On Succinct Obfuscation via Propositional Proofs

Abhishek Jain
NTT Research & JHU
abhishek.jain@ntt-research.com

Zhengzhong Jin
Northeastern University
zhengzhong.jin@northeastern.edu

Surya Mathialagan
NTT Research
surya.mathialagan@ntt-research.com

Omer Paneth
Tel Aviv University
omerpa@tau.ac.il

***Abstract*—A central line of inquiry in the study of indistinguishability obfuscation (IO) is to minimize the size of the obfuscation. Today we know how to obfuscate programs represented as Turing machines, where the size of the obfuscation grows only with the input size and not with the machine's running time. Jain and Jin [FOCS 2022] showed how to remove the dependency on the input size for functionally equivalent programs where equivalence can be proven in Cook's theory PV.**

In this work we investigate the limits of the pursuit of succinct obfuscation. We consider the task of obfuscating a program with a large description, most of which can be made public while some portion of the description is secret. We put forth a new notion of *fully succinct IO* where the size of obfuscated program only grows with the size of the program's secret part and not with the public part or with the input size.

Starting with input-succinct IO for PV-equivalent machines, which is known from super-polynomially hard IO for circuits and LWE, we construct fully succinct IO for the same class of programs. We refer to such an obfuscation as fully succinct pv-IO. Next, we show how to bootstrap our fully succinct pv-IO to achieve full IO security. Our bootstrapping theorems are based on succinct cryptographic primitives with seemingly weaker functionality: either succinct witness encryption or SNARGs for NP with unique proofs. We also require that the correctness of these primitives can be proven in theory PV. We show that these assumptions are sufficient and necessary.

We demonstrate several applications of fully succinct IO and pv-IO:

(i) We give the first IO construction where the size of the obfuscated program is less than twice the size of the original program for a large class of useful programs.
(ii) We show how to avoid padding the program before obfuscating it – a step often necessitated by security analysis – by replacing the padding with a public random string.
(iii) We give the first construction of succinct computational secret sharing for access structures represented by polynomial-size monotone circuits where the share size does not grow with the size of the access structure.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The notion of indistinguishability obfuscation (IO) [6], [28] guarantees that the obfuscations of any pair of functionally equivalent programs are computationally indistinguishable. In the last decade, IO has emerged as the ultimate cryptographic magic wand that can be used to build most cryptographic primitives. Moreover, recent work has shown that this magic wand can be realized from well-founded assumptions [43], [44], [60].

Beyond establishing feasibility, a central goal in the study of IO is to minimize the overhead introduced by obfuscation in terms of program size. Motivated by understanding the very limits of this pursuit, we ask:

How small can an obfuscated program be?

In its simplest form, the notion of IO considers obfuscating programs represented as circuits and the size of the obfuscated circuit is required to be polynomial in the size of the original circuit. A sequence of works [9], [16], [51] constructed more efficient IO for Turing machines, where the size of the obfuscated program grows with the size of the Turing machine representation, but not with its running time.

Input-succinct IO. A major limitation of existing IO schemes for Turing machines is that the obfuscation size grows with the size of the input to the program. In more detail, these schemes set an *a priori bound* on the size of the input to the obfuscated program, and the obfuscation size grows with this bound. Ideally, we would like to obfuscate Turing machines such that the obfuscated program can be evaluated on inputs of

Identify applicable funding agency here. If none, delete this.

arbitrary polynomial size. We refer to this notion as *input-succinct* IO.

A recent work of [41] gives a partial feasibility result for this notion. Assuming (sub-exponentially hard) IO for circuits and other standard assumptions, they construct an input-succinct obfuscation with the following security property that we refer to as *pv-IO*: obfuscations of two functionally equivalent programs are guaranteed to be computationally indistinguishable only if the equivalence of the two programs can be proven in Cook's theory PV [20]. From Gödel's incompleteness theorem, however, it follows that there exist functionally equivalent programs without a PV proof of equivalence. Furthermore, prior works [29], [41], [53] have described an informal "input-size barrier" to achieving input succinct IO for general programs: Intuitively, the reduction must run long enough to decide the equivalence of the programs. Thus, the feasibility of this notion remains a major open question.

Fully succinct IO: A New Frontier. While input-succinctness is already an ambitious goal, we could go a step further and ask:

Can the obfuscated program be shorter than the program itself?

While the answer, in general, is clearly negative, we observe that in many applications, the program we would like to obfuscate has a large description, but the actual secret code that we are aiming to protect is small. Is it possible to obfuscate only the secret part of the program, leaving the public part unchanged? For example, recall the construction of *witness encryption* from IO [28]: to encrypt a bit b under an NP statement x, we obfuscate a program that contains x hardwired. Given an input w, the program outputs b only if w is a valid witness for x. In this example, the bit b is secret but the statement x is public. Nonetheless, to protect b, we obfuscate a program that includes x. Using any existing obfuscation scheme, the obfuscated program grows with the description of x which may be large. Is it possible to encrypt b under a statement x via an obfuscated program whose size does not grow with the size of x?

Motivated by this setting, we formalize a new notion of *description-succinct IO*: Consider a program whose description consists of a public part pub and a (potentially) secret part M and denote by $M[\mathsf{pub}]$ the description of the program including both parts. We refer to such programs as having a *split description*. A description-succinct IO takes both M and pub as inputs and produces an obfuscated program $\widetilde{M}$ such that $M[\mathsf{pub}]$ and $\widetilde{M}[\mathsf{pub}]$ have the same functionality. Importantly, $\widetilde{M}$ should only grow with the size of M and not with the size of pub. For security, we require that for every M_0, M_1 of the same size, and for every pub such that $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ are functionally equivalent, the obfuscations $\widetilde{M_0}$ and $\widetilde{M_1}$ are computationally indistinguishable, even given pub.

We refer to an IO scheme that is both input-succinct and description-succinct as *fully succinct*. Using this terminology, the main question we tackle in this work is:

Are input-succinct and fully succinct IO possible? If so, under what assumptions?

While the notion of description-succinctness was not previously considered,[1] we observe that constructions of description-succinct IO which are not input-succinct can be realized from prior work [3], [24], [31]. Fully succinct constructions, however, are not known even for the weaker notion of pv-IO.

II. Our Results

In this work, we investigate the feasibility and applications of fully succinct IO:

- **Upgrading to full succinctness.** Our first result focuses on pv-IO, where indistinguishability only holds for pairs of programs with a PV-proof of equivalence. We formalize the notion of fully succinct pv-IO and show how to upgrade any input-succinct pv-IO to fully succinct pv-IO under standard assumptions.
- **Upgrading to full IO security.** Next, we show how to upgrade any fully succinct (resp. input-succinct) pv-IO to achieve full security, namely indistinguishability for any two functionally equivalent programs. This bootstrapping theorem is based on the existence of either succinct witness encryption [13] or succinct non-interactive arguments for NP (SNARGs) with unique proofs and short CRS where the correctness of the scheme can be proven in theory PV. While these are strong assumptions, we argue they are necessary since they are implied by fully succinct IO.
- **Applications.** We demonstrate several applications of fully succinct pv-IO and IO. This includes the first construction of succinct computational secret sharing for general monotone access structures where the share size does not grow with the size of

[1]A recent concurrent work [55] introduced the notion of IO for database-aided circuits, which is similar to description-succinct IO except that their focus is on the running time of the obfuscated program and not its description size. See Section II-D for more details.

the access structure, thus overcoming the so-called "representation-size barrier" [4]. We also show how to further reduce the size of the obfuscated programs in many applications of IO. For example, we give the first IO construction with rate above $1/2$ (i.e., where the obfuscated program is less than twice as large as the original program) for a large class of useful programs. Finally, we show how to generically avoid the "program padding" necessary in many IO applications [37] by replacing it with a public random string.

Technical highlight: How to use pv-IO. The notion of pv-IO is instrumental in our results on succinct obfuscation. A central challenge that we encounter when using pv-IO is that the programs we need to transition between often do not have a PV-proof of equivalence. This significantly limits the applicability of pv-IO; indeed, so far, this notion has found only limited applications.

In this work, we demonstrate how to use pv-IO in conjunction with other "PV-friendly" cryptographic primitives (e.g. primitives with a PV-proof of correctness) to transition between functionally equivalent programs that are *not* PV-equivalent. Using this new approach, we build new applications based on pv-IO, without requiring full security. Our approach is reminiscent of the work of [62] who demonstrated how cryptography can be combined with IO security when obfuscating programs that are not functionally equivalent. Our techniques are also inspired by the recent line of work that demonstrated how to use propositional proofs in the context on SNARGs [46], [47].

Next we elaborate on each of our contributions.

A. Upgrading to Full Succinctness

We start by introducing the notion of fully succinct pv-IO. The work of [41] introduced the notion of pv-IO where security holds for all pairs of programs M_0 and M_1 whose equivalence can be proven in Cook's theory PV. Following [56], we use an alternative formulation of pv-IO based on the more general notion of uniform $\mathcal{EF}$-equivalence. Roughly, we require that there is a uniform polynomial-time algorithm that given 1^n outputs a proof for the fact that M_0 and M_1 are equivalent on all n-bit inputs in the extended Frege propositional proof system. Cook's propositional translation theorem shows that every pair of PV-equivalent programs are also uniform $\mathcal{EF}$-equivalent [20], [22].

Let $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ be a pair of programs with split description and the same public part. We say that programs are (ρ, T)-$\mathcal{EF}$ *equivalent* for a constant ρ and a polynomial T if they have the same size and running time, and there exists a TM $\mathcal{M}_\Pi$ of size ρ that on input $n, M_0, M_1, \mathsf{pub}$ runs in time $T(n)$ and outputs an extended Frege proof that $\forall x \in \{0,1\}^n$, $M_0[\mathsf{pub}](x) = M_1[\mathsf{pub}](x)$. In our formulation of pv-IO, the obfuscation algorithm is parameterized by ρ and T, and the security guarantee is that the obfuscations of $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ are indistinguishable assuming the programs are (ρ, T)-$\mathcal{EF}$ equivalent. In *input-succinct pv-IO*, the size of the obfuscated program depends polynomially on ρ and on the size of the programs (both secret and public parts). We also allow the obfuscated program running time on size-n inputs to grow with $\mathsf{poly}(T(n))$. The obfuscation is *fully succinct* if the size of the obfuscated program only depends on ρ and on the size of the programs' secret part, but not on the public part pub.

We show the following result:

Theorem II.1 (Informal)**.** *Assuming input-succinct pv-IO and LWE, both with slightly super-polynomial security, there exists* fully succinct *pv-IO.*

Input-succinct pv-IO can be constructed from IO and one-way functions with sub-exponential security and LWE with slightly super-polynomial security [41]. Therefore, we achieve fully succinct pv-IO under the same assumptions as known constructions of input-succinct pv-IO.

B. Upgrading to Full IO security

In this section, we state our bootstrapping theorem from pv-IO to full-fledged IO. We give two versions of the theorem: one for input-succinctness and one for full succinctness. The theorem requires either witness encryption (WE) or SNARG for NP with unique proofs, with matching succinctness. We also show that these assumptions are implied by IO. We therefore state the result as a conditional equivalence between the notions of succinct IO, WE and unique SNARGs.

Theorem II.2. *Assuming fully succinct pv-IO and LWE, the following are equivalent:*

1) *There exists fully succinct IO (resp. input-succinct IO) with a PV-proof of correctness.*
2) *There exists fully succinct WE (resp. witness-succinct WE) with a PV-proof of correctness.*
3) *There exists fully succinct SNARG (resp. witness-succinct SNARG) with a PV-proof of correctness and uniqueness*[2].

In more detail, we show that:

[2]A previous version of this paper showed an equivalence to a weaker version of fully succinct SNARGs. The result in the current version is inspired by the construction in [45].

- (3) $\Rightarrow$ (2) assuming one-way functions.
- (2) $\Rightarrow$ (1) assuming one-way functions and fully succinct pv-IO.
- (1) $\Rightarrow$ (3) assuming LWE.
- (1) $\Rightarrow$ (2) assuming one-way functions.

Next, we elaborate on the each of the components of Theorem II.2. In witness encryption, we can encrypt a message under an NP statement such that: (a) we can decrypt the ciphertext using any valid witness, and (b) if the statement is false, the message is computationally hidden. We say that a WE scheme is *witness-succinct* if the ciphertext only grows with the size of the message and the statement and not with the size of its witness.[3] The scheme is *fully succinct* if the ciphertext only grows with the security parameter and message size and not with the size of the statement or witness.

SNARGs are succinct non-interactive argument for NP in the common reference string (CRS) model [8], [57]. A SNARG is said to have unique proofs if for every CRS and statement, the honest prover outputs the same proof regardless of the witness [17]. We say that a SNARG is *witness-succinct* if the CRS and proof only grow with the size of the statement and not with the size of a witness. The SNARG is *fully succinct* if the CRS and proof only grow with the security parameter and not with the size of the statement or witness.

As shown in prior works [41], [47], many cryptographic schemes with perfect correctness have proofs of correctness in theory PV. Therefore, we expect "natural" candidate constructions of cryptographic primitives with perfect correctness to have a PV-proof of correctness.

Discussion: The input-size barrier and succinct cryptography. In the non-succinct setting, IO (for circuits) is known to imply WE [29] and SNARGs with unique proof [62], but an implication in the other direction is not known. Nonetheless, Theorem II.2 states that starting from fully succinct pv-IO (that follows from subexponential IO for circuits and LWE with slightly superpolynomial security), the notions are equivalent in both the fully succinct and the input/witness-succinct settings, assuming the correctness of the schemes is provable in theory PV.

As mentioned earlier, we believe that constructing fully succinct or even input-succinct IO requires new techniques that cross the so-called "input-size barrier". We note that witness-succinct WE and SNARGs are subject to similar barriers [29], [32].[4] Therefore, we do not interpret our result as crossing the input-size barrier. Instead we show that if we overcome the barrier in any one of these primitives, we can overcome it in all of them.

Instantiations. While witness-succinct WE is currently not known under standard cryptographic assumptions, a very recent work of [13] proposed lattice-based constructions of witness-succinct and fully succinct WE. While the assumption of evasive LWE used in their construction is subject to attacks [1], [13], [14], [26], [38], [64], we are not aware of attacks to the WE construction itself. Thus, it may potentially be proven secure under a weaker assumption. As we show in the full version of this paper, the correctness of the [13] construction can be shown in theory PV. This therefore gives us a candidate construction of fully succinct IO.

Witness-succinct SNARGs with unique proofs are currently not known under standard cryptographic assumptions. However, [66] recently constructed adaptive SNARGs for NP with unique proofs assuming subexponential IO for circuits. While their construction has fully succinct proofs, the CRS grows with the size of the instance and witness. By proving the correctness of this construction in theory PV, and using a variant of the above theorem, we obtain a fully succinct IO in the CRS model. In this model, the same CRS can be used to obfuscate multiple, adaptively chosen programs. The size of the CRS bounds the size of the entire program (public and secret part) as well as the the size of the input to the program. However, the size of the obfuscated program itself grows only with the security parameter and not with the size of the input or the public part. Previously, [5] constructed input-succinct IO in the CRS model, where the obfuscated program grows with the security parameter and the full description of the program, but not with the input size.

C. Applications

Next, we describe applications of fully succinct IO and pv-IO for overcoming the representation-size barrier in computational secret sharing, and for further reducing the size of obfuscated programs.

I. Overcoming the representation-size barrier in secret sharing. In a secret sharing scheme [11], [40], [63],

[3] I.e., the ciphertext size is bounded by a fixed polynomial in the statement, message size and the security parameter and this polynomial does not depend on the NP relation.

[4] Roughly speaking, this is because the security of these notions is predicated on a non-falsifiable premise: in the case of IO, it is whether two programs are functionally equivalent, while in the case of WE and SNARGs, it is whether an NP statement is false. The difficulty of *efficiently* checking this premise presents a challenge in achieving succinctness.

a dealer distributes a secret between n parties such that only authorized subsets of parties can reconstruct the secret. In more detail, the access structure, which is the collection of authorized subsets, is given by a monotone function $f : 2^{[n]} \to \{0,1\}$. The dealer shares a secret $s \in \{0,1\}$ into shares $s_1, \dots, s_n$ and we require that for every subset T of the parties:

- *Correctness:* if $f(T) = 1$, it is possible to reconstruct s given the shares $\{s_i\}_{i \in T}$.
- *Secrecy:* if $f(T) = 0$, the shares $\{s_i\}_{i \in T}$ reveal nothing about s.

In this work, we focus on the computational setting: We assume that the access structure f has some efficient representation, such as a polynomial-size monotone circuit, and require that the sharing and reconstruction procedures are efficient. Moreover, we only require computational secrecy.

A central goal in secret sharing is to minimize the share size. There is a vast body of literature on this topic, and we refer the reader to II-D for discussion. In this work, our focus is on overcoming a key challenge in this area: Designing schemes where the size of the shares does not grow with the representation-size of the access structure f.

Succinct Computational Secret Sharing for Monotone Circuits. Based on one-way functions, Yao [65], [68] gave a construction where each share is small (grows only with the security parameter), but parties need to access some public information that grows linearly with the size of the monotone circuit for f. Recently, [4] improved the size of the public information to linear in the number of $\wedge$ gates in the circuit assuming either subexponential RSA or a combination of subexponential IO for circuits, and somewhere statistically binding hash.

As an application of fully succinct pv-IO (see Theorem II.1), we show how to completely remove the dependency on the circuit size in the shares. This yields the first scheme that overcomes the representation-size barrier for access structures described as monotone circuits.

Theorem II.3. *Assuming fully succinct pv-IO and LWE, there exists an n-party computational secret sharing scheme for any access structure given by a polynomial-size monotone circuit f where the size of the share is* $\mathsf{poly}(\lambda)$ *and the public information is a uniform n-bit string.*

Our result is significantly more general: First, Theorem II.3 can be generalized to support *any* access structure f for which monotonicity can be proven in theory PV. This includes all polynomial size monotone circuits (as we show in Lemma VII.10), but also structures for which no efficient monotone circuit exists such as matching in a bipartite graph [61]. Previously, no succinct schemes were known for such access structures. Second, we can further reduce the size of public information from n to C if every unauthorized set has a zero certificate with a description size at most C.[5] For example, we get succinct secret sharing for the bipartite matching access structure where the public information is of size $2\sqrt{n}$ (Corollary VII.16).

Succinct Computational Secret Sharing for all of NP. If we instead start from fully succinct IO instead of pv-IO, we can significantly increase the class of supported access structures to include all monotone functions in NP [50].

Theorem II.4. *Assuming fully succinct IO and LWE, there exists an n-party computational secret sharing scheme for any access structure given by a monotone function f in NP, where the size of the share is* $\mathsf{poly}(\lambda)$ *and the public information is a uniform n-bit string.*

Previously, [50] constructed secret sharing for all monotone functions in NP based on WE and one-way functions. Instantiating their scheme with fully succinc WE (instead of IO) yields succinct secret sharing that requires *structured* (as opposed to uniform) public information of size $n \cdot \mathsf{poly}(\lambda)$.

II. Further reducing the obfuscated program size. In fully succinct pv-IO and IO, the obfuscated program size does not grow with the input size or with the public part of the program. Next we tackle two additional sources of blowup in the obfuscated program size:

- **Rate:** The rate of an obfuscation scheme is the ratio between the description size of the original program and the obfuscated program. The IO constructions of [2], [10], [42] achieve rate $1/2$ for circuits and Turing machines (without input succinctness). Crossing the rate $1/2$ barrier for IO is an open question [2], [10], [42], [55].
- **Padding:** In many applications of IO, the program is padded before it is obfuscated to facilitate the security proof where the program being obfuscated is replaced by some larger program. [37] shows that in some cases such padding is *unavoidable*.

As an application of fully succinct pv-IO and IO, we give the first construction of IO with rate higher than

[5]A zero certificate for a set T such that $f(T) = 0$ is any $T' \supseteq T$ satisfying $f(T') = 0$.

$1/2$ for a large class of programs. Moreover, we show how to generically replace padding before obfuscation with a public random string of the same length. More generally, we show how to reduce the size of the obfuscated program in applications where IO security is used for pairs of programs that are not only functionally equivalent, but also *close* to each other in some sense. For simplicity, we present our results below for programs with no public part. However, our results also hold for programs with split description.

ρ-closeness. Very roughly, we say that a pair of programs are ρ-close if it is possible to transition between then through a sequence of functionally equivalent programs such that each program is of size at most ρ, and the Hamming distance between each pair of consecutive programs is constant.

We proceed to describe the notion of ρ-closeness more formally. Let $M_1 = \{M_{1,\lambda}\}_\lambda$ and $M_2 = \{M_{2,\lambda}\}_\lambda$ be a pair of Turing machine families.

- We say that M_1 and M_2 are *adjacent* if $M_{1,\lambda}$ and $M_{2,\lambda}$ are of the same size, they are functionally equivalent, and their descriptions only differ by a constant number of bits.
- We say that M_1 and M_2 are *close* if we can transition between them via a polynomial-size sequence of adjacent machines. That is, if there exists $\ell = \mathsf{poly}(\lambda)$ and $M'_1, \dots, M'_\ell$ such that $M_1 = M'_1$, $M_2 = M'_\ell$ and for every i, M'_i and M'_{i+1} are adjacent.
- We say that M_1 and M_2 are *ρ-close* for a function $\rho(\lambda)$ if M'_1 and M'_2 are close where $M'_{i,\lambda}$ is the program $M_{i,\lambda}$ padded to size $\rho(\lambda)$.

For example, observe that any pair of program families M_1, M_2 are ρ-close for $\rho(\lambda) = |M_{1,\lambda}| + |M_{2,\lambda}| + O(1)$: Start with M_1 padded to size ρ. Next, modify the padding, bit by bit, to include the description of M_2. Then switch from executing M_1 to M_2 by changing only the initial state. Finally, modify the padding, bit by bit, to remove the description of M_1.

Obfuscating ρ-close programs. Using fully succinct IO, we show how to obfuscate pairs of programs that are ρ-close such that the size of the obfuscated program is roughly ρ. Moreover, if the original programs are of size ℓ, then we can reduce the obfuscated program to be of size roughly ℓ by using additional $\rho - \ell$ bits of *public randomness*. More concretely, an obfuscation algorithm with *public randomness* is given a program M and a string r and it produces an obfuscated program $\widetilde{M}$. Correctness states that for every M and r, we have that M and $\widetilde{M}[r]$ are functionally equivalent. We say that the scheme is secure for a pair of machines M_1, M_2 with ℓ bits of public randomness, if the programs $\widetilde{M}_1$ and $\widetilde{M}_2$ obfuscated using a random ℓ-bit string r, are indistinguishable, even given r.

Theorem II.5 (Informal)**.** *Assuming fully succinct IO and one-way functions, there exists an obfuscation scheme with public randomness that is secure for every pair of ρ-close machine families of size ℓ with $\rho - \ell$ bits of public randomness, where the obfuscated programs are of size $\ell + \mathsf{poly}(\lambda)$.*

Since every pair of program families M_1, M_2 of size ℓ are $(2\ell + O(1))$-close, we get IO with rate $1/2$ as an immediate corollary of the above theorem. Compared with the result of [2], this construction has the advantage that given public randomness of size ℓ, the obfuscated program is only $\mathsf{poly}(\lambda)$ bits larger than the original program. Moreover, the construction from [2] is not input-succinct while our construction can be made fully succinct.

We observe that in many IO based constructions, IO security is used for programs that are close to each other. In such applications we can use Theorem II.5 to get higher rate. We give some concrete examples:

- The works of [34], [67] show that several applications of IO can be obtained based on the weaker notion of null-IO where we only require security for programs computing the all-zero function. Since any program of size ℓ that is functionally equivalent to the all-zero program is also $(\ell + O(1))$-close to it, we obtain a construction of null-IO with rate 1.
- In the setting of batch IO, we obfuscate together a batch of k programs, each of size ℓ into a single obfuscated program [2]. In this setting, we achieve obfuscation of size $(k+1) \cdot \ell + \mathsf{poly}(\lambda)$ compared to $2k \cdot \ell + \mathsf{poly}(\lambda, n)$ in [2] where n is the bound on the input length to the TM. Moreover, given ℓ bits of public randomness, the size of the obfuscation can be reduced to $k \cdot \ell + \mathsf{poly}(\lambda)$.
- In Theorem II.1, the security proof involves hard-coding the uniform-$\mathcal{EF}$-proof of equivalence between the obfuscated programs. For example, if machines $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ have a uniform $\mathcal{EF}$ proof of size ρ, then the size of our obfuscation is $\mathsf{poly}(\lambda, |M_b|, \rho$. Using Theorem II.5, we can instead replace the padding with a public random string of size γ, and reduce the obfuscation size to $\mathsf{poly}(\lambda, |M_b|)$.

