

On Succinct Obfuscation via Propositional Proofs

(or: How to use pv-IO)

Abhishek Jain*

Zhengzhong Jin[†]

Surya Mathialagan[‡]

Omer Paneth[§]

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.

*Johns Hopkins University and NTT Research. abhishek.jain@ntt-research.com

[†]Northeastern University. zh.jin@northeastern.edu

[‡]NTT Research. surya.mathialagan@ntt-research.com

[§]Tel Aviv University. omerpa@tauex.tau.ac.il

Contents

1	Introduction	1
1.1	Our Results	2
1.1.1	Upgrading to Full Succinctness	3
1.1.2	Upgrading to Full IO security	4
1.1.3	Applications	5
1.2	Related work	8
1.3	Organization	9
2	Technical Overview	10
2.1	Upgrading to Full Succinctness	10
2.2	Upgrading to Full IO security	15
2.3	Overcoming the Representation-Size Barrier in Secret Sharing	17
2.3.1	A first construction	17
2.3.2	Our final construction	19
2.4	Padding with Public Randomness and High Rate Obfuscation	22
3	Preliminaries	24
3.1	Logic	25
3.1.1	Extended Frege EF	25
3.1.2	Theory PV	25
3.1.3	Uniform EF proofs of equivalence	26
3.2	Cryptographic Tools	27
3.2.1	Security Notions	27
3.2.2	Indistinguishability Obfuscation for Turing Machines	28
3.2.3	Input-succinct IO for uniform EF-equivalent Machines	28
3.2.4	PRGs and Puncturable PRFs	29
3.2.5	Bit commitments	30
3.2.6	Somewhere Statistically Binding Hash	32
4	Fully Succinct IO for EF-Equivalent Programs	38
4.1	Main Theorem	40
4.2	Construction	40
4.3	Proof of Theorem 4.5	42
4.3.1	Proof of Lemma 4.9	49
4.3.2	Proof of Lemma 4.10	54
4.4	Description-succinct IO	56
5	Equivalence between Succinct IO, Witness Encryption and SNARGs	58
5.1	Definitions	58
5.2	Main Theorem	61
5.2.1	From Succinct Witness Encryption to Succinct IO	61
5.2.2	From Succinct IO to Succinct Witness Encryption	63
5.2.3	From Succinct IO to Unique-SNARGs with Short CRS	64
5.2.4	From Unique-SNARGs with Short CRS to Succinct Witness Encryption	67

6	Succinct Computational Secret Sharing	68
6.1	Definitions	68
6.2	Secret Sharing Scheme for Functions with PV Proofs of Monotonicity	70
6.2.1	Proof of Lemma 6.8	74
6.3	Instantiations of Theorem 6.7	75
6.3.1	Monotone circuits	75
6.3.2	Bipartite matching access structure	76
6.4	Secret Sharing for NP via Fully Succinct IO	78
7	Necessary Padding and High-Rate Obfuscation	80
7.1	Construction and Analysis	80
7.2	Applications to High-Rate Obfuscation	83
7.3	Improving EF-IO Rate	85
A	Cook’s Theory PV	92
A.1	Theory PV_1	94
B	PV proofs of relevant constructions	95
B.1	Statistical Binding Property of SSB Hash Construction of [OPWW15], [KLVW23]	95
B.2	Correctness of Obfuscation Construction of [JJ22]	98
B.3	Correctness of Succinct Witness Encryption [BDJ ⁺ 25]	99
C	Deferred Proof from Section 5.2.4	100

1 Introduction

The notion of indistinguishability obfuscation (IO) [BGI⁺01, GGH⁺13] 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 [JLS21, JLS22, RVV24].

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 [BGL⁺15, CHJV15, KLV15] 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 arbitrary polynomial size. We refer to this notion as *input-succinct* IO.

Recent work [JJ22] 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 [Coo75]. From Gödel’s incompleteness theorem, however, it follows that there exist functionally equivalent programs without a PV proof of equivalence. Furthermore, prior works [JJ22, LZ17, GGSW13] 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 [GGH⁺13]: 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 $M[\text{pub}]$ 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 $M[\text{pub}]$ and $\widetilde{M}[\text{pub}]$ 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 $M_0[\text{pub}]$ and $M_1[\text{pub}]$ 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 [GS18a, AL18, DGM23]. Fully succinct constructions, however, are not known even for the weaker notion of pv-IO.

1.1 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 [BDJ⁺25] 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 the access structure, thus overcoming the so-called “representation-size barrier” [ABI⁺23]. 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

¹A recent concurrent work [LMOP24] 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 1.2 for more details.

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 [Hol15] 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 [SW14] 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 [JKLV24, JKLM25].

Next we elaborate on each of our contributions.

1.1.1 Upgrading to Full Succinctness

We start by introducing the notion of fully succinct pv-IO. The work of [JJ22] 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 [MDS25], we use an alternative formulation of pv-IO based on the more general notion of uniform 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 EF-equivalent [Coo75, CR79].

Let $M_0[\text{pub}]$ and $M_1[\text{pub}]$ 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 $\mathcal{M}_{\text{EF}}$ of size ρ that on input n, M_0, M_1, pub runs in time $T(n)$ and outputs an extended Frege proof that $\forall x \in \{0, 1\}^n, M_0[\text{pub}](x) = M_1[\text{pub}](x)$. In our formulation of pv-IO, the obfuscation algorithm is parameterized by ρ and T , and the security guarantee is that the obfuscations of $M_0[\text{pub}]$ and $M_1[\text{pub}]$ 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 $\text{poly}(T(n))$. 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 1.1 (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 [JJ22]. Therefore, we achieve fully succinct pv-IO under the same assumptions as known constructions of input-succinct pv-IO.

1.1.2 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 1.2. *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 uniqueness².*

In more detail, we show that:

- (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 each of the components of Theorem 1.2. 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 arguments for NP in the common reference string (CRS) model [Mic94, BCC⁺17]. 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 [CPW20]. 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 [JJ22, JKLM25], 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 [GGSW13] and SNARGs with unique proof [SW14], but an implication in the other direction is not known. Nonetheless, Theorem 1.2 states that starting from

²A 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 [FJM⁺26].

³I.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.

fully succinct pv-IO (that follows from sub-exponential IO for circuits and LWE with slightly super-polynomial 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 [GGSW13, GW11].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 [BDJ⁺25] 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 [VWW22, BÜW24, BDJ⁺25, DJM⁺25, HJL25, AMYY25], we are not aware of attacks to the WE construction itself. Thus, it may potentially be proven secure under a weaker assumption. By proving the correctness of the [BDJ⁺25] construction in theory PV (Appendix B.3), we can get a candidate construction of fully succinct IO.

Witness-succinct SNARGs with unique proofs are currently not known under standard cryptographic assumptions. However, [WW24] recently constructed adaptive SNARGs for NP with unique proofs assuming sub-exponential 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 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, [BFK⁺19] 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.

1.1.3 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 [Sha79, Bla79, ISN87], 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 $f : 2^{[n]} \rightarrow \{0, 1\}$. 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 .

⁴Roughly 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.

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 Section 1.2 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 [Yao89, VNS⁺03] 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, [ABI⁺23] improved the size of the public information to linear in the number of $\wedge$ gates in the circuit assuming either sub-exponential RSA or a combination of sub-exponential IO for circuits, and somewhere statistically binding hash.

As an application of fully succinct pv-IO (see Theorem 1.1), 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 1.3. *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 $\text{poly}(\lambda)$ and the public information is a uniform n -bit string.*

Our result is significantly more general: First, Theorem 1.3 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 6.9), but also structures for which no efficient monotone circuit exists such as matching in a bipartite graph [RW90]. 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 6.15).

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 [KNY14].

Theorem 1.4. *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 $\text{poly}(\lambda)$ and the public information is a uniform n -bit string.*

Previously, [KNY14] constructed secret sharing for all monotone functions in NP based on WE and one-way functions. Instantiating their scheme with fully succinct WE (instead of IO) yields succinct secret sharing that requires *structured* (as opposed to uniform) public information of size $n \cdot \text{poly}(\lambda)$.

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

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 [BV15, AJS17, JLL23] achieve rate $1/2$ for circuits and Turing machines (without input succinctness). Crossing the rate $1/2$ barrier for IO is an open question [AJS17, BV15, JLL23, LMOP24].
- **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. [Hol15] 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 $1/2$ 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.

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

We proceed to describe the notion of γ -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 $\ell = \text{poly}(\lambda)$ 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 γ -close for a function $\gamma(\lambda)$ if M'_1 and M'_ℓ are close where $M'_{i,\lambda}$ is the program $M_{i,\lambda}$ padded to size $\gamma(\lambda)$.

For example, observe that any pair of program families M_1, M_2 are γ -close for $\gamma(\lambda) = |M_{1,\lambda}| + |M_{2,\lambda}| + O(1)$: Start with M_1 padded to size γ . 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 γ -close programs. Using fully succinct IO, we show how to obfuscate pairs of programs that are γ -close such that the size of the obfuscated program is roughly γ . 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 $\gamma - \ell$ 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 1.5 (Informal). *Assuming fully succinct IO and one-way functions, there exists an obfuscation scheme with public randomness that is secure for every pair of γ -close machine families of size ℓ with $\gamma - \ell$ bits of public randomness, where the obfuscated programs are of size $\ell + \text{poly}(\lambda)$.*

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 [AJS17], this construction has the advantage that given public randomness of size ℓ , the obfuscated program is only $\text{poly}(\lambda)$ bits larger than the original program. Moreover, the construction from [AJS17] 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 1.5 to get higher rate. We give some concrete examples:

- The works of [WZ17, GKW17] 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 [AJS17]. In this setting, we achieve obfuscation of size $(k + 1) \cdot \ell + \text{poly}(\lambda)$ compared to $2k \cdot \ell + \text{poly}(\lambda, n)$ in [AJS17] where n is the bound on the input length to the TM. Moreover, given ℓ bits of public randomness, we can reduce the size of the obfuscation to $k \cdot \ell + \text{poly}(\lambda)$.
- In our construction of fully succinct pv-IO given by Theorem 1.1, to get security for (ρ, T) - $\mathcal{EF}$ equivalent programs the obfuscation size grows polynomially with ρ since, in the analysis, we hardcore the machine $\mathcal{M}_\Pi$ into the obfuscated program. Using Theorem 1.5, we can instead get an obfuscated program that does not grow with ρ using a public random string of size ρ .

1.2 Related work

Negative results in the circuit model. In a concurrent work, [LMOP24] 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 [LMOP24] 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 [LMOP24] the obfuscated circuit cannot even touch the entire public database. The result of [LMOP24] 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 [MDS25] 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 [JJ22] 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 [ABI⁺23] for a more detailed survey. As mentioned earlier, all prior schemes for polynomial-size monotone circuits [Yao89, VNS⁺03, ABI⁺23] 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 [ABI⁺23] 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 [DJWW25] 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 [HIJ⁺16, BCG⁺19]; 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 [KNY14] with a non-input-succinct IO for TMs [KLW15].

In a concurrent work, Lu, Nassar and Waters [LNW25] provide a different construction of succinct computational secret sharing for monotone circuits matching the efficiency of our scheme as stated in Theorem 1.3. 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 below n for some structures.

SNARGs from Propositional Proofs. A recent work of [JKLM25] (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.

1.3 Organization

We first give a technical overview in Section 2. In Section 3, we give an overview of definitions, tools and instantiations of cryptographic primitives which we will use throughout our work. In Section 4, we construct fully succinct pv-IO. In Section 5, we show a three-way equivalence between fully succinct (resp. witness-succinct) WE, fully succinct (resp. instance-succinct) SNARGs and

fully succinct (resp. input-succinct) IO. We then give our construction and analysis of succinct computational secret sharing in Section 6. We then show our “necessary padding” technique and applications to high-rate IO in Section 7. We defer PV proofs of existing constructions to the appendix.

2 Technical Overview

We provide an overview of the main ideas underlying our results. In Section 2.1, we describe how to upgrade input-succinct pv-IO to fully succinct pv-IO (Theorem 1.1). Our bootstrapping theorem from fully succinct pv-IO to fully succinct IO (Theorem 1.2) is described in Section 2.2. We discuss our applications to succinct secret sharing (Theorems 1.3 and 1.4) in Section 2.3 and to obfuscating γ -close programs (Theorem 1.5) in Section 2.4.

2.1 Upgrading to Full Succinctness

Let siO be an input-succinct pv-IO scheme. The idea behind our construction of fully succinct pv-IO is as follows: given a program $M[\text{pub}]$, use siO to obfuscate a program that takes as input both a public part pub' and an input x and outputs $M[\text{pub}'](x)$. To “bind” the obfuscation of the secret part to its public part, we hard-code into the obfuscated program a digest d_{pub} computed by applying a collision-resistant hash function with key hk on pub . If the public part pub' given as input to the obfuscated program does not hash to d_{pub} , the program outputs $\perp$. Overall, our obfuscator algorithm fsiO is as follows:

$\text{fsiO}(1^\lambda, M[\text{pub}]) :$

- Sample a hash key hk and set $d_{\text{pub}} = \text{Hash}(\text{hk}, \text{pub})$.
- Let $U_{M, \text{hk}, d_{\text{pub}}}$ be the program that, given as input pub' and x , outputs $M[\text{pub}'](x)$ if $d_{\text{pub}} = \text{Hash}(\text{hk}, \text{pub}')$ and $\perp$ otherwise.
- Output $\text{siO}(U_{M, \text{hk}, d_{\text{pub}}})$.

Since siO is input-succinct and the hash is shrinking, the new scheme is fully succinct. The main challenge is in arguing pv-IO security. Fix a pair of programs M_0, M_1 and a public input pub such that $M_0[\text{pub}]$ and $M_1[\text{pub}]$ are (ρ, T) -EF-equivalent. That is, they have the same size and running time, and there exists a TM $\mathcal{M}_{\text{EF}}$ of size ρ that on input n, M_0, M_1, pub runs in time $T(n)$ and outputs an extended Frege proof that $\forall x \in \{0, 1\}^n, M_0[\text{pub}](x) = M_1[\text{pub}](x)$. Note that while $M_0[\text{pub}]$ and $M_1[\text{pub}]$ are EF-equivalent, the programs $U_{M_0, \text{hk}, d_{\text{pub}}}$ and $U_{M_1, \text{hk}, d_{\text{pub}}}$ may not be. In fact, they may not even be functionally equivalent since there could exist some $\text{pub}' \neq \text{pub}$ such that $d_{\text{pub}} = \text{Hash}(\text{hk}, \text{pub}')$ but $M_0[\text{pub}']$ and $M_1[\text{pub}']$ are not functionally equivalent. Therefore, we cannot directly apply the security of siO .

Instead, we prove security by transitioning through a sequence of hybrids while relying both on the security of siO and on additional security properties of the hash function (introduced below). In what follows, we assume that the obfuscated program in each hybrid is padded such that the size and running time of the programs in all hybrids are the same. Additionally, in every transition that relies on the security of siO , we transition between two programs that are (ρ', T') -EF-equivalent for some suitable ρ' and T' . We will argue that, despite the padding and the overhead introduced by siO (as a function of ρ' and T'), the obfuscation remains fully succinct.

Step 1: Forcing EF-equivalence. Let P_0^b denote the program $U_{M_b, \text{hk}, d_{\text{pub}}}$. We start by considering a hybrid program P_1^b that given pub' , in addition to checking its digest, also checks that $M_0[\text{pub}']$ and $M_1[\text{pub}']$ are EF-equivalent. To this end, we consider a predicate R that, given as input n and pub' , runs the machine $\mathcal{M}_{\text{EF}}(n, M_0, M_1, \text{pub}')$ and verifies that it outputs a valid extended Frege proof that $\forall x \in \{0, 1\}^n, M_0[\text{pub}'](x) = M_1[\text{pub}'](x)$. Let P_1^b be the program that, given as input pub' and x , outputs $P_0^b(\text{pub}', x)$ if $R(|x|, \text{pub}') = 1$ and $\perp$ otherwise. It is easy to see that the programs P_0^b and P_1^b are (ρ', T') -EF-equivalent for $\rho' = \rho + O(1)$ and $T'(n) = O(T(n))$ and therefore $\text{siO}(P_0^b) \approx_c \text{siO}(P_1^b)$. It remains to show that $\text{siO}(P_0^b) \approx_c \text{siO}(P_1^b)$.

Intuitively, while the program P_0^b checks that the input pub' has the same digest as the “correct” input pub , the program P_1^b also checks that the output $R(n, \text{pub}')$ is correct (since $R(n, \text{pub}) = 1$). Our strategy is to transition from P_0^b to P_1^b by going through programs that check the correctness of intermediate configurations of the computation $R(n, \text{pub})$. In more detail, we show that for every input length n , and for every step i of the computation $R(n, \text{pub}')$, an obfuscated program checking that the digest of the configuration *before* step i is correct, is indistinguishable from the obfuscated program that is checking correctness of the digest of the configuration *after* step i .

Step 2: Isolating the input size. Fix $b \in \{0, 1\}$ and let $P_0 = P_0^b$ and $P_1 = P_1^b$. Next, we isolate the size of the input x by considering a sequence of hybrid programs P_2^n for $n \in [N(\lambda)]$ where $N(\lambda)$ is some super-polynomial upper bound on the size of the input x .⁶ Each P_2^n outputs $P_0(\text{pub}', x)$ if $|x| \geq n$ and $P_1(\text{pub}', x)$ otherwise. It remains to show that for every $n \in [N(\lambda)]$, $\text{siO}(P_2^n) \approx_c \text{siO}(P_2^{n+1})$.

Step 3: Isolating the R-step. Fix $n \in [N(\lambda)]$ and let $P_2 = P_2^n$ and $P_2' = P_2^{n+1}$. Additionally, redefine $R(n, \text{pub}')$ as $R(\text{pub}')$. Let t be the running time of R . For every $i \in \{0, \dots, t\}$, let cf_i be the i -th configuration in the execution of $R(\text{pub})$ and let $d_i = \text{Hash}(\text{hk}, \text{cf}_i)$. Let P_3^i be the following program that has the digest d_i hardcoded:

$P_3^i(\text{pub}', x) :$

- If $|x| \neq n$ output $P_2(\text{pub}', x)$.
- Execute $R(\text{pub}')$ and let cf'_i be the i -th configuration in the execution.
- Set $d'_i = \text{Hash}(\text{hk}, \text{cf}'_i)$.
- If $d_i = d'_i$ output $P_2(\text{pub}', x)$.
- Otherwise, output $\perp$.

Note that since cf'_0 (i.e., the initial configuration $R(\text{pub}')$) is independent of pub' , the programs P_3^0 and P_2 are EF-equivalent. Moreover, we assume WLOG that R has only two possible terminal configurations (accepting and rejecting) and that their digests are distinct.⁷ Therefore, since $R(\text{pub}) = 1$, we have that cf_t is accepting and the programs P_3^t and P_2' are EF-equivalent. It remains to show that for every $i \in [t]$, $\text{siO}(P_3^{i-1}) \approx_c \text{siO}(P_3^i)$.

⁶Our notion of obfuscation for TMs only guarantees correctness for inputs of size bounded by some super-polynomial function $N(\lambda)$. See Definition 3.5.

⁷We further assume that there is a uniformly generated extended Frege proof that for every λ , the two digests are distinct. This can be achieved by using a somewhere statistically binding hash function with an EF proof of binding as described below.

The hash function. To prove the above indistinguishability, we place two additional requirements on the hash function. First we require that the hash is computed along a tree. That is, on input $z \in \{0,1\}^L$, we identify each $j \in [L]$ with a string $j \in \{0,1\}^\ell$ for $\ell = \lceil \log L \rceil$, and for every $j \in \{0,1\}^{\leq \ell}$ we denote by $d_z[j]$ the value at location j in the hash tree. For every $j \in \{0,1\}^\ell$, we have that $d_z[j] = z_j$ and for every $j \in \{0,1\}^{<\ell}$, $d_z[j]$ is fixed by $d_z[j0]$ and $d_z[j1]$. The digest $d_z = \text{Hash}(\text{hk}, z)$ is just the value $d_z[\epsilon]$ at the root (where ϵ denotes the empty string).

Second, we require the hash to be *somewhere statistically binding* (SSB) [HW15]. That is, for every $j \in \{0,1\}^{\leq \ell}$, we can sample a hash key hk_j such that (i) hk_j is computationally indistinguishable from an honestly generated key hk , and (ii) the digest $d_z = \text{Hash}(\text{hk}_j, z)$ statistically binds $d_z[j]$. More generally, for every prefix j' of j and prefix j'' of j' , we require that $d_z[j'']$ statistically binds $d_z[j']$. Finally, we require that there is a uniformly generated extended Frege proof of binding, i.e.

$$\left(\begin{array}{l} d_z \leftarrow \text{Hash}(\text{hk}_j, z) \\ \wedge \alpha \leftarrow \text{Ext}(\text{td}_j, z) \end{array} \right) \rightarrow d_z[j] = \alpha$$

Step 4: A single R -step. Fix $i \in [t]$ and let $P_3 = P_3^{i-1}$ and $P'_3 = P_3^i$. Recall that P_3 checks that the $(i-1)$ -st configuration cf'_{i-1} has the correct digest, while P'_3 checks the digest of the i -th configuration cf'_i . We transition between P_3 and P'_3 through a sequence of programs that check the correctness of not only the digests, but also of individual bits of the two configurations as well as values in their hash trees. Since the programs must hardcode the correct value of each bit checked, we can only check a small number of locations simultaneously.

Intuitively, the SSB property of the hash allows us to transition from a program that is checking the digest of cf'_{i-1} to a program that is also checking the correctness of individual bits of cf'_{i-1} by making the digest bind the values of these bits one by one. Since each bit in cf'_i is fixed by a small number of bits in cf'_{i-1} , we can also transition to a program that is checking individual bits of cf'_i . To transition to P'_3 , however, we need to fix the digest of cf'_i which seems to require fixing all the bits of cf'_i simultaneously.

The solution is to rely on the tree structure of the hash. After fixing two adjacent bits of cf'_i , we can transition to a program that also checks the value of their parent in the hash tree which is fixed by the two bits. Then we again use the SSB property of the hash to transition to a program that is checking only the parent, but not the children by making the value of the parent bind the value of the children one by one.

By repeating this sequence of transitions, we reach a program where the root of the hash tree computed over cf'_i is fixed. Finally, we follow a similar strategy to remove the check of the digest of cf'_{i-1} , reaching the program P'_3 .

We proceed to describe the sequence of hybrids in more detail. Let $L = |\text{cf}_i|$ and $\ell = \lceil \log L \rceil$. Recall that for $i' \in \{i-1, i\}$, $\text{cf}_{i'}$ is the i' -th configuration in the execution of $R(\text{pub})$ and $d_{i'} = \text{Hash}(\text{hk}, \text{cf}_{i'})$. For every $j \in \{0,1\}^{\leq \ell}$, let $d_{i'}[j]$ be the value at location j in the hash tree rooted at $d_{i'}$.

For all sets $J, K \subseteq \{0,1\}^{\leq \ell}$, let $P_4^{J,K}$ be the program that is defined like P_3 , except that instead of checking d'_{i-1} , it checks the values in the hash tree over cf'_{i-1} in all the locations in J and the values in the hash tree over cf'_i in all the locations in K . In more detail, $P_4^{J,K}$ is the following program that has the digests $\{d_{i-1}[j]\}_{j \in J}$ and $\{d_i[k]\}_{k \in K}$ hardcoded:

$$P_4^{J,K}(\text{pub}', x) :$$

- If $|x| \neq n$ output $P_2(\text{pub}', x)$.
- Execute $R(\text{pub}')$ and let cf'_{i-1} and cf'_i be the $(i-1)$ -st and i -th configurations in the execution.
- Set $d'_{i-1} = \text{Hash}(\text{hk}, \text{cf}'_{i-1})$ and let $d'_{i-1}[j]$ be the value at location j in the hash tree.
- Set $d'_i = \text{Hash}(\text{hk}, \text{cf}'_i)$ and let $d'_i[k]$ be the value at location k in the hash tree.
- If $d_{i-1}[j] = d'_{i-1}[j]$ for every $j \in J$ and $d_i[k] = d'_i[k]$ for every $k \in K$, output $P_2(\text{pub}', x)$.
- Otherwise, output $\perp$.

Note that $P_4^{\{\epsilon\}, \emptyset}$ is EF-equivalent to P_3 and that $P_4^{\emptyset, \{\epsilon\}}$ is EF-equivalent to P'_3 . (Recall that $J = \{\epsilon\}$ corresponds to checking the digest $d'_{i-1} = d'_{i-1}[\epsilon]$ and $K = \{\epsilon\}$ corresponds to checking d'_i .) It remains to show that $\text{siO}(P_4^{\{\epsilon\}, \emptyset}) \approx_c \text{siO}(P_4^{\emptyset, \{\epsilon\}})$. We prove this in two steps:

1. $\text{siO}(P_4^{\{\epsilon\}, \emptyset}) \approx_c \text{siO}(P_4^{\{\epsilon\}, \{\epsilon\}})$.
2. $\text{siO}(P_4^{\{\epsilon\}, \{\epsilon\}}) \approx_c \text{siO}(P_4^{\emptyset, \{\epsilon\}})$.

Intuitively, in the first step, we transition from only checking the correctness of d'_{i-1} to checking both d'_{i-1} and d'_i . Then, in the second step, we transition from checking both d'_{i-1} and d'_i to checking only d'_i .

Step 4.1: Adding a check. To show that $\text{siO}(P_4^{\{\epsilon\}, \emptyset}) \approx_c \text{siO}(P_4^{\{\epsilon\}, \{\epsilon\}})$, we rely on the following claim: For any $J, K \subseteq \{0, 1\}^{\leq \ell}$ and $j, k \in \{0, 1\}^{\leq \ell}$:

1. $\text{siO}(P_4^{J, K}) \approx_c \text{siO}(P_4^{J \cup \{j\}, K})$ if $\epsilon \in J$ and $|j| = \ell$.
2. $\text{siO}(P_4^{J, K}) \approx_c \text{siO}(P_4^{J, K \cup \{k\}})$ if $\epsilon \in J$ and $|k| = \ell$.
3. $\text{siO}(P_4^{J, K}) \approx_c \text{siO}(P_4^{J, K \cup \{k\}})$ if $k0, k1 \in K$.
4. $\text{siO}(P_4^{J, K}) \approx_c \text{siO}(P_4^{J, K \setminus \{k0, k1\}})$ if $k \in K$.

To argue that $\text{siO}(P_4^{\{\epsilon\}, \emptyset}) \approx_c \text{siO}(P_4^{\{\epsilon\}, \{\epsilon\}})$, we apply the claim $O(L)$ times: For every $k \in \{0, 1\}^\ell$ in lexicographical order, we use a type 2 switch from $P_4^{\{\epsilon\}, K}$ to $P_4^{\{\epsilon\}, K \cup \{k\}}$. After every such transition, if $k0, k1 \in K$ for some k , we replace $k0, k1$ by k using a type 3 transition followed by a type 4 transition. We repeat this replacement until no such k exists. (Type 1 transitions are used to justify type 2 transitions.) Note that while the size of the program $P_4^{J, K}$ grows with $|J| + |K|$, in each of the above transitions the sets J and K are of size at most $\ell + O(1)$. Next, we outline the proof of the claim.

Type 1 transition. Fix $J, K \subseteq \{0, 1\}^{\leq \ell}$ such that $\epsilon \in J$ and fix $j \in \{0, 1\}^\ell$. Let $\tilde{P}_4^{J, K}$ and $\tilde{P}_4^{J \cup \{j\}, K}$ be the programs that are defined just like $P_4^{J, K}$ and $P_4^{J \cup \{j\}, K}$, respectively, except that instead of sampling the hash key hk honestly, we sample a hash key hk_j which is binding on index j . By the SSB requirement on the hash, there is a uniform EF proof that:

$$d_{i-1} = d'_{i-1} \rightarrow d_{i-1}[j] = d'_{i-1}[j].$$

Since $\epsilon \in J$, both $\tilde{P}_4^{J,K}$ and $\tilde{P}_4^{J \cup \{j\},K}$ explicitly check that:

$$d_{i-1} = d_{i-1}[\epsilon] = d'_{i-1}[\epsilon] = d'_{i-1} ,$$

and output $\perp$ if the check fails. Therefore, the two programs are EF-equivalent and $\text{siO}(\tilde{P}_4^{J,K}) \approx_c \text{siO}(\tilde{P}_4^{J \cup \{j\},K})$. Since $\text{hk} \approx_c \text{hk}_j$ we also have that:

$$\text{siO}(P_4^{J,K}) \approx_c \text{siO}(\tilde{P}_4^{J,K}) \approx_c \text{siO}(\tilde{P}_4^{J \cup \{j\},K}) \approx_c \text{siO}(P_4^{J \cup \{j\},K}) .$$

Type 2 transition. Fix $J, K \subseteq \{0, 1\}^{\leq \ell}$ such that $\epsilon \in J$ and fix $k \in \{0, 1\}^\ell$. Note that the k -th bit of the configuration cf'_i is fixed by one bit of the input pub' and a constant number of bits from the previous configuration cf'_{i-1} . Let $s \in [|\text{pub}'|]$ and $S \subseteq \{0, 1\}^\ell$ be such that the k -th bit of cf'_i is fixed by the bits pub'_s and $d'_{i-1}[j]$ for $j \in S$. By using one type 1 transition for every $j \in S \setminus J$ we get that $\text{siO}(P_4^{J,K}) \approx_c \text{siO}(P_4^{J \cup S, K})$.

Let $\tilde{P}_4^{J \cup S, K}$ and $\tilde{P}_4^{J \cup S, K \cup \{k\}}$ be the programs that are defined just like $P_4^{J \cup S, K}$ and $P_4^{J \cup S, K \cup \{k\}}$, respectively, except that instead of hk we sample a hash key hk_s . By the SSB requirement on the hash, there is a uniform EF proof that:

$$d_{\text{pub}} = \text{Hash}(\text{hk}_s, \text{pub}') \rightarrow \text{pub}_s = \text{pub}'_s .$$

Both $\tilde{P}_4^{J \cup S, K}$ and $\tilde{P}_4^{J \cup S, K \cup \{k\}}$ explicitly check that:

$$d_{\text{pub}} = \text{Hash}(\text{hk}, \text{pub}') , \forall j \in S, d_{i-1}[j] = d'_{i-1}[j] ,$$

and output $\perp$ if the check fails. Therefore, the two programs are EF-equivalent and $\text{siO}(\tilde{P}_4^{J \cup S, K}) \approx_c \text{siO}(\tilde{P}_4^{J \cup S, K \cup \{k\}})$. Since $\text{hk} \approx_c \text{hk}_s$ we also have that:

$$\text{siO}(P_4^{J \cup S, K}) \approx_c \text{siO}(\tilde{P}_4^{J \cup S, K}) \approx_c \text{siO}(\tilde{P}_4^{J \cup S, K \cup \{k\}}) \approx_c \text{siO}(P_4^{J \cup S, K \cup \{k\}}) .$$

By using one type 1 transition for every $j \in S \setminus J$ we get that $\text{siO}(P_4^{J \cup S, K \cup \{k\}}) \approx_c \text{siO}(P_4^{J, K \cup \{k\}})$. Overall we conclude that $\text{siO}(P_4^{J,K}) \approx_c \text{siO}(P_4^{J, K \cup \{k\}})$.

Type 3 transition. Fix $J, K \subseteq \{0, 1\}^{\leq \ell}$ and $k \in \{0, 1\}^{\leq \ell}$ such that $k0, k1 \in K$. Recall that $d'_i[k]$ is fixed by $d'_i[k0]$ and $d'_i[k1]$. Since $k0, k1 \in K$, both $P_4^{J,K}$ and $P_4^{J, K \cup \{k\}}$ explicitly check that $d_i[k0] = d'_i[k0]$ and $d'_i[k1] = d_i[k1]$ and output $\perp$ if the check fails. Therefore, the two programs are EF-equivalent and $\text{siO}(P_4^{J,K}) \approx_c \text{siO}(P_4^{J, K \cup \{k\}})$.

Type 4 transition. Fix $J, K \subseteq \{0, 1\}^{\leq \ell}$ and $k \in K$. Let $\tilde{P}_4^{J,K}$ and $\tilde{P}_4^{J, K \setminus \{k0\}}$ be the programs that are defined just like $P_4^{J,K}$ and $P_4^{J, K \setminus \{k0\}}$, respectively, except that instead of hk we sample a hash key hk_{k0} . By the SSB requirement on the hash, there is a uniform EF proof that:

$$d_i[k] = d'_i[k] \rightarrow d_i[k0] = d'_i[k0] .$$

Since $k \in K$, both $\tilde{P}_4^{J,K}$ and $\tilde{P}_4^{J, K \setminus \{k0\}}$ explicitly check that $d_i[k] = d'_i[k]$ and output $\perp$ if the check fails. Therefore, the two programs are EF-equivalent and $\text{siO}(\tilde{P}_4^{J,K}) \approx_c \text{siO}(\tilde{P}_4^{J, K \setminus \{k0\}})$. Since $\text{hk} \approx_c \text{hk}_{k0}$ we also have that:

$$\text{siO}(P_4^{J,K}) \approx_c \text{siO}(\tilde{P}_4^{J,K}) \approx_c \text{siO}(\tilde{P}_4^{J, K \setminus \{k0\}}) \approx_c \text{siO}(P_4^{J, K \setminus \{k0\}}) .$$

By following a similar sequence of hybrids and replacing $k0$ with $k1$ we get that $\text{siO}(P_4^{J,K \setminus \{k0\}}) \approx_c \text{siO}(P_4^{J,K \setminus \{k0,k1\}})$. Overall we conclude that $\text{siO}(P_4^{J,K}) \approx_c \text{siO}(P_4^{J,K \setminus \{k0,k1\}})$.

