Indistinguishability Obfuscation via Mathematical Proofs of Equivalence

(*or: How to Overcome Non-Falsifiability*)

Abhishek Jain Zhengzhong Jin

Johns Hopkins University

Abstract

Over the last decade, indistinguishability obfuscation (iO) has emerged as a seemingly omnipotent primitive in cryptography. Moreover, recent breakthrough work has demonstrated that iO can be realized from well-founded assumptions. A thorn to all this remarkable progress is a limitation of all known constructions of general-purpose iO: the security reduction incurs a loss that is *exponential in the input length* of the function. This "input-length barrier" to iO stems from the non-falsifiability of the iO definition and is discussed in folklore as being possibly inherent. It has many negative consequences; notably, constructing iO for programs with inputs of *unbounded* length remains elusive due to this barrier.

We present a new framework aimed towards overcoming the input-length barrier. Our approach relies on *short mathematical proofs of functional equivalence* of circuits (and Turing machines) to avoid the brute-force "input-by-input" check employed in prior works.

- We show how to obfuscate circuits that have efficient proofs of equivalence in Propositional Logic with a security loss *independent* of input length.
- Next, we show how to obfuscate Turing machines with *unbounded* length inputs, whose functional equivalence can be proven in Cook's Theory PV.
- Finally, we demonstrate applications of our results to succinct non-interactive arguments and witness encryption, and provide guidance on using our techniques for building new applications.

To realize our approach, we depart from prior work and develop a new gate-by-gate obfuscation template that preserves the topology of the input circuit.

Contents

1 Introduction

Program obfuscation is the technique of converting a computer program into a new version that retains the functionality of the original but is immune to reverse-engineering. While a formal study of this notion was initiated at the turn of this century [Had00, BGI+01], the past decade has seen a renewed push towards its study. The notion of indistinguishability obfuscation (iO) [BGI+01] has emerged as the central figure, with a long sequence of works aimed towards investigating its existence (see e.g., [GGH+13, PST14, GLSW15, AJ15, BV15, Lin16, LV16, LPST16b, GMM+16, LPST16a, AS17, Lin17, LT17, Agr19, JLMS19, AJL+19, BDGM20, AP20, GJLS21, WW21, GP21]). This line of work recently led to the breakthrough result of [JLS21] who constructed iO for general functions from well-founded assumptions.

A parallel line of research over the last decade has demonstrated that most cryptographic primitives, including several powerful ones such as witness encryption [GGSW13], multiparty non-interactive key exchange [BZ14], succinct non-interactive arguments [SW21, BCG+18], software watermarking [CHN+16], and deniable encryption [SW21] can be built from iO. Moreover, iO has also found appeal outside cryptography, e.g. for establishing hardness of Nash equilibrium [BPR15] and the hardness of certain tasks in differential privacy [BZ14, BZ16]. These results have established iO as a "central hub" of theoretical cryptography.

Input-Length Barrier. A thorn to all this remarkable progress is a limitation of all known constructions of iO: the security reduction incurs a loss that is *exponential in the input length* of the function. This has severe negative consequences on the necessary assumptions and the efficiency of the scheme. In particular, it requires the program input length to be a priori bounded. This, in turn, prevents us from realizing iO for efficient computing models such as Turing machines with *unbounded* input length.[1]

This state of affairs motivates the following question:

Can we build iO with a loss in the security reduction independent of the input length?

To answer the above question, it is first important to understand whether the input-length barrier stems from technical limitations or something more fundamental. To develop intuition, it is useful to recall a folklore argument that explains the origin of the input-length barrier. Here, we sketch the informal idea[2] (adapted from [GGSW13, LZ17]) based on the meta-reduction technique [BV98].

Let us first recall the security definition of iO: if two programs P_1 and P_2 are *functionally equivalent* (i.e., for any input x, $P_1(x) = P_2(x)$), then their obfuscations must be indistinguishable to any polynomial-time algorithm. Now, suppose that there is a construction of iO whose security can be based on some polynomial-time hardness assumption (say) Y. That is, there is a polynomial-time reduction such that given black-box access to an adversary for the iO scheme, it can break the assumption Y. Consider the following "trivial" polynomial-time adversary that chooses two programs P_1, P_2 that are functionally equivalent except that their outputs differ at some input (say) x^*. Such an adversary can easily distinguish between obfuscations of P_1 and P_2 by evaluating them on x^*; yet the reduction must seemingly work for such an adversary as well. Then, combining the reduction with this trivial adversary, we have found a polynomial-time algorithm for Y, which is unlikely.

To prevent the above argument, it seems that the reduction must check whether the two programs P_1, P_2 are functionally equivalent so as to not be "fooled" by the trivial adversary. But how can the reduction check equivalence? One natural way is to iterate through all the inputs one by one. Indeed, this is the strategy implicit in the security proofs of all general-purpose constructions of iO. This strategy, however, leads to a security loss that is exponential in the input length.

[1]Some prior works overcome this barrier by relying on non-standard assumptions.

[2]We stress that this is *not* a formal proof. Turning this argument into a formal proof runs into subtle technical challenges.

Can we use an alternative strategy that does not incur such a loss? A sequence of prior works [GPS16, GS16, GPSZ17, LZ17] demonstrate that the exponential loss can be avoided in some cases when functional equivalence can be decided in *polynomial time* [LZ17]. This naturally limits their applicability (see Section 1.4 for discussion). Indeed, in general, functional equivalence may not be efficiently checkable. We ask whether it is possible to overcome the input-length barrier in such cases as well.

A Broader Perspective. The seeming necessity of checking functional equivalence and its consequences is in fact an example of a broader phenomenon in cryptography. The security definition of many cryptographic primitives is predicated on a mathematical premise that is not decidable in $\mathcal{NP}$. For example, the security of witness encryption [GGSW13] for a language L requires that a ciphertext encrypted using an instance $x \notin L$ must remain semantically secure. Similarly, the soundness definition of a proof system for a language L requires that any proof for an instance $x \notin L$ must be rejected by the verifier. In both of these cases, "$x \notin L$" is the mathematical premise, and deciding its truthfulness is a $co\mathcal{NP}$ problem that might require exponential time.

The difficulty of checking the mathematical premise can be leveraged to employ a similar meta-reduction technique as discussed above to establish barriers for other cryptographic primitives. This is reflected in the case of witness encryption, where all known constructions incur a security loss exponential in the witness length. In the regime of proof systems, Gentry and Wichs [GW11] leverage this observation to rule out adaptively-sound succinct non-interactive arguments [Mic00] based on falsifiable assumptions [Nao03]. Moreover, even known non-adaptively sound constructions (obtained by instantiating [SW21] with existing iO constructions) incur an exponential loss in the witness length.[3]

A New Approach. We present a new framework aimed towards overcoming the input-length barrier to iO. We then leverage the power of iO to overcome analogous barriers for other cryptographic primitives.

Our starting point is the following simple observation: suppose we are given a secure indistinguishability obfuscator. In order to leverage its security for a given pair of programs, we first write a *mathematical proof* to convince ourselves (and others) that the two programs are functionally equivalent. Importantly, this proof is *short* so that anyone can verify it. In particular, it is significantly shorter than the "brute-force" proof that involves iterating over every input. Our key insight is to rely on such (short) mathematical proofs of functional equivalence *for proving the security of the obfuscator.*

This raises the following question: *How can we use the mathematical proof of equivalence in proving security?* Our approach involves two principal steps:

- **Incremental Proofs of Equivalence:** We first rely on the following *local* property of mathematical proofs: recall that a mathematical proof consists of a series of true propositions, one followed by another. The truthfulness of each proposition is derived from only a constant number of previous propositions and an inference rule. We leverage this property to show that a short mathematical proof (of specific form) of "$C_1(x) = C_2(x)$" for two circuits C_1 and C_2 can be translated to a small number of *incremental changes* that transform the circuit C_1 into C_2. Crucially, each incremental change is of small size.

- **New Template for iO:** Next, we provide a new construction template for iO to leverage the above proofs of equivalence. Our template involves obfuscating an input circuit in a gate-by-gate manner to *preserve its topology* in the obfuscated circuit. This allows us to devise a security proof consisting of a polynomial number of steps, where in each step we only switch an obfuscated *subcircuit* corresponding to an incremental change. This results in a security loss exponential only in the size of the subcircuit but *independent* of the input length.

[3]We discuss more on this later in Section 1.2.

1.1 Our Results

We now proceed to describe our results.

I. iO for Circuits. We first consider the circuit model of computation. Our results rely on proofs in Propositional Logic [Bus98] — a branch of logic that deals with propositions and relations among them.

We define a notion of *propositional proof of equivalence* for circuits. Roughly speaking, we say that two circuit families $\{C_n^1\}_{n\in\mathbb{N}}$ and $\{C_n^2\}_{n\in\mathbb{N}}$ have a propositional proof of equivalence, if there exists a proof in propositional logic system to establish that C_n^1 and C_n^2 are functionally equivalent. Furthermore, we say that the proof is *efficient* if it is polynomial-sized.

Our first result is an obfuscation scheme for any two families of circuits with efficient propositional proofs of equivalence, with security loss independent of input length.

Theorem 1 (iO for Circuits from Propositional Proofs of Equivalence, Informal)**.** *There exist polynomials $p_1(\cdot), p_2(\cdot,\cdot,\cdot)$, such that assuming the hardness of the following, there exists a construction of* iO *for any two families of circuits $\{C_n^1\}_{n\in\mathbb{N}}, \{C_n^2\}_{n\in\mathbb{N}}$ with* efficient *propositional proofs of equivalence:*

- *Polynomial-hardness of Learning with Errors (LWE),*
- *$2^{p_1(\lambda)}$-secure one-way functions,*
- *$2^{p_1(\lambda)}$-secure indistinguishability obfuscation for circuits of size $p_2(\lambda, \log|C_n^1|, \log|C_n^2|)$,*

where λ is the security parameter of the iO *scheme.*

A few remarks are in order:

- Unlike prior works, we allow n, namely, the input length of circuits C_n^1, C_n^2 (and their sizes) to arbitrarily depend on λ, and not be bounded by p_1, p_2.
- The above theorem only requires an underlying indistinguishability obfuscator for *small* circuits of size essentially independent of C_n^1, C_n^2.

We obtain the above result in two steps: we first define a new notion of Δ-equivalent circuits and show how Δ-equivalent circuits can be constructed via Proofs in Propositional Logic [Bus98]. We then show how to construct iO for Δ-equivalent circuits, with security loss independent of input length.

Step 1: Δ-Equivalent Circuits. Informally, we say that two circuit families are Δ-*equivalent*, if there exist a polynomial number of intermediate circuits such that each two adjacent circuits only differ by a *logarithmic* number of gates, and the two subcircuits formed by these gates are functionally equivalent.

We demonstrate that efficient propositional proof of equivalence implies Δ-equivalence for circuits.

Lemma 1 (Δ-Equivalence from Propositional Logic Proofs)**.** *If there exist* polynomial-size *propositional proofs of equivalence for the circuit families $\{C_n^1\}_{n\in\mathbb{N}}$ and $\{C_n^2\}_{n\in\mathbb{N}}$, then $\{C_n^1\}_{n\in\mathbb{N}}$ and $\{C_n^2\}_{n\in\mathbb{N}}$ are Δ-equivalent.*

Given a pair of circuits (C_n^1, C_n^2) and a propositional proof of equivalence, we prove this lemma by embedding the propositional formulas (in the proof of equivalence) inside C_n^1 to gradually transform it into C_n^2, while preserving the functionality. We leverage the "local" property of the proof as well as the truthfulness of each formula to establish Δ-equivalence. See Section 2.1 for an overview of the proof.

Step II: iO for Δ-Equivalent Circuits. We next provide a construction of iO for Δ-equivalent circuits.

Lemma 2 (iO for Δ-Equivalent Circuits, Informal)**.** *There exist polynomials $p_1(\cdot), p_2(\cdot,\cdot,\cdot)$, such that assuming the same hardness assumptions as in Theorem 1, there exists a construction of* iO *for any two Δ-equivalent circuit families $\{C_n^1\}_{n\in\mathbb{N}}, \{C_n^2\}_{n\in\mathbb{N}}$.*

In order to prove the above lemma, we depart from prior templates for iO. To leverage Δ-equivalence, we develop a new (albeit, natural) *gate-by-gate* obfuscation template that preserves the topology of the input circuit. Due to such a design, a key challenge is to overcome various "mix-and-match" attacks, and we develop several techniques towards that end. A central component in our construction is a new notion of *somewhere extractable hash functions with consistency proofs*. We show how to build this object by combining somewhere extractable hash functions [HW15] with (publicly-verifiable) non-interactive batch arguments [CJJ21]. Both of these objects, in turn, can be based on the LWE assumption. We refer the reader to Section 2 for an overview of our technical approach.

II. iO for Turing Machines. We next tackle the challenging problem of constructing iO for Turing machines with unbounded length inputs. All prior results can either handle inputs of a priori bounded length [BGL+15, CHJV15, KLW15], or require very strong assumptions [BCP14, ABG+13, IPS15, LPST16b] (some of which are in fact known to be implausible in general [BCPR14, GGHW14, BSW16, LPST16b]).

We show how to obfuscate Turing machines with arbitrary length inputs based on similar assumptions as used for obfuscating circuits. Our approach is applicable to Turing machines whose functional equivalence can be proven in Cook's theory PV [Coo75]. Cook introduced the theory PV in 1975 to formalize the intuition of polynomial-time reasoning. PV is a fundamental theory in the area of proof complexity [Par71, Coo75, Bus86], and is useful for translating theorems to propositional logic proofs.

We say that two Turing machines M_1 and M_2 have a *PV-proof of equivalence* if the functional equivalence of M_1 and M_2 is provable in PV. We prove the following result:

Theorem 2 (iO for Turing Machines, Informal). *Assuming quasi-polynomial hardness of Learning with Errors, sub-exponentially secure one-way functions, and sub-exponentially secure indistinguishable obfuscation for circuits, there exists a construction of* iO *for Turing machines with unbounded-length inputs and PV-proofs of equivalence.*

On the use of Sub-exponential Assumptions. Although we rely on the sub-exponential security of the underlying primitives in our results, the hardness requirement for the underlying primitives is independent of the input length of the input circuits.

To our understanding, there is no obvious barrier to avoiding these sub-exponential assumptions due to the following observation: given a series of intermediate circuits, verifying Δ-equivalence only takes polynomial time, since checking whether two subcircuits of size $O(\log n)$ are functionally equivalent or not only takes $2^{O(\log n)} = \mathsf{poly}(n)$ time. Hence, constructing ΔiO for Δ-equivalent circuits from polynomial hardness is not ruled out by the input-length barrier. We therefore view our use of sub-exponential assumptions as a technical limitation that we can hope to overcome in the future.

1.2 Applications

We now discuss applications of our results towards building witness encryption and succinct non-interactive arguments (SNARGs) with properties that were not known to be achievable earlier. Our results for these primitives apply for a subclass of $\mathcal{NP} \cap co\mathcal{NP}$ languages whose disjointness with its complement can be proven in some logic system.

We start by characterizing this class of languages.

Mathematical Proof of Disjointness. Intuitively, we say a language $L \in \mathcal{NP} \cap co\mathcal{NP}$ has proof of disjointness, if "$L \cap \overline{L} = \phi$" can be proven in some mathematical logic system, where $\overline{L} = \{0,1\}^* \setminus L$ is the complement of L and both $L, \overline{L}$ are represented by circuits or Turing machines.

Specifically, let $\{M_n\}_{n\in\mathbb{N}}$ and $\{\overline{M}_n\}_{n\in\mathbb{N}}$ be the circuit families that define the $\mathcal{NP}$-relation of L and $\overline{L}$ respectively. We say that L has propositional proof of disjointness, if "$\overline{M}_n(x, \overline{w}) = 1 \longrightarrow M_n(x, w) \neq 1$" has polynomial-size proofs in the extended Frege system. This intuitively requires that the statement

"For any x, if $\overline{M}_n(x, \cdot)$ is satisfiable, then $M_n(x, \cdot)$ is not."

can be proven in propositional logic sytem. Similarly, let $M, \overline{M}$ be the Turing machines that defines $L, \overline{L}$ respectively. We say L has PV proof of disjointness, if $\overline{M}(x, \overline{w}) = 1 \rightarrow M(x, w) \neq 1$ can be proven in Cook's theory PV. Since propositional translation [Coo75] can translate a PV proof to polynomial-size propositional proofs, PV proof of disjointness implies propositional proof of disjointness.

What languages have proofs of disjointness? We expect that for most $\mathcal{NP} \cap co\mathcal{NP}$ languages that we are interested in, we can write a mathematical proof of disjointness. Indeed, otherwise it is hard to convince ourselves that the language is in $\mathcal{NP} \cap co\mathcal{NP}$. We give a concrete example below, namely, the language TAUT in computational complexity. Furthermore, we will show in Section 1.3 that for cryptographic applications, a large part of such mathematical proofs can be formalized in theory PV.

Example. TAUT is the language that contains all tautologies. Recall that a tautology is a formula that always evaluates to true for any truth assignment. TAUT is known to be $co\mathcal{NP}$-complete and hence is an important language in complexity theory.

By the completeness theorem of propositional logic [Bus98], any tautology has a proof in propositional logic. However, such a proof may not have a polynomial size. Hence, to ensure the honest prover/encryptor runs in the polynomial-time in the setting of SNARGs/WE, we consider a slight variant of TAUT, which is the following promise language $L_{\mathsf{TAUT}} = (L_{\mathsf{YES}}, L_{\mathsf{NO}})$. L_{YES} contains all tautologies with a polynomial-bounded propositional logic proof, whereas L_{NO} contains all non-tautologies. Then PV proof of disjointness can be extended naturally to promise languages: we require "$L_{\mathsf{YES}} \cap L_{\mathsf{NO}} = \phi$" can be proven in theory PV. Cook [Coo75] showed that the soundness of propositional logic system is provable in PV, which implies that L_{TAUT} has PV proof of disjointness.

We now proceed to discuss applications to witness encryption and SNARGs.

I. Witness Encryption. A witness encryption (WE) scheme allows an encryptor to use an instance x from a language L to encrypt a message m such that anyone who knows a witness w for x can retrieve the message m. Security requires that if $x \notin L$, then the ciphertext hides m. As discussed earlier, all prior constructions of WE only support bounded witness lengths due to the input-length barrier.

As a generic application of Theorem 1, we build a WE scheme for any language $L \in \mathcal{NP} \cap co\mathcal{NP}$ with propositional proof of disjointness, with security loss independent of the witness length. Furthermore, as an application of Theorem 2, we build a WE scheme for Turing machines for any language L in $\mathcal{NP} \cap co\mathcal{NP}$ with PV proof of disjointness. The latter scheme can support witnesses of *unbounded length*. The ciphertext size is independent of the witness length, but grows with the running time of the Turing machine $\overline{M}$.

II. Succinct Non-Interactive Arguments. A non-interactive argument system for an $\mathcal{NP}$ language L is said to be *succinct* if the proof size is much smaller than the witness size. Gentry and Wichs (GW) [GW11] proved that such argument systems cannot be constructed with a black-box proof of *adaptive*[4] soundness to falsifiable assumptions. On the other hand, a *non-adaptively* sound construction based on iO was given by Sahai and Waters (SW) [SW21].

While iO is not a falsifiable assumption, one can instantiate the SW construction with a recent iO scheme (such as [JLS21]) to obtain a scheme based on falsifiable assumptions. This resulting scheme, however, incurs a security loss exponential in the witness length due to the input-length barrier to iO. This has two consequences: first, this means that the scheme bypasses the GW lower bound due to the fact that the security reduction is able to decide the language.[5] Second, the scheme can only handle witnesses

[4]Adaptive (resp., non-adaptive) soundness refers to the setting where the adversary can choose the challenge instance after (resp., before) viewing the common reference string.

[5]Indeed, this scheme can also achieve adaptive security by standard complexity leveraging (over the instances) without further security degradation.

of a priori bounded length; in particular, the size of the common reference string (which contains the obfuscation) grows with the size of the witness.

We show how to overcome these limitations by constructing SNARGs that can support witnesses of *unbounded length* for any language $L \in \mathcal{NP} \cap co\mathcal{NP}$ with PV proof of disjointness. The CRS size is independent of the witness length and only depends on the running time of the Turing machine $\overline{M}$ that defines $\overline{L}$. Our base scheme is non-adaptively sound, but by standard complexity leveraging over the instances, it can also achieve adaptive soundness.

An important step towards this obtaining this result is to build puncturable pseudorandom functions (PRFs) [BW13, BGI14, KPTZ13] with a PV-proof of functionality preservation. As we discuss shortly, puncturable PRFs based on the GGM PRFs [BW13, BGI14, KPTZ13, SW21] satisfy this property.

1.3 How to Use iO with Proofs of Equivalence

We provide some general guidance for building new applications using our results. We consider some tools that are commonly used within iO-based applications and demonstrate how one can formalize properties about such tools in propositional logic or theory PV. Such proofs can then be used to build proofs of equivalence of circuits or Turing machines involved in the desired application.

In Section 1.3.1, we consider puncturable PRFs that are used ubiquitously in constructions involving iO [SW21]. Specifically, we show that the functionality preservation property of GGM-based puncturable PRFs [BW13, BGI14, KPTZ13, SW21] can be proven in theory PV. Next, in Section 1.3.2, we provide general guidance on proving properties of tools in group-based cryptography and lattice-based cryptography. As concrete examples, we demonstrate that the correctness of ElGamal encryption [ElG85] and Regev's encryption [Reg05] can be proven in theory PV .

1.3.1 Puncturable PRFs

A *puncturable* PRF [BW13, BGI14, KPTZ13, SW21] PRF_{punc} is a pseudorandom function with the additional property that allows one to puncture the PRF key k at any point x^* to obtain a punctured key $k \setminus \{x^*\}$. For each $x \neq x^*$, the functionality preservation property guarantees that $\mathsf{PRF}(k, x) = \mathsf{PRF}_{punc}(k \setminus \{x^*\}, x)$.

iO-based constructions that involve the use of puncturable PRFs require the functionality preservation property to establish the functional equivalence of the two programs being obfuscated. Since our constructions require proofs of equivalence in theory PV, this translates to requiring that the functionality preservation property of PRF_{punc} can be proven in theory PV. Formally, we say that a puncturable PRF PRF_{punc} has a PV proof of functionality preservation if the algorithms PRF_{punc}, PRF and the puncturing algorithm can be defined in PV as function symbols and there exists a proof in PV for $x \neq x^* \rightarrow \mathsf{PRF}(k, x) = \mathsf{PRF}_{punc}(k \setminus \{x^*\}, x)$.

We observe that the GGM-based construction of puncturable PRFs has a PV proof of functionality preservation. We emphasize that we do not need to modify the GGM construction nor its natural mathematical proof of functionality preservation. All we need to do is formalize the existing mathematical proof of functionality preservation in theory PV. It is important to note that theory PV does not allow general proof-by-induction rules. Instead, it only allows the following "polynomial-time induction" rule.

$$\text{If } \Phi(0) \text{ holds and } \Phi(x) \rightarrow (\Phi(2x) \wedge \Phi(2x+1)) \text{ holds for every } x, \text{ then } \Phi(x) \text{ holds for all } x,$$

where $\Phi(x)$ is a formula in PV.

Fortunately, the binary tree structure of the GGM construction is naturally compatible with the polynomial-time induction rule. Hence, the functionality preservation property can be naturally formalized in PV.

1.3.2 Proving Arithmetic Properties in *PV*

In addition to puncturable PRFs, iO-based applications often involve the use of cryptographic primitives such as commitment schemes and encryption schemes. In such cases, key properties of these primitives such as perfect binding or correctness of decryption are essential for establishing the functional equivalence of the programs being obfuscated. We now discuss how such properties can be proven in theory *PV* when the cryptographic primitives are instantiated using group-based cryptography and lattice-based cryptography.

The general principle involves the following two steps:

- First, write a mathematical proof of such property in natural language.
- Second, examine the basic theorems and axioms used in the mathematical proof to ensure that they can be formalized in theory *PV*.

For illustration purposes, we demonstrate how to prove correctness of group-based and lattice-based public key encryption schemes in theory *PV*.

Instantiation from Groups. As an example in group-based cryptography, we show how to prove the correctness of ElGamal encryption [ElG85] in theory *PV*.

Recall that the public key of ElGamal encryption is of the form (g, g^s) where s is the secret key, and $g \in \mathbb{G}$ is a group element. To encrypt a message $m \in \mathbb{G}$ with random coins r under the public key, we compute the ciphertext as $(g^r, (g^s)^r \cdot m)$.

Following the general principle described above, we can prove the correctness in *PV* as follows:

- We first write down the mathematical proof of correctness of ElGamal in natural language, as follows. Let (c_1, c_2) be a ciphertext, then $c_1 = g^r, c_2 = (g^s)^r \cdot m$. The decryption algorithm Dec computes

$$\mathsf{Dec}((c_1, c_2), s) = c_2/{c_1}^s = (g^s)^r \cdot m/(g^r)^s = (g^s)^r \cdot m/(g^s)^r = m \cdot ((g^s)^r \cdot /(g^s)^r) = m.$$

- **Formalization in *PV*:** The above mathematical proof only relies on some basic theorems in arithmetic such as commutative law and associative law of modular multiplication and $(g^s)^r = (g^r)^s$. All such basic theorems can be formalized and proven in *PV* [Coo75, Bus86]. Therefore, the above mathematical proof can be formalized in *PV*.

For more details, see Section 8.

Instantiation from Lattices. Using the above ideas, one can also prove the correctness of Regev's public key encryption scheme [Reg05] in *PV*. The main point is that the proof of correctness in natural language only uses some basic arithmetic theorems such as commutative law, distributive law, and some basic properties about inequalities to reason about rounding operations. By Buss's work [Bus86], all such theorems can be proven in *PV*. For more details, see Section 8.

1.4 Discussion and Future Directions

On Propositional Logic and Theory *PV*. Since proofs in propositional logic are central to our results, it is important to understand their provability. If one does not care about the *proof length*, propositional logic is quite powerful due to the completeness theorem [Bus98] which says that any semantically true formula[6] in propositional logic has a proof. Furthermore, we expect that most theorems proven in mathematical logic systems other than propositional logic (e.g. Peano Arithmetic) can also be represented in

[6]A propositional formula is semantically true if it always evaluates to true under any truth assignments.

propositional logic if we set a bound on the number of digits in the natural numbers, and use truth variables in propositional logic to represent the digits of natural numbers.

Propositional logic is powerful enough for proving the equivalence of two Turing machines: for any two functionality equivalent polynomial-time Turing machines, we can set an upper bound on the input length that is super-polynomial in the security parameter. Then, by the completeness theorem of propositional logic, there always exists a propositional logic proof of equivalence for the two Turing machines under the given input bound. However, there is no guarantee that such proofs in propositional logic have *polynomial size*.

Our results crucially require the proof size to be a polynomial. Thus, it is important to understand what can be proven with *polynomial-size propositional proofs*. This question has been extensively studied in proof complexity. In [Coo75], Cook introduced a theory PV to formalize the intuition of "polynomial-time reasoning" and showed that any proof in PV can be translated to a polynomial-size propositional logic proof. Later, a series of works [PW85, Bus86, KP90] proposed other propositional translations. In this work, we use PV since it is conceptually the simplest. PV allows the definition of new function symbols using Cobham's characterization of polynomial time functions [Cob65]. Basic arithmetic operations can be introduced in this way, and their related properties can be proved in PV.

On the positive side, Cook [Coo75] suggested that a good part of elementary number theory can be formalized in PV if the theorems are stated carefully. In the area of linear algebra, [SC04] showed that the Cayley–Hamilton theorem, basic properties of determinants, and basic matrix properties can be proven in PV. For theorems in complexity, it is known that the Cook-Levin theorem and PCP theorem can be formalized and proven in PV [CK07, Pic15]. Indeed, Cook observed that the correctness of "natural" polynomial-time algorithms usually can be proven in PV [Coo]. In this work, we give evidence that a large part of cryptographic algorithms fall in this category. They include functionality preservation of puncturable PRFs and the correctness of ElGamal Encryption [ElG85] and Regev's encryption [Reg05] (see Section 1.3).

On the negative side, it is known that Fermat's little theorem is unlikely to be provable in PV unless factoring can be solved in polynomial time, due to the witnessing theorem [Bus86]. Because of the same reason, the correctness of any polynomial-time algorithm that decides primes is unlikely to be proven in PV. Moreover, assuming $\mathcal{NP} \neq co\mathcal{NP}$, there are tautologies that can not be proven with polynomial-size proofs in propositional logic, because TAUT is $co\mathcal{NP}$-complete [CR79].