D. Related work

Negative results in the circuit model. In a concurrent work, [55] give an impossibility result for IO with rate very close to 1. Their result considers IO for circuits where the obfuscated program is also represented as a circuit and the rate of the obfuscation is defined as the ratio between the size of the original circuit and the obfuscated one. In contrast, in this work the obfuscated program is modeled as a Turing machine and the rate is defined via the size of the program representation.

The work of [55] also gives an impossibility for a notion of IO with access to a public database where the obfuscated program (represented as a circuit) does not grow with the database size. This notion is similar to our notion of description-succinct IO, except that we only restrict the size of the obfuscated program modeled as Turing machine. In particular, in the notion of [55] the obfuscated circuit cannot even touch the entire public database. The result of [55] can be interpreted as showing that extending our notion of fully succinct obfuscation to also require sublinear running time is unlikely.

Quasi-Linear pv-IO. In a recent work, Ma, Dai and Shi [56] construct input-succinct pv-IO with quasi-linear overhead, namely, where the running time of the obfuscated program grows quasi-linearly in the total size of the Turing machine and the PV-proof of equivalence. This improves upon the previous work of [41] which achieved larger runtimes. While their work also aims to improve the efficiency of obfuscation, their focus is on minimizing the obfuscation overhead in terms of running time while our focus in this work is on minimizing the obfuscated program size.

Succinct Secret Sharing. There is a vast body of literature dedicated to the study of share size in secret sharing. We highlight the most relevant prior work and refer the reader to [4] for a more detailed survey. As mentioned earlier, all prior schemes for polynomial-size monotone circuits [4], [65], [68] require a share size that grows with the size of the circuit. Assuming sub-exponential RSA, or sub-exponential IO for circuits and SSB hash family, the work of [4] also constructs succinct secret sharing for CNF formulas where the share size is independent of the number of clauses. This, in turn, yields a succinct but *inefficient* secret sharing scheme for arbitrary access structures represented by truth tables. A similar result was recently obtained by [23] by constructing a succinct secret sharing for DNFs based on witness encryption and LWE (in the random oracle model). In contrast to these schemes, our scheme has efficient sharing and reconstruction for functions represented as monotone circuits and more.

We also mention two generic results based on IO. Assuming sub-exponential IO for circuits and SSB hash family, one can generically compress the shares of any non-succinct secret sharing scheme [12], [35]; the resulting scheme, however, requires large public information whose size grows with the total share size of the underlying non-succinct scheme. For uniform access structures that are succinctly described by TMs, one can build succinct secret-sharing by instantiating the approach of [50] with a non-input-succinct IO for TMs [51].

In a concurrent work, Lu, Nassar and Waters [54] provide a different construction of succinct computational secret sharing for monotone circuits matching the efficiency of our scheme as stated in Theorem II.3. Their scheme relies on weaker assumptions, namely IO for circuits and injective one-way functions (while we additionally require LWE). Our result, however, is more general and applies to all monotone functions as long as monotonicity can be proven in PV. We also reduce the size of the public share blow n for some structures.

SNARGs from Propositional Proofs. A recent work of [47] (JKLM) constructs SNARGs for NP from WE with PV-proof of correctness, and other standard assumptions. Given a witness-succinct (resp., fully succinct) WE, the resulting SNARG is witness-succinct (resp., fully succinct) SNARG. Unlike the SNARGs given by our equivalence results, the SNARG constructed in JKLM does not have unique proofs. Similar to JKLM, our work also leverages PV-proofs of correctness. Furthermore, we also demonstrate the usefulness of formalizing other properties in theory PV such as PV-proof of binding and PV-proof of monotonicity.

E. Organization

We first give a technical overview in Section III. In Section IV, we give an overview of definitions, tools and instantiations of cryptographic primitivies which we will use throughout our work. In Section V, we construct fully succinc pv-IO. In Section VI, we show a three-way equivalence between succinct versions of IO, WE and SNARGs. We then give our construction of succinct computational secret sharing and analysis is Section VII. We then show our "necessary padding" technique and applications to high-rate IO in Section VIII. We include more details on Cook's theory PV proofs in Appendix A. We defer many proofs to the full version of this work.

III. Technical Overview

In this section, we provide an overview of the main ideas underlying our results. In Section III-A, we first describe an overview of Theorem II.1. Then, we demonstrate how to obtain our three-way equivalence as in Theorem II.2 in Section III-B. We then discuss our construction of succinct secret sharing (Theorems II.3 and II.4) in Section III-C. Finally, we discuss our padding results as in Theorem II.5 in Section III-D.

A. Upgrading to Full Succinctness

Before we show how to upgrade input-succinct pv-IO to additionally be fully succinct, we will discuss how to first construct IO satisfying description succinctness (Section III-A1). Then, we will discuss our construction and proof of security in Section III-A.

1) Warm-up: Description-Succinct IO: We start by describing a simple construction of description-succinct IO[6] from somewhere statistically binding (SSB) hash functions and IO for circuits with sub-exponential security. Looking ahead, we will use ideas from this construction in our transformation from input-succinct pv-IO to fully succinct pv-IO. The description-succinct IO construction is based on the notion of succinct randomized encoding [3], [9], [16], [31] where given TM M and input x, we can generate an encoding of (M, x) such that:

Correctness: Given the encoding we can decode the output $M(x)$.

Security: The encoding reveals nothing except the output.

Succinctness: The time to generate the encoding grows only with $|M|$ and $|x|$ and not with the running time of M.

We focus on the succinct randomized encoding construction of [31] where the encoding of (M, x) consists of two parts:

- A short encoding $\widehat{M}$ that may depend on both M and x, but whose size grows only with $|M|$.
- A long encoding $\widehat{x}$ that does not depend on M.

Furthermore, we observe that if only M is secret and input x can be made public, the long encoding $\widehat{x}$ can just be x itself and the encoding $\widehat{M}$ can be computed given only M and a short digest d_x of x without knowing x. In the setting where x is public, the security guarantee is that for every pair of TMs M_0, M_1 and public input x such that $M_0(x) = M_1(x)$ the short encodings $\widehat{M_0}$ and $\widehat{M_1}$ are indistinguishable given x.

To turn this succinct randomized encoding into description-succinct IO, we use the standard transformation from randomized encoding to IO for TMs that is based on IO for circuits with sub-exponential security. To obfuscate M and pub, we use IO for circuits to obfuscate a program $U_{M,d_{\mathsf{pub}}}$ that contains the description of M and the digest d_{pub} of pub hard-coded. Given an input x, $U_{M,d_{\mathsf{pub}}}$ proceeds as follows:

- Derive randomness r for the randomized encoding by evaluating a puncturable PRF on x.
- Let M_x be the TM that has x hardcoded and given input pub outputs $M[\mathsf{pub}](x)$.
- Using d_{pub}, generate and output the short encoding $\widehat{M_x}$ of (M_x, pub).

Given the public input pub, we evaluate the obfuscation on input x by first evaluating the obfuscation of $U_{M,d_{\mathsf{pub}}}$ on x obtaining the short encoding $\widehat{M_x}$. Then we decode $(\widehat{M_x}, \mathsf{pub})$ to obtain the output $M[\mathsf{pub}](x)$. The construction is description-succinct since the size of $U_{M,d_{\mathsf{pub}}}$ does not grow with pub or with the running time of M. However, the construction is *not* input-succinct for two reasons:

- Since we are using IO for circuits to obfuscate $U_{M,d_{\mathsf{pub}}}$, the obfuscation grows with $|x|$.
- The security of the construction follows via a hybrid argument with one hybrid experiment for every input x. Therefore, we must rely on IO with sub-exponential security and let the security parameter grow with $|x|$.

2) Our approach: The above construction of description succinct IO suggest a natural approach for upgrading input-succinct pv-IO to fully succinct pv-IO[7]: simply obfuscate the program $U_{M,d_{\mathsf{pub}}}$ using input-succinct pv-IO instead of IO for circuits. The problem with this approach is that even if a given pair of programs $M_0[\mathsf{pub}], M_1[\mathsf{pub}]$ are $\mathcal{EF}$-equivalent, the programs $U_{M_0,d_{\mathsf{pub}}}$ and $U_{M_1,d_{\mathsf{pub}}}$ are *not* functionally equivalent.

To achieve full succinctness, we combine input-succinct pv-IO with succinct randomized encodings in a much more careful manner. In a nutshell, we "open up" the construction of succinct randomized encodings based on IO for circuits and SSB hash, and re-prove its security relying only on pv-IO security.

[6]Recall that this means that the obfuscation size does not grow with the public description of the program, but may still grow polynomially with the input length.

[7]As alluded to earlier, we only need to construct a fully succinct scheme $\mathcal{O}$ satisfying ρ-R-PV security, but we will defer this detail to the proof sketch.

Our construction. We now describe our construction of fully succinct pv-IO. Let $\mathcal{O}$ be any input-succinct pv-IO scheme (whose parameters ρ' and T' we will choose later) and let Hash be a collision resistant hash.

$\mathsf{pviO}(1^\lambda, M[\mathsf{pub}])$: Given M and pub, we use $\mathcal{O}$ to obfuscate the following program $U_{M,\mathsf{hk},d_{\mathsf{pub}}}$:

- The program contains a hash key hk and the hash $d_{\mathsf{pub}} = \mathsf{Hash}(\mathsf{hk}, \mathsf{pub})$ hardcoded.
- Given as input pub' and x, $U_{M,\mathsf{hk},d_{\mathsf{pub}}}$ outputs $M[\mathsf{pub}'](x)$ if $\mathsf{Hash}(\mathsf{hk}, \mathsf{pub}') = h$ and outputs $\perp$ otherwise.

Output $\mathcal{O}(U_{M,\mathsf{hk},d_{\mathsf{pub}}})$.

Note that while the obfuscated program takes pub as an additional input, the size of the obfuscated program does not need to grow with pub since $\mathcal{O}$ is input-succinct.

To argue security, fix a pair of programs M_0, M_1 and a public input pub such that $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ are (ρ, T)-$\mathcal{EF}$-equivalent. We cannot directly rely on the security of $\mathcal{O}$ since, while $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ are $\mathcal{EF}$-equivalent, the programs $U_{M_0,\mathsf{hk},d_{\mathsf{pub}}}$ and $U_{M_1,\mathsf{hk},d_{\mathsf{pub}}}$ may not be functionally equivalent, let alone $\mathcal{EF}$-equivalent. Instead we transition from $U_{M_0,\mathsf{hk},d_{\mathsf{pub}}}$ and $U_{M_1,\mathsf{hk},d_{\mathsf{pub}}}$ trough a sequence of hybrids. In some of the hybrids we transition between two PV-equivalent programs relying on the security of $\mathcal{O}$, while in other hybrids we argue indistinguishability by invoking the security of other cryptographic primitives.

Proof sketch. We denote by pviO the resulting scheme. Suppose we would like to show security for (ρ, T)-equivalent machines $P_0[\mathsf{pub}]$ and $P_1[\mathsf{pub}]$. We will in fact break our proof up into multiple hybrids. For $i \in \{0, 1, \ldots, N\}$, consider the following sequence of programs:

$$P^{(i)}[\mathsf{pub}](x) = \begin{cases} P_1[\mathsf{pub}](x) & \text{if } |x| \le i, \\ P_0[\mathsf{pub}](x) & \text{otherwise.} \end{cases}$$

If we show that

$$\mathsf{pviO}(1^\lambda, M^{(i)}[\mathsf{pub}]) \approx_c \mathsf{pviO}(1^\lambda, M^{(i+1)}[\mathsf{pub}]),$$

it is easy to see that a standard hybrid argument gives us that

$$\mathsf{pviO}(1^\lambda, M^{0}[\mathsf{pub}]) \approx_c \mathsf{pviO}(1^\lambda, M^{N}[\mathsf{pub}]).$$

Therefore, it suffices to consider the above special case.

Fix $i \in \{0, \ldots, N-1\}$, and let $M_0 = P^{(i)}$ and $M_1 = P^{(i+1)}$. We now define the following relation R (which has i and T hardcoded in it).

$R(M_0, M_1, \mathsf{pub}, \mathcal{M}_\Pi)$:

- Verify that there exists P_0 and P_1 such that $M_b = P^{(i+b)}$, where $P^{(i)}$ as defined above.
- Let $\tau \leftarrow \mathcal{M}_\Pi(i+1, P_0, P_1, \mathsf{pub})$, where $\mathcal{M}_\Pi$ is run for $T(i+1)$ steps.
- Output 1 if τ is an $\mathcal{EF}$ proof that $M_0[\mathsf{pub}](x) = M_1[\mathsf{pub}](x)$ for all $x \in \{0,1\}^{i+1}$. Output 0 otherwise.

In our security proof, we will use this relation R. Looking forward, we will use the fact that the following claim can be proven in PV:

$$\forall M_0, M_1, \mathsf{pub}, x,$$
$$(R(P_0, P_1, \mathsf{pub}, s) = 1) \rightarrow (M_0[\mathsf{pub}](x) = M_1[\mathsf{pub}](x).) \quad (1)$$

where s is a constant. The proof proceeds as follows:

- We consider three cases, x of length at most i, length at least $i+2$, and length equal to $i+1$.
- For all x of length at most i, R certifies that there exists $P_1[\mathsf{pub}]$ such that both $M_0[\mathsf{pub}](x)$ and $M_1[\mathsf{pub}](x)$ output $P_1[\mathsf{pub}](x)$. Hence, one can prove in PV that the two machines are equivalent for $|x| \le i$.
- Similarly, for all x of length at least $i+2$, R certifies that there exists $P_0[\mathsf{pub}]$ such that $M_0[\mathsf{pub}](x)$ and $M_1[\mathsf{pub}](x)$ output $P_1[\mathsf{pub}](x)$. Hence, one can prove in PV that they are also equivalent in this case.
- Finally, if $|x| = i+1$, $M_0[\mathsf{pub}](x) = P_0[\mathsf{pub}](x)$ and $M_1[\mathsf{pub}](x) = P_1[\mathsf{pub}](x)$. Moreover, R certifies that there exists an $\mathcal{EF}$ proof (given by $\mathcal{M}_\Pi$) that $P_0[\mathsf{pub}](x) = P_1[\mathsf{pub}](x)$. As Cook showed, PV can be used to prove the consistency of $\mathcal{EF}$, as shown by Cook [20]. Therefore, one can prove in PV that they are also equivalent in this case.

Therefore, this demonstrates a PV proof of (1).

We are now ready to show security of our construction for $M_0 = P^{(i)}$ and $M_1 = P^{(i+1)}$.

$\boxed{\mathsf{H}_1}$ Output $\mathcal{O}(U_{M_0,\mathsf{hk},d_{\mathsf{pub}}})$ where $U_{M_0,\mathsf{hk},d_{\mathsf{pub}}}$ (padded such that the size and runtimes accomodate the proof) is the following machine which on input pub', x:

- If $d_{\mathsf{pub}} \ne \mathsf{Hash}(\mathsf{hk}, \mathsf{pub}')$, output $\perp$.
- Else, output $M_0[\mathsf{pub}'](x)$.

$\boxed{\mathsf{H}_2}$ Output $\mathcal{O}(Q_1)$ where the TM Q_1 is defined as follows:

a) If $d_{\mathsf{pub}} \ne \mathsf{Hash}(\mathsf{hk}, \mathsf{pub}')$, output $\perp$.
b) Compute the computation tableau of R on input $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$.
c) Else, output $M_0[\mathsf{pub}'](x)$.

It is clear that $U_{M_0,k,d_{\mathsf{pub}}}$ and P_1 are PV-equivalent machines, and hence we can rely on $\mathcal{O}$ security.

In other words, Q_1 additionally has M_1, R and $\mathcal{M}_\Pi$ hardcoded in it.

At this point, suppose for a moment that we are able to show that H_2 is indistinguishable from the following program where we additionally check that the output of R is 1 (e.g. as in H_3), then we can hope to invoke the PV-equivalence of $M_0[\mathsf{pub}](\cdot)$ and $M_1[\mathsf{pub}](\cdot)$. Concretely, we can argue that the following hybrids are indistinguishable:

$\boxed{\mathsf{H}_3}$ Output $(\mathsf{pub}, \mathcal{O}(Q_2))$ where the TM Q_2 is defined as follows:

a) If $d_{\mathsf{pub}} \neq \mathsf{Hash}(\mathsf{hk}, \mathsf{pub}')$, output $\perp$.
b) Compute the computation tableau of R on input $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$.
c) ... (additional checks that we are ignoring for now)
d) If the output of R in the computation tableau is 0, output $\perp$.
e) Else, output $M_0[\mathsf{pub}'](x)$.

$\boxed{\mathsf{H}_4}$ Output $(\mathsf{pub}, \mathcal{O}(Q_3))$ where the TM Q_3 is defined as follows:

a) If $d_{\mathsf{pub}} \neq \mathsf{Hash}(\mathsf{hk}, \mathsf{pub}')$, output $\perp$.
b) Compute the computation tableau of R on input $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$.
c) ... (additional checks that we are ignoring for now)
d) If the output of R in the computation tableau is 0, output $\perp$.
e) Else, output $M_1[\mathsf{pub}'](x)$.

Here, we can argue $\mathcal{EF}$-equivalence of the machines Q_2 and Q_3 using the fact that the last line of Q_2 and Q_3 is only reached if $R(M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi) = 1$. Then, we use (1) to prove in PV that $M_0[\mathsf{pub}'](x) = M_1[\mathsf{pub}'](x)$. Then, we can invoke Cook's translation to construct a machine $\mathcal{M}'_\Pi$ of size $\rho' = O(1)$ and polynomial T' demonstrating that Q_2 and Q_3 are (ρ', T)-$\mathcal{EF}$-equivalent. Therefore, the two hybrids are indistinguishable via $\mathcal{O}$ security with parameters ρ' and T. After reaching H_4, we can argue the hybrids essentially in reverse to argue that the above hybrid is indistinguishable from $(\mathsf{pub}, \mathcal{O}(U_{M_1,\mathsf{hk},d_{\mathsf{pub}}}))$, completing the proof.

Therefore, it suffices to argue that H_2 is indistinguishable from a machine of the form specified in H_3. However, as discussed earlier, these are not even functionally equivalent (let alone PV-equivalent) programs! This is the main technical challenge in this proof, and as alluded to earlier, we will use "crypto hybrids" interspersed with "PV hybrids".

Informally, our goal is as follows: (cryptographically) prove that given a *digest* d_{pub} of pub, the output of R in the obfuscated machine coincides with the output of $R(M_0, M_1, \mathsf{pub}, \mathcal{M}_\Pi)$. To argue this, we will draw inspiration from the security analyses of succinct randomized encoding constructions [3], [9], [16], [31], [51].

Somewhere statistically binding hash. We will rely on a special type of collision-resistant hash family known as a somewhere statistically binding hash (SSB) [39] (see Section IV-F for full details). A somewhere statistically binding hash family has a key with two computationally indistinguishable modes: (i) In the normal mode, the key is uniformly random and generated from SSB.Gen; and (ii) in the trapdoor mode, the key is generated using a trapdoor generation algorithm $\mathsf{SSB.TGen}$ according to on an index i of the hashed message. In the trapdoor mode, one can use the trapdoor td to extract the message on i. Note that this extraction property implies a statistical binding property for the coordinates in S. SSB functions have been shown to be very useful in the construction of RAM delegation protocols [18], [48], [49], and have also been shown to be "IO friendly" [31], [39].

Intuitively, the reason SSB hashing pairs well with IO is the *statistical* binding property. Here's an example of an argument one might make. Consider program P with a hash key hk and digest d_{pub} hardcoded in it, and accepts a string x if $\mathsf{Hash}(\mathsf{hk}, x) = d_{\mathsf{pub}}$. Then, given $\mathcal{O}(P)$, we might want to reason as follows:

1) Switch hk to be binding on index i. This follows from the index-hiding property of the SSB.
2) Change the obfuscated program P to only accept bit b at index i. This follows from the statistical-binding property as well as $\mathcal{O}$ security.
3) Switch SSB to be binding on some other index i', once again relying on the index-hiding property of SSB.

To make the above argument, it was crucial that dig was *statistically* binding rather than computationally binding to assert that the program was functionally equivalent.

"PV-friendly SSB". Since we would like to rely on pv-IO security instead of IO security, we can no longer argue that (2) holds. Therefore, we will need the following additional properties from SSB to make it "PV friendly".

- **PV proof of binding:** Since we will be pairing this SSB binding property with $\mathcal{O}$ security, we will need a PV proof of the binding property of the hash family, i.e.:

$$\vdash_{\mathsf{PV}} \left(\begin{array}{c} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SSB.TGen}(i; r) \\ \wedge\ h \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, x) \\ \wedge\ y = \mathsf{SSB.Ext}(\mathsf{hk}, \mathsf{td}, h) \end{array} \right) \rightarrow x[i] = y.$$

In words, it says that if a hash key hk and a trapdoor td were generated with respect to index i, if h is hash of a string x, then $x[i] = \mathsf{SSB.Ext}(\mathsf{hk}, \mathsf{td}, h)$. As we show in the full version of the paper, the construction of [25] via LWE has such a PV proof of statistical binding[8]. Given this guarantee, it is now clear how one might try to argue security in (2) - prove that hk and td are generated honestly (e.g. using the randomness r), and then use the PV proof of binding to argue PV-equivalence.

- **Hash-tree structure:** Additionally, we will need a SSB hash family that are constructed via a hash-tree using two-to-one SSB (Two-SSB) hash (following [48], [59]). At a high level, a Two-SSB hash takes as input two blocks of size s, and outputs a hash of size $s + \mathsf{poly}(\lambda, \log s)$ (i.e. is rate-1). This hash can be made binding on either of the two blocks. One can then use such a rate-1 two-to-one hash to compute a hash on a string x via a hash-tree, somewhat akin to a Merkle hash. If we implement such a hash-tree with a Two-SSB hash at each internal node, the hash value of each node binds one of its two children. This hash-tree structure will be crucial in our proof.

Back to our proof. With PV-friendly SSB hash families in hand, we now describe the intermediate hybrids between H_2 and H_3. Essentially, we will augment each configuration $\mathsf{cf}_i^{\mathsf{pub}'}$ of the tableau of R with a hash tree computed via a two-to-one SSB. We then argue inductively what the values of the internal nodes of the hash tree should be.

Checking the hash of $\mathsf{cf}_1^{\mathsf{pub}'}$. We change the machine Q_2 in H_2 as follows.

$\boxed{\mathsf{H}_{2,1}}$ First, sample a new hash key hk' for the obfuscated program. Compute configuration $\mathsf{cf}_1^{\mathsf{pub}}$ of the starting configuration of the tableau of R on the input M_0, M_1, pub and $\mathcal{M}_\Pi$ (recall that the program is computing the same tableau on $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$, where pub' may not be equal to pub). Compute $\phi_1 = \mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}})$. We now compute the program $Q_{2,1,\phi_1}$ as follows:

a) If $d_{\mathsf{pub}} \neq \mathsf{SSB.Hash}(\mathsf{hk}, \mathsf{pub}')$, output $\perp$.
b) Compute the computation tableau of R on input $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$.
c) Check that $\mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}'}) = \phi_1$, and abort if does not hold.
d) Else, output $M_0[\mathsf{pub}'](x)$.

Clearly, H_2 is not functionally equivalent to $\mathsf{H}_{2,1}$, since SSB.Hash is compressing. We now describe the hybrid in detail to illustrate how to use both "PV" hybrids and "Crypto" hybrids.

Recall that both the hash $\mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}'})$ and $\phi_1 = \mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}})$ are constructed via hash-trees. We inductively argue that the internal nodes of the hash-tree are consistent between the two computations.

- *Add check to leaf nodes.* When computing $\mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}'})$, recall that the bits of $\mathsf{cf}_1^{\mathsf{pub}'}$ either correspond to the hard-coded values of of M_0, M_1 or $\mathcal{M}_\Pi$, or the input pub'. It is easy to argue that the hard-coded values are consistent. When checking the leaf corresponding to the ith bit of pub', we argue in a few steps:
  - "Crypto hybrid": Switch hk to be binding on the ith bit of pub. This can be done by the *index-hiding* property of the hash.
  - "PV hybrid": Now, add the following check to the program: if $\mathsf{pub}'[i] \neq \mathsf{pub}[i]$, output $\perp$. We can now argue using the PV-proof of statistical binding of the SSB hash to argue that if $\mathsf{SSB.Hash}(\mathsf{hk}, \mathsf{pub}') = h$ and $\mathsf{SSB.Ext}(\mathsf{td}, d_{\mathsf{pub}}) = b$, then $\mathsf{pub}'[i] = b$. Therefore, adding this check is PV-equivalent (and hence $\mathcal{EF}$-equivalent).
- *Add check to an internal node.* Suppose that there is an internal node v of the tree, whose children are checked against the hash values φ_L and φ_R. Now, introduce a check that v is labeled by $\varphi = \mathsf{Two\text{-}SSB.Hash}(\mathsf{hk}', (\varphi_L, \varphi_R))$. This is clearly PV-equivalent (and hence $\mathcal{EF}$-equivalent) by definition of the tree-based hash.

