in},i}$ all the keys $\left\{K_j^0, K_j^1\right\}_{j \in \mathsf{in}(i)}$ corresponding to the input wires to gate g. Similarly, $\mathcal{K}_{\mathsf{out},i}$ denotes the keys $\left\{K_j^0, K_j^1\right\}_{j \in \mathsf{out}(i)}$ corresponding to the output

[7]More accurately, we mean that for any candidate $padding - freeindistinguishabilityobfuscation$ $\mathcal{O}$, there exists two distributions $\mathcal{C}_0, \mathcal{C}_1$, and $\mathcal{O}'$ such that $\mathcal{O}(\mathcal{C}_0) \approx_c \mathcal{O}(\mathcal{C}_1)$, but $\mathcal{O}(\mathcal{C}_0) \not\approx_c \mathcal{O}(\mathcal{C}_1)$. To get a sense of why this is unlikely, consider $\mathcal{C}_b$ which compute a pseudorandom function only on T values, where T is a large enough polynomial related to the blowup of $\mathcal{O}$. In addition they output a succinct obfuscation of a program that given a small circuit, checks whether it computes the PRF correctly on the T values and if so, outputs b. $\mathcal{O}'$ is simply an obfuscator that doesn't have the key for the PRF, but rather has the PRF values hardwired. Using a standard incompressibility argument $\mathcal{O}'(\mathcal{C}_0) \approx_c \mathcal{O}'(\mathcal{C}_1)$; indeed, had we replaced the pseudo-random values with truly random values, a small circuit computing it wouldn't exist. On the other hand, $\mathcal{O}(\mathcal{C}_b)$ will already be itself such a small circuit, and thus will allow learning b. (For this to work, the succinct obfuscation needs to also be independent of the input size and not just the running time.)

wires to gate g. $\mathcal{K}_{\mathsf{in}}$ and $\mathcal{K}_{\mathsf{out}}$ denote the set of wire keys for the input and output wires of C^M respectively. Also, for a possible assignment $x \in \{0,1\}^{|\mathsf{in}(i)|}$ to these wires, we denote by $\mathcal{K}^x_{\mathsf{in},i}$ the keys $\left\{K_j^{x_j}\right\}_{j\in\mathsf{in}(i)}$ corresponding to the bits of x. $\mathcal{K}^x_{\mathsf{out},i}$, $\mathcal{K}^x_{\mathsf{in}}$ and $\mathcal{K}^x_{\mathsf{out}}$ are defined similarly.

The encoder $\mathsf{sRE.enc}(M, x, t, 1^\lambda)$**:**

1. $\mathsf{sRE.enc}$ first reduces the machine M, input length $|x|$ and time bound t to a circuit C^M_{sc} which is a succinct representation of circuit C^M as per Lemma 3.1.
2. Next, $\mathsf{sRE.enc}$ samples a puncturable PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$.
3. Let $\mathsf{pad}_\ell(G[x, C^M_{\mathsf{sc}}, K])$ be the gate garbling circuit from Figure 4 padded to size $\ell \leq \mathrm{poly}(|C^M_{\mathsf{sc}}|, |x|, \lambda)$, which will be specified exactly in the analysis. $\mathsf{sRE.enc}$ outputs $\mathcal{O}(\mathsf{pad}_\ell(G[x, C^M_{\mathsf{sc}}, K]))$.

Gate garbling circuit $G[x, C^M_{\mathsf{sc}}, K]$

Hardwired: a succinct representation C^M_{sc} of the circuit C^M of size s, a PRF key $K \in \{0,1\}^\lambda$ and input x to C^M

Input: either of the following labels: (i) $\mathsf{inputkeys}$ for the keys of the input wires corresponding to input x, (ii) $\mathsf{outputkeys}$ for the keys of the output wire, or (iii) $i \in [s]$ for a gate in the circuit C^M.

Case (i): input is $\mathsf{inputkeys}$
Generate and output wire keys $\mathcal{K}^x_{\mathsf{in}}$ for the input wires of C^M corresponding to input x, by running $(K^0_j, K^1_j) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, j)$ and picking $K_j^{x_j}$ for every input wire j of C^M.

Case (ii): input is $\mathsf{outputkeys}$
Generate and output wire keys $\mathcal{K}_{\mathsf{out}}$ for the output wire of C^M, by running $(K^0_j, K^1_j) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, j)$ for the output wire j of C^M.

Case (iii): input is $i \in [s]$

1. Compute $g_i \leftarrow C^M_{\mathsf{sc}}(i)$, the i^{th} gate of C^M.
2. Generate wire keys $\mathcal{K}_{\mathsf{in},i}$ and $\mathcal{K}_{\mathsf{out},i}$ for the input and output wires for g_i, by running $(K^0_j, K^1_j) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, j)$ and for every $j \in \mathsf{in}(i)$ and $\mathsf{out}(i)$.[1]
3. Output a garbled g_i:
$$\tilde{g}_i \leftarrow \mathsf{Garb.enc}(C_i, \mathcal{K}_{\mathsf{in},i}, \mathcal{K}_{\mathsf{out},i}, 1^\lambda) \ .$$

Figure 4: Circuit $G[x, C^M_{\mathsf{sc}}, K]$

The decoder $\mathsf{sRE.dec}(\widehat{M}_{x,t})$**:**

1. Generate wire keys $\mathcal{K}^x_{\mathsf{in}} \leftarrow \widehat{M}_{x,t}(\mathsf{inputkeys})$ and $\mathcal{K}_{\mathsf{out}} \leftarrow \widehat{M}_{x,t}(\mathsf{outputkeys})$.
2. For every $i \in [s]$, compute the garbled gate $\tilde{g}_i \leftarrow \widehat{M}_{x,t}(i)$. Let $\tilde{G} = \{\tilde{g}_i\}_{i\in[s]}$.

[1]Throughout, we assume WLOG that $w \leq 2^\lambda$, and thus wires can be represented as inputs to the puncturable PRF in $\{0,1\}^\lambda$.

3. Compute $\mathcal{K}^b_{\mathsf{out}} \leftarrow \mathsf{Garb.dec}(\tilde{G}, \mathcal{K}^x_{\mathsf{in}})$ and according to $\mathcal{K}_{\mathsf{out}}$ which contains the keys of both $\mathcal{K}^0_{\mathsf{out}}$ and $\mathcal{K}^1_{\mathsf{out}}$, output the bit b.

Correctness and Succinctness: Correctness follows directly from the correctness of the circuit garbling scheme. Furthermore, the running time of $\mathsf{sRE.enc}$ and the size of the encoding $\widehat{M}_{x,t}$ are $\mathrm{poly}(|C^M_{\mathsf{sc}}|, |x|, \lambda)$ which by Lemma 3.1 is bounded by $\mathrm{poly}(|M|, |x|, \log t, \lambda)$; hence, the encoding scheme is succinct.

3.4.2 Proof of Security

In this section, we prove

Theorem 3.3. *The randomized encoding scheme described above is a (succinct) secure randomized encoding.*

Proof. Our goal is to construct a PPT simulator Sim satisfying, for any poly-size distinguisher $\mathcal{D}$, machine M, input x, and time bound t:

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

for some negligible $\alpha(\cdot)$, where $\widehat{M}_{x,t} \leftarrow \mathsf{sRE.enc}(M, x, t, 1^{|x|}, 1^\lambda)$, y is the output of $M(x)$ after t steps, and $p(\cdot)$ is a fixed polynomial that does not depend on (M, x, t).

Recall that $\widehat{M}_{x,t} = (\tilde{G}, \mathcal{K}^x_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}})$ contains an obfuscation $\tilde{G}$ of a gate garbling circuit, input wire keys $\mathcal{K}^x_{\mathsf{in}}$, and output wire keys $\mathcal{K}_{\mathsf{out}}$. At high-level, Sim simulates $\widehat{M}_{x,t}$ by producing an obfuscation of a circuit that applies the garbled circuit simulator to simulate fake garbled gates, and a fake input keys.

Simulator $\mathsf{Sim}(y, M, t, 1^{|x|}, 1^\lambda)$ **(differs from** $\mathsf{sRE.enc}$ **in the third of three steps):**

1. Sim first reduces the machine M, input length $|x|$ and time bound t to a circuit C^M_{sc} which is a succinct representation of circuit C^M as per Lemma 3.1.