Beyond Theory PV. As discussed earlier, our approach relies on polynomial-size proofs in propositional logic. To increase the scope of our approach, a future direction is to handle *super-polynomial size* propositional proofs. The main challenge is that in our present approach, the sizes of the intermediate circuits grows with the size of the propositional proofs, and thus the obfuscated program will be super-poly size if we naively rely on super-polynomial size propositional proofs. We hypothesize that a potential solution could be to restrict the logic system to "bounded-space reasoning" theories, and finding a more clever way to build the intermediate circuits from propositional logic proofs.

An alternate future direction is to generalize our idea to leverage the "local" property of proofs in more powerful logic systems such as Buss's theories S_2^i, T_2^i [Bus86] since more theorems can be proven in them. Ultimately, one might ask if we can build iO for programs whose equivalence is provable in Zermelo–Fraenkel set theory with the axiom of choice (ZFC). Since ZFC is the most common foundation of mathematics, such a result might be sufficient for most applications of iO. The main seeming difficulty towards this goal is that our current method crucially relies on the property that each line of the propositional proof is also a *circuit*, whereas a line in ZFC is naturally a *Turing machine* that evaluates the truthfulness of that line. Hence, an interesting future direction is to extend our "gate-by-gate" framework to "Turing-machine-by-Turing-machine" framework to support ZFC.

Towards $\mathcal{NP} \cap co\mathcal{NP}$. Our method of leveraging mathematical proofs limits us to circuits whose equivalence can be *verified* in polynomial time. Since circuit equivalence is trivially in $co\mathcal{NP}$, ideally, we could hope to bypass the input-length barrier for the language of circuit pairs in $\mathcal{NP} \cap co\mathcal{NP}$ [LZ17].

Our work makes an important attempt in this direction. We note, however, that not all pairs of circuits whose functionally equivalence is in $\mathcal{NP} \cap co\mathcal{NP}$ necessarily have short mathematical proofs. Therefore, fully realizing the above vision is an important goal for future work.

Comparison with Decomposable iO. Liu and Zhandry [LZ17] introduced the notion of decomposable iO to unify prior works [GPS16, GS16, GPSZ17] that attempt to avoid the use of sub-exponential hardness assumptions in some specific applications of iO. In the same work [LZ17], Liu and Zhandry proved that deciding whether two circuits are "decomposing-equivalent" is in $\mathcal{P}$. This naturally limits the applicability of their framework. For example, it cannot support the Sahai-Waters construction of public-key encryption from iO and pseudorandom generators [SW21]. This is because the security of the pseudorandom generator implies that the two circuits of consideration in the security proof *cannot* be "decomposing-equivalent" since the latter is in $\mathcal{P}$. Indeed, a similar issue arises in many other applications and for this reason, decomposable iO is applicable when it is easy to check equivalence. (See Section 1.5 in [LZ17] for more discussion.)

Our work does not require circuit equivalence to be decidable in $\mathcal{P}$. Instead, we only require the *existence* of a *witness* that allows us to verify the equivalence of two circuits, where the witness is a polynomial-size propositional logic proof. In general, deciding whether the equivalence of two circuits has a short propositional logic proof is not known to be in $\mathcal{P}$.

On Our Gate-by-Gate Template for iO. In this work, we develop a new (topology preserving) "gate-by-gate" template for building iO for general circuits from iO for "small" circuits.

While this approach is crucial towards obtaining our results, we observe that it also yields some additional features that might be beneficial in specific use cases. Suppose after distributing an obfuscated circuit, one wishes to modify some gates in the underlying circuit (e.g., for patching a vulnerability in a program) [AJS17, GP17]. Instead of obfuscating the modified circuit from scratch (which might be costly), our "gate-by-gate" template allows for easy replacement of the relevant gates in the obfuscated circuit. We defer a formal treatment of this property to future work.

2 Technical Overview

We now provide an overview of our results. In Section 2.1, we discuss how to establish Δ-Equivalence starting from propositional proofs of equivalence of two circuits. In Section 2.2, we describe our construction of iO for Δ-equivalent circuits. Finally, in Section 2.3, we describe our construction of iO for unbounded-input Turing machines with *PV* proofs of equivalence.

2.1 Δ-Equivalence from Propositional Proofs

We show that given two circuits C_1, C_2, if the proposition "$C_1(x) = C_2(x)$" can be proven in a propositional logic system with *extension axioms* such as *extended Frege system* ($\mathcal{EF}$), then C_1, C_2 are Δ-equivalent up to some padding. That is, we can find a series of intermediate circuits $C'_1, C'_2, \ldots, C'_\ell$ with the same topology such that every two adjacent circuits C'_i, C'_{i+1} only differ in a *logarithmic* number of gates, and the subcircuits formed by these gates in C'_i, C'_{i+1} are functionality equivalent. Furthermore, the initial circuit C'_1 and the final circuit C'_ℓ are obtained by padding C_1, C_2, respectively, with some dummy gates.

Background. We first recall the definition of propositional proof systems with extension axioms. Such logic systems can be described as a set of variables and connectives including "$\rightarrow$", "$\leftrightarrow$", "$\wedge$", "$\vee$", and "$\neg$", which refers to "imply", "equal", "and", "or", and "negation", respectively. A *proof* in the propositional proof system is a series of propositional formulas, where each formula is derived from one of the following cases.

- **Axiom:** The formula is in one of the following forms: $P \to (Q \to P)$, $(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$, or $\neg\neg P \to P$ where P, Q, R are formulas.
- **Modus Ponens:** The formula is in the form Q, and there are two previous formulas $P, P \to Q$ derived before the current formula.
- **Extension:** The formula is in the form $e \leftrightarrow Q$, where e is a new variable that does not appear in Q and all previous formulas. This rule is used to introduce intermediate variables and hence can shorten the proof size.

For a more detailed description of propositional logic proofs, see our preliminary (Section 3.1). For any circuit, we can treat each of its wires as a variable in propositional logic, whose truth value represents the wire value. Then the mathematical statement "$C_1(x) = C_2(x)$" can be formalized in $\mathcal{EF}$ as a formula.

Assuming there exists a proof $\pi = (\theta_1, \theta_2, \ldots, \theta_k)$ in propositional logic for $C_1(x) = C_2(x)$, we now prove that C_1, C_2 are Δ-equivalent. Equivalently, we only need to show that we can transform from C_1 to C_2 via a series of incremental changes, where each change replaces a logarithmic size subcircuit with a functionally equivalent new subcircuit. To illustrate our high-level ideas, we firstly ignore the topology of the circuits, and hence we can add gates and delete gates arbitrarily. Since we can always treat extension rules as introducing a new wire in the circuit, we also assume there are no extension rules for simplicity.

Our transformation is based on the following key observations.

- The proof π is "local", i.e., the truthfulness of each θ_i follows from a constant number of previous formulas in $\theta_1, \ldots, \theta_{i-1}$.
- The propositional formulas $\theta_1, \theta_2, \ldots, \theta_k$ can also be regarded as boolean circuits, since the connectives including "$\to$" can be expressed as the combination of $\wedge$, $\vee$, and $\neg$ gates.

A Sketch of the Transformation. Based on these observations, our transformation from C_1 to C_2 proceeds in the following phases. We start with a circuit C that is the same as C_1. After the following incremental changes to C, C will become C_2.

- **Grow C_2.** We add the circuit $C_2(x)$ to C in a gate-by-gate manner. Specifically, we add each gate of C_2 in the topological order to C, while the output wire of C is still set to be the output wire of $C_1(x)$.

  We only change the circuit C for a constant number of gates when we add a gate, since we can always assume such a gate has a constant arity without loss of generality.
- **Grow the Proof.** We add the formulas $\theta_1, \theta_2, \ldots \theta_k$ in the proof π one by one to C as follows. Note that each formula θ_i can be regarded as a circuit that computes the truth values of θ_i from its variables.

  Firstly, we add θ_1 to C by modifying the output of C as $C_1(x) \wedge \theta_1$. Similarly, to add θ_2, we further modify the output of C to be $C_1(x) \wedge \theta_1 \wedge \theta_2$. We continue this process until all $\theta_1, \theta_2, \ldots, \theta_k$ are added. Then the output of C becomes $C_1(x) \wedge \theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_k$.

  We now show that we only change a small subcircuit in each step of the above process. There are three cases for each i, depending on how θ_i is derived.

  - **Axiom:** In this case θ_i is one of the axioms, for example, θ_i is in the form $P \to (Q \to P)$. We can assume without loss of generality that P, Q are constant size formulas, as we can always reduce the size of P, Q by assigning their subformulas to new variables using the extension rule.

In this case the change from $C_1(x) \wedge \theta_1 \wedge \ldots \wedge \theta_{i-1}$ to $C_1(x) \wedge \theta_1 \wedge \ldots \wedge \theta_{i-1} \wedge \theta_i$ can be regarded as replacing a subcircuit that always outputs 1 with a new subcircuit θ_i. The functionality equivalence between the two subcircuits follows from the fact that axioms must be tautologies.

  - **Modus Ponens:** For this case, there exists some P, Q such that $P, P \to Q$ are the formulas derived in the first $(i-1)$ formulas, and the current formula θ_i is Q. Similar to the case of axioms, we can assume P, Q are constant-size formulas.

    In this case the change from $C_1(x) \wedge \ldots \wedge P \wedge \ldots \wedge (P \to Q) \wedge \ldots \wedge \theta_{i-1}$ to $C_1(x) \wedge \ldots \wedge P \wedge \ldots \wedge (P \to Q) \wedge \ldots \wedge \theta_{i-1} \wedge Q$ can be regarded as replacing a subcircuit $P \wedge (P \to Q)$ with a new subcircuit $P \wedge (P \to Q) \wedge Q$. The functionality equivalence can be proved by enumerating all possible truth assignment to P and Q.

- **Change the Output.** Let o_1, o_2 be the output wires of C_1, C_2 respectively. Then a proof of "$C_1(x) = C_2(x)$" ends with $o_1 \leftrightarrow o_2$. Namely, θ_k is the formula $o_1 \leftrightarrow o_2$. Hence, we can replace the output of C, which is $o_1 \wedge \theta_1 \wedge \ldots \wedge \theta_k$, with $o_2 \wedge \theta_1 \wedge \ldots \wedge \theta_k$. This step is an incremental change, since it can be regarded as replacing the subcircuit $o_1 \wedge (o_1 \leftrightarrow o_2)$ with $o_2 \wedge (o_1 \leftrightarrow o_2)$.

- **Shrink the Proof.** This phase deletes $\theta_1, \theta_2, \ldots, \theta_k$ in the circuit C. Specifically, we remove $\theta_k, \theta_{k-1}, \ldots \theta_1$ one by one in the reversing order that they are added.

  This process is a series of incremental changes for the same reason as the "Grow the Proof" phase.

- **Shrink C_1.** At the beginning of this phase, the circuit C outputs o_2, which is the output wire of $C_2(x)$. The circuit C_1 is still in C, but its output wire o_1 is not used anywhere. Then we delete the gates of C_1 in C one by one in the reverse topological order. Finally, we obtain the circuit $C = C_2$.

  Deleting a gate of C_1 in this phase is an incremental change for the same reason as the "Grow C_2" phase.

The reader may already notice that the above sketch oversimplifies many details. For example, the output of the circuit C is computed as a series of $\wedge$-gates, i.e. $C(x) = o_1 \wedge \theta_1 \wedge \theta_2 \ldots$ in the "Grow the Proof" phase, and we argue that we change the subcircuit $P \wedge (P \to Q)$ to $P \wedge (P \to Q) \wedge Q$. However, in the reality, we need to use the arity-2 $\wedge$-gates to implement the series of $\wedge$-gates in $C(x)$. Then $P \wedge (P \to Q)$ and $P \wedge (P \to Q) \wedge Q$ may not be subcircuits, since the positions of P, Q may not be consecutive in the circuit.

Building An AND Tree. We resolve this issue by implementing the series of $\wedge$-gates as a binary tree of $\wedge$-gates. Initially, on every leaves there is a gate that always outputs 1. Then in the "Grow the Proof" phase, we replace the leaves with θ_i's one by one. Now, for each $\theta_i = Q$ obtained from modus ponens, the subcircuit consists of the root-to-leaf paths of $P, P \to Q$ and θ_i. This subcircuit contains only $O(\log k)$ gates, which is logarithmic.

Handling Extension Rules. Another issue is how to handle the extension rules. Indeed, there is an additional phase "Grow the Extension" between the "Grow C_2" phase and the "Grow the Proof" phase, where we handle all the extensions by introducing new wires in the circuit. Specifically, for any extension of the form $e \leftrightarrow Q$, we add a new wire e and set it as the output wire of a circuit that computes Q. Here we can also assume Q is only constant size for the same reason as the "Grow the Proof" phase. Also, between the "Shrink the Proof" phase and the "Shrink C_1" phase, we add a phase "Shrink the Extension" to delete the wires in the reverse order that they are introduced.

More technical issues raise when we build iO leveraging the series of incremental changes above. As we will show later, our construction of iO for Δ-equivalent circuits does not hide the *topology* of the input circuit. As a result, in our Δ-equivalence definition, we require the circuits C_1, C_2 and their intermediate circuits $C'_1, \dots, C'_\ell$ have the *same topology*.

Padding the Circuit. To further preserve the topology during the incremental changes, we build a padding algorithm Pad that can pad C_1, C_2 to the two circuits with the same topology. Moreover, the above series of incremental changes can now be modified to further preserve the topology of the circuit.

To achieve this, the idea is to implement the adding and deleting gates via changing the functionality of the gates, instead of really adding or deleting them. Specifically, we build the following two helper circuits Copy and Proj, which provide multiple output gate functionality and select input wires from all existing wires, respectively.

- Copy: The Copy circuit copies a wire to multiple wires of the same value. It is constructed as a binary tree of gates, where each tree node is a 2-output-1-input gate that either copies its input wire to the two output wires, or always outputs zeros.

- Proj: The Proj circuit selects a wire from multiple wires. This circuit is also construed as a binary tree of gates, where each tree node is a 2-input-1-output gate that either outputs its left child or its right child.

Due to the tree structure, we can add a copy of the input wire in Copy or change the wire selected in Proj by only changing a *logarithmic* number of gates.

Now we pad the input circuit C as follows. For each gate in C, the padding algorithm Pad attaches a Copy circuit to each output wire of the gates in C, and also attaches a Proj circuit to each input wire of the gates in C. Moreover, it connects the leaves of Proj circuits to all previous leaves of the Copy circuits. Finally, we add some dummy gates that do nothing with their output wires padded with Copy and input wires padded with Proj.

Now, to avoid changing the topology in the above incremental changes, instead of adding or deleting gates, we change the functionality of existing dummy gates, or modify the functionality of Proj and Copy circuits, while still keeping the size of the subcircuit logarithmic. For more details, see Section 4.3.

2.2 Construction of iO for Δ-equivalent Circuits

We now describe our construction of iO for Δ-equivalent circuits. Our high-level strategy is as follows:

- We first consider a notion of δiO, namely, iO for circuits that only differ by a small subcircuit. Specifically, we build δiO for any two circuits that only differ by two logarithmic-size functionally equivalent subcircuits.

- Next, we use δiO to obfuscate Δ-equivalent circuits as follows. Recall that for any Δ-equivalent circuits C_1, C_2, there is a polynomial number of intermediate circuits $C_1 = C'_1, C'_2, \dots, C'_\ell = C_2$, and each two adjacent circuits C'_i, C'_{i+1} only differ by two functionality equivalent logarithmic subcircuits. From the first step, it follows that for every i, $\delta\mathsf{iO}(C'_i)$ and $\delta\mathsf{iO}(C'_{i+1})$ are indistinguishable. By a hybrid argument, we can now establish the indistinguishability of $\delta\mathsf{iO}(C_1)$ and $\delta\mathsf{iO}(C_2)$.

Let us explain why this approach overcomes the input-length barrier. Whether two circuits only differ by two functionality equivalent subcircuits of *logarithmic* size can be decided in polynomial-time, since we only need to check all inputs to the *subcircuit* instead of all inputs to the *entire circuit*. Hence, the input-length barrier does not apply to δiO. Therefore, we can hope to build δiO without a security loss that is exponential in the input length.

Thus, the main task towards our goal is to build δiO. Towards this end, we present a new template for obfuscation that preserves the *topology* of the input circuit. This feature is crucial to proving security without incurring a loss exponential in the input length. In particular, it allows us to make "local" changes to leverage the fact the input pair of circuits only differ by a logarithmic-size functionally equivalent subcircuit. To the best of our understanding, this property is not satisfied by prior templates for obfuscation (see, e.g., [AJ15, BV15, CHJV15, BGL⁺15, KLW15, GS18]).

Our Gate-by-Gate iO Template. Our first attempt is to mimic the gate-by-gate construction of garbled circuits [Yao86] that preserves the structure of the input circuit. Specifically, for each gate g in an input circuit C, we use a "small" iO to obfuscate the gate functionality. Note that the input and output wires need to be encrypted since otherwise, an adversary can run the obfuscated program on arbitrary inputs and observe the truth table of the gate g. Towards this end, we associate a puncturable PRF key to each wire of the circuit, and use it to encrypt the wire value. Then, for each gate g, we obfuscate the following circuit $\mathsf{Gate}_g(\cdot,\cdot)$: it takes as input two ciphertexts that correspond to encryptions of g's input wires. It first decrypts the ciphertexts, computes the functionality of the gate g, and then encrypts the output wire value. In order to perform the decryption and encryption steps, Gate_g contains the puncturable PRF keys for the input and output wires of g hardwired in its description. The obfuscated circuit consists of the obfuscation of $\mathsf{iO}(\mathsf{Gate}_g)$'s for every gate g in C.

In order to prove security, the main idea is to only modify the obfuscation of the gates that correspond to the logarithmic-size subcircuit where the input circuits differ. Note that our use of existing iO schemes (that incur security loss exponential in the input length) does not pose a problem towards bypassing the input-length barrier because the input length of each Gate_g is much smaller than the input length of the entire circuit C.

Mix-and-Match Attacks. This initial attempt, unfortunately, suffers from "mix-and-match" attacks. An adversary can run the obfuscated program for several different inputs, and keep the ciphertexts of the intermediate wires. Later, the adversary can provide the "mixed" input ciphertexts *sourced from different inputs* to some gate $\mathsf{iO}(\mathsf{Gate}_g)$. Then the adversary might learn more input-output pairs of Gate_g than the functionality of the circuit Gate_g should have provided, and thus we have no hope to prove the security of the above construction.

To prevent such attacks, we can modify the construction as follows: let $\mathsf{ct}_l, \mathsf{ct}_r$ denote the "left" and "right" input ciphertexts to Gate_g. The modified Gate_g additionally takes the *entire* inputs x_l, x_r to C that lead to the input ciphertexts $\mathsf{ct}_l, \mathsf{ct}_r$ and checks whether $x_l = x_r$. In order to "tie" the entire input with a ciphertext, we use another puncturable PRF to compute a message-authenticate code (MAC) over the pair (ct_l, x_l) and similarly (ct_r, x_r)

It is not difficult to see that this modified construction prevents "mix-and-match" attacks. Intuitively, only the ciphertexts generated by Gate_g can have a valid MAC, and hence the mix-and-match attacks can be caught by the consistency check over the inputs x_l, x_r. Unfortunately, however, the input length of Gate_g is now as large as the input length of C. This means that this construction will incur a security loss exponential in the input length of C.

An Intermediate Step. Towards overcoming this problem, we first describe a modified construction that improves upon the above but only for specific circuits, namely, ones in NC^0. As we will see shortly, it serves as a useful basis towards our final solution for general circuits.

Our starting idea is to leverage the fact that each gate in C might not depend on the *entire* input of C. Hence, we can modify Gate_g such that it only takes as input the input wire values of C that g depends upon. To characterize *dependency*, we introduce the notation $\mathsf{dep}(w)$ to denote the set containing *all the intermediate wires* that a wire w depends upon, excluding itself. Note that $\mathsf{dep}(w)$ includes not only the input wires but also the internal wires of C. Formally, $\mathsf{dep}(w)$ is defined inductively as follows:

– For each input wire w of C, define $\mathsf{dep}(w)$ as an empty set.

– For each wire w that is the output wire of some gate g with two input wires l, r, we define $\mathsf{dep}(w) = \mathsf{dep}(l) \cup \mathsf{dep}(r) \cup \{l, r\}$.

We modify the circuit Gate_g as described in Figure 1. The input wires to g are denoted as l, r and o denotes its output wire.

Circuit Gate_g

- **Input**: Ciphertexts $\mathsf{ct}_l, \mathsf{ct}_r$ and their MACs σ_l, σ_r; Ciphertexts that l, r, o depend on, respectively: $CT_l := \{\mathsf{ct}_w\}_{w \in \mathsf{dep}(l)}, CT_r := \{\mathsf{ct}_w\}_{w \in \mathsf{dep}(r)}, CT_o := \{\mathsf{ct}_w\}_{w \in \mathsf{dep}(o)}$.
- **Hardwires:** The MAC keys and puncturable PRF keys of encryption for l, r, o.
- **Consistency Check:** Check whether the sets CT_l (resp. CT_r) and CT_o contains the same ciphertexts in $\mathsf{dep}(l)$ (resp. $\mathsf{dep}(r)$). Also check whether ct_l and ct_r is consistent with CT_o.
- **Verification:** Verify the MACs σ_l, σ_r with respect to the messages $\mathsf{ct}_l || CT_l$, $\mathsf{ct}_r || CT_r$, respectively.
- **Compute:** Decrypt the ciphertexts $\mathsf{ct}_l, \mathsf{ct}_r$, compute the gate g, and encrypt the output wire value using the randomness generated by the puncturable PRF with input CT_o. Sign $\mathsf{ct}_o || CT_o$ for the output wire.
- **Output:** Ciphertext ct_o and its MAC.

Figure 1: Modified circuit Gate_g.

The intuition behind security is the same as earlier. For simplicity, we only consider the indistinguishability of a pair of intermediate circuits C'_i, C'_{i+1} (obtained from Δ-equivalence). Let S denote the (logarithmic-size) subcircuit where these two circuits differ. In the security proof, for every $g \in S$, we replace the obfuscation of Gate_g with an obfuscation of circuit $\mathsf{Gate}_g^{\mathsf{direct}}$ that does the same consistency check and verification as above, but decrypts the ciphertexts in $\{\mathsf{ct}_w \mid w \in \mathsf{dep}(o) \text{ and } w \text{ is an input wire of } C\}$ and computes the output wire o *directly* from those input wire values. We perform such replacement for every $g \in S$ inductively in the topological order. At an induction step, for a given gate g, all gates that g depends on have been replaced with the above new gates. We first leverage the security of the MACs to argue that if the verification of the MACs passes, then the decrypted input wire values must match the values computed directly from the input wires of C. Next, we can replace Gate_g with $\mathsf{Gate}_g^{\mathsf{direct}}$.

The security loss incurred by this modified construction is exponential in the input length of Gate_g. This loss is small when C is in NC^0 since any output bit of an NC^0 circuit only depends on a constant number of input bits. However, for general circuits, $\mathsf{dep}(l)$ and $\mathsf{dep}(r)$ may contain the entire input in the worst case. In such a scenario, the security loss is still exponential in the input length of C.

Shrinking Input Length via Hashing. To resolve this issue, we observe that in the above security proof, $\mathsf{Gate}_g^{\mathsf{direct}}$ does not even need to know every ciphertext in $\mathsf{dep}(l) \cup \mathsf{dep}(r)$ to compute the wire value of o. Instead, the wire o only depends on the wires in $\mathsf{dep}(o)$ that are also the input wires of the subcircuit S. For ease of representation, we use $\mathsf{inp}(S)$ to denote the input to S. Since the size of $\mathsf{dep}(o) \cap \mathsf{inp}(S)$ is only logarithmic, if we modify Gate_g to take as input the ciphertexts in $\mathsf{dep}(o) \cap \mathsf{inp}(S)$ instead, then we significantly shorten the input length of Gate_g.

However, we can not provide the above set as an explicit input to Gate_g since S is not known in the *construction* of δiO; instead, it is only available in the security reduction. If we hardwire S in (the public description of) Gate_g in an intermediate hybrid of the security proof, then we can not hope to argue indistinguishability. Hence, we need to *hide* the set S and at the same time also provide the above set of ciphertexts in $\mathsf{dep}(o) \cap \mathsf{inp}(S)$ as an input to Gate_g.

To achieve these two properties simultaneously, we use a somewhere extractable hash function (SEH) [HW15] to hash the ciphertexts in $\mathsf{dep}(l)$ and the ciphertexts in $\mathsf{dep}(r)$. We set the hash function to be extractable for the ciphertexts in $\mathsf{dep}(l) \cap \mathsf{inp}(S)$ and $\mathsf{dep}(r) \cap \mathsf{inp}(S)$. The key indistinguishability property of SEH guarantees that the extraction locations are hidden in the hash key. Moreover, the size of the SEH hash value grows linearly in $|S|$.

Next, we modify the circuit Gate_g to take $\mathsf{Hash}(CT_l), \mathsf{Hash}(CT_r)$ as additional inputs, where CT_l (resp. CT_r) contains all ciphertexts that the wire l (resp. r) depends on. Then in the security proof, for each two adjacent intermediate circuits C'_i, C'_{i+1}, we first switch the set S to be the subcircuit that C'_i and C'_{i+1} differ on. Then, we replace Gate_g with a new $\mathsf{Gate}_g^{\mathsf{direct}\prime}$ that extracts the sets of ciphertexts in $\mathsf{dep}(o) \cap \mathsf{inp}(S)$ from the hash values and computes the output wires o directly from them.

However, an issue arises in arguing security since we need to enforce the consistency check of the ciphertexts in $\mathsf{dep}(l)$ and the ciphertexts in $\mathsf{dep}(r)$ given only their hash values. A natural idea is to further attach a *succinct* non-interactive proof that proves that the two hash values are consistent. Note that we seemingly need such a proof to be *statistically sound*; such proofs, however, are unlikely to exist.

Our key observation is that we in fact do not need a succinct proof with full statistical soundness. Instead, we only succinct non-interactive arguments (SNARGs) with the following *somewhere statistical soundness* property: for two hash values computed as above, the extracted ciphertexts are consistent. Namely, given the hash values h_l, h_r, h_o with respect to $\mathsf{dep}(l), \mathsf{dep}(r), \mathsf{dep}(o)$, respectively, if the extracted ciphertexts in $\mathsf{dep}(l) \cap \mathsf{inp}(S)$ and $\mathsf{dep}(o) \cap \mathsf{inp}(S)$ are inconsistent, or the extracted ciphertexts in $\mathsf{dep}(r) \cap \mathsf{inp}(S)$ and $\mathsf{dep}(o) \cap \mathsf{inp}(S)$ are inconsistent, then any proof computed by an unbounded cheating prover must be rejected.

We build such somewhere statistically sound SNARGs with only *poly-logarithmic* size proof and verification time from the polynomial hardness of learning with errors (LWE) by relying on the techniques in the recent work of [CJJ21]. In [CJJ21], the authors constructed SNARGs for the so-called *batch index* language with (semi-adaptive) somewhere extraction property from LWE, where an index language is an $\mathcal{NP}$ language where the instances are treated as indices that can be described in a logarithmic number of bits. We observe that a minor modification of their construction achieves (semi-adaptive) somewhere *statistical* soundness.

Armed with somewhere statistically sound SNARGs for the batch index language, we show how to build an SEH with consistency proofs. We start with the somewhere statistical binding hash construction of [HW15]. Their construction also allows extraction of the binding positions, and hence is also an SEH. Moreover, their construction has a Merkle tree structure, and thus supports succinct local openings. Namely, one can use a root-to-leaf in the Merkle tree to serve as a small-size opening for each bit in the string being hashed. To hash the index set CT_l, we first assign a unique integer to each wire. Then we arrange the elements in CT_l as an array. At the index w, if w index is non-empty in CT_l then we put ct_w at the w-th index. Otherwise, we put a special symbol $\perp$ at the w-th index. To generate a consistency proof for h_l and h_o, we use a SNARG for batch-index language to prove that for each wire w, there exists valid local openings to h_l and h_o at the index w, and if CT_l has a non-empty element at the index w, then CT_o also has the same element at the index w. Then the somewhere statistical soundness of SNARGs for batch-index implies the property we want from the consistency proof.