We can use this strategy to inductively argue that the hash-tree must be consistent with the honest hash-tree. However, the above argument, as is, will introduce $\mathsf{poly}(|\mathsf{cf}_1^{\mathsf{pub}'}|)$ hard-coded checks in the program, resulting in a program that is not succinct in pub. Therefore, we need hybrids where we "forget" some checks while maintaining indistinguishability.

- *Forgetting checks on children nodes.* Suppose a node v, and its children u_L and u_R are checked against the hash values ϕ, ϕ_L and ϕ_R respectively. We argue as follows:

[8]This construction has a $\mathsf{negl}(\lambda)$ of outputting $\perp$. However, it does not output $\perp$, there is a PV proof that the statistical binding property holds. This weaker notion will be sufficient for us.

- Set ϕ to be extracting on the left child. This follows from the index-hiding property.
- Now, we can show that by PV-proof of statistical binding, if the check against ϕ_L fails, then the check against ϕ must also fail. Therefore, removing the check on the left child against ϕ_L is PV-equivalent (and hence $\mathcal{EF}$-equivalent), and we can invoke $\mathcal{O}$ security.
- Set ϕ to be binding on the right child. This follows from the index-hiding property[9].
- Once again, by we can argue PV-equivalence and remove the check against ϕ_R.

By inducting over the hash-tree carefully, we can ensure that we hard-code at most $O(|\log \mathsf{cf}_1^{\mathsf{pub}'}|)$ checks, therefore maintaining succinctness of the obfuscated program.

Checking the hash of $\mathsf{cf}_i^{\mathsf{pub}'}$. Now, suppose the hash of $\mathsf{cf}_i^{\mathsf{pub}'}$ is checked against the true hash of $\mathsf{cf}_i^{\mathsf{pub}}$, where $\mathsf{cf}_i^{\mathsf{pub}}$ is ith configuration of the tableau obtained from running R on input $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$. Then, one can show inductively on the hash-tree of $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ that the hash must be consistent with the hash of $\mathsf{cf}_{i+1}^{\mathsf{pub}}$ by arguing just as in the case of $\mathsf{cf}_1^{\mathsf{pub}'}$. To argue consistency of the leaf nodes, we use the fact that $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ is a deterministic and local function of $\mathsf{cf}_i^{\mathsf{pub}'}$, so we can verify the leaf nodes of the hash-tree on $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ by extracting only $O(1)$ locations of $\mathsf{cf}_i^{\mathsf{pub}'}$.

Checking the last hash. By iterating this argument, we can reach the following hybrid:

$\boxed{\mathsf{H}_{2,T}}$ Compute configurations $\mathsf{cf}_1^{\mathsf{pub}}$ and $\mathsf{cf}_T^{\mathsf{pub}}$ of the starting configuration and ending configuration of the tableau of R on the input M_0, M_1, pub and $\mathcal{M}_\Pi$. Compute $\phi_1 = \mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}})$ and $\phi_T = \mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_T^{\mathsf{pub}})$. Construct the program $Q_{2,1,\phi_1,T,\phi_T}$:

a) If $d_{\mathsf{pub}} \neq \mathsf{SSB.Hash}(\mathsf{hk}, x)$, output $\perp$.
b) Compute the computation tableau of R on input $M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi$.
c) Check that $\mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_1^{\mathsf{pub}'}) = \phi_1$, and abort if does not hold.
d) ...
e) Check that $\mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_T^{\mathsf{pub}'}) = \phi_T$, and abort if does not hold.
f) Else, output $M_0[\mathsf{pub}'](x)$.

At this point, we argue how to reach H_3 as follows. First, we set hk' to bind ϕ_T on the position of $\mathsf{cf}_T^{\mathsf{pub}}$ that corresponds to the output wire of R. This can be done by the index-hiding property of the SSB hash key.

Since $R(M_0, M_1, \mathsf{pub}, \mathcal{M}_\Pi) = 1$, we have that $\mathsf{SSB.Ext}(\mathsf{td}, \phi_T) = 1$. Therefore, using the fact that the statistical binding property of the SSB hash has a PV proof, we can show that:

$$\begin{pmatrix} (\mathsf{hk}', \mathsf{td}) \leftarrow \mathsf{SSB.TGen}(1^\lambda, 1^\ell, i) \\ \wedge\ h \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, \mathsf{cf}_T^{\mathsf{pub}'}) \\ \wedge\ 1 = \mathsf{SSB.Ext}(\mathsf{hk}', \mathsf{td}, h) \end{pmatrix} \vdash_{\mathsf{PV}} \mathsf{cf}_T^{\mathsf{pub}'}[i] = 1.$$

In particular, we can prove that $\mathsf{SSB.Hash}(\mathsf{hk}', \mathsf{cf}_T^{\mathsf{pub}'}) = \phi_T \rightarrow R(M_0, M_1, \mathsf{pub}', \mathcal{M}_\Pi) = 1$. Therefore, $\mathsf{H}_{2,T}$ (after switching where hk' is binding) is PV-equivalent to H_3. This completes the outline of the proof.

Ordering the hybrids via a pebbling game. While we have sketched the intuition behind the hybrids in the proof, there is a problem in that we cannot afford to hard-code the hash-value of *every* layer of the tableau of R, as that will cause the obfuscation to grow with size $O(T)$, and hence pub.

On the other hand, it is not clear that one can stop hard-coding the check on $\mathsf{cf}_i^{\mathsf{pub}'}$ after introducing the check at layer $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ while remaining indistinguishable. In particular, it was crucial in our proof that the configuration $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ is uniquely determined by the configuration $\mathsf{cf}_i^{\mathsf{pub}'}$. However, $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ does not necessarily deterministically define $\mathsf{cf}_i^{\mathsf{pub}'}$. Therefore, our template only allows us to argue hybrids in one of two ways:

- *Introduce* the check $\mathsf{Hash}(\mathsf{hk}', \mathsf{cf}_{i+1}^{\mathsf{pub}'}) = \phi_{i+1}$ if the program is already checking $\mathsf{Hash}(\mathsf{hk}', \mathsf{cf}_i^{\mathsf{pub}'}) = \phi_i$.
- *Remove* the check $\mathsf{Hash}(\mathsf{hk}', \mathsf{cf}_{i+1}^{\mathsf{pub}'}) = \phi_{i+1}$ if the program is already checking $\mathsf{Hash}(\mathsf{hk}', \mathsf{cf}_i^{\mathsf{pub}'}) = \phi_i$. (While we did not explicitly argue this can be done, we can just reverse the same argument that was used to introduce the check.)

How can we hard-code the $\mathsf{Hash}(\mathsf{hk}', \mathsf{cf}_T^{\mathsf{pub}'}) = \phi_T$ while introducing as few intermediate checks as possible? To remedy this, we take inspiration from succinct garbling literature [30], [31] which use the so-called "pebbling game" [7] to define the sequence of hybrids[10] . Turns out, the optimal strategy for the pebbling game gives us a sequence of poly(T) hybrids where we have to hard-code at most $O(\log T)$ checks. Therefore, this allows us to argue the indistinguishability of the programs over

[9]We note that although the previous hybrid used the trapdoor of the SSB in the security analysis, it is not used in the construction. Therefore, we may invoke index-hiding.

[10]An alternative view of this is that one can first make the computation of R reversible using the pebbling argument [7], at which point we can argue that $\mathsf{cf}_{i+1}^{\mathsf{pub}'}$ deterministically defines $\mathsf{cf}_i^{\mathsf{pub}'}$.

$\mathsf{poly}(T)$ hybrids while maintaining the program size at $\mathsf{poly}(\lambda, |M|, |s|, \log T, \log |\mathsf{pub}|)$.

B. Bootstrapping to all TMs

In this section, we show how to bootstrap fully succinct pv-IO (resp. input-succinct pv-IO) to fully succinct IO (resp. input-succinct IO). We first describe the proof of Theorem II.2 based on fully succinct WE (resp. witness-succinct WE) and then describe the proof based on succinct SNARGs.

Weak IO from Witness Encryption. We first show that WE implies a (significantly) weaker notion of IO that we refer to as *weak* IO. Informally speaking, weak IO is defined by changing the order of quantifiers in the definition of IO. Specifically, instead of requiring an obfuscator algorithm that works for *all* pairs of functionally equivalent programs, we only require the obfuscator to work for a *specific pair* of programs (say) M_0, M_1, and public input pub. More formally, a weak IO obfuscation scheme consists of two algorithms $(\mathcal{O}, \mathcal{E})$:

- The obfuscator $\mathcal{O}$ takes as input a pair of machines (M_0, M_1), public input pub and a bit b, and outputs an obfuscation of M_b.
- The evaluation algorithm $\mathcal{E}$ also takes as input a pair of machines (M_0, M_1) and public input pub, and an obfuscation of M_b, and uses them to compute $M_b[\mathsf{pub}](x)$ for any x.

Correctness and security are defined similarly to standard IO: Obfuscations of M_0 and M_1 must be computationally indistinguishable if M_0 and M_1 are functionally equivalent. Correctness, on the other hand, must always hold regardless of whether or not M_0 and M_1 are functionally equivalent.

Let us now explain how we build weak IO from WE. Consider the following language

$$L = \left\{ (M_0, M_1, \mathsf{pub}, n, T) \;\middle|\; \begin{array}{c} \exists x, |x| \le n, \\ M_0[\mathsf{pub}] \text{ and } M_1[\mathsf{pub}] \\ \text{abort on } x \text{ in } T \text{ steps}, \\ M_0[\mathsf{pub}](x) \neq M_1[\mathsf{pub}](x) \end{array} \right\},$$

and let $\mathcal{R}_L$ be the corresponding relation. Consider two Turing machines $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ which run in time $T(n)$ on inputs of length n. Let $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ be a witness encryption scheme for the language L.

To obfuscate M_b, we compute the ciphertext $\mathsf{ct} = \mathsf{Enc}((M_0, M_1, \mathsf{pub}, n, T), b)$ that encrypts the bit b under the instance $(M_0, M_1, \mathsf{pub}, n, T)$ of the language L. The obfuscated program simply consists of this ciphertext ct. To evaluate this obfuscated program on an input x, the evaluator first computes $z_0 = M_0[\mathsf{pub}](x)$ and $z_1 = M_1[\mathsf{pub}](x)$. It then proceeds as follows:

- If $z_0 = z_1$, it simply outputs z_0.
- Else, it uses x as a witness to decrypt the ciphertext ct and recover the bit b. It outputs z_b.

Correctness of this obfuscation scheme follows from inspection and by appealing to the correctness of the WE scheme. Security, on the other hand, follows from the semantic security of the WE scheme. Specifically, when $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ are functionally equivalent, the statement $(M_0, M_1, \mathsf{pub}, n, T)$ is *false*. Therefore, we have that

$$\begin{aligned} &\mathsf{Enc}((M_0, M_1, \mathsf{pub}, n, T), 0) \\ \approx_c\; &\mathsf{Enc}((M_0, M_1, \mathsf{pub}, n, T), 1) \end{aligned}$$

are computationally indistinguishable.

Finally, a remark on the *succinctness* of this scheme. Suppose that the WE scheme is fully succinct (resp. witness-succinct). Then, the weak IO scheme constructed above is fully succinct (resp. input-succinct).

IO from Weak IO. We are now ready to describe our construction of fully succinct IO. We will rely on two ingredients:

- Our new fully succinct pv-IO scheme.
- A fully succinct weak IO scheme whose correctness can be proven in theory PV.

We first establish some notation. Let $(\mathcal{O}, \mathcal{E})$ denote a fully succinct weak IO scheme with a PV proof of correctness. For any pair of machines (M_0, M_1), we use $\widetilde{M_b}$ to denote the obfuscation of M_b computed by the obfuscator $\mathcal{O}$ using some randomness s. That is:

$$\widetilde{M_b} \leftarrow \mathcal{O}(M_0, M_1, \mathsf{pub}, b; s).$$

Let $\mathcal{E}_{[M_0, M_1, \widetilde{M_b}]}$ denote the modified evaluation algorithm of the weak IO scheme that contains the machines M_0, M_1 and the obfuscation $\widetilde{M_b}$ hardwired in its description. On input x, it evaluates the obfuscation to learn $M_b(x)$.

We will now show that our fully succinct pv-IO scheme is indeed a fully succinct IO scheme for all programs. Our goal is to show that for all functionally equivalent Turing machines $M_0[\mathsf{pub}], M_1[\mathsf{pub}]$, $\mathcal{O}(M_0, \mathsf{pub})$ is computationally indistinguishable from $\mathcal{O}(M_1, \mathsf{pub})$. We consider the following sequence of hybrids:

$$\begin{aligned} \mathcal{O}(M_0, \mathsf{pub}) &\approx_c \mathcal{O}(\mathcal{E}_{[M_0, M_1, \widetilde{M_0}]}, \mathsf{pub}) \\ &\text{(by PV proof of correctness using randomness } s) \\ &\approx_c \mathcal{O}(\mathcal{E}_{[M_0, M_1, \widetilde{M_1}]}, \mathsf{pub}) \\ &\text{(by } \mathcal{O} \text{ indistinguishability, since } s \text{ is hidden)} \\ &\approx_c \mathcal{O}(M_1, \mathsf{pub}). \\ &\text{(by PV proof of correctness using randomness } s) \end{aligned}$$

Note that in the above, we use the fact that if $(\mathcal{O}, \mathcal{E})$ has a PV proof of correctness, then,

$$\vdash_{PV} \quad M_b[\mathsf{pub}](x) = \mathcal{E}_{[M_0, M_1, \widetilde{M_b}]}(\mathsf{pub}, x),$$

where x is a free variable.

As mentioned above, the PV proof of correctness depends on the randomness s used by the obfuscator. What if the size of the randomness s grows with the input size? In this case, the size of the PV proof, and thereby, the size of $\mathcal{O}$ would also grow with the input size, and the resulting scheme would *not* be input succinct.

To remedy this, we generate the randomness for $\mathcal{O}$ via a pseudorandom generator (PRG) with arbitrary polynomial stretch. This yields a fully succinct construction. The security proof proceeds in a similar manner as above, except that we now have two additional (symmetric) hybrid steps which rely upon the security of the PRG. These hybrids are invoked immediately before and after the intermediate hybrid which relies upon the security of weak IO.

Equivalence to Unique-SNARGs. To obtain the equivalence to SNARGs with unique proofs, we follow the work of Sahai and Waters [62], and the work of Chakraborty, Prabhakaran and Wichs [17]. Let n be the input-size and m be the witness-size. Recall that a SNARG is witness-succinct if the CRS grows with $\mathsf{poly}(\lambda, n, \log m)$, and fully succinct if $\mathsf{poly}(\lambda, \log n, \log m)$.

To see that witness-succinct (resp. fully succinct) SNARGs imply WE, we rely on the work of [17]. This work shows that unique witness maps (UWM) can be used to construct witness encryption. In short, a UWM deterministically maps all witnesses for an instance x to a single representative (cryptographic) witness π. On the other hand, for $x \notin L$, it is hard to find an valid proof π. Clearly, unique SNARGs imply UWMs, where the proof π is succinct. We observe that the [17] transformation from UWM witness-encryption is in fact witness-succinct if we start with a witness-succinct SNARG. In fact, if we started with a *fully succinct* SNARG, the resulting witness encryption is also full-succinct. Moreover, if the unique-SNARG (or UWM) has a PV-proof of uniqueness, i.e.

$$\vdash_{PV} \begin{pmatrix} R(x, w_0) = 1 \ \wedge \ R(x, w_1) = 1 \\ \wedge \ \pi_0 \leftarrow \mathsf{SNARG.Prove}(\mathsf{crs}, x, w_0) \\ \wedge \ \pi_1 \leftarrow \mathsf{SNARG.Prove}(\mathsf{crs}, x, w_1) \end{pmatrix} \rightarrow \pi_0 = \pi_1.$$

then, the resulting witness encryption has a PV-proof of correctness.

To go from input-succinct IO to witness-succinct (resp. fully succinct) SNARG, we rely on the work of Sahai and Waters (SW) [62] which showed how to use IO to construct a SNARG with unique proofs[11]. We observe that if we use an input-succinct IO scheme, their construction additionally achieves a witness-succinct crs. This therefore gives us three-way equivalence between these primitives in the input-succinct/witness-succinct settings.

However, using fully succinct IO in place of input-succinct IO in the SW construction does not seem to yield fully succinct SNARG. The issue is the following: in one of the hybrids in the proof of non-adaptive soundness in [62], we have to hard-code in the program the statement x^* and a PRF key $k\{x^*\}$ punctured at x^*. The size of this hard-coded information grows at least with the size of x^*. This, in turn, means that even the program in the real scheme needs to be padded by this amount. Therefore, the crs size still grows with $\mathsf{poly}(\lambda, n)$ and is therefore not fully succinct.

Instead, we take inspiration from the work of [45]. Recall that in the SW SNARG, a proof for a statement x^* was simply a PRF evaluation $f_K(x^*)$. The twist proposed by [45] is the following: instead of outputting $f_K(x)$, output $f_K(\mathsf{Hash}_{\mathsf{hk}}(x^*))$ where $\mathsf{Hash}_{\mathsf{hk}}$ is a hash function. Then, one could hope that in the security proof, one would only need to hardcode $h^* = \mathsf{Hash}_{\mathsf{hk}}(x^*)$, which is much smaller than x^*.

However, this seems problematic - what if there is a different $x' \in \mathcal{L}$ such that $\mathsf{Hash}_{\mathsf{hk}}(x') = \mathsf{Hash}_{\mathsf{hk}}(x^*)$? It therefore seems tricky to argue security. Luckily, we already tackled a similar problem in our construction of fully succinct pv-IO - we hardcoded a hash of the public input pub, and argued security. Therefore, by using our fully succinct pv-IO construction in a white-box way, one can show the security of this variant of the SW SNARG.

Therefore, this shows us a three-way equivalence between IO, WE and somewhere-sound almost fully succinct unique-SNARGs.

C. Overcoming the Representation-Size Barrier in Secret Sharing

Recall that in secret sharing, the access structure is *public* information. The goal is to minimize the total share size given the description of the access structure. As discussed in [4], for many access structures, it is not known how to capitalize on the fact that f is public to obtain share sizes smaller than $|f|$, even in the computational setting. This is referred to as the "representation-size barrier". While [4] show how to overcome this barrier in some special cases such as

[11]We note that they present their construction as a non-interactive zero-knowledge (NIZK) proof system, but it is also a SNARG.

monotone CNF formulas, it is not known how to address this challenge in the case of general access structures.

Construction template from KNY. Our starting point is the secret sharing for NP template of Komargodski, Naor and Yogev [50] (KNY). The main ingredients in their construction are:

- A perfectly binding bit commitment scheme $(\mathsf{Com.Gen}, \mathsf{Com.Enc}, \mathsf{Com.Open})$.
- A witness encryption scheme $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$.

The KNY template is as follows:

Sharing algorithm: Given a monotone access structure $f : \{0,1\}^n \to \{0,1\}$ on n parties and a secret s, the dealer does the following,

- Set $\mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda)$, where pk is the public key
- Sample n commitments $c_i \leftarrow \mathsf{Com.Enc}(1; r_i)$ (i.e. committing to 1, with randomness r_i).
- Consider the following NP language $\mathcal{L} = \mathcal{L}_{\mathsf{pk},f}$ containing strings $(c_1, \dots, c_n) \in \mathcal{L}_{\mathsf{pk},f}$ for which:
  - there exists a subset $I \subseteq [n]$ for which there exists r_i such that $\mathsf{Com.Open}(\mathsf{pk}, c_i, r_i) = 1$,
  - and $f(I) = 1$.
- Let $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(R_\mathcal{L}, (c_1, \dots, c_n), s)$, where $R_\mathcal{L}$ is the relation TM corresponding to $\mathcal{L}$.
- Set the public information to be $\mathsf{crs} = ((c_1, \dots, c_n), \mathsf{pk}, \mathsf{ct})$, and each party is given r_i.

Reconstruction algorithm: A subset $I \subseteq [n]$ of paries use $\{r_i\}$ to decrypt ct and obtain s.

Completeness of the above scheme follows immediately from the completeness of the witness encryption. We now outline the proof analysis. Given an unauthorized subset $I \subseteq [n]$ such that $f(I) = 0$, KNY proved security as follows:

$\boxed{\mathsf{H}_1}$ The public parameters and shares are generated honestly, and the adversary is given $((c_1, \dots, c_n), \mathsf{pk}, \mathsf{ct})$ and $\{r_i\}_{i \in I}$.

$\boxed{\mathsf{H}_2}$ For all c_i for $i \notin I$, switch c_i to commitments of 0 instead. This is indistinguishable from the previous hybrid by the hiding property of the commitment scheme.

$\boxed{\mathsf{H}_3}$ At this point, one can change the witness encryption to an encryption of $1 - s$ by appealing to semantic security of the witness encryption scheme. Note that instance $(c_1, \dots, c_n) \notin \mathcal{L}_{\mathsf{pk},f}$ by the perfect binding property of the commitment scheme. To see this, note that the only $\{c_i\}_{i \in I}$ can be opened to 1 but since $f(I) = 0$. Therefore, we can assert $(c_1, \dots, c_n) \notin \mathcal{L}_{\mathsf{pk},f}$ by the monotonicity of f.

While this construction captures all monotone functions, even ones that are non-deterministic, the issue is that it is not *succinct*. Indeed, if one implements the witness encryption scheme naively, the the size of the witness encryption grows with the size of the relation circuit and the statement, i.e. $|\mathsf{ct}| = \mathsf{poly}(\lambda, n, |\mathsf{pk}|, |f|)$, i.e. grows with the size of f. Therefore, the size of the public parameters grows with f.

We now describe how we overcome this barrier, with one improvement at a time. We note that since the construction in Section VII directly implements all of these optimizations at once, we hope that this step-by-step exposition aids in understanding the intuition behind each optimization.

Improvement #1: Use fully succinct WE. Our first observation is that by using fully succinct witness encryption in place of usual witness encryption, one can immediately improve the efficiency of the scheme. Treating the statement $(c_1, \dots, c_n)$ and relation $\mathcal{R}$ as the witness encryption statement, and improve the length of $|\mathsf{ct}| \le \mathsf{poly}(\lambda)$, independent of n and $|f|$. To evaluate the witness encryption, a subset I of parties only needs the statements $(c_1, \dots, c_n)$, the relation $\mathcal{R}$ (which is described by pk and f), and $\{r_i\}_{i \in I}$. Therefore, the size of the public information is: $\sum_i |c_i| + |\mathsf{pk}| \le O(n \cdot \mathsf{poly}(\lambda))$. Therefore, instantiating the KNY template with fully succinct WE, we get individual share size $\mathsf{poly}(\lambda)$ (note that we can give ct to each party) with public share size $O(n \cdot \mathsf{poly}(\lambda))$.

Improvement #2: pv-IO is sufficient for "natural" monotone classes. While the previous improvement required *fully succinct witness encryption* (which requires strong assumptions), for many "natural" (deterministic) monotone classes, we show that witness encryption constructed via fully succinct pv-IO is in fact sufficient. For this analysis, we assume the following additional properties about the commitment scheme, and the class of monotone functions.

- The *binding property* of the commitment scheme can be proven in PV, i.e.

$$\begin{pmatrix} b \in \{0,1\}, \\ \mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda), \\ c \leftarrow \mathsf{Com.Enc}(\mathsf{pk}, b; r) \end{pmatrix}$$
$$\vdash_{\mathsf{PV}} \forall r', c \neq \mathsf{Com.Enc}(\mathsf{pk}, 1 - b; r').$$

  In words, the above asserts that any commitment of a bit $b \in \{0,1\}$ under key pk cannot be opened to bit $1 - b$. We note that the standard construction of commitments by Naor [58] does not have this property. Therefore, we instead rely on an alternative construction from public-key encryption with PV proof of correctness.

- The *monotonicity* of $f \in \mathcal{F}$ can be proven in PV, i.e.

$$\vdash_{\mathsf{PV}} \begin{array}{c} \forall I, J \subseteq [n], \\ (f(I) = 0 \wedge J \subseteq I) \rightarrow (f(J) = 0). \end{array}$$

  In words, we can show that if $f(I) = 0$, then for any subset $J \subseteq I$, we can also assert that $f(J) = 0$. We call any function class $\mathcal{F}$ for which we can prove this a "natural" monotone function class. As we will show, the class of all monotone circuits is an example of such a class.