Step 4.2: Removing a check. It remains to show that $\text{siO}(P_4^{\{\epsilon\},\{\epsilon\}}) \approx_c \text{siO}(P_4^{\emptyset,\{\epsilon\}})$. To transition between the programs, the idea is to follow the same sequence of hybrids as in Step 4.1, but in reverse order and replacing the roles of the configurations cf_{i-1} and cf_i . There is one transition, however, that we cannot reverse: while every bit of cf_i is fixed by a constant number of bits from cf_{i-1}' (and the input pub'), there could be more than one configuration at step $i-1$ that lead to the same configuration at step i . If this is the case, removing the checks that $d_{i-1} = \text{Hash}(\text{hk}, \text{cf}_{i-1}')$ may alter the behavior of the program in a distinguishable way. To resolve this issue, we make the computation of R *reversible* [Ben89] such that every bit of cf_{i-1}' is, in fact, fixed by a constant number of bits from cf_i' .

2.2 Upgrading to Full IO security

In this section, we outline the proof of Theorem 1.2: Assuming fully succinct pv-IO and LWE we show an equivalence between fully succinct IO, WE and unique SNARGs, all with a PV-proof of correctness and (and, for the SNARGs, also a PV-proof of uniqueness). We also show a similar equivalence between input-succinct IO and witness-succinct WE and unique SNARGs.

The proof consists of three implications:

1. From unique SNARGs to WE assuming one-way functions.
2. From WE to IO assuming one-way functions and fully succinct pv-IO.
3. From IO to unique SNARGs assuming LWE.

To prove (1) we rely on the the work of [CPW20] that constructed WE from unique SNARGs and one-way functions. We observe that their construction preserves succinctness. That is, starting from a fully succinct (resp. witness-succinct) SNARGs, we get WE that is also fully succinct (resp. witness-succinct). Moreover, if the SNARG has a PV-proof of correctness and uniqueness then the WE will also have a PV-proof of correctness.

We proceed to describe the proof of (2) and (3).

From WE to IO. The proof of (2) is based on the following connection between IO and WE. We show that WE is equivalent to a relaxation of IO where the order of quantifiers in the security definition is switched: 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. We refer to this notion as *weak IO*.

In more detail, a weak IO for programs with split description is an algorithm wiO that takes as input a pair of machines (M_0, M_1) , public input pub , and a bit b . It outputs an obfuscated program $\widetilde{M}_b$ that is functionally equivalent to $M_b[\text{pub}]$. The security requirement is that for every $M_0[\text{pub}]$ and $M_1[\text{pub}]$ that are functionally equivalent, $\widetilde{M}_0$ and $\widetilde{M}_1$ are computationally indistinguishable, even given pub .

Theorem 2.1 (Informal). *Assuming fully succinct (resp. witness-succinct) WE, there exists a fully succinct (resp. input-succinct) weak IO. If the WE has a PV-proof of correctness, then so does the weak IO scheme.*

Our weak IO is constructed from a witness encryption scheme for the following language L :

$$L = \{(M_0, M_1, \text{pub}) \mid \exists x, M_0[\text{pub}](x) \neq M_1[\text{pub}](x)\},$$

Given (M_0, M_1) , pub and b the obfuscator computes a WE ciphertext ct encrypting the message b under the statement (M_0, M_1, pub) and outputs the obfuscated program $\widetilde{M}_{\text{ct}}$ that has M_0, M_1 and ct hardcoded:

$\widetilde{M}_{\text{ct}}[\text{pub}](x) :$

- Compute $z_0 = M_0[\text{pub}](x)$ and $z_1 = M_1[\text{pub}](x)$.
- If $z_0 = z_1$, output z_0 .
- Else, use x as a witness to decrypt ct , recover b and outputs z_b .

Correctness follows from the correctness of the WE scheme. For security, observe that if $M_0[\text{pub}]$ and $M_1[\text{pub}]$ are functionally equivalent, then $(M_0, M_1, \text{pub}) \notin L$ and, therefore, $\widetilde{M}_{\text{ct}}$ hides b . Finally, it is easy to verify that the construction preserves the succinctness properties of the WE.

From Weak IO to IO assuming pv-IO. To conclude the proof of (2) we construct fully succinct (resp. input-succinct) IO from the following ingredients:

- A fully succinct pv-IO fsiO .
- A fully succinct (resp. input-succinct) weak IO wiO with a PV-proof of correctness.

Our fully succinct IO scheme is simply the fully succinct pv-IO scheme (with sufficient padding). The fully succinct weak IO scheme is only used in the security proof.

Let $M_0[\text{pub}], M_1[\text{pub}]$ be a pair of functionally equivalent Turing machines. To argue security, we consider the following sequence of hybrids:

$$\begin{aligned} \text{fsiO}(M_0[\text{pub}]) &\approx_c \text{fsiO}(\text{wiO}((M_0, M_1), \text{pub}, 0)) \\ &\approx_c \text{fsiO}(\text{wiO}((M_0, M_1), \text{pub}, 1)) \approx_c \text{fsiO}(M_1[\text{pub}]). \end{aligned}$$

Where the first and last transitions are using the security of fsiO and the fact that $M_b[\text{pub}]$ and $\text{wiO}((M_0, M_1), \text{pub}, b)$ are PV-equivalent, and the middle transition is using the security of wiO .

From IO to unique SNARGs. The proof of (3) follows the construction in [SW14] of unique SNARGs from IO. We observe that starting from an input-succinct IO, we get a unique SNARG that is witness succinct. However, this construction does not preserve full succinctness.

Recall that in the construction of [SW14], the CRS of the SNARG contains an obfuscation of a program M that takes both the statement x and its witness as input. However, even if the IO is input-succinct, we will need to pad the obfuscated program so that its size grows with x . The reason is that in order to argue soundness for a particular false statement x^* , the reduction generates a CRS that obfuscates a functionally equivalent program M_{x^*} that has x^* hardcoded and given input $x = x^*$, it triggers an alternate execution path.

Our solution is to obfuscate a program that has $\text{Hash}(x^*)$ hardcoded instead of the entire input x^* , and trigger the alternate execution path on every input x such that $\text{Hash}(x) = \text{Hash}(x^*)$.⁸ While this program is no longer functionally equivalent to M , we can argue that their obfuscations are nonetheless indistinguishable, by a sequence of hybrids similar to the one used in the proof of Theorem 1.1 (see Section 2.1 for more details.)

2.3 Overcoming the Representation-Size Barrier in Secret Sharing

In this section, we give a technical overview of Theorems 1.3 and 1.4. 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 [ABI⁺23], 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.

We first give a construction whose share sizes are independent of $|f|$. We then reduce the public share to $n + \text{poly}(\lambda)$ bits, where n is the number of parties, and show how to improve this bound using short descriptions of unauthorized sets.

2.3.1 A first construction

We first give a simpler construction following the template of Komargodski, Naor and Yogev [KNY14]. The main ingredients in our construction are:

- A perfectly binding bit commitment scheme $(\text{Com}, \text{Open})$.
- A fully succinct pv-IO scheme pviO .

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

- Sample n commitments $c_i \leftarrow \text{Com}(1; r_i)$ (i.e. committing to 1, with randomness r_i).
- Let $P_s[c_1, \dots, c_n, f]$ be the following program:
 - **Input:** A set $J \subseteq [n]$ and commitment openings $\{\hat{r}_j\}_{j \in J}$.
 - For every $j \in J$, check that $c_j = \text{Com}(1; \hat{r}_j)$. If any check fails, output $\perp$.
 - If $f(J) = 1$, output s ; otherwise, output $\perp$.
- Let $\widetilde{P}_s \leftarrow \text{pviO}(P_s[c_1, \dots, c_n, f])$, where $c_1, \dots, c_n, f$ are treated as the public portion of the obfuscated program.
- Output $\text{crs} = ((c_1, \dots, c_n), \widetilde{P}_s)$ as the public share, and r_i as the share of party i .

Reconstruction algorithm: An authorized subset $I \subseteq [n]$ of parties uses $\{r_i\}_{i \in I}$ to compute $s \leftarrow \widetilde{P}_s[c_1, \dots, c_n, f](I, \{r_i\}_{i \in I})$.

By the full succinctness of the pv-IO scheme, it is easy to see that the size of the obfuscated program $\widetilde{P}_s$ does not grow with $|f|$.

⁸A previous version of this manuscript considered a weaker version of fully succinct SNARGs. The solution described here is inspired by ideas from the followup work of [FJM⁺26].

Template for security analysis. If the obfuscation used were a fully succinct IO rather than pv-IO, then the proof can be done as follows. For an unauthorized subset $I \subseteq [n]$ such that $f(I) = 0$, we argue security via the following hybrids.

- H₁** Sample all c_i for $i \notin I$ as commitments of 0 rather than 1. Indistinguishability follows from the hiding property of the commitment scheme.
- H₂** Since I is not an authorized set and the commitment scheme is perfectly binding, the programs $P_0[c_1, \dots, c_n, f]$ and $P_1[c_1, \dots, c_n, f]$ both output $\perp$ on every input. Therefore, by IO security, $\widetilde{P}_0$ and $\widetilde{P}_1$ are indistinguishable, thereby achieving security.

However, we can only rely on pv-IO security in the second step above, i.e. we now need to prove the equivalence between P_0 and P_1 in Theory PV.

Security analysis for natural monotone functions. We therefore restrict our analysis to classes of functions $\mathcal{F}$ for which monotonicity has a PV proof: for every $f \in \mathcal{F}$,

$$\forall I, J \subseteq [n], \quad (f(I) = 0 \wedge J \subseteq I) \rightarrow f(J) = 0.$$

We refer to such classes as *natural*. In particular, monotone circuits satisfy this requirement.

Additionally, we require that binding of the commitment scheme is provable in PV:

$$\left(\begin{array}{l} b \in \{0, 1\} \\ \wedge c \leftarrow \text{Com}(b; r) \end{array} \right) \rightarrow \forall r', c \neq \text{Com}(1 - b; r').$$

Such commitment schemes follow from public key encryption schemes which have a PV proof of correctness.

With the above properties, we can establish **H₂** by arguing in PV that $P_0[c_1, \dots, c_n, f]$ and $P_1[c_1, \dots, c_n, f]$ are equivalent. Recall that by the previous hybrid, we can write $c_i = \text{Com}(b_i; r_i)$, where $b_i = 1$ for $i \in I$, and 0 otherwise. We now argue equality by splitting the inputs into two cases:

Case (a): Inputs $J, \{\widehat{r}_j\}_{j \in J}$ where $J \not\subseteq I$,

- By construction, $c_j = \text{Com}(0; r_j)$ for $j \notin I$.
- By the binding property of the commitment scheme, for every $j \notin I$ and every $\widehat{r}_j$, we have $c_j \neq \text{Com}(1; \widehat{r}_j)$.
- If $J \not\subseteq I$, there exists $j \in J$ where $j \notin I$.
- The equality check $c_j = \text{Com}(1; \widehat{r}_j)$ will fail and both P_0 and P_1 will output $\perp$.

Case (b): Inputs $J, \{\widehat{r}_j\}_{j \in J}$ where $J \subseteq I$,

- By construction, $f(I) = 0$.
- By the monotonicity of f , for all $J \subseteq I$, $f(J) = 0$. Since f is natural, this can be proven in PV.
- Therefore, for all inputs $J, \{\widehat{r}_j\}_{j \in J}$ where $J \subseteq I$, the check $f(J) = 1$ will fail and both P_0 and P_1 will output $\perp$.

By Cook's propositional translation, the above PV argument gives a uniform machine $\mathcal{M}_{\text{EF}}$ generating extended Frege proofs of equivalence for every input length. A direct implementation hardcodes the characteristic vector of I and the commitment randomness $r_1, \dots, r_n$.

To reduce this description length, we introduce an additional hybrid, after switching the commitments outside I to 0, in which all commitment randomness is generated from a single short PRG seed sd . This change is indistinguishable by PRG security. The proof-generating machine can now reconstruct $r_1, \dots, r_n$ from sd , so it only needs to hardcode the n -bit characteristic vector of I and the seed. Its description therefore has size $n + \text{poly}(\lambda)$.

Thus, taking $\rho = n + \text{poly}(\lambda)$ and letting T bound the proof-generation time, we can apply pv-IO security to show that the obfuscations of $P_0[c_1, \dots, c_n, f]$ and $P_1[c_1, \dots, c_n, f]$ are indistinguishable in this hybrid. Full succinctness therefore gives $|\widetilde{P}_s| = \text{poly}(n, \lambda)$.

Let $\ell_{\text{com}}(\lambda)$ and $\ell_{\text{open}}(\lambda)$ denote the commitment and opening lengths, respectively. Consequently, the above construction yields a public share crs of size $n \cdot \ell_{\text{com}}(\lambda) + |\widetilde{P}_s| = \text{poly}(n, \lambda)$, and the individual share size is $\ell_{\text{open}}(\lambda) = \text{poly}(\lambda)$.

2.3.2 Our final construction

To achieve rate-1 dependence on n , we reduce both terms in the public-share bound $n \cdot \ell_{\text{com}}(\lambda) + |\widetilde{P}_s|$. We first hash the commitments so that they can be given to the individual parties. For the obfuscation, we replace the padding needed to describe I in the security proof by public randomness.

Reducing the public share. We first modify the obfuscated program so that it no longer takes all of the commitments $\{c_i\}_{i \in [n]}$ as public input. Instead, we compute a hash of all the commitments $\text{dig} \leftarrow \text{Hash}_{\text{hk}}(c_1, \dots, c_n)$, and hardcode both the hash key hk and the digest dig in the obfuscated program. For readability, we omit hk from the program subscripts below.

$P_{s, \text{dig}}[f](J, \{\widehat{c}_j, \widehat{\rho}_j, \widehat{r}_j\}_{j \in J}) :$

- For every $j \in J$, check that $\widehat{c}_j = \text{Com}(1; \widehat{r}_j)$ and $\widehat{\rho}_j$ is a valid opening under hk of dig to $\widehat{c}_j$ at index j . If any check fails, output $\perp$.
- If $f(J) = 1$, output s ; otherwise, output $\perp$.

The scheme is as follows.

Sharing algorithm:

- For every $i \in [n]$, sample randomness r_i and set $c_i \leftarrow \text{Com}(1; r_i)$.
- Sample a hash key hk , compute $\text{dig} \leftarrow \text{Hash}_{\text{hk}}(c_1, \dots, c_n)$, and let ρ_i be the opening of dig to c_i at index i .
- Publish a fully succinct pv-IO obfuscation $\widetilde{P}_{s, \text{dig}}$ of $P_{s, \text{dig}}$, and give (c_i, ρ_i, r_i) to party i .

Reconstruction algorithm: Parties in an authorized set J evaluate $\widetilde{P}_{s, \text{dig}}[f](J, \{(c_j, \rho_j, r_j)\}_{j \in J})$.

Each individual share still has size $\text{poly}(\lambda)$. The public share now consists only of the obfuscation, whose size depends on the description of I used in the security proof.

Since f is public, we can describe I by a program that computes its indicator vector from f . Let τ denote the description of this program together with any auxiliary input, and write $I = \tau(f)$, where the computation takes at most time t . Note that the description τ is used only in the security proof, and the sharing algorithm needs only a bound on its length.

Security after hashing. We can follow H_1 as before and replace all commitments c_i for $i \notin I$ with commitments of 0. However, the resulting programs $P_{0,\text{dig}}$ and $P_{1,\text{dig}}$ may not even be functionally equivalent: dig may admit openings to commitments different from $c_1, \dots, c_n$. To recover the previous security argument, we would like to check that each input commitment at an index outside I agrees with the commitment of 0 generated by the dealer.

As before, we can generate the commitment randomness from a single short PRG seed sd . To introduce these checks, we use a somewhere statistically binding hash with a PV proof of binding. We can then follow a sequence of hybrids using SSB index hiding and pv-IO security, as in Section 2.1, to reach an obfuscation of the following program.

$Q_{s,\text{dig},\tau,\text{sd}}[f](J, \{\hat{c}_j, \hat{\rho}_j, \hat{r}_j\}_{j \in J}) :$

- For every $j \in J$, check that $\hat{c}_j = \text{Com}(1; \hat{r}_j)$ and $\hat{\rho}_j$ is a valid opening under hk of dig to $\hat{c}_j$ at index j . If any check fails, output $\perp$.
- Compute $I \leftarrow \tau(f)$.
- Compute $r_1, \dots, r_n \leftarrow G(\text{sd})$.
- For $j \in J \setminus I$, if $\hat{c}_j \neq \text{Com}(0; r_j)$, output $\perp$.
- If $f(J) = 1$, output s ; otherwise, output $\perp$.

The same two-case argument now shows that $Q_{0,\text{dig},\tau,\text{sd}}$ and $Q_{1,\text{dig},\tau,\text{sd}}$ are PV-equivalent: inputs with $J \not\subseteq I$ fail a commitment check, while inputs with $J \subseteq I$ have $f(J) = 0$. However, the hybrid programs and their proof-generating machines still contain τ , so the obfuscation must be padded to accommodate its description.

Replacing padding by public randomness. To achieve rate-1 dependence on $|\tau|$, we rely on the padding ideas from Section 2.4 to replace this padding by public randomness. Let ℓ bound $|\tau|$. We add a uniformly random string $\text{pp} \in \{0, 1\}^\ell$ alongside f as public input to the obfuscation; the reconstruction program ignores it. These ideas allow the hybrid programs and their proof-generating machines to recover τ from the public input using only a short seed. Consequently, the resulting obfuscation $\tilde{P}$ has size $\text{poly}(\lambda)$, and publishing $(\text{pp}, \tilde{P})$ gives a public share of size $\ell + \text{poly}(\lambda)$. Since I can always be described using $n + O(1)$ bits, this gives the claimed rate-1 dependence on n . As usual, we suppress logarithmic terms in these size bounds and pad the running time to accommodate the computation of $\tau(f)$.

Rate-1 in Zero-Complexity. More generally, it suffices that $\tau(f)$ outputs a set $I' \supseteq I$ with $f(I') = 0$. The security argument then checks consistency only for commitments outside I' . With this observation, it is easy to see that we can achieve rate-1 dependence on a measure that we call the *zero-complexity* of f . For $x \in \{0, 1\}^n$ with $f(x) = 0$, we say that $y \in \{0, 1\}^n$ is a *zero-certificate* for x if $f(y) = 0$ and $y_i \geq x_i$ for every $i \in [n]$. Here, we are interested in zero-certificates that can be

generated from f by a small program. For a time bound t , let $K^t(y \mid f)$ denote the minimum total length of a program and auxiliary input that, given f , outputs y within time t . We define the t -time zero-complexity of f by

$$Z_{K^t}(f) := \max_{\substack{x \in \{0,1\}^n \\ f(x)=0}} \min_{\substack{y \succeq x \\ f(y)=0}} K^t(y \mid f).$$

For each unauthorized set I , choose τ to describe a program and auxiliary input witnessing $K^t(I' \mid f) \leq Z_{K^t}(f)$ for some zero-certificate $I' \supseteq I$. We can then achieve a public share of size

$$|\text{pp}| + |\tilde{P}| = \ell + \text{poly}(\lambda), \quad \ell = Z_{K^t}(f).$$

The final scheme. To obtain the public parameters in Theorem 1.3, we include a copy of $\tilde{P}$ in each party's share, giving $(c_i, \rho_i, r_i, \tilde{P})$ to party i , and publish only the uniform string $\text{pp} \in \{0,1\}^\ell$. An authorized set J reconstructs by evaluating $\tilde{P}[f, \text{pp}](J, \{(c_j, \rho_j, r_j)\}_{j \in J})$. Each individual share has size $\text{poly}(\lambda)$, while the public parameters have size $\ell = Z_{K^t}(f)$, with logarithmic terms suppressed as above.

Example 1: Monotone CNFs. Consider a monotone CNF

$$f(x) = \bigwedge_{\alpha=1}^k \bigvee_{j \in C_\alpha} x_j.$$

For every unauthorized set I , some clause C_α is disjoint from I . Let $I' = [n] \setminus C_\alpha$. Then $I' \supseteq I$ and $f(I') = 0$, so I' is a zero-certificate for I . Given the public description of f , the index α suffices to reconstruct I' . Thus the zero-complexity of f is at most $\lceil \log k \rceil + O(1)$, for a time bound sufficient to read f and output the characteristic vector of I' .

Example 2: 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 W can be described in $2m + O(1)$ bits. This yields a public share of size $2\sqrt{n} + \text{poly}(\lambda)$, improving the general bound of $n + \text{poly}(\lambda)$.

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 [Hal35], 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, $E \subseteq W$. 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 $N_E(S)$, which can be represented by a string of size $2m + O(1)$. This completes the claim.

Extension to monotone NP access structures. Suppose f is specified by a polynomial-time verifier R_f , where $f(J) = 1$ if and only if there exists a witness w such that $R_f(J, w) = 1$. We modify reconstruction to take w as an additional input and replace the test $f(J) = 1$ with $R_f(J, w) = 1$.

Using fully succinct IO, the same security argument applies without requiring PV proofs of equivalence. After introducing the consistency checks, any input that passes them satisfies $J \subseteq I'$, where $f(I') = 0$. Monotonicity implies $f(J) = 0$, so no witness w can pass the verifier. Thus both programs always output $\perp$, and we can invoke IO security. This gives the same rate-1 dependence on zero-complexity.

2.4 Padding with Public Randomness and High Rate Obfuscation

We give an overview of Theorem 1.5 and its applications to null IO, batch obfuscation, and improving the rate of the fully succinct pv-IO scheme from Theorem 1.1.

In IO-based constructions, we often pad the program being obfuscated to accommodate larger programs appearing in the security proof. We now show how this padding can be replaced by public randomness when the proof proceeds through small changes to the program description. The amount of padding is determined by the largest program in the sequence.

Recall that we define two functionally equivalent programs to be *adjacent* if their descriptions differ in only $O(1)$ bits. We call M_0 and M_1 γ -close if we can transition between them through a polynomial-length sequence of adjacent programs, each of size at most γ . Assuming fully succinct IO and one-way functions, we construct an obfuscator padO such that

$$\text{padO}(M_0, 1^\rho) \approx_c \text{padO}(M_1, 1^\rho) \quad \text{for every } \rho \geq \gamma.$$

The total size of $\text{padO}(M, 1^\rho)$ is $\rho + \text{poly}(\lambda)$, of which $\rho - |M|$ bits are public randomness. Thus, to obtain a good rate, it suffices to find a sequence whose intermediate programs are small. We first describe the construction and its security, and then give two examples of such sequences.

For simplicity, we consider programs without a public part and suppress logarithmic description-length terms. We pad the programs in each IO transition to have the same running time.

Construction. Let iO be a fully succinct IO scheme. We encrypt M bit-wise using a puncturable PRF and pad the ciphertext to length ρ with public randomness. We then use iO to obfuscate a program $U_{\ell,k}$ that takes this string as its public part, decrypts the first $\ell = |M|$ bits using the key k , and evaluates M .

$\text{padO}(M, 1^\rho, \text{URS})$, where $\rho \geq |M|$:

- Let $\ell = |M|$ and sample $k \leftarrow \text{PRF.Gen}(1^\lambda)$.
- Set $\text{ct}_i = M_{[i]} \oplus \text{PRF}_k(i)$ for every $i \in [\ell]$.
- Truncate URS to length $\rho - \ell$ and set $r = \text{ct} \parallel \text{URS}$.
- Let $U_{\ell,k}[r](x)$ be the following program:
 - Parse the first ℓ bits of r as ct and ignore the remaining bits.
 - Recover $M_{[i]} = \text{ct}_i \oplus \text{PRF}_k(i)$ for every $i \in [\ell]$.
 - Output $M(x)$.
- Output (r, Γ) , where $\Gamma \leftarrow \text{iO}(U_{\ell,k}, \text{pub} = r)$ and the secret description is padded to accommodate the security hybrids.

When URS is omitted, it is sampled uniformly at random. The public string r has length ρ , while the secret description of $U_{\ell,k}$ contains only k and ℓ . Full succinctness therefore gives the claimed total size $\rho + \text{poly}(\lambda)$. The padding contributes exactly $\rho - \ell$ public random bits, which the evaluator ignores.

Security. We first show security for adjacent programs M_0 and M_1 . Suppose they have the same length, and let I be the $O(1)$ locations where their descriptions differ. To change the ciphertext at I , we first modify the obfuscated program so that it uses the corresponding hardcoded bits of M_0 and no longer decrypts these positions. For the fixed public string r , this preserves functionality, so the change follows from IO security.

We can now puncture the PRF key at I and use punctured PRF security to change these ciphertext bits to encrypt the corresponding bits of M_1 . Next, we change the hardcoded bits to those of M_1 . This again follows from IO security, since M_0 and M_1 are functionally equivalent. Restoring ordinary decryption gives $\text{pad}\mathcal{O}(M_1, 1^\rho)$. Only $O(1)$ indices and plaintext bits are hardcoded, so the intermediate secret descriptions remain short. The full argument, including programs of different lengths, appears in Lemma 7.2.

For γ -close programs, we apply this argument at every step of the sequence, using the same padding parameter $\rho \geq \gamma$ throughout. Security then follows by a hybrid argument. In particular, the number of steps affects the security loss, while the largest intermediate program determines the padding needed.

Rate-1 null IO. Recall that null IO preserves functionality for all programs, but only guarantees indistinguishability for programs that output zero on every input. For this application, suppose the programs have a fixed common polynomial running time; Theorem 7.5 accounts for the cost of describing this running time.

Let M_0 and M_1 be two such programs of length ℓ . We can transition between them using only $O(1)$ additional space:

- Start from M_0 .
- Add a control bit that forces the output to zero.
- Replace the unused description of M_0 by M_1 , one bit at a time.
- Switch back to outputting $M_1(x)$ and remove the control bit.

Every program in the sequence still outputs zero and has size at most $\ell + O(1)$. Thus, the two programs are $(\ell + O(1))$ -close. Setting $\rho = \ell + O(1)$ gives total size $\ell + \text{poly}(\lambda)$, achieving rate 1.

High-rate batch obfuscation. Suppose we are given $k \leq \text{poly}(\lambda)$ programs $M_1, \dots, M_k$, each of length ℓ , and want an obfuscation that on input (i, x) outputs $M_i(x)$. Security requires that

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

whenever $M_i \equiv M'_i$ and $|M_i| = |M'_i|$ for every $i \in [k]$. This setting was considered in [AJS17].

Let $U_{M_1, \dots, M_k}(i, x) = M_i(x)$. To transition between the two batches, we replace one component at a time, reserving space for a single additional program. For the first component, we proceed as follows:

- Add a temporary copy of M'_1 to the program, one bit at a time.
- On inputs $(1, x)$, switch to evaluating the temporary copy. This preserves functionality since $M_1 \equiv M'_1$.
- Replace the unused description of M_1 by M'_1 , one bit at a time.
- Switch back to using the first copy of M'_1 .
- Erase the temporary copy one bit at a time.

The resulting program is $U_{M'_1, M_2, \dots, M_k}$. We reuse the same temporary space for each subsequent component. The largest intermediate description therefore contains only $k + 1$ programs, giving padding length $(k + 1)\ell$ up to the control information. Applying padO yields total size $(k + 1)\ell + \text{poly}(\lambda)$ and rate $k/(k + 1)$. Taking $k = 1$ gives rate-1/2 IO.

Improving the rate of pv-IO. Finally, we use the same padding scheme inside the construction of pv-IO. Here it suffices to instantiate padO using *description-succinct* IO. This gives total size $\rho + \text{poly}(\lambda, n)$, where n bounds the input length of the program being obfuscated, and allows us to replace the circuit IO used in JJ22 [JJ22, Figure 11].

Recall that JJ22 obfuscates a program $\widehat{M}$ that, on input (i, g) , generates the circuit for running M on inputs of length i and outputs the g th gate of its gate-by-gate obfuscation. In the security proof, $\widehat{M}$ must be padded to accommodate both M_0 and M_1 , together with a program $\mathcal{M}_{\text{EF}}$ that generates their uniform EF proofs of equivalence. Let ρ bound the description length of $\mathcal{M}_{\text{EF}}$, and let T bound its running time.

Replacing circuit IO with padO allows this padding to be replaced by public randomness. The remaining changes, such as puncturing PRF keys, contribute only $\text{poly}(\lambda)$ additional bits. As formalized in Theorem 7.7, assuming description-succinct IO and sub-exponential LWE, we obtain input-succinct (ρ, T) -EF-secure IO with

$$|\text{IO}_{\rho, T}(1^\lambda, M)| = \rho + 2|M| + \text{poly}(\lambda).$$

3 Preliminaries

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

Notation. We will use the following notation throughout this work.

- For $n \in \mathbb{N}$, we use $[n]$ to denote $\{1, \dots, n\}$.
- We use the notation $x \in \{0, 1\}^{\leq n}$ to denote that x is a bit-string of length at most n .
- We use the notation $|x|$ or $\text{desc}(x)$ to denote the bit-length of string x . We use $\text{desc}(x)$ 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 $\text{RT}(M, x)$ 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 , $\text{RT}(M, x) \leq t(|x|)$.

3.1 Logic

3.1.1 Extended Frege EF

In this work, we use the extended Frege (EF) system when discussing propositional logic. A propositional proof system is specified by a set of variables, Boolean connectives, and inference rules. Variables (denoted $x, y, z, \dots$) take values in $\{0, 1\}$, and Boolean formulas are built from variables using the connectives $\rightarrow, \neg, \wedge, \vee, \oplus, =$.

An inference rule is an expression of the form

$$A_1, A_2, \dots, A_k \vdash A_0,$$

where $A_0, A_1, \dots, A_k$ are formulas. Intuitively, this means that once $A_1, \dots, A_k$ have been derived, one may derive A_0 . If $k = 0$, the rule is called an axiom. We use the following axioms together with modus ponens:

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

A derivation is a finite sequence of formulas $\theta_1, \dots, \theta_\ell$ such that for each $i \in [\ell]$, either θ_i is an instance of one of the axioms, or there exist indices $j_1, \dots, j_k < i$ such that $\theta_{j_1}, \dots, \theta_{j_k} \vdash \theta_i$ is an instance of an inference rule. If $\theta_\ell = \varphi$, we say that the sequence is a proof of φ and write $\vdash \varphi$.

The extended Frege system additionally allows *extension axioms*. Namely, in a derivation $(\theta_1, \dots, \theta_\ell)$, a formula θ_i may also be of the form

$$t \leftrightarrow A,$$

where t is a fresh variable that does not occur in $\theta_1, \dots, \theta_{i-1}$ nor in A . The size of a proof $(\theta_1, \dots, \theta_\ell)$ is defined as $\sum_{i=1}^{\ell} |\theta_i|$.

3.1.2 Theory PV

Cook introduced theory PV [Coo75] to capture the intuition of feasibly constructive proofs (i.e. polynomial-time reasoning). In this work we use Cook's Theory PV to argue that a mathematical claim has a "uniform" EF proofs (Theorem 3.2).

The language of PV consists of *function symbols* representing basic polynomial-time computable functions, together with *terms*, which are expressions obtained by applying these symbols to variables. Concretely, every variable x (ranging over $\{0, 1\}^*$) 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 formulae of PV are simply *equations* of the form $t = u$, asserting that two such computations yield the same output. Proofs in PV are equational derivations from the defining axioms of the function symbols using the standard rules of equality (reflexivity, symmetry, transitivity, and substitution). Thus, the theorems of PV correspond to identities that hold between polynomial-time computable functions. For more details, we refer the reader to Appendix A.

Remark 3.1. Variables appearing in a PV equation are implicitly free. Thus, if one can prove $\vdash_{\text{PV}} t(x) = u(x)$ for terms t and u , then one can meta-mathematically interpret this 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 $\vdash_{\text{PV}} \forall x, t(x) = u(x)$. Note, however, that the universal quantifier is not part of the syntax for PV.

Theory PV_1 . In the same work, Cook also introduced theory PV_1 which includes truth-functional combinations of equations, using “ $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ ” [Coo75]. Moreover, Cook 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. We will primarily rely on PV_1 and will not distinguish between PV and PV_1 explicitly. See Appendix A.1 for further details.

Throughout this work, we will often write theorems of the form $\vdash_{\text{PV}} s \rightarrow (t = u)$, or equivalently, $s \vdash_{\text{PV}} (t = u)$. This should be interpreted as a PV_1 proof of the fact that $\vdash_{\text{PV}} \text{ITE}(s, t, u) = u$, where ITE is the function symbol corresponding to the if-then-else function:

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

As shown in [Li25], the function symbol ITE is definable in PV.

Propositional Translation. The main property of PV proofs that we rely on is that any proof in PV can be translated to a polynomial size EF proof [Coo75, CR79]. Let $t(x) = u(x)$ be a PV equation with PV proof Π . Then for $n \in \mathbb{N}$ we denote by $\llbracket t = u \rrbracket_n$ the EF formula:

$$t(x_1, \dots, x_n) = u(x_1, \dots, x_n).$$

Theorem 3.2 (Corollary of Extended Resolution Simulation Theorem in [Coo75]). *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 EF proof of $\llbracket t = u \rrbracket_n$ in $q(n)$ time.*

3.1.3 Uniform EF proofs of equivalence

Following the formulation of Ma, Dai and Shi [MDS25], in this work we consider pairs of programs M_0 and M_1 for which there exists a uniform EF-proof of equivalence. We formalize this as follows.

Definition 3.3 ($\mathcal{EF}$ -equivalent machines). Let $\rho \in \mathbb{N}$ 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 such that for every $n \in \mathbb{N}$, $\mathcal{M}_\Pi(n, M_0, M_1)$ outputs an EF proof of $\llbracket M_0(x) = M_1(x) \rrbracket_n$ in time $T(n)$.

Combining this with the translation theorem Theorem 3.2, we have that if $\vdash_{\text{PV}} \forall x, M_0(x) = M_1(x)$, then there exists ρ and a polynomial T such that M_0 and M_1 are (ρ, T) - $\mathcal{EF}$ -equivalent.

Establishing (ρ, T) - $\mathcal{EF}$ -equivalence of parameterized programs. In order to establish (ρ, T) - $\mathcal{EF}$ -equivalence between two machines which are parameterized, we will often use the following lemma. One can view this as a way to obtain uniform EF proofs for a pair of machines that have a PV proof of equivalence when given some advice aux .