Summary of the Construction So Far. We summarize the modified construction of the circuit Gate_g. We replace the sets of ciphertexts CT_l, CT_r, CT_o in Figure 1 with their hash values, and replace the consistency check with the verification of the consistency proofs. To check the consistency between $\mathsf{ct}_l, \mathsf{ct}_r$ and ct_o, we additionally verify the local openings of $\mathsf{ct}_l, \mathsf{ct}_r$ with respect to the hash h_o. Specifically, we modify Gate_g as described in Figure 2.

<u>Circuit Gate_g</u>

- **Input**:
  - Ciphertexts $\mathsf{ct}_l, \mathsf{ct}_r$ and their MACs σ_l, σ_r
  - Hashes of the ciphertexts that the wires l, r, o depend on: h_l, h_r, h_o
  - Consistency proof π_l between h_l and h_o, and also the consistency proof π_r between h_r and h_o
  - Local openings to $\mathsf{ct}_l, \mathsf{ct}_r$ with respect to h_o: ρ_l, ρ_r;
- **Hardwires:** The MAC keys and puncturable PRF keys of encryption for l, r, o.
- **Consistency Check:** Verify the consistency proofs π_l, π_r with respect to $(h_l, h_o), (h_r, h_o)$, respectively. Also verify the local openings ρ_l, ρ_r with respect to the hash h_o for $\mathsf{ct}_l, \mathsf{ct}_r$, respectively.
- **Verification:** Verify the MACs σ_l, σ_r of the messages $\mathsf{ct}_l||h_l$, $\mathsf{ct}_r||h_r$, respectively.
- **Compute:** Decrypt the ciphertexts $\mathsf{ct}_l, \mathsf{ct}_r$, compute the gate g, and encrypt the output wire value using the randomness generated by the puncturable PRF with input h_o. Then sign $\mathsf{ct}_o||h_o$ for the output wire.
- **Output:** Ciphertext ct_o and its MAC.

Figure 2: Modified circuit Gate_g.

The security proof remains almost the same as before. We still inductively replace Gate_g with $\mathsf{Gate}_g^{\mathsf{direct}}$ for each gate g in the subcircuit S in the topological order. The only difference is that we now leverage the somewhere statistical soundness of the consistency proof to ensure that for any ciphertexts $\mathsf{ct}_l, \mathsf{ct}_r$ that passes the consistency check and verification of the MACs, the decrypted input wire values from $\mathsf{ct}_l, \mathsf{ct}_r$ must match the values computed directly from the ciphertexts extracted from h_o. For more details, see Section 6.3.

It looks like we have bypassed the input-length barrier, since the input length to Gate_g seems to be independent of the input length of C. However, a careful examination reveals that this is not the case. Specifically, the bit-length of the hash value h_o is at least the size of one ciphertext plus a $\mathsf{poly}(\lambda)$ term, and the size of one ciphertext is at least the size of h_l or h_r. Hence, we have $|h_o| \geq |h_l| + \mathsf{poly}(\lambda)$. Therefore, the size of the hash value h_o grows at least linearly in the depth of the circuit. This leads to a linear dependence on the depth of the circuit in the input length of Gate_g.

Removing the Depth Dependence. To overcome this issue, we need to further shrink the hash values. Towards this end, our key observation is that we only require a weaker extraction property from SEH: instead of extracting the ciphertexts, we only need to extract the underlying messages. To implement this idea while achieving a reduction in the size of hash values, we make use of fully homomorphic encryption (FHE). Specifically, we use FHE to encrypt the SEH extraction trapdoor together with the puncturable PRF keys for the wires whose values we wish to extract from SEH. Now, given the SEH hash values h_l, h_r, h_o inside the circuit Gate_g, we first homomorphically extract the ciphertexts using the FHE encryption of the SEH trapdoor. Then we homomorphically decrypt the extracted ciphertexts. Finally, we obtain the FHE ciphertexts h'_l, h'_r, h'_o of size $\mathrm{poly}(\lambda)$-bits. Then we use these FHE ciphertexts as the input to the puncturable PRFs to encrypt the wires l, r, o and also authenticate them in MACs. This allows us to remove the hash size dependence on the depth of the circuit, thus finally allowing us to bypass the input-length barrier.

2.3 iO For Turing Machines with PV-proof of Equivalence

In this section, we leverage PV proof of equivalence for Turing machines to build iO for Turing machines with *unbounded* length inputs. We start by briefly recalling Cook's theory PV [Coo75].

Background on Theory PV. PV is a theory of open equations, i.e. theorems and lines in PV are equations about natural numbers (e.g. $a + b = b + a$). Roughly speaking, theory PV allows us to reason about propositions for all natural numbers, whereas formulas in propositional logic systems can only take finite many truthful assignments. In the initial work [Coo75], Cook showed that proofs in PV can be translated to polynomial-size propositional logic proofs[7], if one sets an upper bound on the length of the natural numbers in PV, and represents the digits of natural numbers by propositional variables. Hence, loosely speaking, proofs in PV can be viewed as a "uniform" version of polynomial-size propositional proofs. For more background on theory PV, see Section 3.2.

Starting Approach. To build iO for Turing machines with unbounded length inputs, we depart from prior templates.

To obfuscate a Turing machine M, we first translate it to a sequence of circuits $[\![M]\!]_1, [\![M]\!]_2, \ldots, [\![M]\!]_{\lambda^{\log\lambda}}$, where $[\![M]\!]_n$ represents the circuit that computes M with input-length n. We then use our iO scheme for Δ-equivalent circuits to obfuscate these circuits. Since we are only interested in polynomial-length inputs, it is sufficient to consider all n bounded by some super-polynomial $\lambda^{\log\lambda}$. Let us put aside the efficiency of this construction for the moment. Looking ahead, in our security proof, to prove the indistinguishability of the obfuscations of two Turing machines M_1, M_2 with PV proof of equivalence, we only use $\lambda^{\log\lambda}$ hybrids to switch the set of circuits $\{[\![M_1]\!]_i\}_i$ to $\{[\![M_2]\!]_i\}_i$ one-by-one, where in the i-th hybrid, switching from $[\![M_1]\!]_i$ to $[\![M_2]\!]_i$ only incurs a polynomial loss. Here, we leverage the fact that Cook's translation implies that $[\![M_1]\!]_i$'s and $[\![M_2]\!]_i$'s have polynomial-size propositional proofs of equivalence. In total, we only incur a $\lambda^{\log\lambda}$ loss in the security reduction, which is independent of the input length.

While promising, the above idea does not work as is since the above obfuscator is not efficient. In particular, it needs to obfuscate a super-polynomial number of circuits; hence it is not a polynomial-time algorithm.

Towards an Efficient Construction. To build a polynomial-time obfuscator, our intuition is that we can describe the circuit being obfuscated by a succinct circuit $[\![M]\!](n, i)$ that takes as input an input-length n and an index i, and outputs the description of i-th gate in $[\![M]\!]_n$. Next, instead of obfuscating the circuits $[\![M]\!]_1, [\![M]\!]_2, \ldots$ in a gate-by-gate manner as we did in Section 2.2, we obfuscate a "uniform gate" UGate, which uses $[\![M]\!](n, i)$ internally and provides the functionality for the i-th gate in $[\![M]\!]_n$.

Indeed, the construction of UGate circuits is almost the same as Gate_g in our construction of iO for circuits in Section 2.2, since most parts of the description of Gate_g are already "uniform" for every g, except the following two places.

- K_g, which contains independently sampled PRF keys for the input wires and output wires of the gate g. We can generate K_g uniformly by a new puncturable PRF, and assign each wire an index. Crucially,

[7]The "propositional logic" in this paper refers to extended Frege systems ($\mathcal{EF}$). Cook's initial translation is from PV to extended resolution. Later, it is shown that extended resolution and extended Frege can "simulate" each other with a polynomial overhead [CR79].

the bit-length of the index needs to be *succinct*, in the sense that it is independent of the input length of the Turing machine. Then we use the succinct index as the input to the new puncturable PRF to generate the puncturable PRF keys for the wires.

- The functionality of the gate g in the "**Compute**" phase of Gate_g. We can make this part uniform by computing the functionality of the gate g on-the-fly by $[\![M]\!](n, \ulcorner g \urcorner)$, where $\ulcorner g \urcorner$ is the index of the gate g.

In summary, we let the "uniform gate" $\mathsf{UGate}(n, \ulcorner g \urcorner, \cdot)$ be the following circuit. It takes as input the input-length n with $n \leq \lambda^{\log \lambda}$, the index of a gate $\ulcorner g \urcorner$, and all the inputs to Gate_g. UGate uses $[\![M]\!](n, \ulcorner g \urcorner)$ to obtain the functionality of g-th gate and also the indices of its input and output wire. Then it uses a puncturable PRF to generate the puncturable PRF keys K_g used by Gate_g using the gate information. Next, it computes Gate_g internally, and outputs whatever Gate_g outputs.

Indexing Wires and Gates Succinctly. In the above description, UGate internally uses the indices of the gates and wires of the circuit. In this work, we use $\ulcorner g \urcorner$ to denote the index of a gate g and use $\ulcorner w \urcorner$ to denote the index of a wire w. It is important that the size of such indices is essentially independent of the input length of the Turing machine, and the indices of the input and output wire of each gate g can be computed in polynomial-time given $\ulcorner g \urcorner$. Otherwise, the size of UGate will grow with the input length.

To assign a unique succinct number to each gate and wire, we observe that the gates in the padded circuit can be classified into the following four types.

- **Regular:** It carries out the actual computation in the input circuit.
- **Copy:** Gates of this type are part of the Copy helper circuit attached to the output wires of the regular gates.
- **Projection:** Gates of this are part of the Proj helper circuit attached to the input wires of the regular gates.
- **Tree:** Gates of this type are part of the AND tree used to compute the output of the entire circuit.

We observe that each type can be naturally assigned to a succinct number indicating the location of the gates. For example, for each gate in the "Copy" type, it can be uniquely indexed by $(\ulcorner g \urcorner, i)$, where $i \in \{0, 1\}^*$ indicates the path from the current gate to the root, and $\ulcorner g \urcorner$ is the index of a regular gate, denoting which output wire the Copy circuit copies. The actual numbering is more involved, for more details, see Section 7.3.

2.3.1 Proof of Security

To prove security, the idea is to enumerate an input-length n_0 in the hybrid argument. Then we puncture the "uniform gate" circuit UGate for the input-length n_0. Namely, for any two Turing machines M_1, M_2, we have UGate compute M_1 for $n < n_0$, and compute M_2 for $n \geq n_0$. Then we only need $\lambda^{\log \lambda}$ hybrids to change from M_1 to M_2. Hence, the main challenge of the security proof is how to switch from n_0 to $n_0 + 1$, without a loss that is exponential in the input-length.

As we mentioned earlier, we will use Cook's propositional translation [Coo75]. That is, if $M_1(x) = M_2(x)$ can be proven in *PV* for two Turing machines M_1, M_2, then for any integer n_0, there exists a propositional logic proof for $[\![M_1]\!]_{n_0} \leftrightarrow [\![M_2]\!]_{n_0}$ of size polynomial in n_0. Our initial idea is to use the same strategy as in the security proof of iO for Δ-equivalent circuits. However, the existence of such a propositional proof is not sufficient, because the obfuscations of two functionally equivalent circuits can only be indistinguishable when they are padded to the same length. This requires us to further ensure that the intermediate circuits discussed in Section 2.1 can be described succinctly by small circuits, otherwise the size of the obfuscated program will grow with the size of the intermediate circuits, which are super polynomial.

We abstract this succinctness requirement on the intermediate circuits as the following Δ-equivalence for Turing machines.

Δ-equivalent Turing Machines. Specifically, we say two Turing machines are Δ-equivalent, if for any integer n_0, there is a sequence of intermediate circuits that can be *described succinctly* such that they transform from (the padding of) $[\![M_0]\!]_{n_0}$ to $[\![M_1]\!]_{n_0}$ via a sequence of incremental changes. Here, by succinctness, we require that the sizes of the small circuits describing the intermediate circuits are essentially independent of the input length to the Turing machine.

We show that if two Turing machines M_1, M_2 can be proven functionally equivalent in Cook's theory PV, then they are Δ-equivalent Turing machines. Intuitively, we can hope for such an implication, because Cook's propositional translation can be described succinctly in a paper, and thus a large part of the translated propositional proof is "uniform" so that they can be described by small circuits. To implement this intuition, we first abstract the property we need from the propositional translation and hence we separate the propositional translation and our construction of the intermediate circuits.

Our key observation is that, in our construction of the intermediate circuits in Section 2.1, at any point, we only need the following information from the propositional proofs to describe the intermediate circuits.

- Given an index i, we need the i-th "line" θ_i of the propositional proof.

- Given a variable, we need the indices of the lines where the variable appears.

At a high level, each line of the propositional proof corresponds to a gate in the intermediate circuit. Hence, θ_i tells us the functionality of the gates. The topology of the intermediate circuit, which is implemented as the functionally of the Copy and the Proj helper circuits, is determined by the location where a variable appears.

Succinct Description of Cook's Translation. We abstract the above property as the succinct description of propositional proof. Namely, we require that there is a pair of small circuits $(\mathsf{Get}, \mathsf{Where})$ "describing" the propositional logic proof, with the following syntax. 'Get' takes as input an index i, and outputs θ_i encoded as a binary string (Gödel number). 'Where' takes as input the variable encoded as a binary string, and outputs a set of indices of the lines where the variable appears.

We then observe that Cook's translation has succinct descriptions. Namely, we can construct small circuits $\mathsf{Get}, \mathsf{Where}$ whose sizes are independent of the input length of the Turing machine. The idea is as follows. In Cook's translation, to translate a PV proof $(e_1, e_2, \ldots, e_\ell)$ to propositional proof, one first translates each e_i in propositional proof and concatenates them together. The main challenge is that, $\mathsf{Get}(i)$ needs to directly output the i-th line of the proof, without computing the first i lines of proofs. Otherwise, the size of the circuit Get is not polynomial in its input and hence not succinct. To compute $\mathsf{Get}(i)$ succinctly, the key idea is the following way to concatenate two succinct descriptions of propositional proof $\Pi_1 = (\mathsf{Get}_1, \mathsf{Where}_1), \Pi_2 = (\mathsf{Get}_2, \mathsf{Where}_2)$ as a new succinct description $(\mathsf{Get}, \mathsf{Where})$.

- $\mathsf{Get}(i)$: decide whether i-th line is in the first proof Π_1 or the second proof Π_2, by comparing i with the number of lines in Π_1. Then it invokes either Get_1 or Get_2 to obtains the i-th line.

- $\mathsf{Where}(\ulcorner v \urcorner)$: It invokes $\mathsf{Where}_1(\ulcorner v \urcorner)$ and $\mathsf{Where}_2(\ulcorner v \urcorner)$ and outputs the union of them.

In this way, we can concatenate two propositional proofs without explicitly writing down all lines in Π_1 and Π_2, and thus the sizes of $\mathsf{Get}, \mathsf{Where}$ are still small. For a more detailed proof sketch, see Appendix A.

On the Size of Obfuscated Program. In the above discussion, we only require the sizes of the description circuits (Get, Where) to be independent of the input length of the Turing machine. The reader may wonder what the circuit sizes exactly are, since the sizes of the intermediate circuits and thus the size of the obfuscated program will depend on them.

A natural attempt is to use the size of the PV proof to upper bound the sizes of Get, Where. However, given a PV proof of equivalence of two Turing machines with ℓ lines, we can define a Turing machine that runs in time $n^{2^{O(\ell)}}$ somewhere in the proof. This would affect our cryptographic applications, since the Turing machines we are interested in usually have some $\mathsf{poly}(\lambda)$ bits hardwired (e.g. a PRG seed), and hence the proof length ℓ could also be polynomial in λ. Then the obfuscated program size could be $2^{\mathsf{poly}(\lambda)}$. However, we expect that for most of the cases in practice, the time complexity of the Turing machines we are interested in does not reach that high. For instance, for most of the programs in the real world, the time complexity of the program does not grow exponentially in the size of data it hardwired.

Hence, we introduce a more fine-grained complexity measure $\mathrm{Desc}[\cdot]$ to characterize the size of the circuit describing the terms in PV. Intuitively, for any term t in PV, $\mathrm{Desc}[t]$ measures how many terms we need to evaluate before evaluating t. Then we extend the notion $\mathrm{Desc}[\Pi]$ naturally to any proof Π in PV. Then the size of our obfuscated program is polynomial in $\mathrm{Desc}[\Pi]$, where Π is the PV proof of functionality equivalence. We expect the complexity measure $\mathrm{Desc}[\cdot]$ to be polynomials in most of cryptographic applications. For more details, see Section 7.2.

We stress that usually our PV-proof of equivalence can be written in a "highly uniform" way. That is, only a constant number of function symbols are introduced in the proof. In such a case, we have $\mathrm{Desc}[\Pi] = \mathsf{poly}(|\Pi|)$, where $|\Pi|$ is the length of the binary string encoding Π. Namely, the size of the obfuscated Turing machine is $\mathsf{poly}(\lambda, |\Pi|)$.

3 Preliminaries: Part I

A family of elements $\{x_i\}_{i\in I}$ in a set S indexed by a set I, is a map $\phi : I \longrightarrow S$ from I to S, with $\phi(i) = x_i$. A family of circuits $\{C_n\}_{n\in\mathbb{N}}$ in $\mathsf{P/poly}$ is an aforementioned map, with the additional property that there exists a polynomial $s = s(n)$ such that the size of the circuit C_n is bounded by $s(n)$.

For any integer n, we use $[n]$ to denote the set of integers $\{1, 2, \ldots, n\}$.

We say two circuits $C_1(\cdot), C_2(\cdot)$ are *functionality equivalent* or $C_1(x) \equiv C_2(x)$, if for every input x, $C_1(x) = C_2(x)$.

3.1 Propositional Logic Systems

We use extended Frege systems (denote as $\mathcal{EF}$) for propositional logic [Bus98]. Such a system is described by a set of *variables*, a set of *connectives*, and a set of *inference rules*. *Variables* are the most basic elements, usually represented by letters such as x, y, z. *Connectives* are used to connect variables. We only use two connectives $\longrightarrow$ and $\neg$ for "imply" and "negation", respectively. Other connectives such as $\wedge, \vee, \oplus$ for "and", "or", "xor", can be defined using $\longrightarrow$ and $\neg$. We use $\leftrightarrow$ to denote "if and only if", and formally, $a \leftrightarrow b$ is the abbreviation of $(a \longrightarrow b) \wedge (b \longrightarrow a)$. We use F to represent "false", and define "true" T as $\neg\mathsf{F}$.

Formulas are defined inductively: F is a formula; any variable is a formula; if u, v are formulas, then $u \longrightarrow v$, $\neg u$ are formulas. A formula can be treated as a labeled tree, where leaves are labeled with variables, and internal nodes are labeled with connectives. A *subformula* of A is defined as a subtree of A. We define the following complexity measures of formulas. For each formula A, we define the *size* of A as the number of nodes in the tree. We denote $\mathsf{ldp}(A)$ as the *logical depth* of A, which represent the depth of the tree. Formally, it can be inductively defined as follows: $\mathsf{ldp}(\mathsf{F}) = 0$; for any variable a, $\mathsf{ldp}(a) = 0$; for any two formulas u, v, $\mathsf{ldp}(u \longrightarrow v) = 1 + \max(\mathsf{ldp}(u), \mathsf{ldp}(v))$, and $\mathsf{ldp}(\neg u) = 1 + \mathsf{ldp}(u)$.

A *substitution* σ is a map from the set of variables to the set of formulas. If A is a formula, then the result of *applying* σ to A is denoted as $A\sigma$, which is a formula obtained by replacing each occurrence of the variables in A by its image under σ. For example, let $A = p \to (q \to p)$ and let a substitution σ be $p \mapsto a \wedge b, q \mapsto a \vee b$, then $A\sigma = (a \wedge b) \to ((a \vee b) \to (a \wedge b))$.

A *Frege system* is specified by a set of *inference rules*. Each inference rule is defined as $A_1, A_2, \dots, A_k \vdash A_0$, where $A_0, A_1, \dots, A_k$ are formulas. Intuitively, it means that "if $A_1, A_2, \dots, A_k$ are valid, then A_0 is also valid". If $k = 0$, then we say such inference rule is an *axiom*.

In this work, we use the following set of axioms and modus ponens as inference rules for positional logic.

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

In a proposition $p_1, p_2, \dots, p_n \vdash q$, we call $p_1, p_2, \dots, p_n$ as *premise* and call q as *conclusion*. A *derivation* for the proposition $p_1, p_2, \dots, p_n \vdash q$ is a series of formulas $\theta_1, \theta_2, \dots, \theta_\ell$ with $\theta_\ell = q$, and for each $i \in [\ell]$, θ_i is either

- A premise p_j with $j \in [k]$), or
- $A_0\sigma$, where $A_1, A_2, \dots, A_k \vdash A_0$ is an inference rule, and σ is a substitution, and $\{A_1\sigma, A_2\sigma, \dots, A_k\sigma\}$ are a subset of the formulas $\{\theta_1, \theta_2, \dots, \theta_{i-1}\}$.

A *proof* is a derivation with no premise.

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

Gödel Numbering. For any formula f in $\mathcal{EF}$, we can uniquely encode it as a binary string $\{0,1\}^*$ of length $O(|f|)$ in a natural way, by encoding the variables, connectives, and brackets used to express f as binary strings and concatenating them. We denote this encoding function as $\ulcorner \cdot \urcorner$.

What kind of mathematical statements have polynomial size proofs in $\mathcal{EF}$? This problem has been extensively studied in the field of *proof complexity*. In this field, *bounded arithmetics* developed by [Par71, Coo75, Bus86] and many others provides subtheories of Peano arithemtics (the axiomatization of natural numbers), which allows efficient translations of proofs involving arithmetics to propositional logic proofs. Hence, to show the applications (Section 8) of our method, we will use several results in Cook's Theory *PV* [Coo75] and Buss's theory S^1_2 [Bus86].

3.2 Cook's Theory *PV*

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 Cobham's definition of polynomial-time functions [Cob65]. Hence, any polynomial-time function is definable in *PV* [Coo75]. Moreover, common 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].

Formally, Cook's theory PV [Coo75] is defined as follows. PV works on the natural numbers that are represented in the dyadic notation, where any natural number x is uniquely represented as a finite string of integers in $\{1,2\}^*$. Specifically, we represent x as the string $x_n x_{n-1} x_{n-2} \dots x_1 \in \{1,2\}^n$, if $\sum_{i=1}^n x_i 2^i = x$, and use an empty string to represent 0. It's easy to see that such presentation is unique for any natural number. The function $s_i(x) = 2x + i, i = 1,2$ appends i to the string x. Hence, we also denote $s_i(x)$ as $x||i$.

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

Introducing Function Symbols. A *new function symbol* f can be introduced in PV in the following two ways. The first way is to define

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

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

The second way is to use 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*

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

where $\mathbf{y} = (y_1, \dots, y_k)$ is a series of k variables. Then how f is computed for any $x, \mathbf{y}$ is fully specified. To avoid any undecidable issue, PV further requires that the output length of f is bounded by a polynomial. To ensure this, Cook requires that "$|h_i(x, \mathbf{y}, z)| \le |z| + |k_i(x, \mathbf{y})|$" is provable in PV, where $|\cdot|$ is the length of the dyadic presentation. To achieve this, Cook introduced the LESS function, and it is defined with other *initial functions* as follows. $s_i, i = 1, 2$ has no defining functions. 0 is also function symbol with arity 0, and has no defining function.

- TR: $\mathsf{TR}(0) = 0, \mathsf{TR}(x||i) = x, i = 1, 2$. It cuts off the least significant digit in the dyadic notion.
- $\star$: $\star(x, 0) = x, \star(x, y||i) = s_i(x, y), i = 1, 2$. It concatenates the string x and y.
- $\circledast$: $\circledast(x, 0) = x, \circledast(x, y||i) = \star(x, \circledast(x, y)), i = 1, 2$. It concatenates $|y|$ copies of x.
- LESS : $\mathsf{LESS}(x, 0) = x, \mathsf{LESS}(x, y||i) = \mathsf{TR}(\mathsf{LESS}(x, y)), i = 1, 2$. It cuts off the $|y|$ right most digits of x in the dyadic notion. Then we can use $\mathsf{LESS}(x, y) = 0$ to express $|x| \le |y|$.

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

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

- R_1: $t = u \vdash u = t$.
- R_2: $t = u, u = v \vdash t = u$
- R_3: $t_1 = u_1, t_2 = u_2, \dots, t_k = t_k \vdash f(t_1, t_2, \dots, t_k) = f(u_1, u_2, \dots, u_k)$.
- R_4: $t = u \vdash t(v/x) = u(v/x)$. Here, the notation "$t(v/x)$" means replacing each occurrence of the variable x with the term v. "$u(v/x)$" is defined in the same way.

- R_5: $E_1, E_2, \dots, E_6 \vdash f_1(x, y) = f_2(x, y)$, where $E_1, E_2, \dots, E_6$ are the defining equations 1 for f_1, f_2, with the same function symbols g, h_1, h_2.

Propositional Translation. In the same work [Coo75], Cook showed that any proofs in PV can be translated to polynomial size propositional logic proofs. The original theorem statement uses *extended resolutions* logic. Later [CR79] showed that extended resolution and extended Frege system are essentially equivalent in terms of proof size. For simplicity, we use extended Frege system in this work, and state Cook's result in extended Frege system.

Before we formally state the theorem, we first describe how to transform a theorem statement in PV to proposition logic. The idea is to use variables in $\mathcal{EF}$ to present each digit in the dyadic notation. Specifically, let m be an integer. For each term t in PV, let $P_0[t], P_1[t], \dots, P_m[t]$ and $Q_0[t], Q_1[t], \dots, Q_m[t]$ be a set of variables in $\mathcal{EF}$. For each $i \in [m]$, use $Q_i[t]$ to indicate whether t has i-th digit, and use $P_i[t]$ to indicate the i-th digit of t, i.e.

$$Q_i[x] = \begin{cases} \mathsf{T}, \text{if } t \geq 2^{i+1} - 1 \\ \mathsf{F}, \text{otherwise} \end{cases} \quad P_i[x] = \begin{cases} \mathsf{T}, \text{if the } i\text{-th dyadic digit of } t \text{ is } 2 \\ \mathsf{F}, \text{otherwise} \end{cases}$$

For the easy of representation, in this work we use the following notation $\mathsf{Var}_m[t]$ to denote the variables $\{P_i[t], Q_i[t]\}_{i=1}^m$ corresponds to t. For each term t, one can associate it with a proposition formula $\mathsf{Prop}_m[t]$, asserting $\mathsf{Var}_m[t]$ is computed correctly from the variables $\mathsf{Var}_m[p_1], \dots, \mathsf{Var}_m[p_k]$, where $p_1, p_2, \dots, p_k$ are all variables appear in t. For any variable x, $\mathsf{Prop}_m[x]$ is the formula asserting $\mathsf{Var}_m[x]$ is well-formed, i.e. $\neg Q_i[x]$ implies $\neg Q_{i+1}[x]$ for $i \in [m-1]$. The definition of $\mathsf{Prop}_m[t]$ can be inductively defined for any term t. For more details, see [Coo75].

For any integer n, if the computation of all terms in the proof only needs m dyadic digits, then m is called a *bounding value*. For any equation $t = u$, where t and u are both terms. $[\![t = u]\!]_m^n$ is defined as the propositional formula asserting that if the variables in t are all less than n digit, then the value of t and u are equal. For its formal definition, see [Coo75].

Next, we present the theorem statement for Cook's propositional translation.

Theorem 3 (Corollary of ER Simulation Theorem in [Coo75])**.** *For any two terms t and u, and any n and any polynomial bounding value $m = m(n)$, if $\vdash_{PV} t = u$, then $[\![t = u]\!]_m^n$ has polynomial size logic proofs in extended Frege logic.*

The idea of the Corollary 3 is to do an induction on the length of the proofs in PV, and translate each step of the proof in PV to a polynomial size proof in the propositional logic.

Numerals. *Numerals* are a way to express nature numbers in PV. We will use them to express hardwired values in the Turing machine. Formally, a *numeral* is a term of the form $i_1||i_2||\dots||i_k$ (or $s_{i_1}(s_{i_2}(\dots s_{i_k}(0)))$, where $i_1, i_2, \dots, i_k \in \{1, 2\}$.

3.3 Theory PV_1

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 *formulas*, which is either equations, or truth-functional combinations of equations, using "$\wedge, \vee, \neg, \rightarrow, \leftrightarrow$", which express "and", "or", "negation", "imply", and "equivalent".