It will be helpful to think about the witness encryption scheme explicitly in terms of pv-IO. For the statement $(c_1, \ldots, c_n)$ with respect to commitment public key pk and access structure f, the witness encryption is the pv-IO of the following program:

```
P[(c_1, ..., c_n, pk, f)](J, {r_j}_{j∈J}):
  • For all j ∈ J, check if c_j = Com.Enc(1; r_j).
    If any fail, abort.
  • If f(J) = 1, output s. Else, output ⊥.
```

With these additional properties, we now show the modified analysis. Hybrids H_1 and H_2 are as before. Then, we proceed as follows. (We note that the consecutive hybrids using pvIO security can be a collapsed hybrid. However, we will spell it out over several hybrids for clarity.)

$\boxed{\mathsf{H}_3}$ Sample a PRG seed sd, and sample $(r, r'_1, \ldots, r'_n) \leftarrow G(\mathsf{sd})$, sample $\mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda; r)$, and set $c_i = \mathsf{Com.Enc}(\mathsf{pk}, 1; r'_i)$. This follows from PRG security. Looking forward, we are compressing the randomness sd so that we can hardcode this in the pvIO in the next step hybrid.

$\boxed{\mathsf{H}_4}$ Hardcode sd in the obfuscated program, and modify the functionality of the programs as follows:

```
Q_sd[(c_1, ..., c_n, pk, f)](J, {r_j}_{j∈J}) :
  a) Compute (r, r'_1, ..., r'_n) ← G(sd).
  b) Sample pk' ← Com.Gen(1^λ; r). If pk ≠ pk', abort.
  c) If c_i = Com.Enc(pk, 1; r'_i), then add i to I.
     Else, if c_i ≠ Com.Enc(pk, 0; r'_i), abort.
  d) If f(I) ≠ 0, abort.
  e) For all j ∈ J, check if c_j = Com.Enc(1; r_j). If any fail, abort.
  f) If f(J) = 1, output s. Else, output ⊥.
```

The first and second lines clearly do not interfere with the functionality of the program by construction. For the third line, we note that $c_i \neq \mathsf{Com.Enc}(\mathsf{pk}, b; r'_i)$ for either $b = 0$ or $b = 1$. Therefore, by construction, the program does not abort at this step. Finally, the check that $f(I) = 0$ also passes by construction (the adversary must choose I such that $f(I) \neq 0$). Note that here, we use the fact that running f on I is a PV proof of the fact that $f(I) = 0$.

$\boxed{\mathsf{H}_5}$ At this point, we can switch the program to one that always outputs $\perp$ via a PV-proof of equivalence. We sketch the PV proof:

(i) If the input $J \subseteq I$, then since $f(I) = 0$, by the proof of *monotonicity* of f, we can conclude $f(J) = 0$.
(ii) Hence, if $J \subseteq I$, the program outputs $\perp$.
(iii) If $J \not\subseteq I$, there exists $j \in J$ such that $j \notin I$.
(iv) Recall that the last line of the program is only reached if $c_j = \mathsf{Com.Enc}(\mathsf{pk}, 0; r'_j)$.
(v) By the PV proof of binding of the commitment scheme, we know for all r_j, $c_j \neq \mathsf{Com.Enc}(\mathsf{pk}, 1; r_j)$. Hence, the program always aborts.
(vi) Therefore, if $J \not\subseteq I$, the program outputs $\perp$.
(vii) By the law of excluded middle, (ii) and (vi) imply that the program outputs $\perp$ for all inputs.

Therefore, we can invoke pv-IO security.

At this point, s is no longer in the adversary's view, therefore proving security of the scheme.

At a high level, we relied on the fact that given the short seed sd used to generate the commitments as well as the PV proof of monotonicity, one can argue that the statement $(c_1, \ldots, c_n, \mathsf{pk}, f)$ has a small "PV proof of non-membership", following the terminology of [41], [46].

Improvement #3: Improving public parameters to a URS of size n. In the previous construction, the public share was the list of commitments $(c_1, \ldots, c_n)$. We now show that we can improve the public parameters in two ways:

- The public share is a *uniformly* random string.
- The length of the public share is size n (as opposed to $O(n \cdot \mathsf{poly}(\lambda))$).

To achieve this, we first have to change the program so that it no longer takes all of the commitments $\{c_i\}_{i \in [n]}$ as public input, and instead, we compute an SSB hash of all the commitments $\mathsf{dig} \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, (c_1, \ldots, c_n))$, and hardcode dig in the obfuscated program. We then give each party c_i, a commitment opening r_i, along with an SSB local opening ρ_i so that $\mathsf{SSB.Ver}(\mathsf{hk}, \mathsf{dig}, i, c_i, \rho_i) = 1$.

To ensure that an adversary cannot feed "incorrect" commitments to the program, we rely on the fact that there exists a small program of size $n + \mathsf{poly}(\lambda)$ which outputs the commitments $\{c_i\}_{i \in [n]}$ if all the commitments are generated from a single, short PRG seed sd:

```
$\mathcal{M}(I, \mathsf{sd})$ :
  • Sample $(r, r_1, \ldots, r_n) \leftarrow G(\mathsf{sd})$.
  • Sample $\mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda; r)$.
  • For $i \in I$, sample $c_i \leftarrow \mathsf{Com.Enc}(\mathsf{pk}, 1; r_i)$.
  • For $i \notin I$, sample $c_i \leftarrow \mathsf{Com.Enc}(\mathsf{pk}, 0; r_i)$.
  • Output $\{c_i\}_{i \in [n]}$.
```

Therefore, in one of the hybrids, we can switch to computing c_i "on the fly" from $\mathcal{M}(I, \mathsf{sd})$, and rely on SSB security to ensure that the adversary's inputs are consistent with these values. To achieve rate-1 in this program size, we then rely on our padding techniques from Section III-D to pay for the "padding" in the form of a size n public random string.

Improvement #4: Reducing public share size below n. To reduce the size of the share below n, we rely on two facts:

- Our security analysis only relied on the fact that commitments of 0 cannot be switched to commitments of 1 to assert Step (iv) of the PV proofs. Therefore, we do not need to ensure that *every* commitment that the evaluator feeds is generated honestly, but only that commitments of zero are not flipped to commitments of ones.
- Suppose there exists $T \supseteq I$ such that $f(T) = 0$. Then, it suffices to ensure that commitments of $i \notin T$ correspond to commitments of 0. Such a T is called a *zero-certificate* of I.

In other words, we can use any set $T \supseteq I$ such that $f(T) = 0$ in our security analysis. If T has a small description, we can then use a program of size smaller than n (i.e. t-bounded conditional Kolmogorov complexity less than n) to generate the commitments corresponding to 0, and conduct the same security proof as before.

Example: Bipartite matching. As an example, we consider the bipartite matching access structure f on $n = m^2$ parties, where each party represents a unique edge in a bipartite graph on V_1 and V_2, each of size m. Then, $f(E) = 1$ if and only if E contains a perfect matching.

We now show that for any E such that $f(E) = 0$, there exists a "zero-certificate" $W \supseteq E$ such that $f(W) = 0$, and $f(W)$ can be described in $2m + O(1)$ bits. With this, one can then improve our secret sharing scheme in the setting of bipartite matching to have a public share of size $2m = 2\sqrt{n}$ rather than n.

Consider an E such that $f(E) = 0$. We now construct a new set $W \supseteq E$ such that $f(W) = 0$ which has a short description. By Hall's marriage theorem [36], we know that $f(E) = 0$ if and only if there exists some $S \subseteq V_1$ such that $|N_E(S)| < |S|$, where $N_E(S)$ denotes the set of all neighbours of S in the subgraph defined by E. Let W be the subgraph obtained by taking the whole bipartite graph and deleting all edges between S and $V_2 \setminus N_E(S)$. Clearly, $W \subseteq E$. Moreover, it is clear that $|N_W(S)| < |S|$ by construction, and hence W does not contain a perfect matching (i.e. $f(W) = 0$). Moreover, W can be described by S and $V_2 \setminus (S)$, which can be represented by a string of size $2m+O(1)$. This completes the claim.

D. Padding with Public Randomness and High Rate Obfuscation

In this section, we describe a general template to overcome the common "necessary padding" issue via a public random string. As a corollary, we show how to improve the rate of IO schemes for many classes of programs. For simplicity, we present the result here for programs M without a public part.

Our construction uses two main ingredients: a puncturable PRF family, and a fully succinct IO scheme iO. At a high-level, the algorithm takes as input a program M, a padding parameter 1^ρ and a public string URS, and first encrypts M using a "pseudorandom" rate-1 encryption via the puncturable PRF key k. It then uses URS to pad the encryption to length ρ. It then constructs a program $U_{\ell,k}$ which takes the padded string as public input and uses ℓ and k to parse the input and compute M.

```
$\mathsf{pad}\mathcal{O}(M, 1^\rho, \mathsf{URS})$:
  • Let $\ell = |M|$.
  • **Encrypt and pad:**
    – Sample $k \leftarrow \mathsf{PRF.Gen}(1^\lambda)$.
    – Let $\mathsf{ct} = \{M[i] \oplus \mathsf{PRF}_k(i)\}_{i \in [\ell]}$, i.e. use the
      PRF to compute a rate-1 encryption of the
      program.
    – Truncate URS to length $\rho - \ell$.
```

- Let $r = \mathsf{ct}||\mathsf{URS}$ be a string of length ρ (with no delimiter separating the two parts).
- **Obfuscate:** Construct the following program $U_{\ell,k}[r](x)$:
  - Parse $r = (\mathsf{ct}, \mathsf{URS})$ using the parameter ℓ, and ignore URS.
  - Decrypt ct using k to obtain M.
  - Obtain $M(x)$.
- Let $\Gamma = \mathsf{iO}(U_{\ell,k}, \mathsf{pub} = r)$, where iO is a fully succinct IO scheme.
- Output (r, Γ).

By construction, it is clear that the program that is output has length $\rho + \mathsf{poly}(\lambda)$, and that $\rho - \ell$ bits of the obfuscation correspond to a public random string of the obfuscator's choice.

Remark III.1. Note that if we instead relied on description-succinct IO (which is not input-succinct), we get an obfuscated program of size $\rho + \mathsf{poly}(\lambda, n)$, where n is a bound on the input-length. This gives a construction of pad$\mathcal{O}$ from iO for circuits and LWE (without making any non-standard assumptions such as the existence of succinct witness encryption). For most of this overview, we will instantiate iO as a fully succinct IO scheme.

For simplicity, we will omit URS from the syntax of pad$\mathcal{O}$ for the rest of this overview.

Security for adjacent and close programs. Our first observation is that for programs M_0 and M_1 that are "adjacent" (recall that this means that they are functionally-equivalent but differ in $\tau = O(1)$ bits), then $\mathsf{pad}\mathcal{O}(M_0, 1^\rho)$ is indistinguishable from $\mathsf{pad}\mathcal{O}(M_1, 1^\rho)$ for $\rho \geq \max(|M_0|, |M_1|)$. Our proof strategy is relatively simple: we will puncture the PRF key at the points that the two programs differ, and the hardcode the differing bits in the obfuscation.

We now outline the proof in more detail for the case where $\tau = 1$ and $|M_0| = |M_1|$ (we argue how to modify the proof when $|M_0| > |M_1|$ after the sketch). Suppose the two programs differ at index i. We then proceed in a few hybrids as follows:

$\boxed{\mathsf{H}_1}$ Output $(r, \Gamma) \leftarrow \mathsf{pad}\mathcal{O}(M_0, 1^\rho)$.

$\boxed{\mathsf{H}_2}$ In this hybrid, puncture the PRF key k at index i, and hardcode $b = M_0[i]$ in the obfuscated program $U_{\ell,k\{i\},i,b}$. It then uses ct and punctured key $k\{i\}$ to derive all the bits of M_0 other than the ith bit, and uses b at the ith bit. This follows from iO security since the functionality of the program has not changed.

$\boxed{\mathsf{H}_3}$ Switch the hardcoded bit b in the program to $M_1[i]$. This does not change the functionality of the program because $M_0 \equiv M_1$ and they differ only on index i.

$\boxed{\mathsf{H}_4}$ Switch the ith bit of ct to a uniformly random bit. Since k is hidden from the distinguisher's view (the distinguisher only sees $k\{i\}$), $\mathsf{PRF}_k(i)$ looks pseudorandom. Therefore, this change is indistinguishable.

$\boxed{\mathsf{H}_5}$ Switch the ith bit of ct to $\mathsf{PRF}_k(i) \oplus \boxed{M_1[i]}$. Once again, this follows from punctured key security.

$\boxed{\mathsf{H}_6}$ Unpuncture the key k, and change the obfuscated program to $U_{\ell,k}$ as in the pad$\mathcal{O}$ algorithm. This is indistinguishable from the previous hybrid by functionality equivalence of the obfuscated program. This hybrid is now identical to $\mathsf{pad}\mathcal{O}(M_1, 1^\rho)$.

This completes the proof. In the event that $|M_0| > |M_1|$, note that $i = \ell$ (they differ in the last bit). Then, we do not need to perform H_5, and we instead obfuscate $U_{\ell-1,k}$ in the last hybrid.

Security for ρ-close programs. Once we have that pad$\mathcal{O}$ is secure for adjacent programs, we can now show that it is secure for ρ-close programs M_0 and M_1. Recall that we call M_0 and M_1 ρ-close if there exists a $\mathsf{poly}(\lambda)$-sequence of adjacent machines starting with M_0' and ending with M_1', where M_b' is M_b padded to size ρ. Then, by a simple hybrid argument over the sequence of adjacent machines, we have that

$$\mathsf{pad}\mathcal{O}(M_0, 1^{\rho'}) \approx_c \mathsf{pad}\mathcal{O}(M_1, 1^{\rho'}).$$

for all $\rho' \geq \rho$. Moreover, by construction, we saw that $\rho' - |M|$ bits of the construction indeed correspond to a uniform public string.

Corollary: Rate-1 Null-IO. Recall that null-IO preserves functionality, but only guarantees indistinguishability for programs M_0 and M_1 which are identically zero on all inputs. With our general theorem in hand, it suffices to show that for any two programs M_0 and M_1, both of length ℓ which are identically zero on all inputs, M_0 and M_1 are $(\ell + O(1))$-close. To see this:

- Start from M_0.
- Add a line to the program that changes the output to 0 (this is an $O(1)$ change).
- Switch the bits of M_0 one at a time to bits of M_1.
- Change the output of the program to the output of M_1.

It is clear that the above programs are functionally equivalent, since all programs evaluate to 0. Moreover, it is easy to see that that the programs are $\ell + O(1)$-close. Therefore, we get that $\mathsf{pad}\mathcal{O}(M_0, 1^\rho) \approx_c \mathsf{pad}\mathcal{O}(M_0, 1^\rho)$

for $\rho = \ell + O(1)$, thereby giving us a rate-1 Null-IO scheme.

Corollary: High-rate batch obfuscation. Suppose we are given k programs $M_1, \ldots, M_k$ to obfuscate simultaneously, and the goal is to construct an obfuscation $\Gamma \leftarrow \mathcal{O}(M_1, \ldots, M_k)$ with the functionality that $\Gamma(i, x) = M_i(x)$ (this setting was considered by [2]). Additionally, we want the guarantee that $(M'_1, \ldots, M'_k)$ such that $M_i \equiv M'_i$, we have that

$$\mathcal{O}(M_1, \ldots, M_k) \approx_c \mathcal{O}(M'_1, \ldots, M'_k).$$

We claim that $(M_1, \ldots, M_k)$ and $(M'_1, \ldots, M'_k)$ are $(k+1) \cdot \ell + O(1)$-close.

- Start with the program $U_{M_1,\ldots,M_k}(i, x) = M_i(x)$.
- Add the description of M'_1 to the program (without changing the functionality of the program).
- On inputs of the form $(1, x)$, switch the output to $M'_1(x)$.
- Switch the hardcoded value M_1 to M'_1, one bit at a time.
- Switch back to using the first copy of M'_1.
- Erase the second copy of M'_1 from the program one bit at a time. The resulting program is now of the form $U_{M'_1,M_2,\ldots,M_k}$.

Repeating this argument for each $i \in [k]$, we have that the two programs are in fact ρ-close for $\rho \geq (k+1) \cdot \ell + O(1)$. Therefore, we can set the batch-obfuscation algorithm

$$\mathcal{O}(M_1, \ldots, M_k) = \mathsf{pad}\mathcal{O}((M_1, \ldots, M_k), 1^\rho)$$

for $\rho = (k+1) \cdot \ell + O(1)$, and guarantee security. Therefore, this gives us rate-$\frac{k}{k+1}$ batch obfuscation. Note that as a corollary, we obtain rate-1/2 IO.

Better rate for pv-IO. While the above claims crucially relied on IO for all TMs, we also show results in the setting of pv-IO. Consider the explicit construction of JJ22 of pv-IO ([41, Figure 11]). Their construction is roughly as follows: An obfuscation of M is an obfuscation of the program $\widehat{M}$ which does the following: on input (i, g),

- Compute the circuit C_i corresponding to running M on an input of size i.
- Compute a "gate-by-gate" circuit obfuscation of C_i.
- Output the obfuscation of gate g.

The program $\widehat{M}$ is padded size ρ to additionally contain a program $\mathcal{M}_\Pi$ that outputs a propositional proof corresponding to the PV proof Π. Other hybrids make small changes to the program (e.g. puncturing PRF keys, etc). Therefore, $\widehat{M_0}$ and $\widehat{M_1}$ are $\gamma + \mathsf{poly}(\lambda)$-close.

Our observation is that if we used our $\mathsf{pad}\mathcal{O}$ scheme (instantiated via description-succinct IO) in place of IO for circuits, we can pay for $\mathcal{M}_\Pi$ in rate-1! Therefore, this improves the JJ22 efficiency as follows: for families M_1 and M_2 with a PV proof of size at most γ, we have that

$$|\mathcal{O}(1^\lambda, 1^\gamma, M)| = \gamma + 2|M| + \mathsf{poly}(\lambda).$$

Propogating this through our construction of fully succinct IO and $\mathsf{pad}\mathcal{O}$, we can construct high-rate pv-IO. For example, there exists a null-IO scheme which is rate-1 in the program M and PV proof Π of the fact that $\forall x, M(x) = 0$.

IV. Preliminaries

Throughout, we will use λ to denote the security parameter.

- We say that a function $f(\lambda)$ is negligible in λ if $f(\lambda) = \lambda^{-\omega(1)}$, and we denote it by $f(\lambda) = \mathsf{negl}(\lambda)$.
- We say that a function $g(\lambda)$ is polynomial in λ if $g(\lambda) = p(\lambda)$ for some fixed polynomial p, and we denote it by $g(\lambda) = \mathsf{poly}(\lambda)$.

Notation. We will use the following notation through the work.

- For $n \in \mathbf{N}$, we use $[n]$ to denote $\{1, \ldots, n\}$.
- We use the notation $|x|$ or $\mathsf{desc}(x)$ to denote the bit-length of string x. We use $\mathsf{desc}(x)$ when $|x|$ has a different interpretation, e.g. when x is a description of a polynomial.
- If R is a random variable, then $r \leftarrow R$ denotes sampling r from R. If T is a set, then $i \leftarrow T$ denotes sampling i uniformly at random from T.
- We write $\mathsf{RT}(M, x)$ to denote the runtime of machine M on an input x. We say that the runtime of M is bounded by a polynomial $t(\cdot)$ if for all x, $\mathsf{RT}(M, x) \leq t(|x|)$.

A. *Extended Frege $\mathcal{EF}$*

In this work, we will use the extended Frege ($\mathcal{EF}$) system when discussing propositional logic. Such a system is described by a set of *variables*, a set of *connectives*, and a set of *inference rules*. *Variables* are the most basic elements, usually represented by letters such as x, y, z, and should be interpreted as taking on the value of either "true" or "false". We can then use standard Boolen connectives such as $\rightarrow, \neg, \wedge, \vee, \oplus$ (representing "imply", "negation", "and", "or" and "xor" respectively) to construct Boolean formulae. Finally, we define a finite set of *inference rules*. Each inference rule is defined as $A_1, A_2, \dots, A_k \vdash A_0$, where $A_0, A_1, \dots, A_k$ are formulas. Intuitively, it means that "if $A_1, A_2, \dots, A_k$ are valid, then A_0 is also valid". If $k = 0$, then we say such inference rule is an *axiom*. We use the following set of axioms and modus ponens as inference rules for propositional logic.

- **Axiom 1:** $p \to (q \to p)$
- **Axiom 2:** $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$
- **Axiom 3:** $\neg\neg p \to p$
- **Modus Ponens:** $p, p \to q \vdash q$

While the above description captures Frege systems, the *extended Frege system* additionally has the following *extension axiom*. Namely, for a derivation $(\theta_1, \theta_2, \dots, \theta_\ell)$ in $\mathcal{EF}$, for each $i \in [\ell]$, θ_i needs to satisfy the aforementioned constraint *or* θ_i is of the form $t \leftrightarrow A$, where A is a formula, and t is a new variable that has not been occurred in $\theta_1, \theta_2, \dots, \theta_{i-1}$, and also does not occur in A. We define the *size* of a proof $(\theta_1, \theta_2, \dots, \theta_\ell)$ as the summation of the sizes of the formulas $\theta_1, \theta_2, \dots, \theta_\ell$.

B. *Theory* PV

In this work, we will often use Cook's Theory PV to argue that a mathematical claim has a "uniform" $\mathcal{EF}$ proofs (Theorem IV.2).

Cook introduced theory PV [20] to capture the intuition of feasibly constructive proofs (i.e. polynomial-time reasoning). Its language consists of *function symbols* representing basic polynomial-time computable functions, and *terms*, which are syntactic expressions built by composing these function symbols with variables (formally, every variable is a term, and if f is a k-ary function symbol and $t_1, \dots, t_k$ are terms, then $f(t_1, \dots, t_k)$ is also a term). The formulas of PV are *equations* of the form $t_1 = t_2$ between terms, expressing that the corresponding polynomial-time computations produce the same output. Proofs in PV consist of equational derivations obtained from the defining axioms of the function symbols using the standard rules of equality (reflexivity, symmetry, transitivity, and substitution), and the theorems of PV capture identities that hold between polynomial-time computable functions. For more formal details on PV, we refer the reader to [41], [52].

We use the notation $\vdash_{\mathsf{PV}} t = u$ to denote that there exists a PV proof of the fact that $t = u$.

Remark IV.1. Note that if one can prove $\vdash_{\mathsf{PV}} t(x) = u(x)$ where t and u are function symbols and x is a free variable, then, one can then meta-mathematically interpret as a PV proof that *for all* x, $t(x) = u(x)$. For ease of readability and to be explicit about free variables, we sometimes write $\vdash_{\mathsf{PV}} \forall x, t(x) = u(x)$. Note, however that $\forall$ quantifier is not part of the formal syntax for PV.

Theory PV_1**.** In the same work, Cook also introduced theory PV_1 (we defer the details to Appendix A) which includes truth-functional combinations of equations, using "$\wedge, \vee, \neg, \to, \leftrightarrow$", which express "and", "or", "negation", "imply", and "equivalent". Moreover, Cook [20] showed that PV_1 is a conservative extension of PV. In other words, any theorem of the fact that "$t = u$" (where t and u are terms) in PV_1 can also be proven in PV. Therefore, we will primarily rely on PV_1 [20] since it makes formalizing proofs easier. We will not distinguish between PV and PV_1 explicitly.

Throughout this work, we will often write theorems of the form $\vdash_{\mathsf{PV}} s \to (t = u)$, or equivalently, $s \vdash_{\mathsf{PV}} (t = u)$. Although this claim is not an equation (and in fact, PV does not allow for $\to$), one can should interpret this claim instead as a PV_1 proof of the fact that $\vdash_{\mathsf{PV}} \mathsf{ITE}(s, t, u) = u$, where ITE is the function symbol corresponding to the if-then-else function:

$$\mathsf{ITE}(s, t, u) = \begin{cases} t & \text{if } s \text{ is true,} \\ u & \text{otherwise.} \end{cases} \tag{2}$$

As shown in [52], the function symbol ITE is in fact definable in PV. For brevity and ease of readability, we abuse notation and write $\vdash_{\mathsf{PV}} s \to (t = u)$ to mean $\vdash_{\mathsf{PV}} \mathsf{ITE}(s, t, u) = u$.

Propositional Translation. The main property about PV proofs that we will rely on is that any proof in PV can be translated to polynomial size $\mathcal{EF}$ proof [20], [22]. Let $t(x) = u(x)$ be a PV equation with PV proof Π. Then, let $[\![t = u]\!]_n$ denote the $\mathcal{EF}$ claim that

$$\forall x \in \{0,1\}^n, t(x) = u(x).$$

Theorem IV.2 (Corollary of ER Simulation Theorem in [20])**.** *Let $t(x)$ and $u(x)$ be terms in* PV*, and let Π be a* PV *proof that $t = u$. Then, there exists a machine $\mathcal{M}_\Pi$ and polynomial q such that on input 1^n, outputs a $\mathcal{EF}$ proof of $[\![t = u]\!]_n$ in $q(n)$ time.*

C. *Indistinguishability Obfuscation for Turing Machines*

In this section, we define indistinguishability obfuscation for Turing machines.