2. Next, Sim samples a PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$.

3. Let $\mathsf{pad}_\ell(G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K])$ be the gate simulating circuit from Figure 5 padded to size ℓ which is the maximum size of $G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K]$ and $G[x, C^M_{\mathsf{sc}}, K]$. Note that $\ell \leq \mathrm{poly}(|C^M_{\mathsf{sc}}|, |x|, \lambda)$. Sim outputs $\mathcal{O}(\mathsf{pad}_\ell(G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K]))$.

Therefore, to complete the proof it suffices to show that for any poly-size distinguisher $\mathcal{D}$, machine M, input x and time bound t, $\mathcal{D}$ can't distinguish between the distribution ensembles

$$\{\mathcal{O}(\mathsf{pad}_\ell(G[x, C^M_{\mathsf{sc}}, K])) : K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda\}_{\lambda \in \mathbb{N}}$$

and

$$\{\mathcal{O}(\mathsf{pad}_\ell(G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K])) : K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)\}_{\lambda \in \mathbb{N}}$$

.

Subsequently, we will assume that G and G^{sim} are padded to size ℓ and drop the pad_ℓ notation. Since $\mathcal{O}$ is a padding-free indistinguishability obfuscation, it suffices to show that there exists a function-preserving PPT $\mathcal{O}'$ such that $\{\mathcal{O}'(G[x, C^M_{\mathsf{sc}}, K])\}$ and $\{\mathcal{O}'(G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K])\}$ are indistinguishable. We describe $\mathcal{O}'$ as follows

Function-preserving PPT $\mathcal{O}'$:

Gate simulating circuit $G^{\mathsf{sim}}[|x|, y, C_{\mathsf{sc}}^M, K]$

Hardwired: a succinct representation C_{sc}^M of the circuit C^M which has size s, a PRF key $K \in \{0,1\}^\lambda$, input length $|x|$ and an output bit $y \in \{0,1\}$.

Input: either of the following labels: (i) $\mathsf{inputkeys}$ for the keys of the input wires, (ii) $\mathsf{outputkeys}$ for the keys of the output wire, or (iii) $i \in [s]$ for a gate in the circuit C^M.

Case (i): input is $\mathsf{inputkeys}$
Generate and output wire keys $\mathcal{K}_{\mathsf{in}}^{0^{|x|}}$ for the input wires of C^M corresponding to input $0^{|x|}$, by running $(K_j^0, K_j^1) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, j)$ and picking K_j^0 for every input wire j of C^M.

Case (ii): input is $\mathsf{outputkeys}$
Generate and output wire keys $\mathcal{K}_{\mathsf{out}}$ for the output wire of C^M, by running $(K_j^0, K_j^1) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, j)$ for output wire j of C^M.

Case (iii): input is $i \in [s]$

1. Compute $g_i \leftarrow C_{\mathsf{sc}}^M(i)$, the i^{th} gate of C^M.
2. Generate wire keys $\mathcal{K}_{\mathsf{in},i}$ and $\mathcal{K}_{\mathsf{out},i}$ for the input and output wires for g_i, by running $(K_j^0, K_j^1) \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, j)$ and for every $j \in \mathsf{in}(i)$ and $\mathsf{out}(i)$
3. Simulate a garbled g_i as follows:
   - if g_i is the output gate (*i.e.* the output wire of g_i is the output wire of C^M) then $\tilde{g}_i \leftarrow \mathsf{Garb.Sim}(g_i, \mathcal{K}_{\mathsf{in},i}, \mathcal{K}_{\mathsf{out},i}^y, 1^\lambda)$.
   - Otherwise, $\tilde{g}_i \leftarrow \mathsf{Garb.Sim}(g_i, \mathcal{K}_{\mathsf{in},i}, \mathcal{K}_{\mathsf{out},i}^{0^{|\mathsf{out}(i)|}}, 1^\lambda)$,
4. Output $\tilde{g}_i$.

Figure 5: Circuit $G^{\mathsf{sim}}[|x|, y, C_{\mathsf{sc}}^M, K]$

1. $\mathcal{O}'$ gets as input a circuit C and generates the truth table for C for inputs $\mathsf{inputkeys}$, $\mathsf{outputkeys}$ and $i \in [s]$ where s is the size of C^M. That is, it generates the output of C on each of these inputs and stores the outputs in a table T.
2. $\mathcal{O}'$ outputs a circuit $C'[T]$ that has the table T hardwired in. $C'[T]$ on an input simply looks up the corresponding entry in T and outputs it.

We observe that the table hardwired in the circuit $C' \leftarrow \mathcal{O}'(G[x, C_{\mathsf{sc}}^M, K])$ consists of

- the input wire keys $\mathcal{K}_{\mathsf{in}}^x$ corresponding to input x and the output wire keys $\mathcal{K}_{\mathsf{out}}$ where the wire keys for any wire j come from $\mathsf{Eval}_{\mathcal{F}}(K, j)$
- the garbled circuit $\tilde{C} \leftarrow \mathsf{Garb.enc}(C^M, \mathcal{K}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}}, 1^\lambda)$ where the randomness used by $\mathsf{Garb.enc}$ also comes from $\mathsf{Eval}_{\mathcal{F}}(K, \cdot)$.

Relying on the pseudorandomness of $\mathsf{Eval}_{\mathcal{F}}$, we can replace the output of $\mathsf{Eval}_{\mathcal{F}}(K, \cdot)$ by uniformly random values. That is, the following distributions are indistinguishable.

$$\{\mathcal{O}'(G[x, C^M_{\mathsf{sc}}, K]) : K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda\}_{\lambda\in\mathbb{N}} \tag{1}$$

$$\approx_c \{C'[(\mathcal{K}^x_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}}, \mathsf{Garb.enc}(C^M, \mathcal{K}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}}, 1^\lambda))] : \mathcal{K}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}} \leftarrow \{0,1\}^{2\lambda(|x|+1)}\}_{\lambda\in\mathbb{N}} \tag{2}$$

where in the second distribution ensemble, the randomness used by $\mathsf{Garb.enc}$ is uniformly random.

Similarly, the table hardwired in the circuit $C' \leftarrow \mathcal{O}'(G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K])$ consists of

- the input wire keys $\mathcal{K}^x_{\mathsf{in}}$ corresponding to input $0^{|x|}$ and the output wire keys $\mathcal{K}_{\mathsf{out}}$ where the wire keys for any wire j come from $\mathsf{Eval}_{\mathcal{F}}(K, w)$
- the simulated circuit $\tilde{C} \leftarrow \mathsf{Garb.Sim}(C^M, \mathcal{K}_{\mathsf{in}}, \mathcal{K}^y_{\mathsf{out}}, 1^\lambda)$ where $y = C^M(x)$ and the randomness used by $\mathsf{Garb.Sim}$ also comes from $\mathsf{Eval}_{\mathcal{F}}(K, \cdot)$.

Relying on the pseudorandomness of $\mathsf{Eval}_{\mathcal{F}}$, we have that the following distribution ensembles are indistinguishable.

$$\{\mathcal{O}'(G^{\mathsf{sim}}[|x|, y, C^M_{\mathsf{sc}}, K])) : K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)\}_{\lambda\in\mathbb{N}} \tag{3}$$

$$\approx_c \{C'[(\mathcal{K}^{0^{|x|}}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}}, \mathsf{Garb.Sim}(C^M, \mathcal{K}_{\mathsf{in}}, \mathcal{K}^y_{\mathsf{out}}, 1^\lambda))] : \mathcal{K}_{\mathsf{in}}, \mathcal{K}_{\mathsf{out}} \leftarrow \{0,1\}^{2\lambda(|x|+1)}\}_{\lambda\in\mathbb{N}} \tag{4}$$

where in the second distribution ensemble, the randomness used by $\mathsf{Garb.Sim}$ is uniformly random. By the security of the circuit garbling scheme we have that the distribution ensembles (2) and (4) are indistinguishable. Therefore, by a hybrid argument, it follows that the distribution ensembles (1) and (3) are indistinguishable. This completes the proof of the theorem. □

4 Applications

In this section we will provide applications of succinct randomized encodings.

Remark 4.1. Recall that a succinct randomized encoding consists of an obfuscated program. Looking ahead, in our application we will be considering obfuscations of circuits that output succinct randomized encodings. In other words if were to open up the abstraction we're actually double obfuscating, i.e. one obfuscated program generates another obfuscated program. We note that this is done just for the sake of clean exposition and modular analysis; we could obfuscate everything together at the level of the application itself.