Lemma 3.4. Consider PV definable TMs $M^{(0)}$ and $M^{(1)}$, and consider the family of machines $M_{\text{params}}^{(0)}(x) := M^{(0)}(\text{params}, x)$ and $M_{\text{params}}^{(1)}(x) := M^{(1)}(\text{params}, x)$. Suppose that there exists a PV definable function R such that the following is provable in PV:

$$\forall(\text{params}, \text{aux}, x), \quad R(\text{params}, \text{aux}) = 1 \vdash_{\text{PV}} M_{\text{params}}^{(0)}(x) = M_{\text{params}}^{(1)}(x).$$

Then, there exists some polynomial q such that the following holds. For any params and corresponding aux such that $R(\text{params}, \text{aux}) = 1$, we have that $M_{\text{params}}^{(0)}, M_{\text{params}}^{(1)}$ are $(|\text{aux}| + O(1), q')$ -EF-equivalent, where $q'(n) = q(|\text{aux}| + |\text{params}| + n)$.

Proof. By Theorem 3.2, there exists some machine $\mathcal{M}_{\Pi}$ which, on input a length bound for the tuple $(\text{params}, \text{aux}, x)$, outputs the corresponding EF proof in time polynomial in that bound. In particular, for fixed $\text{params}, \text{aux}$ and inputs $x \in \{0, 1\}^n$, this takes time $p(n + |\text{params}| + |\text{aux}|)$ for some polynomial p . Let p_R be the runtime of R . Let $q(n) = \text{poly}(p(n) + p_R(n))$, where poly is some fixed polynomial.

Consider some fixed params and corresponding aux such that $R(\text{params}, \text{aux}) = 1$. Then, we argue that $M_{\text{params}}^{(0)}, M_{\text{params}}^{(1)}$ are $(|\text{aux}| + O(1), q')$ -EF-equivalent for some polynomial $q'(n) = q(n + |\text{aux}| + |\text{params}|)$ as follows: Construct a machine $\mathcal{M}_{\text{EF}}$ which on input $n, M_{\text{params}}^{(0)}, M_{\text{params}}^{(1)}$,

- Prove in EF that $R(\text{params}, \text{aux}) = 1$. Recall that EF can prove the correctness of a deterministic computation with size polynomial in the runtime, e.g., by applying the Cook–Levin reduction to its computation tableau.
- Use Cook’s translation on the above PV proof for length inputs $(\text{params}, \text{aux}, x)$ of length $|\text{params}| + |\text{aux}| + n$, write an EF proof of the fact that for all $x \in \{0, 1\}^n$,

$$(R(\text{params}, \text{aux}) = 1) \rightarrow (M_{\text{params}}^{(0)}(x) = M_{\text{params}}^{(1)}(x)).$$

- By Modus ponens, deduce that for all $x \in \{0, 1\}^n$, $M_{\text{params}}^{(0)}(x) = M_{\text{params}}^{(1)}(x)$.

For appropriate choice of q , the above machine runs in time $q(n + |\text{aux}| + |\text{params}|)$. $\square$

3.2 Cryptographic Tools

3.2.1 Security Notions

We will use the parameters (s, ε) to describe the security parameters of our primitive. In particular, we say that a primitive is (s, ε) -secure if for all sufficiently large λ and all adversaries $\mathcal{A}$ of size $s(\lambda)$, the advantage of $\mathcal{A}$ is at most $\varepsilon(\lambda)$. For convenience, we use the following shorthand throughout.

- **Polynomial Security:** We say that a primitive is polynomially secure if it is $(s, \text{negl}(\lambda))$ -secure for all polynomials $s = s(\lambda)$. We denote this by saying it is polynomially-secure.
- **Super-polynomial Security:** We say that a primitive is Γ -secure for a super-polynomial function $\Gamma(\lambda)$ if for every polynomial s , it is $(s(\Gamma(\lambda)), \Gamma(\lambda)^{-O(1)})$ -secure. We denote this by saying it is Γ -secure.
- **Sub-exponential Security:** We say that a primitive is sub-exponentially secure if it is Γ -secure for some $\Gamma(\lambda) = 2^{\lambda^c}$, where $0 < c < 1$.

3.2.2 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 and an input length bound n . It outputs an obfuscation $\widetilde{M}$.

Definition 3.5. An indistinguishability obfuscation scheme iO for TMs satisfies the following properties:

Correctness: For every $\lambda, n \in \mathbb{N}, n \leq 2^\lambda$, machine M and input x such that $|x| \leq n$ and $\text{RT}(M, x) \leq 2^\lambda$, we have that

$$\Pr \left[\widetilde{M}(x) = M(x) : \widetilde{M} \leftarrow \text{iO}(1^\lambda, M, n) \right] = 1 .$$

We say that the scheme is N -correct for a function $N(\lambda) \leq 2^\lambda$ if the above only holds for $n \leq N(\lambda)$. We say that the scheme is polynomially-correct if it is N -correct for every $N = \text{poly}(\lambda)$.

Efficiency: In the correctness experiment above:

- The running time of iO is $\text{poly}(\lambda, |M|, n)$.
- There exists a polynomial q such that $\text{RT}(\widetilde{M}, x) = q(\lambda, \text{RT}(M, x))$.
- We say that the scheme is *input-succinct* if the running time of iO is $\text{poly}(\lambda, |M|)$.

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 $n \leq p(\lambda)$ such that

- $|M_0| = |M_1| \leq p(\lambda)$.
- For every input $x \in \{0, 1\}^{\leq n}$, it holds that $M_0(x) = M_1(x)$ and $\text{RT}(M_0, x) = \text{RT}(M_1, x)$.

we have that

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

For a function $\Gamma(\lambda) = \lambda^{\omega(1)}$, we say that the scheme is Γ -secure if the advantage above is at most $(\Gamma(\lambda))^{-O(1)}$ for all adversaries of size $\text{poly}(\Gamma)$.

Remark 3.6 (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). We additionally modify security to consider pairs of machines that agree on all inputs $x \in \{0, 1\}^{\leq n}$ for $n = 2^\lambda$ (or $n = N(\lambda)$) which is WLOG.

3.2.3 Input-succinct IO for uniform EF-equivalent Machines

In this section, we recap the result on IO for TMs of Jain and Jin [JJ22] (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 $\vdash_{\text{PV}} \forall x, M_0(x) = M_1(x)$. For convenience, in this work, we will instead follow the formulation of Ma, Dai and Shi [MDS25] that consider instead pairs of programs M_0 and M_1 for which there exists a uniform EF-proof of equivalence (Definition 3.3).

We define IO with EF-security: a relaxation of the security in Definition 3.5 that holds only for pairs of EF-equivalent programs instead of every pair of functionally equivalent programs. For convenience, we will only define this for input-succinct IO and drop the input-length bound n from the parameters of IO.

Definition 3.7 ((ρ, T) -EF security). Let $\rho(\lambda)$ and $T(\lambda, n)$ be polynomials. We say that an N -correct IO scheme iO satisfies (ρ, T) -EF security if for every poly-size $\mathcal{A}$ and polynomial p , there exists a negligible function μ such for every λ , M_0 and M_1 satisfying:

- $|M_0| = |M_1| \leq p(\lambda)$.
- For every input $x \in \{0, 1\}^{\leq N}$, $\text{RT}(M_0, x) = \text{RT}(M_1, x)$.
- M_0 and M_1 are $(\rho(\lambda), T(\lambda, \cdot))$ -equivalent.

we have that

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

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

Definition 3.8. (Input-succinct EF-IO) Let $N(\lambda) \leq 2^\lambda$ be a function. An N -correct EF-IO scheme is given by an algorithm $\text{iO}_{\rho, T}$ such that for every pair of polynomials $\rho(\lambda)$ and $T(\lambda, n)$, $\text{iO}_{\rho, T}$ is an N -correct input-succinct IO scheme with (ρ, T) -security, satisfying the following efficiency: In the correctness experiment described in Definition 3.5:

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

Theorem 3.9 ([JJ22, MDS25]). Let $N(\lambda) \leq 2^\lambda$ be a function. Assuming $\text{poly}(N)$ -hardness of LWE, sub-exponential hardness of one-way functions, and sub-exponential security of IO for circuits, there exists a N -correct input-succinct EF-IO scheme.

Remark 3.10 (Comparing Definition 3.8 with JJ). The work of JJ shows the above theorem in the case where $\vdash_{\text{PV}} M_0(x) = M_1(x)$. However, the size of the obfuscation grows with a more fine-grained measure which they call the “description complexity” ($\text{Desc}[\cdot]$) of the PV proof Π . We refer the reader to [JJ22, Section 7 and Appendix A] for more details. We choose to state the theorem instead in terms of uniform EF to simplify the theorem statement (this version of the theorem is implicit in JJ and made explicit in [MDS25, Section 7]). Moreover, in our proofs and applications, the programs in question will have PV proofs of equivalence Π which introduce only a constant number of function symbols, in which case we can rely on Theorem 3.2 to show that there exists a uniform EF proof of size $\text{poly}(\Pi)$.

3.2.4 PRGs and Puncturable PRFs

Definition 3.11 (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 $\mathcal{A}$,

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

Syntax. 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 (PPRF.Gen, PPRF.Punc, PPRF.Eval) with the following interface.

- PPRF.Gen(1^λ) : On input the security parameter 1^λ , outputs a key $k \in \mathcal{K}_\lambda$.
- 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 $k(\{S\}) \in \mathcal{K}_\lambda$.
- PPRF.Eval(k, x): On input $k \in \mathcal{K}_\lambda$ and $x \in \mathcal{X}_\lambda$, outputs some value $y \in \mathcal{Y}_\lambda$.

Definition 3.12 (Puncturable PRF). A (PPRF.Gen, PPRF.Eval, PPRF.Punc) family satisfies the following conditions:

Functionality preserving: For all sets $S \subseteq \mathcal{X}$ of size $|S| \leq s(\lambda)$, for all $x \notin S$,

$$\Pr[k \leftarrow \text{Gen}(1^\lambda), k\{S\} \leftarrow \text{Punc}(k, S) : \text{Eval}(k\{S\}, x) = \text{Eval}(k, x)] = 1.$$

Polynomial pseudorandomness at punctured points: Consider the following experiment $\text{EXP}_{\mathcal{A}}^R(b, \lambda)$ 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 $k \leftarrow \text{PPRF.Gen}(1^\lambda)$, and computes $k\{S\} \leftarrow \text{PPRF.Punc}(k, S)$.
 - If $b = 0$, compute $L_0 = \{x_i, \text{PPRF.Eval}(k, x_i)\}_{x_i \in S}$.
 - 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 $\text{Adv}_{\mathcal{A}}^R(\lambda) = |\Pr[W_0] - \Pr[W_1]|$. We say that the PPRF is pseudorandom at punctured points if for all PPT adversaries $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^R(\lambda) = \text{negl}[\lambda]$.

Definition 3.13 (Puncturable PRF with PV proof of Correctness, [JJ22]). We say a family of puncturable PRFs (PPRF.Gen, PPRF.Eval, PPRF.Punc) has a PV proof of correctness if Gen, Eval, Punc can be formalized as function symbols in PV, and there is a proof that:

$$\left(\begin{array}{l} S \subseteq \mathcal{X}_\lambda, |S| \leq s(\lambda) \\ \wedge x \notin S, x \in \mathcal{X}_\lambda \\ \wedge k \leftarrow \text{PPRF.Gen}(1^\lambda) \end{array} \right) \vdash_{\text{PV}} \left(\begin{array}{l} \text{PPRF.Eval}(\text{PPRF.Punc}(k, S), x) \\ = \text{PPRF.Eval}(k, x) \end{array} \right).$$

Lemma 3.14 ([JJ22, Lemma 14]). The GGM construction [GGM84] of puncturable PRFs has a PV proof of correctness.

3.2.5 Bit commitments

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

- Com.Gen is a probabilistic algorithm that takes as input a security parameter 1^λ , and outputs a common reference string crs.

- $\text{Com.Enc}(\text{pk}, m)$ 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 .
- $\text{Com.Ver}(\text{crs}, c, m, r)$ 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 3.15 (Commitment scheme in the CRS model). A tuple of PPT algorithms $(\text{Com.Gen}, \text{Com.Enc}, \text{Com.Ver})$ 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:

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

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

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

Computational Hiding: We say that the scheme is (s, ϵ) -secure if for all adversaries $\mathcal{A}$ of size $s(\lambda)$,

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

Definition 3.16 (Commitment scheme with PV proof of binding.). We say that the commitment scheme $(\text{Com.Gen}, \text{Com.Enc}, \text{Com.Ver})$ has a PV proof of binding if:

$$\vdash_{\text{PV}} \left(\begin{array}{l} b \in \{0, 1\}, \\ \text{pk} \leftarrow \text{Com.Gen}(1^\lambda), \\ c \leftarrow \text{Com.Enc}(\text{crs}, b; r) \end{array} \right) \rightarrow \forall r', 1 \neq \text{Com.Ver}(\text{crs}, c, 1 - b, r').$$

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

Lemma 3.17. Suppose there exists a public-key encryption scheme $(\text{PKE.Gen}, \text{PKE.Enc}, \text{PKE.Dec})$ with a PV-proof of correctness, i.e.

$$\vdash_{\text{PV}} \left(\begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{PKE.Gen}(1^\lambda; r_1) \\ \text{ct} \leftarrow \text{PKE.Enc}(\text{pk}, b; r_2) \\ b' \leftarrow \text{PKE.Dec}(\text{sk}, \text{ct}) \end{array} \right) \rightarrow b = b'.$$

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

Proof sketch. The construction is straightforward.

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

- $\text{Com.Ver}(\text{pk}, \text{ct}, b, r)$: Accept if $\text{ct} = \text{PKE.Enc}(\text{pk}, b; r)$.

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

Therefore, as shown in [JJ22], such a commitment scheme can be instantiated from LWE, or from DDH.

3.2.6 Somewhere Statistically Binding Hash

In this section, we recall the notion of statistical binding hash functions [HW15], 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 ($\text{Gen}, \text{TGen}, \text{Hash}, \text{Ver}, \text{Ext}$) with the following syntax:

$\text{Gen}(1^\lambda, 1^N, 1^s) \rightarrow \text{hk}$. On input a security parameter 1^λ , message length 1^N , and block size 1^s , outputs a hash key hk .

$\text{TGen}(1^\lambda, 1^N, 1^s, S \subseteq [N]) \rightarrow (\text{hk}^*, \text{td})$. On input a security parameter 1^λ , message length 1^N , block size 1^s , and subset S , outputs a hash key hk^* along with a trapdoor td .

$\text{Hash}(\text{hk}, \mathbf{x} \in \{0, 1\}^{sN}) \rightarrow \tau \text{ or } \perp$. On input a hash key hk and a string $\mathbf{x}$ parsed as N blocks of size s , output a hash value τ or a verifiable failure $\perp$.

$\text{Open}(\text{hk}, \mathbf{x}, i) \rightarrow \rho \text{ or } \perp$. On input a hash key hk , a string $\mathbf{x} \in \{0, 1\}^{sN}$ parsed as N blocks of size s , and an index $i \in [N]$, output a “local opening” ρ , or $\perp$ if hashing fails.

$\text{Ver}(\text{hk}, \tau, i, y, \rho) \rightarrow 0/1$. On input a hash key hk , hash value τ , index $i \in [N]$, block $y \in \{0, 1\}^s$ and opening ρ , the verification algorithm decides to accept or reject the local opening. It rejects failed hashes and openings.

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

Definition 3.18. A somewhere statistically binding (SSB) family ($\text{Gen}, \text{TGen}, \text{Hash}, \text{Ver}, \text{Ext}$) is required to satisfy the following properties:

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

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

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

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

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

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

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

Opening Completeness. For every supported message length N and block size s , every fixed message $\mathbf{x} \in \{0, 1\}^{sN}$ parsed as blocks $(x_1, \dots, x_N)$, and every index $i \in [N]$, consider the experiment of hashing $\mathbf{x}$ and verifying an honest opening of x_i . The key is sampled by $\text{Gen}(1^\lambda, 1^N, 1^s)$ or as the key component of $\text{TGen}(1^\lambda, 1^N, 1^s, S)$, for any fixed $S \subseteq [N]$. The message and index are chosen independently of the sampled key, and the probability is over key generation and any hashing and opening randomness.

Polynomial correctness. We say the family is polynomially correct if, for every polynomial p and $N, s \leq p(\lambda)$,

$$\Pr \left[\text{Ver}(\text{hk}, \tau, i, x_i, \rho_i) = 1 : \begin{array}{l} \tau \leftarrow \text{Hash}(\text{hk}, \mathbf{x}), \\ \rho_i \leftarrow \text{Open}(\text{hk}, \mathbf{x}, i) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

The negligible bound may depend on p , but is uniform over $N, s, \mathbf{x}, i, S$ and the key-generation mode.

Γ -correctness. Let $\Gamma(\lambda) = \lambda^{\omega(1)}$. We say the family is Γ -correct if, for every polynomial p and $N, s \leq p(\Gamma(\lambda))$,

$$\Pr \left[\text{Ver}(\text{hk}, \tau, i, x_i, \rho_i) = 1 : \begin{array}{l} \tau \leftarrow \text{Hash}(\text{hk}, \mathbf{x}), \\ \rho_i \leftarrow \text{Open}(\text{hk}, \mathbf{x}, i) \end{array} \right] \geq 1 - \text{negl}(\Gamma(\lambda)).$$

The uniformity of the bound is as above. Here $\text{negl}(\Gamma(\lambda))$ denotes a bound smaller than $\Gamma(\lambda)^{-c}$ for every constant $c > 0$ and all sufficiently large λ .

Extraction Correctness. For any subset $S \subseteq [N]$, any trapdoor key $(\text{hk}^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, 1^s, S)$, any hash τ , any index $i \in S$, any block $x_{i^*} \in \{0, 1\}^s$, and any proof ρ_{i^*} , we have

$$\Pr [\text{Ver}(\text{hk}, \tau, x_{i^*}, i^*, \rho_{i^*}) = 1 \Rightarrow \text{Ext}(\tau, \text{td})|_{i^*} = x_{i^*}] = 1.$$

Since the extracted value $\text{Ext}(\tau, \text{td})|_{i^*}$ is unique, the extraction correctness implies statistical binding property. If S contains a single element $i \in [N]$, we abuse notation and write i instead of $\{i\}$ for brevity.

Remark 3.19. We observe that the construction is a Merkle-tree based construction, and thus the input length N can in fact be *unbounded* (i.e. as large as 2^λ). That is, the running time of Gen , 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 .

Two-to-one SSB hashing. In this work, we rely on a two-to-one SSB (Two-SSB) family. At a high level, a Two-SSB is an SSB with the following additional properties:

- It hashes exactly two blocks of equal length s (i.e., it is the $N = 2$ case of an SSB hash).
- The length of the hash is $s + \text{poly}(\lambda, \log s)$.

Definition 3.20 (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:

$$\left(\begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Two-SSB.TGen}(1^\lambda, 1^s, b) \\ \wedge x_0, x_1 \in \{0, 1\}^s, \\ \wedge h \leftarrow \text{Two-SSB.Hash}(\text{hk}, (x_0, x_1)) \\ \wedge h \neq \perp \\ \wedge y = \text{Two-SSB.Ext}(\text{hk}, \text{td}, h) \end{array} \right) \vdash_{\text{PV}} x_b = y.$$

Theorem 3.21. Assuming *LWE*, there exists a two-to-one SSB hash family with a PV proof of the binding property (Definition 3.20).

The proof of the theorem can be found in Appendix B.1. The construction is a slight modification of that from the work of [DGI⁺19, OPWW15, KLVW23].

Tree-SSB In Section 4, we rely on a tree-like hash construction where the underlying hash is Two-SSB. First we describe the construction. Then, we show that the construction is indeed a SSB and state some additional useful properties of it.

We define the Tree-SSB hash with local opening here for blocks of size s . Leaves are numbered from 1, so the path to leaf j is encoded by the binary representation of $j - 1$. Hashing and opening return $\perp$ if any internal hash computation returns $\perp$; verification rejects failed or malformed openings. Thus opening correctness is asserted only when hashing succeeds.

Tree-SSB.Gen($1^\lambda, 1^s$) $\rightarrow$ hk: On input a security parameter 1^λ and block size 1^s :

1. Let $\ell_0 = s$.
2. Sample $\text{hk}_1 \leftarrow \text{Two-SSB.Gen}(1^\lambda, 1^{\ell_0})$, where hk_1 hashes two blocks, each in $\{0, 1\}^s$. Let ℓ_1 be the size of a hash generated by hk_1 .
3. For $i = 2, \dots, \lambda$:
 - Sample $\text{hk}_i \leftarrow \text{Two-SSB.Gen}(1^\lambda, 1^{\ell_{i-1}})$, which hashes two blocks, each of size ℓ_{i-1} .
 - Let ℓ_i be the size of a hash obtained by hashing two blocks of size ℓ_{i-1} with hash key hk_i .
4. We will use the short-hand hk to denote $(\text{hk}_1, \dots, \text{hk}_\lambda)$.
5. Output hk .

Tree-SSB.TGen($1^\lambda, 1^s, j \in [2^\lambda]$) $\rightarrow$ (hk, td):

1. Let $\ell_0 = s$.
2. Let $j_{\lambda-1} \dots j_0$ be the bit decomposition of $j - 1$, so that $j - 1 = \sum_{i=0}^{\lambda-1} 2^i j_i$.

3. For $i = 1, \dots, \lambda$, sample $(\text{hk}_i, \text{td}_i) \leftarrow \text{Two-SSB.TGen}(1^\lambda, 1^{\ell_{i-1}}, j_{i-1})$, where ℓ_i is as defined in Tree-SSB.Gen .
4. We will use the short-hand hk to denote $(\text{hk}_1, \dots, \text{hk}_\lambda)$ and $\text{td} = (\text{td}_1, \dots, \text{td}_\lambda)$.
5. Output (hk, td) .

$\text{Tree-SSB.Hash}_k(\text{hk} = (\text{hk}_1, \dots, \text{hk}_\lambda), x \in \{0, 1\}^{sk}) \rightarrow (\tau, k)$ or $\perp$:

1. If $k \notin [2^\lambda]$ or $|x| \neq sk$, output $\perp$.
2. If $k = 1$, output $(x, 1)$.
3. If k is not a power of 2,
 - Let $k' = 2^{\lceil \log_2 k \rceil}$.
 - Pad x with the fixed padding block 0^s to obtain x' of length k' .
4. If k is a power of 2, treat $x' = x$, and $k' = k$.
5. Compute a binary hash-tree T of height $\log_2 k' \leq \lambda$ on x' , where the leaf nodes are labeled with the values of x' . We call this level 0 of the tree.
6. Consider an internal node v at level i with children labeled with y_0 and y_1 . Label v with $\text{Two-SSB.Hash}(\text{hk}_i, (y_0, y_1))$. If this value is $\perp$, stop and output $\perp$ for the entire tree hash.
7. Let τ be the label of the root of the tree.
8. Output (τ, k) .

$\text{Tree-SSB.Open}(\text{hk}, x \in \{0, 1\}^{sk}, j \in [k]) \rightarrow \rho$ or $\perp$:

1. If $j \notin [k]$, output $\perp$. Compute the tree of Two-SSB values as in $\text{Tree-SSB.Hash}_k(\text{hk}, x)$, and output $\perp$ if hashing fails.
2. Let $\ell = \lceil \log_2 k \rceil$.
3. Let $\sigma_0, \sigma_1, \dots, \sigma_\ell$ be the Two-SSB values on the path from x_j to the root of the tree.
4. Let $\pi_0, \pi_1, \dots, \pi_{\ell-1}$ be the Two-SSB values of the siblings of the path from x_j .
5. Output $\rho = (\sigma_0, \dots, \sigma_\ell, \pi_0, \dots, \pi_{\ell-1})$.

$\text{Tree-SSB.Ver}(\text{hk}, (\tau, k), j, y, \rho) \rightarrow 0/1$: On input a hash key hk , hash value (τ, k) , index j , block $y \in \{0, 1\}^s$ and opening ρ :

- Reject if the key or digest is malformed, the digest or opening is $\perp$, $k \notin [2^\lambda]$, $j \notin [k]$, or $|y| \neq s$.
- Let $\ell = \lceil \log_2 k \rceil$ and parse $\rho = (\sigma_0, \dots, \sigma_\ell, \pi_0, \dots, \pi_{\ell-1})$.
- Reject if parsing fails, or if any path or sibling label is $\perp$ or has the wrong length for its level.
- Check $\sigma_0 = y$ and $\sigma_\ell = \tau$, and reject otherwise.
- Let $j_{\ell-1} \dots j_0$ be the ℓ -bit decomposition of $j - 1$ (the empty string if $k = 1$).

- For $i = 1, \dots, \ell$, check that σ_i is consistent with $\text{Two-SSB.Hash}(\text{hk}_i, \cdot)$ on σ_{i-1} and π_{i-1} . Concretely,
 - If $j_{i-1} = 0$, check if $\sigma_i = \text{Two-SSB.Hash}(\text{hk}_i, (\sigma_{i-1}, \pi_{i-1}))$.
 - If $j_{i-1} = 1$, check if $\sigma_i = \text{Two-SSB.Hash}(\text{hk}_i, (\pi_{i-1}, \sigma_{i-1}))$.

Reject if any check fails; otherwise accept.

$\text{Tree-SSB.Ext}(\text{hk}, \text{td}, (\tau, k)) \rightarrow \{0, 1\}^s \cup \{\perp\}$: On input a hash key hk , trapdoor td , and hash value (τ, k) :

- If the key, trapdoor, or digest is malformed, the digest is $\perp$, or $k \notin [2^\lambda]$, output $\perp$.
- Parse $(\text{hk}_1, \dots, \text{hk}_\lambda) \leftarrow \text{hk}$ and $(\text{td}_1, \dots, \text{td}_\lambda) \leftarrow \text{td}$.
- Let $\ell = \lceil \log_2 k \rceil$. Set $\tau_\ell := \tau$.
- For $i = \ell, \ell - 1, \dots, 1$: Recursively set $\tau_{i-1} \leftarrow \text{Two-SSB.Ext}(\text{hk}_i, \text{td}_i, \tau_i)$. If this value is $\perp$ or has the wrong length, output $\perp$.
- Output τ_0 .

Theorem 3.22. *The above Tree-SSB construction has succinct keys, hashes, and local openings, efficient verification, and index hiding inherited from Two-SSB. Its opening correctness holds conditional on hashing not returning $\perp$. If Two-SSB is polynomially correct (respectively, Γ -correct), then so is the opening of Tree-SSB: for every polynomial p , every $s, k \leq p(\lambda)$ (respectively, $s, k \leq p(\Gamma(\lambda))$) with $1 \leq k \leq 2^\lambda$, every fixed $x \in \{0, 1\}^{sk}$, and every $j \in [k]$ chosen independently of the key,*

$$\Pr \left[\text{Tree-SSB.Ver}(\text{hk}, \tau, j, x_j, \rho) = 1 : \begin{array}{l} \tau \leftarrow \text{Tree-SSB.Hash}_k(\text{hk}, x), \\ \rho \leftarrow \text{Tree-SSB.Open}(\text{hk}, x, j) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

(respectively, $1 - \text{negl}(\Gamma(\lambda))$). This bound holds for keys generated by Tree-SSB.Gen , by Tree-SSB.TGen at any fixed index, and with any fixed choice of Gen or either TGen mode at each independently sampled level. Additionally, if the underlying Two-SSB hash function has a PV proof of its binding property (Definition 3.20), then the following properties can be formalized in theory PV:

- **PV proof of opening correctness:** The opening correctness of the scheme can be proven in PV:

$$\left(\begin{array}{l} \text{hk} \leftarrow \text{SSB.Gen}(1^\lambda, 1^s) \\ \wedge 1 \leq j \leq k \leq 2^\lambda \\ \wedge x \in \{0, 1\}^{sk} \\ \wedge \tau \leftarrow \text{SSB.Hash}(\text{hk}, x) \\ \wedge \tau \neq \perp \\ \wedge \rho \leftarrow \text{SSB.Open}(\text{hk}, x, j) \end{array} \right) \vdash_{\text{PV}} \text{SSB.Ver}(\text{hk}, \tau, j, x_j, \rho) = 1$$

The same opening correctness statement holds for keys generated by Tree-SSB.TGen .

- **PV proof of binding property:** The statistical binding property can be proven in PV:

$$\left(\begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{SSB.TGen}(1^\lambda, 1^s, j) \\ \wedge 1 \leftarrow \text{SSB.Ver}(\text{hk}, \tau, j, y, \rho) \\ \wedge y' \leftarrow \text{SSB.Ext}(\text{hk}, \text{td}, \tau) \end{array} \right) \vdash_{\text{PV}} y = y'.$$

- **PV proof of extraction correctness:** *The extraction correctness can be proven in PV:*

$$\left(\begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{SSB.TGen}(1^\lambda, 1^s, j) \\ \wedge 1 \leq j \leq k \leq 2^\lambda \\ \wedge x \in \{0, 1\}^{sk} \\ \wedge \tau \leftarrow \text{SSB.Hash}(\text{hk}, x) \\ \wedge \tau \neq \perp \\ y \leftarrow \text{SSB.Ext}(\text{hk}, \text{td}, \tau) \end{array} \right) \vdash_{\text{PV}} y = x_j$$

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

Sketch. We refer the readers to [KLW23, OPWW15] for the security and efficiency analysis. We can achieve the PV proofs of opening, binding and extraction by following the same proof as was done in [JKLM25] for Merkle Trees.

□

4 Fully Succinct IO for EF-Equivalent Programs

In this section we construct a fully succinct IO for programs with uniform EF proofs of equivalence.

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) and an input x to the program and simulates (M, pub) on x . We use $M[\text{pub}](\cdot)$ to denote the machine $\mathcal{U}(M, \text{pub}, \cdot)$.

The syntax of IO for Turing machines with split description is the same as the syntax of IO for Turing machines given in Definition 3.5 except that we consider input programs $M[\text{pub}]$ that include both public and secret parts. The correctness and efficiency properties are also the same as Definition 3.5. 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 4.1 (IO for TMs for Programs with Split Description). An indistinguishability obfuscation scheme iO for $\mathcal{U}$ satisfies the following properties:

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

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

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 $\text{poly}(\lambda, |M[\text{pub}]|, n)$.
 - * We say that the scheme is *input-succinct* if the running time of iO is $\text{poly}(\lambda, |M[\text{pub}]|)$.
 - * We say that the scheme is *description-succinct* if $|\widetilde{M}| = \text{poly}(\lambda, |M|, n)$.
 - * We say the scheme is *fully succinct* if the running time of iO is $\text{poly}(\lambda, |M[\text{pub}]|)$ and $|\widetilde{M}| = \text{poly}(\lambda, |M|)$.
 - There exists a polynomial q such that $\text{RT}(\widetilde{M}[\text{pub}], x) = q(\lambda, \text{RT}(M[\text{pub}], x))$.
- **Security for split programs:** For every poly-size $\mathcal{A}$ and polynomial p , there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, M_0, M_1, pub and $n \leq p(\lambda)$,
 - $|M_0[\text{pub}]| = |M_1[\text{pub}]| \leq p(\lambda)$.
 - For every input $x \in \{0, 1\}^{\leq n}$, it holds that $M_0[\text{pub}](x) = M_1[\text{pub}](x)$ and $\text{RT}(M_0[\text{pub}], x) = \text{RT}(M_1[\text{pub}], x)$.

we have that

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

Just as in Remark 3.6, we will omit the input length n from the input to IO when the scheme is input-succinct or fully succinct.

IO with PV proof of correctness. We now define what it means for an IO scheme to have a PV proof of correctness. Following [Li25, Section 2.1.1], we introduce the PV symbol $\mathcal{U}_{\text{clock}}$ which on input $M[\text{pub}]$, x and 1^t , simulates $M[\text{pub}](x)$ for t steps, and outputs $M[\text{pub}](x)$ if the computation has halted, and $\perp$ otherwise.

Definition 4.2 (Indistinguishability obfuscation with PV proof of N -correctness). Let N be a PV definable function. We say that an indistinguishability obfuscation scheme iO has a PV proof of N -correctness if its correctness can be formalized in theory PV: there exists a PV definable polynomial q such that:

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

Fully succinct EF-IO for programs with split description. We now define a restriction of IO security to EF-equivalent programs, extending the definitions in Section 3.2.3 to the case of programs with split description.

For convenience, we will only define this for fully succinct IO and drop the input-length bound n from the parameters of IO.

Definition 4.3 ((ρ, T) -EF security for programs with split description). Let $\rho(\lambda)$ and $T(\lambda, n)$ be polynomials. We say that an N -correct IO scheme iO satisfies (ρ, T) -EF security if for every polynomial-sized adversary $\mathcal{A}$ and polynomial p , there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, $M_0[\text{pub}]$ and $M_1[\text{pub}]$ satisfying:

- $|M_0[\text{pub}]| = |M_1[\text{pub}]| \leq p(\lambda)$.
- For every input $x \in \{0, 1\}^{\leq N}$, $\text{RT}(M_0[\text{pub}], x) = \text{RT}(M_1[\text{pub}], x)$.
- $M_0[\text{pub}]$ and $M_1[\text{pub}]$ are $(\rho(\lambda), T(\lambda, \cdot))$ -EF-equivalent.

we have that

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

For $\Gamma = \lambda^{\omega(1)}$, we say the scheme is Γ -secure if the advantage above is at most $(\Gamma(\lambda))^{-O(1)}$ for all adversaries of size $\text{poly}(\Gamma)$.

Definition 4.4 (Fully succinct EF-IO). Let $N(\lambda) \leq 2^\lambda$ be a function. An N -correct EF-IO for programs with split description is given by an algorithm $\text{iO}_{\rho, T}$ such that for every pair of polynomials $\rho(\lambda)$ and $T(\lambda, n)$, $\text{iO}_{\rho, T}$ is an N -correct input-succinct IO scheme with (ρ, T) -security, satisfying the following efficiency. In the correctness experiment described in Definition 4.1:

- The running time of $\text{iO}_{\rho, T}(1^\lambda, M[\text{pub}])$ is $\text{poly}(\lambda, |M[\text{pub}]|, \rho(\lambda))$ and the size of the obfuscation is $\text{poly}(\lambda, |M|, \rho(\lambda))$.
- There exists a polynomial q such that $\text{RT}(\widetilde{M}[\text{pub}], x) = q(\lambda, \text{RT}(M[\text{pub}], x), \rho(\lambda), T(\lambda, |x|))$.