Syntax. An indistinguishability obfuscation scheme for Turing machines (TMs) consists of a probabilistic algorithm iO that takes as input a security parameter 1^λ, a program M, an input length bound n. It outputs an obfuscation $\widetilde{M}$.

Definition IV.3. An indistinguishability obfuscation scheme iO for TMs satisfies the following properties:

- **Correctness:** For every λ, $n \leq 2^\lambda$, machine M, and input x such that $|x| \leq n$ and $\mathsf{RT}(M, x) \leq 2^\lambda$, we have that
$$\Pr\left[\ \widetilde{M}(x) = M(x) \ : \ \widetilde{M} \leftarrow \mathsf{iO}(1^\lambda, M, n)\ \right] = 1 \ .$$
We say that the scheme is N-correct for a function $N(\lambda) \leq 2^\lambda$ if the above only holds for $n \leq N(\lambda)$.
- **Efficiency:** In the correctness experiment above:
  - The running time of iO is $\mathsf{poly}(\lambda, |M|, n)$.
  - There exists a polynomial q such that $\mathsf{RT}(\widetilde{M}, x) = q(\lambda, \mathsf{RT}(M, x))$.
  - We say that the scheme is *input-succinct* if the running time of iO is at most $\mathsf{poly}(\lambda, |M|)$.
- **Security:** For every poly-size $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every λ, M_0, M_1 and bound $n \leq p(\lambda)$ such that
  - $|M_0| = |M_1| \leq p(\lambda)$.
  - For every input x of length at most 2^λ, it holds that $M_0(x) = M_1(x)$ and $\mathsf{RT}(M_0, x) = \mathsf{RT}(M_1, x)$.

  we have that
$$\Pr\left[\ \mathcal{A}(\widetilde{M_b}) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\} \\ \widetilde{M_b} \leftarrow \mathsf{iO}(1^\lambda, M_b, n) \end{array}\ \right] \leq \frac{1}{2} + \mu(\lambda) \ .$$
We say the scheme is Γ-secure if the advantage above is at most $(\Gamma(\lambda))^{-O(1)}$ for all adversaries of size $\mathsf{poly}(\Gamma)$.

Remark IV.4 (Augmenting the definition for input succinct IO)**.** For IO schemes that are input-succinct, we can remove the bound n on the input size from the parameters of iO and instead fix $n = 2^\lambda$ (or $n = N(\lambda)$ if the scheme is only N-correct).

1) IO for uniform $\mathcal{EF}$-equivalent Machines: In this section, we recap the result on IO for TMs of Jain and Jin [41] (JJ). JJ constructs a candidate input-succinct IO scheme for TMs, and show security for all pairs of programs M_0 and M_1 for which $\vdash_{\mathsf{PV}} \forall x, M_0(x) = M_1(x)$. For convenience, in this work, we will instead follow the formulation of Ma, Dai and Shi [56] that consider instead pairs of programs M_0 and M_1 for which there exists a uniform $\mathcal{EF}$-proof of equivalence. The relation between PV proofs and uniform $\mathcal{EF}$ proofs is given by Theorem IV.2.

To state the theorem, we first introduce the following definition.

Definition IV.5 ($\mathcal{EF}$ equivalent machines)**.** Let ρ be a constant and T be a polynomial. We say that two TMs M_0 and $M_1$are (ρ, T)-$\mathcal{EF}$ equivalent if there exists a TM $\mathcal{M}_\Pi$ of size at most ρ which on input n, M_0, M_1 and outputs an $\mathcal{EF}$ proof of $[\![M_0(x) = M_1(x)]\!]_n$ in time $T(n)$.

Next we define $\mathcal{EF}$-security: a relation of the security in Definition IV.3 that holds only for pairs of $\mathcal{EF}$-equivalent programs instead of every pair of functionally equivalent programs.

Definition IV.6 ((ρ, T)-$\mathcal{EF}$ security)**.** Let $\rho(\lambda)$ and $T(\lambda, n)$ be polynomials. We say that an IO scheme satisfies (ρ, T)-$\mathcal{EF}$-security if for every λ, and polynomial-sized adversary $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every M_0 and M_1 satisfying:

- $|M_0| = |M_1| \leq p(\lambda)$,
- For every input x, $\mathsf{RT}(M_0, x) = \mathsf{RT}(M_1, x)$.
- M_0 and M_1 are $(\rho(\lambda), T(\lambda, \cdot))$-equivalent,

$$\Pr\left[\ \mathcal{A}(1^\lambda, \widetilde{M_b}) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\} \\ \widetilde{M_b} \leftarrow \mathsf{iO}(1^\lambda, M_b) \end{array}\ \right] \leq \frac{1}{2} + \mu(\lambda) \ .$$

We say the scheme is Γ-secure if the advantage above is at most $(\Gamma(\lambda))^{-O(1)}$ for all adversaries of size $\mathsf{poly}(\Gamma)$.

Definition IV.7. ($\mathcal{EF}$-IO) Let $N(\lambda) \leq 2^\lambda$ be a function. An N-correct $\mathcal{EF}$-IO scheme is given by an algorithm $\mathsf{iO}_{\rho,T}$ such that for every pair of polynomials $\rho(\lambda)$ and $T(\lambda, n)$, $\mathsf{iO}_{\rho,T}$ is an N-correct input-succinct IO scheme with (ρ, T)-security, satisfying the following efficiency: In the correctness experiment described in Definition IV.3:

- The running time of $\mathsf{iO}_{\rho,T}(1^\lambda, M)$ is $\mathsf{poly}(\lambda, |M|, \rho(\lambda))$.
- The exists a polynomial q such that $\mathsf{RT}(\widetilde{M}, x) = q(\lambda, \mathsf{RT}(M, x), \rho(\lambda), T(\lambda, |x|))$.

Theorem IV.8 ([41], [56])**.** *Let $N(\lambda) \leq 2^\lambda$ be a function. Assuming $\mathsf{poly}(N)$-hardness of LWE, sub-exponential hardness of one-way functions, and sub-exponential security of IO for circuits, for every $\rho \in \mathbb{N}$ and polynomial T, there exists a N-correct $\mathcal{EF}$-IO scheme.*

D. PRGs and Puncturable PRFs

Definition IV.9 (Pseudorandom Generators (PRGs))**.** Fix a polynomial p. A pseudorandom generator $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{p(\lambda)}$ is a function mapping λ bits to $p(\lambda) > \lambda$ bits such that for any polynomial-sized adversary $\mathcal{A}$,

$$\left| \Pr_{r\in\{0,1\}^\lambda}[\mathcal{A}(1^\lambda, G(r)) = 1] - \Pr_{z\in\{0,1\}^{p(\lambda)}}[\mathcal{A}(1^\lambda, z) = 1] \right| \leq \mathsf{negl}(\lambda).$$

Definition IV.10 (Puncturable PRF)**.** A puncturable PRF family on key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_\lambda$, domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_\lambda$, output space $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_\lambda$ and polynomial $s(\lambda)$ is a tuple of PPT algorithms $(\mathsf{PPRF.Gen}, \mathsf{PPRF.Punc}, \mathsf{PPRF.Eval})$ with the following interface and properties.

- $\mathsf{PPRF.Gen}(1^\lambda)$: On input the security parameter 1^λ, outputs a key $k \in \mathcal{K}_\lambda$.
- $\mathsf{PPRF.Punc}(k, S)$: On input a key $k \in \mathcal{K}_\lambda$ and a set $S \subseteq \mathcal{X}_\lambda$ such that $|S| \leq s(\lambda)$, outputs a new key $k(\{S\}) \in \mathcal{K}_\lambda$.
- $\mathsf{PPRF.Eval}(k, x)$: On input $k \in \mathcal{K}_\lambda$ and $x \in \mathcal{K}_\lambda$, outputs some value $y \in \mathcal{Y}_\lambda$.
- **Functionality preserving:** For all sets $S \subseteq \mathcal{X}$ of size $|S| \leq s(\lambda)$, for all $x \notin S$,

$$\Pr\left[\begin{array}{c} \mathsf{Eval}(k\{S\}, x) \\ = \mathsf{Eval}(k, x) \end{array} : \begin{array}{c} k \leftarrow \mathsf{Gen}(1^\lambda), \\ k\{S\} \leftarrow \mathsf{Punc}(k, S) \end{array} \right] = 1.$$

- **Polynomial pseudorandomness at punctured points:** Consider the following experiment $\mathtt{EXP}^R_{\mathcal{A}}(b, \lambda)$ between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$:
  - First, $\mathcal{A}$ chooses a set $S \subseteq \mathcal{X}_\lambda$ such that $|S| \leq s(\lambda)$, and sends it to $\mathcal{C}$.
  - $\mathcal{C}$ samples $k \leftarrow \mathsf{PPRF.Gen}(1^\lambda)$, and computes $k\{S\} \leftarrow \mathsf{PPRF.Punc}(k, S)$.
    * If $b = 0$, compute $L_0 = \{x_i, \mathsf{PPRF.Eval}(k, x_i)\}_{x_i \in S}$.
    * If $b = 1$, compute $L_1 = \{x_i, y_i\}_{x_i \in S}$ where each $y_i \leftarrow \mathcal{Y}$.

    Give $k\{S\}, L_b$ to $\mathcal{A}$.
  - $\mathcal{A}$ produces a bit b'.

  Let W_b be the event that the adversary outputs 1 in experiment b, and define the adversary's advantage to be $\mathtt{Adv}^R_{\mathcal{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$. We say that the PPRF is pseudorandom at punctured points if for all PPT adversaries $\mathcal{A}$, $\mathtt{Adv}^R_{\mathcal{A}}(\lambda) = \mathsf{negl}[\lambda]$.

Definition IV.11 (Puncturable PRF with PV proof of Correctness, [41])**.** We say a family of puncturable PRFs $(\mathsf{PPRF.Gen}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Punc})$ has a PV proof of correctness if $\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punc}$ can be formalzied as function symbols in PV, and there is a proof that:

$$\left(\begin{array}{c} x \notin S \\ \wedge\, k \leftarrow \mathsf{PPRF.Gen}(1^\lambda) \end{array} \right)$$

$$\vdash_{\mathsf{PV}} \left(\begin{array}{c} \mathsf{PPRF.Eval}(\mathsf{PPRF.Punc}(k, S), x) \\ = \mathsf{PPRF.Eval}(k, x) \end{array} \right).$$

Lemma IV.12 ([41, Lemma 14])**.** *The GGM construction [33] of puncturable PRFs has a* PV *proof of correctness.*

E. Bit commitments

Syntax. A commitment scheme in the CRS model consists of polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Ver})$ with the following syntax:

- $\mathsf{Com.Gen}$ is a probabilistic algorithm that takes as input a security parameter 1^λ, and outputs a common reference string crs.
- $\mathsf{Com.Enc}(\mathsf{pk}, m)$ is a probabilistic algorithm that takes as input the common reference string crs and a bit $m \in \{0, 1\}$, and outputs a commitment c along with an opening r.
- $\mathsf{Com.Ver}(\mathsf{crs}, c, m, r)$ is a deterministic algorithm which takes as input the common reference string crs, commitment c, message m and opening r, and either accepts or rejects.

Definition IV.13 (Commitment scheme in the CRS model)**.** A tuple of PPT algorithms $(\mathsf{Com.Gen}, \mathsf{Com.Enc}, \mathsf{Com.Ver})$ is a commitment scheme in a CRS model if it satisfies the following conditions:

Correctness: For all bits $m \in \{0, 1\}$, we have that:

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, c, m, r) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \\ (c, r) \leftarrow \mathsf{Enc}(\mathsf{crs}, m) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

We say that the construction has *perfect* correctness if the above probability is exactly 1.

Statistical Binding: For all $\mathsf{crs} \leftarrow \mathsf{Com.Gen}(1^\lambda)$, and $(c, r) \leftarrow \mathsf{Com.Enc}(\mathsf{crs}, m)$, we have that for all r', $\mathsf{Com.Ver}(\mathsf{crs}, c, 1 - m, r') = 0$.

Γ-secure-Computational Hiding: For all adversaries $\mathcal{A}$ of size $\mathsf{poly}(\Gamma(\lambda))$,

$$\left| \Pr\left[\mathcal{A}(\mathsf{crs}, c) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Com.Gen}(1^\lambda), \\ (c, r) \leftarrow \mathsf{Com.Enc}(\mathsf{crs}, 0) \end{array} \right] - \Pr\left[\mathcal{A}(\mathsf{crs}, c) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Com.Gen}(1^\lambda), \\ (c, r) \leftarrow \mathsf{Com.Enc}(\mathsf{crs}, 1) \end{array} \right] \right| \leq (\Gamma(\lambda))^{-O(1)}$$

Definition IV.14 (Commitment scheme with PV proof of binding.)**.** We say that the commitment scheme $(\mathsf{Com.Gen}, \mathsf{Com.Enc}, \mathsf{Com.Ver})$ has a PV proof of binding if:

$$\vdash_{\mathsf{PV}} \left(\begin{array}{c} b \in \{0, 1\}, \\ \mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda), \\ c \leftarrow \mathsf{Com.Enc}(\mathsf{crs}, b; r) \end{array} \right)$$

$$\rightarrow \forall r', 0 \neq \mathsf{Com.Ver}(\mathsf{crs}, c, 1-b, r).$$

In words, the condition says that for every CRS, any commitment of the bit b cannot be opened to a bit $1-b$.

Lemma IV.15. *Suppose there exists a public-key encryption scheme* ($\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec}$) *with a* PV-*proof of correctness, i.e.*

$$\vdash_{\mathsf{PV}} \left(\begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, b; r) \\ b' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct}) \end{array} \right) \rightarrow b = b'.$$

Then, there exists a perfectly binding commitment scheme with a PV *proof of binding.*

Proof sketch.. The construction is straightforward.

- $\mathsf{Com.Gen}(1^\lambda)$: Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, and output pk.
- $\mathsf{Com.Enc}(\mathsf{pk}, b; r)$: Compute $\mathsf{PKE.Enc}(\mathsf{pk}, b; r)$, and output ct as the commitment, and r as the opening.
- $\mathsf{Com.Ver}(\mathsf{pk}, \mathsf{ct}, b, r)$: Accept if $\mathsf{ct} = \mathsf{PKE}, \mathsf{Enc}(\mathsf{pk}, b; r)$.

It suffices to argue that for all r, r', $\mathsf{PKE.Enc}(\mathsf{pk}, 0; r') \neq \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$. Suppose otherwise, and there exists some ct. Then, $\mathsf{PKE.Dec}(\mathsf{ct}) = 0$ and $\mathsf{PKE.Dec}(\mathsf{ct}) = 1$ by the correctness of the PKE scheme, leading to a contradiction. This completes the proof. □

Therefore, as shown in [41], such a commitment scheme can be instantiated from LWE, or from DDH.

F. Somewhere Statistically Binding Hash

In this section, we recall the notion of statistical binding hash functions [39], additionally with an extraction property (sometimes also referred to as a Somewhere Extractable Hash (SEH)).

Syntax. A somewhere statistically binding (SSB) hash scheme is a type of algorithms ($\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Ver}, \mathsf{Ext}$) with the following syntax:

$\mathsf{Gen}(1^\lambda, 1^N, \Sigma) \rightarrow \mathsf{hk}$. On input a security parameter 1^λ, message length 1^N, alphabet Σ, outputs a hash key hk.

$\mathsf{TGen}(1^\lambda, 1^N, \Sigma, S \subseteq [N]) \rightarrow (\mathsf{hk}^*, \mathsf{td})$**.** On input a security parameter 1^λ, message length 1^N, alphabet Σ, outputs a hash key hk^* along with a trapdoor td.

$\mathsf{Hash}(\mathsf{hk}, \mathbf{x} \in \Sigma^N) \rightarrow \tau$**.** On input a hash key hk and a string $\mathbf{x}$, output a hash value τ.

$\mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \rightarrow \rho$. On input a hash key hk, a string $\mathbf{x} \in \Sigma^N$ and an index $i \in [N]$, output a "local opening" ρ.

$\mathsf{Ver}(\mathsf{hk}, \tau, i, y, \rho) \rightarrow 0/1$**.** On input a hash key hk, hash value τ, index $i \in [N]$, symbol $y \in \Sigma$ and opening ρ, the verification algorith mdecides to accept or reject the local opening.

$\mathsf{Ext}(\mathsf{hk}^*, \mathsf{td}, \tau) \rightarrow \mathbf{y}$**.** On input the hash key hk, trapdoor td and hash value τ, the extraction algorithm outputs a string $\mathbf{x}_S$ on the subset S (the subset which was chosen during the TGen algorithm).

Definition IV.16. A somewhere statistically binding (SSB) family ($\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Ver}, \mathsf{Ext}$) is required to satisfy the following properties:

Succinct Key. The size of the key is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.

Succinct Hash. The size of the hash value c is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.

Succinct Local Opening. The size of the local opening $\rho_i \leftarrow \mathsf{Open}(K, m, i, r)$ is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.

Succinct Verification. The running time of the verification algorithm is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.

Key Indistinguishability. For any non-uniform PPT adversary $\mathcal{A}$ and any polynomial $N = N(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\left| \Pr\left[\mathcal{A}(\mathsf{hk}) = 1 \ : \ \begin{array}{c} S \leftarrow \mathcal{A}(1^\lambda, 1^N), \\ \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^N, \Sigma, 1^{|S|}) \end{array} \right] - \Pr\left[\mathcal{A}(\mathsf{hk}^*) = 1 \ : \ \begin{array}{c} S \leftarrow \mathcal{A}(1^\lambda, 1^N), \\ (\mathsf{hk}*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, \Sigma, S) \end{array} \right] \right| \leq \nu(\lambda).$$

We say the scheme is Γ-secure if the advantage above is at most $(\Gamma(\lambda))^{-O(1)}$ for all adversaries of size $\mathsf{poly}(\Gamma)$.

Opening Completeness. For any hash key hk, any message $\mathbf{x} = (x_1, \dots, x_N) \in \Sigma^N$, any randomness r, and any index $i \in [N]$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{hk}, \tau, x_i, i, \rho_i) = 1 \ : \ \begin{array}{c} \tau \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathbf{x}), \\ \rho_i \leftarrow \mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \end{array} \right] = 1.$$

Extraction Correctness. For any subset $S \subseteq [N]$, any trapdoor key $(\mathsf{hk}^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, any hash τ, any index $i \in S$, any bit $x_{i^*} \in \Sigma$, and any proof ρ_{i^*}, we have

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{hk}, \tau, x_{i^*}, i^*, \rho_{i^*}) = 1 \\ \Rightarrow \mathsf{Ext}(\tau, \mathsf{td})|_{i^*} = x_{i^*} \end{array} \right] = 1.$$

Since the extracted value $\mathsf{Ext}(\tau, \mathsf{td})|_{i^*}$ is unique, the extraction correctness implies statistical binding property.

Remark IV.17. We observe that the construction is a Merkle-tree based construction, and thus the input length

N can in fact be *unbounded*. That is, the running time of $\mathsf{Gen}, \mathsf{TGen}$ grows poly-logarithmically in N. If we set N to be a slightly super-polynomial in λ, then the hash key can support any polynomial input length. Hence, in our construction, we suppress the input length N in Gen and TGen.

"PV-friendly" SSB hashing. In this work, we need will rely on a specific template towards constructing SSB family via a two-to-one SSB (Two-SSB) family. At a high level, a Two-SSB is a special SSB (without local opening) with the following additional properties:

- It hashes exactly two blocks of equal length s (where the length of s is any $\mathsf{poly}(\lambda)$).
- The length of the hash is $s + \mathsf{poly}(\lambda, \log s)$.

We write the explicit transformation in the full version of the paper in order to argue that the resulting SSB hash satisfies Definition IV.21 if the underlying Two-SSB satisfies Definition IV.20.

Theorem IV.18 ([48]). *There exists a two-to-one* SSB *hash family assuming a rate-1 string oblivious transfer scheme (see [25] for a definition) with either perfect correctness or verifiable correctness, i.e. there is an algorithm that efficiently checks that the output of an OT sender message can be decrypted correctly.*

Theorem IV.19 ([25]). *There exists a rate-1 string OT from the following assumptions:*

- *Quadratic residuosity (QR). This scheme has perfect correctness.*
- *Decisional composite residuosity (DCR). This scheme has perfect correctness.*
- *Decisional Diffie Hellman (DDH). This scheme has verifiable correctness.*
- *Learning with error (LWE). This scheme has verifiable correctness.*

In the full version of the paper, we show that the LWE based instantiation satisfies Definition IV.20.

Definition IV.20 (Two-to-one SSB with PV proof of binding)**.** We say a two-to-one SSB family has a PV proof of binding if the following binding property can be formalized in theory PV as function symbols:

$$\left(\begin{array}{c} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Two\text{-}SSB.TGen}(1^\lambda, 1^s, b) \\ \wedge\, x_0, x_1 \in \sigma^k, \\ \wedge\, k \leq 2^\lambda \quad \wedge\, h \leftarrow \mathsf{Two\text{-}SSB.Hash}(\mathsf{hk}, (x_0, x_1)) \\ \wedge\, h \neq \bot \\ \wedge\, y = \mathsf{Two\text{-}SSB.Ext}(\mathsf{hk}, \mathsf{td}, h) \end{array}\right)$$
$$\vdash_{\mathsf{PV}} x_b = y.$$

Definition IV.21 (PV-friendly SSB hashing.)**.** We say that a SSB family is "PV-friendly" if the following properties can be proven in PV: We can then additionally prove the following properties about SSB in PV.

- PV **proof of correctness:** One can prove that if a string is hashed honestly, then an honest opening will be accepted. Formally,
$$\left(\begin{array}{c} \mathsf{hk} \leftarrow \mathsf{SSB.Gen}(1^\lambda, \Sigma) \\ \wedge\, x \leftarrow \Sigma^k \\ \wedge\, \tau \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, x) \\ \wedge\, \rho \leftarrow \mathsf{SSB.Open}(\mathsf{hk}, x, j) \end{array}\right)$$
$$\vdash_{\mathsf{PV}} \mathsf{SSB.Ver}(\mathsf{hk}, \tau, j, x_j, \rho) = 1.$$
- PV **proof of binding property:** One can prove that a hash key generated in trapdoor mode is statistically binding, i.e.
$$\left(\begin{array}{c} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SSB.TGen}(1^\lambda, \Sigma, j) \\ \wedge\, 1 \leftarrow \mathsf{SSB.Ver}(\mathsf{hk}, \tau, j, y, \rho) \\ \wedge\, y' \leftarrow \mathsf{SSB.Ext}(\mathsf{hk}, \mathsf{td}, j, k, \tau) \end{array}\right) \vdash_{\mathsf{PV}} y = y'.$$
- PV **correctness of extraction:** One can prove that extraction on an honestly generated hash is correct, i.e.
$$\left(\begin{array}{c} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SSB.TGen}(1^\lambda, \Sigma, j) \\ \wedge\, x \leftarrow \Sigma^k \\ \wedge\, (\tau, k) \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, x) \\ y \leftarrow \mathsf{SSB.Ext}(\mathsf{hk}, \mathsf{td}, (\tau, k)) \end{array}\right) \vdash_{\mathsf{PV}} y = x_j$$

  The above is a combination of the correctness and binding properties.

V. Fully Succinct Indistinguishability Obfuscation for $\mathcal{EF}$-Equivalent Programs

In this section, we will construct a fully succinct IO for programs with uniform $\mathcal{EF}$ proofs of equivalence.

A. *Definitions*

We start by introducing the notion of IO for Turing machines with *split description*. We consider programs that are represented by a public part pub and a (possibly) secret part M. Formally, we fix some universal machine $\mathcal{U}$ that takes as input both parts of the program (M, pub), an input x to the program and a runtime parameter 1^T and simulates (M, pub) on x. We use $M[\mathsf{pub}](\cdot)$ to denote the machine $\mathcal{U}(M, \mathsf{pub}, \cdot)$.

The syntax of IO for Turing machines with split description is the same as the syntax of IO for Turing machines given in Definition IV.3 except that we consider input programs $M[\mathsf{pub}]$ that include both public and secret parts. The correctness and efficiency properties are also the same as Definition IV.3. For machines with public description we additionally define the notions of description-succinctness and full succinctness. We also redefine the security property to account for the program's public part. For convenience, we rewrite here the full definition of IO for TMs with split description.