4.1 Succinct Indistinguishability Obfuscation

In this section we will provide our construction for succinct obfuscating. We will start by presenting our definition of what it means to obfuscate a machine succinctly and then provide our construction for it.

Definition 4.1. *[Succinct Indistinguishability Obfuscator] A* succinct indistinguishability obfuscator *for a machine class $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of a uniform PPT machine $i\mathcal{O}\mathsf{M}$ that works as follows:*

- *$i\mathcal{O}\mathsf{M}$ takes as input the security parameter 1^λ, the machine M to obfuscate, and an input length n and time bound $\overline{T}$ for M.*
- *$i\mathcal{O}\mathsf{M}$ outputs a machine $\mathsf{ob}M$ which is an obfuscation of M corresponding to input length n and time bound $\overline{T}$. $\mathsf{ob}M$ takes as input $x \in \{0,1\}^n$ and $t \leq \overline{T}$.*

The scheme should satisfy the following three requirements.

- ***Correctness**: For all security parameters $\lambda \in \mathbb{N}$, for all $M \in \mathcal{M}_\lambda$, for all inputs $x \in \{0,1\}^n$, time bounds $\overline{T}$ and $t \leq \overline{T}$, let y be the output of M on t steps, then we have that:*

$$\Pr[\mathsf{obM}(x,t) = y : \mathsf{obM} \leftarrow i\mathcal{O}\mathsf{M}(1^\lambda, 1^n, 1^{\log \overline{T}}, M)] = 1$$

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

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

- ***Efficiency and Succinctness**: We require that the running time of $i\mathcal{O}\mathsf{M}$ and the length of its output, namely the obfuscated machine obM, is $\mathrm{poly}(|M|, \log \overline{T}, n, \lambda)$. We also require that the obfuscated machine on input x and t runs in time $\mathrm{poly}(|M|, t, n, \log \overline{T}, \lambda)$ (or $\mathrm{poly}(t, \lambda)$ for short).*
  ***Semi-Succinctness**: The requirement for semi-succinctness is very similar to succinctness except that all the parameters are now additionally allowed to grow in s, the maximum space used by M in t steps.*

Remark 4.2 (Succinctness vs Semi-Succinctness). We will only present our construction for the succinct case and note that it naturally extends to the semi-succinct setting.

Remark 4.3 (Input Specific Running Time). In the definition above the obfuscated program takes as the parameter t as an additional input, which specifies the time the machine M is to be executed. This allows us to achieve input specific running times for our construction whereby on an input x the obfuscated program runs in time proportional to the running time of M on input x. Specifically, this can be achieved by first executing the obfuscated program with input time parameter 2 and then doubling it each time, till the output is obtained.

Remark 4.4 (Multi-bit Output). The obfuscation definition and construction that we present in this section only consider machines that output single bit outputs. This result can easily be extended to multi-bit output setting by providing a separate obfuscation for each output bit. This increases the running time of both $i\mathcal{O}\mathsf{M}$ and actual evaluation of the obfuscated machines by a multiplicative factor of the output length.

Remark 4.5 (Different size machines). The security requirement above requires that the two machines M_0 and M_1 to be of the same size. We note that this is not a strict requirement. The security definition can be equipped to handle machines of different sizes but at the cost of additional padding.

4.1.1 Our Construction

Our succinct obfuscator $i\mathcal{O}\mathsf{M}$ is very simple and we describe it formally in Figure 6. Our construction uses a succinct randomized encoding scheme, an Indistinguishability Obfuscator for $P/poly$ and a Puncturable Pseudo-Random function.

Let $(\mathsf{sRE.enc}, \mathsf{sRE.dec})$ be a succinct randomized encoding scheme (satisfying Definition 3.1), $i\mathcal{O}$ be an Indistinguishability Obfuscator for $P/poly$ (satisfying Definition 2.1) and a $(\mathsf{Key}_{\mathcal{F}}, \mathsf{Puncture}_{\mathcal{F}}, \mathsf{Eval}_{\mathcal{F}})$ be a Puncturable Pseudo-Random function (satisfying Definition 2.2).

- **Obfuscation:** $i\mathcal{O}\mathsf{M}(1^{\lambda}, 1^{n}, 1^{\log \overline{T}}, M)$: The obfuscation algorithm $i\mathcal{O}\mathsf{M}$ on input a security parameter λ, an input size n, a time bound $\log \overline{T}$ and a machine $M \in \mathcal{M}$, samples $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^{\lambda+n+\log \overline{T}})$ and outputs $\mathsf{ob}M = i\mathcal{O}(1^{\lambda+n+\log \overline{T}}, \mathsf{pad}_{\ell}(\mathsf{C}[K, M, \lambda + n + \log \overline{T}]))$ where $\mathsf{pad}_{\ell}(\mathsf{C}[K, M, \lambda + n + \log \overline{T}])$ is the circuit $\mathsf{C}[K, M, \lambda + n + \log \overline{T}]$ described below, padded to size ℓ (where ℓ is the maximum size of $\{\mathsf{C}, \mathsf{C}^1, \mathsf{C}^2, \mathsf{C}^3\}$ and $\mathsf{C}^1, \mathsf{C}^2, \mathsf{C}^3$ are described in the proof):

  1. $\mathsf{C}[K, M, \lambda + n + \log \overline{T}]$ on input x, t[a] first generates $r_{x,t} = \mathsf{Eval}_{\mathcal{F}}(K, x||t)$, where $||$ denotes the concatenation operation.
  2. Next output $\mathsf{sRE.enc}(UM, (M, x), t, 1^{\lambda+n+\log \overline{T}})$ using $r_{x,t}$ as the random coins. Here UM is a Universal Machine[b] that takes as input a Machine M and value x. UM when executed for time t outputs the output of M on input x when executed for time t.

- **Evaluation:** Given an obfuscated machine $\mathsf{ob}M$, an evaluator can evaluates $\mathsf{ob}M$ on input (x, t) obtaining $\widehat{M}_{x,t}$. It then outputs $\mathsf{sRE.dec}(\widehat{M}_{x,t})$.

[a]We implicitly assume that the program does not take as input a time vale greater than $\overline{T}$.

[b]Note that depending on whether we are interested in Turing Machines or a Random Access Machine we can use a Universal Turing Machine or a Universal RAM Machine.

Figure 6: Succinct Indistinguishability Obfuscator.

4.1.2 Correctness and Succinctness

We start by describing the correctness of the scheme from Figure 6. Let M be any machine that is obfuscated. Note that on an input (x, t), by correctness of $i\mathcal{O}$, $\mathsf{ob}M$ outputs a randomized encoding $\widehat{M}_{x,t}$. Next by the correctness of succinct randomized encoding scheme we have that $\mathsf{sRE.dec}(\widehat{M}_{x,t})$ evaluates to the output of M on input x for time t.

Note that the obfuscation procedure from from Figure 6 basically outputs an obfuscation of circuit of size ℓ and succinctness follows by bounding this value. Recall that ℓ is the maximum size of circuits C, C^1, C^2 and C^3 as described in the following security proof. The size of these circuits depend on the running time of the $\mathsf{sRE.enc}$ procedure on inputs of length n with time bound $\log \overline{T}$. Therefore, by the succinctness property of $\mathsf{sRE.enc}$, we have that $\ell \leq \mathrm{poly}(|M|, \log \overline{T}, n, \lambda)$ which gives us succinctness for $i\mathcal{O}\mathsf{M}$.

4.1.3 Proof of Security

Theorem 4.1. *Assuming that a sub-exponentially secure succinct randomized encoding scheme, a sub-exponentially secure Indistinguishability Obfuscator for $P/poly$ and a sub-exponentially secure Puncturable Pseudo-Random function exist, then the obfuscator $i\mathcal{O}\mathsf{M}$ in Figure 6 satisfies the security Definition 4.1.*

Proof. We need to prove that no (possibly non-uniform) PPT distinguisher can distinguish between the two distributions generated by $i\mathcal{O}\mathsf{M}(1^\lambda, 1^n, 1^{\log \overline{T}}, M_0)$ and $i\mathcal{O}\mathsf{M}(1^\lambda, 1^n, 1^{\log \overline{T}}, M_1)$, where M_0 and M_1 are of the same size; and for every $x \in \{0,1\}^n$ and $t \leq \overline{T}$ we have that, the outputs M_0 and M_1 on input x when executed for time t are the same. We prove this by a sequence of $2^{n+\log \overline{T}}$ hybrids. Assuming exponentially hardness[8] for all the underlying computational primitives, we can claim that the distinguishing advantage among any two consecutive hybrids will be exponentially small and hence cumulative distinguishing probability can still be bounded by negligible.