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

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

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

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

The inference rules of PV_1 are as follows.

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

4 Δ-Equivalent Circuits

In this section, we introduce a new notion of equivalence between circuits: Δ-equivalence. In Section 4.1, we define some terminologies for subcircuits. In Section 4.2, we formally define δ-equivalence and Δ-equivalence for circuits. In Section 4.3, we show that propositional proofs of equivalence for circuits implies our notion of Δ-equivalence.

4.1 Background

Circuits and Subcircuits. A circuit is a acyclic graph, where each node of the graph represents a gate, and each edge of the graph represents a wire. The *topology* of the circuit refers to how the nodes are connected in the graph, ignoring what functions that the nodes compute. For any circuit C, we use $\mathsf{inp}(C)$ and $\mathsf{out}(C)$ to denotes the sets of its input and output wires.

Let C be a circuit. A subcircuit S of C is defined as a subset of the gates in C. Moreover, we define the following terms.

- **Input wire.** We say a wire w is an input wire of S, if w is the output wire of a gate $g \notin S$, meanwhile w is also an input wire of a gate $g' \in S$. We denote the set of input wires as $\mathsf{inp}(S)$.
- **Output wire.** We say a wire w is an output wire of S, if w is an input wire of a gate $g \notin S$, meanwhile w is also the output wire of a gate $g' \in S$. We denote the set of output wires as $\mathsf{out}(S)$.
- **Functionality.** The circuit C naturally induces a circuit $\{0,1\}^{|\mathsf{inp}(S)|} \to \{0,1\}^{|\mathsf{out}(S)|}$ for the subcircuit S. We denote this circuit as C_S.

We remark that in the definition of subcircuit, we do *not* require the subset S is "connected". Instead, we allow the elements in the subcircuit S scatter arbitrarily in the circuit C.

4.2 Definition of Δ-Equivalent Circuits

Before we define the Δ-equivalence, we first introduce the notion of δ-equivalence. Intuitively, we say two circuits are δ-equivalent, if they only differ by two small *functionality equivalent* subcircuits of poly-logarithmic size at the same location. In the following, we define the δ-equivalence via a subcircuit S in order to be general. The reader can keep in mind that the size of S is very small.

Definition 1 (δ-Equivalent via Subcircuit S)**.** *We say that two circuits C_1, C_2 are δ-equivalent via a subcircuit S, if*

- *The topology of C_1 and C_2 are identical.*
- *The corresponding gates of C_1 and C_2 outside of S are identical.*
- *The subcircuits of C_1, C_2 induced by S are identical. Namely,*

$$C_{1S}(x) = C_{2S}(x), \quad \forall x \in \{0,1\}^{|\mathsf{inp}(S)|}.$$

Next, we define the notion of Δ-equivalent circuits. Intuitively, two circuits are Δ-equivalent, if one can be transformed to another by a polynomial number of steps, where each step only modifies a logarithmic size subcircuit, while preserving the functionality of the subcircuit at each step.

Definition 2 (Δ-Equivalent Circuits)**.** *We say that two families of circuits $\{C_n^1\}_n, \{C_n^2\}_n$ are Δ-equivalent, if there exists a polynomial $\ell = \ell(n)$ and a function $B = O(\log n)$ such that, for any positive integer n, there exist $\ell \geq 2$ intermediate circuits $C_1', C_2', \ldots, C_\ell'$ and a series of subcircuits $\{S_i\}_{i\in[\ell-1]}$ of size B with $C_1' = C_n^1$, $C_\ell' = C_n^2$, and for any $i \in [\ell-1]$, C_i and C_{i+1} are δ-equivalent via the subcircuit S_i.*

4.3 Δ-Equivalence from Propositional Proofs

In this section, we will show that the Δ-equivalence in Definition 2 is implied by any polynomial size propositional of functionally equivalence. To formalize this implication, we first define the propositional formula naturally induced by the circuits.

Circuit Induced Propositions. For any Boolean circuit C, we denote the set of propositions naturally induced by C as $\mathsf{Prop}[C]$. Namely, for each wire w in the circuit, we use a variable v_w in the extended Frege system to represent that wire. Then for each gate g with output wire o and input wires l and r, it corresponds to a formula $v_w \leftrightarrow f(v_l, v_r)$, where $f(v_l, v_r)$ is the formula expressing the computation of g. For example, if g is an AND gate, then $f(v_l, v_r) = v_l \wedge v_r$, and the cases for OR gates and NOT gates are similar. $\mathsf{Prop}[C]$ is defined to be the set containing all such formulas for every gates in the circuit C.

For simplicity, we use the notation $o := \mathsf{Prop}[C](x)$ to denote that the input wires of C corresponds to the tuple of variables $x = (x_1, x_2, \dots, x_{|\mathsf{inp}(C)|})$ and the output wire corresponds to the variable o in the extended Frege system (Section 3.1), where $\mathsf{inp}(C)$ is the set of input wires of C, as we define in Section 4.1.

Propositional Proofs of Equivalence. Next, we define the notion of propositional proofs of equivalence. Intuitively, we say two circuit families have propositional proofs of equivalence, if there exists a polynomial size proof of functional equivalence in the propositional logic.

Definition 3 (Propositional Proofs of Equivalence). *We say that two families of circuits $\{C_n^1\}_{n\in\mathbb{N}}$, $\{C_n^2\}_{n\in\mathbb{N}}$ have propositional proofs of equivalence of size s for a function $s(\cdot)$, if for every positive integer n, there exists a derivation of size $s(n)$ of*

$$o_1 := \mathsf{Prop}[C_n^1](x), o_2 := \mathsf{Prop}[C_n^2](x) \vdash o_1 \leftrightarrow o_2.$$

If s is bounded by a polynomial of n, then we say $\{C_n^1\}_{n\in\mathbb{N}}$ and $\{C_n^2\}_{n\in\mathbb{N}}$ have efficient *propositional proofs of equivalence. If s is clear from the context or irrelevant, then we omit s and just say that the two families of circuits have propositional proofs of equivalence.*

Lemma 3 (Δ-Equivalence from Propositional Proofs of Equivalence). *There exists a polynomial time algorithm* Pad *which takes as input a circuit C, and an integer s, it outputs a new circuit C' with following properties.*

- *The functionalities of C' and C are identical.*
- *The size of the circuit C' is bounded by* $\mathsf{poly}(|C|, s)$.
- *If two families of circuits $\{C_n^1\}_{n\in\mathbb{N}}$, $\{C_n^2\}_{n\in\mathbb{N}}$ have efficient propositional proofs of equivalence of size s, then $\{\mathsf{Pad}(C_n^1, s(n))\}_n$ and $\{\mathsf{Pad}(C_n^2, s(n))\}_n$ are Δ-equivalent circuits.*
- *Each gate in C' has at most 2 input wires, and at most 2 output wires.*

Proof. As described in Section 2, we will transform from circuit $\{C_n^1\}_{n\in\mathbb{N}}$ to $\{C_n^2\}_{n\in\mathbb{N}}$ in several steps, where each step changing the functionality of some gates and how wires connect to them. However, our construction of iO later can only support limited number of input and output wires for each gate, and it does not allow the change of topology.

To resolve this issue, we build the following helper circuits to pad the original circuit. The Copy circuit copies an output wire of a gate to multiple wires of the same value. It provides the support for the multi-output wires. The projection circuit "selects" a wire value from multiple wire values. It allows us to change the way how wires are connected, without changing the topology of the circuits.

Copy Circuit Copy. For any integer n, any subset $S \subseteq [n]$, $\mathsf{Copy}_n^S : \{0,1\} \to \{0,1\}^n$ copies the input bit to $|S|$ copies. Namely, $\mathsf{Copy}_n^S(x) = \{x_i\}_{i\in[n]}$, where $x_i = x$ if $i \in S$ and $x_i = 0$ otherwise. We construct

Copy_n^S as a binary tree, where each node corresponds to a gate $\mathsf{Copy}_2^{\{1,2\}}(x) = (x, x)$ if the node is on the root-to-leaf path of an element in S, otherwise it corresponds to a gate that always outputs 0. In this way, for any n, any two subsets $S_1, S_2 \subseteq [n]$, the topology of $\mathsf{Copy}_n^{S_1}$ and $\mathsf{Copy}_n^{S_2}$ are identical. Note that for each gate in Copy_n^S circuit, we only set it to be a "copy" functionality *when necessary*. If the root-to-leaf paths for S doesn't go through a gate, then we set the functionality of such a gate to be always outputting 0.

Projection Circuit Proj**.** The projection circuit $\mathsf{Proj}_n^i : \{0,1\}^n \to \{0,1\}$ takes $x_1, x_2, \ldots, x_n$ as input, and outputs x_i. To build Proj_n^i, we also build a binary tree, where $x_1, x_2, \ldots, x_n$ are on the leaves. Each internal node of the tree computes a projection function $\mathsf{Proj}_2^{b_h}(x_1, x_2) = x_b$, for the h-th bit b_h of i, at the height h.

Next, we describe the padding algorithm Pad in Figure 3. When describing the algorithm, we use the term *add wire w with copy* to represent that we add a new wire w and a copy circuit Copy_n^{ϕ}, where w is treated as the input wire to the copy circuit. We use the term *add wire w with projection* to represent the following procedure.

- Add a new wire w and a projection circuit Proj_n^1, where w is treated as the output wire of the projection circuit.
- For each existing copy circuit, take an unused output wire of it, and replace it with an unused input wire in the projection circuit.

We use the term *connect wire w with wire w′* to represent that we modify the projection circuit associated with w and the copy circuit associated with w' such that the wire value of w equals w'. Note that we change the superscripts S for Copy and i for Proj in this procedure.

Now we examine each property of the new circuit C' we want to prove. For the size of C', the size is clearly polynomial, since we set the parameter ℓ to be a polynomial of $|C|$ and s. For the parameter n, we can set it to be an integer larger than $2(\ell + |C|)$ rounded to a power of 2, which is also a polynomial.

Preserving Functionality. The functionality of C is preserved, since we use the gates $\{g'\}_g$ (g range over all gates in C) to simulate the circuit C. Finally, in the binary tree, the first leaf compute $C(x)$, and the rest of leaves always computes 1, hence the circuit C' computes $C(x) \wedge 1 \wedge \ldots \wedge 1 = C(x)$.

Δ-Equivalence from Propositional Proofs. For any two circuit C_1, C_2 with a propositional proof of equivalence of size s, to show that $C_1' = \mathsf{Pad}(C_1, s)$ and $C_2' = \mathsf{Pad}(C_2, s)$ are Δ-equivalent, we only need to show how to modify $\mathsf{Pad}(C_1, s)$ by changing a $O(\log|C_1| + \log s)$-size subcircuit each time to obtain $\mathsf{Pad}(C_2, s)$. We proceed with the following phases.

Grow C_2. This phases takes $O(|C_2|)$ steps to gradually "grow" C_2 in C_1', by using the dummy gates. Namely, we iterate on the gates of C_2 in the topological order. In each step, we take an unused dummy gate g' in C_1' and change it to compute a gate in C_2. We also connect its input wires to the corresponding output wires of its children. In each step, the subcircuit that we make changes has size $O(\log n)$ since we only need to change a root-to-leaf path in the copy circuits and the projection circuits.

Given a polynomial-size propositional proof of equivalence $(\theta, \theta_2, \ldots, \theta_m)$. In the next "Grow the Extension" phase, we will add all extensions rules in the proof to C_1'. However, if the formulas are too long, then the subcircuit we make change to may be too large. Hence, we need to break θ_i's into constant-size formulas. The idea is to exploit the *extension axiom* in the *extended* Frege system, by introducing new atoms that represent some intermediate subformulas.

Formally, we iterate over $i \in [m]$ and build the following set Q of formulas. Initially, Q is set to be an empty set. For each $i \in [m]$, if θ_i follows from an extension rule $t_i \leftrightarrow A_i$, then we add all subformulas of A_i to Q. Otherwise, we add all subformulas of θ_i to Q. Now, for each non-atomic formula A in Q, we assign it with a new atom v_A.

Padding Algorithm $\mathsf{Pad}(C, s)$

- Initialize a new circuit C' as follows. Create $|\mathsf{inp}(C)|$ input wires. For each input wire x_i, add x_i with copy.
- Iterate on the gate of C in the topological order, for each gate g in C, let h, f be its children.
  - Create a gate g' in C'. Add its output wires with copy. We refer to such gate g' as **regular** gates.
  - Create two input wires l, r to g', and add them with projection.
  - Let g' compute the same functionality as g. Connect the input wires l, r with the output wires of h, g, respectively.
- Let $\ell = 2(|C| + s)$, and round up ℓ to a power of 2. Add ℓ "dummy" arity-2 gates, by adding their output wires with copy, and adding their input wires with projection. Each gate always outputs 1.
- Create a binary tree of gates with ℓ leaf gates in C'. Each internal node of the binary tree computes an $\wedge$ of its two children, and each leaf gate always outputs 1, except the first leaf gate, which computes the identity function. For each leaf gate, we add one input wire with projection.
- Finally, connect the first leaf wire with the output wire of the output gate of C, and set the output wire of C' to be the output wire of the root. Output C' as the augmented circuit.

Figure 3: Description of the padding algorithm Pad.

Grow the Extension. This phase takes $O(s)$ steps to take care of all extension rules used in the proof. We firstly iterate all non-atomic formulas A in the set Q according to their logical depth from 1 to maximum, and "add" v_A to the circuit C_1'. Specifically, if A is of the form $U \to V$, then v_U, v_V has already been added in C_1' since we iterate the formulas based on their logical depth. To add v_A, we take a unused dummy gate g' in C_1', and change it to compute the "$\to$" function, and connect the input wires of g' with the output wires of v_U, v_V. If A is of the form $\neg U$, then we add it similarly.

Next, we also iterate over θ_i in the proof, and "add" all θ_i's that follow from the extension rules to C_1'. Specifically, for each $i \in [m]$, if θ_i follows from an extension rule, i.e. θ_i is $t_i \leftrightarrow A_i$, where A_i is a formula. Then we replace an unused dummy gate g' in C_1' to represent t_i, connect one of its input wire to the output wire of v_{A_i}, and set functionality of g' to be the identity function.

The size of the subcircuit which we makes changes in and the functional equivalence of the subcircuit can be argued in the same way as in the "Grow C_2" phase, since each step modifies a dummy gate.

Grow the Proof. This phase takes $O(s)$ steps to replacing the 1's on the leaves of the binary tree with the formulas θ_i obtained from the inference rules. For each $i = 1, 2, \ldots, m$, if θ_i is obtained from an inference rule $A_1, A_2, \ldots, A_k \vdash A_0$ with a substitution $\sigma : p_1 \mapsto \phi_1, p_2 \mapsto \phi_2, \ldots, p_t \mapsto \phi_t$, where $p_1, p_2, \ldots, p_t$ are all variables used in $A_0, A_1, A_2, \ldots, A_k$, and $\phi_1, \phi_2, \ldots, \phi_t$ are formulas. Then $\theta_i = A_0\sigma$, and $A_1\sigma, \ldots, A_k\sigma$ must have appeared in $\{\theta_j\}_{j<i}$. Hence, by our construction of Q, we have $A_j\sigma \in Q$, for $j = 0, 1, \ldots, k$. Now, we choose an unused leaf gate (which always outputs 1) in the binary tree in C_1', and connect one of its input wire with the output wire of $v_{A_0\sigma}$.

To argue the δ-equivalence for each step. We first explain how we choose the subcircuit S. Firstly, we add all gates "between" $v_{A_j\sigma}$'s and v_{ϕ_j}'s to S. Formally, S contains all v_ϕ, where ϕ is a subformula in one of $A_1\sigma, A_2\sigma, \ldots, A_k\sigma$, and also a superformula for one of $\phi_1, \ldots, \phi_t$. Clearly, the size of S now is the total size of the formulas $A_0, A_1, \ldots, A_k$, which is a constant. Then we add the "connections" between the gates in S via the copy circuits and the projection circuits. This step only blow up the size of S by a $O(\log n)$-factor. Finally, we add all the gates in the root-to-leaf path of $v_{A_1\sigma}, v_{A_2\sigma}, \ldots, v_{A_k\sigma}$. Note that we can do this because $v_{A_1\sigma}, v_{A_2\sigma}, \ldots, v_{A_k\sigma}$ are already "selected" by one of the leaves previously, since they have already appeared in the first $(i-1)$ θ_j's. Since the root to leaf path contains $O(\log \ell)$ gates, after this, S still contains logarithmic gates.

To prove that the change we make preserves functionality of the subcircuit S, the initial observation is that only the output wire of the root is affected by our change. Hence, if we let $\sigma' : p_1 \mapsto v_{\phi_1}, p_2 \mapsto v_{\phi_2}, \ldots, p_t \mapsto v_{\phi_t}$, then before the change, the output wire at the root gate computes $A_1\sigma' \wedge A_2\sigma' \ldots A_k\sigma' \wedge t$, where t is a term that contains all other inputs to the subcircuit S. After we make the change, it computes $A_1\sigma' \wedge A_2\sigma' \ldots A_k\sigma' \wedge A_0\sigma' \wedge t$. Since $A_1, A_2, \ldots A_k \vdash A_0$ is an inferece rule, the formula $A_1\sigma' \wedge A_2\sigma' \ldots A_k\sigma' \rightarrow A_0\sigma'$ is a tautology. Hence, $A_1\sigma' \wedge A_2\sigma' \ldots A_k\sigma' \wedge A_0\sigma' \wedge t = A_1\sigma' \wedge A_2\sigma' \ldots A_k\sigma \wedge t$, and thus the functionality of the subcircuit is preserved.

Change the Output. In this phase we replace the first leaf, which is originally set to select the output of C_1 in C_1', with a gate that selects the output of C_2 computed by the dummy gates. The δ-equivalence of this step can be argued in a similar way as the previous phase.

Shrink the Proof. This phase undo the changes we make in "Grow the Proof" phase. Namely, we replace each leaf gate of the binary tree back to a gate that always output 1 in the reverse order of the "Grow the Proof" phase. We can do this because any argument we made about the equivalence of the subcircuit in "Grow the Proof" phase does not involve the first leaf, which is now switched to output $C_2(x)$. Hence, we can replace 1 back in the leaves.

Shrink the Extension. This phase undo the changes we make in the "Grow the Extension" phase. Namely, we remove the gates that introduce the extensions, in the reverse order that they're added.

Move C_2 to the Place of C_1. The start of this phase is almost C_2', except that C_2 is computed by the $|C_1|$-th regular gate to the $|C_1| + |C_2|$-th regular gate added in C_1' (See the definition of Regular gates in Figure 3), while C_2 is computed by the first $|C_2|$-gates in C_2'. Hence, in this phase we move the C_2 computed by the dummy gates to the first $|C_2|$ gates. Finally, we obtain C_2'. □

5 Preliminaries: Part II

In this section, we define several cryptographic building blocks that we shall use in the next section.

5.1 Learning with Errors

We start by recalling the learning with errors problem below.

Definition 4 (Learning with Error Assumption)**.** *For any positive integers n, q, any $\mathbf{s} \in \mathbb{Z}^n$, and any error distribution χ over $\mathbb{Z}$, the LWE (Learning with Error) distribution $A_{\mathbf{s},\chi}$ is defined by uniformly sampling a vector $\mathbf{a}$, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $e \leftarrow \chi$.*

The $\mathsf{LWE}_{n,q,\chi}$ assumption states that no non uniform PPT adversary can distinguish, with non-negligible probability, between (i) the distribution $A_{\mathbf{s},\chi}$ for a single $\mathbf{s} \leftarrow \mathbb{Z}_q^n$; and (ii) the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

A standard instantiation of LWE chooses χ as *discrete Gaussian* distribution over $\mathbb{Z}$ with parameters $r = 2\sqrt{n}$. For this parameterization, LWE is at least as hard as quantumly approximating some "short vector" problem on n-dimensional lattices in the worst case to $\tilde{O}(q\sqrt{n})$ factors [Reg09, PRS17]. There are also classical reductions for different parameterizations [Pei09, BLP+13].

5.2 Pseudorandom Generators

A pseudorandom generator (PRG) [ILL89] stretches random strings into strings that look uniformly random to a computationally bounded adversary.

Definition 5 (Pseudorandom Generator). *A function* $\mathsf{PRG} : \{0,1\}^\lambda \longrightarrow \{0,1\}^\theta$ *is pseudorandom if* $\theta > \lambda$ *and if the following distributions are computationally indistinguishable:*

$$r \approx \mathsf{PRG}(\mathsf{seed})$$

where $r \leftarrow\!\!\$\ \{0,1\}^m$ *and* $\mathsf{seed} \leftarrow\!\!\$\ \{0,1\}^\lambda$.

5.3 Puncturable Pseudorandom Functions

We define the notion of puncturable pseudorandom functions below.

Definition 6. *A pseudorandom function of the form* $\mathsf{PRF}_{punc}(K,\cdot)$ *is said to be a* μ*-secure puncturable PRF if there exists a PPT algorithm* $\mathsf{PRFPunc}$ *that satisfies the following properties:*

- **Functionality preserved under puncturing.** $\mathsf{PRFPunc}$ *takes as input a PRF key* K *and an input* x *and outputs* $K\backslash\{x\}$ *such that for all* $x' \neq x$, $\mathsf{PRF}_{punc}(K\backslash\{x\}, x') = \mathsf{PRF}_{punc}(K, x')$.

- **Pseudorandom at punctured points.** *For every PPT adversary* $(\mathcal{A}_1, \mathcal{A}_2)$ *such that* $\mathcal{A}_1(1^\lambda)$ *outputs an input* x*, consider an experiment where* $K \xleftarrow{\$} \{0,1\}^\lambda$ *and* $K\backslash\{x\} \leftarrow \mathsf{PRFPunc}(K, x)$*. Then for all sufficiently large* $\lambda \in \mathbb{N}$,

  $$\left|\Pr[\mathcal{A}_2(K\backslash\{x\}, x, \mathsf{PRF}_{punc}(K,x)) = 1] - \Pr[\mathcal{A}_2(K\backslash\{x\}, x, U_{\chi(\lambda)}) = 1]\right| \leq \mu(\lambda)$$

  where $U_{\chi(\lambda)}$ *is a string drawn uniformly at random from* $\{0,1\}^{\chi(\lambda)}$.

If μ *is negligible, we refer to* PRF_{punc} *as a secure puncturable PRF.*

As observed by [BW13, BGI14, KPTZ13], the GGM construction [GGM84] of PRFs from one-way functions yields puncturable PRFs.

Theorem 4 ([GGM84, BW13, BGI14, KPTZ13]). *If* $\frac{\mu}{\mathsf{poly}(\lambda)}$*-secure one-way functions exist, for some fixed polynomial* $\mathsf{poly}(\lambda)$*, then there exist* μ*-secure puncturable pseudorandom functions.*

5.4 Homomorphic Encryption

We recall the notion of homomorphic encryption [Gen09].

Definition 7 (Homomorphic Encryption). *A homomorphic encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *for a function family* $\mathcal{F}$ *is composed of the following* PPT *algorithms.*

- *The key generation algorithm* KGen *takes as input the security parameter* λ *and returns a key pair* $(\mathsf{sk}, \mathsf{pk})$.

- *The encryption algorithm* Enc *takes as input a public key* pk *and a message* m *and returns a ciphertext* c.

- *The evaluation algorithm* Eval *takes as input a public key* pk*, a ciphertext* c *and a function* f *and returns an evaluated ciphertext* c.

– *The decryption algorithm takes as input a secret key* sk *and a ciphertext c and returns a message m.*

The homomorphic encryption scheme is correct if for all functions $f \in \mathcal{F}$, all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KGen}(1^\lambda)$, all message m, all ciphertexts $c \in \mathsf{Enc}(\mathsf{pk}, m)$, then $\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, c, f)) = f(m)$. Moreover, we say an FHE scheme is compact, if the size of the homomorphically evaluated ciphertext $\mathsf{Eval}(\mathsf{pk}, c, f)$ does not depends on the circuit being evaluated, and is bounded by $\mathsf{poly}(\lambda, |f(m)|)$.

If $\mathcal{F}$ is the family of all polynomially computable functions we say that the encryption scheme is *fully* homomorphic (FHE). Furthermore, if the maximum size of the computable circuit is a priori bounded, we say that the FHE is *leveled*. We say that a scheme is secure if it satisfies the standard notion of CPA-security [GM82].

5.5 Indistinguishability Obfuscation for Circuits

We define the notion of indistinguishability obfuscation (iO) for circuits [BGI+01, GGH+13] below.

Definition 8 (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform PPT algorithm* iO *is called an ε-secure indistinguishability obfuscator for a circuit family $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where $\mathcal{C}_\lambda$ consists of circuits C of the form $C : \{0,1\}^\ell \to \{0,1\}$, if the following holds:*

– **Completeness:** *For every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, every input $x \in \{0,1\}^\ell$, where $\ell = \ell(\lambda)$ is the input length of C, we have that*

$$\Pr\left[C'(x) = C(x) \;:\; C' \leftarrow \mathsf{iO}(1^\lambda, C)\right] = 1$$

– **ε-Indistinguishability:** *For any PPT distinguisher D, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: for all sufficiently large $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$ such that $C_0(x) = C_1(x)$ for all inputs $x \in \{0,1\}^\ell$, where $\ell = \ell(\lambda)$ is the input length of C_0, C_1, we have:*

$$\left|\Pr\left[D(\lambda, \mathsf{iO}(1^\lambda, C_0)) = 1\right] - \Pr\left[D(1^\lambda, \mathsf{iO}(1^\lambda, C_1)) = 1\right]\right| \le \varepsilon$$

If ε is negligible in λ then we refer to iO *as a secure indistinguishability obfuscator.*

5.6 Somewhere Extractable Hash

We recall the notion of somewhere extractable hash functions that are defined similarly to somewhere statistical binding hash functions [HW15], except that we explicitly require an extraction property.

A somewhere extractable hash has a key with two computationally indistinguishable modes: (i) In the *normal mode*, the key is *uniformly random*; and (ii) in the *trapdoor mode*, the key is generated according to a subset S denoting the coordinates of the hashed message. Furthermore, we require the following properties.

– **Efficiency:** We require that the size of the key and hash value roughly grow with $|S|$.

– **Extraction:** The trapdoor mode hash key is associated with a trapdoor td, such that given the trapdoor, one can extract the message on coordinates in S. Note that the extraction implies the statistical binding property for the coordinates in S.

– **Local Opening:** We allow the prover to generate a *local opening* for any single coordinate of the message. The local opening needs to have a small size, which only grows poly-logarithmically with the total length of the message. Moreover, we require that the value from the local opening should be consistent with the extracted value.

More formally, a somewhere extractable hash scheme is a tuple of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$ described below.

$\mathsf{Gen}(1^\lambda, 1^N, 1^{|S|})$**:** On input a security parameter, the length of the message N, and the size of a subset $S \subseteq [N]$, the "normal mode" key generation algorithm outputs a *uniformly random* hash key K.

$\mathsf{TGen}(1^\lambda, 1^N, S)$**:** On input a security parameter, the length of the message N, an extraction subset $S \subseteq [N]$, the "trapdoor mode" key generation algorithm outputs a hash key K^* and a trapdoor td.

$\mathsf{Hash}(K, \mathbf{m} \in \{0,1\}^N)$**:** On input the hash key K, a vector $\mathbf{m} = (m_1, m_2, \ldots, m_N) \in \{0,1\}^N$, it outputs a hash value c.

$\mathsf{Open}(K, \mathbf{m}, i)$**:** On input the hash key K, a vector $\mathbf{m} = (m_1, m_2, \ldots, m_N) \in \{0,1\}^N$, an index $i \in [N]$, the opening algorithm outputs a *local opening* π_i to m_i.

$\mathsf{Verify}(K, c, m_i, i, \pi_i)$**:** On input the hash key K, a hash value c, a bit $m_i \in \{0,1\}$, and a local opening π_i, the verification algorithm decides to accept (output 1) or reject (output 0) the local opening.

$\mathsf{Ext}(c, \mathsf{td})$**:** On input a hash value c, and the trapdoor td generated by the trapdoor key generation algorithm TGen with respect to the subset S, the extraction algorithm outputs an extraction string m_S^* on the subset S.

Furthermore, we require the hash scheme to satisfy the following properties.