Definition V.1 (IO for TMs for Split Description)**.** An indistinguishability obfuscation scheme iO for $\mathcal{U}$ satisfies the following properties:

- **Correctness:** For every λ, $n \leq 2^{\lambda}$, machine $M[\mathsf{pub}]$, and input x such that $|x| \leq n$ and $\mathsf{RT}(M[\mathsf{pub}], x) \leq 2^{\lambda}$, we have that

$$\Pr\left[\begin{array}{c}\widetilde{M}[\mathsf{pub}](x)\\ = M[\mathsf{pub}](x)\end{array} : \widetilde{M} \leftarrow \mathsf{iO}(1^{\lambda}, M[\mathsf{pub}], n)\right] = 1.$$

  We say that the scheme is N-correct for a function $N(\lambda) \leq 2^{\lambda}$ if the above only holds for $n \leq N(\lambda)$.
- **Efficiency:** In the correctness experiment above:
  - The running time of iO is $\mathsf{poly}(\lambda, |M|, |\mathsf{pub}|)$.
  - There exists a polynomial q such that $\mathsf{RT}(\widetilde{M}[\mathsf{pub}], x) = q(\lambda, \mathsf{RT}(M[\mathsf{pub}], x))$.
  - The size of the obfuscation $|\widetilde{M}| = \mathsf{poly}(\lambda, |M|, |\mathsf{pub}|, n)$.
    - *Input-succinctness:* We say that the scheme is *input-succinct* if additionally, $|\widetilde{M}| = \mathsf{poly}(\lambda, |M|, |\mathsf{pub}|)$.
    - *Description-succinctness:* We say that the scheme is *description-succinct* if $|\widetilde{M}| = \mathsf{poly}(\lambda, |M|, n)$.
    - *Full succinctness:* We say the scheme is *fully succinct* if it is both input-succinct and description-succinct, i.e. $|\widetilde{M}| = \mathsf{poly}(\lambda, |M|)$.
- **Security for split programs:** For every poly-size $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every λ, M_0, M_1 and bounds $n \leq p(\lambda)$ such that
  - $|M_0| = |M_1| \leq p(\lambda)$, and $|\mathsf{pub}| \leq p(\lambda)$.
  - For every input x of length at most n, it holds that $M_0[\mathsf{pub}](x) = M_1[\mathsf{pub}](x)$ and $\mathsf{RT}(M_0[\mathsf{pub}], x) = \mathsf{RT}(M_1[\mathsf{pub}], x)$.

  we have that

$$\Pr\left[\mathcal{A}(\widetilde{M}_b, \mathsf{pub}) = b : \begin{array}{c} b \leftarrow \{0,1\} \\ \widetilde{M}_b \leftarrow \mathsf{iO}(1^{\lambda}, M_b[\mathsf{pub}], n)\end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \ .$$

We now define what it means for an IO scheme to have a PV proof of correctness. For this, we introduce the PV symbol $\mathcal{U}_{\mathsf{clock}}$ which on input $M[\mathsf{pub}]$, x and 1^t, simulates $M[\mathsf{pub}](x)$ for t steps, and outputs $M[\mathsf{pub}](x)$ if the computation has halted, and $\perp$ otherwise. As discussed in [52, Section 2.1.1], this is in fact a valid function symbol.

Definition V.2 (Indistinguishability obfuscation with PV proof of correctness)**.** We say that an indistinguishability obfuscation scheme iO has a PV proof of completeness if its correctness can be formalized in theory PV as function symbols:

$$\left(\begin{array}{c}\widetilde{M} = \mathsf{iO}(1^{\lambda}, M[\mathsf{pub}], n; r),\\ |x| \leq n,\\ \mathcal{U}_{\mathsf{clock}}(M[\mathsf{pub}], x, 1^t) \neq \perp\end{array}\right) \to \left(\begin{array}{c}\mathcal{U}_{\mathsf{clock}}(M[\mathsf{pub}], x, 1^t)\\ = \mathcal{U}_{\mathsf{clock}}(\widetilde{M}[\mathsf{pub}], x, 1^{q(t)})\end{array}\right)$$

where q is some polynomial.

Just as in Remark IV.4, we will omit the input length n from the input to IO when the scheme is input or fully-succinct.

1) Uniform-$\mathcal{EF}$-equivalent programs with split description.: In this section we define a restriction of IO security to $\mathcal{EF}$-equivalent programs, extending the definition in Section IV-C1 to the case of programs with split description.

Definition V.3 ($\mathcal{EF}$-equivalence with split description)**.** Let $\rho \in \mathbb{N}$ and T be a polynomial. We say that programs $M_0[\mathsf{pub}](\cdot)$ and $M_1[\mathsf{pub}](\cdot)$ are (ρ, T)-$\mathcal{EF}$-equivalent if there exists a machine $\mathcal{M}_{\Pi}$ of size ρ which takes as input a length bound 1^n, along with M_0, M_1, pub and outputs an $\mathcal{EF}$ proof that $[\![M_0[\mathsf{pub}](x) = M_1[\mathsf{pub}](x)]\!]_n$ in $T(n)$ time.

Definition V.4 ((ρ, T)-$\mathcal{EF}$ security for IO for split programs)**.** Let $\rho(\lambda)$ and $T(\lambda, n)$ be polynomials. We say that an IO scheme satisfies (ρ, T)-$\mathcal{EF}$-security if for every λ, and polynomial-sized adversary $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ satisfying:

- $|M_0[\mathsf{pub}]| = |M_1[\mathsf{pub}]| \leq p(\lambda)$.
- for all x, $\mathsf{RT}(M_0[\mathsf{pub}](x)) = \mathsf{RT}(M_1[\mathsf{pub}](x))$,
- $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ are $(\rho(\lambda), T(\lambda, \cdot))$-equivalent,

$$\Pr\left[\mathcal{A}(\widetilde{M}_b[\mathsf{pub}]) = b : \begin{array}{c} b \leftarrow \{0,1\} \\ \widetilde{M}_b[\mathsf{pub}] \leftarrow \mathsf{iO}(1^{\lambda}, M_b[\mathsf{pub}])\end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \ .$$

We say the scheme is Γ-secure if the advantage above is at most $(\Gamma(\lambda))^{-O(1)}$ for all adversaries of size $\mathsf{poly}(\Gamma)$.

Definition V.5. ($\mathcal{EF}$-IO for programs with split description) Let $N(\lambda) \leq 2^{\lambda}$ be a function. An N-correct $\mathcal{EF}$-

IO for split programs is given by an algorithm $\mathsf{iO}_{\rho,T}$ for every pair of polynomials $\rho(\lambda)$ and $T(\lambda, n)$, $\mathsf{iO}_{\rho,T}$ is an N-correct input-succinct IO scheme with (ρ, T)-security, satisfying the following efficiency: In the correctness experiment described in Definition V.1:

- The running time of $\mathsf{iO}_{\rho,T}(1^\lambda, M[\mathsf{pub}])$ is $\mathsf{poly}(\lambda, |M|, |\mathsf{pub}|, \rho(\lambda))$. We say the scheme is additionally *fully succinct* if the size of the obfuscation is $\mathsf{poly}(\lambda, |M|, \rho(\lambda))$.
- The exists a polynomial q such that $\mathsf{RT}(\widetilde{M}[\mathsf{pub}], x) = q(\lambda, \mathsf{RT}(M[\mathsf{pub}], x), \rho(\lambda), T(\lambda, |x|))$.

B. Main Theorem

Theorem V.6. *Let $N(\lambda) \le 2^\lambda$ be a function. Assuming*

- *a $\mathsf{poly}(N)$-secure, N-correct $\mathcal{EF}$-IO (as in Definition IV.7),*
- *a $\mathsf{poly}(N)$-secure, PV-friendly somewhere statistically binding hash function (Definition IV.21),*

there exists an N-correct fully succinct $\mathcal{EF}$-IO for TMs with split description.

By instantiating the above theorem with Theorem IV.8, we have the following corollary.

Corollary V.7. *Let $N(\lambda) \le 2^\lambda$ be a function. Assuming*

- *$\mathsf{poly}(N)$-hardness of Learning with Errors,*
- *sub-exponentially secure one-way functions, and*
- *sub-exponentially secure indistinguishability obfuscation for circuits,*

there exists a N-correct, (ρ, T)-secure fully succinct $\mathcal{EF}$-IO scheme. Moreover, the scheme has a PV proof of correctness.

We defer the proof of this theorem to the full version of this paper.

C. Description-succinct IO

While constructions of description-succinct IO (without input-succinctness) have been implicit in prior works, we provide an alternative construction that follows as a corollary of Theorem V.6.

Corollary V.8 (Alternative construction of description–succinct IO.)**.** *Assuming*

- *sub-exponentially secure Learning with Errors,*
- *sub-exponentially secure one-way functions, and*
- *sub-exponentially secure indistinguishability obfuscation for circuits,*

there exists a description-succinct IO scheme for split programs.

We defer the proof of this theorem to the full version of the paper.

VI. Equivalence between Succinct IO, Witness Encryption and SNARGs

In this section, we show an equivalence (up to standard assumptions) between the existence of

- input-succinct (resp. fully succinct) IO with PV-proof of correctness,
- witness-succinct (resp. fully succinct) witness encryption with PV-proof of correctness, and
- and witness-succinct (resp. fully succinct) non-adaptive SNARGs with PV-proof of correctness and uniqueness.

We first define witness encryption and SNARGs along with these efficiency guarantees in Section VI-A, and then present the main theorem in Section VI-B.

A. Definitions

In this section, we recall the definition of witness encryption [29] and succinct non-interactive arguments (SNARGs), and introduce terminology to refer to various efficiency metrics for the size of the witness encryption ciphertext.

Syntax. A witness encryption scheme for NP is a tuple of algorithms $(\mathsf{Enc}, \mathsf{Dec})$:

- Enc is a probabilistic algorithm that takes as input a security parameter λ, a machine $\mathcal{R}$, instance x, a witness length bound n, and a bit b. It outputs a ciphertext ct.
- Dec is a deterministic algorithm that takes as input a a machine $\mathcal{R}$, an instance x, a witness w and a ciphertext ct. It outputs a bit b or $\perp$.

Definition VI.1 (Witness encryption)**.** A witness encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ for NP satisfies the following properties:

- **Correctness:** For every machine $\mathcal{R}$, instance x, witness w satisfying $|w| \le n \le 2^\lambda$, $\mathcal{R}(x, w) = 1$ and $\mathsf{RT}(x, w) \le 2^\lambda$, bit $b \in \{0, 1\}$, we have that
$$\Pr\left[\ \mathsf{Dec}(\mathcal{R}, x, w, \mathsf{ct}) = b \ :\ \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, \mathcal{R}, x, n, b)\ \right] = 1\ .$$
We say that the scheme is N-correct for a function $N(\lambda) \le 2^\lambda$ if the above only holds for $n \le N(\lambda)$.
- **Efficiency:** In the correctness experiment above:
  - The running time of Enc is $\mathsf{poly}(\lambda, |\mathcal{R}|, |x|, n)$.
  - The running time of Dec is $\mathsf{poly}(\lambda, \mathsf{RT}(\mathcal{R}, (x, w)))$.
  - *Witness-succinctness*: The scheme is *witness-succinct* if $|\mathsf{ct}| = \mathsf{poly}(\lambda, |\mathcal{R}|, |x|)$.

- *Full succinctness*: We say that the scheme is *fully succinct* if $|\mathsf{ct}| = \mathsf{poly}(\lambda, |\mathcal{R}|)$.

- **Security:** For every poly-size $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every λ, machine $\mathcal{R}$, instance x and bounds $n \leq p(\lambda)$ such that
  - $|\mathcal{R}|, |x| \leq p(\lambda)$.
  - For every w, $\mathcal{R}(x, w) = 0$.

  we have that

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \;:\; \begin{array}{c} b \leftarrow \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, \mathcal{R}, x, n) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \;.$$

Definition VI.2 (Witness encryption with PV proof of completeness)**.** We say that a witness encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ has a PV proof of completeness if its correctness can be formalized in theory PV as function symbols:

$$\left(\begin{array}{c} \mathsf{ct} = \mathsf{Enc}(1^\lambda, \mathcal{R}, x, n, b; r) \wedge \\ |w| \leq n \;\wedge\; R(x, w) = 1 \end{array}\right) \vdash_{\mathsf{PV}} \mathsf{Dec}(\mathcal{R}, x, w, \mathsf{ct}) = b.$$

If the runtime t and input length n are clear from context, we omit these conditions for readability.

Syntax. A succinct non-interactive argument (SNARG) system for NP consists of algorithms $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ with the following syntax:

- $\mathsf{SNARG.Gen}$ is a probabilistic algorithm that takes as input a security parameter λ, a relation $\mathcal{R}$ (represented as a Turing machine), a time bound t, an instance length bound n_x and a witness length bound n_w. It outputs a common reference strings crs.
- $\mathsf{SNARG}.\mathcal{P}$ is a deterministic algorithm that takes as input the common reference strings crs, a relation $\mathcal{R}$, an instance x and a witness w. It outputs a proof Π.
- $\mathsf{SNARG}.\mathcal{V}$ is a deterministic algorithm that takes as input the common reference strings crs, a machine $\mathcal{R}$, an instance x and proof Π. It outputs a bit indicating if it accepts or rejects.

Definition VI.3 (SNARG)**.** A succinct non-interactive argument system for NP $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ satisfies the following properties:

- **Correctness:** For every λ, $n_x, n_w \leq 2^\lambda$, machine $\mathcal{R}$, instance x, witness w such that $\mathcal{R}(x, w) = 1$ and bounds $|x| \leq n_x, |w| \leq n_w$ where $\mathsf{RT}(\mathcal{R}, (x, w)) \leq 2^\lambda$, we have that

$$\Pr\left[\mathcal{V}(\mathsf{crs}, \mathcal{R}, x, \pi) = 1 \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{R}, n_x, n_w), \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, \mathcal{R}, x, w) \end{array}\right] = 1 \;.$$

- **Efficiency:** In the correctness experiment above:
  - The running time of Gen is $\mathsf{poly}(\lambda, |\mathcal{R}|, n_x, n_w)$.
  - The running time of $\mathcal{P}$ is $\mathsf{poly}(\lambda, |\mathcal{R}|, n_x, n_w)$.
  - The running time of $\mathcal{V}$ is $\mathsf{poly}(\lambda, |\mathcal{R}|, n_x)$.
  - We say that the scheme is *witness-succinct* if $|\mathsf{crs}| = \mathsf{poly}(\lambda, |\mathcal{R}|, n_x, \log n_w)$ and $|\Pi| = \mathsf{poly}(\lambda, |\mathcal{R}|, n_x)$.
  - We say that the scheme is *fully succinct* if $|\mathsf{crs}| = \mathsf{poly}(\lambda, |\mathcal{R}|)$ and $|\Pi| = \mathsf{poly}(\lambda, |\mathcal{R}|)$.
- **Non-adaptive Soundness:** For every poly-size $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every λ, machine $\mathcal{R}$, instance x and bounds $n_x, n_w \leq p(\lambda)$ such that
  - $|\mathcal{R}|, |x| \leq p(\lambda)$.
  - For every w, $\mathcal{R}(x, w) = 0$.

  we have that

$$\Pr\left[\mathcal{V}(\mathsf{crs}, \mathcal{R}, x, \pi) = 1 \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{R}, n_x, n_w), \\ \pi \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mu(\lambda) \;.$$

Definition VI.4 (SNARG with unique proofs.)**.** A succinct non-interactive argument system for NP $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ has unique proofs if for every machine $\mathcal{R}$, instance x, pair of witnesses w_1, w_2 such that $\mathcal{R}(x, w) = 1$, $\mathcal{R}(x, w_2) = 1$ and bounds $n_x \geq |x|$, $n_w \geq \max(|w_1|, |w_2|)$ and $t \geq \max(\mathsf{RT}(\mathcal{R}, (x, w_1)), \mathsf{RT}(\mathcal{R}, (x, w_2)))$ we have that

$$\Pr\left[\pi_1 = \pi_2 \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{R}, n_x, n_w), \\ \pi_1 \leftarrow \mathcal{P}(\mathsf{crs}, \mathcal{R}, x, w_1), \\ \pi_2 \leftarrow \mathcal{P}(\mathsf{crs}, \mathcal{R}, x, w_2) \end{array}\right] = 1 \;.$$

Definition VI.5 (SNARG with PV proof of completeness)**.** We say that a succinct non-interactive argument system $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ has a PV proof of completeness and uniqueness if its correctness can be formalized in theory PV as function symbols:

$$\left(\begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{R}, n_x, n_w) \\ \wedge\, |w| \leq n_w \\ \wedge\, \mathcal{R}(x, w) \\ \wedge\, \pi \leftarrow \mathcal{P}(\mathsf{crs}, \mathcal{R}, x, w) \end{array}\right) \vdash_{\mathsf{PV}} (\mathcal{V}(\mathsf{crs}, \mathcal{R}, x, \pi_1) = 1)$$

Definition VI.6 (SNARG with PV proof of completeness and uniqueness)**.** We say that a succinct non-interactive argument system $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ has a PV proof of completeness and uniqueness if its correctness can be formalized in theory PV as function symbols:

$$\left(\begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{R}, n_x, n_w) \\ \wedge\, |w_1|, |w_2| \le n_w \\ \wedge\, \mathcal{R}(x, w_1) = 1 \wedge\, \mathcal{R}(x, w_2) = 1 \\ \wedge\, \pi_1 \leftarrow \mathcal{P}(\mathsf{crs}, \mathcal{R}, x, w_1) \\ \wedge\, \pi_2 \leftarrow \mathcal{P}(\mathsf{crs}, \mathcal{R}, x, w_2) \end{array}\right) \vdash_{\mathsf{PV}} (\pi_1 = \pi_2).$$

B. Main Theorem

Theorem VI.7. *Let $N(\lambda) \le 2^\lambda$ be a function. Assuming the existence of:*

- $\mathsf{poly}(N)$*-secure fully succinct $\mathcal{EF}$-IO (Definition V.5) as well as one-way functions,*
- $\mathsf{poly}(N)$*-secure PV-friendly* SSB *hash,*

we have that the following statements are equivalent:

(I) There exists an N-correct fully succinct (resp. input-succinct) IO with PV*-proof of correctness.*
(II) There exists an N-correct fully succinct (resp. witness-succinct) WE with PV*-proof of correctness.*
(III) There exists an N-correct fully succinct (resp. witness-succinct) non-adaptive SNARGs with unique proofs and PV-proofs of both correctness and uniqueness.

We defer the proof of this theorem to the full version of this paper.

VII. Succinct Computational Secret Sharing

In this section, we present our succinct computational secret sharing schemes.

A. Definitions

We first introduce some definitions, following the terminology from the work of [4], [50].

Notation. For $x, y \in \{0,1\}^n$, we write $x \succeq y$ if for all $i \in [n]$, $x_i \ge y_i$. For an $x \in \{0,1\}^n$, we denote by S_x the subset $\{i : x_i = 1\}$.

Monotone functions. We say that a function P a function $P : \{0,1\}^n \to \{0,1\}$ is monotone if for all $x, y \in \{0,1\}^n$ such that $x \succeq y$, if $P(x) = 0$, then $P(y) = 0$.

For a non-deterministic $\mathcal{M}_P : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, we define $P(x) = 1$ if and only if there exists some $w \in \{0,1\}^w$ such that $P(x, w) = 1$. We say that $\mathcal{M}_P$ is an *monotone non-deterministic function* if the corresponding function P is monotone.

Computational secret sharing. WE now recall the definition of computational secret sharing.

Definition VII.1 (Authorized sets and access structures)**.** Consider a set of n parties, and let $x \in \{0,1\}^n$ be an indicator vector denoting a subset $S_x = \{i : x_i = 1\}$ of the parties. Given a boolean function $P : \{0,1\}^n \to \{0,1\}$, we say a set S_x is authorized if $P(x) = 1$. The set P then represents an *access structure*, comprising all authorized sets S_x.

We can now define a computational secret sharing scheme for a function P.

Definition VII.2 (Computational secret sharing)**.** A computational secret sharing (CSS) scheme for a class of (possibly non-deterministic) monotone programs $\mathcal{F}$ is a pair of algorithms $\mathsf{CSS} = (\mathsf{CSS.Share}, \mathsf{CSS.Recon})$ such that:

- $\mathsf{CSS.Share}(1^\lambda, 1^n, P, s) \to (\mathsf{crs}, \mathsf{sh}_1, \ldots, \mathsf{sh}_n)$. This is a randomized polynomial time algorithm which takes as input the security parameter λ, the number of parties n, a (possibly non-deterministic) monotone program $P \in \mathcal{F}$ and a secret $s \in \{0,1\}$, and outputs a public share crs, and n shares $\{\mathsf{sh}_i\}_{i\in[n]}$. Each share sh_i is given to party i, and crs is a public string available to all parties.
- $\mathsf{CSS.Recon}(P, x, w, \mathsf{crs}, \{\mathsf{sh}_i\}_{i\in S_x}) \to s$. This is a deterministic algorithm which takes as input a program P, a string $x \in \{0,1\}^n$ indicating a subset of parties of $[n]$, a non-deterministic witness $w \in \{0,1\}^m$, and a subset of shares $\{\mathsf{sh}_i\}_{i\in S_x}$. The algorithm outputs a secret $s \in \{0,1\}$.

We require a CSS scheme to additionally satisfy the following properties.

Correctness. For every $\lambda \in \mathbb{N}$ and secret $s \in \{0,1\}$, for $x \in \{0,1\}^n$, a witness $w \in \{0,1\}^m$, and a program P such that $P(x, w) = 1$, we have that

$$\Pr\Big[s \leftarrow \mathsf{CSS.Recon}(P, x, w, \mathsf{crs}, \{\mathsf{sh}_i\}_{i\in S_x}) : (\mathsf{crs}, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{CSS.Share}(1^\lambda, 1^n, P, s)\Big] = 1.$$

Security. Consider the following game between a non-uniform adversary $\mathcal{A}$ of size $\mathsf{poly}(\lambda)$, and a challenger:

- On input 1^λ, 1^n and P, $\mathcal{A}$ chooses an input $x \in \{0,1\}^n$ such that $P(x) = 0$. It sends x to the challenger.
- The challenger chooses $s \in \{0,1\}$, and computes shares

$$(\mathsf{crs}, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{CSS.Share}(1^\lambda, 1^n, P, s).$$

Send $\mathsf{crs}, \{\mathsf{sh}_i\}_{i \in S_x}$ to $\mathcal{A}$.

- $\mathcal{A}$ outputs a bit b.
- Challenger accepts if $s = b$ and $P(x) = 0$.

We say the scheme is secure if the probability that $\mathcal{A}$ wins (i.e. the challenger accepts) is at most $1/2 + \mathsf{negl}(\lambda)$ for large enough λ.

Remark VII.3. If P is non-deterministic, we say that the CSS is a "secret sharing scheme for NP" (as defined by [50]). If it is deterministic, we refer to the scheme as a "secret sharing scheme for P".

We now characterize the classes of monotone functions that admit proofs of monotonicity in Theory PV. Looking forward, for these classes, we will derive results based only on fully succinct pviO.

Definition VII.4 (Deterministic Monotone families with PV proofs of monotonicity)**.** We say that a family $\mathcal{F}$ of deterministic monotone functions has a PV proof of monotonicity if the following conditions hold:

- Given a program P, there exists a polynomial time machine $\mathsf{Member}_{\mathcal{F}}$ such that $\mathsf{Member}_{\mathcal{F}}(P) = 1$ if and only if $P \in \mathcal{F}$.
- There is a PV proof that:

$$\begin{pmatrix} \mathsf{Member}_{\mathcal{F}}(P) = 1 \\ \wedge\, P(x) = 0 \\ \wedge\, \partial\mathsf{Less}(y, x) \end{pmatrix} \vdash_{\mathsf{PV}} (P(y) = 0).$$

  where $\partial\mathsf{Less}(y, x)$ is a PV function formalizing $\preceq$. In words, the above condition says that for $P \in \mathcal{F}$, one can prove that P is indeed a monotone function.

Zero certificate complexity of a monotone function. Now, we introduce a notion of "zero certificate complexity" of a monotone function. Note that for our setting, we care about the time-bounded Kolmogorov complexity of the zero certificate rather than Hamming weight.

Definition VII.5 (t-Time-bounded Conditional Kolmogorov complexity)**.** The t-time bounded conditional Kolmogorov complexity of a string $x \in \{0,1\}^*$ given a string $z \in \{0,1\}^*$, denoted by $K^t(x|z)$, is the length of the shortest program and input pair (Π, y) such that $\Pi[z](y)$ outputs x in time $t(|x|)$.

We introduce the following new notion of zero-complexity of a monotone function in terms of conditional Kolmogorov complexity, which will be convenient in our proof later.