- $\mathcal{H}_{M_0}$: In this hybrid, we execute $i\mathcal{O}\mathsf{M}$ on input M_0 as described in Figure 6. Namely sample K, the key for puncturable PRF, and output the obfuscation of $\mathsf{C}[K, M_0, \lambda + n + \log \overline{T}]$ (described in Figure 6) padded to size ℓ.

- $\mathcal{H}_i$: For $i \in \{0, \ldots, 2^{n+\log \overline{T}}\}$, in hybrid $\mathcal{H}_i$ we generate the obfuscation as follows. Sample K, the key for puncturable PRF, and output the obfuscation of $\mathsf{C}^1[K, M_0, M_1, \lambda + n + \log \overline{T}, i]$ described below, padded to size ℓ.

  1. $\mathsf{C}^1[K, M_0, M_1, \lambda + n + \log \overline{T}, i]$ on input x, t first generates $r_{x,t} = \mathsf{Eval}_\mathcal{F}(K, x||t)$.
  2. Next if $x||t > i$ then output $\mathsf{sRE.enc}(UM, (M_0, x), t, 1^{\lambda+n+\log \overline{T}})$ and $\mathsf{sRE.enc}(UM, (M_1, x), t, 1^{\lambda+n+\log \overline{T}})$ otherwise, using $r_{x,t}$ as the random coins in either case.

  Note that $\mathcal{H}_{M_0}$ and $\mathcal{H}_0$ obfuscate functional equivalent circuits and therefore are computationally indistinguishable based on Indistinguishability Obfuscation property of $i\mathcal{O}$.

- $\mathcal{H}_{i,1}$: Roughly $\mathcal{H}_{i,1}$ is same as $\mathcal{H}_i$ except that in $\mathcal{H}_{i,1}$ we puncture the key K at point $i+1$ and hardcode the output on input $i+1$. More formally, sample K, the key for puncturable PRF, and obtain $K' = \mathsf{Puncture}_\mathcal{F}(K, \{i+1\})$ and output the obfuscation of $\mathsf{C}^2[K', M_0, M_1, \lambda + n + \log \overline{T}, \mathsf{out}_{i+1}, i]$ described below, padded to size ℓ. Here $\mathsf{out}_{i+1} := \mathsf{sRE.enc}(UM, (M_0, x^*), t^*, 1^{\lambda+n+\log \overline{T}})$ using random coins $\mathsf{Eval}_\mathcal{F}(K, i+1)$, where $x^*||t^* = i+1$.

  1. If $x||t = i+1$ then output out_{i+1}. Otherwise proceed to the next step.
  2. $\mathsf{C}^2[K', M_0, M_1, \lambda + n + \log \overline{T}, i]$ on input x, t first generates $r_{x,t} = \mathsf{Eval}_\mathcal{F}(K, x||t)$.
  3. Next if $x||t > i$ then output $\mathsf{sRE.enc}(UM, (M_0, x), t, 1^{\lambda+n+\log \overline{T}})$ and $\mathsf{sRE.enc}(UM, (M_1, x), t, 1^{\lambda+n+\log \overline{T}})$ otherwise, using $r_{x,t}$ as the random coins in either case.

  Note that the only change in $\mathcal{H}_{i,1}$ from $\mathcal{H}_i$ is that the output obfuscation on input $i+1$ outputs the hardcoded value out_{i+1}. The same value was computed inside $\mathcal{H}_i$. In other words, $\mathcal{H}_{i,1}$ and $\mathcal{H}_i$ obfuscate functional equivalent circuits and therefore are computationally indistinguishable based on Indistinguishability Obfuscation property of $i\mathcal{O}$.

- $\mathcal{H}_{i,2}$: Just as in previous hybrid, output obfuscation of $\mathsf{C}^2[K', M_0, M_1, \lambda + n + \log \overline{T}, \mathsf{out}_{i+1}, i]$ padded to size ℓ. However use fresh randomness to generate out_{i+1} instead of using the randomness generated using seed K. More formally generate $\mathsf{out}_{i+1} := \mathsf{sRE.enc}(UM, (M_0, x^*), t^*, 1^{\lambda+n+\log \overline{T}})$ using fresh random coins, where $x^*||t^* = i+1$.

[8]We give the construction assuming exponential hardness on the underlying computational assumptions. The same result can be obtained assuming only sub-exponential hardness by setting the security parameter of the underlying primitives appropriately larger.

Note that the indistinguishability of $\mathcal{H}_{i,2}$ from $\mathcal{H}_{i,1}$ follows from the *pseudorandom at punctured points* property described in Definition 2.2.

- $\mathcal{H}_{i,3}$: Just as in previous hybrid, output obfuscation of $\mathsf{C}^2[K', M_0, M_1, \lambda + n + \log \overline{T}, \mathsf{out}_{i+1}, i]$ padded to size ℓ. However use machine M_1 to generate out_{i+1} instead of M_0. More formally generate $\mathsf{out}_{i+1} := \mathsf{sRE.enc}(UM, (M_1, x^*), t^*, 1^{\lambda+n+\log \overline{T}})$ using fresh random coins, where $x^*||t^* = i + 1$.

  The indistinguishability between $\mathcal{H}_{i,3}$ from $\mathcal{H}_{i,2}$ follows from the security property of succinct randomized encoding in Definition 3.1.

- $\mathcal{H}_{i,4}$: Just as in previous hybrid, output obfuscation of $\mathsf{C}^2[K', M_0, M_1, \lambda + n + \log \overline{T}, \mathsf{out}_{i+1}, i]$ padded to size ℓ. However we go back to using randomness generated using seed K in order to generate out_{i+1}, instead of using fresh randomness. More formally generate $\mathsf{out}_{i+1} := \mathsf{sRE.enc}(UM, (M_1, x), t, 1^{\lambda+n+\log \overline{T}})$ using random coins $\mathsf{Eval}_{\mathcal{F}}(K, i + 1)$, where $x^*||t^* = i + 1$.

  Note that the indistinguishability of $\mathcal{H}_{i,4}$ from $\mathcal{H}_{i,3}$ follows from the *pseudorandom at punctured points* property described in Definition 2.2.

- $\mathcal{H}_{i,5}$: $\mathcal{H}_{i,5}$ is same as $\mathcal{H}_{i,4}$ except that we do not hardcode out_{i+1} and use the circuit to compute it. More formally sample K, the key for puncturable PRF, and output the obfuscation of $\mathsf{C}^3[K, M_0, M_1, \lambda + n + \log \overline{T}, i]$ described below, padded to size ℓ.

  1. $\mathsf{C}^3[K, M_0, M_1, \lambda + n + \log \overline{T}, i]$ on input x, t first generates $r_{x,t} = \mathsf{Eval}_{\mathcal{F}}(K, x||t)$.
  2. Next if $x||t > i+1$ then output $\mathsf{sRE.enc}(UM, (M_0, x), t, 1^{\lambda+n+\log \overline{T}})$ and $\mathsf{sRE.enc}(UM, (M_1, x), t, 1^{\lambda+n+\log \overline{T}})$ otherwise, using $r_{x,t}$ as the random coins in either case.

  Note that the only change in $\mathcal{H}_{i,5}$ from $\mathcal{H}_{i,4}$ is that the output obfuscation on input $i + 1$ does not output the hardcoded value out_{i+1}. The same value is computed inside $\mathcal{H}_{i,5}$. In other words, $\mathcal{H}_{i,4}$ and $\mathcal{H}_{i,5}$ obfuscate functional equivalent circuits and therefore are computationally indistinguishable based on the Indistinguishability Obfuscation property of $i\mathcal{O}$.