4.1 Main Theorem

Theorem 4.5. *Let N, Γ, S be functions such that:*

$$\lambda \leq N(\lambda) \leq \Gamma(\lambda) \leq 2^\lambda \quad , \quad S(\lambda) = \lambda^{\omega(1)} \quad .$$

Assuming

- *a Γ -secure, $(N + S)$ -correct input-succinct EF-IO (as in Definition 3.8),*
- *a Γ -secure, Γ -correct two-to-one somewhere statistically binding hash family with a PV proof of binding (Definitions 3.18 and 3.20),*

there exists a Γ -secure, N -correct fully succinct EF-IO. Moreover, if the underlying EF-IO scheme has a PV proof of $(N + S)$ -correctness, then the resulting fully succinct EF-IO scheme has a PV proof of N -correctness.

By instantiating the above theorem with Theorem 3.9 (which we argue in Appendix B.2 has a PV proof of its correctness property), we have the following corollary.

Corollary 4.6. *Assuming*

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

there exists a sub-exponentially secure fully succinct EF-IO scheme. Moreover, the scheme has a PV proof of correctness for every PV-definable function N .

4.2 Construction

We now present the construction of the fully succinct IO for Theorem 4.5.

Ingredients. Let $N'(\lambda) = N(\lambda) + S(\lambda)$. We rely on the following ingredients:

- Let pviO be an N' -correct, Γ -secure EF-IO scheme.
- A Γ -secure, Γ -correct two-to-one somewhere statistically binding hash family Two-SSB with a PV proof of binding, as in Definitions 3.18 and 3.20.
 - Given a Two-SSB, one can construct the Tree-SSB scheme Tree-SSB via a standard Merkle-tree style aggregation (following [OPWW15, KLVW23], explicitly detailed in Section 3.2.6). Since our proof will require this transformation white-box, we have included the details of the Tree-SSB construction in Section 3.2.6. For readability, we state our construction itself will only use the Tree-SSB.Gen, Tree-SSB.TGen, Tree-SSB.Hash and Tree-SSB.Ext algorithms (rather than the Two-SSB). Throughout this section, whenever the block size is clear from context, we suppress the 1^s block-size argument from Tree-SSB.Gen and Tree-SSB.TGen (and similarly from Two-SSB.Gen and Two-SSB.TGen).

We assume without loss of generality that the above Tree-SSB Gen algorithm only requires $\text{poly}(\lambda, \log(\Gamma(\lambda)))$ bits of randomness (e.g. by composing with a PRG to shrink the randomness). We analyze hash failures with independently sampled level keys; any replacement of their randomness by PRG output is accounted for separately as a computational hybrid.

We can now give our main construction.

Algorithm 1 N -correct fully succinct IO construction for (ρ, T) -EF-equivalent programs.

On input $1^\lambda, M, \text{pub}$:

Generate hash key: Sample $\text{hk} \leftarrow \text{Tree-SSB.Gen}(1^\lambda, 1)$.

Compute obfuscation:

1. Let $h = \text{Tree-SSB.Hash}(\text{hk}, \text{pub})$. If $h = \perp$, output $(\text{hk}, \perp, M)$.
2. Let $M^{\text{hk}, h}$ be the machine that takes as input $y \in \{0, 1\}^{|\text{pub}|}$, and an input $x \in \{0, 1\}^{\leq N}$:
 - Check if $\text{Tree-SSB.Hash}(\text{hk}, y) = h$, and abort if this does not hold.
 - Output $M[y](x)$.

Pad $M^{\text{hk}, h}$ to be an equivalent machine of size $|M^{\text{hk}, h}| + |M| + \rho + \text{poly}(\lambda)$, with runtime on each input x padded by $P(|x|, \lambda) = \text{poly}(T(|x|), \lambda)$, where the exact poly is chosen based on the sizes and runtimes of the programs in the hybrids of our security analysis.

3. Let $\rho' = \text{poly}(\lambda)$ and $T' = \text{poly}(T, \lambda)$, where the exact poly is chosen based on the hybrids in our security analysis.
4. Let pviO be an N' -correct and Γ -secure EF-IO scheme for parameters (ρ', T') .
5. Let $\Pi \leftarrow \text{pviO}(M^{\text{hk}, h})$.
6. Output (hk, h, Π) .

Evaluation: If $h = \perp$, run $M[\text{pub}](x)$; otherwise, run Π on input (pub, x) .

Correctness. If $h = \perp$, correctness is immediate since evaluation runs $M[\text{pub}](x)$ directly. Otherwise, correctness follows from the correctness of pviO .

PV correctness. We now argue more formally that the scheme has PV correctness if the underlying pviO is correct. Suppose that pviO has a proof of N' -correctness, i.e. there exists a polynomial q such that:

$$\left(\begin{array}{l} \widetilde{M} = \text{pviO}(M; r), \\ |x| \leq N'(\lambda), \\ \mathcal{U}_{\text{clock}}(M, x, 1^t) \neq \perp \end{array} \right) \vdash_{\text{PV}} \left(\mathcal{U}_{\text{clock}}(\widetilde{M}, x, 1^t) = \mathcal{U}_{\text{clock}}(M, x, 1^{q(\lambda, t)}) \right)$$

In the above, we treat $\text{pub} = \emptyset$ in the input to $\mathcal{U}_{\text{clock}}$. Now, we argue that the scheme $\mathcal{O}$ defined in Algorithm 1 has a PV proof of its correctness property. Consider the following polynomial $q'(\lambda, t) = q(\lambda, t + \text{poly}(T, \lambda))$. By construction and correctness of pviO , we have that

$$\left(\begin{array}{l} \widetilde{M} = \text{pviO}(M^{\text{hk}, h}; r), \\ |x| + |\text{pub}| \leq N'(\lambda), \\ \mathcal{U}_{\text{clock}}(M^{\text{hk}, h}, x, 1^t) \neq \perp \end{array} \right) \vdash_{\text{PV}} \left(\mathcal{U}_{\text{clock}}(\widetilde{M}, (\text{pub}, x), 1^t) = \mathcal{U}_{\text{clock}}(M, x, 1^{q(\lambda, t)}) \right)$$

Recall that by construction, we have that $N'(\lambda) = N(\lambda) + S(\lambda)$, and we choose $S(\lambda)$ so that $S(\lambda) \geq |\text{pub}|$. Also, $\mathcal{O}(1^\lambda, M[\text{pub}]) = \text{pviO}(M^{\text{hk}, h})$. Moreover, by construction, we have that

$$\vdash_{\text{PV}} \mathcal{U}_{\text{clock}}(M[\text{pub}], x, 1^t) = \mathcal{U}_{\text{clock}}(M^{\text{hk}, h}, (\text{pub}, x), 1^{t+P(|x|, \lambda)}),$$

where $P(|x|, \lambda) = \text{poly}(T(|x|), \lambda)$ as defined in Step (2). Therefore, we have that

$$\left(\begin{array}{l} \widetilde{M} = \mathcal{O}(1^\lambda, M[\text{pub}]; r), \\ |x| \leq N(\lambda), \\ \mathcal{U}_{\text{clock}}(M[\text{pub}], x, 1^t) \neq \perp \end{array} \right) \vdash_{\text{PV}} \left(\mathcal{U}_{\text{clock}}(M[\text{pub}], x, 1^t) = \mathcal{U}_{\text{clock}}(\widetilde{M}[\text{pub}], x, 1^{q'(\lambda, t)}) \right)$$

as desired.

Remark 4.7 (Stronger PV correctness with respect to digest). In Section 5, we will rely on the following stronger correctness property of the construction: correctness holds for any pub' whose hash value collides with pub . Formally,

$$\left(\begin{array}{l} (\text{hk}, h, \Pi) \leftarrow \mathcal{O}(1^\lambda, M[\text{pub}]) \\ \wedge |\text{pub}'| = |\text{pub}| \\ \wedge \text{Tree-SSB.Hash}(\text{hk}, \text{pub}') = h \\ \wedge |x| \leq N(\lambda) \\ \wedge \mathcal{U}_{\text{clock}}(M[\text{pub}'], x, 1^t) \neq \perp \end{array} \right) \vdash_{\text{PV}} \left(\mathcal{U}_{\text{clock}}(M[\text{pub}'], x, 1^t) = \mathcal{U}_{\text{clock}}(\widetilde{M}[\text{pub}'], x, 1^{q(\lambda, t)}) \right)$$

for some polynomial q .

4.3 Proof of Theorem 4.5

We begin by giving an overview of the hybrids in the proof and the high level idea of the proof strategy between each pair of hybrids. We defer the detailed proofs to Sections 4.3.1 and 4.3.2. Throughout this proof, we will rely on Theorem 3.2 and Lemma 3.4 to invoke pviO security.

To show security, it suffices to show that for all $\text{poly}(\Gamma)$ -sized adversaries $\mathcal{A}$ and (ρ, T) -EF-equivalent $M_0[\text{pub}]$ and $M_1[\text{pub}]$ satisfying the premise of Definition 4.3, we have that

$$\Pr \left[\mathcal{A}(\widetilde{M}_b[\text{pub}]) = b : \begin{array}{l} b \leftarrow \{0, 1\} \\ \widetilde{M}_b[\text{pub}] \leftarrow \mathcal{O}(1^\lambda, M_b[\text{pub}]) \end{array} \right] \leq \frac{1}{2} + \Gamma^{-O(1)} . \quad (1)$$

Suppose the machine $\mathcal{M}_{\text{EF}}$ is the size ρ machine which outputs the above EF proof that $\llbracket M_0[\text{pub}](x) = M_1[\text{pub}](x) \rrbracket_n$ on input $1^n, M_0, M_1, \text{pub}$.

We will do this via a sequence of hybrids. For $i \in [N]$, consider the machine $P^{(i)}$ defined as follows:

$$P^{(i)}[\text{pub}](x) = \begin{cases} M_1[\text{pub}](x) & \text{if } |x| \leq i, \\ M_0[\text{pub}](x) & \text{otherwise.} \end{cases}$$

It is easy to see that $M_0[\text{pub}]$ is identical to $P^{(0)}$ and $M_1[\text{pub}]$ is identical to $P^{(N)}$. We will show that for $i \in \{0, 1, \dots, N-1\}$,

$$\Pr \left[\mathcal{A}(\widetilde{W}[\text{pub}]) = b : \begin{array}{l} b \leftarrow \{0, 1\} \\ \widetilde{W}[\text{pub}] \leftarrow \mathcal{O}(1^\lambda, P^{(i+b)}[\text{pub}]) \end{array} \right] \leq \frac{1}{2} + \Gamma^{-O(1)} \cdot \frac{1}{\text{poly}(N)} . \quad (2)$$

Then, by a standard hybrid argument, we obtain our desired inequality. In the rest of this proof, we will establish (2) for $i \in \{0, 1, 2, \dots, N-1\}$.

Hash failures. For the security analysis, modify each hybrid to output a fixed failure symbol if $h = \perp$ or if any reference hash needed in that hybrid or the transition under consideration is $\perp$. Reference hashes are hashes of pub and of configurations in the computation of C_{EF} on the fixed input $(1^{i+1}, M_0, M_1, \text{pub})$, including their internal tree nodes. For each transition, use the union of the reference computations needed on its two sides. Their unhashed messages are independent of the sampled hash keys. By Theorem 3.22, their failure probabilities are negligible in Γ , including when individual level keys are switched to trapdoor generation in the index-hiding hybrids.

The number of reference hash computations and hybrid transitions is

$$\text{poly}(\lambda, N, T(\lambda, N), |M_0|, |M_1|, |\text{pub}|) :$$

the tableau has polynomially many cells in its time bound, and both pebbling strategies use polynomially many moves. Since $N \leq \Gamma$, T is polynomial, and the program descriptions have polynomial size, this is $\text{poly}(\Gamma)$. A union bound therefore bounds the total statistical loss from these modifications by

$$\text{poly}(\Gamma(\lambda)) \cdot \text{negl}(\Gamma(\lambda)) = \text{negl}(\Gamma(\lambda)).$$

This also accounts for the real construction's branch that reveals M when $h = \perp$. In the modified hybrids, every reference digest used for binding is non- $\perp$. Each relation R certifying a PV equivalence below includes checks that the reference digests it uses are well formed and non- $\perp$. Equality with such a digest supplies the nonfailure premise for extraction on a candidate input. Key-indistinguishability reductions use the original key distributions and perform the failure checks after sampling; no conditioning of the key distributions is required. All hybrids below are understood with this failure convention, and their statistical losses are included in the displayed bounds.

Overview of hybrids. We now give an overview of the hybrids and give a high level idea of the proof strategy between each pair of hybrids, with detailed proofs of some hybrids deferred to Sections 4.3.1 and 4.3.2. Since the parameter 1^λ is clear from context, we will drop them from the syntax of the obfuscation.

$H_{i,1}$ Let $Q^{(1)} := Q_{\lambda, N, |\text{pub}|, \text{hk}, h, M_0, M_1, i}^{(1)}$ denote the machine constructed in Step (2) of $\mathcal{O}(P^{(i)}[\text{pub}])$. Output $(\text{hk}, h, \Pi) \leftarrow \mathcal{O}(P^{(i)}[\text{pub}])$, where $\mathcal{O}$ is the obfuscation algorithm in Algorithm 1. Recall that $\Pi = \text{pviO}(Q^{(1)})$.

$H_{i,2}$ Sample hk and h as in Algorithm 1. Let $s = O(1)$ be the number of bits required to describe the contents of a single cell in a TM tape. Additionally **sample a second hash key $\text{hk}' \leftarrow \text{Tree-SSB.Gen}(1^\lambda)$** , where $\text{hk}' = (\text{hk}'_1, \text{hk}'_2, \dots, \text{hk}'_\lambda)$ is a list of Two-SSB keys. First, we will define the following **helper machine C_{EF}** which has M_{EF} hardwired in it:

C_{EF} on input $(1^i, M_0, M_1, y)$:

- Run $\pi \leftarrow M_{\text{EF}}(1^i, M_0, M_1, y)$ for $T(\lambda, i)$ time.
- Verify that π is an EF proof of the fact that $M_0[y](x) = M_1[y](x)$ for $x \in \{0, 1\}^i$. Output 1 if the check passes, and 0 otherwise.

It is easy to see that C_{EF} can be described by a program of size $\rho + O(1)$. Construct a machine $Q^{(2)} := Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i'}^{(2)}$ which does the following:

- $Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i}^{(2)}$ on input $y \in \{0, 1\}^{|\text{pub}|}$ and $x \in \{0, 1\}^{\leq N}$:
- Check if $\text{Tree-SSB.Hash}(\text{hk}, y) = h$, and abort if this does not hold.
 - If $|x| = i + 1$,
 - Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $\text{cf}_0, \text{cf}_1, \dots, \text{cf}_K$ denote the TM computation tableau of C_{EF} , where $\text{cf}_0 = (1^{i+1}, M_0, M_1, y)^a$.
 - For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(\text{hk}', \text{cf}_j)$.
 - Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

^a Here, we treat a computation tableau as a $(K + 1) \times W$ array whose entries are s -bit encodings of tape-cell contents at each time step. The tape alphabet includes head and state markers, so each cell of the next configuration is determined by a constant-size neighborhood in the current configuration. We refer to the j row of the array by cf_{j-1} . For simplicity, we assume W is a power of two.

Let $\Pi \leftarrow \text{pviO}(Q^{(2)})$, and output (hk, h, Π) .

Lemma 4.8. *For all size Γ adversaries $\mathcal{A}$, we have that:*

$$\left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,2} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,1} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)}.$$

Proof. By definition, $Q^{(2)}$ simply adds some additional (independent) computation W on top of $Q^{(1)}$. It is therefore easy to see that

$$\begin{aligned} \forall \text{params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i), \forall (y, x), \\ \vdash_{\text{PV}} Q_{\lambda, N, |\text{pub}|, \text{hk}, h, M_0, M_1, i}^{(1)}(y, x) = Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i}^{(2)}(y, x). \end{aligned}$$

By applying Cook's translation (Theorem 3.2), one can then obtain a $O(1)$ -size machine R which on input 1^n , $Q^{(1)}$, $Q^{(2)}$ outputs an EF proof of the fact that $\llbracket Q^{(1)}(y, x) = Q^{(2)}(y, x) \rrbracket_n$ in polynomial time. Therefore, one can invoke $(O(1), p(\lambda, \log N, \log \Gamma, \cdot))$ -EF-security of pviO to obtain the result, where p is some fixed polynomial. $\square$

$H_{i,3}$ In this hybrid, sample hk, hk', h as in $H_{i,2}$. Additionally, compute the following hash values:

- Let $\text{cf}_0^{\text{pub}}, \text{cf}_1^{\text{pub}}, \dots, \text{cf}_K^{\text{pub}}$ denote the computation tableau of C_{EF} on input $(1^{i+1}, M_0, M_1, \boxed{\text{pub}})$.
- For $j \in \{0, \dots, K\}$, let $\phi_j^{\text{pub}} = \text{Tree-SSB.Hash}(\text{hk}', \text{cf}_j^{\text{pub}})$.

Now, we define the machine $Q^{(3)} := Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}}^{(3)}$.

$Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}}^{(3)}$ on input $y \in \{0, 1\}^{|\text{pub}|}$ and $x \in \{0, 1\}^{\leq N}$:

- Check if $\text{Tree-SSB.Hash}(\text{hk}, y) = h$, and abort if this does not hold.
- If $|x| = i + 1$,
 - Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $\text{cf}_0, \text{cf}_1, \dots, \text{cf}_K$ denote the TM computation tableau of C_{EF} , where $\text{cf}_0 = (1^{i+1}, M_0, M_1, y)$.
 - For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(\text{hk}', \text{cf}_j)$.
 - If $\phi_0 \neq \phi_0^{\text{pub}}$, abort and output $\perp$.
- Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

Output (hk, h, Π) , where $\Pi \leftarrow \text{pviO}(Q^{(3)})$.

Lemma 4.9. For all size Γ adversaries $\mathcal{A}$, we have that:

$$\left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,3} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,2} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)} \cdot \text{poly}(T(\lambda, i)).$$

We defer the full proof to Section 4.3.1.

$H_{i,4}$ In this hybrid, we define $\text{hk}, \text{hk}', h, \{\phi_j^{\text{pub}}\}_j$ as in $H_{i,3}$.

We define the machine $Q^{(4)} := Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}}^{(4)}$.

$Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}}^{(4)}$ on input $y \in \{0, 1\}^{|\text{pub}|}$ and $x \in \{0, 1\}^{\leq N}$:

- Check if $\text{Tree-SSB.Hash}(\text{hk}, y) = h$, and abort if this does not hold.
- If $|x| = i + 1$,
 - Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $\text{cf}_0, \text{cf}_1, \dots, \text{cf}_K$ denote the TM computation tableau of C_{EF} , where $\text{cf}_0 = (1^{i+1}, M_0, M_1, y)$.
 - For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(\text{hk}', \text{cf}_j)$.
 - If $\phi_0 \neq \phi_0^{\text{pub}}$, abort and output $\perp$.
 - If $\phi_K \neq \phi_K^{\text{pub}}$, abort and output $\perp$.
- Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

Output (hk, h, Π) , where $\Pi \leftarrow \text{pviO}(Q^{(4)})$.

Lemma 4.10. For all size Γ adversaries $\mathcal{A}$, we have that:

$$\left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,4} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,3} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)} \cdot \text{poly}(T(\lambda, i)).$$

We defer the full proof to Section 4.3.2.

$H_{i,5}$ In this hybrid, we define $hk, hk', h, \{\phi_j^{\text{pub}}\}_j$ as in $H_{i,4}$.

We define the machine $Q^{(5)} := Q_{\lambda, N, |\text{pub}|, hk, hk', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}}^{(5)}$ as follows:

$Q_{\lambda, N, |\text{pub}|, hk, hk', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}}^{(5)}$ on input $y \in \{0, 1\}^{|\text{pub}|}$ and $x \in \{0, 1\}^{\leq N}$:

- Check if $\text{Tree-SSB.Hash}(hk, y) = h$, and abort if this does not hold.
- If $|x| = i + 1$,
 - Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $\text{cf}_0, \text{cf}_1, \dots, \text{cf}_K$ denote the TM computation tableau of C_{EF} , where $\text{cf}_0 = (1^{i+1}, M_0, M_1, y)$.
 - For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(hk', \text{cf}_j)$.
 - If $\phi_0 \neq \phi_0^{\text{pub}}$, abort and output $\perp$.
 - If $\phi_K \neq \phi_K^{\text{pub}}$, abort and output $\perp$.
 - If the output of C_{EF} is not equal to 1, output $\perp$.
- Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

Output (hk, h, Π) , where $\Pi \leftarrow \text{pviO}(Q^{(5)})$.

Lemma 4.11. For all size Γ adversaries $\mathcal{A}$, we have that:

$$\left| \Pr \left[\mathcal{A}(hk, h, \Pi) = 1 : (hk, h, \Pi) \leftarrow H_{i,5} \right] - \Pr \left[\mathcal{A}(hk, h, \Pi) = 1 : (hk, h, \Pi) \leftarrow H_{i,4} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)}.$$

Proof. First, by using the index-indistinguishability of the hash key, sample hk' so that ϕ_K^{pub} is extractable on $\text{cf}_K^{\text{pub}}[1]$, i.e. the bit corresponding to the output of $C_{\text{EF}}(1^{i+1}, M_0, M_1, y)$. Now, let r denote the randomness used to sample hk' .

Define the following relation R which does the following: on input $\text{params} = (\lambda, hk, hk', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}, y, x)$ and $\text{aux} = (r, \text{td}^*)$,

- Ensure that the sizing of the elements in params and aux are consistent (e.g. r is of the appropriate size for Tree-SSB.TGen , that hk, hk', h, td^* have the appropriate sizes, etc).
- Compute $(hk^*, \tilde{\text{td}}) \leftarrow \text{Tree-SSB.TGen}(1^\lambda, j = 1; r)$, and verify that $hk' = hk^*$ and $\text{td}^* = \tilde{\text{td}}$.
- Verify that $\text{Tree-SSB.Ext}(hk', \text{td}^*, \phi_K^{\text{pub}}) = 1$.

We now argue in PV that

$$\forall \text{params} = (\lambda, hk, hk', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}), \forall \text{aux} = (r, \text{td}^*), \forall (y, x) \left(R(\text{params}, \text{aux}) = 1 \right) \rightarrow \left(Q_{\text{params}}^{(4)}(y, x) = Q_{\text{params}}^{(5)}(y, x) \right)$$

We argue as follows: on input (y, x) ,

- If $\phi_K \neq \phi_K^{\text{pub}}$, both programs output $\perp$.
- Else, $\phi_K = \phi_K^{\text{pub}}$.
- Recall that R certifies that hk' is generated from $\text{Tree-SSB.TGen}(1^\lambda, j = 1; r)$ along with trapdoor td^* .
- By correctness of the Tree-SSB extraction, we have that

$$C_{\text{EF}}(1^{i+1}, M_0, M_1, y) = \text{Tree-SSB.Ext}(\text{hk}', \text{td}^*, \phi_K^{\text{pub}}).$$

- Since $R(\text{params}, \text{aux}) = 1$, the RHS of the above is 1.
- Therefore, if $\phi_K = \phi_K^{\text{pub}}$, then by transitivity,

$$C_{\text{EF}}(1^{i+1}, M_0, M_1, y) = 1.$$

- Therefore, if $\phi_K = \phi_K^{\text{pub}}$, both programs output $P^{(i)}[y](x)$.

Now, one can use the above PV proof to construct a machine R which on input $1^n, Q^{(4)}, Q^{(5)}$ outputs an EF proof of $\llbracket Q^{(4)}(y, x) = Q^{(5)}(y, x) \rrbracket_n$ by simply fixing all of the variables other than y and x . Therefore, by $(O(|r|), p)$ -EF-equivalence of the two machines, one can invoke the EF-security of pviO to argue indistinguishability of the two hybrids. By assumption, we have that $|r| = \text{poly}(\lambda, \log N, \log \Gamma)$.

We get the desired security by relying on $\text{poly}(\Gamma, N, \lambda)$ -hardness of the underlying assumptions. $\square$

$H_{i,6}$ In this hybrid, sample $\text{hk}, \text{hk}', h, \{\phi_j^{\text{pub}}\}_j$ as in $H_{i,3}$. Define

$$Q^{(6)} := Q_{\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}}^{(6)}$$

to be the program which is identical to $Q^{(5)}$ with the last line changed to “Run $P^{(i+1)}[y](x)$ ”. Output (hk, h, Π) , where $\Pi \leftarrow \text{pviO}(Q^{(6)})$.

Lemma 4.12. *For all size Γ adversaries $\mathcal{A}$, we have that:*

$$\begin{aligned} & \left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,6} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,5} \right] \right| \\ & \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)} \cdot \text{poly}(T(\lambda, i)). \end{aligned}$$

Proof. We argue this by directly arguing that $Q^{(5)}$ and $Q^{(6)}$ are $(O(1), \text{poly})$ -EF-equivalent, and invoking the security of pviO .

Recall that $Q^{(5)}$ and $Q^{(6)}$ are defined as functions of $\text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}$. We now show that the following:

$$\begin{aligned} & \forall \text{ params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \phi_0^{\text{pub}}, \phi_K^{\text{pub}}), \forall (y, x) \\ & \vdash_{\text{PV}} \left(Q_{\text{params}}^{(5)}(y, x) = Q_{\text{params}}^{(6)}(y, x) \right), \end{aligned}$$

We will crucially use the following fact: The consistency of EF can be proven in PV [Coo75], i.e.

$$\forall M_0, M_1, y, x, \quad \left(\begin{array}{c} C_{\text{EF}}(1^{i+1}, M_0, M_1, y) = 1 \\ \wedge |x| = i + 1 \end{array} \right) \vdash_{\text{PV}} (M_0[y](x) = M_1[y](x)) \quad (3)$$

We argue this in two cases conditioned on the length of x .

Case 1: $|x| \neq i + 1$. We can argue as follows:

- If $\text{Tree-SSB.Hash}(\text{hk}, y) \neq h$, then $Q^{(5)}(y, x) = \perp$ and $Q^{(6)}(y, x) = \perp$.
- If $\text{Tree-SSB.Hash}(\text{hk}, y) = h$, then

$$\begin{aligned} Q^{(5)}(y, x) &= P^{(i)}[y](x), \\ Q^{(6)}(y, x) &= P^{(i+1)}[y](x). \end{aligned}$$

- If $|x| \leq i$, by definition, $P^{(i)}[y](x) = P^{(i+1)}[y](x) = M_1[y](x)$.
- If $|x| \geq i + 2$, by definition, $P^{(i)}[y](x) = P^{(i+1)}[y](x) = M_0[y](x)$.
- By combining the case work, we have that if $|x| \neq i + 1$, then $Q^{(5)}(y, x) = Q^{(6)}(y, x)$.

Case 2: $|x| = i + 1$. We argue as follows:

- If $C_{\text{EF}}(1^{i+1}, M_0, M_1, y) \neq 1$ or if any of the hash checks fail, both programs output $\perp$.
- If $C_{\text{EF}}(1^{i+1}, M_0, M_1, y) = 1$, then

$$\begin{aligned} Q^{(5)}(y, x) &= P^{(i)}[y](x) = M_0[y](x), \\ Q^{(6)}(y, x) &= P^{(i+1)}[y](x) = M_1[y](x). \end{aligned}$$

- If $C_{\text{EF}}(1^{i+1}, M_0, M_1, y) = 1$, by (3), we also have that $M_0[y](x) = M_1[y](x)$.
- Therefore, we have that for $|x| = i + 1$, $Q^{(5)}(y, x) = Q^{(6)}(y, x)$ by transitivity.

Combining the above, we have the desired PV proof of equivalence. By applying Cook's translation (Theorem 3.2) to the above PV proof, one can then obtain a uniform machine R which on input 1^n outputs an EF proof of equivalence for $\llbracket Q^{(5)}(y, x) = Q^{(6)}(y, x) \rrbracket_n$ after substituting the values of all variables (except y and x). Therefore, one can invoke the uniform-EF security of pviO to argue indistinguishability. $\square$

$\boxed{H_{i,7}}$ Sample $(\text{hk}, h, \Pi) \leftarrow \mathcal{O}(\boxed{P^{(i+1)}}[\text{pub}])$ and output (hk, h, Π) .

Lemma 4.13. For all size Γ adversaries $\mathcal{A}$, we have that:

$$\begin{aligned} & \left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,7} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,6} \right] \right| \\ & \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)} \cdot \text{poly}(T(\lambda, i)). \end{aligned}$$

Proof. Indistinguishability follows by arguing as was done in hybrids $H_{i,5}$ to $H_{i,1}$, but in reverse. $\square$

4.3.1 Proof of Lemma 4.9

In this section, we prove Lemma 4.9. First, it will be useful to consider the following standard pebbling game on balanced binary trees; see, e.g., [CS76, LT80, Nor13].

Pebbling game on binary trees. Consider a balanced binary tree of depth d and $2^{d+1} - 1$ nodes (i.e. root node is at level 0 and leaf nodes are at level d). Initially, there are no pebbles at any node in the tree. The goal of the game is to place a pebble at the root node of the tree. The game proceeds according to the following rules.

Rule A: We can place a pebble at any leaf node.

Rule B: We can place a pebble at a node if there are pebbles at both of its children nodes.

Rule C: We can remove a pebble at a node if there is a pebble at its parent node.

Following the technical overview, Rule A corresponds to Types 1 and 2, Rule B corresponds to Type 3 and Rule C corresponds to Type 4.

Claim 4.14. *There exists a strategy to pebble the tree using $O(d)$ pebbles and $O(2^d)$ steps.*

Proof. We describe the algorithm recursively.

Algorithm 2 Recursive depth-first pebbling strategy for binary trees (Claim 4.14).

To place a pebble at node v at height h (i.e. depth $d - h$),

- If v is a height 0 (i.e. a leaf node), use Rule A to place a pebble.
- Else, v has two children v_L and v_R .
 - Recursively place a pebble on v_L so that v_L is the only node in this subtree with a pebble.
 - Recursively place a pebble on v_R so that v_R is the only node in this subtree with a pebble.
- Use Rule B to place a pebble at v .
- Use Rule C to remove the pebbles at v_L and v_R .

At the end, within the subtree rooted at v , the only pebbled node is v .

We now analyze the algorithm.

Correctness. The correctness of the algorithm is self-evident, as it is clear that the pebbles are placed/removed according to the stipulated rules.

Runtime. Let $T(h)$ denote the number of steps required to place a pebble at a node at height h . Clearly, $T(0) = 1$, and $T(h) = 2T(h - 1) + 3$. Solving the recurrence, it is clear that $T(h) = 2^{h+2} - 3$.

Number of pebbles. Suppose $P(h)$ pebbles are required to place a pebble at a node v at height h . Clearly, $P(0) = 1$. To place a pebble at v at height h , first $P(h - 1)$ pebbles are used to place a pebble at v_L , leaving only 1 pebble at v_L . Then, $P(h - 1)$ additional pebbles are used to place a pebble at v_R . Therefore, $P(h) = P(h - 1) + 1 = h + 1$.

This completes the proof. $\square$

We are now ready to prove Lemma 4.9. Recall the syntax of the computation tableau as in Footnote a. Suppose the tableau has width W (WLOG W is a power of 2), and let $d = \log_2 W$. Let $\rho_0, \rho_1, \dots, \rho_w$ denote the pebbling strategy given by Claim 4.14 for binary trees of depth d , where $\rho_j \subseteq [2^{d+1} - 1]$ denotes the set of pebbles placed at the j th step of the pebbling strategy. Let $\rho_0 = \emptyset$.

$H_{i,2,j}$ Sample hk, hk', h as in Item $H_{i,2}$, and set $cf_0^{\text{pub}} = (1^{i+1}, M_0, M_1, \text{pub})$ (padded to width W accordingly). Additionally:

- Compute $\phi_0^{\text{pub}} = \text{Tree-SSB.Hash}(hk', cf_0^{\text{pub}})$ via the Two-SSB, and let $\phi_{0,v}^{\text{pub}}$ denote the node v in the Merkle tree computation of ϕ_0^{pub} .
- Let $P_j = \{\phi_{0,v}^{\text{pub}}\}_{v \in \rho_j}$, where ρ_j is the j th step of the pebbling strategy.
- Now, construct the following machine $Q_j^{(2)} := Q_{j,\lambda,N,|\text{pub}|,hk,hk',h,M_0,M_1,i,\rho_j,P_j}^{(2)}$ with P_j hardcoded in it.

$Q_{j,\lambda,N,|\text{pub}|,hk,hk',h,M_0,M_1,i,\rho_j,P_j}^{(2)}$ on input $y \in \{0, 1\}^{|\text{pub}|}$ and $x \in \{0, 1\}^{\leq N}$:

- Check if $\text{Tree-SSB.Hash}(hk, y) = h$, and abort if this does not hold.
- If $|x| = i + 1$,
 - * Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $cf_0, cf_1, \dots, cf_K$ denote the TM computation tableau of C_{EF} , where $cf_0 = (1^{i+1}, M_0, M_1, y)$.
 - * For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(hk', cf_j)$.
 - * Let $\phi_{0,v}$ denote the intermediate nodes of the Merkle tree in the computation of ϕ_0 . For $v \in \rho_j$, if $\phi_{0,v} \neq \phi_{0,v}^{\text{pub}}$, abort and output $\perp$.
- Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

By definition, since $H_{i,2,0} = H_{i,2}$ and $H_{i,2,w} = H_{i,3}$, and by Claim 4.14, $w = O(W)$.

We now show the following claim.

Claim 4.15. For $j \in \{0, 1, \dots, w - 1\}$, we show that: For all size Γ adversaries $\mathcal{A}$, we have that:

$$\left| \Pr \left[\mathcal{A}(hk, h, \Pi) = 1 : (hk, h, \Pi) \leftarrow H_{i,2,j+1} \right] - \Pr \left[\mathcal{A}(hk, h, \Pi) = 1 : (hk, h, \Pi) \leftarrow H_{i,2,j} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)}$$

Proof. We now consider three different cases based on if the set ρ_{j+1} is constructed via Rule A, B or C of the pebbling game.