Definition VII.6 (t-Time Zero Complexity of a Monotone Function)**.** Given a (possibly non-deterministic) monotone function $P : \{0,1\}^n \to \{0,1\}$, we define the zero certificate complexity of P to be

$$C_{K^t}(P) := \max_{x \in \{0,1\}^n : P(x) = 0} \left(\min_{y : S_y \supseteq S_x} K^t(y|P) \right),$$

where K^t is the t-time bounded Kolmogorov complexity, as defined in Definition VII.5.

It is easy to see that for $t \leq \mathsf{poly}(n)$, $P : \{0,1\}^n \to \{0,1\}$, $C_{K^t}(P) \leq n + O(1)$ since $K^t(y|P) \leq n + O(1)$ for all n-bit strings y.

Remark VII.7. We note that it might potentially inefficient to compute the corresponding programs matching the certificate complexity given the program P and input x. However, looking forward, this program will only show up in the security analysis.

B. Construction

Our construction will utilize the following main ingredients:

- index-hiding somewhere statistically binding hash family with PV proof of binding Definition IV.21,
- A perfectly binding commitment scheme with PV proof of binding (Definition IV.14):

$$\begin{pmatrix} b \in \{0,1\}, \\ \mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda), \\ c \leftarrow \mathsf{Com.Enc}(\mathsf{crs}, b; r) \end{pmatrix}$$
$$\vdash_{\mathsf{PV}} \forall r', \quad c \neq \mathsf{Com.Enc}(\mathsf{crs}, 1-b; r').$$

  where r' on the RHS is a free variable (i.e. the proof holds for all r'). Note that we refer to the output of Com.Gen as pk rather than a common reference string for notational convenience in the description of our algorithm.
- Fully succinct $\mathcal{EF}$-IO scheme iO (Definition V.5) (or a fully succinct IO scheme for the result on secret sharing for all of NP, as in Theorem VII.8).
- A length-expanding PRG G.

We now present our construction for secret sharing for NP. In the following, let $P : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ ($m = 0$ if P is a deterministic circuit).

For notational convenience, we will let our CSS.Share algorithm take as input additional parameters 1^ℓ and t. Looking forward, we will choose $t \geq \mathsf{poly}(\lambda, \mathsf{RT}(P, x))$, and $\ell \geq C_{K^t}(P)$ (as defined in Definition VII.6). Recall that one can always choose $\ell = n$ and $t = \mathsf{poly}(\lambda, \mathsf{RT}(P, x))$.

$\underline{\mathsf{CSS.Share}(1^\lambda, 1^n, P, s, (1^\ell, t))}$:

- **Construct commitments.**

- Sample $\mathsf{pk} \leftarrow \mathsf{Com.Gen}(1^\lambda)$.
- For $i = 1, 2, \ldots, n$, sample randomness $r_i \leftarrow \{0,1\}^\lambda$, and $c_i \leftarrow \mathsf{Com.Enc}(\mathsf{pk}, 1; r_i)$.
- Sample $\mathsf{hk} \leftarrow \mathsf{SSB.Gen}(1^\lambda)$, and compute $\mathsf{dig} \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n))$, and let ρ_i denote the local openings from dig to c_i.
- For $i \in [n]$, set $\pi_i = (c_i, \rho_i, r_i)$.

- **Constructed obfuscated program.**
  - Sample $\mathsf{crs} \leftarrow \{0,1\}^\ell$.
  - Consider the program $M_{\mathsf{pk},\mathsf{hk},\mathsf{dig},s}[\mathsf{pub} = (P, \mathsf{crs})]$ which on input $x \in \{0,1\}^n$, $w \in \{0,1\}^m$ and $\{\pi_i\}_{i \in S_x}$ does the following:
    - ∗ Abort if $P(x, w) = 0$.
    - ∗ For each $i \in S_x$:
      - · Parse $\pi_i = (c_i, \rho_i, r_i)$.
      - · Check that $\mathsf{Hash.Ver}(\mathsf{hk}, \mathsf{dig}, c_i, \rho_i) = 1$.
      - · Check that $c_i = \mathsf{Com.Enc}(\mathsf{pk}, 1; r_i)$.
      - · Abort if any test fails.
    - ∗ If $P(x, w) = 1$, then output s. Else, output $\bot$.
  - Set $\Gamma = \mathsf{iO}(M_{\mathsf{pk},\mathsf{hk},\mathsf{dig},s}, \mathsf{pub} := (P, \mathsf{crs}))$.
- **Create shares:**
  - Set $\mathsf{sh}_i = \{\pi_i, \Gamma\}$.
  - Output $(\mathsf{crs}, \{\mathsf{sh}_i\}_{i \in [n]}, 1^\rho, t)$, where ρ is an appropriate padding parameter chosen according to our security analysis.

$\mathsf{CSS.Recon}(P, x, \mathsf{crs}, \{\mathsf{sh}_i\}_{i \in S_x})$:

- Parse $(c_i, \rho_i, r_i, \Gamma) \leftarrow \mathsf{sh}_i$ for $i \in S_x$.
- Compute $s \leftarrow \Gamma[\mathsf{pub} = (P, \mathsf{crs})](x, \{\mathsf{sh}_i\}_{i \in S_x})$.
- Output s.

Theorem VII.8 (Succinct secret sharing for NP)**.** *Assuming*

- *fully succinct IO,*
- *perfectly-binding commitment schemes,*
- *index-hiding somewhere statistically binding hash family,*
- *pseudorandom-generators,*

there exists a n-party computational secret sharing for all monotone function families $\mathcal{F}$ such that for an access structure $P \in \mathcal{F}$:

- *Common reference string is a uniformly random string of length $C_{K^t}(P)$ for some time parameter t.*
- *Each share size is* $\mathsf{poly}(\lambda, \log n, \log t, \log |P|)$.
- *The reconstruction algorithm runs in time* $\mathsf{poly}(\mathsf{RT}(P, (x, w)), n, t, \lambda)$

Theorem VII.9 (Succinct secret sharing for P)**.** *Assuming*

- *fully succinct $\mathcal{EF}$-IO (as in Definition V.5),*
- *perfectly-binding commitment schemes with* PV *proof of binding (Definition IV.14),*
- *index-hiding somewhere statistically binding hash family with* PV *proof of binding (Definition IV.21),*
- *pseudorandom-generators,*

there exists a n-party computational secret sharing for deterministic monotone function families $\mathcal{F}$ which have PV *proofs of monotonicity Π (Definition VII.4) such that for $P \in \mathcal{F}$:*

- *Common reference string is a uniformly random string of length $C_{K^t}(P)$ for some time-parameter t.*
- *Each share size is* $\mathsf{poly}(\lambda, \log n, \log |P|, |\Pi|)$*, where $P \in \mathcal{F}$ is the access structure for the scheme.*
- *The reconstruction algorithm runs in time* $\mathsf{poly}(\mathsf{RT}(P, x), n, t, \lambda)$.

We defer the proof of these theorems to the full version of this paper.

C. Examples

Monotone circuits. Recall that a circuit $C : \{0,1\}^n \to \{0,1\}$ is a *monotone circuit* if it only consists $\wedge$ and $\vee$ gates (no $\neg$ gates).

Lemma VII.10 (PV proof of monotonicity of monotone circuits)**.** *The monotonicity of monotone circuits can be proven in Cook's theory* PV*. Namely, the following statement can be proven in* PV*:*

$$(\mathsf{MonoCirc}(C) = 1 \wedge \partial\mathsf{Less}(x, y) = 1) \\ \to \partial\mathsf{Less}(U(C, x), U(C, y)) = 1,$$

where MonoCirc *is a* PV *function deciding whether a circuit is a monotone circuit, $\partial\mathsf{Less}(x, y)$ is a* PV *function deciding whether every entry of the truth assignments x is less than or equal to the corresponding entry of y, and $U(C, x)$ is an universal Turing machine formalized as a* PV *function that computes $C(x)$.*

Proof Sketch. We prove this by an induction argument on the size of the monotone circuit C. Without loss of generality, we can always assume that $U(C, x)$ not only outputs $C(x)$, but also all the intermediate wire values during the execution of $C(x)$.

Base Case. The base case of the induction is the case when C only consists of a single gate. Namely, the circuit C takes a bit from the input and outputs it directly. In this case, $U(C, x) = \mathsf{At}(x, i)$ for some index i, where $\mathsf{At}(x, i)$ is the PV function that takes x and i as input and outputs the i-th bit in the binary representation of x. Clearly, we have

$$U(C,x) = \mathsf{At}(x,i) \leq \mathsf{At}(y,i) = U(C,y),$$

since $\partial\mathsf{Less}(x,y) = 1 \to \mathsf{At}(x,i) \leq \mathsf{At}(y,i)$ can be proven in PV.

Induction Step. Let g be an output gate of C, and let C' be the circuit C with g removed. In the induction step, we assume the theorem holds for C' and now we prove the theorem for C. By induction hypothesis, we have $(\mathsf{MonoCirc}(C') = 1 \wedge \partial\mathsf{Less}(x,y) = 1) \to \partial\mathsf{Less}(U(C',x), U(C',y)) = 1$.

The input wire values of g are two output bits in $U(C',\cdot)$. Let their indices be i,j, respectively. Then the output of g can be expressed as $g(\mathsf{At}(U(C',\cdot),i), \mathsf{At}(U(C',\cdot),j))$. Since g is either an $\wedge$ or an $\vee$, we can derive in PV that

$$\begin{aligned} &g(\mathsf{At}(U(C',x),i), \mathsf{At}(U(C',x),j)) \\ &\leq g(\mathsf{At}(U(C',y),i), \mathsf{At}(U(C',y),j)) \end{aligned}$$

from $\partial\mathsf{Less}(U(C',x), U(C',y)) = 1$. This proves that the output value of g is also monotone, which finishes the induction step.

Finally, we argue that each step in the above mathematical proof can be formalized in PV. Specifically, the induction step can be formalized using the induction rule . This finishes the proof sketch. □

Definition VII.11 (Matching access structure)**.** Let G_n for even n be the complete undirected graph with n vertices and $\binom{n}{2}$ edges. A perfect matching in the graph is a subset of edges E of size $|E| = n/2$ such that each $v \in V$ appears in exactly one edge in E. Then, Γ is the access structure which takes input $E \in \{0,1\}^{\binom{n}{2}}$, and

$$\Gamma_{\mathsf{general}}(E) = \begin{cases} 1 & \text{if } E \text{ contains a perfect matching.} \\ 0 & \text{otherwise.} \end{cases}$$

One can efficiently implement Γ_n via Edmonds' matching algorithm [27].

Definition VII.12 (Bipartite matching access structure)**.** In a bipartite matching access structure, we instead consider the graph $G_{n,n}$ which is a complete undirected **bipartite** graph on V_1, V_2, each of size n. We say that a matching access structure is *bipartite* if we instead consider the bipartite graph $G_{n,n}$ on $2n$ vertices. Then,

$$\Gamma(E) = \begin{cases} 1 & \text{if } E \text{ contains a perfect matching.} \\ 0 & \text{otherwise.} \end{cases}$$

One can efficiently implement Γ via the augmenting-path algorithm (see [21] for details).

Lemma VII.13 (PV proof of monotonicity of bipartite matching access structure, [21])**.** *The monotonicity of the bipartite matching access structure can be proven in theory* PV*. Namely, the following statement can be proven in* PV*:*

$$\partial\mathsf{Less}(x,y) = 1 \to (\Gamma(x) = 1 \to \Gamma(y) = 1)$$

where $\partial\mathsf{Less}(x,y)$ *is a* PV *function deciding whether every entry of the truth assignments* x *is less than or equal to the corresponding entry of* y*.*

Proof. Let Hungarian be the PV function formalizing the augmenting-path algorithm of finding the perfect matching in a bipartite graph E. The work [21] formalized the correctness of augmenting-path algorithm in PV. Hence, we have the following formulas proven in PV:

$$\begin{aligned} &\Gamma(E) = 1 \\ &\to (\mathsf{Hungarian}(E) = w \to \mathsf{Verify}(E,w) = 1), \\ &\mathsf{Verify}(E,w') = 1 \to \Gamma(E) = 1 \end{aligned} \tag{3}$$

where w is the variable denoting the matching outputted by Hungarian, and Verify is a PV function verifying whether a given matching w is a perfect matching in E.

Now, consider any two bipartite graphs $x, y \in \{0,1\}^{\binom{n}{2}}$ with $2n$ vertices and $\partial\mathsf{Less}(x,y) = 1$. If $\Gamma(x) = 1$, then we have $\mathsf{Hungarian}(x) = w \to \mathsf{Verify}(x,w) = 1$. Hence, w is a perfect matching in x, and from $\partial\mathsf{Less}(x,y) = 1$, we know that x is a subgraph of y. Therefore, we also have $\mathsf{Verify}(y,w) = 1$. Applying Equation (3), we derive that $\Gamma(y) = 1$. This finishes the proof.

This proof can be formalized in PV because the key step in the above argument $(\mathsf{Verify}(x,w) = 1 \wedge \partial\mathsf{Less}(x,y) = 1) \to \mathsf{Verify}(y,w) = 1$ can be formalized in PV. □

Remark VII.14. By a similar argument, if one can show that Edmonds' algorithm for matchings [27] has a PV proof of correctness, then one can show that $\Gamma_{\mathsf{general}}$ (for general graphs) has a PV proof of monotonicity.

Moreover, we show that the zero complexity of this matching access structure is in fact better than the trivial bound of n^2.

Lemma VII.15 (Zero complexity of bipartite matching access structure.)**.** *There exists* $t = \mathsf{poly}(n)$ *such that:* $C_{K_t}(\Gamma_{n,n}) \leq 2n + O(1)$.

Proof. Consider the bipartite graph G on vertex sets V_1

and V_2. Consider any set $E \subseteq V_1 \times V_2$ that does not contain a perfect matching. Then, by Hall's marriage lemma, this is true if and only if there exists $S \subseteq V_1$ such that $|N_E(S)| < |S|$, where $N_E(S)$ denotes the set of all neighbors of $v \in S$ in the subgraph defined by E. In other words, E contains no edges from $S \times V_2 \setminus \{N(S)\}$. Consider the zero certificate obtained by the following procedure:

- Takes as input S and $V_2 \setminus \{N(S)\}$.
- Let $W = S \times V_2 \setminus \{N(S)\}$.
- Output $\overline{W}$ (i.e. the complement of the graph).

Let Z denote the output of the above algorithm. Clearly, $K_t(Z) \leq 2n + O(1)$, for $t = O(n^2)$. By definition, E is a subgraph of Z. Moreover, it is clear by construction that $N_Z(S) = N_E(S)$, and hence $|N_Z(S)| < |S|$. Therefore, Z does not have a perfect matching either, and $\Gamma_{n,n}(Z) = 0$.

The above gives a procedure for constructing a zero-certificate for any E with K^t complexity $2n+O(1)$. ☐

Corollary VII.16. *From the same assumptions as in Theorem VII.9, there exists a n^2-party secret sharing scheme for the bipartite matching access structure $\Gamma_{n,n}$ with:*

- *a public share which is a random string of size $2n$,*
- *individual shares of size $\mathsf{poly}(\lambda, \log n)$.*

Proof. Combining Theorem VII.9 with Lemma VII.13 and Lemma VII.15, the result follows immediately. ☐

VIII. Necessary Padding and High-Rate Obfuscation

In this section, we first give a general template algorithm to overcome "necessary-padding" via a public random string, as in Algorithm 1. We then introduce a notion of "ρ-closeness", and we show how to analyze Algorithm 1 for in these settings in Lemma VIII.2 and Theorem VIII.3. We then show several corollaries to high-rate obfuscation in Section VIII-B.

A. Construction and Analysis

We first give a general construction for superfluous padding in Algorithm 1.

Ingredients. For this construction, we will need two main ingredients:

- An IO scheme iO, which is either description-succinct, fully succinct, or fully succinct only for PV-equivalent machines (as guaranteed by Theorem V.6). Looking forward, we will get different efficiency and security guarantees based on which notion of IO we use.
- A puncturable PRF family (PPRF.Gen, PPRF.Eval, PPRF.Punc). We will additionally assume that the family has a PV proof of correctness (as defined in Definition IV.11), as this is achieved by the standard GGM construction of PRFs from one-way functions (see Lemma IV.12).

Construction. We can now describe the construction in Algorithm 1. In words, the algorithm takes as input a padding parameter 1^ℓ, and pads the input program M to size ℓ by sampling a uniformly random string of length $\ell - |M|$. We show that this strategy is in fact sound.

Algorithm 1 iO with Padding via a Uniform Random String (URS)

```
Parameters:
• Let $\ell$ be some padding parameter for the unobfuscated program.
• Let the polynomial $T$ describe a time bound on the runtime of the unobfuscated program.
• Let $\mathsf{Pad}(M, 1^\kappa)$ denote a program which is functionally equivalent to $M$ of size $|M| + \kappa$.
$\mathsf{pad}\mathcal{O}.\mathsf{Obf}(1^\lambda, M, \mathsf{pub}, 1^\ell, \mathsf{URS})$:
• Let $S = |M|$.
• Sample $k \leftarrow \mathsf{PPRF.Gen}(1^\lambda)$ such that $\mathsf{PPRF.Eval}(k, y) \in \{0,1\}$.
• Let $\mathsf{ct} = \{M[i] \oplus \mathsf{PPRF}(k, i)\}_{i \in [S]}$.
• Truncate URS to length $\ell - |M|$.
• Set $r = \mathsf{ct}||\mathsf{URS}$.
• Let $U_{k,S}$ be the machine that takes as public input $(\mathsf{pub}, r)$ and input $x$ and does the following:
    – Parse $r = (\mathsf{ct}, \mathsf{URS})$ by taking the first $S$ bits of $r$ to be ct.
    – Ignore URS, and compute $M \leftarrow \{\mathsf{ct}[i] \oplus \mathsf{PPRF.Eval}(k, i)\}_{i \in [S]}$.
    – Output $M[\mathsf{pub}](x)$.
  Moreover, pad the runtime of $U_{k,S}$ to match the runtime of the programs in the hybrids of Lemma VIII.2.
• Let $\Gamma = \mathsf{iO.Obf}(1^\lambda, \mathsf{Pad}(U_{k,s}, 1^\kappa), (\mathsf{pub}, r))$, where $\kappa = \mathsf{poly}(\lambda, \gamma)$ (where poly refer to a fixed polynomial from the proof analysis, and $\gamma$ is the PV-size parameter which we will choose in the proof).
• Output $(\Gamma, r)$.
```

Definition VIII.1 (ρ-closeness)**.** Let $M_1[\mathsf{pub}] = \{M_{1,\lambda}[\mathsf{pub}_\lambda]\}_\lambda$ and $M_2[\mathsf{pub}] = \{M_{2,\lambda}[\mathsf{pub}_\lambda]\}_\lambda$ be a pair of Turing machine families with split descriptions satisfying that for all x, $\mathsf{RT}(M_1[\mathsf{pub}], x) = \mathsf{RT}(M_2[\mathsf{pub}], x)$.

- We say that $M_1[\mathsf{pub}]$ and $M_2[\mathsf{pub}]$ are *adjacent* if $M_{1,\lambda}$ and $M_{2,\lambda}$ if they are functionally equivalent, and their descriptions only differ by a constant number of bits (not necessarily of the same size).
- We say that $M_1[\mathsf{pub}]$ and $M_2[\mathsf{pub}]$ are *close* if we can transition between them via a polynomial-size sequence of adjacent machines. That is, if there exists $\ell = \mathsf{poly}(\lambda)$ and $M'_1, \ldots, M'_\ell$ such that $M_1[\mathsf{pub}] = M'_1[\mathsf{pub}]$, $M_2[\mathsf{pub}] = M'_\ell[\mathsf{pub}]$ and for every i, $M'_i[\mathsf{pub}]$ and $M'_{i+1}[\mathsf{pub}]$ are adjacent.
- We say that $M_1[\mathsf{pub}]$ and $M_2[\mathsf{pub}]$ are ρ*-close* for a function $\rho(\lambda)$ if they are close via a sequence $M'_1, \ldots, M'_\ell$ such that $\max_i |M'_i| \le \rho$.

Lemma VIII.2. *Suppose one-way functions exist. Consider programs $M_0 = \{M_{0,\lambda}\}_\lambda$ and $M_1 = \{M_{1,\lambda}\}$ that are adjacent (i.e. differ in at most $\tau = O(1)$ bits). Then,*

$$\mathsf{pad}\mathcal{O}(1^\lambda, M_0, \mathsf{pub}, 1^\rho) \approx_c \mathsf{pad}\mathcal{O}(1^\lambda, M_1, \mathsf{pub}, 1^\rho)$$

for $\rho \ge \max(|M_0|, |M_1|)$. Moreover, the efficiency of the size of the program $\Gamma \leftarrow \mathsf{pad}\mathcal{O}(1^\lambda, M_0, \mathsf{pub}, 1^\rho)$ is as follows:

- *Assuming fully succinct IO, $|\Gamma| = \rho + \mathsf{poly}(\lambda, \tau, \log \ell)$.*
- *Assuming description-succinct IO, $|\Gamma| = \rho + \mathsf{poly}(\lambda, \tau, n, \log \ell)$, where n is a bound on the input length supported by Γ.*

Proof. The efficiency guarantees follow in a straightforward way from the efficiency guarantees of fully succinct/description-succinct IO. It suffices to show security.

Suppose we have two functionality equivalent adjacent machines $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ where $|M_b| = S_b$. Without loss of generality, suppose that $S_0 \ge S_1$. Suppose that the two machines differ on indices $i_1, \ldots, i_\tau \in [S]$. Let $I = \{i_j\}_{j \in [\tau]}$. Let $\beta_j^{(b)} = M_b[i_j]$, i.e. the i_jth bit of M_b. We use the short-hand $L_b = \{\beta_j^{(b)}\}_{j \in [\tau]}$.

$\boxed{\mathsf{H}_1}$ Sample $(\Gamma, r) \leftarrow \mathsf{pad}\mathcal{O}(1^\lambda, M_0, \mathsf{pub}, 1^\rho)$ as in Algorithm 1.

$\boxed{\mathsf{H}_2}$ Hardcode I, L_0, L_1 in the obfuscated program. This is clearly functionally equivalent (and in fact $\mathcal{EF}$-equivalent) to the previous hybrid because we have only added additional bits to the program. Hence, this follows from iO security.

$\boxed{\mathsf{H}_3}$ Change the obfuscated program to the following program:

U_{k,S_0,S_1,I,L_0,L_1} :
- Parse $r = (\mathsf{ct}, \mathsf{URS})$ by taking the first S_0 bits of r to be ct.
- For $i_j \in I$, set $M[i_j] = \beta_j^{(0)}$.
- For $i \notin I$, set $M[i] \leftarrow \mathsf{ct}[i] \oplus \mathsf{PPRF.Eval}(k, i)$.
- Output $M[\mathsf{pub}](x)$.

By construction, we know that for $i_j \in L$, $\beta_j^{(0)} = \mathsf{ct}[i_j] \oplus \mathsf{PPRF.Eval}(k, i_j)$. Therefore, this is functionally equivalent to the previous hybrid (and in fact $\mathcal{EF}$-equivalent) and indistinguishability follows from iO security.

$\boxed{\mathsf{H}_4}$ Puncture the key k on index set I, and hardcode $k\{I\}$ instead of k in the obfuscated program. This follows from the fact that the obfuscated program does not need $\mathsf{PPRF.Eval}(k, i_j)$ for $i_j \in I$, and by the correctness of the PPRF family. Therefore, this is functionally equivalent to the previous hybrid (and in fact $\mathcal{EF}$-equivalent by Lemma IV.12) and indistinguishability follows from iO security.

$\boxed{\mathsf{H}_5}$ For $i_j \in I$, switch $\mathsf{ct}[i_j]$ to $\mathsf{PPRF.Eval}(k, i_j) \oplus \beta_j^{(1)}$ if $i_j \le S_1$, and switch to a uniformly random bit if $i_j > S_1$. This follows from punctured key indistinguishability (note that the adversary only sees the puntured key $k\{I\}$).

$\boxed{\mathsf{H}_6}$ In this hybrid, we change the obfuscated program to use L_0 instead of L_1.

$U_{k\{I\},S_0,S_1,I,L_0,L_1}$:
- Parse $r = (\mathsf{ct}, \mathsf{URS})$ by taking the first $\boxed{S_1}$ bits of r to be ct.
- For $i_j \in I$ and $i_j \le S_1$, set $M[i_j] = \boxed{\beta_j^{(1)}}$.
- For $i \le S_b$, set $M[i] \leftarrow \mathsf{ct}[i] \oplus \mathsf{PPRF.Eval}(k, i)$.
- Output $M[\mathsf{pub}](x)$.