  Note that the circuits $\mathsf{C}^3[K, M_0, M_1, \lambda + n + \log \overline{T}, i]$ and $\mathsf{C}^2[K, M_0, M_1, \lambda + n + \log \overline{T}, i + 1]$ are functionally equivalent and the computational indistinguishability between $\mathcal{H}_{i,5}$ and $\mathcal{H}_{i+1}$ follows based on the Indistinguishability Obfuscation property of $i\mathcal{O}$.

- $\mathcal{H}_{M_1}$: In this hybrid, we execute $i\mathcal{O}\mathsf{M}$ on input M_1 as described in Figure 6.

  Note that $\mathcal{H}_{2^{n+\log \overline{T}}}$ and $\mathcal{H}_{M_1}$ obfuscate functionally equivalent circuits and therefore are computationally indistinguishable based on Indistinguishability Obfuscation property of $i\mathcal{O}$.

□

4.2 Succinct Functional Encryption

In this section we will provide our construction of succinct functional encryption. Our construction is based on randomized functional encryption for a particular class of NC^1 circuits. Goyal et al. [GJKS13] provide a selectively secure construction of such a randomized functional encryption scheme for all $\mathbf{NC}^1$ circuits based on indistinguishability obfuscation. Using their scheme, we get a selectively secure, succinct functional encryption scheme. In recent results Garg et al. [GGHZ14] have shown that a fully secure variant of their construction can be based on polynomial hardness of relatively natural computational assumptions, thus giving us a succinct functional encryption scheme with full security.

We will start by presenting our definition of succinct functional encryption and the intermediate special functional encryption that we need. We then provide our construction for succinct functional encryption.

4.2.1 Definitions

We use (taken verbatim) the definition of functional encryption for Turing Machines in Ananth et al. [ABG$^+$13], also defined in [BCP14], and adapt it to the setting of general succinct machines.

Definition 4.2. *[Succinct Functional Encryption] A functional encryption scheme defined for a family of machines $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of four algorithms $\{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}\}$:*

- $\mathsf{Setup}(1^\lambda, 1^n)$ - *a polynomial time algorithm that takes as input the security parameter λ, message size n and outputs a public parameters* PP *and a master secret key* MSK. *(We assume that* PP *and* MSK *implicitly contain 1^λ and 1^n.)*
- $\mathsf{KeyGen}(\mathsf{MSK}, 1^{\log \overline{T}}, M)$ - *a polynomial time algorithm that takes as input the master secret key* MSK*, a time bound $\overline{T}$ and a machine $M \in \mathcal{M}$ and outputs a corresponding private key* SK_M.
- $\mathsf{Encrypt}(\mathsf{PP}, x)$ - *a polynomial time algorithm that takes the public parameters* PP *and a message $x \in \{0,1\}^n$ as input and outputs a ciphertext* CT.
- $\mathsf{Decrypt}(\mathsf{SK}_M, \mathsf{CT}, 1^t)$ - *a polynomial time algorithm that takes a private key* SK_M*, a ciphertext* CT *encrypting message $x \in \{0,1\}^n$ and a time $t \leq \overline{T}$ as input and outputs the output of M on input x after running it for time t.*

We require the following properties:-

- **Correctness**: A functional encryption scheme for family $\{\mathcal{M}_\lambda\}_\lambda$ is said to be *correct* if for all $\lambda, n \in \mathbb{N}$, machines $M \in \mathcal{M}$, messages $x \in \{0,1\}^n$, time bounds $\overline{T}$ and running times $t \leq \overline{T}$:

  $$\Pr[(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \mathsf{Decrypt}(\mathsf{KeyGen}(\mathsf{MSK}, 1^{\log \overline{T}}, M), \mathsf{Encrypt}(\mathsf{PP}, x), 1^t) = y)] = 1,$$

  where y is the output of M when executed for time t on input x.

- **Efficiency and Succinctness**: We require that the running time of Setup and $\mathsf{Encrypt}$ is $\mathrm{poly}(\lambda, n)$. We also require that the running time of KeyGen and its output is $\mathrm{poly}(|M|, \log \overline{T}, n, \lambda)$. Finally we require that $\mathsf{Decrypt}$ runs in time $\mathrm{poly}(|M|, t, n, \log \overline{T}, \lambda)$ (or $\mathrm{poly}(t, \lambda)$ for short).
  Semi-Succinctness: The requirement for semi-succinctness is very similar to succinctness except that all the parameters are now additionally allowed to grow in s, the maximum space used by M in t steps.

- **Security**: We now define the *full security* definition for succinct functional encryption which is described in form of an indistinguishability game between an adversary $\mathcal{A}$ and a challenger, parameterized by a polynomial $\overline{T}(\cdot)$.

  - **Setup**: The challenger runs $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and gives PP to $\mathcal{A}$.
  - **Query I**: $\mathcal{A}$ submits queries $M_i \in \mathcal{M}$ for $i \in 1, \ldots, q_1$ and to each one of them the challenger responds with $\mathsf{SK}_{M_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, 1^{\log \overline{T}(\lambda)}, M_i)$.
  - **Challenge**: The adversary generates two messages (x_0, x_1) and provides them to the challenger. We require that for every query M_i ($i \in [q_1]$) we have that for all $t \leq \overline{T}(\lambda)$ it is the case that $M_i(x_0)$ and $M_i(x_1)$ output the same value when executed for time t.

- **Query II**: $\mathcal{A}$ submits queries $M_i \in \mathcal{M}$ for $i \in q_1, \ldots, q$ and to each one of them the challenger responds with $\mathsf{SK}_{M_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, 1^{\log \overline{T}(\lambda)}, M_i)$. We require that for every query M_i ($i \in q_1, \ldots, q$) we have that for all $t \leq \overline{T}(\lambda)$ it is the case that $M_i(x_0)$ and $M_i(x_1)$ output the same value when executed for time t.
- **Guess**: $\mathcal{A}$ outputs a bit b' in $\{0, 1\}$.

The advantage of an adversary $\mathcal{A}$ is defined to be $|\Pr[b' = b] - \frac{1}{2}|$.

Definition 4.3. *A functional encryption scheme for machine family $\mathcal{M}$ is said to be fully secure if for any PPT adversary $\mathcal{A}$ and polynomial time bound $\overline{T}$, the advantage of $\mathcal{A}$ in the above game is negligible.*

Remark 4.6 (Fully-secure vs Selectively-secure). The security requirement in the definition presented above is for the setting of fully secure functional encryption. We can also define a weaker selectively secure variant in which the adversary is required to specify the challenge messages (x_0, x_1) prior to the public key is provided to it.

Remark 4.7 (Succinctness vs Semi-Succinctness). We will only present our construction for the succinct case and note that it naturally extends to the semi-succinct setting.

Remark 4.8 (Multi-bit Output). Just like for obfuscation we present our functional encryption result for a setting where machines output a single bit. This result can easily be extended to multi-bit output setting by providing a separate private key for each output bit. This increases the running time of both private key generating and decryption procedures by a multiplicative factor of the output length.

Remark 4.9 (Separate time bound $\overline{T}$ for each private key). We note that the definition above (and also the construction) each private key can be generated parameterized by a key specific time-bound. However we assume that each of these time bounds in polynomial in λ.

Remark 4.10 (Input Specific Running Time). In our construction we change the KeyGen to issue secret keys for specific running times. Secret keys that allow for arbitrary running time up to a time bound $\overline{T}$ can be implemented by giving out a sequence of $\log \overline{T}$ secret keys, one for each power of 2 that is less than $\overline{T}$. This allows us to achieve input specific running times for our construction whereby on an input x the decryption process takes time proportional to the running time of M on input x. Specifically, this can be achieved by first executing $\mathsf{SK}_{M,t}$ for $t = 2$ and then doubling it each time, till the output is obtained.

Special Functional Encryption Scheme. We will use a special functional encryption that builds on a arbitrary randomized encoding scheme ($\mathsf{RE.enc}$, $\mathsf{RE.enc}$) (as defined in Definition 3.1) that is not necessarily succinct. Special Functional Encryption is a special case of a randomized functional encryption defined by Goyal et al. [GJKS13]. Very roughly this special functional encryption scheme has the property that the secret key evaluation instead of doing the computation outputs the randomized encoding of the computation being performed.