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

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

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

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

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

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

Opening Completeness. For any hash key K, any message $\mathbf{m} = (m_1, \ldots, m_N) \in \{0,1\}^N$, any randomness r, and any index $i \in [N]$, we have

$$\Pr\left[c \leftarrow \mathsf{Hash}(K, \mathbf{m}; r), \pi_i \leftarrow \mathsf{Open}(K, \mathbf{m}, i, r) : \mathsf{Verify}(K, c, m_i, i, \pi_i) = 1\right] = 1.$$

Extraction Correctness. For any subset $S \subseteq [N]$, any trapdoor key $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, any hash c, any index $i \in S$, any bit $m_{i^*} \in \{0,1\}$, and any proof π_{i^*}, we have

$$\Pr\left[\mathsf{Verify}(K, c, m_{i^*}, i^*, \pi_{i^*}) = 1 \Rightarrow \mathsf{Ext}(c, \mathsf{td})|_{i^*} = m_{i^*}\right] = 1.$$

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

Theorem 5. *There exists a construction of somewhere extractable hash from LWE.*

The proof of the above theorem is implicit in [HW15].

Remark 1. *We observe that the construction is a Merkle-tree based construction, and thus the input length N can in fact be* unbounded. *That is, the running time of* $\mathsf{Gen}, \mathsf{TGen}$ *grows poly-logarithmically in N. If we set N to be a slightly super-polynomial in λ, then the hash key can support any polynomial input length. Hence, in our construction, we suppress the input length N in* Gen *and* TGen.

5.7 SNARGs for Batch-$\mathcal{NP}$

Let SAT be the following language

$$\mathsf{SAT} = \{(C, x) \mid \exists\, w \text{ s.t. } C(x, w) = 1\},$$

where $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is a Boolean function, and $x \in \{0,1\}^n$ is an instance.

A SNARGs for batch-$\mathcal{NP}$ (or non-interactive batch argument) is a protocol between a prover and a verifier. The prover and the verifier first agree on a circuit C, and a series of T instances $x_1, x_2, \ldots, x_T$. Then the prover sends a single message to the verifier and tries to convince the verifier that (C, x_1), (C, x_2), $\ldots, (C, x_T) \in \mathsf{SAT}$.

More formally, such a protocol is specified by a tuple of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{P}, \mathsf{V})$ that work as follows.

- $\mathsf{Gen}(1^\lambda, 1^T, 1^{|C|})$: On input a security parameter λ, the number of instances T, and the size of the circuit C, the CRS generation algorithm outputs crs.
- $\mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$: On input a security parameter λ, the number of instances T, the size of the circuit C and an index i^*, the trapdoor CRS generation algorithm outputs crs*.
- $\mathsf{P}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \omega_1, \omega_2, \ldots, \omega_T)$: On input crs, a circuit C, and T instances $x_1, x_2, \ldots, x_T$ and their corresponding witnesses $\omega_1, \omega_2, \ldots, \omega_T$, the prover algorithm outputs a proof π.
- $\mathsf{V}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \pi)$: On input crs, a circuit C, a series of instances $x_1, x_2, \ldots, x_T$, and a proof π, the verifier algorithm decides to accept (output 1) or reject (output 0).

Furthermore, we require the aforementioned algorithms to satisfy the following properties.

- **Succinct Communication.** The size of π is bounded by $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Compact CRS.** The size of crs is bounded by $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Succinct Verification.** The verification algorithm runs in time $\mathsf{poly}(\lambda, T, n) + \mathsf{poly}(\lambda, \log T, |C|)$. Moreover, it can be split into the following two parts[8]:
  - **Pre-processing:** There exists a deterministic algorithm $\mathsf{PreVerify}(\mathsf{crs}, x_1, x_2, \ldots, x_T)$ that takes as input the CRS, and T instances $x_1, x_2, \ldots, x_T$, and outputs a short sketch c, where $|c| = \mathsf{poly}(\lambda, \log T, |x_1|)$.
  - **Online Verification:** There exists an online verification algorithm $\mathsf{OnlineVerify}(\mathsf{crs}, c, C, \pi)$ that takes as input the sketch c, a circuit C, and a proof π, and outputs 1 (accepts) or 0 (rejects). Furthermore, the running time of the online verification algorithm is $\mathsf{poly}(\lambda, |C|, |c|, |\pi|) = \mathsf{poly}(\lambda, \log T, |C|)$.
- **CRS Indistinguishability.** For any non-uniform PPT adversary $\mathcal{A}$, and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Big|\Pr\Big[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T) : \mathcal{A}(\mathsf{crs}) = 1\Big] - \Pr\Big[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i^*) : \mathcal{A}(\mathsf{crs}^*) = 1\Big]\Big| \le \nu(\lambda).$$

[8]We note this is a stronger property than previously considered. However, it is natural, and our construction achieves this property.

– **Completeness.** For any circuit C, any T instances $x_1, \dots, x_T$ such that $(C, x_1), (C, x_2), \dots, (C, x_T) \in$ SAT and witnesses $\omega_1, \omega_2, \dots, \omega_T$ for $(C, x_1), (C, x_2), \dots, (C, x_T)$, we have

$$\Pr[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T, 1^{|C|}), \pi \leftarrow \mathsf{P}(\mathsf{crs}, C, x_1, x_2, \dots, x_T, \omega_1, \omega_2, \dots, \omega_T) : \\ \mathsf{V}(\mathsf{crs}, C, x_1, x_2, \dots, x_T, \pi) = 1] = 1.$$

– **Semi-Adaptive Somewhere Soundness.** For any non-uniform PPT adversary $\mathcal{A}$, and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda) \leq \nu(\lambda)$, where $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda)$ is defined as

$$\Pr\Big[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i^*), (C, x_1, x_2, \dots, x_T, \Pi) \leftarrow \mathcal{A}(\mathsf{crs}^*) : \\ i^* \in [T] \wedge (C, x_{i^*}) \notin \mathsf{SAT} \wedge \mathsf{V}(\mathsf{crs}, C, x_1, x_2, \dots, x_T, \Pi) = 1\Big].$$

Index Language. Let the index language be the following language

$$\mathcal{L}^{\mathsf{idx}} = \{(C, i) \mid \exists\, w \text{ s.t. } C(i, w) = 1\},$$

where C is a Boolean function, and i is an index.

SNARGs for batch-index language is a special case of SNARGs for batch-$\mathcal{NP}$ when the instances $x_1, x_2, \dots, x_T$ are simply the indices $1, 2, \dots, T$. We therefore omit $x_1, x_2, \dots, x_T$ as inputs to the prover and the verifier algorithms (and also as an output of the adversary $\mathcal{A}$ describing the semi-adaptive somewhere soundness property). Furthermore, since the verifier does not need to read the instances, there is no pre-processing in this case, and the succinct verification property requires the verifier to run in time $\mathrm{poly}(\lambda, \log T, |C|)$.

Theorem 6 ([CJJ21]). *Assuming LWE, there exists a SNARGs for batch-index.*

Somewhere Statistical Soundness. To use such SNARGs in our constructions, we require the following stronger soundness property.

Definition 9 (Semi-Adaptive Somewhere Statistical Soundness). *We say a SNARG for batch-index satisfies semi-adaptive somewhere statistical soundness, if for any* unbounded *adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda)$ defined above is negligible.*

We observe that the construction in [CJJ21] can be easily modified to obtain the following result:

Theorem 7. *Assuming LWE, there exists a SNARGs for batch-index. Furthermore, the construction satisfies the semi-adaptive somewhere statistical soundness.*

The above theorem can be obtained as follows. Choudhuri et al. [CJJ21] constructed SNARGs for index language language, with (*computational*) semi-adaptive somewhere soundness defined above. Their construction relies on correlation intractable hash functions (CIH) for efficiently verifiable product relations [HLR21]. We observe that if the underlying CIH is *statistically* secure, then we can achieve semi-adaptive somewhere *statistical* soundness. Furthermore, we observe that the construction of CIH in [Theorem 3.7, [HLR21]] makes generic use of the underlying CIH for efficiently searchable relations by [PS19], and [PS19] also constructed a CIH with *somewhere statistical* security [Definition 2.5,[PS19]]. Hence, if we apply the generic construction in [HLR21] to the *somewhere statistical* CIH [Construction 3.1, [CJJ21]], then we obtain a *somewhere statistical* CIH for efficiently verifiable product relations. Plugging it in the construction of SNARGs for batch-index language in [CJJ21], we obtain Theorem 7.

6 iO for Δ-Equivalent Circuits

In this section, we define ΔiO, which means iO for Δ-equivalent circuits. Then we construct ΔiO. Our construction needs a new notion of somewhere statistical hash with a consistency proof, which is introduced and constructed in Section 6.1. Then in Section 6.2, we construct ΔiO.

We start by defining the notion of ΔiO.

Definition 10 (ΔiO)**.** *Let C_λ be a family of circuits C of the form $C : \{0,1\}^\ell \to \{0,1\}$. A uniform PPT algorithm* iO *is a secure Δ-indistinguishability obfuscator (ΔiO) for a class of circuits $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ if it satisfies:*

- *Completeness for any circuit in $\{C_\lambda\}_{\lambda\in\mathbb{N}}$,*
- *Indistinguishability for any pair of Δ-equivalent circuit families in $\{C_\lambda\}_{\lambda\in\mathbb{N}}$.*

6.1 Somewhere Extractable Hash with Consistency Proof

We first extend the notion of somewhere extractable hash (SEH) functions (see Section 5.6) to SEH for a family of elements in some alphabet Σ. Then we introduce the notion of consistency proofs. Namely, SEH with consistency proofs is a SNARGs for the following promise language. For completeness, it allows any prover with two "consistent" families of elements to provide a succinctly verifiable proof for their hash values. For soundness, given two hash values, if the *elements extracted by SEH* from the hash values is not "consistent", then any proof will be rejected for *unbunded prover*. Here, by "consistent", we mean one family of elements is a subset of the other.

Recall that, a family of elements $\{v_i\}_{i\in I}$ in Σ with the index set I, is defined as a map $\phi : I \to \Sigma$ where $v_i = \phi(i)$.

SEH for a Family of Elements. Let Σ be an alphabet. For an integer N, let $W \subseteq [N]$ be a subset of the integers in $[N]$, and let $\{v_w\}_{w\in W}$ be a family of elements indexed by W, where each v_w is an element in Σ. Given a somewhere extractable hash $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$, and a hash key K, we define $\mathsf{Hash}(K, \{v_w\}_{w\in W})$ as $\mathsf{Hash}(K, \{V_i\}_{i\in[N]})$, where $V_i = v_w$ if $i \in W$, and $V_i = \perp$ otherwise, and $\mathsf{Hash}(K, \{V_i\}_{i\in[N]})$ is computed by firstly encoding the elements in $\Sigma \cup \{\perp\}$ as binary strings and then applying the SEH for binary strings in Section 5.6. The extraction Ext is defined accordingly as follows, for each subset $S \subseteq [N]$ and a trapdoor td for S, and a hash value h, $\mathsf{Ext}(h, \mathsf{td})$ output a family $\{V_i^*\}_{i\in S}$ of elements in $\Sigma \cup \{\perp\}$, where V_i^* is $\perp$ if i is not in the index set W.

SEH with Consistency Proof. A somewhere extractable hash with consistency proof is a somewhere extractable hash for families with the following additional algorithms (P, V).

- $\mathsf{P}(K, \{v_w^1\}_{w\in W_1}, \{v_w^2\}_{w\in W_2})$: The prover takes as input a hash key K, two families of elements $\{v_w^1\}_{w\in W_1}$, $\{v_w^2\}_{w\in W_2}$ indexed by the subsets $W_1, W_2 \subseteq [N]$ in some universe $[N]$. It outputs a proof π, proving that $\{v_w^2\}_{w\in W_2} \subseteq \{v_w^1\}_{w\in W_1}$. Namely, $W_2 \subseteq W_1$, and for each $w \in W_2$, $v_w^1 = v_w^2$.
- $\mathsf{V}(K, H_1, H_2, \pi)$: The verifier takes two hash values H_1, H_2 and a proof π. Then it decides to accept or reject.

Furthermore, we require the following properties.

- **Succinct Verification.** The verification circuit V has size $\mathrm{poly}(\lambda, |S|, \log N)$.

- **Completeness.** If $\{v^1_w\}_{w\in W_1}$ and $\{v^2_w\}_{w\in W_2}$ satisfies that $W_2 \subseteq W_1$ and $v^1_w = v^2_w, \forall w \in W_2$, then the honestly generated proof is accepted. Namely,

$$\Pr\Big[H_1 = \mathsf{Hash}(K, \{v^1_w\}_{w\in W_1}), H_2 = \mathsf{Hash}(K, \{v^2_w\}_{w\in W_2}),$$
$$\pi \leftarrow \mathsf{P}(K, \{v^1_w\}_{w\in W_1}, \{v^2_w\}_{w\in W_2}) : \mathsf{V}(K, H_1, H_2, \pi) = 1\Big] = 1.$$

- **Somewhere Statistical Soundness.** If the hash key is in the trapdoor mode with subset $S \subseteq [N]$ and trapdoor td, for any two hash values H_1, H_2, we denote $\{V^{b*}_i\}_{i\in S} \leftarrow \mathsf{Ext}(H_b, \mathsf{td})$ for $b = 1, 2$, and there exists an $i \in S$ such that $V^{2*}_i \neq \perp$ and $V^{1*}_i \neq V^{2*}_i$ (which means $\{V^{2*}_i\}_i \subseteq \{V^{1*}_i\}_i$ doesn't hold), then any proofs should be rejected with overwhelming probability. Namely, there exits a negligible function such that

$$\Pr_K\Big[\exists \pi, H_1, H_2 : \{V^{b*}_i\}_{i\in S} \leftarrow \mathsf{Ext}(H_b, \mathsf{td}), b = 1, 2$$
$$\exists i \in S : V^{2*}_i \neq \perp \wedge V^{1*}_i \neq V^{2*}_i \wedge \mathsf{V}(K, H_1, H_2, \pi) = 1\Big] \leq \mathsf{negl}(\lambda).$$

Lemma 4. *Assuming polynomial hardness of learning with errors, there exists a somewhere extractable hash with consistency proof.*

Ingredient. Before we present our construction, we firstly list our ingredients.

- A somewhere extractable hash $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.TGen}, \mathsf{H.Hash}, \mathsf{H.Open}, \mathsf{H.Verify}, \mathsf{H.Ext})$.
- SNARGs for Batch-Index language $\Pi = (\Pi.\mathsf{Gen}, \Pi.\mathsf{P}, \Pi.\mathsf{V})$. Recall that, such SNARGs satisfy semi-adaptive somewhere soundness 5.7.

Construction. The construction for the key generation, hash algorithm, local opening are almost the same as the underlying somwehere extractable hash, except that we additionally generate a key for the SNARGs for batch-index in the key generation. Namely, the key for the new somewhere extractable hash we're constructing becomes $(\mathsf{H}.K, \Pi.\mathsf{crs})$, where $\mathsf{H}.K$ is the hash key for the underlying somewhere extractable hash H and $\Pi.\mathsf{crs}$ is the CRS for the underlying SNARGs Π for batch-index language.

The trapdoor mode hash key K^* for a subset S of indices is generated by firstly generating a trapdoor mode key $\mathsf{H}.K^*$ for S in the underlying SEH, then we generate a trapdoor CRS $\Pi.\mathsf{crs}^*$ for the underlying SNARGs for Batch-Index language with semi-adaptive somewhere soundness for the instances in S. Finally, we set $K^* = (\mathsf{H}.K^*, \Pi.\mathsf{crs}^*)$ as the trapdoor mode hash key. Then we describe the construction in Figure 4.

Security. The completeness of the construction follows from the completeness of the underlying protocols. Next, we prove the somewhere statistical soundness.

Lemma 5 (Succinct Verification)**.** *The construction in Figure 4 satisfies the succinct verification property.*

Proof. The verification circuit takes input K, H_1, H_2, π. $|K| = \mathsf{poly}(\lambda, |S|, \log N)$ by the key succinctness of SEH (Section 5.6) and the compact CRS property of SNARGs for Batch-Index (Section 5.7). $|H_b| = \mathsf{poly}(\lambda, |S|, \log N)$ for $b = 1, 2$ by the hash value succinctness of SEH. The size of π and $C_{[\mathsf{H}.K, H_1, H_2]}$ is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$ from the succinct verification property of SNARGs for Batch-Index. □

Lemma 6. *The construction in Figure 4 satisfies the somewhere statistical soundness.*

Somewhere extractable hash with consistency proof

- **Prover** $\mathsf{P}(K, \{v^1_w\}_{w \in W_1}, \{v^2_w\}_{w \in W_2})$**:** Parse the key $K = (\mathsf{H}.K, \Pi.\mathsf{crs})$.
  - For $b \in \{1, 2\}$, compute the hash values $H_i = \mathsf{H}.\mathsf{Hash}(\mathsf{H}.K, \{v^b_w\}_{w \in W_b})$, and prepare the local openings for all indices,
  $$\rho^b_i \leftarrow \mathsf{H}.\mathsf{Open}(\mathsf{H}.K, \{v^v_w\}_{w \in W_b}, i), \quad \forall i \in [N].$$
  - Let $C_{[\mathsf{H}.K, H_1, H_2]}(\cdot, \cdot)$ be the circuit in Figure 5, and let $\{(\rho^1_i, \rho^2_i, V^1_i, V^2_i)\}_{i \in [N]}$ be the witness for the Batch-Index language for the instances $\{(C, 1), (C, 2), \ldots, (C, N)\}$, where $V^b_i = \perp$ for $i \notin W_b$ and $V^b_i = v^b_i$ otherwise. Generate a proof π using SNARGs for batch-index as follows.
  $$\Pi.\pi \leftarrow \Pi.\mathsf{P}(\Pi.\mathsf{crs}, C_{[\mathsf{H}.K, H_1, H_2]}, \{(\rho^1_i, \rho^2_i, V^1_i, V^2_i)\}_{i \in [N]}).$$
  - Output $\Pi.\pi$
- **Verifier** $\mathsf{V}(K, H_1, H_2, \pi)$**:** Parse the key $K = (\mathsf{H}.K, \Pi.\mathsf{crs})$, and parse the proof $\pi = \Pi.\pi$. Then verify the proof as
$$\Pi.\mathsf{Verify}(\Pi.\mathsf{crs}, C_{[\mathsf{H}.K, H_1, H_2]}, \pi) \overset{?}{=} 1.$$

Figure 4: Description of the somewhere extractable hash with consistency proof.

Circuit $C_{[\mathsf{H}.K, H_1, H_2]}(i, (\rho^1, \rho^2, V^1, V^2))$

- **Hardwire**: a hash key $\mathsf{H}.K$ and two hash values H_1, H_2.
- **Input**: an index i, and the local openings ρ^1 and ρ^2, and two elements $V^1, V^2 \in \Sigma \cup \{\perp\}$.
- Verify if V^1, V^2 are the i-th elements of H_1, H_2 with local openings ρ^1, ρ^2, respectively. Namely, it checks whether
$$\mathsf{H}.\mathsf{Verify}(\mathsf{H}.K, H_b, V^b, i, \rho^b) \overset{?}{=} 1, \quad \forall b \in \{1, 2\}.$$
- Then it checks whether $(V^2 \neq \perp) \rightarrow (V^1 = V^2)$ holds. If both checking passes, then output 1. Otherwise output 0.

Figure 5: Description of the circuit $C_{[\mathsf{H}.K, H_1, H_2]}(\cdot, \cdot)$.

Proof. By the semi-adaptive somewhere soundness of the underlying SNARGs for Batch-Index, we have that, except for negligible probability over $\Pi.\mathsf{crs}^*$, $(C_{[\mathsf{H}.K, H_1, H_2]}, i) \in \mathcal{L}^{\mathsf{idx}}$ for every $i \in S$ (See Section 5.7). By the construction of $C_{[\mathsf{H}.K, H_1, H_2]}$, for every $i \in S$, there exists local openings ρ^1_i, ρ^2_i and elements $V^1_i, V^2_i \in \Sigma \cup \{\perp\}$ with $(V^2_i \neq \perp) \rightarrow (V^1_i = V^2_i)$. By the extraction correctness, $\mathsf{Ext}(H_b, \mathsf{td})|_i = V^b_i$ for $b \in \{1, 2\}$ and

$i \in S$. However, this implies that the predicate $\exists i \in S : V_i^2 \neq \perp \wedge V_i^2 \neq V_i^1$ is false. Hence, we finish the proof. □

6.2 Construction of ΔiO

In this section, we proceed to construct ΔiO.

Ingredients. Before we present our construction, we list the necessary building blocks as follows.

- Polynomial-secure leveled fully homomorphic encryption (FHE) $\mathsf{FHE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$.
- $2^{\mathsf{poly}(\lambda,|S|)}$-secure pseudorandom generator $\mathsf{PRG} : \{0,1\}^{\lambda_{\mathsf{PRG}}} \to \{0,1\}^{\theta_{\mathsf{PRG}}}$, with $\lambda_{\mathsf{PRG}} = \mathsf{poly}(\lambda, |S|)$.
- $2^{\mathsf{poly}(\lambda,|S|)}$-secure puncturable PRF $\mathsf{PRF}_{punc}(\cdot,\cdot)$ with output length λ_{PRG}.
- Polynomial-secure Somewhere extractable hash with consistent proof $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext}, \mathsf{P}, \mathsf{V})$.
- $2^{\mathsf{poly}(\lambda,|S|)}$-secure indistinguishable obfuscation scheme iO.
- A circuit Gate in Figure 7 emulating the computation at each gate for the input circuit.
- A circuit $\mathsf{Shrink}_{[H]}(\cdot,\cdot)$ in Figure 8 that decrypts the symmetric key encryption inside the FHE, and thus shrink the size of the somewhere extractable hash value H.

Firstly, we construct an algorithm δiO for δ-equivalent circuits in Figure 6. In Lemma 7, we prove the security that δiO provides for δ-equivalent circuits. For the ease of presentation, we present an evaluation algorithm separately for δiO in Figure 9. Then we apply the padding algorithm in Lemma 3 to build an obfuscator ΔiO in Figure 10. By combining the δiO with Lemma 3, we show that ΔiO is an obfuscator for Δ-equivalent circuits in Theorem 8.

6.3 Security

In Lemma 7, we prove the security for δiO for δ-equivalent circuits. Here we provide a *stronger* security guarantee than the indistinguishability security of iO (Definition 8) for δ-equivalent circuits.

Note that the construction of δiO is gate-by-gate (See Figure 6). Namely, for each gate it outputs an obfuscated circuit $\widetilde{\mathsf{Gate}}_g$. For any two δ-equivalent circuit via a subcircuit S, the *only* difference between the outputs of $\delta\mathsf{iO}(C_1)$ and $\delta\mathsf{iO}(C_2)$ is $\{\widetilde{\mathsf{Gate}}_g\}_{g\in S}$, since by our δ-equivalence Definition 2, all gates outside of S are identical in C_1, C_2.

At a high level, in Lemma 7 we prove that even if we are *given all other variables*, including the PRF keys outside of S, and only switch $\{\widetilde{\mathsf{Gate}}_g\}_g$, then the two output distributions of δiO for C_1, C_2 are still indistinguishable.

Lemma 7 (Security for δiO). *The algorithm δiO in Figure 6 provides the following security. For any two circuits C_1, C_2 that are δ-equivalent via the subcircuit S, let $\{K^m_{b,w}, K^\sigma_{b,w}\}_w, \widetilde{\mathsf{td}}_b, \widetilde{K}_{b,i}, K_b, \mathsf{pk}_b$ be the variables generated in $\delta\mathsf{iO}(1^\lambda, C_b)$, for $b = 1, 2$. Then we have*

$$\left\{K_1, \mathsf{pk}_1, \widetilde{\mathsf{td}}_1, \{\widetilde{K}_{1,i}\}_{i\in[|S|]}, \{K^m_{1,w}, K^\sigma_{1,w}\}_{w\in\partial S}, \{\widetilde{\mathsf{Gate}}_{1,g}\}_{g\in S}\right\}_{\lambda\in\mathbb{N}} \approx$$
$$\left\{K_2, \mathsf{pk}_2, \widetilde{\mathsf{td}}_2, \{\widetilde{K}_{2,i}\}_{i\in[|S|]}, \{K^m_{2,w}, K^\sigma_{2,w}\}_{w\in\partial S}, \{\widetilde{\mathsf{Gate}}_{2,g}\}_{g\in S}\right\}_{\lambda\in\mathbb{N}},$$

where $\partial S = \mathsf{inp}(S) \cup \mathsf{out}(S)$.

$\underline{\delta\mathsf{iO}(1^\lambda, C)}$

- Generate a somewhere extractable hash key $K \leftarrow \mathsf{Gen}(1^\lambda)$. Then for each wire w in the circuit C, generate the following
  - Puncturable PRF key K_w^m for symmetric key encryption.
  - Puncturable PRF key K_w^σ for the MACs of the ciphertexts.
- Generate a FHE key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KGen}(1^\lambda)$, and let $\widetilde{\mathsf{td}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{|\mathsf{td}|})$. For each $i \in [|S|]$, let $\widetilde{K_i} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{|K_i^m|})$.
- For each gate g in C, let $\mathsf{inp}(g)$ and $\mathsf{out}(g)$ be the set of input and output wires of g, and for each $o \in \mathsf{out}(g)$ let $g_o : \{0,1\}^{|\mathsf{inp}(g)|} \to \{0,1\}$ be the function that computes the value of the output wire o. For each gate g, we obfsucate a circuit

$$\widetilde{\mathsf{Gate}_g} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{Gate}_{[K, \{K_i^m, K_i^\sigma\}_{i\in\mathsf{inp}(g)}, \{g_o, K_o^m, K_o^\sigma\}_{o\in\mathsf{out}(g)}, \mathsf{pk}, \widetilde{\mathsf{td}}, \{\widetilde{K_i}\}_{i\in[|S|]}]}).$$

- Output $\widetilde{C} = \left(\{\widetilde{\mathsf{Gate}_g}\}_{g\in C}, \{K_w^m, K_w^\sigma\}_{w\in\mathsf{inp}(C)\cup\mathsf{out}(C)}, K, \mathsf{pk}, \widetilde{\mathsf{td}}, \{\widetilde{K_i}\}_{i\in[|S|]}\right)$.

Figure 6: Description of the obfuscator for δ-equivalent circuits, where the circuit Gate is described in Figure 7.

Proof. We prove the lemma via a series of hybrids. Let C_1, C_2 be two δ-equivalent circuits via subcircuit S. Then C_1, C_2 only differ by such a subcircuit, and the functionalities of the subcircuit in C_1, C_2 are equivalent.

Hybrid H_0: This hybrid is the same as $\delta\mathsf{iO}(1^\lambda, C_1)$.

Hybrid H_1: This hybrid is almost the same as H_0, except that we replace the somewhere extractable hash key K to the trapdoor mode that is extractable for the indices $I = \{i\}_{i\in\mathsf{inp}(S)}$, such that we can extract the ciphertexts and hash values $\{\mathsf{ct}_i, h_i\}_{i\in\mathsf{inp}(S)}$ given the trapdoor td. Then we replace the FHE ciphertexts $\widetilde{\mathsf{td}}$ and $\widetilde{K}$ as $\widetilde{\mathsf{td}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{td})$ and $\widetilde{K_i} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, K_i^m)$, for every $i \in \mathsf{inp}(S)$.

This hybrid is computationally indistinguishable with H_0 from the key indistinguishability of the somewhere extractable hash and the semantic security of FHE.

Hybrid H_2: This hybrid is almost as the same as H_0, except that for each gate $g \in S$, instead of obfuscate the Gate circuit in Figure 7, we obfuscate the following circuit. This new circuit has the puncturable PRF keys $\{K_i^m\}_{i\in\mathsf{inp}(S)}$ hardwired.

- …(The same verification of the local openings, proof of consitency, and MACs)…
- Extract the input to the subcircuit via the decryption of the FHE:

$$\{m_i^*\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h).$$

  If any of $\{m_i^*\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)}$ is $\perp$, then abort.

- Compute the output wire values $\{m_o\}_{o\in\mathsf{out}(g)}$:

$$m_o = {C_1}_S^o(\{m_i^*\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)}),$$

  where ${C_1}_S^o$ is the circuit that computes the output wire value o directly from the inputs $\{m_i^*\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)}$.