It is clear that the obfuscated program in the previous hybrid is functionally equivalent to M_0, and the program in this hybrid is functionally equivalent to M_1. Therefore, the two programs are functionally equivalent by the fact that $M_0[\mathsf{pub}] \equiv M_1[\mathsf{pub}]$. Hence, indistinguishability follows from iO security.

$\boxed{\mathsf{H}_7}$ Unpuncture the key k on index set I in the obfuscated program. Indistinguishability follows from the (PV) correctness of the puncturing.

$\boxed{\mathsf{H}_8}$ Sample $(\Gamma, r) \leftarrow \mathsf{pad}\mathcal{O}(1^\lambda, M_1, \mathsf{pub}, 1^\rho)$. This is indistinguishable from the previous hybrid by doing the hybrids in reverse and relying on IO security.

This completes the proof of security. □

Theorem VIII.3. *Consider two programs $M_0 = \{M_{0,\lambda}\}_\lambda$ and $M_1 = \{M_{1,\lambda}\}$ that are $\rho(\lambda)$-close. Then,*

$$\mathsf{pad}\mathcal{O}(1^\lambda, M_0, \mathsf{pub}, 1^\ell) \approx_c \mathsf{pad}\mathcal{O}(1^\lambda, M_1, \mathsf{pub}, 1^\ell)$$

for $\ell \geq \rho$. Moreover, the efficiency of the size of the program depends on the version of IO used, just as in Lemma VIII.2.

Proof. If M_0 and M_1 are $\rho(\lambda)$-close, by definition, then there exists a sequence $M'_1, M'_2, \ldots, M'_\ell$ for $\ell = \mathsf{poly}(\lambda)$ such that M'_i and M'_{i+1} are adjacent and $\max_i |M'_i| \leq \rho$. By Lemma VIII.2, we have that M'_i and M'_{i+1}, we know that they are adjacent, and by Lemma VIII.2,

$$\mathsf{pad}\mathcal{O}(1^\lambda, M'_i, \mathsf{pub}, 1^\rho) \approx_c \mathsf{pad}\mathcal{O}(1^\lambda, M'_{i+1}, \mathsf{pub}, 1^\ell).$$

Therefore, by a hybrid argument, we have the desired result. □

B. Applications to High-Rate Obfuscation

Rate-1 Null IO. Recall the notion of null-IO [34], [67], where we only require security for programs that are equivalent to the all-zeros program (note that this still requires correctness for all programs).

Definition VIII.4 (Null IO for Programs with Split Description)**.** We say that a IO scheme as in Definition V.1 is a null IO scheme if it satisfies the following weaker security guarantee: For every poly-size $\mathcal{A}$ and polynomial p, there exists a negligible function μ such that for every λ, M_0, M_1 and bounds $n \leq p(\lambda)$ such that

- $|M_0| = |M_1| \leq p(\lambda)$, and $|\mathsf{pub}| \leq p(\lambda)$.
- For every input x of length at most n, it holds that $M_0[\mathsf{pub}](x) = M_1[\mathsf{pub}](x) = 0$ and $\mathsf{RT}(M_0[\mathsf{pub}], x) = \mathsf{RT}(M_1[\mathsf{pub}], x)$.

For technical reasons, our rate-1 null IO result will only apply to programs $M[\mathsf{pub}]$ whose runtime $f(x) = \mathsf{RT}(M[\mathsf{pub}], x)$ has a small description. For simplicity, we will restrict our attention to polynomial TMs M for which there exists a polynomial q such that $\mathsf{RT}(M[\mathsf{pub}], x) = q(|x|)$. We will call such machines "runtime-regular". We say a null IO scheme is secure for runtime-regular programs if Definition VIII.4 satisfies security for pairs $M_0[\mathsf{pub}]$ and $M_1[\mathsf{pub}]$ which are runtime-regular.

Theorem VIII.5 (Rate-1 Null IO)**.** *Suppose one-way function exists. Then, there exists null IO for runtime-regular programs with the following efficiency: in the correctness experiment of Definition V.1,*

- *assuming fully succinct IO, there exists $|\widetilde{M}| = |M| + |\mathsf{desc}(p)| + \mathsf{poly}(\lambda)$, where p is the polynomial which describes the runtime of $M[\mathsf{pub}]$, and $\mathsf{desc}(p)$ is the size of the program which describes p.*
- *assuming description-succinct IO, $|\widetilde{M}| = |M| + |\mathsf{desc}(p)| + \mathsf{poly}(n, \lambda)$.*

We defer the construction and proof to the full version of the paper.

Batch-obfuscation. We now consider an IO scheme that takes a batch of $k \leq \mathsf{poly}(\lambda)$ programs $M_1, \ldots, M_k$, and obfuscates them simultaneously to obtain a program Π, with the property that $\Pi(i, x) = M_i(x)$. We say that the batch obfuscation scheme is secure if

$$\mathcal{O}(1^\lambda, 1^n, (M_1, \ldots, M_k)) \approx_c \mathcal{O}(1^\lambda, 1^n, (M'_1, \ldots, M'_k)).$$

for $k \leq \mathsf{poly}(\lambda)$, where for all i, $M_i \equiv M'_i$ and $|M_i| = |M'_i|$.

Theorem VIII.6. *Suppose one-way functions exist. Then, there exists a batch-IO that obfuscates $k \leq \mathsf{poly}(\lambda)$ programs $\Gamma \leftarrow \mathcal{O}(1^\lambda, 1^n, M_1, \ldots, M_k)$ with a following efficiency:*

- *Assuming fully succinct IO,*
$$\Gamma = \sum_{i \in [k]} |M_i| + \max_{i \in [k]} |M_i| + \mathsf{poly}(\lambda, \log k).$$
In particular, assuming that $|M_1| = |M_2| = \cdots = |M_k|$, we have $|\Gamma| = \frac{k+1}{k}|M_i| + \mathsf{poly}(\lambda)$.
- *Assuming description-succinct IO,*
$$\Gamma = \sum_{i \in [k]} (|M_i| + \max_{i \in [k]} |M_i| + \mathsf{poly}(\lambda, n, \log k),$$
where n is an upper bound on the input length supported by Γ.

We defer the proof to the full version of the paper.

C. Improving $\mathcal{EF}$-IO Rate

In this section, we show how to use Theorem VIII.3 to improve the rate of the our fully succinct $\mathcal{EF}$-IO construction in Theorem V.6.

Theorem VIII.7 (Fully-succinct pv-IO with Rate-1 in uniform $\mathcal{EF}$ Proof)**.** *Suppose the following assumptions hold:*

- *quasi-polynomial hardness of Learning with Errors,*
- *sub-exponentially secure one-way functions, and*

- *sub-exponentially secure indistinguishable obfuscation for circuits.*

Then, there is a (ρ, T)*-secure* $\mathcal{EF}$*-IO scheme satisfying that* $|\mathsf{iO}_{\rho,T}(1^\lambda, M, \mathsf{pub})| = \rho + 2|M| + \mathsf{poly}(\lambda)$.

We defer the proof to the full version of the paper.

Appendix

This section is taken nearly verbatim from [41].

Cook introduced a theory PV [20] to capture the intuition of feasibly constructive proofs (i.e. polynomial-time reasoning). PV is an equational theory, i.e, each statement in PV asserts that two terms are equal. Moreover, it allows the introduction of new function symbols by recursive definition (i.e. Cobham's definition of polynomial-time functions [19]). Hence, any polynomial-time function is definable in PV [20]. Moreover, commonly used arithmetical operations such as addition, multiplication, and modulus functions can also be defined in PV. Their related properties such as commutative law, associative law etc. can be proven in PV [15].

Formally, Cook's theory PV [20] is defined as follows. PV works on the natural numbers that are represented in the dyadic notation, where any natural number x is uniquely represented as a finite string of integers in $\{1,2\}^*$. Specifically, we represent x as the string $x_n x_{n-1} x_{n-2} \dots x_1 \in \{1,2\}^n$, if $\sum_{i=1}^n x_i 2^i = x$, and use an empty string to represent 0 . It's easy to see that such presentation is unique for any natural number. The function $s_i(x) = 2x + i, i = 1,2$ appends i to the string x. Hence, we also denote $s_i(x)$ as $x||i$.

We introduce the following terminologies. *Terms* are defined inductively as follows: any variable is a term; any function symbol of arity 0 is a term; if $t_1, t_2, \dots, t_k$ are terms, and f is a function symbol, then $f(t_1, t_2, \dots, t_k)$ is a term. *Equations* are of the form $t = u$, where both t and u are terms. A *derivation* for the statement $E_1, E_2, \dots, E_n \vdash_{\mathsf{PV}} E$ in PV is a series of equations $D_1, D_2, \dots, D_\ell$ such that $D_\ell = E$ and for any $i \in [\ell]$, the equation D_i is either a premise $E_j (j \in [n])$, or a defining equation for some function symbol that we will introduce later, or follows from some inference rule that we will introduce later. A *proof* in PV is a derivation with no premise $(n = 0)$.

Introducing Function Symbols. A *new function symbol* f can be introduced in PV in the following two ways.

- (*Composition*). The first way is to define

$$f(x_1, x_2, \dots x_k) = t,$$

where t is a term with variables $x_1, x_2, \dots, x_k$.

- *(Recursion).* The second way is to recursively define the function on the dyadic notion (i.e. Cobham's characterization of polynomial-time functions [19]). Specifically, for existing function symbols g, h_1, h_2, k_1, k_2 in PV, define the following equations as *defining equations*

$$f(0, \mathbf{y}) = g(\mathbf{y}), \tag{4}$$
$$f(x||i, \mathbf{y}) = h_i(x, \mathbf{y}, f(x, \mathbf{y})), i = 1,2, \tag{5}$$

where $\mathbf{y} = (y_1, \dots, y_k)$ is a series of k variables. Then how f is computed for any $x, \mathbf{y}$ is fully specified. To avoid any undecidable issue, PV further requires that the output length of f is bounded by a polynomial. To ensure this, Cook requires that "$|h_i(x, \mathbf{y}, z)| \le |z| + |k_i(x, \mathbf{y})|$" is provable in PV, where $|\cdot|$ is the length of the dyadic presentation. To achieve this, Cook introduced the LESS function, and it is defined with other *initial functions* as follows. $s_i, i = 1,2$ has no defining functions. 0 is also function symbol with arity 0, and has no defining function.
  - TR: $\mathsf{TR}(0) = 0, \mathsf{TR}(x||i) = x, i = 1,2$. It cuts off the least significant digit in the dyadic notion.
  - $\star$: $\star(x, 0) = x, \star(x, y||i) = s_i(x, y), i = 1,2$. It concatenates the string x and y.
  - $\circledast$: $\circledast(x, 0) = x, \circledast(x, y||i) = \star(x, \circledast(x, y)), i = 1,2$. It concatenates $|y|$ copies of x.
  - LESS : $\mathsf{LESS}(x, 0) = x, \mathsf{LESS}(x, y||i) = \mathsf{TR}(\mathsf{LESS}(x, y)), i = 1,2$. It cuts off the $|y|$ right most digits of x in the dyadic notion. Then we can use $\mathsf{LESS}(x, y) = 0$ to express $|x| \le |y|$.

  To complete the definition of function f, PV requires two proofs π_1, π_2 in PV for $\mathsf{LESS}(h_i(x, \mathbf{y}, z), z \star k_i(x, \mathbf{y})) = 0, i = 1,2$. Then a function symbol f is defined as the tuple $(g, h_1, h_2, k_1, k_2, \pi_1, \pi_2)$.

The *inference rules* are in the following. Here, t, u, v are any terms, x is any variable, and $\mathbf{y} = (y_1, y_2, \dots, y_k)$ is any tuple of $k \ge 0$ variables. f is any function symbol (we will define later).

- R_1: $t = u \vdash u = t$.
- R_2: $t = u, u = v \vdash t = u$
- R_3: $t_1 = u_1, t_2 = u_2, \dots, t_k = t_k \vdash f(t_1, t_2, \dots, t_k) = f(u_1, u_2, \dots, u_k)$.
- R_4: $t = u \vdash t(v/x) = u(v/x)$. Here, the notation "$t(v/x)$" means replacing each occurrence of the variable x with the term v. "$u(v/x)$" is defined in the same way.
- R_5: $E_1, E_2, \dots, E_6 \vdash f_1(x, \mathbf{y}) = f_2(x, \mathbf{y})$, where $E_1, E_2, \dots, E_6$ are the defining (5) for f_1, f_2, with

the same function symbols g, h_1, h_2.

PV **proofs for arithmetic.** The work of [15] shows that basic arithmetic operations such as $+, -, \cdot, /, \lfloor \cdot \rfloor, \leq$ can be introduced in Cook's theory PV [20] as function symbols, and their basic laws such as commutative law, associative law, distributive law, etc. can be proven in PV.

A. *Theory* PV_1

In the same work [20], Cook also introduced a theory PV_1 in which formalizing proofs is easier than PV. [20] showed that the theory PV_1 is a conservative extension of PV, which means that any theorem proven in PV_1 can also be proven in PV. Hence, in this work, we do not distinguish PV and PV_1.

The theory PV_1 contains all variables, function symbols, and terms in PV. Furthermore, it contains *formulae*, which are either equations, or truth-functional combinations of equations, using "$\wedge, \vee, \neg, \rightarrow, \leftrightarrow$", which express "and", "or", "negation", "imply", and "equivalent".

The axioms of PV_1 are defined as follows. Here, x is a variable, t, u, t_i, u_i are terms.

- E_1: $t = t$
- E_2: $t = u \rightarrow u = t$
- E_3: $t = u \wedge u = v \rightarrow t = v$
- E_4: $(t_1 = u_1 \wedge \ldots \wedge t_k = u_k) \rightarrow f(t_1, \ldots, t_k) = f(u_1, \ldots, u_k)$, where f is a function symbol in PV.
- E_5: $x = y \leftrightarrow x||i = y||i$, $i = 1, 2$
- E_6: $\neg(x||1 = x||2)$
- E_7: $\neg(0 = x||i)$, $i = 1, 2$
- **Defining Functions:** Defining equations for any function symbol in PV is axioms in PV_1. Moreover, PV_1 allows defining multi-variable functions via recursion as follows. Let g_{00}, g_{01}, g_{10}, $\{h_{ij}, k_{ij}\}_{i,j \in \{1,2\}}$ be function symbols. Then a new function symbol f can be defined by the following defining equations.

$$\begin{aligned} f(0, 0, z) &= g_{00}(z) \\ f(0, y||j, z) &= g_{01}(z), j = 1, 2 \\ f(x||i, 0, z) &= g_{10}(z), i = 1, 2 \\ f(x||i, y||j, z) &= h_{ij}(x, y, z, f(x, y, z)), i, j \in \{1, 2\} \end{aligned}$$

  Moreover, $\mathsf{LESS}(h_{ij}(x, y, z, u), u \star k_{ij}(x, y, z)) = 0, i, j \in \{1, 2\}$ needs to be provable in PV. Finally, the defining of the initial functions $\mathsf{TR}, \star, \circledast$, and LESS are also axioms of PV_1.
- **Tautology:** The truth-functionally valid formulae of PV_1 are axioms.

The inference rules of PV_1 are as follows.

- **Substitution.** $A \vdash A(t/x)$, where A is a formula in PV_1, t is a term and x is a variable. Here, we use $A(t/x)$ to denote the formula $A\sigma$, where σ is the substitution $\sigma : x \mapsto t$.
- **Implication.** $A_1, A_2, \ldots, A_k \vdash B$, where the formula B is a truth-functional implication of formulae $A_1, \ldots A_k$.
- **k-Induction.** $\{A(0/x_i)\}_{i \in [k]}, \{A \rightarrow A(x_1||j_1/x_1, \ldots, x_k||j_k/x_k)\}_{j_1, j_2, \ldots, j_k \in \{1,2\}} \vdash A$, where A is a formula of the variables $x_1, \ldots, x_k$.

References

[1] S. Agrawal, A. Modi, A. Yadav, and S. Yamada. Evasive LWE: Attacks, variants & obfustopia. Cryptology ePrint Archive, Paper 2025/375, 2025. 4

[2] P. Ananth, A. Jain, and A. Sahai. Indistinguishability obfuscation for Turing machines: Constant overhead and amortization. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 252–279. Springer, Cham, Aug. 2017. 5, 6, 19

[3] P. Ananth and A. Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Cham, Nov. 2018. 2, 8, 10

[4] B. Applebaum, A. Beimel, Y. Ishai, E. Kushilevitz, T. Liu, and V. Vaikuntanathan. Succinct computational secret sharing. In B. Saha and R. A. Servedio, editors, *55th ACM STOC*, pages 1553–1566. ACM Press, June 2023. 3, 5, 7, 14, 28

[5] S. Badrinarayanan, R. Fernando, V. Koppula, A. Sahai, and B. Waters. Output compression, MPC, and iO for Turing machines. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 342–370. Springer, Cham, Dec. 2019. 4

[6] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, Aug. 2001. 1

[7] C. H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. 12

[8] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, and E. Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, Oct. 2017. 4

[9] N. Bitansky, S. Garg, H. Lin, R. Pass, and S. Telang. Succinct randomized encodings and their applications. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015. 1, 8, 10

[10] N. Bitansky and V. Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In V. Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, Oct. 2015. 5

[11] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979. 4

[12] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, Aug. 2019. 7

[13] P. Branco, N. Döttling, A. Jain, G. Malavolta, S. Mathialagan, S. Peters, and V. Vaikuntanathan. Pseudorandom obfuscation and applications. In Y. T. Kalai and S. F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 663–698. Springer, Cham, Aug. 2025. 2, 4

[14] C. Brzuska, A. Ünal, and I. K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In K.-M. Chung and Y. Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 418–449. Springer, Singapore, Dec. 2024. 4

[15] S. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, Italy, 1986. 35, 36

[16] R. Canetti, J. Holmgren, A. Jain, and V. Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 429–437. ACM Press, June 2015. 1, 8, 10

[17] S. Chakraborty, M. Prabhakaran, and D. Wichs. Witness maps and applications. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 220–246. Springer, Cham, May 2020. 4, 14

[18] A. R. Choudhuri, A. Jain, and Z. Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, Feb. 2022. 10

[19] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*, pages 24–30. North-Holland Publishing, 1965. 35

[20] S. A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, page 83–97, New York, NY, USA, 1975. Association for Computing Machinery. 2, 3, 9, 20, 35, 36

[21] S. A. Cook and D. T. M. Le. Formalizing randomized matching algorithms. *Logical Methods in Computer Science*, 8, 2012. 31

[22] S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979. 3, 20

[23] L. Devadas, A. Jain, B. Waters, and D. J. Wu. Succinct witness encryption for batch languages and applications. To appear at Asiacrypt, 2025. 7

[24] N. Döttling, P. Gajland, and G. Malavolta. Laconic function evaluation for Turing machines. In A. Boldyreva and V. Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 606–634. Springer, Cham, May 2023. 2

[25] N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor hash functions and their applications. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Cham, Aug. 2019. 11, 24

[26] N. Döttling, A. Jain, G. Malavolta, S. Mathialagan, and V. Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. In Y. T. Kalai and S. F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 73–92. Springer, Cham, Aug. 2025. 4

[27] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. 31

[28] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, Oct. 2013. 1, 2

[29] S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. 2, 4, 26

[30] S. Garg and A. Srinivasan. Adaptively secure garbling with near optimal online complexity. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 535–565. Springer, Cham, Apr. / May 2018. 12

[31] S. Garg and A. Srinivasan. A simple construction of iO for Turing machines. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 425–454. Springer, Cham, Nov. 2018. 2, 8, 10, 12

[32] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. 4

[33] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, Oct. 1984. 22

[34] R. Goyal, V. Koppula, and B. Waters. Lockable obfuscation. In C. Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, Oct. 2017. 6, 34

[35] S. Halevi, Y. Ishai, A. Jain, E. Kushilevitz, and T. Rabin. Secure multiparty computation with general interaction patterns. In M. Sudan, editor, *ITCS 2016*, pages 157–168. ACM, Jan. 2016. 7

[36] P. Hall. On representatives of subsets. J. London Math. Soc. 10, 1935. 17

[37] J. Holmgren. On necessary padding with IO. Cryptology ePrint Archive, Report 2015/627, 2015. 3, 5

[38] Y.-C. Hsieh, A. Jain, and H. Lin. Lattice-based post-quantum iO from circular security with random opening assumption (part II: zeroizing attacks against private-coin evasive LWE assumptions). Cryptology ePrint Archive, Paper 2025/390, 2025. 4

[39] P. Hubacek and D. Wichs. On the communication complexity of secure function evaluation with long output. In T. Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, Jan. 2015. 10, 23

[40] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. IEEE Global Telecommunication Conf. (Globecom'87)*, pages 99–102, 1987. 4

[41] A. Jain and Z. Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd FOCS*, pages 1023–1034. IEEE Computer Society Press, Oct. / Nov. 2022. 2, 3, 4, 7, 16, 19, 20, 21, 22, 23, 35

[42] A. Jain, H. Lin, and J. Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 479–510. Springer, Cham, Apr. 2023. 5

[43] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from well-founded assumptions. In S. Khuller and V. V. Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021. 1

[44] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In O. Dunkelman and S. Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. 1

[45] A. Jain and S. Mathialagan. Canonical protocols for succinct cryptography via io and propositional logic. To appear soon, 2025. 3, 14

[46] Z. Jin, Y. Kalai, A. Lombardi, and V. Vaikuntanathan. SNARGs under LWE via propositional proofs. In B. Mohar, I. Shinkar, and R. O'Donnell, editors, *56th ACM STOC*, pages 1750–1757. ACM Press, June 2024. 3, 16

[47] Z. Jin, Y. T. Kalai, A. Lombardi, and S. Mathialagan. Universal SNARGs for NP from proofs of correctness. In M. Koucký and N. Bansal, editors, *57th ACM STOC*, pages 933–943. ACM Press, June 2025. 3, 4, 7

[48] Y. Kalai, A. Lombardi, V. Vaikuntanathan, and D. Wichs. Boosting batch arguments and RAM delegation. In B. Saha and R. A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023. 10, 11, 24

[49] Y. T. Kalai, O. Paneth, and L. Yang. How to delegate computations publicly. In M. Charikar and E. Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. 10

[50] I. Komargodski, M. Naor, and E. Yogev. Secret-sharing for NP. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume

8874 of *LNCS*, pages 254–273. Springer, Berlin, Heidelberg, Dec. 2014. 5, 7, 15, 28, 29

[51] V. Koppula, A. B. Lewko, and B. Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015. 1, 7, 10

[52] J. Li. An introduction to feasible mathematics and bounded arithmetic for computer scientists. *Electronic Colloquium on Computational Complexity (ECCC)*, 2025. 20, 25

[53] Q. Liu and M. Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In Y. Kalai and L. Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 138–169. Springer, Cham, Nov. 2017. 2

[54] G. Lu, S. Nassar, and B. Waters. Succinct computational secret sharing for monotone circuits. Cryptology ePrint Archive, Paper 2025/850, 2025. 7

[55] Z. Lu, N. Mazor, I. C. Oliveira, and R. Pass. Lower bounds on the overhead of indistinguishability obfuscation. Cryptology ePrint Archive, Report 2024/1524, 2024. 2, 5, 7

[56] Y. Ma, C. Dai, and E. Shi. Quasi-linear indistinguishability obfuscation via mathematical proofs of equivalence and applications. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 157–186. Springer, Cham, May 2025. 3, 7, 21

[57] S. Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, Nov. 1994. 4

[58] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, Jan. 1991. 16

[59] T. Okamoto, K. Pietrzak, B. Waters, and D. Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Berlin, Heidelberg, Nov. / Dec. 2015. 11

[60] S. Ragavan, N. Vafa, and V. Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In E. Boyle and M. Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 3–36. Springer, Cham, Dec. 2024. 1

[61] R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. In *22nd ACM STOC*, pages 287–292. ACM Press, May 1990. 5

[62] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In D. B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. 3, 4, 14

[63] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979. 4

[64] V. Vaikuntanathan, H. Wee, and D. Wichs. Witness encryption and null-IO from evasive LWE. In S. Agrawal and D. Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Cham, Dec. 2022. 4

[65] V. Vinod, A. Narayanan, K. Srinathan, C. P. Rangan, and K. Kim. On the power of computational secret sharing. In T. Johansson and S. Maitra, editors, *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 162–176. Springer, Berlin, Heidelberg, Dec. 2003. 5, 7

[66] B. Waters and D. J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In B. Mohar, I. Shinkar, and R. O'Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024. 4

[67] D. Wichs and G. Zirdelis. Obfuscating compute-and-compare programs under LWE. In C. Umans, editor, *58th FOCS*, pages 600–611. IEEE Computer Society Press, Oct. 2017. 6, 34

[68] A. C.-C. Yao. Unpublished manuscript. Presented at Oberwolfach and DIMACS workshops, 1989. 5, 7