Definition 4.4. *[Special Functional Encryption] A special functional encryption scheme is defined by a randomized encoding scheme* ($\mathsf{RE.enc}$, $\mathsf{RE.dec}$) *and consists of four algorithms* (Setup, KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$)*:*

- Setup, KeyGen *and* $\mathsf{Encrypt}$ *behave exactly as in Definition 4.2.*
- $\mathsf{Decrypt}(\mathsf{SK}_M, \mathsf{CT})$ - *a polynomial time algorithm that takes a private key* SK_M *and a ciphertext* CT *encrypting message $x \in \{0, 1\}^n$ as input and outputs a value sampled from* $\mathsf{RE.enc}(M, x, \overline{T}, 1^{\lambda})$.

We require the following properties. The Correctness and Security properties are similar to those defined in Definition 4.2. We also require that the running times of the above algorithms are related to those of the underlying randomized encoding scheme, as described in the Efficiency Preserving property.

- **Correctness**: A special functional encryption scheme is said to be *correct* if for all $\lambda, n \in \mathbb{N}$, $y \in \{0,1\}^{p(\lambda,n)}$, all messages $x \in \{0,1\}^n$.

  $$\Pr[(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \mathsf{RE.dec}(\mathsf{Decrypt}(\mathsf{KeyGen}(\mathsf{MSK}, y), \mathsf{Encrypt}(\mathsf{PP}, x))) = y(x)] = 1.$$

  We note that Goyal et al. [GJKS13] needed to define correctness in a more involved manner because they considered general randomized functionalities. But however since in our case the final output is deterministic we can avoid that here.

- **Security**: Exactly as in Definition 4.2.

- **Efficiency preserving**: A special functional encryption is *efficiency preserving* if it satisfies the following conditions:

  - The running time of Setup and $\mathsf{Encrypt}$ is $\mathrm{poly}(\lambda, n)$
  - The running time of $\mathsf{KeyGen}(\mathsf{MSK}, 1^{\log \overline{T}}, M)$ is polynomial in λ (implicit in MSK) and the running time of $\mathsf{RE.enc}(M, x, \overline{T}, 1^\lambda)$ on inputs x of length n (implicit in MSK).
  - The running time of $\mathsf{Decrypt}$ also meets the above bound.

As elaborated in Remark 4.6 we can also define a selectively secure variant of the above definition.

Note that if the underlying randomized encoding scheme is succinct then, the corresponding special functional encryption scheme meets the succinctness condition in Definition 4.2.

Proposition 4.1. *Fully-secure special functional encryption (as in Definition 4.4) for all randomized encoding schemes with encoding circuits in NC^1 exist assuming polynomial hardness of appropriate computational assumptions on composite order multilinear maps as in [GGHZ14].*

Proposition 4.2. *Selectively-secure special functional encryption (as in Definition 4.4) for all randomized encoding schemes exist assuming indistinguishability obfuscation and other appropriate computational assumptions as in [GJKS13].*

We would like to note here that in the special functional encryption schemes constructed by [GGHZ14] and [GJKS13], KeyGen gets as input a randomized circuit C and outputs a secret key corresponding to C. The result of $\mathsf{Decrypt}$ outputs a value sampled from the output of C. However, in our setting KeyGen gets as input a machine M and time bound $\overline{T}$, and at first it may seem their construction can't be used directly. However, we stress that, in our setting, KeyGen outputs a secret key corresponding to the *circuit* that computes the randomized encoding of M and $\overline{T}$. Therefore, we can use their constructions directly.

4.2.2 Our Construction

In this section we will provide our fully-secure (resp., selectively secure) construction of succinct functional encryption using a fully-secure (resp., selectively secure) special functional encryption scheme. We note that since we already have access to a special functional encryption scheme all we need to do is instantiate it with a randomized encoding that is both succinct and has encoding in NC^1 (we will also need to modify the decryption procedure of the special functional encryption scheme to include decoding the randomized encoding). We explain how to obtain such a scheme next.

In order to construct such a succinct randomized encoding scheme we give a transformation from an arbitrary succinct randomized encoding scheme $(\mathsf{sRE.enc}, \mathsf{sRE.dec})$ to a succinct randomized encoding scheme $(\mathsf{sRE.enc}', \mathsf{sRE.dec}')$ which additionally has the property that $\mathsf{sRE.enc}'$ can be computed by an NC^1 circuit. Let $(\mathsf{RE.enc}, \mathsf{RE.enc})$ be a randomized encoding scheme as in Definition 3.1 but without the succinctness requirement. Additionally $\mathsf{RE.enc}$ does not take a time parameter t as input. Constructions of such a randomized encoding scheme, with $\mathsf{RE.enc}$ in NC^1 are known, e.g, Yao's garbled circuits [Yao82]. Let $C_{\mathsf{sRE.enc}}$ be the circuit for evaluating the function $\mathsf{sRE.enc}$. Our succinct randomized encoding scheme $(\mathsf{sRE.enc}', \mathsf{sRE.dec}')$ is as follows:

- $\widehat{C}_{M,x,t} \leftarrow \mathsf{sRE.enc}'(M, x, t, 1^\lambda)$: Output $\mathsf{RE.enc}(C_{\mathsf{sRE.enc}}, (M, x, t, 1^\lambda, r), 1^\lambda)$ where r is fresh randomness sampled for $C_{\mathsf{sRE.enc}}$.
- $y \leftarrow \mathsf{sRE.dec}'(\widehat{C}_{M,x,t})$: Generate $\widehat{M}_{x,t} \leftarrow \mathsf{RE.dec}(\widehat{C}_{M,x,t})$ and output $\mathsf{sRE.dec}(\widehat{M}_{x,t})$.

Note that in the above scheme the $\mathsf{sRE.enc}'$ can be computed in NC^1 due to the fact that $\mathsf{RE.enc}$ can be computed in NC^1. The succinctness of the $(\mathsf{sRE.enc}', \mathsf{sRE.dec}')$ scheme follows from the fact that the circuit $C_{\mathsf{sRE.enc}}$ is succinct. Finally security follows form the security of the two randomized encoding schemes. This allows us to claim.

Theorem 4.2. *A fully-secure (resp., selectively-secure) special FE scheme instantiated with a succinct randomized encoding scheme yields a fully-secure (resp., selectively-secure) succinct FE scheme.*

Corollary 4.1. *Fully secure succinct functional encryption exists assuming appropriate polynomial hardness of assumptions on composite order multilinear maps as in [GGHZ14, GLSW14].*

Corollary 4.2. *Selectively secure succinct functional encryption exists assuming indistinguishability obfuscation and other appropriate computational assumptions as in [GJKS13].*

A folklore corollary of FE is reusable randomized encodings; accordingly, succinct FE implies succinct reusable randomized encodings, where encoding complexity is the same as key derivation complexity. Indeed, succinct FE directly gives a reusable randomized encoding with an indistinguishability guarantee: the encoded machine $\hat{M}$ is just the functional secret key for the machine, an encoded input $\hat{x}$ is just an encryption of x. FE security directly implies that one cannot distinguish two encoded computations $\hat{M}, \hat{x_0}$ and $\hat{M}, \hat{x_1}$, given that $M(x_0) = M(x_1)$. To get simulation, we can use a standard transformation from [DIJ+13].

4.3 Publicly-Verifiable Delegation and SNARGs for P

4.3.1 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 machine, input, time bound, and security parameter $z = (M, x, t, \lambda)$, $\mathcal{G}$ generates a message σ. The prover, given (z, σ), produces a proof π attesting that M accepts x within 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 t, and that the time to prove is polynomially related to t.

Definition 4.5 (**P**-Delegation)**.** *A prover and verifer $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ constitute a delegation scheme for* **P** *if it satisfies:*

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

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

2. **Soundness:** *for any poly-size prover $\mathcal{P}^*$, polynomial $T(\cdot)$, there exists a negligible $\alpha(\cdot)$ such that for any $z = (M, x, t, \lambda)$, such that $t \leq T(\lambda)$, and M does* **not** *accept x within 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. **Fast verification and relative prover efficiency:** *There exists a (universal) polynomial p such that for every $z = (M, x, t, \lambda)$:*

   - *the verifier algoritms $(\mathcal{G}, \mathcal{V})$ run in time $p(\lambda, |M|, |x|, \log t)$;*
   - *the prover $\mathcal{P}$ runs in time $p(\lambda, |M|, |x|, t)$.*

3'. **Semi-fast verification:** *The scheme is semi-succinct 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 $z = (M, x, t, \lambda)$:*

   - *the verifier algoritms $(\mathcal{G}, \mathcal{V})$ run in time $p(\lambda, |M|, |x|, \log t, s)$, where s is the maximal space used by $M(x)$ in t steps.*

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

Remark 4.11 (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 M is being delegated, beyond the output $M(x)$. 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. For this to make sense, we also require that completeness holds even when the honest prover $\mathcal{P}$ is not given the input z.