$\underline{\mathsf{Gate}_{[K,\{K_i^m,K_i^\sigma\}_{i\in\mathsf{inp}(g)},\{g_o,K_o^m,K_o^\sigma\}_{o\in\mathsf{out}(g)},\mathsf{pk},\widetilde{\mathsf{td}},\{\widetilde{K}_i\}_{i\in[|S|]}]}(H,\{H_i,\mathsf{ct}_i,\rho_i,\pi_i,\sigma_i\}_{i\in\mathsf{inp}(g)})}$

- Homomorphically evaluate the circuit Shrink (Figure 8) for $H, \{H_i\}_{i\in\mathsf{inp}(g)}$,

$$h \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{[H]}, (\widetilde{\mathsf{td}}, \{\widetilde{K}_i\}_{i\in[|S|]})),$$

$$h_i \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{[H_i]}, (\widetilde{\mathsf{td}}, \{\widetilde{K}_w\}_{w\in[|S|]})), \quad \forall i \in \mathsf{inp}(g).$$

- For each input wire $i \in \mathsf{inp}(g)$ of the gate g, we verify the following:
  - Local opening ρ_i for the hash value H with message ct_i and index i,

$$\mathsf{Verify}(K, H, (\mathsf{ct}_i, h_i), i, \rho_i) \overset{?}{=} 1.$$

  - Proof of consistency for H and H_i with proof π_i: $\mathsf{V}(K, H, H_i, \pi_i) \overset{?}{=} 1$.
  - MAC σ_i for the ciphertext ct_i: $\mathsf{PRG}(\sigma_i) \overset{?}{=} \mathsf{PRG}(\mathsf{PRF}_{punc}(K_i^\sigma, (\mathsf{ct}_i, h_i)))$.
- Decrypt the symmetric key ciphertexts ct_i, $i \in \mathsf{inp}(g)$

$$m_i = \mathsf{ct}_i \oplus \mathsf{PRF}_{punc}(K_i^m, h_i), \quad \forall i \in \mathsf{inp}(g).$$

- Compute the wire value m_o of for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$, where g_o the function computing the wire o at gate g.
- Encrypt m_o under the key K_o^m, and MAC the ciphertext to obtain σ_o for each output wire o, i.e.

$$\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_{punc}(K_o^m, h), \quad \sigma_o = \mathsf{PRF}_{punc}(K_o^\sigma, (\mathsf{ct}_o, h)), \quad \forall o \in \mathsf{out}(g).$$

- Output $\{(\mathsf{ct}_o, \sigma_o)\}_{o\in\mathsf{out}(g)}$.

Figure 7: Description of the circuit Gate.

- …(Compute the ciphertexts and MACs in the same way as in before)…

To show that Hyrbid H_1 and Hybrid H_2 are indistinguishable, we will use a series of intermediate hybrids.

Hybrid $\mathsf{H}_{1.5}^{<g}$: This hybrid is indexed by a gate g. In this hybrid, the obfuscator processes all gates that g depends on in the same way as H_2, but on the gate g, it still obfsucates the circuit Gate in Figure 7. Hence, to show that the Hybrid H_1 and the Hybrid H_2 are indistinguishable, it suffices to show that the Hybrid $\mathsf{H}_{1.5}^{<g}$ and the following Hybrid $\mathsf{H}_{1.5}^{\le g}$ are indistinguishable.

Hybrid $\mathsf{H}_{1.5}^{\le g}$: This gate is almost the same as Hybrid $\mathsf{H}_{1.5}^{<g}$, except that we also obfuscate the gate g in the same way as the Hybrid H_2. To show that $\mathsf{H}_{1.5}^{<g}$ and $\mathsf{H}_{1.5}^{\le g}$ are indistinguishable, we again use a series of intermediate hybrids. For simplicity, let us consider a typical case, where the gate g has arity-2 with two input wires $\mathsf{inp}(g) = \{l, r\}$, and a single output wire $\mathsf{out}(g) = \{o\}$. The situations for arity-1 gate or two output wires are similar.

$\underline{\mathsf{Shrink}_{[H]}(\mathsf{td}, \{K_i^m\}_{i\in[|S|]})}$

- Let I be the subset of indices that SEH can extract. Extract the ciphertexts and hash values by SEH,
$$\{(\mathsf{ct}_i^*, h_i^*)\}_{i\in I} \leftarrow \mathsf{Ext}(H, \mathsf{td}).$$
- For each $i \in I$, if ct_i^*, h_i^* contain $\perp$, then let $m_i^* = \perp$. Otherwise, decrypt the message
$$m_i^* \leftarrow \mathsf{ct}_i^* \oplus \mathsf{PRF}_{punc}(K_i, h_i^*).$$
- Output $\{m_i^*\}_{i\in I}$.

Figure 8: Description of the circuit Shrink.

Hybrid $\mathsf{H}_{1.5,l}^{<g}$**:** This hybrid is almost the same as Hybrid $\mathsf{H}_{1.5}^{<g}$, except that we additionally extract $\{m_i^*\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h)$ and checks whether $\{m_i^*\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ contains $\perp$. If it does, then abort. To show that the current Hybrid $\mathsf{H}_{1.5,l}^{<g}$ and the Hybrid $\mathsf{H}_{1.5}^{<g}$ are indistinguishable, we use the following series of hybrids $\mathsf{H}_{1.5,l,(b,h_l^1)}^{<g}$, and we will show that $\mathsf{H}_{1.5,l,(b,h_l^1)}^{<g}$ and $\mathsf{H}_{1.5,l,(b,h_l^1)+1}^{<g}$ are indistinguishable.

Hybrid $\mathsf{H}_{1.5,l,(b,h_l^1)}^{<g}$**:** This hybrid is almost the same as the previous hybrid, except that we add an index (b, h_l^1), and decide whether to check the extracted messages by the index (b, h_l^1).

- …(The same verification of the local openings, proof of consistency, and MACs)…
- Extract $\{m_i^*\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h)$.
- If $(\mathsf{ct}_l, h_l) < (b, h_l^1)$, then check whether $\{m_i^*\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ contains $\perp$. If it does, then abort.
- …(Compute the ciphertexts and MACs in the same way as before)…

To show that the Hybrid $\mathsf{H}_{1.5,l,(b,h_l^1)}^{<g}$ and the Hybrid $\mathsf{H}_{1.5,l,(b,h_l^1)+1}^{<g}$ are indistinguishable, we need to utilize the consistency proof of SEH.

Let's denote $\{m_i^1\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h_l^1)$. By *statistical somewhere soundness* of the SEH with consistency proof (See Section 6.1), $\{m_i^*\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h)$ extracted in the gate g always satisfies $m_i^1 \neq \perp \rightarrow (m_i^* = m_i^1)$, for an overwhelming fraction of the SEH key K. Hence, it suffices to consider the case when $\{m_i^1\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ contains $\perp$, since for other cases, $\{m_i^*\}_{i\in\mathsf{inp}(S)\mathsf{dep}(l)}$ also does not contains $\perp$, and then the check of $\perp$ doesn't affect the functionality, so that the indistinguishability follows from the security of the underlying iO.

Now, let's consider the case when $\{m_i^1\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ contains a $\perp$. If l is an input wire to the subcircuit S, then $\mathsf{inp}(S) \cap \mathsf{dep}(l) = \phi$ the additional check does nothing, and hence the indistinguishability also follows from the indistinguishability security of the underlying iO.

Hence, we only need to consider the case when the wire l is the output wire of some gate g_l that is also in the subcircuit S (otherwise l is an input wire). In this case, we first show that $\mathsf{H}_{1.5,l,(b,h_l^1)}^{<g}$ and the following new hybrid $\mathsf{H}_{1.5,l,(b,h_l^1),1}^{<g}$ are indistinguishable.

Hybrid $\mathsf{H}_{1.5,l,(b,h_l^1),1}^{<g}$**:** This hybrid is almost the same as the hybrid $\mathsf{H}_{1.5,l,(b,h_l^1)}^{<g}$, except that we replace the PRF key for verifying the MAC σ_l with the PRF key punctured at the point (b, h_l^1). Note that, in this hybrid, we also modify the gate g_l. Namely, for gate g, we modify the circuit Gate as follows.

Evaluation of δiO

- Given an obfuscated circuit $\widetilde{C} = \left(\{\widetilde{\mathsf{Gate}}_g\}_{g\in[|C|]}, \{K_i^m, K_i^\sigma\}_{i\in[|\mathsf{inp}(C)|]}, K, \widetilde{\mathsf{td}}, \{\widetilde{K}_i\}_{i\in[|S|]}\right)$, and an input $x \in \{0,1\}^{|\mathsf{inp}(C)|}$, we now compute $C(x)$, assuming the topology of C is public.
- We iterate the gates of C in its topological order. For each wire w, we denote $\mathsf{dep}(w)$ the set of the wires that w depends on, not including w. For each gate g, we do the following.
  - Prepare the hash of the ciphertexts that the input wires and the output wires depends on:
  $$H = \mathsf{Hash}(K, \{\mathsf{ct}_w, h_w\}_{w\in\mathsf{dep}(o)}), \quad \forall i \in \mathsf{inp}(g), H_i = \mathsf{Hash}(K, \{\mathsf{ct}_w, h_w\}_{w\in\mathsf{dep}(i)}),$$
  where $o \in \mathsf{out}(g)$ is one of the output wire of g. (Note that all output wires have the same set of the dependent wires.)
  - Shrink the hash values $H, \{H_i\}_{i\in\mathsf{inp}(g)}$ by homomorphically evaluating Shrink (Figure 8) and obtain values $h, \{h_i\}_{i\in\mathsf{inp}(g)}$ as in Figure 7.
  - Prepare the local opening for $\{\mathsf{ct}_i, h_i\}_{i\in\mathsf{inp}(g)}$ in the hash value H, let
  $$\rho_i \leftarrow \mathsf{Open}(K, \{\mathsf{ct}_w, h_w\}_{w\in\mathsf{dep}(o)}, i), \quad \forall i \in \mathsf{inp}(g).$$
  - For each input wire $i \in \mathsf{inp}(g)$, if i is an input wire of the entire circuit, then compute the ciphertext ct_i from x_i using the key K_i^m, and MAC the ciphertext using the key K_i^σ as σ_i. Otherwise, since we iterate the gates in the topological order, ct_i and σ_i must have already been produced by previous gates.
  - Compute the gate g. Let
  $$\{(\mathsf{ct}_o, \sigma_o)\}_{o\in\mathsf{out}(g)} \leftarrow \widetilde{\mathsf{Gate}}_g(h, \{h_i, \mathsf{ct}_i, \rho_i, \pi_i, \sigma_i\}_{i\in\mathsf{inp}(g)}).$$
- Finally, use the key $\{K_o^m\}_{o\in\mathsf{out}(C)}$ to decrypt the output wires $\{\mathsf{ct}_o\}_{o\in\mathsf{out}(C)}$ and output the wire values.

Figure 9: Description of evaluation of δiO.

ΔiO$(1^\lambda, C, s)$

- Padding the circuit C by $C' \leftarrow \mathsf{Pad}(1^\lambda, C, s)$, where Pad is the padding algorithm in Lemma 3.
- Obfuscate the padded circuit C' as $\widetilde{C} \leftarrow \delta\mathsf{iO}(1^\lambda, C')$, and output $\widetilde{C}$.

Figure 10: Description of the obfuscator for Δ-equivalent circuits.

- …(The same verification of the local openings, proof of consistency, and the MAC for σ_r)…
- If $(\mathsf{ct}_l, h_l) \neq (b, h_l^1)$, then verify if

$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_{punc}(K_l^{\sigma} \setminus \{(b, h_l^1)\}, (\mathsf{ct}_l, h_l))).$$

  Otherwise, verify if

$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_{punc}(K_l^{\sigma}, (b, h_l^1))).$$

- Extract $\{m_i^*\}_{i \in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h)$.
- If $(\mathsf{ct}_l, h_l) < (b, h_l^1)$, then check whether $\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(l)}$ contains $\perp$. If it does, then abort.
- …(Compute the ciphertexts and MACs in the same way as before)…

For the gate g_l, we modify the gate Gate for g_l as follows.

- …(The same verification and computing $\{m_o\}_{o \in \mathsf{out}(g_l)}$ as H_2) …
- …(The same encryption of ct_l and other output wires as H_2, and the same MACs for other output wires except l) …
- MAC the ciphertext ct_l and hash value h_l for l as follows. If $(\mathsf{ct}_l, h_l) \neq (b, h_l^1)$, then let $\sigma_l \leftarrow \mathsf{PRF}_{punc}(K_l^{\sigma} \setminus \{(b, h_l)\}, (\mathsf{ct}_l, h_l))$. Otherwise, let $\sigma_l \leftarrow \mathsf{PRF}_{punc}(K_l^{\sigma}, (b, h_l^1))$.

The indistinguishability from $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1)}$ and $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),1}$ follows from the preservation of the functionality for puncturable PRF (Section 5.3) and iO indistinguishability.

Hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),2}$**:** This hybrid is almost the same as the last hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),1}$, except that we replace each occurrence of the value $\mathsf{PRF}_{punc}(K_l^{\sigma}, (b, h_l^1))$ with a random value s. This hybrid is indistinguishable with the last hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),1}$, by the pseudorandomness at the punctured point property (Section 5.3).

Hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),3}$**:** This hybrid is almost the same as last hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),2}$, except that we replace each occurrence of $\mathsf{PRG}(s)$ with a random value u in the range of the PRG PRG.

For the gate g, the circuit we obfuscate becomes the following circuit, with a random u hardwired.

- …
- If $(\mathsf{ct}_l, h_l) \neq (b, h_l^1)$, then verify if

$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_{punc}(K_l^{\sigma} \setminus \{(b, h_l)\}, (\mathsf{ct}_l, h_l))).$$

  Otherwise, verify if

$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} u.$$

- Extract $\{m_i^*\}_{i \in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h)$.
- If $(\mathsf{ct}_l, h_l) < (b, h_l^1)$, then check whether $\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(l)}$ contains $\perp$. If it does, then abort.
- …

For the gate g_l, we obfuscate the following circuit.

- …
- MAC the ciphertext ct_l and hash value h_l for l as follows. If $(\mathsf{ct}_l, h_l) \neq (b, h_l^1)$, then let $\sigma_l = \mathsf{PRF}_{punc}(K_l^\sigma \setminus \{(b, h_l)\}, (\mathsf{ct}_l, h_l))$. Note that here we no longer have the branch $(\mathsf{ct}_l, h_l) = (b, h_l^1)$.

To show the indistinguishability between the current hybrid and the last hybrid, we consider the following two cases. Since $\{m_i^1\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(l)}$ contains $\perp$, then by our ordering on the gates, such h_l^1 must be rejected at gate g_l. Hence, the assignment $\sigma_l = u$ never happens. Then the indistinguishability between the current hybrid and the last hybrid follows from the indistinguishability of iO and the security of PRG.

Hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1),4}$**:** This hybrid is almost the same as the last hybrid, except that modify the check in the circuit for g as follows.

- …
- If $(\mathsf{ct}_l, h_l) < (b, h_l^1)$, then check whether $\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(l)}$ contains $\perp$. If it does, then abort.
- If $(\mathsf{ct}_l, h_l) = (b, h_l^1)$, and $\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(l)}$ contains $\perp$, then abort.
- …

Since u is uniformly random, if we set the *expansion* of the PRG to be $\lambda + |(b, h_l^1)|$, then with $(1 - 1/2^{\lambda + |h_l^1| + 1})$-fraction of u, it doesn't fall in the range of the PRG G. Hence, the functionalities of the circuit for g in this hybrid and the last hybrid are identical with probability $1 - 1/2^{\lambda + |h_l^1| + 1}$. Then the indistinguishability follows from the security of the underlying iO.

Note that this hybrid is also the same as the Hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1)+1}$. Hence, by the hybrid argument, the Hybrid $\mathsf{H}^{<g}_{1.5,l}$ and the Hybrid $\mathsf{H}^{<g}_{1.5}$ are indistinguishable.

Hybrid $\mathsf{H}^{<g}_{1.5,l,r}$**:** This hybrid is almost the same as $\mathsf{H}^{<g}_{1.5,l}$, except that for the gate g, we also checks whether $\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(r)}$ contains $\perp$, and if it does, then we abort.

The indistinguishability between the Hybrid $\mathsf{H}^{<g}_{1.5,l}$ and the current Hybrid $\mathsf{H}^{<g}_{1.5,l,r}$ follows from the similar argument from $\mathsf{H}^{<g}_{1.5}$ to $\mathsf{H}^{<g}_{1.5,l}$, where we add the check for the left input wire l there, and now we are adding the check for the right input wire r.

Hybrid $\mathsf{H}^{<g}_{1.5,\perp}$**:** This hybrid is almost the same as the Hybrid $\mathsf{H}^{<g}_{1.5,l,r}$, except that it aborts if $\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(o)}$ contains $\perp$, while $\mathsf{H}^{<g}_{1.5,l,r}$ only checks for $i \in \mathsf{inp}(S) \cap \mathsf{dep}(l)$ and $i \in \mathsf{inp}(S) \cap \mathsf{dep}(r)$. Note that $\mathsf{inp}(S) \cap \mathsf{dep}(o)$ equals to the *disjoint* union of the following three sets: $\mathsf{inp}(S) \cap \mathsf{dep}(l)$, $\mathsf{inp}(S) \cap \mathsf{dep}(r)$, and $\mathsf{inp}(S) \cap \{l, r\}$. Hence, from $\mathsf{H}^{<g}_{1.5,l,r}$ to $\mathsf{H}^{<g}_{1.5,\perp}$, we only additionally check $\mathsf{inp}(S) \cap \{l, r\}$.

Note that, since we check whether the local openings ρ_l, ρ_r to $(\mathsf{ct}_l, h_l), (\mathsf{ct}_r, h_r)$ are accepted (See Figure 7), if such check passes, then for any $i \in \mathsf{inp}(S) \cap \{l, r\}$, we have that the extracted values ct_i^*, h_i^* from the Ext algorithm of SEH do not contain $\perp$ (by extraction correctness in Definition 5.6). Hence, by the Shrink circuit (Figure 8), it outputs $m_i^* \neq \perp$. Hence, m_i^* decrypted by Dec is also not $\perp$, for each $i \in \mathsf{inp}(S) \cap \{l, r\}$. Now, we can argue the indistinguishability between $\mathsf{H}^{<g}_{1.5,l,r}$ and $\mathsf{H}^{<g}_{1.5,\perp}$ by the security of iO.

Now, by the hybrid argument, we derive that $\mathsf{H}^{<g}_{1.5}$ and $\mathsf{H}^{<g}_{1.5,\perp}$ are indistinguishable.

Hybrid $\mathsf{H}^{<g}_{1.5,\perp,h_l^1}$**:** This hybrid is almost the same as $\mathsf{H}^{<g}_{1.5,\perp}$, except that we additionally add a checking for the decrypted values m_l of the ciphertext (ct_l, h_l), depending on whether $h_l < h_l^1$ or not. Namely, we modify the circuit we obfuscate at gate g as follows.

- …

- If $h_l < h_l^1$, then check $m_l \overset{?}{=} C1_S^l(\{m_i^*\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)})$. If not, then abort.

- …(Compute and output the ciphertexts and MACs of the output wires) …

Now we use a series of hybrids to argue that $\mathsf{H}^{<g}_{1.5,\perp,h_l^1}$ and $\mathsf{H}^{<g}_{1.5,\perp,h_l^1+1}$ are indistinguishable.

Hybrid $\mathsf{H}^{<g}_{1.5,\perp,h_l^1,1}$**:** This hybrid is almost the same as $\mathsf{H}^{<g}_{1.5,\perp,h_l^1}$, except that we puncture the point for h_l^1 in the circuit we obfuscate at the gate g. If l is not an input wire of S, let g_l be the gate whose output wire is l. Then we also puncture the h_l^1 when we compute the MAC for l at g_l. Namely, we modify the circuit at the gate g as follows. Let $\{m_i^1\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ be a set of input wires extracted by $\{m_i^1\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h_l^1)$. If $\{m_i^1\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ *contains* $\perp$, then by a similar argument in the hybrid from $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1)}$ to $\mathsf{H}^{<g}_{1.5,l,(b,h_l^1)+1}$, we can argue the indistinguishability. Hence, it suffices to consider the case when $\{m_i^1\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$ doesn't contain $\perp$.

Let $m_l^1 = C1_S^l(\{m_i^1\}_{i\in\mathsf{inp}(S)})$ be the unique message computed by $\{m_i^1\}_i$ given the subcircuit S, and let $\mathsf{ct}_l^1 = m_l^1 \oplus \mathsf{PRF}_{punc}(K_l^m, h_l^1)$ be its ciphertext. We denote $\overline{\mathsf{ct}_l^1} = \mathsf{ct}_l^1 \oplus 1$ as the flip of ct_l^1. We now puncture the point $(\overline{\mathsf{ct}_l^1}, h_l^1)$ for $\mathsf{PRF}_{punc}(K_l^\sigma, \cdot)$.

- …

- If $h_l \neq h_l^1$, then verify if

$$\mathsf{PRG}(\sigma_l) \overset{?}{=} \mathsf{PRG}(\mathsf{PRF}_{punc}(K_l^\sigma \setminus \{(\overline{\mathsf{ct}_l^1}, h_l^1)\}, (\mathsf{ct}_l, h_l))).$$

  Otherwise, if $h_l = h_l^1$ and $\mathsf{ct}_l = \mathsf{ct}_l$, then verify if

$$\mathsf{PRG}(\sigma_l) \overset{?}{=} \mathsf{PRF}_{punc}(K_l^\sigma \setminus \{(\overline{\mathsf{ct}_l^1}, h_l^1)\}, (\mathsf{ct}_l, h_l^1)).$$

  Otherwise, $h_l = h_l^1$ and $\mathsf{ct}_l = \overline{\mathsf{ct}_l^1}$, then we verify the following

$$\mathsf{PRG}(\sigma_l) \overset{?}{=} \mathsf{PRG}(\mathsf{PRF}_{punc}(K_l^\sigma, (\overline{\mathsf{ct}_l^1}, h_l^1))).$$

- …

And we modify the circuit at gate g_l as follows.

- …(Verifications)…

- …(Extract wire values in h_l and compute m_l directly from the extracted values) …

- MAC the ciphertext ct_l and hash value h_l for l as follows,

$$\sigma_l = \mathsf{PRF}_{punc}(K_l^\sigma \setminus \{(\overline{\mathsf{ct}_l^1}, h_l^1)\}, (\mathsf{ct}_l, h_l)).$$

The indistinguishability of the current hybrid $\mathsf{H}^{<g}_{1.5,\perp,h_l^1,1}$ and the last hybrid $\mathsf{H}^{<g}_{1.5,\perp,h_l^1}$ follows from the security of the underlying iO, since the functionality equivalence of the obfuscated circuit at the gate g_l can be argued as follows.

The only change we make is how the MAC σ_l is computed. Note that m_l is derived from h_l by extraction. Hence, if $h_l = h_l^1$, then $m_l = m_l^1$, and $\mathsf{ct}_l = \mathsf{ct}_l^1$. Hence, the point $(\overline{\mathsf{ct}_l^1}, h_l^1)$ is never used in $\mathsf{H}^{<g}_{1.5,\perp,h_l^1}$, and thus we keep the functionality unchanged at the gate g_l.

Hybrid $\mathsf{H}^{<g}_{1.5,\perp,h^1_l,2}$**:** This hybrid is almost the same as the last hybrid $\mathsf{H}^{<g}_{1.5,\perp,h^1_l,1}$, except that, at the gate g, we directly abort if $h_l = h^1_l$ and $\mathsf{ct}_l \neq \mathsf{ct}^1_l$. This hybrid is indistinguishable from the last hybrid $\mathsf{H}^{<g}_{1.5,\perp,h^1_l,1}$, following the same argument as the hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h^1_l),2}$ to hybrid $\mathsf{H}^{<g}_{1.5,l,(b,h^1_l),4}$.

From Hybrid $\mathsf{H}^{<g}_{1.5,\perp,h^1_l,2}$ **to Hybrid** $\mathsf{H}^{<g}_{1.5,\perp,h^1_l+1}$**:** Note that in hybrid $\mathsf{H}^{<g}_{1.5,\perp,h^1_l,2}$, we check whether $\mathsf{ct}_l = \mathsf{ct}^1_l$ when $h_l = h^1_l$. Hence, we can add an additional checking $m_l \overset{?}{=} m^1_l$ without changing the functionality of the circuit being obfuscated. By the hybrid argument, $\mathsf{H}^{<g}_{1.5,\perp,h^1_l,2}$ and $\mathsf{H}^{<g}_{1.5,\perp,h^1_l+1}$ are indistinguishable.

By the hybrid argument, $\mathsf{H}^{<g}_{1.5,\perp,h^1_l}$ and $\mathsf{H}^{<g}_{1.5,\perp,h^1_l+1}$ are indistinguishable.

Hybrid $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check}}$**:** This hybrid is almost the same as hybrid $\mathsf{H}^{<g}_{1.5,\perp,1^{|h_l|}+1}$, except that we add an additional check for the right input wire r. Namely, the circuit we obfuscate at g becomes

- ...
- Check $m_w \overset{?}{=} C_1{}^w_S(\{m^*_i\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)})$ for $w \in \{l, r\}$. If not, then abort.
- ...

Hybrid $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check},C_S}$**:** This hybrid is almost the same as the last hybrid $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check}}$, except that we modify the computation of m_o at the gate g as $m_o = C_1{}^o_S(\{m^*_i\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)})$. Given the check on the input wires, the functionality of the circuit we obfuscate at the gate g is the same as the Hybrid $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check}}$. The indistinguishability of the current hybrid and the last Hybrid $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check},C_S}$ follows from the security of the underlying iO.

From Hybrid $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check},C_S}$ **to Hybrid** $\mathsf{H}^{\leq g}_{1.5}$**:** The only difference between $\mathsf{H}^{<g}_{1.5,\perp,\mathsf{check},C_S}$ and $\mathsf{H}^{\leq g}_{1.5}$ is that, in $\mathsf{H}^{\leq g}_{1.5}$ we additionally check whether the input wire values are correctly computed. We can undo this checking by reversing the hybrids that allows us to introduce it.

By the hybrid argument, $\mathsf{H}^{<g}_{1.5}$ and $\mathsf{H}^{\leq g}_{1.5}$ are indistinguishable, and thus H_1 and H_2 are indistinguishable.

Hybrid H_3**:** In this hybrid, for each gate in the subcircuit S, instead of computing each output wires by C_1, we compute it by the circuit C_2. Namely, at each gate $g \in S$, we obfuscate the following circuit. Note that in H_2, each gate g in the subcircuit S only computes its output wire value directly from the input wires in $\mathsf{inp}(S)$.

- ...(The same verification of the local openings, proof of consitency, and MACs)...
- Extract the input to the subcircuit via the decryption of the FHE:

  $$\{m^*_i\}_{i\in I} \leftarrow \mathsf{Dec}(\mathsf{sk}, h).$$

  If any of $\{m^*_i\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)}$ is $\perp$, then abort.
- Compute the output wire values $\{m_o\}_{o\in\mathsf{out}(g)}$:

  $$m_o = C_2{}^o_S(\{m^*_i\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)}),$$

  where $C_1{}^o_S$ is the circuit that computes the output wire value o directly from the inputs $\{m^*_i\}_{i\in\mathsf{inp}(S)\cap\mathsf{dep}(o)}$.
- ...(Compute the ciphertexts and MACs in the same way as in before)...

To show that the Hybrid H_2 and H_3 are indistinguishable, we use a series of intermediate hybrids that is indexed by a gate $g^1 \in S$, as follows.

Hybrid $\mathsf{H}_{2.5,g^1}$**:** For each gate g with $g < g^1$, it computes its output wire using C_2, but for each gate $g \geq g^1$, it still uses C_1 to compute its output wires. We will show that $\mathsf{H}_{2.5,g^1}$ and $\mathsf{H}_{2.5,g^1+1}$ are indistinguishable. Let o be the output wire of g. If o is also an output wire of the subcircuit S, then we can directly switch ${C_1}_S^o$ to ${C_2}_S^o$, since the functionality of the subcircuit is preseved. Next, we are going to use the security of the symmetric key encryption to show that we can still switch to use C_2 when o is not an output wire of S, by using the following Hybrids $\mathsf{H}_{2.5,g^1,h_o^1}$.

Hybrid $\mathsf{H}_{2.5,g^1,h_o^1}$**:** This hybrid is almost the same as the hybrid $\mathsf{H}_{2.5,g^1}$, except that we compute the wire value m_o and its ciphertext ct_o as the following.