Rule A: In this case, ρ_{j+1} contains a new leaf node v . Recall that $cf_0 = (1^{i+1}, M_0, M_1, y)$ and $cf_0^{\text{pub}} = (1^{i+1}, M_0, M_1, \text{pub})$. If this leaf node corresponds to an entry of $1^{i+1}, M_0$ or M_1 (independent of y), by construction, it is easy to verify that $\phi_{0,v} = \phi_{0,v}^{\text{pub}}$.

If v corresponds to index ℓ of pub , we then argue as follows. First, sample hk so that hk is statistically binding on the ℓ th bit of pub . This is indistinguishable by index-hiding. Suppose randomness r was used to generate this hk along with the trapdoor td .

We introduce a function symbol R which takes as input $\text{params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \rho_j, P_j, \rho_{j+1}, P_{j+1}, v, \phi_{0,v}^{\text{pub}})$ and $\text{aux} = (r, \text{td})$ and checks the following:

- **Verify syntax:**
 - **Valid parameter sizing:** Ensures that the sizing of the parameters are all consistent (e.g. the size of $\text{hk}, \text{hk}', |v|$ is consistent with λ , the size of ϕ_0 and $\phi_{0,v}^{\text{pub}}$, etc).
 - **Consistency of v and ℓ :** Check that the node v in the Merkle tree corresponds to the leaf node reading the ℓ th bit of y .
 - **Consistency of ρ_j, ρ_{j+1} and P_j, P_{j+1} :** Verify that $\rho_{j+1} = \rho_j \cup \{v\}$ and $P_{j+1} = P_j \cup \{\phi_{0,v}^{\text{pub}}\}$.
- **Certify $\text{hk}, \text{td}, r, \ell$:** Verify $(\text{hk}, \text{td}) \leftarrow \text{Tree-SSB.TGen}(1^\lambda, \ell; r)$.
- **Verify extracted value of h :** Check that $\phi_{0,v}^{\text{pub}} = \text{Tree-SSB.Ext}(\text{hk}, \text{td}, h)$.

Suppose randomness r was used to sample hk . We can now argue in PV that:

$$\begin{aligned} \forall \text{params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \rho_j, P_j, \rho_{j+1}, P_{j+1}, v, \phi_{0,v}^{\text{pub}}), \\ \forall \text{aux} = (r, \text{td}), \forall (y, x), \\ R(\text{params}, \text{aux}) = 1 \quad \vdash_{\text{PV}} \quad Q_j^{(2)}(y, x) = Q_{j+1}^{(2)}(y, x) \end{aligned}$$

We sketch the proof below.

- If $\text{Tree-SSB.Hash}(\text{hk}, y) \neq h$, both programs output $\perp$. Suffices to consider the case where $\text{Tree-SSB.Hash}(\text{hk}, y) = h$.
- Recall that R certifies that $(\text{hk}, \text{td}) \leftarrow \text{Tree-SSB.TGen}(1^\lambda, \ell; r)$. By PV correctness of extraction, we have that

$$\left(\begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Tree-SSB.TGen}(1^\lambda, \ell; r) \\ \wedge k \leq 2^\lambda \wedge y \in \{0, 1\}^k \\ \wedge \tau \leftarrow \text{Tree-SSB.Hash}(\text{hk}, y) \wedge \tau \neq \perp \\ \alpha \leftarrow \text{Tree-SSB.Ext}(\text{hk}, \text{td}, \tau) \end{array} \right) \vdash_{\text{PV}} y[\ell] = \alpha.$$

- Since R certifies that the node v corresponds to the ℓ th index of y , $\phi_{0,v} = y[\ell]$.
- By transitivity, $\phi_{0,v} = \phi_{0,v}^{\text{pub}}$.
- Therefore, the added check that $\phi_{0,v} = \phi_{0,v}^{\text{pub}}$ does not change the functionality, completing the proof.

Recall that $\text{aux} = (r, \text{td})$ can be generated given only r . Therefore, by using the above PV proof, one can construct a machine of size $|r| + O(1)$ which takes as input $1^n, Q_j^{(2)}, Q_{j+1}^{(2)}$, first asserts that the premise of the above PV proof holds, and then outputs $\llbracket Q_j^{(2)}(y, x) = Q_{j+1}^{(2)}(y, x) \rrbracket_n$ in $\text{poly}(\lambda, n)$ time (combined via modus ponens). Therefore, one can rely on the EF-security of pviO to argue indistinguishability.

Rule B: In this hybrid, $v = \rho_{j+1} \setminus \rho_j$ corresponds to a node at height $H \geq 1$ where both children $v_L, v_R \in \rho_j, \rho_{j+1}$. Let $\text{hk}' = (\text{hk}'_1, \dots, \text{hk}'_\lambda)$ be the underlying Two-SSB keys. We now introduce the following PV function symbol R : on input $\text{params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \rho_j, P_j, \rho_{j+1}, P_{j+1}, v, v_L, v_R, H)$,

- **Verify syntax:**
 - **Valid parameter sizing:** Ensures that the sizing of the parameters are all consistent (e.g. the size of $\text{hk}, \text{hk}', |v|$ is consistent with λ , the size of ϕ_0 and $\phi_{0,v}^{\text{pub}}$, etc).
 - **Consistency of ρ_j, ρ_{j+1} and P_j, P_{j+1} :** Verify that $\rho_{j+1} = \rho_j \cup \{v\}$ and $P_{j+1} = P_j \cup \{\phi_{0,v}^{\text{pub}}\}$.
 - **Verify v, v_L, v_R :** Verify that nodes v_L and v_R are indeed the left and right child of v respectively, and that $v_L, v_R \in \rho_j$.
 - **Correctness of H :** Verify that node v is indeed at height H of the Merkle tree.
- **Verify consistency of hashes:** Verify that $\phi_{0,v}^{\text{pub}} = \text{Two-SSB.Hash}(\text{hk}'_H, (\phi_{0,v_L}^{\text{pub}}, \phi_{0,v_R}^{\text{pub}}))$.

Then, we can prove easily in PV that:

$$\forall \text{params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \rho_j, P_j, \rho_{j+1}, P_{j+1}, v, v_L, v_R, H), \forall (y, x) \\ R(\text{params}) = 1 \quad \vdash_{\text{PV}} \quad Q_j^{(2)}(y, x) = Q_{j+1}^{(2)}(y, x).$$

We can argue as follows:

- If $\phi_{0,v_L} \neq \phi_{0,v_L}^{\text{pub}}$ or $\phi_{0,v_R} \neq \phi_{0,v_R}^{\text{pub}}$, both programs output $\perp$. Suffices to consider the case where both tests pass.
- Else, by construction of Tree-SSB.Hash from Two-SSB , we have that both machines compute

$$\phi_{0,v} = \text{Two-SSB.Hash}(\text{hk}'_H, (\phi_{0,v_L}, \phi_{0,v_R})).$$

- Recall that R certifies that $\phi_{0,v}^{\text{pub}} = \text{Two-SSB.Hash}(\text{hk}'_H, (\phi_{0,v_L}^{\text{pub}}, \phi_{0,v_R}^{\text{pub}}))$.
- By transitivity, $\phi_{0,v} = \phi_{0,v}^{\text{pub}}$. Therefore, adding this check does not change the functionality.

We can then use this PV proof along with Theorem 3.2 to create a machine of size $O(1)$ which outputs a uniform EF proof of equivalence between the two machines. Therefore, we can invoke pviO security to argue indistinguishability.

Rule C: In this hybrid, let $v = \rho_j \setminus \rho_{j+1}$ be removed via Rule C. Without loss of generality, suppose v is the left child of $v' \in \rho_j$ (a symmetric argument applies if v is a right child), where v' is at height H of the binary tree.

First, we switch to sampling the Two-SSB key hk'_H to be extractable on the left block (resp. right block if v is the right child). This is indistinguishable by the index indistinguishability of Two-SSB. Let s_H be the block length input to Two-SSB.Gen for the level H key of Tree-SSB . Suppose r is the randomness used to sample hk'_H , and let td'_H be the corresponding trapdoor.

We now define the following PV function symbol R which takes as input $\text{params} = (\lambda, N, |\text{pub}|, \text{hk}, \text{hk}', h, M_0, M_1, i, \rho_j, P_j, \rho_{j+1}, P_{j+1}, v, v', H, s_H)$ and $\text{aux} = (r, \text{td}'_H)$,

- **Verify syntax:**
 - **Valid parameter sizing:** Ensures that the sizing of the parameters are all consistent (e.g. the size of $hk, hk', |v|$ is consistent with λ , the size of ϕ_0 and $\phi_{0,v}^{\text{pub}}$, etc).
 - **Consistency of ρ_j, ρ_{j+1} and P_j, P_{j+1} :** Verify that $\rho_j = \rho_{j+1} \cup \{v\}$ and $P_j = P_{j+1} \cup \{\phi_{0,v}^{\text{pub}}\}$, and that v' is in both ρ_j and ρ_{j+1} .
 - **Verify v, v' :** Verify that node v is left (resp. right child) of v' .
 - **Correctness of H :** Verify that node v' is indeed at height H of the Merkle tree.
- **Verify somewhere extractability of hk'_H :** Check that $(hk'_H, td'_H) \leftarrow \text{Two-SSB.TGen}(1^\lambda, 1^{s_H}, 0; r)$ (extractable on 1 instead if v is the right child).
- **Check consistency between $\phi_{0,v}^{\text{pub}}$ and $\phi_{0,v'}^{\text{pub}}$:** Verify that $\phi_{0,v}^{\text{pub}} = \text{Two-SSB.Ext}(hk'_H, td'_H, \phi_{0,v'})$.

Then, we can prove in PV that:

$$\begin{aligned} \forall \text{params} &= (\lambda, N, |\text{pub}|, hk, hk', h, M_0, M_1, i, \rho_j, P_j, \rho_{j+1}, P_{j+1}, v, v', H, s_H), \\ \forall \text{aux} &= (r, td'_H), \forall (y, x) \\ R(\text{params}, \text{aux}) &= 1 \quad \vdash_{\text{PV}} \quad Q_j^{(2)}(y, x) = Q_{j+1}^{(2)}(y, x). \end{aligned}$$

We sketch the proof here.

- If $\phi_{0,v'} \neq \phi_{0,v'}^{\text{pub}}$, then both programs abort.
- Recall that R certifies that $(hk'_H, td'_H) \leftarrow \text{Two-SSB.TGen}(1^\lambda, 1^{s_H}, 0; r)$.
- By correctness of Two-SSB extraction, we have that

$$\text{Two-SSB.Ext}(hk'_H, td'_H, \text{Two-SSB.Hash}(hk'_H, (a, b))) = a.$$

- Recall that by definition $\phi_{0,v'} = \text{Two-SSB.Hash}(hk'_H, (\phi_{0,v}, \phi_{0,v'}^*))$, where v^* is the right child (symmetric argument if v is the right child).
- Therefore, by transitivity, it must be the case that if $\phi_{0,v'} = \phi_{0,v'}^{\text{pub}}$, then

$$\phi_{0,v} = \text{Two-SSB.Ext}(hk'_H, td'_H, \phi_{0,v'}) = \phi_{0,v}^{\text{pub}}.$$

- Therefore, if $\phi_{0,v'} = \phi_{0,v'}^{\text{pub}}$, the test $\phi_{0,v} = \phi_{0,v}^{\text{pub}}$ must also pass by definition. Therefore, the two programs are EF-equivalent.

We can therefore translate the above PV proof via Theorem 3.2 to obtain a $|r| + O(1)$ -sized machine which outputs the uniform EF proof of equivalence in time $\text{poly}(\lambda, \cdot)$. Hence, we can argue indistinguishability by invoking pviO security.

□

Recall that $W = O(T(\lambda, i))$. By Claim 4.15, we know that the program $Q_j^{(2)}$ is of size $|M_0| + |M_1| + \text{poly}(\lambda, \log \Gamma, \log W)$. Therefore, by combining Claim 4.15 and a hybrid argument over $O(W)$ steps, we have the desired inequality as in Lemma 4.9.

4.3.2 Proof of Lemma 4.10

Before establishing this lemma, we will first require the following different pebbling game.

Pebbling game on a line. Before we show this indistinguishability, we will first introduce the “pebbling game”⁹. Consider the integer line $0, 1, 2, \dots, N$. There is a pebble placed at position 0 of the line. There are now two rules:

Rule A: We can *place* a pebble at position $i + 1$ if there is a pebble at position i .

Rule B: We can *remove* a pebble at position $i + 1$ if there is a pebble at position i .

The goal of the game is to place a pebble at position N (and remove all pebbles at other positions) while minimizing both the number of pebbles placed on the number line at any point, and the number of moves needed. The work of [Ben89] shows that there does in fact exist a strategy which only uses $O(\log N)$ pebbles and takes $\text{poly}(N)$ steps.

Theorem 4.16 ([Ben89, GPSZ17]). *There exists a pebbling strategy that uses $O(\log N)$ pebbles to place a pebble at point N in $O(N^{\log_2 3})$ moves.*

We will use the optimal pebbling strategy to define our sequence of hybrids in the proof. Let η_k denote the set of pebbles placed after the k th move of the pebbling strategy on the number line $\{0, 1, \dots, K\}$. Let K' denote the total number of moves in this strategy, so $\eta_0 = \{0\}$ and $\eta_{K'} = \{0, K\}$. For $k \in \{0, 1, \dots, K'\}$ define the following hybrid.

$H_{i,3,k}$ Sample $hk, hk', h, \{\phi_j^{\text{pub}}\}_j$ as in $H_{i,3}$. Define the following machine $Q_k^{(3)}$:

$Q_{k,\lambda,N,|\text{pub}|,hk,hk',h,M_0,M_1,i,\eta_k,\{\phi_j^{\text{pub}}\}_{j \in \eta_k}}^{(3)}$ on input $y \in \{0, 1\}^{|\text{pub}|}$ and $x \in \{0, 1\}^{\leq N}$:

- Check if $\text{Tree-SSB.Hash}(hk, y) = h$, and abort if this does not hold.
- If $|x| = i + 1$,
 - Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $cf_0, cf_1, \dots, cf_K$ denote the TM computation tableau of C_{EF} , where $cf_0 = (1^{i+1}, M_0, M_1, y)$.
 - For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(hk', cf_j)$.
 - If $\phi_0 \neq \phi_0^{\text{pub}}$, abort and output $\perp$.
 - For $k' \in \eta_k$, check $\phi_{k'} = \phi_{k'}^{\text{pub}}$
- Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

Output $hk, h, \text{pviO}(Q_k^{(3)})$.

It is easy to see that $H_{i,3} \equiv H_{i,3,0}$ and $H_{i,4} \equiv H_{i,3,K'}$. We now show the following claim.

⁹This exposition follows closely to the work of [GPSZ17, GS18b, GS18a], but we only need pebbles of a single colour.

Claim 4.17. For $k \in \{0, 1, \dots, K' - 1\}$, we show that: For all size Γ adversaries $\mathcal{A}$, we have that:

$$\left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,3,k+1} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,3,k} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)} \cdot \text{poly}(T(\lambda, i)).$$

Proof. Consider some $k \in \{0, 1, \dots, K' - 1\}$. By the number line guarantee of the pebbling strategy, we have that there exists some index $\kappa \in \{0, 1, \dots, K\}$ such that both of the following hold:

- $\kappa \in \eta_k, \eta_{k+1}$.
- either $\eta_{k+1} = \eta_k \cup \{\kappa + 1\}$ (i.e. pebble at $\kappa + 1$ is added via Rule A) or $\eta_k = \eta_{k+1} \cup \{\kappa + 1\}$ (i.e. pebble at $\kappa + 1$ is removed via Rule B).

Without loss of generality, suppose that $\eta_{k+1} = \eta_k \cup \{\kappa + 1\}$ (a symmetric argument applies in the other setting).

We now prove the claim for k by considering further intermediate hybrids resembling the proof of Lemma 4.9. Let $\rho_0, \rho_1, \dots, \rho_w$ denote the binary pebbling strategy given by Claim 4.14 for binary trees of depth d , where $\rho_j \subseteq [2^{d+1} - 1]$ denotes the set of pebbles placed at the j th step of the pebbling strategy.

$H_{i,3,k,m}$ Sample $\text{hk}, \text{hk}', h, \{\phi_j^{\text{pub}}\}_j$ as in $H_{i,3}$. Define the following machine $Q_{k,m}^{(3)}$:

$Q_{k,m,\lambda,N,|\text{pub}|,\text{hk},\text{hk}',h,M_0,M_1,i,\eta_k,\{\phi_j^{\text{pub}}\}_{j \in \eta_k},\rho_m,\{\phi_{\kappa+1,v}^{\text{pub}}\}_{v \in \rho_m}}$ on input $y \in \{0,1\}^{|\text{pub}|}$ and $x \in \{0,1\}^{\leq N}$:

- Check if $\text{Tree-SSB.Hash}(\text{hk}, y) = h$, and abort if this does not hold.
- If $|x| = i + 1$,
 - Run C_{EF} on $(1^{i+1}, M_0, M_1, y)$, and let $\text{cf}_0, \text{cf}_1, \dots, \text{cf}_K$ denote the TM computation tableau of C_{EF} , where $\text{cf}_0 = (1^{i+1}, M_0, M_1, y)$.
 - For $j \in \{0, \dots, K\}$, let $\phi_j = \text{Tree-SSB.Hash}(\text{hk}', \text{cf}_j)$.
 - If $\phi_0 \neq \phi_0^{\text{pub}}$, abort and output $\perp$.
 - For $k' \in \eta_k$, check $\phi_{k'} = \phi_{k'}^{\text{pub}}$.
 - Let $\phi_{\kappa+1,v}$ denote the intermediate nodes of the Merkle tree in the computation of $\phi_{\kappa+1}$. For $v \in \rho_m$, check if $\phi_{\kappa+1,v} = \phi_{\kappa+1,v}^{\text{pub}}$.
- Run $P^{(i)}[y](x)$.

Pad the above program to match the runtime of the obfuscated program in the $H_{i,1}$.

Output $\text{hk}, h, \text{pviO}(Q_{k,m}^{(3)})$.

It is clear that $H_{i,3,k,0} \equiv H_{i,3,k}$ and $H_{i,3,k,w} \equiv H_{i,3,k+1}$. We now show that for all size Γ adversaries $\mathcal{A}$, $m \in \{0, 1, \dots, w - 1\}$, we have that:

$$\left| \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,3,k,m+1} \right] - \Pr \left[\mathcal{A}(\text{hk}, h, \Pi) = 1 : (\text{hk}, h, \Pi) \leftarrow H_{i,3,k,m} \right] \right| \leq \frac{1}{\text{poly}(\Gamma, N, \lambda)}.$$

The proof proceeds case-wise depending on the rule used to add κ to ρ_m , just as in Claim 4.15. While the argument for Rules B and C are identical to the argument in Claim 4.15, the argument for Rule A differs slightly in the case where v is a leaf node. We sketch the argument in this case.

Since we model the computation in the TM setting, the value of $\text{cf}_{\kappa+1}[v]$ depends on $O(1)$ locations of cf_{κ} . Suppose v depends on locations $v_1, \dots, v_c$ where $c = O(1)$, and the honest values of $\text{cf}_{\kappa}^{\text{pub}}[v_1], \dots, \text{cf}_{\kappa}^{\text{pub}}[v_c]$ are $\beta_1, \dots, \beta_c \in \{0, 1\}$. We argue via a few hybrids as follows:

1. For $\alpha \in [c]$, we iteratively argue as follows:
 - Set hk' to be extractable at position v_{α} via the index-hiding of Tree-SSB.
 - Add to the program the check that $\text{cf}_{\kappa}[v_{\alpha}] = \beta_{\alpha}$, and argue via PV correctness of Tree-SSB extraction that the machines are EF-equivalent and one can invoke pviO security.
2. Argue via the correctness of the TM computation that adding the check that $\text{cf}_{\kappa+1}[v] = \text{cf}_{\kappa+1}^{\text{pub}}[v]$ results in an EF-equivalent machine. We can then invoke pviO security.
3. Therefore, adding the check that $\phi_{\kappa+1,v} = \phi_{\kappa+1,v}^{\text{pub}}$ also results in an EF-equivalent machine. We can then invoke pviO security.
4. Undo the checks added in the previous hybrids by arguing in reverse. We can then invoke pviO security.

This completes the proof.

Efficiency. By the efficiency of both the line pebbling strategy in Theorem 4.16 and the binary tree pebbling strategy in Claim 4.14, we know that there are at most $O(\text{poly}(\log T(\lambda, i)))$ hard-coded values. $\square$

4.4 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 4.5.

Corollary 4.18 (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.

Proof. We first show security for programs $M_0[\text{pub}]$ and $M_1[\text{pub}]$ whose runtime is $p(n)$ on inputs of length n , where p is some polynomial.

Consider functionally equivalent machines M_0 and M_1 which accept inputs of length at most n . Let $P_y := P_{y, M_0, M_1, n}$ be the following program:

$$P_{y, M_0, M_1, n}[\text{pub}](x) = \begin{cases} M_1[\text{pub}](x) & \text{if } x < y, \\ M_0[\text{pub}](x) & \text{if } y \leq x \leq 2^n \end{cases}$$

Let R be the relation which takes as input $\text{params} = (y, M_0, M_1, n)$ and pub and outputs 1 if $|y| \leq n$ and $M_0[\text{pub}](y) = M_1[\text{pub}](y)$ where both machines run in time $p(|y|)$. It is clear that there is a PV proof of the fact that:

$$\begin{aligned} \forall \text{params} = (n, M_0, M_1, y), \forall x, \text{pub} \\ R(\text{params}, \text{pub}) = 1 \rightarrow P_{y-1}[\text{pub}](x) = P_y[\text{pub}](x). \end{aligned}$$

Therefore, one can construct a program $\mathcal{M}_{\text{EF}}(1^n, P_y, P_{y+1}, \text{pub})$ which does the following:

- Construct an EF proof that $R(\text{params}, \text{pub}) = 1$ (via Cook-Levin).
- Use Cook's translation to output a proof that

$$\llbracket R(\text{params}, \text{pub}) = 1 \rightarrow P_{y-1}[\text{pub}](x) = P_y[\text{pub}](x) \rrbracket_{|\text{params}|+|\text{pub}|+n}.$$

- Use modus ponens to deduce that $P_{y-1}[\text{pub}](x) = P_y[\text{pub}](x)$ for $|x| = n$.

Therefore, $P_{y-1}[\text{pub}]$ and $P_y[\text{pub}]$ are $(O(1), q)$ -EF-equivalent where q is some polynomial.

Let pviO be a Γ -secure fully succinct EF-IO scheme, where $\Gamma(\lambda) = 2^{\lambda^c}$ for some constant $0 < c < 1$. We choose

$$\lambda' = \lceil (C(\lambda + n))^{1/c} \rceil,$$

where C is a sufficiently large constant. Let $\mathcal{O}(1^\lambda, n, M[\text{pub}])$ output $\text{pviO}(1^{\lambda'}, M_n[\text{pub}])$, where $M_n[\text{pub}]$ evaluates $M[\text{pub}]$ on inputs of length at most n and rejects longer inputs.

Then, via 2^n hybrids, one can show that

$$\begin{aligned} \mathcal{O}(1^\lambda, n, M_0[\text{pub}]) &= \text{pviO}(1^{\lambda'}, P_{0, M_0, M_1, n}[\text{pub}]) \\ &\approx_c \dots \\ &\approx_c \text{pviO}(1^{\lambda'}, P_{2^n, M_0, M_1, n}[\text{pub}]) \\ &= \mathcal{O}(1^\lambda, n, M_1[\text{pub}]). \end{aligned}$$

By Γ -security, each hybrid transition has distinguishing advantage at most $2^{-(\lambda+n)}$. Therefore, the total advantage over the 2^n hybrids is at most

$$2^n \cdot 2^{-(\lambda+n)} = 2^{-\lambda}.$$

To obtain an obfuscation with input-by-input slowdown, one can simply carry out the standard “powers-of-two” trick. Let $M_{b,k}[\text{pub}]$ be the machine which executes $M_b[\text{pub}](x)$ for 2^k steps. If $M_b[\text{pub}](x)$ does not halt in 2^k , then $M_{b,k}[\text{pub}](x)$ outputs $\perp$. It is clear that for all $k \in [\lambda]$, $M_{0,k}[\text{pub}](x) = M_{1,k}[\text{pub}](x)$ (recall that the two programs have the same runtime and value on every input x). Therefore, consider the following obfuscation algorithm:

- **Obfuscation:** On input $M[\text{pub}]$, Let $\widetilde{M}_k = \mathcal{O}(1^\lambda, n, M_k[\text{pub}])$, where $M_k[\text{pub}]$ runs $M[\text{pub}]$ for k steps. Output $\Gamma = \{\widetilde{M}_k\}_{k \in [\lambda]}$.
- **Evaluation:** Given an input x , run $\widetilde{M}_1[\text{pub}], \widetilde{M}_2[\text{pub}], \dots$ on x until one finds $i \in [\lambda]$ such that $\widetilde{M}_i[\text{pub}] \neq \perp$.

It is easy to see that security follows from the functionality equivalence between $M_{0,k}[\text{pub}](x) = M_{1,k}[\text{pub}](x)$. Polynomial slowdown follows from the fact that if $2^{i-1} < \text{RT}(M[\text{pub}], x) \leq 2^i$, then $M_i[\text{pub}](x) \neq \perp$. Therefore, the program halts in time $\sum_j^i 2^j = O(2^i) = O(\text{RT}(M[\text{pub}], x))$. $\square$

5 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 5.1, and then present the main theorem in Section 5.2.

5.1 Definitions

In this section, we recall the definition of witness encryption [GGSW13] and succinct non-interactive arguments (SNARGs), and introduce terminology for encryption time and ciphertext size.

Syntax. A witness encryption scheme for NP is a tuple of algorithms (Enc, 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 machine $\mathcal{R}$, an instance x , a witness w and a ciphertext ct. It outputs a bit b or $\perp$.

Definition 5.1 (Witness encryption). A witness encryption scheme (Enc, Dec) for NP satisfies the following properties:

- **Correctness:** For every machine $\mathcal{R}$, instance x , witness w satisfying $|w| \leq n \leq 2^\lambda$, $\mathcal{R}(x, w) = 1$ and $\text{RT}(x, w) \leq 2^\lambda$, bit $b \in \{0, 1\}$, we have that

$$\Pr \left[\text{Dec}(\mathcal{R}, x, w, \text{ct}) = b : \text{ct} \leftarrow \text{Enc}(1^\lambda, \mathcal{R}, x, n, b) \right] = 1 .$$

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:** On all inputs within the stated bounds:
 - The running time of Enc is $\text{poly}(\lambda, |\mathcal{R}|, |x|, n)$.
 - The running time of Dec is $\text{poly}(\lambda, \text{RT}(\mathcal{R}, (x, w)))$.
 - *Witness-succinctness:* The scheme is *witness-succinct* if the running time of Enc is $\text{poly}(\lambda, |\mathcal{R}|, |x|)$, independent of the witness-length bound n .
 - *Full succinctness:* The scheme is *fully succinct* if it is witness-succinct and $|\text{ct}| = \text{poly}(\lambda, |\mathcal{R}|)$. Thus encryption may take time polynomial in the instance length, while the ciphertext size is independent of that length.
- **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

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

Later in the section, when we write WE.Enc and WE.Dec on a longer string $m \in \{0, 1\}^\ell$, we mean encrypting and decrypting m bit-by-bit.

Definition 5.2. We say that a witness encryption scheme (Enc, Dec) has a PV proof of completeness if its correctness can be formalized in theory PV as function symbols:

$$\left(\begin{array}{l} \text{ct} = \text{Enc}(1^\lambda, \mathcal{R}, x, n, b; r) \wedge \\ |w| \leq N(\lambda) \wedge \mathcal{U}_{\text{clock}}(1^t, \mathcal{R}, (x, w)) = 1 \end{array} \right) \vdash_{\text{PV}} \mathcal{U}_{\text{clock}}(1^{q(\lambda, t)}, \text{Dec}, (\mathcal{R}, x, w, \text{ct})) = b.$$

where q is some PV-definable polynomial.

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

- 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 .
- 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 .
- 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 5.3 (SNARG). A succinct non-interactive argument system for NP (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 $\text{RT}(\mathcal{R}, (x, w)) \leq 2^\lambda$, we have that

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

- **Efficiency:** On all inputs within the stated bounds:
 - The running time of Gen is $\text{poly}(\lambda, |\mathcal{R}|, n_x, n_w)$.
 - The running time of $\mathcal{P}$ is $\text{poly}(\lambda, |\mathcal{R}|, n_x, n_w)$.
 - The running time of $\mathcal{V}$ is $\text{poly}(\lambda, |\mathcal{R}|, n_x)$.
 - *Witness-succinctness:* The scheme is *witness-succinct* if the running time of Gen is $\text{poly}(\lambda, |\mathcal{R}|, n_x)$ and $|\Pi| = \text{poly}(\lambda, |\mathcal{R}|, n_x)$, both independent of the witness-length bound n_w .
 - *Full succinctness:* The scheme is *fully succinct* if it is witness-succinct, the running time of Gen is $\text{poly}(\lambda, |\mathcal{R}|)$, and $|\Pi| = \text{poly}(\lambda, |\mathcal{R}|)$. Thus setup time, CRS size, and proof size are independent of both length bounds, while verification may take time polynomial in the instance-length bound.

- **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

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

Definition 5.4 (SNARG with unique proofs.). A succinct non-interactive argument system for NP $(\text{Gen}, \mathcal{P}, \mathcal{V})$ has unique proofs if for every machine $\mathcal{R}$, instance x , pair of witnesses w_1, w_2 such that $\mathcal{R}(x, w_1) = 1, \mathcal{R}(x, w_2) = 1$ and bounds $n_x \geq |x|, n_w \geq \max(|w_1|, |w_2|)$ we have that

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

Definition 5.5 (SNARG with PV proof of completeness). We say that a succinct non-interactive argument system $(\text{Gen}, \mathcal{P}, \mathcal{V})$ has a PV proof of completeness if its correctness can be formalized in theory PV as function symbols:

$$\left(\begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, \mathcal{R}, n_x, n_w) \\ \wedge |w| \leq n_w \\ \wedge \mathcal{U}_{\text{clock}}(1^t, \mathcal{R}, (x, w)) = 1 \\ \wedge \pi \leftarrow \mathcal{U}_{\text{clock}}(1^{q(\lambda, t)}, \mathcal{P}, (\text{crs}, \mathcal{R}, x, w)) \end{array} \right) \vdash_{\text{PV}} (1 \leftarrow \mathcal{V}(\text{crs}, \mathcal{R}, x, \pi))$$

where q is some PV definable polynomial.

Definition 5.6 (SNARG with PV proof of uniqueness). We say that a succinct non-interactive argument system $(\text{Gen}, \mathcal{P}, \mathcal{V})$ has a PV proof of uniqueness if the following can be formalized in theory PV as function symbols:

$$\left(\begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, \mathcal{R}, n_x, n_w) \\ \wedge |w_1|, |w_2| \leq n_w \\ \wedge \mathcal{U}_{\text{clock}}(1^t, \mathcal{R}, (x, w_1)) = 1 \\ \wedge \mathcal{U}_{\text{clock}}(1^t, \mathcal{R}, (x, w_2)) = 1 \\ \wedge \pi_1 \leftarrow \mathcal{U}_{\text{clock}}(1^{q(\lambda, t)}, \mathcal{P}, (\text{crs}, \mathcal{R}, x, w_1)) \\ \wedge \pi_2 \leftarrow \mathcal{U}_{\text{clock}}(1^{q(\lambda, t)}, \mathcal{P}, (\text{crs}, \mathcal{R}, x, w_2)) \end{array} \right) \vdash_{\text{PV}} (\pi_1 = \pi_2) .$$

where q is some PV definable polynomial.

Remark 5.7 (Augmenting the definitions for succinct WE and SNARGs). For WE schemes and SNARGs that are witness-succinct, we can remove the bounds n and n_w on the witness size from the parameters of Enc and Gen , respectively, and instead fix these bounds to 2^λ (or $N(\lambda)$ if the scheme is only N -correct). For SNARGs that are fully succinct, we can additionally remove the instance size bound n_x and fix it in the same way. We additionally modify WE security and SNARG non-adaptive soundness to hold for these fixed bounds, while keeping the adversaries, relation descriptions, and challenge instances polynomial-sized in λ .

5.2 Main Theorem

Theorem 5.8. *Let $N(\lambda) \leq 2^\lambda$ be a function. Assuming the existence of:*

- *$\text{poly}(N, \lambda)$ -correct and secure fully succinct EF-IO (Definition 4.4) with a PV proof of correctness, as well as one-way functions,*
- *$\text{poly}(N, \lambda)$ -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.*

5.2.1 From Succinct Witness Encryption to Succinct IO

As we discussed in Section 2.2, we will show that the following construction is a $\text{poly}(N)$ -correct input-succinct (resp. fully succinct) IO is secure. Let ef-IO be a fully succinct EF-IO scheme.

Padding. Let $m = |M|$. By witness-succinctness, we have that the size of $P_{M_0, M_1, \text{ct}}$ is bounded by a fixed polynomial $L(\lambda, m, |\text{pub}|)$ whenever $|M_0| = |M_1| = m$. We choose L large enough to also accommodate M , and pad both programs to this length. These bounds depend only on the input to the real obfuscator.

If the WE scheme is fully succinct, the ciphertext size is independent of $|\text{pub}|$, and we can choose $L = L(\lambda, m)$. In particular, the size padding preserves full succinctness. We pad the runtime to match the hybrids.

Construction. We define the input-succinct (resp. fully succinct) IO scheme as follows: on input 1^λ , and machine $M[\text{pub}]$,

- Let M^{pad} be M with the above size and runtime padding.
- Output ef-IO($1^\lambda, M^{\text{pad}}[\text{pub}]$).