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

The scheme. Let f be a one-way function and (sRE.enc, sRE.dec) be a randomized encoding scheme. We describe $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ as follows. Let $z = (M, x, t, \lambda)$ be a tuple consisting of a machine, input, time bound and security parameter.

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

For $r \leftarrow \{0,1\}^\lambda$, let $M'(x, r)$ be the machine that returns r if $M(x) = 1$ and $\perp$ otherwise. $\mathcal{G}$ generates and outputs $\sigma \leftarrow \mathsf{sRE.enc}(M', (x, r), t', 1^\lambda)$ and $\tau = f(r)$ where $t' = t + O(\lambda)$ is the bound on the running time of M'.

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

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

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

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

Theorem 4.3. *If* $(\mathsf{sRE.enc}, \mathsf{sRE.dec})$ *is a succinct (resp. semi-succinct) randomized encoding scheme, then* $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ *as described above is a publicly verifiable P-delegation scheme with fast verification (resp. semi-fast verification)*

Proof. The completeness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ follows directly from the correctness of $(\mathsf{sRE.enc}, \mathsf{sRE.dec})$. Also, note that the running time of the verifier algorithms $(\mathcal{G}, \mathcal{V})$ is related to the running time of $\mathsf{sRE.enc}$. Therefore, it also follows directly that if $(\mathsf{sRE.enc}, \mathsf{sRE.dec})$ is succinct (resp. semi-succinct) then $(\mathcal{G}, \mathcal{V})$ satisfies the property of fast verification (resp. semi-fast verification). It remains to show the soundness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$.

To show soundness, we will rely on the security of $(\mathsf{sRE.enc}, \mathsf{sRE.dec})$ and the one-wayness of f. Assume for contradiction there exists poly-size prover $\mathcal{P}^*$ and polynomial $p(\cdot)$ such that for infinitely many $z = (M, x, t, \lambda)$ where M does not accept x within t steps, we have that

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

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

$$\Pr\left[\mathcal{V}(z, \sigma, \tau, \pi) = 1 \;\middle|\; \begin{array}{r} r \leftarrow \{0,1\}^\lambda \\ \sigma \leftarrow \mathsf{Sim}(\perp, M', t', 1^{|x|+|r|}, 1^\lambda) \\ \tau \leftarrow f(r) \\ \pi \leftarrow \mathcal{P}^*(z, \sigma, \tau) \end{array}\right] \geq \frac{1}{p(\lambda)} - \alpha(\lambda) \;.$$

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

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

4.3.2 SNARGs

A succinct non-interactive argument system (SNARG) for **P** is a delegation system where the first message σ is *reusable*, it is independent of any specific computation, and can be used to verify an unbounded number of computations. In a privately-verifiable 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 4.6 (SNARG). *A 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 $\lambda,\overline{T},n \in \mathbb{N}$, and does not get M,x as before. We require that*

1. **Completeness:** *for any $z=(M,x,t,\lambda)$, such that $t \le \overline{T}$ and $|M,x| \le n$, and M accepts x within t steps:*
$$\Pr\left[\mathcal{V}(z,\sigma,\tau,\pi)=1 \;\middle|\; \begin{array}{r}(\sigma,\tau)\leftarrow\mathcal{G}(\lambda,\overline{T},n)\\ \pi\leftarrow\mathcal{P}(z,\sigma)\end{array}\right]=1 \ .$$

2. **Soundness:** *for any poly-size prover $\mathcal{P}^*$, polynomials $\overline{T}(\cdot),n(\cdot)$, there exists a negligible $\alpha(\cdot)$ such that for any $z=(M,x,t,\lambda)$, where $t \le \overline{T}(\lambda)$, $|M,x| \le n(\lambda)$, and M does* **not** *accept x within t steps:*
$$\Pr\left[\mathcal{V}(z,\sigma,\tau,\pi)=1 \;\middle|\; \begin{array}{r}(\sigma,\tau)\leftarrow\mathcal{G}(\lambda,\overline{T}(\lambda),n(\lambda))\\ \pi\leftarrow\mathcal{P}^*(z,\sigma)\end{array}\right]\le\alpha(\lambda) \ .$$

2*. **Strong soundness:** *for any poly-size oracle-aided prover $\mathcal{P}^*$, polynomials $\overline{T}(\cdot),n(\cdot)$, there exists a negligible $\alpha(\cdot)$ such that for any $z=(M,x,t,\lambda)$, where $t \le \overline{T}(\lambda)$, $|M,x| \le n(\lambda)$, and M does* **not** *accept x within t steps:*
$$\Pr\left[\mathcal{V}(z,\sigma,\tau,\pi)=1 \;\middle|\; \begin{array}{r}(\sigma,\tau)\leftarrow\mathcal{G}(\lambda,\overline{T}(\lambda),n(\lambda))\\ \pi\leftarrow\mathcal{P}^{*\mathcal{V}(\cdot,\sigma,\tau,\cdot)}(z,\sigma)\end{array}\right]\le\alpha(\lambda) \ .$$

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

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

Remark 4.12 (Non-adaptive soundness). Note that in the definition above and in our construction, we will consider only *non-adaptive* soundness, as opposed to *adaptive* soundness where the malicious prover $\mathcal{P}^*_d$ 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 SNARG for **P**. The transformation works in either the public or private verification setting. Furthermore, it always results in a 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, $(\mathsf{Key}_{\mathcal{F}},\mathsf{Eval}_{\mathcal{F}},\mathsf{Puncture}_{\mathcal{F}})$ be a puncturable PRF scheme, and $i\mathcal{O}$ be an indistinguishability obfuscator. We describe a SNARG $(\mathcal{P},(\mathcal{G},\mathcal{V})))$ as follows.

Let $z=(M,x,t,\lambda)$ be a tuple consisting of a machine, input, time bound and security parameter such that $|M,x| \le n$ and $t \le \overline{T}$. 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 $\mathcal{G}(\lambda,\overline{T},n)$**:**

1. $\mathcal{G}$ samples a puncturable PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$.

2. Let $C_\sigma[K]$ be a circuit that on input z, runs $r \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, z)$ 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 $r \leftarrow \mathsf{Eval}_{\mathcal{F}}(K, z)$ 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 specified exactly later in the analysis. For now, we mention that if we use a delegation scheme with fast verification then $\ell_\sigma, \ell_\tau \leq \mathrm{poly}(\lambda, n, \log \overline{T})$. We subsequently assume the circuits C_σ and C_τ are padded.

3. $\mathcal{G}$ runs $\sigma \leftarrow i\mathcal{O}(C_\sigma[K], 1^\lambda)$, $\tau \leftarrow i\mathcal{O}(C_\tau[K], 1^\lambda)$ 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 4.4. *If $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ is a privately verifiable (resp. publicly verifiable) P-delegation scheme, then $(\mathcal{P}, (\mathcal{G}, \mathcal{V})))$ as described above is a privately verifiable (resp. publicly verifiable) SNARG with strong soundness.*

Remark 4.13 (SNARGs for NP with a short witness). The above SNARGs are for deterministic computations. We can also deal with a restricted case of **NP** where the witnesses are of bounded size, using standard leveraging. Namely, we will assume that all the underlying primitives are $2^{-O(\ell)}$-secure for the witness length ℓ considered. Verification time and the proof length will accordingly scale with ℓ, but not with the time of the computation.

Proof. The completeness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$ follows directly from that of $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ and the correctness of $i\mathcal{O}$. The running time of $\mathcal{G}(\lambda, \overline{T}, n)$ is polynomial in λ and the maximum running time of $\mathcal{G}_d$ on inputs $z = (M, x, t, \lambda)$ where $|M, x| \leq n$ and $t \leq \overline{T}$. 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 fast verification and prover efficiency of $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$ implies that the same properties hold for $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$.