- …
- If $h < h_o^1$, compute $m_o = {C_2}_S^o(\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(o)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_{punc}(K_o^m \setminus \{h_o^1\}, h)$.
- Otherwise, If $h = h_o^1$, compute $m_o = {C_1}_S^o(\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(o)}$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_{punc}(K_o^m, h_o^1)$.
- Otherwise, $h > h_o^1$, compute $m_o = {C_1}_S^o(\{m_i^*\}_{i \in \mathsf{inp}(S) \cap \mathsf{dep}(o)}$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_{punc}(K_o^m \setminus \{h_o^1\}, h)$.
- …

Next, we can use the pseudorandomness of the punctured PRF at the point h_o^1 to argue that the hybrid $\mathsf{H}_{2.5,g^1,h_o^1}$ and hybrid $\mathsf{H}_{2.5,g^1,h_o^1+1}$ are indistinguishable. We can do this because in the hybrid $\mathsf{H}_{2.5,g^1,h_o^1}$, the PRF key K_o^m is only used to encrypt the message, and is never used elsewhere for decryption.

Hence, by the hybrid argument, $\mathsf{H}_{2.5,g^1}$ and $\mathsf{H}_{2.5,g^1+1}$ are indistinguishable, and thus H_2 and H_3 are indistinguishable.

Hybrid H_4**:** This hybrid is the final hybrid, it is almost the same as hybrid H_0, except that we use each gate in C_2 to obfuscate the circuit.

This hybrid is indistinguishable with the last H_3, by using the same hybrid arguments from H_0 to H_2 in the reverse order.

By the hybrid argument, we finish the proof. $\square$

Theorem 8 (δiO is a ΔiO)**.** *The algorithm in Figure 6 is a ΔiO for Δ-equivalent circuits. Namely, there exist polynomials $p(\cdot), q(\cdot,\cdot,\cdot)$, such that assuming the hardness of the following assumptions, there exists a construction of* iO *for any two Δ-equivalent circuit families $\{C_n^1\}_{n\in\mathbb{N}}, \{C_n^2\}_{n\in\mathbb{N}}$:*

- *Polynomial-hardness of Learning with Errors,*
- *Sub-exponential secure pseudorandom generators and puncturable pseudorandom functions, with security parameters $\lambda_{\mathsf{PRG}} = \lambda_{\mathsf{PRF}} = p(\lambda)$*
- *Sub-exponential secure indistinguishability obfuscation for circuits of size $q(\lambda, \log|C_n^1|, \log|C_n^2|)$, with security parameter $\lambda_{\mathsf{iO}} = q(\lambda, \log|C_n^1|, \log|C_n^2|)$*

*where λ is the security parameter of the δ*iO*.*

Remark 2. *Crucially, the security parameters of the underlying primitives and assumptions only rely on a fixed polynomial in λ, and is independent of n, or the input length of the circuits $\{C_n^1\}_{n\in\mathbb{N}}, \{C_n^2\}_{n\in\mathbb{N}}$.*

Since one-way functions imply pseudorandom generators and puncturable pseudorandom functions, we can also base our result on sub-exponential secure one-way functions.

Proof. For any Δ-equivalent circuit families $\{C_n^1\}_{n\in\mathbb{N}}$ and $\{C_n^2\}_{n\in\mathbb{N}}$, for any n, there exists a series of intermediate circuits $C_n^1 = C_1', C_2', \ldots, C_\ell' = C_n^2$, where C_i' and C_{i+1}' are δ-equivalent via a subcircuit of size $O(\log n)$.

By Lemma 7, we have that $\delta\mathsf{iO}(C_i')$ and $\delta\mathsf{iO}(C_{i+1}')$ are indistinguishable. By hybrid argument, $\delta\mathsf{iO}(C_n^1)$ and $\delta\mathsf{iO}(C_n^2)$ are also indistinguishable. □

Corollary 1 (iO for Circuits with Efficient Propositional Proofs of Equivalence)**.** *Under the same assumptions as Theorem 8, the algorithm in Figure 10 is an iO for circuit families with efficient propositional proofs of equivalence.*

The proof follows directly from Lemma 3 and Theorem 8.

7 iO for Turing Machines

In this section, we show the construction of iO for Turing machines with PV-proof of equivalence. In section 7.1, we define Δ-equivalence for Turing machines. Section 7.2 and Section 7.3 will prove that Cook's translation implies Δ-equivalence of Turing machines. In section 7.4, we construct iO for Turing machines with Δ-equivalence, and as a corollary, we obtain iO for Turing machines with PV proof of equivalence.

7.1 Δ-Equivalence for Turing Machines

To build iO for Turing machines, we rely on the conversion of Turing machines to circuits, and then apply our result of iO for circuits. However, doing this conversion naively would result in a bounded-input circuit, which is not sufficient for our goal for unbounded Turing machines. To avoid this, we rely on the *uniform structure* of the converted circuits. Namely, its *description* can be generated by a small circuit of poly-logarithmic size. In the following, we formally state this uniform structure.

Small Description Circuit. There exists a polynomial $S(\cdot,\cdot)$ such that, for any *polynomial-time* Turing machine M that halts in $T(\cdot)$ steps, any input length bound N, we can construct a circuit $[\![M]\!]_N(\cdot,\cdot)$ of size $S(\log N, |M|)$, with the following syntax. Here, we use $|M|$ to denote the number of bits used to describe the Turing machine M.

$$(\mathsf{inp}_g, \mathsf{out}_g, f_g) \leftarrow [\![M]\!]_N(n, g).$$

Specifically, $[\![M]\!]_N$ takes an input length n and an index g of the gates, and outputs the description of the g-th gate, which contains a set of indices for its input wires inp_g, a set of indices for its output wires out_g, and the computation g-th gate performs $f_g : \{0,1\}^{|\mathsf{inp}(g)|} \to \{0,1\}$.

Without loss of generality, we use $T(n)$ to bound the number of gates that $[\![M]\!]_N(n,\cdot)$ describes. Namely, the circuit formed by the gates $\{[\![M]\!]_N(n,g)\}_{g\in[T(n)]}$ emulates the Turing machine for the input length n.

In the following, we define the notion of Δ-equivalence for two Turing machines.

Definition 11 (Δ-equivalent Turing Machines)**.** *We say two Turing machines M_1, M_2 are Δ-equivalent, if there exist functions $\ell = \mathsf{poly}(n, \log N)$, $B(N) = O(\log N)$, and $S(N) = \mathsf{poly}(\log N)$ such that for any positive integer N and any $n < N$, there exists a series of circuit $C_1'(\cdot), C_2'(\cdot), \ldots, C_\ell'(\cdot)$ with the following properties.*

- *$[\![M_1]\!]_N(n,\cdot) \equiv C_1'$ and $C_\ell' \equiv [\![M_2]\!]_N(n,\cdot)$. Recall that "$\equiv$" means functionally equivalence.*
- *Let C_i be the circuit that C_i' describes. That is, $C_i'(g)$ outputs the description $(\mathsf{inp}_g, \mathsf{out}_g, f_g)$ of the g-th gate in C_i. Then for each $i \in [\ell-1]$, C_i and C_{i+1} are δ-equivalent via a subcircuit of size $B(N)$.*
- *For each $i \in [\ell]$, the size of the circuit $|C_i'|$ is bounded by $S(N)$.*

7.2 Succinct Description of Cook's Translation

In this section, we give a uniform presentation of Cook's translation [Coo75] from *PV* to propositional logic. Namely, the translated propositional logic proof can be "described" by a small circuit of size *independent* of the number of digits in the dyadic notation.

To achieve this, we at least need the "lines" in the translated proof to be short, so that the output of the description circuit can be succinct. Hence, as we did in Lemma 3, we assign new variables to the sub-formulas to break the size of any large formula to constant.

Introducing Intermediate Variables. For each formula f in $\mathcal{EF}$, we assign it with a new variable $\mathsf{ATOM}[f]$ to reduce the size of the formulas in the proof, as we did in the proof of Lemma 3. If we treat f as a circuit, then we can represent the computation of each gate in it as an extension. We use $\mathsf{Def}(f)$ to denote the set containing all such extensions in f. Formally, $\mathsf{Def}(f)$ is defined inductively as follows.

- If f is a variable or constant, then $\mathsf{Def}(f)$ is defined to be $\mathsf{Def}(f) = \{\mathsf{ATOM}[f] \leftrightarrow f\}$.
- If f is $f_1 \circ f_2$, where f_1, f_2 are formulas, and $\circ$ is either $\wedge, \vee$ or $\rightarrow$, then we define $\mathsf{Def}(f)$ as $\mathsf{Def}(f_1) \cup \mathsf{Def}(f_2) \cup \{\mathsf{ATOM}[f] \leftrightarrow (\mathsf{ATOM}[f_1] \circ \mathsf{ATOM}[f_2])\}$.
- If f is $\neg f'$, then we define $\mathsf{Def}(f) = \mathsf{Def}(f') \cup \{\mathsf{ATOM}[f] \leftrightarrow \neg\mathsf{ATOM}[f']\}$.

Definition 12 (Small Formula Derivation)**.** *Let S be a set of propositional formulas, and f be a formula in $\mathcal{EF}$. We say a sequence of formulas*

$$\Pi = (\pi_1, \pi_2, \dots, \pi_E, \pi_{E+1}, \dots, \pi_{E+R})$$

is a small formula derivation of $S \vdash f$, if it has the following structure.

- **Extension Phase:** *$\pi_1, \pi_2, \dots, \pi_E$ include all extension axioms used in the proof (but we allow them to be repetitive for convenience). Formally, for each $i \in [E]$, π_i is either*
  - *In the form $v \leftrightarrow a \circ b$ or $v \leftrightarrow c$, where v is a new variable that hasn't appeared in the proof so far, and a, b, c are either variables or their negations, and $\circ$ is $\wedge, \vee$, or $\rightarrow$.*
  - *Repeating some π_j where $j < i$, i.e. $\pi_i = \pi_j$.*

  Moreover, we require that the formulas in $\mathsf{Def}(f)$ and $\mathsf{Def}(\phi)$ for all $\phi \in S$ are "correctly" introduced. That is,

$$\mathsf{Def}(f) \cup \cup_{\phi \in S} \mathsf{Def}(\phi) \subseteq \{\pi_1, \pi_2, \dots, \pi_E\}.$$

- **Reasoning Phase:** *$\pi_{E+1}, \pi_{E+2}, \dots, \pi_{E+R}$ are variables in $\mathcal{EF}$. Moreover, each variable π_{E+i} is derived via one of the following cases.*
  - *π_{E+i} is in the form $\mathsf{ATOM}[\phi_i]$, where ϕ_i is a premise, i.e. $\phi_i \in S$.*
  - *There exists a universal constant c independent of Π, and c indices $i_1, i_2, \dots, i_c < i$ such that*

$$\pi_{i_1}, \pi_{i_2}, \dots, \pi_{i_c} \vdash \pi_{E+i}.$$

  Moreover, the last variable π_{E+R} is $\mathsf{ATOM}[t]$.

Definition 13 (Succinct Description of Small Formula Derivation)**.** *A succinct description of a small formula derivation $(\pi_1, \pi_2, \dots, \pi_E, \pi_{E+1}, \dots, \pi_{E+R})$ is a tuple of circuits* $(\mathsf{Get}, \mathsf{Where})$*. with the following syntax.*

- Get(i) : *Get circuit takes as input an index i, it outputs the representation of the i-th "line" in the proof $\ulcorner \pi_i \urcorner$. Recall that the* Gödel numbering $\ulcorner \cdot \urcorner$ *(see Section 3.1) is the natural way to represent the formulas in $\mathcal{EF}$ as binary strings.*
- Where$(\ulcorner v \urcorner)$*: It takes as input the representation of a variable v, and outputs a subset of indices $T \subseteq [E+R]$, which contains all indices t such that the variable v appears as a variable in π_t.*

We introduce the following complexity measure Desc$[\cdot]$ to characterize the size of the succinct description for Cook's propositional translation. For a high level explanation, see technical overview Section 2.3.

Description Size Desc$[\cdot]$**.** Intuitively, for any term t in PV, we will use Desc$[t]$ to denote the number of terms in PV that t depends on. Specifically, Desc$[t]$ is defined inductively as follows.

- If t is a variable or a function symbol of arity 0, then we define Desc$[t] := 1$.
- If t is of the form $f(t_1, \ldots, t_k)$, where f is a function symbol defined as $f(x_1, x_2, \ldots, x_k)$ in the variables $x_1, x_2, \ldots, x_k$, and $t_1, \ldots, t_k$ are terms. Then we define

  $$\mathrm{Desc}[t] := \mathrm{Desc}[t_1] + \mathrm{Desc}[t_2] + \ldots + \mathrm{Desc}[t_k] + \mathrm{Desc}[f] \cdot (|t_1| + |t_2| + \ldots + |t_k|),$$

  where $|t_i|$ represents the size of the binary encoding $\ulcorner t_i \urcorner$, and Desc$[f]$ for any function symbol f is defined as follows.
- Desc$[f]$ for a function symbol $f(x_1, \ldots, x_k)$ in the variables $x_1, \ldots, x_k$ is defined as follows.
  - If $f(x_1, \ldots, x_k)$ is defined as a term t' of the variables $x_1, \ldots, x_k$. Then we define

    $$\mathrm{Desc}[f] := \mathrm{Desc}[t'] + 1.$$
  - If $f(x_1, \ldots, x_k)$ is defined by limited recursion on notations, i.e.

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

    where $\mathbf{y} = (x_2, \ldots, x_k)$, and there exists k_1, k_2 such that $|h_i(x, y, z)| \leq |z| + |k_i(x, y)|$ can be proven in PV with proofs Π_1, Π_2, respectively. Then we define

    $$\mathrm{Desc}[f] := \mathrm{Desc}[g(\mathbf{x}_2)] + \mathrm{Desc}[h_1] + \mathrm{Desc}[h_2] + \mathrm{Desc}[\Pi_1] + \mathrm{Desc}[\Pi_2]$$

    where we will define Desc$[\cdot]$ for the proofs Π_1, Π_2 soon.
  - If $f(x_1, \ldots, x_k)$ is an initial function, then we define Desc$[\cdot]$ for them as follows. For s_1, s_2, we directly define Desc$[s_i] = 1$. For other initial functions which are defined recursively, we define Desc$[f] :=$ Desc$[g(\mathbf{x}_2)]$ + Desc$[h_1]$ + Desc$[h_2]$ similar to the previous case.

For any equation $t = u$, we define Desc$[t = u] :=$ Desc$[t]$ + Desc$[u] + 1$. Given a proof $\Pi = (\pi_1, \ldots, \pi_\ell)$ in PV, we define Desc$[\Pi] = \sum_{i=1}^{\ell}$ Desc$[\pi_i]$.

We prove the following theorem for the translation from Cook's theory PV to extended Frege system $\mathcal{EF}$.

Theorem 9 (Succinct Description of Cook's Translation)**.** *Let t, u be two terms in PV with $\vdash_{PV} t = u$, and let Π be the proof in PV, n be an integer and m be the bounding value for n related to $t = u$, then there exists a succinct description* (Get, Where) *of small formula derivation of $\vdash_{\mathcal{EF}} [\![t = u]\!]_m^n$. Moreover, the sizes of* Get, Where *are bounded by* poly(Desc$[\Pi]$, $\log m$)*, and the length of the $\mathcal{EF}$-proof they describe is* poly(Desc$[\Pi]$, m).

We defer the proof sketch to the Appendix A.

7.3 Δ-Equivalence for Turing Machines from Cook's Theory PV

In this subsection, we show that if the functional equivalence of two Turing machines can be proven in Cook's theory PV [Coo75], then the two Turing machines are Δ-equivalent. Cook [Coo75] showed that any polynomial-time function can be defined in PV.

Lemma 8. *There exists a padding algorithm* Pad *with the following properties.*

- *It takes as input a security parameter λ and a Turing machine M, and it outputs a new Turing machine M', which preserves the functionality of M for any input of length polynomial in λ.*
- *Let $f_1(x), f_2(x)$ be two function symbols that are definable in Cook's theory PV (See Section 3.2) with $\vdash_{PV}$ $f_1(x) = f_2(x)$. Let Π be a proof of $\vdash_{PV}$ $f_1(x) = f_2(x)$ and let M_1, M_2 be two Turing machines computing f_1, f_2 in a natural way, then* $\mathsf{Pad}(1^\lambda, M_1)$ *and* $\mathsf{Pad}(1^\lambda, M_2)$ *are Δ-equivalent Turing machines. Moreover, the size of the intermediate circuits $S(N)$ in the definition of Δ-equivalent circuit (see Definition 11) is bounded by* $\mathsf{poly}(|\mathsf{Desc}[\Pi]|, \log N)$.

The proof of Lemma 8 follows from the same idea as Lemma 3. The only difference is that here we further need to show that the intermediate circuits in the Δ-equivalent circuits (Definition 2) can be described *succinctly*. Hence, we only sketch the proof here, with the focus on how to describe the intermediate circuits by some small circuits.

Proof Sketch of Lemma 8. From the definition of Δ-equivalence of Turing machines, it suffices to show that for any integer N and any $n < N$, there exists a series of $\ell = \mathsf{poly}(n, \log N)$ small circuits $C'_1, C'_2, \dots, C'_\ell$ such that the circuits they describe $C_1, \dots, C_\ell$ are δ-equivalent for each two adjacent circuits.

The idea is to use the small circuit of size $\mathsf{poly}(\log m, \mathsf{Desc}[\Pi])$ provided by Theorem 9 to *succinctly describe* the circuit in each hybrid in Lemma 3. We briefly recall the proof of Lemma 3 here. We first use an algorithm Pad to pad the input circuit C by attaching the projection circuits Proj to the input wires of each gate and also attaching copy circuits Copy to the output wires of every gate. Both Proj and Copy are built as binary trees of gates. Next, we build a binary tree of AND gates, with all leaves outputting a value 1, except the first leaf outputting the output wire value of C. Then the proof of Lemma 3 shows a series of steps transforming from $\mathsf{Pad}(C_1)$ to $\mathsf{Pad}(C_2)$.

As our first step, we show how to represent each gate of the padded circuit in Lemma 3 as a binary string of $O(\log(m) + \log|\mathsf{Desc}[\Pi]|)$-bits. Next, we show how to represent the wires in the same amount of bits.

Numbering Gates. We classify the gates of the padded circuit $\mathsf{Pad}(C)$ in Lemma 3 into the following four types and assign each gate a succinct numbering. Let ℓ be the depth of the AND tree, and d be the depth of the Copy and Proj circuits.

- **Regular:** If the gate corresponds to a gate in the input circuit C or a dummy gate (See Figure 3). This kind of gates can be uniquely encoded by $(1, \mathsf{idx})$, where "1" represents the "regular" type, and idx is the number of the regular gates that has been added to the circuit before the current regular gate is added.
- **AND-Tree:** The gate is part of the binary tree that computes an multi-arity AND at the end. This type of gate can be uniquely encoded as $(2, i)$, where "2" indicates the "AND Tree" type and $i \in \{0,1\}^{\leq \ell}$ represents the location of the gate in the tree. Specifically, we use the empty string ϕ to denote the root node, and for each node i, we use $i||0$ to represent its left child and $i||1$ for its right child.

- **Copy:** This gate is part of the Copy circuit attached to the output wire of a regular gate or an input wire. Recall that the Copy circuit is built as a binary tree. Hence, this type of gate can be uniquely represented as $(3, g, i)$, where "3" represents the "copy" type, g is either in the form $(1, \mathsf{idx})$, which represents idx-th regular gate, or in the form $(0, \mathsf{idx})$, which represents the idx-th input wire. $i \in \{0,1\}^{\leq d}$ represents the location of the gate in the binary tree.
- **Projection:** The gate that is part of the projection circuit attached to the input wires of a regular gate or a leaf of the AND-tree. Hence, this type of gates can be uniquely represented as $(4, g, i)$, where "4" means "projection" type, and g is either of the form $(1, \mathsf{idx}, b)$ or $(2, \mathsf{idx})$, which represent the b-th input wire of a regular gate with index idx, or the idx-th leaf of the AND-tree, respectively. i represents that the gate is at the location $i \in \{0,1\}^{\leq d}$ of the Proj circuit.

Similarly, we also represent the wires in logarithmic bits.

Numbering Wires. The wires are classified as input and non-input wires. For non-input wires, they are uniquely determined by the gate that has such a wire as output. Specifically, their numbering are defined as follows.

- **Input Wires:** This type of wires can be uniquely represented as $(0, \mathsf{inp})$, where 0 represented "input" and inp is an index of the wire.
- **Non-Input Wires:** This type of wires can be uniquely encoded as $(1, g, b)$, where g is an encoding of a gate defined above, and $b \in \{0,1\}$ is a bit indicating the wire is the left or right output wire of g.

For the ease of presentation, we use $\ulcorner g \urcorner$ to denote the numbering of the gate g and use $\ulcorner w \urcorner$ to denote the numbering of the wire w.

Lengths of the Numbering. We will set the number of the regular gates and the number of leaf nodes of the AND-tree to be a polynomial of the $\mathcal{EF}$ proof length in Cook's translation of Π, which is $\mathsf{poly}(\log m + \log |\mathsf{Desc}[\Pi]|)$. Hence, their indices can be represented in $O(\log m + \log |\mathsf{Desc}[\Pi]|)$ bits. From our numbering, this implies that the encoding of the gates $\ulcorner g \urcorner$ and wires $\ulcorner w \urcorner$ can also be represented in $O(\log m + \log |\mathsf{Desc}[\Pi]|)$ bits.

Topology from Numbering. The topology of any gate g, which includes the numbering of its input and output wires, can be computed from the numbering $\ulcorner g \urcorner$ by a small circuit 'Topology' of size $\mathsf{poly}(\log m, \log |\mathsf{Desc}[\Pi]|)$.

For the regular gates and the gates in AND-Tree, this computation is straightforward. Here we emphasis on the topology of the gates in Copy and Proj circuits. For the leaves with index $i \in \{0,1\}^d$ in Copy circuits attached to the output wire of the idx-th regular gate, we set the output wires of idx as follows. Let E be the total number of regular gates.

- If $i \leq E$ and $i > \mathsf{idx}$, we let the number of output wires of such a gate to be 2. For $b = 0, 1$, we set the b-th output wire of the leaf as the input wires of the idx-th leaf node of the Proj circuit attached to the b-th input wire of the i-th regular gate. Here, we put the restriction "$i > \mathsf{idx}$" to ensure that the circuit can be executed in the order of index of the regular gates.
- Otherwise, if $i > E$, we let the number of output wires of such a gate be 1. Then we set the output wire of the leaf as the input wire of the idx-th leaf node of the Proj circuit attached to the $(i - E)$-th leaf node of the AND-tree.

For other cases of the leaves in the Copy circuit, we don't set any output wires.

Succinct Description of the Hybrids. We first represent the padding circuit $\mathsf{Pad}(\cdot, \cdot)$ in Lemma 3 succinctly as the following small circuit SPad. It takes as input a Turing machine M, an input length n, and the numbering of a gate g, it calculates the topology of the gate g, which includes the numbering of its input and output wires, and also computes the functionality f_g of the gate g. For the ease of presentation, we further require that for idx-th input wire, $\mathsf{out}_{\mathsf{idx}}$ outputted by $[\![M]\!]_N(n, \mathsf{idx})$ contains the indices of all gates which uses the idx-input wire as input.

Succinct Padding $\mathsf{SPad}(1^\lambda, M, n, \ulcorner g \urcorner)$:

Input: A Turing machine M, a input length n and the numbering of a gate g.

- Check if $\ulcorner g \urcorner$ is well-formed. If it's not, then output $\perp$ and abort.
- Compute the topology of the gate g in the same way as the proof of Lemma 3, as described in the above 'Topology from Numbering' paragraph, i.e., compute $(\mathsf{inp}_g, \mathsf{out}_g) \leftarrow \mathsf{Topology}(\ulcorner g \urcorner)$.
- Set $N = \lambda^{\log \lambda}$ to be a slight super-polynomial. Then depending on the type of g, we set the functionality f_g of the gate g as follows.
  - **Regular:** Parse $\ulcorner g \urcorner$ as $(1, \mathsf{idx})$. Recall that we use $T(n)$ to denote the size of the circuit that $[\![M]\!]_N(n, \cdot)$ describes. Hence, if $\mathsf{idx} > T(n)$, then we set $f_g = 0$, since the gate g is a "dummy" gate.

    Otherwise, when $\mathsf{idx} \le T(n)$, we compute the topology of the idx-th gate of the circuit computing M as $(\mathsf{inp}_{\mathsf{idx}}, \mathsf{out}_{\mathsf{idx}}, f_{\mathsf{idx}}) \leftarrow [\![M]\!]_N(n, \mathsf{idx})$, and let $f_g = f_{\mathsf{idx}}$.
  - **AND-Tree:** Parse $\ulcorner g \urcorner$ as $(2, \mathsf{idx})$, where $\mathsf{idx} \in \{0,1\}^{\le \ell}$. Note that $|\mathsf{idx}| = \ell$ if and only if g is a leaf node of the AND-tree. We have the following cases.
    * First Leaf: ($\mathsf{idx} = 0^\ell$), we set $f_g(x) = x$.
    * Other Leaves: ($\mathsf{idx} \ne 0^\ell$ and $|\mathsf{idx}| = \ell$), we set $f_g = 1$.
    * Non-Leaf: ($|\mathsf{idx}| < \ell$), we set $f_g = \wedge$ to be an AND gate.
  - **Copy:** Parse $\ulcorner g \urcorner$ as $(3, g', i)$, where $i \in \{0,1\}^{\le d}$. Parse g' as $(b \in \{0,1\}, \mathsf{idx})$, then compute the topology of idx-th gate of M or the idx-th input wire of M as

    $$(\mathsf{inp}_{\mathsf{idx}}, \mathsf{out}_{\mathsf{idx}}, f_{\mathsf{idx}}) \leftarrow [\![M]\!]_N(n, \mathsf{idx}).$$

    Here we assume $\mathsf{out}_{\mathsf{idx}}$ is the set of gate indices that takes the idx-th input wire as an input, for every $\mathsf{idx} \in [n]$. And the actual gates in $[\![M]\!]_n$ are indexed starting from $n+1$. If $\mathsf{out}_{\mathsf{idx}} \cap [i||0^{d-|i|}, i||1^{d-|i|}] \ne \phi$, then let $f_g(x) = (x, x)$. Else, let $f_g(x) = (0, 0)$. Here we assume that $[\![M]\!]_N(n, \mathsf{idx})$ outputs the numbering of the output wires in .
  - **Projection:** Parse $\ulcorner g \urcorner$ as $(4, g', i)$, where $i \in \{0,1\}^{\le d}$. Depending on the type of g', we have two cases.
    * If g' is the form $(1, \mathsf{idx}, b)$, i.e. g is a gate in the projection circuit attached to the b-th input wire of idx-th regular gate. If $\mathsf{idx} > T(n)$, then we set $f_g = 0$, since the idx-th gate is a "dummy" gate. Otherwise, we compute the f_g as follows. Let

      $$(\mathsf{inp}_{\mathsf{idx}}, \mathsf{out}_{\mathsf{idx}}, f_{\mathsf{idx}}) \leftarrow [\![M]\!]_N(n, \mathsf{idx}).$$

      Let $\mathsf{inp}_{\mathsf{idx}}[b]$ be the b-th element of the array $\mathsf{inp}_{\mathsf{idx}}$. If $\mathsf{inp}_{\mathsf{idx}}[b] \in [i||0||0^{d-|i|-1}, i||0||1^{d-|i|-1}]$, then it means that the b-th input wire is on the "left child" of g. Hence, we let $f_g(x_0, x_1) = x_0$. Otherwise, b-th input wire is on the "right child" of g, we let $f_g(x_0, x_1) = x_1$.

    * If g' is the form $(2, \mathsf{idx})$ where $\mathsf{idx} \in \{0,1\}^\ell$, i.e. g is a gate in the projection circuit attached to the idx-th leaf of the AND-tree. Let o be the index of the output gate of the circuit described by $[\![M]\!]_N(n, \cdot)$. We will have $\ulcorner o \urcorner$ hardwired in the circuit. Next, we have the following two cases.
        - First Leaf: ($\mathsf{idx} = 0^\ell$), if $\ulcorner o \urcorner \in [i||0||0^{d-|i|-1}, i||0||1^{d-|i|-1}]$, then this means that o can be reached from the "left child", and hence we set $f_g(x_0, x_1) = x_0$. Otherwise, we set $f_g(x_0, x_1) = x_1$.
        - Other Leaves: ($\mathsf{idx} \neq 0^\ell$), let $f_g = 0$.