PV Correctness. The PV correctness follows directly from the PV correctness of the underlying EF-IO scheme and the correctness of the padding.

Security analysis. Consider M_0, M_1 and pub such that for all x , $M_0[\text{pub}](x) = M_1[\text{pub}](x)$ and $\text{RT}(M_0[\text{pub}], x) = \text{RT}(M_1[\text{pub}], x)$. We proceed in a few hybrids.

$\boxed{H_{b,1}}$ Output ef-IO($1^\lambda, M_b^{\text{pad}}[\text{pub}]$).

$\boxed{H_{b,2}}$ Define the following language:

$$\mathcal{L} = \left\{ (M_0, M_1, \text{pub}) : \begin{array}{l} \exists x \in \{0, 1\}^{\leq N(\lambda)} \text{ such that } M_0[\text{pub}](x) \neq M_1[\text{pub}](x). \\ \text{or } \text{RT}(M_0[\text{pub}], x) \neq \text{RT}(M_1[\text{pub}], x). \end{array} \right\}$$

Let $\mathcal{R}_{\mathcal{L}}$ denote the corresponding TM that decides the relation of $\mathcal{L}$. We sample the obfuscation in this experiment as follows:

- Sample $s \leftarrow \{0, 1\}^\lambda$. Let $r = G(s)$, where $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^p$ is a PRG that expands λ bits to p bits, where we choose p to be the length of the randomness required by the WE scheme in the next step. We prove that this is indistinguishable from the previous hybrid in Claim 5.9.
- Sample $\text{ct} \leftarrow \text{WE.Enc}(1^\lambda, \mathcal{R}_{\mathcal{L}}, (M_0, M_1, \text{pub}), N(\lambda), \boxed{b}; r)$.
- Construct the following machine $P_{M_0, M_1, \text{ct}}[\text{pub}]$: On input x ,
 - If $M_0[\text{pub}](x) = M_1[\text{pub}](x)$ and the two runtimes are equal¹⁰, output $M_0[\text{pub}](x)$.
 - Else, compute $\beta \leftarrow \text{WE.Dec}(\mathcal{R}_{\mathcal{L}}, (M_0, M_1, \text{pub}), x, \text{ct})$. Output $M_\beta[\text{pub}](x)$.
- Pad $P_{M_0, M_1, \text{ct}}$ as described above and output $\text{pub}, \text{ef-IO}(1^\lambda, P_{M_0, M_1, \text{ct}}[\text{pub}])$.

$\boxed{H_{b,3}}$ This hybrid is identical to $H_{b,2}$, except that we sample $r \leftarrow \{0, 1\}^p$ uniformly when generating ct , instead of setting $r = G(s)$.

It suffices to show that $H_{0,1}$ and $H_{1,1}$ are indistinguishable. For each $b \in \{0, 1\}$, indistinguishability of $H_{b,2}$ and $H_{b,3}$ follows from PRG security. By witness-succinctness, p and the encryption time are polynomial in $\lambda, |M_0|, |M_1|, |\text{pub}|$, and the remaining computation is also polynomial-time. By assumption, we know that $(M_0, M_1, \text{pub}) \notin \mathcal{L}$. Therefore, indistinguishability of $H_{0,3}$ and $H_{1,3}$ follows from witness encryption security, since the output depends on b only through ct . It remains to show the following claim.

Claim 5.9. *For $b \in \{0, 1\}$, for all polynomial-sized adversaries $\mathcal{A}$, we have that:*

$$|\Pr [\mathcal{A}(\Pi) = 1 : \Pi \leftarrow H_{b,1}] - \Pr [\mathcal{A}(\Pi) = 1 : \Pi \leftarrow H_{b,2}]| \leq \text{negl}(\lambda).$$

Proof. We rely on ef-IO security to make this claim. First, we will show the following claim in PV:

$$\begin{aligned} & \forall M_0, M_1, \text{pub}, \text{ct}, b, s, x, \\ & (\text{ct} = \text{WE.Enc}(1^\lambda, \mathcal{R}_{\mathcal{L}}, (M_0, M_1, \text{pub}), N(\lambda), b; G(s))) \rightarrow M_b^{\text{pad}}[\text{pub}](x) = P_{M_0, M_1, \text{ct}}[\text{pub}](x). \end{aligned}$$

We sketch the proof here.

- By definition, for all x , $M_b^{\text{pad}}[\text{pub}](x) = M_b[\text{pub}](x)$.
- If $M_0[\text{pub}](x) = M_1[\text{pub}](x)$ and the runtimes are equal, then $P_{M_0, M_1, \text{ct}}[\text{pub}](x) = M_b[\text{pub}](x)$.
- If $M_0[\text{pub}](x) \neq M_1[\text{pub}](x)$ or the runtimes are not equal, then x is a valid witness for $(M_0, M_1, \text{pub}) \in \mathcal{L}$.
- If $\text{ct} = \text{WE.Enc}(1^\lambda, \mathcal{R}_{\mathcal{L}}, (M_0, M_1, \text{pub}), N(\lambda), b; G(s))$, by correctness of WE, we have that

$$\text{WE.Dec}(\mathcal{R}_{\mathcal{L}}, (M_0, M_1, \text{pub}), x, \text{ct}) = b.$$

- Therefore, $P_{M_0, M_1, \text{ct}}[\text{pub}](x) = M_b[\text{pub}](x)$.
- Hence, for all x , $P_{M_0, M_1, \text{ct}}[\text{pub}](x) = M_b[\text{pub}](x)$.
- By transitivity, $P_{M_0, M_1, \text{ct}}[\text{pub}](x) = M_b^{\text{pad}}[\text{pub}](x)$.

Therefore, we can invoke ef-IO security to complete the proof of indistinguishability.

¹⁰Note that this can be checked in time $O(\min(\text{RT}(M_0[\text{pub}], x), \text{RT}(M_1[\text{pub}], x)))$

Efficiency. Witness-succinctness bounds the encryption time and the number of random bits used in $H_{b,2}$ by $\text{poly}(\lambda, |M_0|, |M_1|, |\text{pub}|)$, independently of $N(\lambda)$. Thus the PRG expansion and ciphertext generation in that hybrid take polynomial time. The remaining size bottleneck is the program $P_{M_0, M_1, \text{ct}}$, whose description has size $\text{poly}(\lambda, |M_0|, |M_1|, |\text{ct}|)$.

- If WE is witness-succinct, then $|\text{ct}| = \text{poly}(\lambda, |M_0|, |M_1|, |\text{pub}|)$. Hence, the obfuscation size is $\text{poly}(\lambda, |M_0|, |M_1|, |\text{pub}|)$, i.e. it is input-succinct (but not description-succinct).
- If WE is fully succinct, then $|\text{ct}| = \text{poly}(\lambda, \log |M_0|, \log |M_1|, \log |\text{pub}|)$. Then the obfuscation is also fully succinct.

This completes the proof. $\square$

5.2.2 From Succinct IO to Succinct Witness Encryption

In this section, we show that the folklore construction (Algorithm 3) of WE from IO preserves full succinctness and PV proof of correctness.

Algorithm 3 Fully succinct WE scheme given a fully succinct IO scheme.

WE.Enc($1^\lambda, \mathcal{R}, x, n, m$): On input the relation $\mathcal{R}$, instance x , witness length bound n and message m ,

- Construct the following program $P_m[\mathcal{R}, x](\cdot)$: on input w ,
 - if $\mathcal{R}(x, w) = 1$, the output m .
 - else, output $\perp$.
- Compute $\Pi = \text{iO}(1^\lambda, P_m, \text{pub} = (\mathcal{R}, x))$.
- Output Π .

WE.Dec($\mathcal{R}, x, w, \text{ct}$) : On input $\mathcal{R}$, instance x , witness w and $\text{ct} = \Pi$:

- Compute $m \leftarrow \Pi[\mathcal{R}, x](w)$, and output m .
-

Proof of security. Semantic security follows trivially from the fact that for $x \notin L$, $P_{m_0}[\mathcal{R}, x](\cdot) \equiv P_{m_1}[\mathcal{R}, x](\cdot)$ are both zero functions. Therefore, we can invoke fully succinct iO security to argue that:

$$\text{iO}(1^\lambda, P_{m_0}[\mathcal{R}, x]) \approx_c \text{iO}(1^\lambda, P_{m_1}[\mathcal{R}, x]).$$

PV proof of correctness. Assuming that the underlying iO scheme has a proof of correctness. Then, it is clear that:

- $\mathcal{U}_{\text{clock}}(1^t, \mathcal{R}, (x, w)) = 1$ implies that $\mathcal{U}_{\text{clock}}(1^{q_1(t)}, P_m[\mathcal{R}, x], w) = m$ for some polynomial q_1 .
- By correctness of iO, there exists q_2 such that

$$\mathcal{U}_{\text{clock}}(1^{q_2(t)}, \Pi[\mathcal{R}, x], w) = \mathcal{U}_{\text{clock}}(1^{q_1(t)}, P_m[\mathcal{R}, x], w) = m.$$

- By definition, we have that $\mathcal{U}_{\text{clock}}(1^{q_2(t)}, \text{WE.Dec}, (\mathcal{R}, x, w, \Pi)) = m$, completing the proof.

Efficiency. Input-succinctness of the underlying IO bounds encryption time by $\text{poly}(\lambda, |\mathcal{R}|, |x|)$, independently of the witness-length bound, so the resulting WE is witness-succinct. If the IO scheme is fully succinct, the ciphertext additionally has size $\text{poly}(\lambda)$, yielding fully succinct WE.

Witness Encryption with correctness with respect to digest. Looking forward to the next section, we will rely on the fact that Algorithm 3, when instantiated via Algorithm 1 (which we showed is universal), has the following form: the ciphertext $\text{ct} = (\text{hk}, h, \Gamma)$ where hk is a hash key under some hash function Hash , $h = \text{Hash}_{\text{hk}}(\mathcal{R}, x)$, and Γ is an obfuscated program. Moreover, the following stronger correctness property which can be shown in PV:

$$\left(\begin{array}{l} (\text{hk}, h, \Pi) \leftarrow \text{WE.Enc}(1^\lambda, \mathcal{R}, x, n, m) \\ |x'| = |x| \\ \text{Hash}_{\text{hk}}(\mathcal{R}, x') = h \\ \mathcal{R}(x', w) = 1 \end{array} \right) \rightarrow \text{WE.Dec}(\mathcal{R}, x', w, \text{ct}) = m.$$

In words, if x and x' have the same hash value under Hash_{hk} , then any witness of x' can also be used to decrypt the ciphertext.

5.2.3 From Succinct IO to Unique-SNARGs with Short CRS

In this section, we describe how to construct SNARGs with a short CRS assuming succinct IO. The construction and analysis are based on the subsequent work of [FJM⁺26].

Ingredients. We use the following ingredients in our construction.

- A input-succinct (resp. fully succinct) IO scheme iO . Recall from Section 5.2.2 that this implies a witness encryption scheme $(\text{WE.Enc}, \text{WE.Dec})$ whose correctness holds with respect to a digest of the instance.
- A puncturable PRF family $(\text{PPRF.Gen}, \text{PPRF.Eval}, \text{PPRF.Punc})$.
- A one-way function f .

We present the construction in Algorithm 4.

PV correctness and Uniqueness. We first argue the correctness. First, it is easy to see that if $\mathcal{U}_{\text{clock}}(1^t, \mathcal{R}, (x, w)) = 1$, then there exists a polynomial q_1 such that one can prove in PV that $\mathcal{U}_{\text{clock}}(1^{q(\lambda, t)}, P_{K, \text{hk}}[\mathcal{R}], (x, w)) = \text{PPRF.Eval}(K, \text{Hash}_{\text{hk}}(\mathcal{R}, x))$. By the correctness of the iO there exists a polynomial q_2 such that $\mathcal{U}_{\text{clock}}(1^{q_2(\lambda, q_1(\lambda, t))}, \tilde{P}[\mathcal{R}], (x, w)) = \text{PPRF.Eval}(K, \text{Hash}_{\text{hk}}(\mathcal{R}, x))$. Finally, it is easy to see that $V_{K, \text{hk}}[\mathcal{R}](x, \text{PPRF.Eval}(K, \text{Hash}_{\text{hk}}(\mathcal{R}, x))) = 1$. This completes the proof of correctness.

The proof of uniqueness follows from the fact that if $\mathcal{R}(x, w) = 1$ then $\pi = \text{PPRF.Eval}(K, \text{Hash}_{\text{hk}}(\mathcal{R}, x))$, which is independent of w .

Soundness. Let $\mathcal{L}$ denote the language corresponding to the NP relation $\mathcal{R}$. Consider some $x^* \notin \mathcal{L}$. First, we consider the following sequence of hybrids which modify the distribution of the common reference string.

Algorithm 4 Witness-succinct (resp. fully succinct) SNARG from input-succinct (resp. fully succinct) IO.

SNARG.Gen($1^\lambda, \mathcal{R}, n_x, n_w$) :

- **Setup hash key:**

- Sample $(\text{hk}, h, \Pi) \leftarrow \text{WE.Enc}(1^\lambda, \mathcal{R}, 0^{n_x}, n_w, 0^{n_\pi})$, where n_x is the input length bound, n_w is the witness length bound, and $n_\pi = \text{poly}(\lambda)$ is the length of the SNARG proof.

- **Construct CRS.**

- Sample $K \leftarrow \text{PPRF.Gen}(1^\lambda)$.
- Construct the following prover algorithm:

$P_{K, \text{hk}}[\mathcal{R}](x, w) :$

- * If $\mathcal{R}(x, w) = 0$, output $\perp$.
- * Else, let $h \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x)$.
- * Output $\text{PPRF.Eval}(K, h)$.

Pad the runtime and size of the program to match the program in H_5 .

- Construct the verification algorithm:

$V_{K, \text{hk}}[\mathcal{R}](x, \pi) :$

- * Let $h \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x)$.
- * If $f(\pi) = f(\text{PPRF.Eval}(K, h))$, accept. Else, reject.

Pad the runtime and size of the program to match the program in H_4 .

- Let $\tilde{P} \leftarrow \text{iO}(1^\lambda, P_{K, \text{hk}}[\mathcal{R}])$ and $\tilde{V} \leftarrow \text{iO}(1^\lambda, V_{K, \text{hk}}[\mathcal{R}])$.
- Output $\text{crs} = (\tilde{P}, \tilde{V})$.

SNARG. $\mathcal{P}(\text{crs}, \mathcal{R}, x, w)$: Output $\tilde{P}[\mathcal{R}](x, w)$.

SNARG. $\mathcal{V}(\text{crs}, \mathcal{R}, x, \pi)$: Output $\tilde{V}[\mathcal{R}](x, \pi)$.

$\boxed{H_1}$ Sample $\text{crs} \leftarrow \text{SNARG.Gen}(1^\lambda, \mathcal{R}, n_x, n_w)$.

$\boxed{H_2}$ Let $h^* \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x^*)$. Let $K^* \leftarrow \text{PPRF.Punc}(K, h^*)$, and let $y^* = \text{PPRF.Eval}(K, h^*)$. Consider the following modified prover and verifier algorithms.

Prover algorithm. $P_{K^*, \text{hk}, h^*, y^*}[\mathcal{R}](x, w) :$

- If $\mathcal{R}(x, w) = 0$, output $\perp$.
- Else, let $h \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x)$.
 - If $h = h^*$, output y^* .
 - Else, output $\text{PPRF.Eval}(K^*, h)$.

Verifier algorithm. $V_{K^*, \text{hk}, h^*, y^*}[\mathcal{R}](x, \pi) :$

- Let $h \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x)$.
- If $h = h^*$,
 - If $f(\pi) = f(y^*)$, accept. Else, reject.
- If $h \neq h^*$,
 - If $f(\pi) = f(\text{PPRF.Eval}(K^*, h))$, accept. Else, reject.

Sample $\tilde{P} \leftarrow \text{iO}(1^\lambda, P_{K^*, \text{hk}, h^*, y^*}[\mathcal{R}])$ and $\tilde{V} \leftarrow \text{iO}(1^\lambda, V_{K^*, \text{hk}, h^*, y^*}[\mathcal{R}])$. Indistinguishability follows from functionality equivalence of the programs in both hybrids.

$\boxed{H_3}$ This hybrid is identical to the previous, except we sample y^* as uniformly random (rather than as $\text{PPRF.Eval}(K, h^*)$). Security follows from the puncturable PRF security.

$\boxed{H_4}$ In this hybrid, let $z^* = f(y^*)$. Change the verification algorithm as follows:

$V_{K^*, \text{hk}, h^*, z^*}[\mathcal{R}](x, \pi) :$

- Let $h \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x)$.
- If $h = h^*$,
 - If $f(\pi) = z^*$, accept. Else, reject.
- If $h \neq h^*$,
 - If $f(\pi) = f(\text{PPRF.Eval}(K^*, h))$, accept. Else, reject.

Sample $\tilde{P} \leftarrow \text{iO}(1^\lambda, P_{K^*, \text{hk}, h^*, y^*}[\mathcal{R}])$ and $\tilde{V} \leftarrow \text{iO}(1^\lambda, V_{K^*, \text{hk}, h^*, z^*}[\mathcal{R}])$. Indistinguishability follows from functionality equivalence of the programs in both hybrids.

$\boxed{H_5}$ Now, sample $\text{ct} = (\text{hk}, h^*, \Pi) \leftarrow \text{WE.Enc}(1^\lambda, \mathcal{R}, x^*, n_w, y^*)$. We now sample the prover algorithm as follows:

$P_{K^*, \text{hk}, h^*, \text{ct}}(x, w) :$

- If $\mathcal{R}(x, w) = 0$, output $\perp$.
- Else, let $h \leftarrow \text{Hash}_{\text{hk}}(\mathcal{R}, x)$.

- If $h = h^*$, output $\text{WE.Dec}(\mathcal{R}, x, w, \text{ct})$.
- Else, output $\text{PPRF.Eval}(K^*, h)$.

Sample $\tilde{P} \leftarrow \text{iO}(1^\lambda, P_{K^*, \text{hk}, h^*, \text{ct}}[\mathcal{R}])$ and $\tilde{V} \leftarrow \text{iO}(1^\lambda, V_{K^*, \text{hk}, h^*, z^*}[\mathcal{R}])$.

To see that the new algorithm is functionally equivalent to the previous algorithm, we use the stronger correctness of the witness encryption as guaranteed in Section 5.2.2: If $\text{Hash}_{\text{hk}}(\mathcal{R}, x) = \text{Hash}_{\text{hk}}(\mathcal{R}, x^*)$ and $\mathcal{R}(x, w) = 1$, then,

$$\text{WE.Dec}(\mathcal{R}, x, w, \text{ct}) = y^*.$$

Therefore, we can rely on iO security.

$\boxed{\text{H}_6}$ This hybrid is identical to the previous, except that we sample $\text{ct} = (\text{hk}, h^*, \Pi) \leftarrow \text{WE.Enc}(1^\lambda, \mathcal{R}, x^*, n_w, \boxed{0})$ instead. This follows from the witness encryption ciphertext indistinguishability since $x^* \notin \mathcal{L}$.

Using one-way function security. Note that H_6 can be sampled given just a one-way function challenge z^* . In this hybrid, any accepting proof for x^* yields a one-way function pre-image. Therefore, by the security of the one-way function and indistinguishability of H_1 and H_6 , we have that any cheating $\mathcal{P}^*$ produces a proof for x^* with probability $\text{negl}(\lambda)$.

Efficiency. It is easy to see that one can pick the SNARG proof length $n_\pi = \text{poly}(\lambda)$. The size of the crs is dominated by the size of the obfuscated prover algorithm. The size of the obfuscated prover program is $O(|K^*|, |\text{ct}|) = O(\text{poly}(\lambda), |\text{ct}|)$.

- If iO is input-succinct, then WE is witness-succinct. Therefore, the size of the crs is $\text{poly}(\lambda, n_w)$.
- If iO is fully succinct, then WE is also fully succinct. Therefore, the size of the crs is $\text{poly}(\lambda)$.

5.2.4 From Unique-SNARGs with Short CRS to Succinct Witness Encryption

Finally, we show that witness-succinct (resp. fully succinct) Unique-SNARGs imply witness-succinct (resp. fully succinct) witness encryption. This transformation is from the work of [CPW23], which shows how to construct a witness encryption scheme given a unique witness map (an object they defined). Since unique-SNARGs give us such a unique witness map, and their transformation applies to us. In Appendix C, we show that their construction preserves succinctness as well as PV correctness.

6 Succinct Computational Secret Sharing

In this section, we present our succinct computational secret sharing schemes. We first recap the definition of computational secret sharing (CSS) in Section 6.1. In Section 6.2, we present a succinct CSS construction for monotone functions which have PV proofs of monotonicity. We then give some instantiations in Section 6.3. Finally, we show in Section 6.4 how to extend our construction and analysis to capture CSS for non-deterministic monotone functions.

6.1 Definitions

We first introduce some definitions, following the terminology from the work of [KNY14, ABI⁺23].

Throughout this section, we will use the following notation. For $x, y \in \{0, 1\}^n$, we write $x \succeq y$ if for all $i \in [n]$, $x_i \geq y_i$. For an $x \in \{0, 1\}^n$, we denote by S_x the subset $\{i : x_i = 1\}$. We say that a function $P : \{0, 1\}^n \rightarrow \{0, 1\}$ 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 $\mathcal{M}_P : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$, we define $P(x) = 1$ if and only if there exists some $w \in \{0, 1\}^m$ such that $\mathcal{M}_P(x, w) = 1$. We say that $\mathcal{M}_P$ is a *monotone non-deterministic function* if the corresponding function P is monotone.

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

Definition 6.1 (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 $P : \{0, 1\}^n \rightarrow \{0, 1\}$, 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 6.2 (Computational secret sharing). A computational secret sharing (CSS) scheme for a class of monotone programs $\mathcal{F}$ is a pair of algorithms $\text{CSS} = (\text{CSS.Share}, \text{CSS.Recon})$ such that:

- $\text{CSS.Share}(1^\lambda, 1^n, P, s) \rightarrow (\text{pp}, \text{sh}_1, \dots, \text{sh}_n)$. This is a randomized polynomial time algorithm which takes as input the security parameter λ , the number of parties n , a monotone program $P \in \mathcal{F}$ and a secret $s \in \{0, 1\}$, and outputs public parameters pp , and n shares $\{\text{sh}_i\}_{i \in [n]}$. Each share sh_i is given to party i , and pp is a public string available to all parties.
- $\text{CSS.Recon}(P, x, \text{pp}, \{\text{sh}_i\}_{i \in S_x}) \rightarrow s$. 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 $[n]$, public parameters pp , and a subset of shares $\{\text{sh}_i\}_{i \in S_x}$. 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 $x \in \{0, 1\}^n$, and a program P such that $P(x) = 1$, we have that

$$\Pr \left[s \leftarrow \text{CSS.Recon}(P, x, \text{pp}, \{\text{sh}_i\}_{i \in S_x}) : (\text{pp}, \text{sh}_1, \dots, \text{sh}_n) \leftarrow \text{CSS.Share}(1^\lambda, 1^n, P, s) \right] = 1.$$

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

- On input $1^\lambda, 1^n$ 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

$$(\text{pp}, \text{sh}_1, \dots, \text{sh}_n) \leftarrow \text{CSS.Share}(1^\lambda, 1^n, P, s).$$

Send $\text{pp}, \{\text{sh}_i\}_{i \in S_x}$ 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 $1/2 + \text{negl}(\lambda)$ for large enough λ .

We first consider *deterministic* monotone functions P . We will consider non-deterministic functions in Section 6.4.

PV proofs of monotonicity. 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 EF-IO.

Definition 6.3 (Monotone function 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 $\text{Member}_{\mathcal{F}}$ such that $\text{Member}_{\mathcal{F}}(P) = 1$ if and only if $P \in \mathcal{F}$.
- There is a PV proof that:

$$\left(\begin{array}{l} \text{Member}_{\mathcal{F}}(P) = 1 \\ \wedge P(x) = 0 \\ \wedge \partial\text{Less}(y, x) \end{array} \right) \vdash_{\text{PV}} (P(y) = 0).$$

where $\partial\text{Less}(y, x)$ is a PV function formalizing $y \preceq x$.

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. We recall the standard notion of certificate complexity for Boolean functions (see, e.g., [AB09]). In the monotone setting, a natural zero-certificate for an input x with $P(x) = 0$ is a dominating input $y \succeq x$ such that $P(y) = 0$ (equivalently, one may take y to be a maximal zero-input, or maxterm). Traditional monotone zero-certificate complexity measures such a certificate combinatorially. In our setting, we instead consider a refinement of this notion in which the size of the zero-certificate is measured by its time-bounded conditional Kolmogorov complexity given P , rather than by its Hamming weight.

Definition 6.4 (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 now define zero-complexity of a monotone function in terms of conditional Kolmogorov complexity.

Definition 6.5 (*t*-Time Zero-Complexity of a Monotone Function). Given a monotone function $P : \{0, 1\}^n \rightarrow \{0, 1\}$, we define the zero-certificate complexity of P to be

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

where K^t is the t -time bounded Kolmogorov complexity, as defined in Definition 6.4.

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

Remark 6.6. We note that it might be 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.

6.2 Secret Sharing Scheme for Functions with PV Proofs of Monotonicity

Our construction will utilize the following main ingredients.

- A statistically binding commitment scheme with PV proof of binding (Definition 3.16):

$$\left(\begin{array}{l} b \in \{0, 1\}, \\ \text{pk} \leftarrow \text{Com.Gen}(1^\lambda), \\ c \leftarrow \text{Com.Enc}(\text{pk}, b; r) \end{array} \right) \vdash_{\text{PV}} \forall r', \quad c \neq \text{Com.Enc}(\text{pk}, 1 - b; r').$$

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.

- A somewhere statistically binding hash family with PV proof of binding (Theorem 3.21), where the input alphabet is the range of commitments produced by the commitment scheme from the previous bullet point.
- A fully succinct EF-IO scheme iO (Definition 4.4).
- A PRG G .

In the following, let $P : \{0, 1\}^n \rightarrow \{0, 1\}$ be a monotone function.

For notational convenience, we will let our CSS.Share algorithm take as input additional parameters 1^ℓ and t . We choose $t = \text{poly}(\lambda, |P|, n)$, where the polynomial may depend on the fixed family $\mathcal{F}$ and is large enough to bound the running times of P on n -bit inputs and of $\text{Member}_{\mathcal{F}}$. We take $\ell \geq Z_{K^t}(P)$ (as defined in Definition 6.5). Recall that one can always choose $\ell = n$.

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

- **Construct commitments.**

- Sample $\text{pk} \leftarrow \text{Com.Gen}(1^\lambda)$.
- For $i = 1, 2, \dots, n$, sample randomness $r_i \leftarrow \{0, 1\}^\lambda$, and $\text{com}_i \leftarrow \text{Com.Enc}(\text{pk}, 1; r_i)$.
- Sample $\text{hk} \leftarrow \text{SSB.Gen}(1^\lambda)$, and compute $\text{dig} \leftarrow \text{SSB.Hash}(\text{hk}, (\text{com}_1, \dots, \text{com}_n))$. If

dig = $\perp$, set $\rho_i = \perp$ for every $i \in [n]$; otherwise, let ρ_i denote the local opening from dig to com_i .

– For $i \in [n]$, set $\pi_i = (\text{com}_i, \rho_i, r_i)$.

• **Construct obfuscated program.**

– Sample $\text{pp} \leftarrow \{0, 1\}^\ell$.

– Consider the program $M_{\text{pk}, \text{hk}, \text{dig}, s}[\text{pub} = (P, \text{pp})]$ which on input $x \in \{0, 1\}^n$ and $\{\pi_i\}_{i \in S_x}$ does the following:

- * If $\text{dig} = \perp$, output s .
- * Abort if $P(x) = 0$.
- * For each $i \in S_x$:
 - Parse $\pi_i = (\text{com}_i, \rho_i, r_i)$.
 - Check that $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_i, \rho_i) = 1$.
 - Check that $\text{com}_i = \text{Com.Enc}(\text{pk}, 1; r_i)$.
 - Abort if any test fails.
- * If $P(x) = 1$, then output s . Else, output $\perp$.

Pad the program to have runtime $\text{poly}(\text{RT}(P, x), n, t, \lambda)$, where the exact poly depends on the programs in the security analysis.

– Set $\Gamma = \text{ef-IO}_{\rho, T}(M_{\text{pk}, \text{hk}, \text{dig}, s}, \text{pub} := (P, \text{pp}))$, where $\rho = \text{poly}(\lambda)$ and $T = \text{poly}(\lambda, t)$.

• **Create shares:**

- Set $\text{sh}_i = \{\pi_i, \Gamma\}$.
- Output $(\text{pp}, \{\text{sh}_i\}_{i \in [n]})$.

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

- Parse $(\pi_i, \Gamma) \leftarrow \text{sh}_i$ for $i \in S_x$.
- Compute $s \leftarrow \Gamma[\text{pub} = (P, \text{pp})](x, \{\pi_i\}_{i \in S_x})$.
- Output s .

Theorem 6.7 (Succinct secret sharing for P). *Assuming*

- *fully succinct EF-IO (as in Definition 4.4),*
- *statistically-binding commitment schemes with PV proof of binding (Definition 3.16),*
- *index-hiding somewhere statistically binding hash family with PV proof of binding (Theorem 3.21),*
- *pseudorandom-generators,*

there exists an n -party computational secret sharing for deterministic monotone function families $\mathcal{F}$ which have PV proofs of monotonicity (Definition 6.3) such that for $P \in \mathcal{F}$:

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

Proof. Recall that in this setting, the function P is *deterministic*, and the obfuscated program does not take any input w .

Correctness follows from the correctness of the hash openings and obfuscation when $\text{dig} \neq \perp$, and from the first instruction of the obfuscated program when $\text{dig} = \perp$. In the security analysis below, all hybrid experiments output $\perp$ immediately if $\text{dig} = \perp$. By correctness of the hash family, this changes the real experiment with only negligible probability.

Fix a monotone function $P \in \mathcal{F}$, and suppose the adversary picks some $y \in \{0, 1\}^n$ such that $P(y) = 0$. Consider a string $z \in \{0, 1\}^n$ such that $S_z \supseteq S_y$, and $P(z) = 0$, and $K^t(z|P) \leq Z_{K^t}(P)$, and let (Π, τ) be the program achieving this time-bounded Kolmogorov complexity, i.e. $z \leftarrow \Pi[P](\tau)$ in time t . We will use this program Π in our proof.

We will first outline the main hybrids in the proof.

H₁ Construct the pp and shares $\{\text{sh}_i\}_{i \in S}$ as in the real scheme.

H₂ In the CSS.Share algorithm, sample the commitments instead as follows:

- For $i = 1, 2, \dots, n$, sample randomness $r_i \leftarrow \{0, 1\}^\lambda$. For $i \in S_y$, sample $\text{com}_i \leftarrow \text{Com.Enc}(\text{pk}, 1; r_i)$. For $i \notin S_y$, sample $\text{com}_i \leftarrow \text{Com.Enc}(\text{pk}, \boxed{0}; r_i)$.

In words, we change the commitments corresponding to the hidden shares to commitments to zero rather than 1. Since the randomness $\{r_i\}_{i \in \overline{S_y}}$ is hidden from the adversary's view, we can invoke the computational hiding property of the commitment scheme.

H₃ Switch pp from a uniform random string to be $(\Pi, \tau) \oplus G(\text{sk})$ (recall $\Pi[P](\tau)$ outputs some $z \in \{0, 1\}^n$ in time t), for some PRG seed sk . This follows from PRG security.

H₄ Sample a PRG seed sd , and use sd to sample all $\{r_i\}_{i \in [n]}$ and sk (i.e. $G(\text{sd}) \rightarrow (r_1, \dots, r_n, \text{sk})$). This follows from the pseudorandomness of the PRG.

H₅ In this hybrid, we change the program that is being obfuscated to the following program $M'_{\text{pk}, \text{hk}, \text{dig}, s, \boxed{\text{sd}}}$ [pub = (P, pp)] which works as follows on input $x \in \{0, 1\}^n$ and $\{\pi_i\}_{i \in S_x}$:

- Abort if $P(x) = 0$.
- Expand sd to $(r'_1, \dots, r'_n, \text{sk})$ using the PRG G .
- Compute $(\Pi, \tau) \leftarrow \text{pp} \oplus G(\text{sk})$, and $z \leftarrow \Pi[P](\tau)$.
- For each $i \in S_x$:
 - Parse $\pi_i = (\text{com}_i, \rho_i, r_i)$.
 - Check that $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_i, \rho_i) = 1$.
 - Check that $\text{com}_i = \text{Com.Enc}(\text{pk}, 1; r_i)$.
 - Abort if any test fails.
- If $P(x) = 1$, then output s . Else, output $\perp$.

Since the program is simply padded with additional, independent computation, one can show that the two machines are $(O(1), \text{poly}(t, \lambda))$ -EF-equivalent. Therefore, we can rely on EF-IO security.

H₆ In this hybrid, we change the program that is being obfuscated to the following program which works as follows on input $x \in \{0, 1\}^n$ and $\{\pi_i\}_{i \in S_x}$:

- Abort if $P(x) = 0$.
- Expand sd to $(r'_1, \dots, r'_n, \text{sk})$ using the PRG G .
- Compute $(\Pi, \tau) \leftarrow \text{pp} \oplus G(\text{sk})$, and $z \leftarrow \Pi[P](\tau)$.
- For $i \notin S_z$, compute $\text{com}'_i \leftarrow \text{Com.Enc}(\text{pk}, 0; r'_i)$.
- For each $i \in S_x$:
 - Parse $\pi_i = (\text{com}_i, \rho_i, r_i)$. If $i \notin S_z$, replace com_i with com'_i .
 - Check that $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_i, \rho_i) = 1$.
 - Check that $\text{com}_i = \text{Com.Enc}(\text{pk}, 1; r_i)$.
 - Abort if any test fails.
- If $P(x) = 1$, then output s . Else, output $\perp$.

Lemma 6.8. $H_5 \approx_c H_6$.