To show strong soundness of $(\mathcal{P}, (\mathcal{G}, \mathcal{V}))$, we will rely on the soundness of $(\mathcal{P}_d, (\mathcal{G}_d, \mathcal{V}_d))$, and the security of $i\mathcal{O}$ and the punctured PRF $(\mathsf{Key}_{\mathcal{F}}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Puncture}_{\mathcal{F}})$. We will first consider the privately verifiable setting. Assume for contradiction there exists poly-size oracle-aided prover $\mathcal{P}^*$, polynomials $\overline{T}(\cdot)$,$n(\cdot)$,$p(\cdot)$ such that for infinitely many $z = (M, x, t, \lambda)$, where $t \leq \overline{T}(\lambda)$, $|M, x| \leq n(\lambda)$, and M does not accept x within t steps:

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

We will refer to the above probability as the advantage $\mathsf{Adv}(z, \mathcal{P}^*)$. We will now construct a malicious prover $\mathcal{P}^*_d$ to break the soundness of the delegation scheme. $\mathcal{P}^*_d$ gets as input z and σ_d which is some first message of the delegation scheme. $\mathcal{P}^*$ runs a subroutine $\mathcal{D}$ described in the following paragraph, on input (z, σ_d), to obtain a "fake" SNARG message and verification state (σ, τ) which it will then use to run $\mathcal{P}^*$

and answer its queries. That is, $\mathcal{P}_d^*$ runs $(\sigma,\tau) \leftarrow \mathcal{D}(z,\sigma_d)$, $\pi \leftarrow \mathcal{P}^{*\mathcal{V}(\cdot,\sigma,\tau,\cdot)}(z,\sigma)$ and outputs π. The subroutine $\mathcal{D}$ is defined as follows:

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

1. $\mathcal{D}$ samples a puncturable PRF key $K \leftarrow \mathsf{Key}_{\mathcal{F}}(1^\lambda)$ and punctures it at the input z to obtain a punctured key $K_z \leftarrow \mathsf{Puncture}_{\mathcal{F}}(K,\{z\})$.

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 $r \leftarrow \mathsf{Eval}_{\mathcal{F}}(K_z,z^*)$ 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 $\bot$. Otherwise, C_τ^* runs $r \leftarrow \mathsf{Eval}_{\mathcal{F}}(K_z,z^*)$ 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 $\sigma \leftarrow i\mathcal{O}(C_\sigma^*[K_z,\sigma_d],1^\lambda)$, $\tau \leftarrow i\mathcal{O}(C_\tau^*[K_z],1^\lambda)$ 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 $\bot$. 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 $z = (M,x,t,\lambda)$ where M does not accept x within t steps the following holds:

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

Let $\mathcal{Z}$ be the sequence of $z = (M,x,t,\lambda)$ such that $t \leq \overline{T}(\lambda)$, $|M,x| \leq n(\lambda)$, and M does not accept x within t steps.

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 $z \in \mathcal{Z}$, $\mathsf{Adv}_d(z,\mathcal{P}_d^*) = \mathsf{Adv}_d(z,\mathcal{P}_d^{\mathsf{Hyb}})$. Next, we show that relying on the security of the indistinguishability obfuscator and the puncturable PRF, $\mathsf{Adv}_d(z,\mathcal{P}_d^{\mathsf{Hyb}})$ is negligibly close to $\mathsf{Adv}(z,\mathcal{P}^*)$ for all $z \in \mathcal{Z}$. By assumption, $\mathsf{Adv}(z,\mathcal{P}^*)$, is noticeable and hence we have that $\mathsf{Adv}_d(z,\mathcal{P}_d^*)$ 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 $r \leftarrow \mathsf{Eval}_{\mathcal{F}}(K_z,z^*)$ 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 $\tau \leftarrow i\mathcal{O}(C_\tau^*[K_z,\tau_d],1^\lambda)$. $\mathcal{D}^{\mathsf{Hyb}}$ outputs (σ,τ).

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

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

. We first observe that in the experiment corresponding to $\mathsf{Adv}_d(z, \mathcal{P}_d^{\mathsf{Hyb}})$, 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

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

Now viewed this way, we can observe that the only difference between the above experiment and that of $\mathsf{Adv}(z, \mathcal{P}^*)$ 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 $\mathsf{Adv}(z, \mathcal{P}^*)$, (σ, τ) comes from $\mathcal{G}$. It suffices to show the following claim:

Claim 4.1. *The following ensembles are computationally indistinguishable.*

$$\{(\sigma, \tau) : (\sigma, \tau, \pi) \leftarrow \mathcal{D}^{\mathsf{Hyb}}(z, \sigma_d, \tau_d), (\sigma_d, \tau_d) \leftarrow \mathcal{G}_d(z)\}_{z \in \mathcal{Z}} \tag{5}$$

$$\approx_c \{(\sigma, \tau) : (\sigma, \tau) \leftarrow \mathcal{G}(\lambda, \overline{T}(\lambda), n(\lambda))\}_{z \in \mathcal{Z}} \tag{6}$$

Proof. Recall that in ensemble (5), $\sigma \leftarrow i\mathcal{O}(C_\sigma^*[K_z, \sigma_d])$ 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 $\mathcal{G}_d(z^*; \mathsf{Eval}_{\mathcal{F}}(K_z, z^*))$. However, in ensemble (6), $\sigma \leftarrow i\mathcal{O}(C_\sigma[K])$ where C on input z^* outputs $\mathcal{G}_d(z^*; \mathsf{Eval}_{\mathcal{F}}(K, z^*))$. The difference between τ in ensembles (5) and (6) is the same. Indistinguishability follows from the security of $i\mathcal{O}$ and that of $(\mathsf{Key}_{\mathcal{F}}, \mathsf{Eval}_{\mathcal{F}}, \mathsf{Puncture}_{\mathcal{F}})$ in the standard way. We provide a brief overview.

Consider a hybrid ensemble that is identical to ensemble (5) except that instead of uniform randomness $\mathcal{G}_d$ uses randomness from $\mathsf{Eval}_{\mathcal{F}}(K, z)$ 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). Furthermore, the circuits obfuscated as σ and τ in this hybrid ensemble and in ensemble (6) are functionally equivalent. Hence, by the security of $i\mathcal{O}$, ensemble (6) is indistinguishable from the hybrid ensemble. A hybrid argument completes the proof of the claim. $\square$

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

Acknowledgements

We thank Boaz Barak and Guy Rothblum for their input regarding the plausibility of interactive proofs with fast verification (relevant to the plausibility of perfectly-private succinct randomized encodings).

References

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

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

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

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

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

[App11b] Benny Applebaum. Randomly encoding functions: A new cryptographic paradigm - (invited talk). In *Information Theoretic Security - 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings*, pages 25–31, 2011.

[App13] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. Cryptology ePrint Archive, Report 2013/699, 2013. `http://eprint.iacr.org/2013/699`.

[BCC$^+$14] Nir Bitansky, Ran Canetti, Henry Cohn, Shafi Goldwasser, Yael Tauman Kalai, Omer Paneth, and Alon Rosen. The impossibility of obfuscation with auxiliary input or a universal simulator. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 71–89, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

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

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

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

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

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Workshop on Theory and Practice in Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Berlin, Germany.

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

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

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Germany.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Berlin, Germany.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

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

[DIJ$^+$13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 519–535, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

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

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Berlin, Germany.

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

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

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

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

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

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

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private ram computation. In *55th Annual Symposium on Foundations of Computer Science*, 2014.

[GJKS13] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. Cryptology ePrint Archive, Report 2013/729, 2013. `http://eprint.iacr.org/2013/729`.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *46th Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PA, USA, October 23–25, 2005. IEEE Computer Society Press.

[GKP$^+$13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

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

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-Proofs. Cryptology ePrint Archive, Report 2011/456, 2011. `http://eprint.iacr.org/2011/456`.

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

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Berlin, Germany.

[HKW14] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. Cryptology ePrint Archive, Report 2014/521, 2014. `http://eprint.iacr.org/2014/521`.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Redondo Beach, California, USA, November 12–14, 2000. IEEE Computer Society Press.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 244–256, 2002.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. Cryptology ePrint Archive, Report 2013/379, 2013. `http://eprint.iacr.org/2013/379`.

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

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

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

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

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

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

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

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *Journal of the ACM*, 26:361–381, April 1979.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

[SU01] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *42nd Annual Symposium on Foundations of Computer Science*, pages 648–657, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014. `http://eprint.iacr.org/`.

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