This is the most involved step in the proof. At a high level, we will rely on the PV-friendliness of the SSB and the fact that $\{\text{com}_i\}_i$ can be computed. We prove this lemma in Section 6.2.1.

H₇ We now change the obfuscated program as follows. The new program $M''_{\text{pk}, \text{hk}, \text{dig}, s, \text{sd}}[\text{pub} = (P, \text{pp})]$ on input $x \in \{0, 1\}^n$ and $\{\pi_i\}_{i \in S_x}$:

- Abort if $P(x) = 0$.
- Expand sd to $(r'_1, \dots, r'_n, \text{sk})$ using the PRG G .
- Compute $(\Pi, \tau) \leftarrow \text{pp} \oplus G(\text{sk})$, and $z \leftarrow \Pi[P](\tau)$.
- For $i \notin S_z$, compute $\text{com}'_i \leftarrow \text{Com.Enc}(\text{pk}, 0; r'_i)$.
- For each $i \in S_x$:
 - Parse $\pi_i = (\text{com}_i, \rho_i, r_i)$. If $i \notin S_z$, replace com_i with com'_i .
 - Check that $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_i, \rho_i) = 1$.
 - Check that $\text{com}_i = \text{Com.Enc}(\text{pk}, 1; r_i)$.
 - Abort if any test fails.
- If $S_x \not\subseteq S_z$, abort.
- If $P(x) = 1$, then output s . Else, output $\perp$.

This follows from the PV proof of statistical binding of the commitment scheme and the security of $(\text{poly}(\lambda), \text{poly}(t, \lambda))$ -EF security of EF-IO.

H₈ Add the check that $P(z) = 0$ in the obfuscated program. This is PV-equivalent by construction of $P, (\Pi, \tau)$ and z .

H₉ At this point, it is clear that the last line of the program is only reached if $S_x \subseteq S_z$ and if $P(z) = 0$. We can now change the obfuscated program to one that **always outputs $\perp$** , therefore removing s from the view of the adversary.

We can argue this via PV-equivalence of the programs by relying on the fact that $\mathcal{F}$ is a deterministic monotone function class with a PV proof of monotonicity.

At this point, the secret s is no longer in the view of the adversary, and the adversary only has a $\frac{1}{2}$ probability of guessing s . Therefore, by the indistinguishability of the hybrids, we have that the CSS scheme is in fact secure for monotone families with proofs of monotonicity. $\square$

6.2.1 Proof of Lemma 6.8

We proceed in a few hybrids.

$H_{5,j,1}$ In this hybrid, we change the program that is being obfuscated to the following program which works as follows on input $x \in \{0, 1\}^n$ and $\{\pi_i\}_{i \in S_x}$:

- Abort if $P(x) = 0$.
- Expand sd to $(r'_1, \dots, r'_n, sk)$ using the PRG G .
- Compute $(\Pi, \tau) \leftarrow pp \oplus G(sk)$, and $z \leftarrow \Pi[P](\tau)$.
- For $i \notin S_z$, compute $com'_i \leftarrow \text{Com.Enc}(pk, 0; r'_i)$.
- For each $i \in S_x$:
 - Parse $\pi_i = (com_i, \rho_i, r_i)$. If $i \notin S_z$ and $i < j$, replace com_i with com'_i .
 - Check that $\text{SSB.Ver}(hk, dig, com_i, \rho_i) = 1$.
 - Check that $com_i = \text{Com.Enc}(pk, 1; r_i)$.
 - Abort if any test fails.
- If $P(x) = 1$, then output s . Else, output $\perp$.

$H_{5,j,2}$ Switch the SSB key to be binding on com_j (i.e. the j th index). Indistinguishability follows from SSB index-hiding.

$H_{5,j,3}$ In this hybrid, we instead obfuscate the following program:

- Abort if $P(x) = 0$.
- Expand sd to $(r'_1, \dots, r'_n, sk)$ using the PRG G .
- Compute $(\Pi, \tau) \leftarrow pp \oplus G(sk)$, and $z \leftarrow \Pi[P](\tau)$.
- For $i \notin S_z$, compute $com'_i \leftarrow \text{Com.Enc}(pk, 0; r'_i)$.
- For each $i \in S_x$:
 - Parse $\pi_i = (com_i, \rho_i, r_i)$. If $i \notin S_z$ and $i \leq j$, replace com_i with com'_i .
 - Check that $\text{SSB.Ver}(hk, dig, com_i, \rho_i) = 1$.
 - Check that $com_i = \text{Com.Enc}(pk, 1; r_i)$.
 - Abort if any test fails.
- If $P(x) = 1$, then output s . Else, output $\perp$.

We establish indistinguishability via EF-IO security. If $j \in S_z$, then the programs are clearly EF-equivalent. We therefore consider the case where $j \notin S_z$. Let $\text{params} = (hk, dig, pk, sd)$. Define $R(\text{params}, \text{aux} = (r, j, com'_j))$ to first expand sd to $(r'_1, \dots, r'_n, sk)$ using G , and output 1 if

$$\begin{aligned} (hk, td) &\leftarrow \text{SSB.TGen}(1^\lambda, j; r), \\ com'_j &= \text{SSB.Ext}(hk, td, dig) = \text{Com.Enc}(pk, 0; r'_j). \end{aligned}$$

Let $Q_{\text{params}}^{(1)}$ be the program in the previous hybrid, and let $Q_{\text{params}}^{(2)}$ be the program in this hybrid. The above follows via the PV proof of the fact that:

$$\begin{aligned} R(\text{params}, \text{aux}) &= 1 \vdash_{\text{PV}} \\ Q_{\text{params}}^{(1)}[\text{pub} = (P, pp)](x, \{\pi_i\}_{i \in S_x}) &= Q_{\text{params}}^{(2)}[\text{pub} = (P, pp)](x, \{\pi_i\}_{i \in S_x}). \end{aligned}$$

We sketch the proof below:

- If $R(\text{params}, \text{aux}) = 1$, one can use the PV proof of binding to establish that if $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_j, \rho_j) = 1$, then $\text{com}_j = \text{SSB.Ext}(\text{hk}, \text{td}, \text{dig})$.
- By definition of R , $\text{com}'_j = \text{Com.Enc}(\text{pk}, 0; r'_j)$, which is the commitment recomputed by the programs.
- Therefore, the check $\text{com}_j = \text{Com.Enc}(\text{pk}, 1; r_j)$ occurs only if $\text{com}_j = \text{com}'_j$.
- Therefore, the output of the two programs is equivalent.

Hence, we can use $(\text{poly}(\lambda), \text{poly}(\lambda, t))$ -EF-equivalence to invoke EF-IO security.

Clearly $H_{5,j,3} \equiv H_{5,j+1,1}$, $H_5 \equiv H_{5,1,1}$ and $H_6 \equiv H_{5,n,3}$. Therefore, the indistinguishability follows via a hybrid argument.

6.3 Instantiations of Theorem 6.7

In this section, we show how Theorem 6.7 applies to the class of monotone circuits (Section 6.3.1) and the bipartite matching access structure (Section 6.3.2).

6.3.1 Monotone circuits

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

Lemma 6.9 (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:*

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

where MonoCirc is a PV function deciding whether a circuit is a monotone circuit, $\partial\text{Less}(x, y)$ 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 a 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, $U(C, x) = \text{At}(x, i)$ for some index i , where $\text{At}(x, i)$ 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

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

since $\partial\text{Less}(x, y) = 1 \rightarrow \text{At}(x, i) \leq \text{At}(y, i)$ 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 $(\text{MonoCirc}(C') = 1 \wedge \partial\text{Less}(x, y) = 1) \rightarrow \partial\text{Less}(U(C', x), U(C', y)) = 1$.

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 $g(\text{At}(U(C', \cdot), i), \text{At}(U(C', \cdot), j))$. Since g is either an $\wedge$ or an $\vee$, we can derive in PV that

$$g(\text{At}(U(C', x), i), \text{At}(U(C', x), j)) \leq g(\text{At}(U(C', y), i), \text{At}(U(C', y), j))$$

from $\partial\text{Less}(U(C', x), U(C', y)) = 1$. 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 (See Appendix A.1). This finishes the proof sketch. $\square$

6.3.2 Bipartite matching access structure

We now show that we can also construct succinct secret sharing for the bipartite matching access structure.

Definition 6.10 (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_{general} is the access structure which takes input $E \in \{0, 1\}^{\binom{n}{2}}$, and

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

One can efficiently implement Γ_{general} via Edmonds' matching algorithm [Edm65].

Definition 6.11 (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 [CL12] for details).

Lemma 6.12 (PV proof of monotonicity of bipartite matching access structure, [CL12]). *The monotonicity of the bipartite matching access structure can be proven in theory PV. Namely, the following statement can be proven in PV:*

$$\partial\text{Less}(x, y) = 1 \rightarrow (\Gamma(x) = 1 \rightarrow \Gamma(y) = 1)$$

where $\partial\text{Less}(x, y)$ 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 [CL12] formalized the correctness of augmenting-path algorithm in PV. Hence, we have the following formulas proven in PV:

$$\Gamma(E) = 1 \rightarrow (\text{Hungarian}(E) = w \rightarrow \text{Verify}(E, w) = 1), \quad (4)$$

$$\text{Verify}(E, w') = 1 \rightarrow \Gamma(E) = 1 \quad (5)$$

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 $x, y \in \{0, 1\}^{n^2}$ with $2n$ vertices and $\partial\text{Less}(x, y) = 1$. If $\Gamma(x) = 1$, then we have $\text{Hungarian}(x) = w \rightarrow \text{Verify}(x, w) = 1$. Hence, w is a perfect matching in x , and from $\partial\text{Less}(x, y) = 1$, we know that x is a subgraph of y . Therefore, we also have $\text{Verify}(y, w) = 1$. Applying Equation (5), 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 ($\text{Verify}(x, w) = 1 \wedge \partial\text{Less}(x, y) = 1 \rightarrow \text{Verify}(y, w) = 1$) can be formalized in PV. $\square$

Remark 6.13. By a similar argument, if one can show that Edmonds' algorithm for matchings [Edm65] has a PV proof of correctness, then one can show that Γ_{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 6.14 (Zero-complexity of bipartite matching access structure.). *There exists $t = \text{poly}(n)$ such that: $Z_{K^t}(\Gamma_{n,n}) \leq 2n + O(1)$.*

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 S in the subgraph defined by E . In other words, E contains no edges from $S \times (V_2 \setminus N_E(S))$. Consider the zero-certificate obtained by the following procedure:

- Take as input S and $V_2 \setminus N_E(S)$.
- Let $W = S \times (V_2 \setminus N_E(S))$.
- 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 $2n + O(1)$. $\square$

Corollary 6.15. *From the same assumptions as in Theorem 6.7, there exists an 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 $\text{poly}(\lambda, \log n)$.

Proof. Combining Theorem 6.7 with Lemma 6.12 and Lemma 6.14, the result follows immediately. $\square$

6.4 Secret Sharing for NP via Fully Succinct IO

Via a nearly identical construction and security proof, one can construct a secret sharing scheme for NP via fully succinct IO (rather than EF-IO) where the shares are independent of the size of the policy.

Recall that for a non-deterministic *monotone* NP relation $\mathcal{M}_P : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$, we define the induced access structure $P : \{0, 1\}^n \rightarrow \{0, 1\}$ by setting $P(x) = 1$ if and only if there exists some $w \in \{0, 1\}^m$ such that $\mathcal{M}_P(x, w) = 1$. One can extend the CSS definition to capture the NP setting by allowing the reconstruction algorithm Recon to additionally take a witness w as input.

Theorem 6.16 (Succinct secret sharing for NP). *Assuming*

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

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

- *Public parameters consist of a uniformly random string of length $Z_{K^t}(P)$ for some time parameter t .*
- *Each share size is $\text{poly}(\lambda, \log n, \log t, \log |\mathcal{M}_P|)$.*
- *The reconstruction algorithm runs in time $\text{poly}(\text{RT}(\mathcal{M}_P, (x, w)), n, t, \lambda)$.*

We state the full construction, with the main differences highlighted in **blue**. Consider a monotone NP relation $\mathcal{M}_P : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$ with induced access structure $P : \{0, 1\}^n \rightarrow \{0, 1\}$.

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

• **Construct commitments.**

- Sample $\text{pk} \leftarrow \text{Com.Gen}(1^\lambda)$.
- For $i = 1, 2, \dots, n$, sample randomness $r_i \leftarrow \{0, 1\}^\lambda$, and $\text{com}_i \leftarrow \text{Com.Enc}(\text{pk}, 1; r_i)$.
- Sample $\text{hk} \leftarrow \text{SSB.Gen}(1^\lambda)$, and compute $\text{dig} \leftarrow \text{SSB.Hash}(\text{hk}, (\text{com}_1, \dots, \text{com}_n))$. If $\text{dig} = \perp$, set $\rho_i = \perp$ for every $i \in [n]$; otherwise, let ρ_i denote the local opening from dig to com_i .
- For $i \in [n]$, set $\pi_i = (\text{com}_i, \rho_i, r_i)$.

• **Construct obfuscated program.**

- Sample $\text{pp} \leftarrow \{0, 1\}^\ell$.
- Consider the program $M_{\text{pk}, \text{hk}, \text{dig}, s}[\text{pub} = (P, \text{pp})]$ which on input $x \in \{0, 1\}^n$, $w \in \{0, 1\}^m$ and $\{\pi_i\}_{i \in S_x}$ does the following:
 - * If $\text{dig} = \perp$, output s .
 - * Abort if $\mathcal{M}_P(x, w) = 0$.
 - * For each $i \in S_x$:
 - Parse $\pi_i = (\text{com}_i, \rho_i, r_i)$.

• Check that $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_i, \rho_i) = 1$. • Check that $\text{com}_i = \text{Com.Enc}(\text{pk}, 1; r_i)$. • Abort if any test fails. * If $\mathcal{M}_P(x, w) = 1$, then output s. Else, output $\perp$. Pad the program to have runtime $\text{poly}(\text{RT}(\mathcal{M}_P, (x, w)), n, t, \lambda)$, where the exact poly depends on the programs in the security analysis. – Set $\Gamma = \text{iO}(M_{\text{pk}, \text{hk}, \text{dig}, s}, \text{pub} := (P, \text{pp}))$. • Create shares: – Set $\text{sh}_i = \{\pi_i, \Gamma\}$. – Output $(\text{pp}, \{\text{sh}_i\}_{i \in [n]})$. <u>$\text{CSS.Recon}(P, x, w, \text{pp}, \{\text{sh}_i\}_{i \in S_x})$:</u> • Parse $(\pi_i, \Gamma) \leftarrow \text{sh}_i$ for $i \in S_x$. • Compute $s \leftarrow \Gamma[\text{pub} = (P, \text{pp})](x, w, \{\pi_i\}_{i \in S_x})$. • Output s.

Proof sketch of Theorem 6.16. Correctness and the convention that all security hybrids abort on hash failure are as in the proof of Theorem 6.7.

The first few hybrids H_1 to H_7 are identical (except that all checks against $P(x)$ are now checks against $\mathcal{M}_P(x, w)$). However, since we are relying on fully succinct iO for all Turing machines, we do not require the corresponding PV proof of binding of the commitment scheme/SSB scheme to assert PV-equivalence. At this point, the obfuscated program is as follows:

• Abort if $\mathcal{M}_P(x, \overline{w}) = 0$. • Expand sd to $(r'_1, \dots, r'_n, \text{sk})$ using the PRG G. • Compute $(\Pi, \tau) \leftarrow \text{pp} \oplus G(\text{sk})$, and $z \leftarrow \Pi[P](\tau)$. • For $i \notin S_z$, compute $\text{com}'_i \leftarrow \text{Com.Enc}(\text{pk}, 0; r'_i)$. • For each $i \in S_x$: – Parse $\pi_i = (\text{com}_i, \rho_i, r_i)$. – Check that $\text{SSB.Ver}(\text{hk}, \text{dig}, \text{com}_i, \rho_i) = 1$. – If $i \notin S_z$, replace com_i with com'_i. Check that $\text{com}_i = \text{Com.Enc}(\text{pk}, 1; r_i)$. – Abort if any test fails. • If $S_x \not\subseteq S_z$, abort. • If $\mathcal{M}_P(x, \overline{w}) = 1$, then output s. Else, output $\perp$.

By construction, we know that $P(z) = 0$. Therefore, by monotonicity of P , we have that $P(x) = 0$ for all $x \preceq z$. Since the last line of the program is only reached if $x \preceq z$, we conclude that there does not exist any witness w such that $\mathcal{M}_P(x, w) = 1$. Hence, the above program is functionally equivalent to one that always outputs $\perp$. One can then rely on iO security to replace the obfuscated program with one that does not contain s , therefore completing the proof. $\square$

7 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 5. We then introduce a notion of “ γ -closeness”, and we show how to analyze Algorithm 5 in these settings in Lemma 7.2 and Theorem 7.3. We then show several corollaries to high-rate obfuscation in Section 7.2.

7.1 Construction and Analysis

We first give a general construction for necessary padding in Algorithm 5.

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 4.5). Looking forward, we will get different efficiency and security guarantees based on which notion of IO we use.
- A puncturable PRF family $(\text{PPRF.Gen}, \text{PPRF.Eval}, \text{PPRF.Punc})$. We will additionally assume that the family has a PV proof of correctness (as defined in Definition 3.13), as this is achieved by the standard GGM construction of PRFs from one-way functions (see Lemma 3.14).

Construction. We can now describe the construction in Algorithm 5. 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.

When URS is omitted from the syntax, it is sampled uniformly at random from $\{0, 1\}^{\ell - |M|}$, where M denotes the secret part of the split-description program $M[\text{pub}]$.

Definition 7.1 (γ -closeness). Let $M_1[\text{pub}] = \{M_{1,\lambda}[\text{pub}_\lambda]\}_\lambda$ and $M_2[\text{pub}] = \{M_{2,\lambda}[\text{pub}_\lambda]\}_\lambda$ be a pair of Turing machine families with split descriptions satisfying that for all x , $\text{RT}(M_1[\text{pub}], x) = \text{RT}(M_2[\text{pub}], x)$.

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

Lemma 7.2. Suppose one-way functions exist. Consider programs $M_0 = \{M_{0,\lambda}\}_\lambda$ and $M_1 = \{M_{1,\lambda}\}_\lambda$ that are adjacent (i.e. differ in at most $\tau = O(1)$ bits). Then,

$$\text{padO}(1^\lambda, M_0[\text{pub}], 1^\gamma) \approx_c \text{padO}(1^\lambda, M_1[\text{pub}], 1^\gamma)$$

for $\gamma \geq \max(|M_0|, |M_1|)$. Moreover, the size of the program $\Gamma \leftarrow \text{padO}(1^\lambda, M_0[\text{pub}], 1^\gamma)$ is as follows:

- Assuming fully succinct IO, $|\Gamma| = \gamma + \text{poly}(\lambda, \tau, \log \gamma)$.

Algorithm 5 iO with Padding via a Uniform Random String (URS)

Parameters:

- Let ℓ be some padding parameter for the *unobfuscated* program.
- Let $\tau = O(1)$ denote an upper bound on the number of bits that change in the adjacent-machine hybrids.
- Let $\text{Pad}(M, 1^\kappa)$ denote a program which is functionally equivalent to M of size $|M| + \kappa$.

$\text{padO}(1^\lambda, M[\text{pub}], 1^\ell, \text{URS})$:

- Let $S = |M|$.
- Sample $k \leftarrow \text{PPRF.Gen}(1^\lambda)$ such that $\text{PPRF.Eval}(k, y) \in \{0, 1\}$.
- Let $\text{ct} = \{M_{[i]} \oplus \text{PPRF.Eval}(k, i)\}_{i \in [S]}$.
- Truncate URS to length $\ell - |M|$.
- Set $r = \text{ct} \parallel \text{URS}$.
- Let $U_{k,S}$ be the machine that takes as public input (pub, r) and input x and does the following:
 - Parse $r = (\text{ct}, \text{URS})$ by taking the first S bits of r to be ct .
 - Ignore URS, and compute $M \leftarrow \{\text{ct}[i] \oplus \text{PPRF.Eval}(k, i)\}_{i \in [S]}$.
 - Output $M[\text{pub}](x)$.

Moreover, pad the runtime of $U_{k,S}$ to match the runtime of the programs in the hybrids of Lemma 7.2.

- Let $\Gamma = \text{iO.Obf}(1^\lambda, \text{Pad}(U_{k,S}, 1^\kappa), (\text{pub}, r))$, where $\kappa = \text{poly}(\lambda, \tau, \log \ell)$ is chosen large enough for the hybrids in Lemma 7.2.
 - Output (Γ, r) .
-

- Assuming description-succinct IO, $|\Gamma| = \gamma + \text{poly}(\lambda, \tau, n, \log \gamma)$, 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. Throughout this proof, $M_{[i]}$ denotes the i th bit of the description of M , whereas $M[\text{pub}]$ denotes M run with public input pub .

Suppose we have two functionally equivalent adjacent machines $M_0[\text{pub}]$ and $M_1[\text{pub}]$ where $|M_b| = S_b$. Without loss of generality, suppose that $S_0 \geq S_1$ and let $S := S_0$. Suppose that the two machines differ on indices $i_1, \dots, i_\tau \in [S]$. Let $I = \{i_j\}_{j \in [\tau]}$. Let $\beta_j^{(0)} = M_{0[i_j]}$, and let $\beta_j^{(1)} = M_{1[i_j]}$ if $i_j \leq S_1$ and $\beta_j^{(1)} = \emptyset$ otherwise. We use the shorthand $L_b = \{\beta_j^{(b)}\}_{j \in [\tau]}$.

$\boxed{\text{H}_1}$ Sample $(\Gamma, r) \leftarrow \text{padO}(1^\lambda, M_0[\text{pub}], 1^\gamma)$ as in Algorithm 5.

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

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

$U_{k,S_0,S_1,I,L_0,L_1} :$

- Parse $r = (\text{ct}, \text{URS})$ 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 $M_{[i]} \leftarrow \text{ct}[i] \oplus \text{PPRF.Eval}(k, i)$.
- Output $M[\text{pub}](x)$.

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

H₄ Compute $k\{I\} \leftarrow \text{PPRF.Punc}(k, I)$, and hardcode $k\{I\}$ instead of k in the obfuscated program.

This follows from the fact that the obfuscated program does not need $\text{PPRF.Eval}(k, i_j)$ 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 EF-equivalent by Lemma 3.14) and indistinguishability follows from iO security.

H₅ For $i_j \in I$, switch $\text{ct}[i_j]$ to $\text{PPRF.Eval}(k, i_j) \oplus \beta_j^{(1)}$ if $i_j \leq S_1$, 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 punctured key $k\{I\}$).

H₆ In this hybrid, we change the obfuscated program to use L_1 instead of L_0 .

$U_{k\{I\},S_0,S_1,I,L_0,L_1} :$

- Parse $r = (\text{ct}, \text{URS})$ by taking the first S_1 bits of r to be ct .
- For $i_j \in I$ and $i_j \leq S_1$, set $M_{[i_j]} = \beta_j^{(1)}$.
- For $i \leq S_1$ and $i \notin I$, set $M_{[i]} \leftarrow \text{ct}[i] \oplus \text{PPRF.Eval}(k\{I\}, i)$.
- Output $M[\text{pub}](x)$.

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 $M_0[\text{pub}] \equiv M_1[\text{pub}]$. Hence, indistinguishability follows from iO security.

H₇ Replace the punctured key $k\{I\}$ with the original key k in the obfuscated program. Indistinguishability follows from the (PV) correctness of the puncturing.

H₈ Sample $(\Gamma, r) \leftarrow \text{padO}(1^\lambda, M_1[\text{pub}], 1^\gamma)$. 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 7.3. Consider two programs $M_0 = \{M_{0,\lambda}\}_\lambda$ and $M_1 = \{M_{1,\lambda}\}_\lambda$ that are $\gamma(\lambda)$ -close. Then,

$$\text{padO}(1^\lambda, M_0[\text{pub}], 1^\ell) \approx_c \text{padO}(1^\lambda, M_1[\text{pub}], 1^\ell)$$

for $\ell \geq \gamma$. Moreover, the size of the program depends on the version of IO used, just as in Lemma 7.2.

Proof. If M_0 and M_1 are $\gamma(\lambda)$ -close, by definition, then there exists a sequence $M'_1, M'_2, \dots, M'_n$ for $n = \text{poly}(\lambda)$ such that M'_i and M'_{i+1} are adjacent and $\max_i |M'_i| \leq \gamma$. By Lemma 7.2, since M'_i and M'_{i+1} are adjacent,

$$\text{padO}(1^\lambda, M'_i[\text{pub}], 1^\ell) \approx_c \text{padO}(1^\lambda, M'_{i+1}[\text{pub}], 1^\ell).$$

Therefore, by a hybrid argument, we have the desired result. $\square$

7.2 Applications to High-Rate Obfuscation

Rate-1 Null IO. Recall the notion of null IO [WZ17, GKW17], 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 7.4 (Null IO for Programs with Split Description). We say that an IO scheme as in Definition 4.1 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 $|\text{pub}| \leq p(\lambda)$.
- For every input $x \in \{0, 1\}^{\leq n}$, it holds that $M_0[\text{pub}](x) = M_1[\text{pub}](x) = 0$ and $\text{RT}(M_0[\text{pub}], x) = \text{RT}(M_1[\text{pub}], x)$.

For technical reasons, our rate-1 null IO result will only apply to programs $M[\text{pub}]$ whose runtime $f(x) = \text{RT}(M[\text{pub}], x)$ has a small description. For simplicity, we will restrict our attention to polynomial TMs M for which there exists a polynomial q such that $\text{RT}(M[\text{pub}], x) = q(|x|)$. We will call such machines “runtime-regular”. We say a null IO scheme is secure for runtime-regular programs if Definition 7.4 satisfies security for pairs $M_0[\text{pub}]$ and $M_1[\text{pub}]$ which are runtime-regular.

Theorem 7.5 (Rate-1 Null IO). *Suppose one-way functions exist. Then, there exists null IO for runtime-regular programs with the following efficiency: in the correctness experiment of Definition 4.1,*

- *assuming fully succinct IO, $|\widetilde{M}| = |M| + |\text{desc}(p)| + \text{poly}(\lambda)$,*
- *assuming description-succinct IO, $|\widetilde{M}| = |M| + |\text{desc}(p)| + \text{poly}(n, \lambda)$*

where p is the polynomial which describes the runtime of $M[\text{pub}]$, and $\text{desc}(p)$ is the size of the program which describes p .

Proof. We consider the following Null IO algorithm which uses padO (Algorithm 5).

$\mathcal{O}(1^\lambda, 1^n, M[\text{pub}]):$

- Construct the following machine U which on input (pub, b, P, x) :
 - If $b = 0$, output $P[\text{pub}](x)$.
 - Else if $b = 1$, output 0.

Clearly, U has size $O(1)$.

- Let $U_{b,M}[\text{pub}] = U(\text{pub}, b, M, \cdot)$ be the TM U with $b = 0$, $P = M$ hardcoded and pub as public

input. Clearly, $U_{b,M}$ has a description of size $|M| + O(1)$.

- Output $\text{pad}\mathcal{O}(1^\lambda, U_{b,M}[\text{pub}], 1^\gamma)$, where $\gamma = |U_{b,M}|$.

Let $\ell = |M|$, and let the polynomial p describe the runtime of $M[\text{pub}]$. Let Z be the program of size ℓ which on input x , performs $p(|x|)$ dummy steps and then outputs 0. It suffices to show that a program $U_{0,M}$ is $(|M| + O(1))$ -close to the zero function $U_{0,Z}$, and the result then follows from Theorem 7.3. Indeed, consider the following sequence:

1. Start with $U_{0,M}$.
2. Switch the bit b to 1 to get $U_{1,M}$.
3. Switch the program P from M to Z one bit at a time, therefore obtaining $U_{1,Z}$.
4. Switch the bit b to 0 to get $U_{0,Z}$.

The above sequence demonstrates that $U_{0,M}$ and $U_{0,Z}$ are indeed $(|M| + O(1))$ -close. $\square$

Batch-obfuscation. We now consider an IO scheme that takes a batch of $k \leq \text{poly}(\lambda)$ programs $M_1, \dots, M_k$, 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

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

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

Theorem 7.6. *Suppose one-way functions exist. Then, there exists a batch-IO that obfuscates $k \leq \text{poly}(\lambda)$ programs $\Gamma \leftarrow \mathcal{O}(1^\lambda, M_1, \dots, M_k)$ with the following efficiency:*

- Assuming fully succinct IO,

$$|\Gamma| = \sum_{i \in [k]} |M_i| + \max_{i \in [k]} |M_i| + \text{poly}(\lambda, \log k).$$

In particular, assuming that $|M_1| = |M_2| = \dots = |M_k|$, we have $|\Gamma| = (k + 1)|M_i| + \text{poly}(\lambda, \log k)$, giving rate $\frac{k}{k+1}$.

- Assuming description-succinct IO,

$$|\Gamma| = \sum_{i \in [k]} |M_i| + \max_{i \in [k]} |M_i| + \text{poly}(\lambda, n, \log k)$$

where n is an upper bound on the input length supported by Γ .

Proof. We first construct a batch-obfuscation scheme using $\text{pad}\mathcal{O}$ (Algorithm 5).

$\mathcal{O}(1^\lambda, M_1, \dots, M_k)$:

- Construct the machine U which takes as input
 - a bit $b \in \{0, 1\}$,
 - an index $\alpha \in [k] \cup \{\perp\}$,
 - $k + 1$ programs $P_1, \dots, P_{k+1}$,
 - input index $j \in [k]$ and instance x

and does the following:

- If $b = 0$, output $P_j(x)$.
- If $b = 1$,
 - * if $j = \alpha$, output $P_{k+1}(x)$.
 - * else, output $P_j(x)$.
- Let $\widehat{U}$ be the program $U_{M_1, \dots, M_k}$ with the following variables hardcoded:
 - Set $b = 0, \alpha = \perp$.
 - For $j \in [k]$, set $P_j = M_j$.
 - Set $P_{k+1} = \perp$ (which is $O(1)$ bits).
 - Leave j and x as free variables.
- Let $\ell = \max_i |M_i|$, and let $\gamma = |\widehat{U}| + \ell$.
- Output $\text{padO}(1^\lambda, \widehat{U}[\emptyset], 1^\gamma)$ (i.e. this gives a URS of size ℓ).

Efficiency follows directly from Theorem 7.3. To show security from fully succinct IO/description-succinct IO, it suffices to show that $U_{M_1, \dots, M_k}$ is γ -close to $U_{M'_1, \dots, M'_k}$. For $j = 0, \dots, k-1$, consider the following sequence of programs:

- $H_{j,1}$ Program is $U_{M'_1, \dots, M'_j, M_{j+1}, \dots, M_k}$ (where $b = 0, \alpha = \perp, P_{k+1} = \perp$).
- $H_{j,2}$ Switch α one bit at a time to set $\alpha = j+1$.
- $H_{j,3}$ Hardcode P_{k+1} one bit at a time to be M'_{j+1} . It is clear that this does not influence the functionality of the program. (Note that the description of this new program is at most γ by construction, since we are hardcoding at most ℓ additional bits.)
- $H_{j,4}$ Switch $b = 0$ to $b = 1$. This follows from the fact that $\forall x, M_{j+1}(x) = M'_{j+1}(x)$.
- $H_{j,5}$ Switch P_{j+1} one bit at a time to M'_{j+1} . It is clear that this does not influence the functionality of the program.
- $H_{j,6}$ Switch $b = 0$. This follows from the fact that $P_{j+1} \equiv P_{k+1}$.
- $H_{j,7}$ Switch α to $\perp$ and free the variable P_{k+1} , one bit at a time. At this point, the program is equivalent to $U_{M'_1, \dots, M'_j, M'_{j+1}, M_{j+2}, \dots, M_k}$.

Therefore, it is clear that the programs are γ -close and we can indeed invoke security of Theorem 7.3. $\square$

7.3 Improving EF-IO Rate

In this section, we show how to use Theorem 7.3 to improve the rate of input-succinct EF-IO.

Theorem 7.7 (Input-succinct pv-IO with rate-1 in uniform EF Proof). *Assuming description-succinct IO and sub-exponential LWE, there exists an input-succinct (ρ, T) -EF-secure IO scheme (as in Theorem 3.9) where $\text{iO}_{\rho, T}(1^\lambda, M)$ has size $\rho + 2|M| + \text{poly}(\lambda)$.*

Proof (sketch). Recall that in the construction of JJ22 (Theorem 3.9, [JJ22, Figure 11]), the program being obfuscated is padded to size ρ in order to accommodate the program $\mathcal{M}_{\text{EF}}$ which outputs the uniform EF proof of equivalence, as well as both M_0 and M_1 . Moreover, other than the hybrids which introduce/delete $M_0, M_1, \mathcal{M}_{\text{EF}}$, all other hybrids are of the following form:

- the hybrid performs a cryptographic switch (not relying on iO security), or
- changes the program in at most $\text{poly}(\lambda)$ positions (e.g. puncturing PRF keys, hardcoding indices, etc) and uses iO security (i.e. invokes iO security for programs which are $\rho + |M_0| + |M_1| + \text{poly}(\lambda)$ -adjacent).

Therefore, using description-succinct $\text{pad}\mathcal{O}$ with padding parameter $\rho + |M_0| + |M_1| + \text{poly}(\lambda)$ in place of iO for circuits in the construction of JJ22, we obtain a construction with a public random string of size $|\mathcal{M}_{\Pi}| + |M_b|$, and an obfuscation of size $|M_b| + \text{poly}(\lambda)$. $\square$

Acknowledgements

We thank Jiatu Li for many helpful comments, and answering many questions about PV. Codex and ChatGPT were used to help polish the exposition.

References

- [AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. 69
- [ABI⁺23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1553–1566. ACM Press, June 2023. 2, 6, 9, 17, 68
- [AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Berlin, Heidelberg, August 2015. 99
- [AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for Turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 252–279. Springer, Cham, August 2017. 7, 8, 23
- [AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Cham, November 2018. 2
- [AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada. Evasive LWE: Attacks, variants & obfustopia. Cryptology ePrint Archive, Paper 2025/375, 2025. 5

- [BCC⁺17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017. 4
- [BCG⁺19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019. 9
- [BDJ⁺25] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 663–698. Springer, Cham, August 2025. , 2, 5, 99, 100
- [Ben89] Charles H Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. 15, 54
- [BFK⁺19] Saikrishna Badrinarayanan, Rex Fernando, Venkata Koppula, Amit Sahai, and Brent Waters. Output compression, MPC, and iO for Turing machines. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 342–370. Springer, Cham, December 2019. 5
- [BGI⁺01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001. 1
- [BGL⁺15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015. 1
- [Bla79] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979. 5
- [BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Cham, April / May 2018. 100
- [Bus86] Samuel Buss. *Bounded Arithmetic*. Bibliopolis, Naples, Italy, 1986. 93, 94, 97
- [BÜW24] Chris Brzuska, Akin Ünäl, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 418–449. Springer, Singapore, December 2024. 5
- [BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015. 7, 99

- [CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 429–437. ACM Press, June 2015. [1](#)
- [CL12] Stephen A Cook and Dai Tri Man Le. Formalizing randomized matching algorithms. *Logical Methods in Computer Science*, 8, 2012. [76](#), [77](#)
- [Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua 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. [93](#)
- [Coo75] Stephen 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. [1](#), [3](#), [25](#), [26](#), [48](#), [92](#), [93](#), [94](#)
- [CPW20] Suviradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. Witness maps and applications. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 220–246. Springer, Cham, May 2020. [4](#), [15](#)
- [CPW23] Suviradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. A map of witness maps: New definitions and connections. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 635–662. Springer, Cham, May 2023. [67](#), [100](#), [101](#)
- [CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979. [3](#), [26](#)
- [CS76] Stephen A. Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13(1):25–37, 1976. [49](#)
- [DGI⁺19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Cham, August 2019. [34](#), [95](#), [96](#)
- [DGM23] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for Turing machines. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 606–634. Springer, Cham, May 2023. [2](#)
- [DJM⁺25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 73–92. Springer, Cham, August 2025. [5](#)

- [DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. To appear at Asiacrypt, 2025. [9](#)
- [Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. [76](#), [77](#)
- [FJM⁺26] Cody Freitag, Abhishek Jain, Surya Mathialagan, Omer Paneth, and Daniel Wichs. Personal communication, 2026. [4](#), [17](#), [64](#)
- [GGH⁺13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. [1](#)
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984. [30](#)
- [GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. [1](#), [4](#), [5](#), [58](#)
- [GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, October 2017. [8](#), [83](#)
- [GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 156–181. Springer, Cham, April / May 2017. [54](#)
- [GS18a] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for Turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 425–454. Springer, Cham, November 2018. [2](#), [54](#)
- [GS18b] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Cham, April / May 2018. [54](#)
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013. [98](#)
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. [5](#)
- [Hal35] P. Hall. On representatives of subsets. *J. London Math. Soc.* 10, 1935. [21](#)

- [HIJ⁺16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Madhu Sudan, editor, *ITCS 2016*, pages 157–168. ACM, January 2016. 9
- [HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia 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. 5
- [Hol15] Justin Holmgren. On necessary padding with IO. *Cryptology ePrint Archive*, Report 2015/627, 2015. 3, 7
- [HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. 12, 32
- [ISN87] M. Ito, A. Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. IEEE Global Telecommunication Conf. (Globecom’87)*, pages 99–102, 1987. 5
- [JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd FOCS*, pages 1023–1034. IEEE Computer Society Press, October / November 2022. , 1, 3, 4, 9, 24, 28, 29, 30, 32, 86, 92, 94, 98, 99
- [JKLM25] Zhengzhong Jin, Yael Tauman Kalai, Alex Lombardi, and Surya Mathialagan. Universal SNARGs for NP from proofs of correctness. In Michal Koucký and Nikhil Bansal, editors, *57th ACM STOC*, pages 933–943. ACM Press, June 2025. 3, 4, 9, 37, 98, 99, 100
- [JKLV24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs under LWE via propositional proofs. In Bojan Mohar, Igor Shinkar, and Ryan O’Donnell, editors, *56th ACM STOC*, pages 1750–1757. ACM Press, June 2024. 3
- [JLL23] Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 479–510. Springer, Cham, April 2023. 7
- [JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021. 1
- [JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0 . In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. 1
- [KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023. , 34, 37, 40, 95, 97, 98

- [KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015. [1](#), [9](#)
- [KNY14] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 254–273. Springer, Berlin, Heidelberg, December 2014. [6](#), [9](#), [17](#), [68](#)
- [Li25] Jiatu Li. An introduction to feasible mathematics and bounded arithmetic for computer scientists. *Electronic Colloquium on Computational Complexity (ECCC)*, 2025. [26](#), [39](#), [92](#), [93](#)
- [LMOP24] Zhenjian Lu, Noam Mazon, Igor C. Oliveira, and Rafael Pass. Lower bounds on the overhead of indistinguishability obfuscation. Cryptology ePrint Archive, Report 2024/1524, 2024. [2](#), [7](#), [8](#)
- [LNW25] George Lu, Shafik Nassar, and Brent Waters. Succinct computational secret sharing for monotone circuits. Cryptology ePrint Archive, Paper 2025/850, 2025. [9](#)
- [LPST16] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 447–462. Springer, Berlin, Heidelberg, March 2016. [99](#)
- [LT80] Thomas Lengauer and Robert E. Tarjan. The space complexity of pebble games on trees. *Information Processing Letters*, 10(4–5):184–188, 1980. [49](#)
- [LZ17] Qipeng Liu and Mark Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 138–169. Springer, Cham, November 2017. [1](#)
- [MDS25] Yaohua Ma, Chenxin Dai, and Elaine Shi. Quasi-linear indistinguishability obfuscation via mathematical proofs of equivalence and applications. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 157–186. Springer, Cham, May 2025. [3](#), [9](#), [26](#), [28](#), [29](#)
- [Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. [4](#)
- [Nor13] Jakob Nordström. Pebble games, proof complexity, and time-space trade-offs. *Logical Methods in Computer Science*, 9(3):1–63, 2013. [49](#)
- [OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Berlin, Heidelberg, November / December 2015. [34](#), [37](#), [40](#), [95](#)

- [QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018. 100
- [RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 3–36. Springer, Cham, December 2024. 1
- [RW90] Ran Raz and Avi Wigderson. Monotone circuits for matching require linear depth. In *22nd ACM STOC*, pages 287–292. ACM Press, May 1990. 6
- [SC04] Michael Soltys and Stephen Cook. The proof complexity of linear algebra. *Annals of Pure and Applied Logic*, 130(1):277–323, 2004. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS). 97
- [Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. 5
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. 3, 4, 16
- [VNS⁺03] V. Vinod, Arvind Narayanan, K. Srinathan, C. Pandu Rangan, and Kwangjo Kim. On the power of computational secret sharing. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 162–176. Springer, Berlin, Heidelberg, December 2003. 6, 9
- [VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Cham, December 2022. 5
- [WW24] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In Bojan Mohar, Igor Shinkar, and Ryan O’Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024. 5
- [WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th FOCS*, pages 600–611. IEEE Computer Society Press, October 2017. 8, 83
- [Yao89] Andrew Chi-Chih Yao. Unpublished manuscript. Presented at Oberwolfach and DIMACS workshops, 1989. 6, 9

A Cook’s Theory PV

This section is taken largely from [JJ22, Li25].

Cook introduced a theory PV [Coo75] 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 [Cob65]). Hence, any polynomial-time function is definable in PV [Coo75]. 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 [Bus86].

We recall the formalization of PV following [Li25], based on Cook's formulation of PV from [Coo75]. In PV, both function symbols and proofs are generated inductively in tandem, and each is assigned a nonnegative integer called its *order*. We begin with the base level, which specifies the objects of order 0. The language includes a constant symbol ε , and function symbols $s_0(x)$, $s_1(x)$, $\circ(x, y)$, $\text{TR}(x)$, and $\text{ITR}(x, y)$ of order 0. Here and throughout, variables such as x, y range over the domain of PV, namely the natural numbers represented as binary strings in $\{0, 1\}^*$. Terms of order i are built by applying order- i function symbols to variables and previously constructed terms, together with the constant ε , and equations of order i are expressions of the form $s = t$ where s, t are terms of order i . The behavior of the order-0 function symbols is specified by the following equations:

$$x \circ \varepsilon = x, \quad x \circ s_i(y) = s_i(x \circ y) \quad i \in \{0, 1\}, \quad (\text{A.1})$$

$$\text{TR}(\varepsilon) = \varepsilon, \quad \text{TR}(s_i(x)) = x \quad i \in \{0, 1\}, \quad (\text{A.2})$$

$$\text{ITR}(x, \varepsilon) = x, \quad \text{ITR}(x, s_i(y)) = \text{TR}(\text{ITR}(x, y)) \quad i \in \{0, 1\}. \quad (\text{A.3})$$

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 [Cob65]). Specifically, for existing function symbols g, h_1, h_2, k_1, k_2 in PV, define the following equations as *defining equations*

$$f(0, \mathbf{y}) = g(\mathbf{y}), \quad (6)$$

$$f(x||i, \mathbf{y}) = h_i(x, \mathbf{y}, f(x, \mathbf{y})), i = 0, 1, \quad (7)$$

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 undecidability issue, PV further requires that the output length of f is bounded by a polynomial. To ensure this, Cook requires that " $|h_i(x, \mathbf{y}, z)| \leq |z| + |k_i(x, \mathbf{y})|$ " is provable in PV, where $|\cdot|$ is the length of the dyadic presentation.

The *inference rules* are 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 $k \geq 0$ variables. f is any function symbol (we will define later).

- $R_1: t = u \vdash u = t$.
- $R_2: t = u, u = v \vdash t = v$
- $R_3: t_1 = u_1, t_2 = u_2, \dots, t_k = u_k \vdash f(t_1, t_2, \dots, t_k) = f(u_1, u_2, \dots, u_k)$.

- 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 equations (7) for f_1, f_2 , with the same function symbols g, h_1, h_2 .

PV proofs for arithmetic. The work of [Bus86] shows that basic arithmetic operations such as $+, -, \cdot, /, \lfloor \cdot \rfloor, \leq$ can be introduced in Cook’s theory PV [Coo75] as function symbols, and their basic laws such as commutativity, associativity, distributivity, etc. can be proven in PV.

A.1 Theory PV_1

The following is taken nearly verbatim from [JJ22].

In the same work [Coo75], Cook also introduced a theory PV_1 in which formalizing proofs is easier than PV. [Coo75] 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”, “implication”, and “equivalence”.

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

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

Moreover, $\text{LESS}(h_{ij}(x, y, z, u), u \star k_{ij}(x, y, z)) = 0, i, j \in \{0, 1\}$ needs to be provable in PV. Finally, the definitions of the initial functions TR, $\star$, $\oplus$, 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.** $A_1, A_2, \dots, A_k \vdash B$, where the formula B is a truth-functional implication of formulae $A_1, \dots, A_k$.
- **k -Induction.** $\{A(0/x_i)\}_{i \in [k]}, \{A \rightarrow A(x_1 || j_1/x_1, \dots, x_k || j_k/x_k)\}_{j_1, j_2, \dots, j_k \in \{0,1\}} \vdash A$, where A is a formula of the variables $x_1, \dots, x_k$.

B PV proofs of relevant constructions

In this section, we give sketches to argue that the relevant constructions that we use in our work have PV proofs.

B.1 Statistical Binding Property of SSB Hash Construction of [OPWW15], [KLVW23]

In this section, we show that the SSB of [OPWW15], [KLVW23] instantiated via the rate-1 oblivious transfer of [DGI⁺19] has a PV proof of the binding property.

Lemma B.1. *The rate-1 string OT construction of [DGI⁺19] from the LWE assumption with super-polynomial modulus (with light modifications) has a PV proof of completeness with verifiable correctness, i.e. there exists an algorithm Valid such that the following holds for every sender random tape r_S :*

$$\vdash_{PV} \left(\begin{array}{l} \text{crs} \leftarrow \text{OT.Gen}(1^\lambda, 1^\ell) \\ \wedge m_R \leftarrow \text{OT.Com}(\text{crs}, b; r) \\ \wedge m_S \leftarrow \text{OT.Send}(\text{crs}, m_R, x_0, x_1; r_S) \\ \wedge m_S \neq \perp \\ \wedge |x_0| \leq \ell, |x_1| \leq \ell \\ \wedge x \leftarrow \text{OT.Receive}(\text{crs}, r, m_S) \end{array} \right) \rightarrow x = x_b$$

Proof. We give the construction of [DGI⁺19] in Algorithm 6. We write $r_S \in \{0, 1\}^\lambda$ for the sender's random tape, which is sampled uniformly in a randomized execution of the OT protocol. We argue the correctness of the algorithm (in the case where the hash algorithm does not abort). Take q to be even, and interpret the arithmetic below modulo q . Write $\text{dist}_q(z, a) = \min_{m \in \mathbb{Z}} |z - a + mq|$ for circular distance modulo q , and let $E = B(2\ell + \lambda)$. The rounding function $\lfloor z \rfloor_2$ changes value at $q/4$ and $3q/4$ modulo q .

We bound the abort probability as in [DGI⁺19, Theorem 4.7]. Replacing the PRF outputs with uniform elements of $\mathbb{Z}_q$ incurs a loss of at most $\ell \varepsilon_{\text{PRF}}$, where ε_{PRF} is the PRF distinguishing advantage. Each z_t is then uniform, so its boundary check fails with probability at most $(4E + 2)/q$. By a union bound, we have that

$$\varepsilon_{\text{OT}}(\lambda, \ell) \leq \frac{\ell(4B(2\ell + \lambda) + 2)}{q} + \ell \varepsilon_{\text{PRF}}.$$

This bound holds for every fixed pair of sender inputs and either receiver choice. We choose $q/B = \Gamma(\lambda)^{\omega(1)}$, with $k, \log q = \text{poly}(\lambda)$, and assume LWE hardness at these parameters against $\text{poly}(\Gamma(\lambda))$ -size adversaries. We instantiate the internal PRFs with distinguishing advantage $\text{negl}(\Gamma(\lambda))$ against such adversaries. The above bound is then $\text{negl}(\Gamma(\lambda))$, uniformly for $\ell \leq \text{poly}(\Gamma(\lambda))$.

Algorithm 6 Rate-1 String OT construction from [DGI⁺19].

OT.Gen($1^\lambda, 1^\ell$): For $j = 1, \dots, 2\ell + \lambda$, choose a random $\mathbf{a}_{j,b} \leftarrow \mathbb{Z}_q^k$.

- Set

$$\mathbf{A} := \begin{pmatrix} \mathbf{a}_{1,0} & \mathbf{a}_{2,0} & \dots & \mathbf{a}_{2\ell+\lambda,0} \\ \mathbf{a}_{1,1} & \mathbf{a}_{2,1} & \dots & \mathbf{a}_{2\ell+\lambda,1} \end{pmatrix}$$

OT.Com(crs, β) :

- For $t \in [\ell]$,
 - $\mathbf{s}_t \leftarrow \mathbb{Z}_q^k$ and for $j = 1, \dots, 2\ell + \lambda$ and $b \in \{0, 1\}$, set

$$u_{j,b} = \mathbf{s}_t^T \mathbf{a}_{j,b} + \mathbf{1}_{j=\beta \cdot \ell + t} \cdot b \cdot \lfloor q/2 \rfloor + e_{j,b}$$

where $e_{j,b} \leftarrow [-B, B]$, for some parameter B which we will choose later.

- Set

$$\mathbf{B}_t := \begin{pmatrix} u_{1,0} & u_{2,0} & \dots & u_{2\ell+\lambda,0} \\ u_{1,1} & u_{2,1} & \dots & u_{2\ell+\lambda,1} \end{pmatrix}$$

- Sample a PRF key K_t .
- Set $\mathbf{ek}_t := (\mathbf{B}_t, K_t)$ and $\mathbf{td}_t := (\mathbf{s}_t, K_t)$.
- Set $m_R := (\{\mathbf{ek}_t\}_{t \in [\ell]})$ and $\mathbf{td} = \{\mathbf{td}_t\}_{t \in [\ell]}$.

OT.Send(crs, $m_R, x_0, x_1; r_S$) :

- Set $\mathbf{y} = x_0 \| x_1 \| r_S$, using the supplied tape $r_S \in \{0, 1\}^\lambda$.
- Compute $\mathbf{h} = \sum_{j=1}^{2\ell+\lambda} \mathbf{a}_{j,\mathbf{y}[j]}$.
- For each $t \in [\ell]$:
 - Compute $\alpha_t = \sum_{j=1}^{2\ell+\lambda} u_{j,\mathbf{y}[j]}$.
 - Compute $z_t \leftarrow \alpha_t + \text{PRF}_{K_t}(\mathbf{h})$. If $\text{dist}_q(z_t, q/4) \leq E$ or $\text{dist}_q(z_t, 3q/4) \leq E$, abort and output $\perp$.
 - Compute $\gamma_t = \lfloor z_t \rfloor_2$, where the notation $\lfloor z \rfloor_2 = \lfloor z \cdot 2/q \rfloor \pmod{2}$ denotes the rounding function.
- Output $m_S = (\mathbf{h}, \{\gamma_t\}_{t \in [\ell]})$.

OT.Receive(crs, $\mathbf{td}, m_S$) :

- For $t \in [\ell]$:
 - Compute $\gamma_0 = \lfloor \mathbf{s}_t^T \mathbf{h} + \text{PRF}_{K_t}(\mathbf{h}) \rfloor_2$ and $\gamma_1 = \lfloor \mathbf{s}_t^T \mathbf{h} + \text{PRF}_{K_t}(\mathbf{h}) + q/2 \rfloor_2$.
 - If $\gamma_t = \gamma_0$, set $y_t = 0$. Else, set $y_t = 1$.

Output $y_1, \dots, y_\ell$.

We now sketch the PV proof of correctness. We prove the correctness of the t th bit of the output of OT.Receive. This proof only uses basic properties of linear algebra which are provable in PV [Bus86, SC04].

$$\begin{aligned}
\alpha_t &= \sum_{j=1}^{2\ell+\lambda} u_{j,\mathbf{y}[j]} = \sum_{j=1}^{2\ell+\lambda} (\mathbf{s}_t^T \mathbf{a}_{j,\mathbf{y}[j]} + e_{j,\mathbf{y}[j]} + \mathbb{1}_{j=\beta\ell+t} \mathbf{y}[j] \cdot q/2) \\
&= \mathbf{s}_t^T \mathbf{h} + \sum_{j=1}^{2\ell+\lambda} (e_{j,\mathbf{y}[j]}) + \mathbf{y}[\beta\ell+t] \cdot q/2 \\
&= \mathbf{s}_t^T \mathbf{h} + x_\beta[t] \cdot q/2 + \sum_{j=1}^{2\ell+\lambda} (e_{j,\mathbf{y}[j]})
\end{aligned}$$

Let $e_t = \sum_{j=1}^{2\ell+\lambda} e_{j,\mathbf{y}[j]}$ and $a_t = \mathbf{s}_t^T \mathbf{h} + \text{PRF}_{K_t}(\mathbf{h})$. Since each error is B -bounded, $|e_t| \leq E$, and the above calculation gives

$$z_t = a_t + x_\beta[t] \cdot q/2 + e_t \pmod{q}.$$

If the check introduced in blue does not abort, moving from z_t to $z_t - e_t$ crosses neither rounding boundary, including across the modular wraparound. Hence

$$\gamma_t = \lfloor z_t \rfloor_2 = \lfloor a_t + x_\beta[t] \cdot q/2 \rfloor_2 = \lfloor a_t \rfloor_2 \oplus x_\beta[t].$$

Since the receiver computes $\gamma_0 = \lfloor a_t \rfloor_2$, its output bit is exactly $x_\beta[t]$. The strict distance from the boundaries also excludes rounding ties. These claims reduce to integer inequalities and case analysis on the two rounding intervals (multiplying by four to avoid fractional endpoints), and can therefore be formalized in PV. This proves correctness whenever the sender does not abort. $\square$

Lemma B.2. *Suppose that there exists a rate-1 string OT construction that has a proof of completeness for every sender random tape r_S , i.e.*

$$\vdash_{\text{PV}} \left(\begin{array}{l} \text{crs} \leftarrow \text{OT.Gen}(1^\lambda, 1^\ell) \\ \wedge m_R \leftarrow \text{OT.Com}(\text{crs}, b; r) \\ \wedge m_S \leftarrow \text{OT.Send}(\text{crs}, m_R, x_0, x_1; r_S) \\ \wedge m_S \neq \perp \\ \wedge |x_0| \leq \ell, |x_1| \leq \ell \\ \wedge x \leftarrow \text{OT.Receive}(\text{crs}, r, m_S) \end{array} \right) \rightarrow x = x_b$$

Then, the resulting two-to-one somewhere statistically binding hash via the transformation in [KLVW23], using a short-seeded PRF to generate the sender's randomness as described below, has a proof of statistical binding.

$$\vdash_{\text{PV}} \left(\begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Two-SSB.TGen}(1^\lambda, 1^\ell, b) \\ \wedge h \leftarrow \text{Two-SSB.Hash}(\text{hk}, (x_0, x_1)) \\ \wedge h \neq \perp \\ \wedge y = \text{Two-SSB.Ext}(\text{hk}, \text{td}, h) \end{array} \right) \rightarrow (y = x_b).$$

Proof sketch. Recall the transformation of [KLVW23]. Let $d = d(\lambda, \ell)$ be the sender random-tape length of the rate-1 string OT scheme ($d = \lambda$ for Algorithm 6). Let $F = \{F_\sigma\}$ be a PRF with a seed of length $\text{poly}(\lambda)$, whose inputs are unambiguous encodings $\langle x_0, x_1 \rangle$ of sender-input pairs and whose outputs have length d . Following [KLVW23, Lemma 4.5], we include an independent public seed in the hash key and construct the two-to-one SSB scheme as follows:

- $\text{Two-SSB.TGen}(1^\lambda, 1^\ell, b)$: Compute $\text{crs} \leftarrow \text{OT.Gen}(1^\lambda, 1^\ell)$ and $m_R \leftarrow \text{OT.Com}(\text{crs}, b; r)$. Independently sample a PRF seed σ . Set $\text{hk} = (\text{crs}, m_R, \sigma)$ and $\text{td} = (b, r)$. Output (hk, td) .
- $\text{Two-SSB.Gen}(1^\lambda, 1^\ell)$: Run $(\text{hk}, \text{td}) \leftarrow \text{Two-SSB.TGen}(1^\lambda, 1^\ell, 0)$ and output hk .
- $\text{Two-SSB.Hash}(\text{hk}, (x_0, x_1))$: Parse $\text{hk} = (\text{crs}, m_R, \sigma)$ and compute

$$r_S = F_\sigma(\langle x_0, x_1 \rangle), \quad h \leftarrow \text{OT.Send}(\text{crs}, m_R, x_0, x_1; r_S).$$

Output h , including $\perp$ if the sender aborts.

- $\text{Two-SSB.Ext}(\text{hk} = (\text{crs}, m_R, \sigma), \text{td} = (b, r), h)$: Output $\text{OT.Receive}(\text{crs}, r, h)$.

Determinism, succinctness, and receiver-choice hiding follow as in [KLW23, Lemma 4.5].

For the PV proof of binding, substitute the polynomial-time term $F_\sigma(\langle x_0, x_1 \rangle)$ for r_S in the OT correctness statement. Whenever $h \neq \perp$, this gives $\text{Two-SSB.Ext}(\text{hk}, \text{td}, h) = \text{OT.Receive}(\text{crs}, r, h) = x_b$, as desired. $\square$

Correctness with a public PRF seed. The derandomization step of [KLW23, Lemma 4.5] incurs only the PRF distinguishing error. Thus, for every fixed sender-input pair (x_0, x_1) and either receiver-choice mode,

$$\Pr[\text{Two-SSB.Hash}(\text{hk}, (x_0, x_1)) = \perp] \leq \varepsilon_{\text{OT}}(\lambda, \ell) + \varepsilon_F(\lambda, \ell),$$

where ε_{OT} uniformly bounds the OT abort probability with uniform sender coins and ε_F is the PRF distinguishing error. The probability is over key generation, with the input pair fixed beforehand.

Taking F to be Γ -secure with $\text{negl}(\Gamma(\lambda))$ advantage bounds ε_F as required. Combining this with the bound on ε_{OT} above gives the required correctness error.

The tree composition follows [KLW23, Lemma 4.5], with independent keys and PRF seeds at each level. Under our abort convention, a union bound over the fewer than $2k$ internal nodes accounts for the total error for a message with k blocks.

B.2 Correctness of Obfuscation Construction of [JJ22]

In this section, we give a sketch of a PV proof of correctness for the JJ22 pviO construction for Turing machines. We first recall the main ingredients used in the JJ22 construction (as listed in Section 7.4 of [JJ22]), and argue how to instantiate these with PV proofs of correctness:

(i) Leveled fully homomorphic encryption.

The work of [JKLM25] showed that Gentry-Sahai-Waters [GSW13] (with a minor modification where the LWE errors are sampled from truncated Gaussians) can be used to implement a leveled FHE scheme with PV-correctness.

(ii) Sub-exponential IO scheme for circuits.

The work of [JKLM25] shows that assuming sub-exponential IO and sub-exponential LWE, one can construct an IO scheme with PV-correctness.

(iii) Somewhere extractable batch-arguments.

The somewhere extractable batch argument (se-BARG) is constructed generically using probabilistically checkable proofs, somewhere extractable hash functions, and correlation intractable hash functions. The correctness of those

building blocks can all be formalized in PV. The construction of seBARG for k statements is recursive, and the recursion depth is $\log_2 k$. Hence, the correctness proof of seBARG needs to argue inductively over the layers of the recursion. This induction can be formalized in PV since PV allows a polynomial induction rule.

(iv) **Somewhere extractable hash with local openings.**

The work of [JKLM25] shows that the local opening property of Merkle trees instantiated with *any hash function* has a proof of completeness. Therefore, instantiating a somewhere extractable hash via a Merkle tree using a Two-SSB as in Appendix B.1 preserves this property.

(v) **Puncturable PRFs.**

We will not need any proofs of correctness involving the puncturing property in the proof of completeness. Therefore, there is no explicit correctness property needed for the proof.

At a high level, the construction of JJ22 follows a gate-by-gate obfuscation template (similar to Yao’s garbled circuit). Intuitively, each gate g is obfuscated to obtain $\mathcal{O}(g)$, which takes as input authenticated ciphertexts, along with hashes, local openings and BARG proofs, verifies these hashes and proofs, and outputs an authenticated ciphertext. Therefore, assuming the PV-correctness of the underlying primitives, one can see that $\mathcal{O}(g)$ (see Figure 7 of [JJ22]) outputs an authenticated encryption of the “true” output of g if the inputs to $\mathcal{O}(g)$ are “true” authenticated encryptions of the input wires. At the end, the decryption key for the output wire is given, and hence one can retrieve the output. We can induct over the topological ordering of the circuit to prove that the final output is correct.

B.3 Correctness of Succinct Witness Encryption [BDJ⁺25]

In this section, we give an informal sketch of the proof of correctness for the fully succinct witness encryption construction of [BDJ⁺25, Section 7.3]. For full details of the construction, we refer the reader to [BDJ⁺25].

Given a pseudorandom obfuscation algorithm PRO (more specifically doubly-PRO) that is correct, it is easy to see that the construction of succinct witness encryption in [BDJ⁺25, Section 7.3] has a straightforward proof of completeness. Therefore, it suffices to argue that the PRO construction of [BDJ⁺25] has a proof of completeness. Recall that [BDJ⁺25] follow the following template in their construction:

1. **Construct xPRO:** Consider a circuit C with truth-table of size 2^n . The xPRO scheme runs in time $\text{poly}[2^n]$, and outputs an obfuscation Π of size $|\Pi| \leq 2^{n(1-\epsilon)}$ for $\epsilon > 0$. Moreover, the depth of this algorithm is $\text{poly}[|C|, \lambda]$.
2. **Delegate with output-compressing Laconic Function Evaluation:** Now, we want to reduce the obfuscator runtime from 2^n to slightly sublinear by delegating the bulk of the computation to the evaluator.
3. **Bootstrapping:** To obtain a full-fledged (d)PRO scheme, they apply a bootstrapping theorem similar to XiO bootstrapping [AJ15, BV15, LPST16].

Since this template is very close to the template of iO constructions from XiO, we will draw inspiration from the recent work of [JKLM25] that shows how to take a sub-exponential IO scheme and convert it into one that has a PV-proof of correctness (by additionally assuming sub-exponential LWE).

- In the xPRO step, since the algorithm is allowed to run in time 2^n and depth $\text{poly}[\lambda, |C|]$, one can first compute an obfuscation Π , and then brute-force check on every input that $\Pi(x) = C(x)$. If any of the checks fail, simply output C . This additional check does not ruin the efficiency of xPRO. This clearly has a PV-proof of correctness (a similar technique was used in [JKLM25]).
- In the delegation step, there are two main tools:
 - Blind garbled circuits [BLSV18]: This is a special variant of Yao’s garbled circuits. Therefore, following a similar proof as was done in [JKLM25] for garbled circuits, one can prove the correctness of blind garbled circuits in PV.
 - Laconic function evaluation [QWW18]: While this does not have perfect completeness, [BDJ⁺25, Section 5.4] gives a way to “verify correctness” of the protocol¹¹. Since they show the verifiable correctness via simple linear algebra, we can show the correctness of the protocol in PV.
- The bootstrapping step is identical to that of iO, so we can obtain a PV proof of this step in an identical manner as was done in [JKLM25].

C Deferred Proof from Section 5.2.4

In this section, we complete the proof of Section 5.2.4. This transformation is from the work of [CPW23], which shows how to construct a witness encryption scheme given a unique witness map (an object they defined). Since unique-SNARGs give us such a unique witness map, their framework applies to us.

Here, we recap the construction so that we can argue that the construction maintains succinctness, and that the construction has a PV proof.

We refer the reader to [CPW23] for the security analysis.

PV proof of correctness. Suppose the underlying SNARG has a PV proof of uniqueness, i.e. the proof that

$$\vdash_{\text{PV}} (R(x, w_0) = 1 \wedge R(x, w_1) = 1) \rightarrow (\text{SNARG.Prove}(\text{crs}, x, w_0) = \text{SNARG.Prove}(\text{crs}, x, w_1))$$

Then, we can argue that the witness encryption is correct as follows:

- (i) $(\text{crs}, s, r, \beta) \leftarrow \text{WE.Enc}(1^\lambda, \mathcal{R}, x, m \in \{0, 1\}; t, r, r_{\text{SNARG}})$.
- (ii) $\text{crs} = \text{SNARG.Gen}(1^\lambda, \mathcal{R}_G; r_{\text{SNARG}})$.
- (iii) $\beta \leftarrow \langle r, \pi \rangle \oplus m$ where $\pi \leftarrow \text{SNARG.P}(\text{crs}, (x, s), t)$.
- (iv) $s = G(t) \rightarrow \mathcal{R}_G((x, s), t) = 1$.
- (v) By uniqueness of the SNARG, we have $\forall w$,

$$\mathcal{R}_G((x, s), w) = 1 \rightarrow \text{SNARG.P}(\text{crs}, (x, s), w) = \pi$$

- (vi) In particular, the proof generated during encryption is the unique proof for (x, s) .

¹¹This is similar to how we argued the verifiable correctness of the rate-1 OT scheme in Appendix B.1.

Algorithm 7 A witness encryption scheme from a unique-SNARG. This transformation follows that of [CPW23] for unique witness maps to witness encryption.

WE.Enc($1^\lambda, \mathcal{R}, x, m \in \{0, 1\}; t, r, r_{\text{SNARG}}$) :

- Let $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$ be a length-doubling PRG.
- Define a new relation $\mathcal{R}_G$ as follows:

$$\mathcal{R}_G((x, s), w) = \begin{cases} 1 & \text{if } \mathcal{R}(x, w) = 1 \text{ or } s = G(w). \\ 0 & \text{otherwise.} \end{cases}$$

- Sample $\text{crs} \leftarrow \text{SNARG.Gen}(1^\lambda, \mathcal{R}_G; r_{\text{SNARG}})$.
- Sample $t \leftarrow \{0, 1\}^\lambda$, and let $s = G(t)$.
- Compute $\pi \leftarrow \text{SNARG.P}(\text{crs}, (x, s), t)$.
- Sample $r \leftarrow \{0, 1\}^{|\pi|}$, and let $b \leftarrow \langle r, \pi \rangle$ (i.e. this is the Goldreich-Levin hardcore bit).
- Output $(\text{crs}, s, r, b \oplus m)$.

WE.Dec($\text{ct} = (\text{crs}, s, r, b), x, w$):

- If $\mathcal{R}(x, w) = 0$, output $\perp$.
 - Compute $\pi \leftarrow \text{SNARG.P}(\text{crs}, (x, s), w)$.
 - Output $b \oplus \langle r, \pi \rangle$.
-

- (vii) $\mathcal{R}(x, w) = 1 \rightarrow \mathcal{R}_G((x, s), w) = 1$ by definition of $\mathcal{R}_G$.
- (viii) $\mathcal{R}(x, w) = 1 \rightarrow \text{SNARG.P}(\text{crs}, (x, s), w) = \pi$.
- (ix) $\mathcal{R}(x, w) = 1 \rightarrow \text{WE.Dec}(\text{ct}, x, w) = \beta \oplus \langle \pi, r \rangle$.
- (x) By construction, $\mathcal{R}(x, w) = 1 \rightarrow \text{WE.Dec}(\text{ct}, x, w) = m$.

Efficiency. It is easy to see that the size of the encryption is $|\text{crs}| + |\pi| + \text{poly}(\lambda)$. If the SNARG is witness-succinct, i.e. $|\text{crs}| = \text{poly}(\lambda, n_x)$ and $|\pi| = \text{poly}(\lambda, n_x)$, then the ciphertext is also witness-succinct, as desired. If the SNARG is fully succinct, then the ciphertext size is $\text{poly}(\lambda)$, as desired.