- Output $(\mathsf{inp}_g, \mathsf{out}_g, f_g)$.

Now for each step i in the transformation in Lemma 3, we describe the following small circuit C'_i that takes an encoding of a gate (described above). By Theorem 9, there exists a succinct description $(\mathsf{Get}, \mathsf{Where})$ of small derivation of $\vdash_{\mathcal{EF}} [\![f_1(x) = f_2(x)]\!]^n_m$. Let $\Pi = (\pi_1, \pi_2, \ldots, \pi_E, \pi_{E+1}, \ldots \pi_{E+R})$ be the small derivation, where $(\pi_1, \ldots, \pi_E)$ is the extension phase and $(\pi_{E+1}, \ldots, \pi_{E+R})$ is the reasoning phase.

"Grow C_2" Phase and "Grow the Extension" Phase. We without loss of generality assume that there exist $T_1, T_2 \le E$ such that $\pi_1, \pi_2, \ldots, \pi_{T_1}$ are the translation of the definition of $f_1(x)$, and $\pi_{T_1+1}, \pi_{T_1+2}, \ldots, \pi_{T_1+T_2}$ are the translation of the definition of $f_2(x)$. Now, we build the following intermediate circuits $C^{I,II}_{i^*}(\cdot)$ for each $i^* \in [E - T_1]$. In $C^{I,II}_{i^*}(\cdot)$, the first $T_1 + i^*$ regular gates corresponds to the extensions $\pi_1, \pi_2, \ldots, \pi_{T_1+i^*}$. The rest of the regular gates remains dummy gates that always output 0.

Succinct Description of the Intermediate Circuits in Phase I, II: $C^{I,II}_{i^*}(\ulcorner g \urcorner)$:

Input: The numbering of a gate g.
Hardwire: Input length n, succinct description $(\mathsf{Get}, \mathsf{Where})$, and an index of the circuits $i^* \in [E - T_1]$.

- Check whether $\ulcorner g \urcorner$ is well-formed or not, and compute the topology of the gate g using the circuit Topology in the same way as SPad above. Let $\mathsf{inp}_g, \mathsf{out}_g$ be the output of Topology($\ulcorner g \urcorner$).

  Then depending on the type of g, we have the following cases.

  - **Regular:** Parse $\ulcorner g \urcorner$ as $(1, \mathsf{idx})$. Note that at the i-th hybrid, only the first $T_1 + i^*$ regular gates should be used. Hence, if $\mathsf{idx} > T_1 + i^*$, then we simply set $f_g = 0$.

    Otherwise, in the case $\mathsf{idx} \le T_1 + i^*$, instead of computing the topology of g via $[\![M]\!]_N(n, \cdot)$, here we execute $\ulcorner \pi_{\mathsf{idx}} \urcorner \leftarrow \mathsf{Get}(\mathsf{idx})$ to obtain π_{idx}. Since π_{idx} is in the extension phase, we can further parse it as $v \leftrightarrow t$, where v is a variable, and t is a formula. Then we set f_g be the function that compute t. That is, let t be the form $v_1 \circ v_2$, where v_1, v_2 are variables and $\circ$ is either $\wedge, \vee$, or $\to$, then we set $f_g(x_0, x_1) = x_0 \circ x_1$.
  - **AND-Tree:** The AND-tree is the same as SPad above.
  - **Copy:** The copy circuit is almost the same as SPad, except that we obtain $\mathsf{out}_{\mathsf{idx}}$ as follows.

$$\ulcorner v \leftrightarrow t \urcorner \leftarrow \mathsf{Get}(\mathsf{idx}), \quad T \leftarrow \mathsf{Where}(\ulcorner v \urcorner),$$

    and we set $\mathsf{out}_{\mathsf{idx}} = T \cap (\mathsf{idx}, T_1 + i^*]$ be the indices in T that falls in $[T_1 + i^*]$, but are larger than the current gate idx. We add this condition to ensure that the output of idx-gate can only be used as input to the gates whose indices are larger than idx, so that the circuit is an *acyclic* graph.
  - **Projection:** The projection circuit is also almost the same as the succinct description of SPad above, except that in the case when the projection circuit is attached to the output wire of a

regular gate, we compute the input wires of $\mathsf{inp}_{\mathsf{idx}}$ as follows: let $\ulcorner v \leftrightarrow t \urcorner \leftarrow \mathsf{Get}(\mathsf{idx})$, and $\{v_s\}_{s\in[T]}$ be the variables in t. We let the input wire set $\mathsf{inp}_{\mathsf{idx}}$ be the set contains the indices where v_t is introduced for the first time. Specifically,

$$\mathsf{inp}_{\mathsf{idx}} = \{\mathsf{Def}_s \mid s \in [T]\},$$

where $\mathsf{Def}_s = \min \mathsf{Where}(\ulcorner v_s \urcorner)$ is the index where v_s is introduced by the extension axiom. Here we take the first indices that v_s appears, since we allow repetition in the extension phase (See Definition 12), and thus v_s could appear many times as the same extension. Note that we only need to compute $\mathsf{inp}_{\mathsf{idx}}$ above when $\mathsf{idx} \leq T_1 + i^*$. Otherwise, if $\mathsf{idx} > T_1 + i^*$, we directly set $f_g = 0$ since idx corresponds to a "dummy" gate that has not been used.
If the projection circuit is attached to a leaf of the AND-Tree, then f_g is defined in the same way as SPad above.

– Finally, we output $(\mathsf{inp}_g, \mathsf{out}_g, f_g)$.

For other phases, the idea to construct the succinct description of the intermediate circuits is the same as above, except that the detail is more involved, and hence are omitted. □

7.4 Construction of iO for Turing machines

In this section, we present our construction of iO for Δ-equivalent Turing machines (See Definition 11).

Ingredients. Before we present our construction, we list the following necessary ingredients.

– Sub-exponentially secure puncturable PRF $\mathsf{PRF}_{punc}(\cdot,\cdot)$.

– Sub-exponentially secure pseudorandom generator PRG.

– Fully homomorphic encryption (FHE) $\mathsf{FHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$.

– Somewhere extractable hash with consistent proof $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext}, \mathsf{P}, \mathsf{V})$.

– A circuit Gate in Figure 7 emulating the computation at each gate for the input circuit.

– A circuit $\mathsf{Shrink}_{[H]}(\cdot,\cdot)$ in Figure 8 that decrypts the symmetric key encryption inside the FHE, and thus shrink the size of the somewhere extractable hash value H.

– Sub-exponentially secure indistinguishable obfuscation scheme iO.

– The Gate algorithm in Figure 7.

We present our construction in Figure 11.

Lemma 9. *The* ΔiO *algorithm in Figure 11 satisfies indistinguishability for any two Δ-equivalent Turing machines M_1, M_2.*

Proof. For the ease of presentation, for any subcircuit S, we define the *closure* of S denoted as $\overline{S}$ to be a set $\overline{S} = \bigcup_{g\in S} \mathsf{inp}(g) \cup \mathsf{out}(g)$. To prove the construction satisfies indistinguishability, we construct the following hybrids.

Hybrid H_0**:** This hybrid outputs $\Delta\mathsf{iO}(1^\lambda, M_1)$.

Hybrid $\mathsf{H}_1^{n_0}$ **:** This hybrid is almost the same as H_0, except that we modify the uniform gate UGate as follows. When compute the gate information for g, it compares whether n with n_0 and decides to use M_1 or M_2.

ΔiO for Δ-equivalent Turing Machines: $\Delta\mathsf{iO}(1^\lambda, M)$

- Let $N_0 = \lambda^{\log\lambda}$ be two values that are superpolynomial in λ, and let $S_0 = \log^2 N_0$ be an upper bound for the number of elements we will extract from SEH.
- Generate a somewhere extractable hash key $K \leftarrow \mathsf{Gen}(1^\lambda, 1^{S_0})$.
- Generate a puncturable PRF key $\mathsf{PRF}_{punc}.K$. We will use puncturable PRF to generate the encryption keys and MAC keys for each gate.
- Generate a FHE key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$, and let $\widetilde{\mathsf{td}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{|\mathsf{td}|})$. For each $i \in [S_0]$, let $\widetilde{K_i} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{|K_i^m|})$.
- Then we obfuscate the circuit UGate

$$\widetilde{\mathsf{UGate}} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{UGate}_{[K, \mathsf{PRF}_{punc}.K, \mathsf{pk}, \widetilde{\mathsf{td}}, \{\widetilde{K_i}\}_{i\in[S_0]}]}).$$

- Output $(\widetilde{\mathsf{UGate}}, K, \mathsf{pk})$.

Figure 11: Description of the iO for Δ-equivalent Turing machines, where UGate in Figure 12 is an uniform way to describe Gate.

$\mathsf{UGate}_{[K, \mathsf{PRF}_{punc}.K, \mathsf{pk}, \widetilde{\mathsf{td}}, \{\widetilde{K_i}\}_{i\in[S_0]}]}(n, g, \overrightarrow{\mathsf{input}} = (H, \{H_i, \mathsf{ct}_i, \rho_i, \pi_i, \sigma_i\}_{i\in[2]}))$

- Obtain the description of the g-th gate

$$(\mathsf{inp}_g, \mathsf{out}_g, \{f_o\}_{o\in\mathsf{out}_g}) \leftarrow [\![M]\!]_{N_0}(n, g).$$

- Generate the keys for the input/output wires of the g-th gate using puncturable PRF, with the input n concatenated with the wire indices.

$$\forall w \in \mathsf{inp}_g \cup \mathsf{out}_g, \quad (K_w^m, K_w^\sigma) \leftarrow \mathsf{PRF}_{punc}(\mathsf{PRF}_{punc}.K, (n, w)).$$

- Use Gate to emulate the g-th gate,

$$\overrightarrow{\mathsf{output}} \leftarrow \mathsf{Gate}_{[K, \{K_i^m, K_i^\sigma\}_{i\in\mathsf{inp}_g}, \{f_o, K_o^m, K_o^\sigma\}_{o\in\mathsf{out}_g}, \widetilde{\mathsf{td}}, \{\widetilde{K_i}\}_{i\in[S_0]}]}(\overrightarrow{\mathsf{input}}).$$

- Output $\overrightarrow{\mathsf{output}}$.

Figure 12: Description of UGate, which is an uniform way to compute Gate.

- If $n < n_0$, it uses M_2 to obtain the description of the g-th gate

$$(\mathsf{inp}_g, \mathsf{out}_g, \{f_o\}_{o\in\mathsf{out}_g}) \leftarrow [\![M_2]\!]_{N_0}(n, g).$$

- Otherwise, $n \geq n_0$, it uses M_1 to generate the description of the g-th gate

$$(\mathsf{inp}_g, \mathsf{out}_g, \{f_o\}_{o \in \mathsf{out}_g}) \leftarrow [\![M_1]\!]_{N_0}(n, g).$$

- ...

Clearly, when $n_0 = 0$, then the functionality of UGate is the same as H_0. Next, we're going to argue the indistinguishability between the Hybrid $\mathsf{H}_1^{n_0}$ and $\mathsf{H}_1^{n_0+1}$ via a series of hybrids.

In the next hybrid we will use the Δ-equivalence of the Turing machines M_1, M_2. Since M_1, M_2 are Δ-equivalent, by definition, for the integer $n_0 < N_0$, there exists a series of circuits $C_1', C_2', \dots, C_\ell'$ that satisfies the properties in the definition.

Hybrid $\mathsf{H}_{1,i}^{n_0}$: This hybrid is almost the same as $\mathsf{H}_1^{n_0}$, except that we modify the uniform gate UGate as the following circuit with C_i' hardwired.

- If $n < n_0$, it uses M_2 to obtain the description of the g-th gate

$$(\mathsf{inp}_g, \mathsf{out}_g, \{f_o\}_{o \in \mathsf{out}_g}) \leftarrow [\![M_2]\!]_{N_0}(n, g).$$

- If $n = n_0$, then we compute the g-th gate as follows.

$$(\mathsf{inp}_g, \mathsf{out}_g, \{f_o\}_{o \in \mathsf{out}_g}) \leftarrow C_i'(g).$$

- If $n > n_0$, it uses M_1 to generate the description of the g-th gate

$$(\mathsf{inp}_g, \mathsf{out}_g, \{f_o\}_{o \in \mathsf{out}_g}) \leftarrow [\![M_1]\!]_{N_0}(n, g).$$

- ...

Clearly, for any n_0, Hybrid $\mathsf{H}_{1,i=1}^{n_0}$ and the last hybrid $\mathsf{H}_1^{n_0}$ are indistinguishable, since the functionalities of UGate in both hybrids are identical, because $C_1'(\cdot)$ and $[\![M_1]\!]_{N_0,T_0}(n_0, \cdot)$ have the same functionality. Next, we are going to show that the Hybrid $\mathsf{H}_{1,i}^{n_0}$ and the Hybrid $\mathsf{H}_{1,i+1}^{n_0}$ are indistinguishable. We show this by firstly puncture the points in $\{n_0\} \times \overline{S}$ in the puncturable PRF.

Hybrid $\mathsf{H}_{1,i,1}^{n_0}$: This hybrid is almost the same as the last hybrid $\mathsf{H}_{1,i}^{n_0}$, except that we replace the PRF evaluation as the puncturable PRF.

- ...(Obtain the description for the g-th gate)...
- For each $w \in \mathsf{inp}_g \cup \mathsf{out}_g$, generate the encryption key and MAC key for w as follows.
  - If $n = n_0$ and $w \in \overline{S}$, then let

$$(K_w^m, K_w^\sigma) \leftarrow \mathsf{PRF}_{punc}(\mathsf{PRF}punc.K, (n_0, w)).$$

  - Otherwise, let

$$(K_w^m, K_w^\sigma) \leftarrow \mathsf{PRF}_{punc}(\mathsf{PRF}punc.K \setminus \{n_0\} \times \overline{S}, (n_0, w)).$$

- ...

By the functionality preservation property of puncturable PRF, this modification does not change the functionality of the circuit UGate. Hence, the current hybrid $\mathsf{H}_{1,i,1}^{n_0}$ and last hybrid $\mathsf{H}_{1,i}^{n_0}$ are indistinguishable by the security of iO.

Hybrid $\mathsf{H}_{1,i,2}^{n_0}$: This hybrid is almost the same as the previous hybrid $\mathsf{H}_{1,i,1}^{n_0}$, except that we replace the PRF evaluation at the punctured point with random values $\{K_w^m, K_w^\sigma\}_{w\in\overline{S}}$, and these random values are hardwired in the circuit description. This hybrid is indistinguishable with the previous hybrid by the pseudorandomness at the punctured point.

Hybrid $\mathsf{H}_{1,i,3}^{n_0}$: This hybrid is almost the same the last hybrid, except that for each $g \in S_i$, we generate

$$\widetilde{\mathsf{Gate}}_g \leftarrow \mathsf{iO}(\mathsf{Gate}_{[K,\{K_i^m,K_i^\sigma\}_{i\in\mathsf{inp}(g)},\{f_o,K_o^m,K_o^\sigma\}_{o\in\mathsf{out}_g},\widetilde{\mathsf{td}},\{\widetilde{K}_i\}_{i\in[S_0]}]]})$$

at the outside, and hardwire these circuits in UGate. Then we change UGate as follows. In this hybrid, we *only* hardwire uniformly random $\{K_w^m, K_w^\sigma\}_{w\in\partial S}$, where $\partial S = \mathsf{inp}(S) \cup \mathsf{out}(S)$.

- …(Obtain gate information for g) …
- For each $w \in \mathsf{inp}_g \cup \mathsf{out}_g$, if $n \notin n_0$, then compute K_w^m, K_w^σ via puncturable PRF. Otherwise, if $n = n_0$ and $w \in \partial S$, then take K_w^m, K_w^σ from the hardwired values in the circuit description. Note that here we do nothing when $n = n_0$ and $w \in \overline{S} \setminus \partial S$.
- If $g \in S$, let $\overrightarrow{\mathsf{output}} \leftarrow \widetilde{\mathsf{Gate}}_g(\overrightarrow{\mathsf{input}})$.
- Otherwise, $g \notin S$, we compute the output in the same way as before,

$$\overrightarrow{\mathsf{output}} \leftarrow \mathsf{Gate}_{[K,\{K_i^m,K_i^\sigma\}_{i\in\mathsf{inp}_g},\{f_o,K_o^m,K_o^\sigma\}_{o\in\mathsf{out}_g},\widetilde{\mathsf{td}},\{\widetilde{K}_i\}_{i\in[S_0]}]}(\overrightarrow{\mathsf{input}}).$$

This hybrid is indistinguishable with the previous hybrid, since iO preserves the functionality of Gate, and hence the functionality of UGate is the same as before, and so the indistinguishability follows from the security of iO.

Hybrid $\mathsf{H}_{1,i,4}^{n_0}$: This hybrid is the almost the same as the last hybrid, except that when we generate $\widetilde{\mathsf{Gate}}_g$, we generate them with the gate information f_o' that is computed by C_{i+1}'. Namely,

$$\forall g \in S, (\mathsf{inp}_g, \mathsf{out}_g, \{f_o'\}_{o\in\mathsf{out}_g}) \leftarrow C_{i+1}'(g).$$

The indistinguishability of this hybrid and the last hybrid follows from Lemma 7.

From Hybrid $\mathsf{H}_{1,i,4}^{n_0}$ to $\mathsf{H}_{1,i+1}^{n_0}$: Note that the Hybrid $\mathsf{H}_{1,i,4}^{n_0}$ and the hybrid $\mathsf{H}_{1,i,3}^{n_0}$ are symmetric: everything is the same except that $\mathsf{H}_{1,i,3}^{n_0}$ generates $\widetilde{\mathsf{Gate}}_g$ using C_i', while $\mathsf{H}_{1,i,4}^{n_0}$ generates it using C_{i+1}'. Hence, we can prove that $\mathsf{H}_{1,i,4}^{n_0}$ and $\mathsf{H}_{1,i+1}^{n_0}$ are indistinguishable, by arguing in the reverse order from $\mathsf{H}_{1,i}^{n_0}$ to $\mathsf{H}_{1,i,3}^{n_0}$.

From Hybrid $\mathsf{H}_1^{n_0}$ to $\mathsf{H}_1^{n_0+1}$. By the hybrid argument, $\mathsf{H}_{1,i}^{n_0}$ and $\mathsf{H}_{1,i+1}^{n_0}$ are indistinguishable. Hence, by using hybrid argument again, $\mathsf{H}_{1,1}^{n_0}$ and $\mathsf{H}_{1,\ell}^{n_0}$ are indistinguishable. Since in $\mathsf{H}_{1,\ell}^{n_0}$, the circuit UGate computes the same functionality as $\mathsf{H}_1^{n_0+1}$, we prove that $\mathsf{H}_1^{n_0}$ and $\mathsf{H}_1^{n_0+1}$ are indistinguishable. We finish the proof.

Hybrid H_2. This hybrid is almost the same as H_1, except that we use M_2 to compute the description of the gates. This hybrid is indistinguishable with $\mathsf{H}_1^{n_0+1}$ by the security of iO, since the UGate in $\mathsf{H}_1^{n_0+1}$ and H_2 computes the same functionality.

By the hybrid argument, we finish the proof. □

The following theorem is straightforward combining Lemma 8 and Lemma 9.

Theorem 10 (iO for Turing Machines with PV-proof of Equivalence)**.** *Assuming quasi-polynomial hardness of LWE, sub-exponential hardness of one-way functions, and sub-exponential security of iO, there exists a construction of iO for* unbounded-input *Turing machines* M_1, M_2 *with* $\vdash_{PV} M_1(x) = M_2(x)$*. Moreover, let* Π *be the* PV*-proof of* $M_1(x) = M_2(x)$*, then the obfuscated Turing machine has size* $\mathsf{poly}(\lambda, |\mathsf{Desc}[\Pi]|, \log m(\lambda^{\log \lambda}))$*, where* $m(\cdot)$ *is the bounding value for* Π*.*

We stress that usually the proof for functionality equivalence we write is highly "uniform". That is, there are only a constant number of function symbols are introduced in Π (instead of $\mathsf{poly}(\lambda)$ number of function symbols), then $|\mathsf{Desc}[\Pi]| = \mathsf{poly}(|\Pi|)$, where we use $|\Pi|$ to denote the length of the binary string encoding Π. Moreover, in this case $m(\cdot)$ is a polynomial in λ. Hence, we have the following corollary.

Corollary 2 (iO for Turing machines with PV-proof of Equivalence, Simplified)**.** *Under the same assumptions as Theorem 10, there exists iO for unbounded-input Turing machines* M_1, M_2 *with obfuscated program size* $\mathsf{poly}(\lambda, |\Pi|)$*, where* Π *is a proof in* PV *for* $M_1(x) = M_2(x)$ *with a constant number of function symbols.*

8 Applications

In this section, we show some examples of witness encryption. We first recall the definition.

Definition 14. *A witness encryption for an* $\mathcal{NP}$ *language* L *is a pair of algorithms* $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ *with the following syntax.*

- **Encryption** $\mathsf{Enc}(1^\lambda, x, m)$**:** *It takes a security parameter* λ*, an instance* $x \in \{0,1\}^*$*, and a message* m *as input, and outputs a ciphertext* ct*.*
- **Decryption** $\mathsf{Dec}(\mathsf{ct}, w)$**:** *It takes a ciphertext* ct *and a witness* w*. If* w *is a witness for* x*, then it outputs a message* m'*.*

Furthermore, we require it to satisfy the following properties.

- **Correctness.** *For any instance* $x \in L$*, and any witness* w *of* x*,*
$$\Pr[\mathsf{Dec}(\mathsf{Enc}(1^\lambda, x, m), w) = m] = 1.$$
- **Indistinguishability Security.** *For any instance* $x \notin L$*, and any two messages* m_0, m_1*, we have*
$$\{\mathsf{Enc}(1^\lambda, x, m_0)\}_{\lambda \in \mathbb{N}} \approx \{\mathsf{Enc}(1^\lambda, x, m_1)\}_{\lambda \in \mathbb{N}}.$$

8.1 Witness Encryption for Circuits

We build witness encryption for a large class of languages in $\mathcal{NP} \cap co\mathcal{NP}$, without relying on the hardness of the assumption that is proportional to the witness length. Specifically, we build witness encryption for the language L such that $L \in co\mathcal{NP}$ can be proved in propositional logic. Formally, we define it as follows.

Definition 15 (Propositional Proof of Disjointness)**.** *Let* L *be an* $\mathcal{NP} \cap co\mathcal{NP}$ *language, where* $\{C_n\}_{n\in\mathbb{N}}$ *and* $\{\overline{C}_n\}_{n\in\mathbb{N}}$ *are the relation circuit families for* L *and* $\overline{L}$ *respectively. We say that* L *has propositional logic proof of disjointness, if there exists an polynomial-size propositional logic proof of* $\vdash \overline{C}_n(x, \overline{w}) \to \neg C_n(x, w)$*.*

We build witness encryption for all languages in $\mathcal{NP} \cap co\mathcal{NP}$ with propositional proof of disjointness, without the security parameter independent of n. Our construction is generic from our Theorem 8.

Corollary 3. *Under the same assumptions of Theorem 8, there exists a witness encryption with security parameters independent of the witness length for any language with propositional proof of disjointness.*

Proof Sketch. The idea is to follow the witness encryption construction from iO in the previous works [SW21]. Namely, to encrypt a message m under an instance x, we compute the ciphertext as the obfuscation of a circuit $G_{[x,m]}(w)$ that takes the w as input and outputs the message m if $C_n(x, w) = 1$. Otherwise, it outputs 0.

To prove indistinguishability security, we use Corollary 1. We need to show that if $x \notin L$, then there exists an *efficient propositional proof of equivalence* between $G_{[x,m]}$ and another circuit G' that always outputs 0. The existence of such an efficient propositional proof follows directly from the propositional proof of disjointness in Definition 15, and a witness $\overline{w}$ for any $x \notin L$. Then by Corollary 1, if $x \notin L$, the ciphertext is indistinguishable with the obfuscation of G'. Hence, the indistinguishability security of the witness encryption follows. □

Example: Commit-and-Prove. As a concrete example, we consider the following commit-and-prove language for the commitment scheme instantiated by a public key encryption $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Let n be an integer, and $\mathsf{pk} \leftarrow \mathsf{Gen}(1^n)$ be a public key. We define the following $\mathcal{NP}$ language

$$L_{\mathsf{pk}} = \{(c, f) \mid \exists (m, r) : c = \mathsf{Enc}(\mathsf{pk}, m; r) \wedge f(m) = 1\},$$

where pk is a public key generated by Gen, and f is any Boolean circuit. To establish our instantiation, We require the public key encryption to satisfy the following property. We will show that this property is naturally achieved by the ElGamal encryption scheme from DDH [ElG85] and Regev's encryption from lattices [Reg05].

Definition 16 (*PV*-Proof of Correctness)**.** *We say a public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has PV-proof of correctness, if its correctness can be proven in PV (or PV), i.e.,* Gen, Enc, Dec *can be formalized in PV as function symbols, and*

$$\vdash_{PV} (\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^n; r') \rightarrow \mathsf{Dec}(\mathsf{Enc}(\mathsf{pk}, m; r), \mathsf{sk}) = m$$

The following lemma shows that *PV*-proof of correctness implies propositional proof of disjointness.

Lemma 10. *If the public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has PV-proof of correctness, then* L_{pk} *is a language in* $\mathcal{NP} \cap co\mathcal{NP}$ *with* propositional proof of disjointness.

Proof. We first describe how to construct the circuits C_n and $\overline{C}_n$. We construct C_n naturally from the definition of L_{pk}. Namely, $C_n((c, f), (m, r))$ takes as input (m, r) as witness, and verifies whether $c = \mathsf{Enc}(\mathsf{pk}, m; r)$ and $f(m) = 1$. For $\overline{C}_n((c, f))$, we compute $\neg f(\mathsf{Dec}(c, \mathsf{sk}))$.

Now we need to show that $\vdash \overline{C}_n(x) \rightarrow \neg C_n(x, w)$ has polynomial size propositional logic proof, where $x = (c, f)$ and $w = (m, r)$. We first prove it in *PV* that $\vdash_{PV} \overline{C}_n(x) = 1 \rightarrow \neg C_n(x, w) = 1$ as follows.

1. $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^n; r') \rightarrow \mathsf{Dec}(\mathsf{Enc}(\mathsf{pk}, m; r), \mathsf{sk}) = m$, where pk, sk are any concrete *numerals* generated honestly by $\mathsf{Gen}(1^n; r')$, and r is also a concrete numeral. (See Section 3.2 for definition of numerals). This follows from the *PV*-proof of correctness and substitution rule (See Section 3.3 for substitution in inference rules).

2. $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^n; r')$. This follows from the fact that pk, sk are generated honestly from Gen with randomness r'.

3. $\mathsf{Dec}(\mathsf{Enc}(\mathsf{pk}, m; r), \mathsf{sk}) = m$. This follows from 1 and 2 and the implication rule.

4. $C_n((c, f), (m, r)) = 1 \rightarrow c = \mathsf{Enc}(\mathsf{pk}, m; r) \wedge f(m) = 1$. (Follows from the defining function of C_n.)

5. $c = \mathsf{Enc}(\mathsf{pk}, m; r) \rightarrow \mathsf{Dec}(c, \mathsf{sk}) = \mathsf{Dec}(\mathsf{Enc}(\mathsf{pk}, m; r), \mathsf{sk})$ (Follows from E_4).

6. $C_n((c,f),(m,r)) = 1 \rightarrow m = \mathrm{Dec}(c,\mathrm{sk}) \wedge f(m) = 1$. (Follows from 3, 4, and 5).

7. $m = \mathrm{Dec}(c,\mathrm{sk}) \rightarrow f(m) = f(\mathrm{Dec}(c,\mathrm{sk}))$ (Follows from E_4).

8. $C_n((c,f),(m,r)) = 1 \rightarrow f(\mathrm{Dec}(c,\mathrm{sk})) = 1$ (Combining 6 and 7).

9. $f(\mathrm{Dec}(c,\mathrm{sk})) = 1 \rightarrow \overline{C}_n((c,f)) = 0$ (From the defining function of $\overline{C}_n$).

10. $C_n((c,f),(m,r)) = 1 \rightarrow \neg\overline{C}_n((c,f)) = 1$ (Combining 8 and 9).

11. $\overline{C}_n((c,f)) = 1 \rightarrow \neg C_n((c,f),(m,r)) = 1$ (Follows from 10).

By Cook's translation (See Section 3.2, propositional translation), any proof in PV (or PV_1) can be translated to a propositional proof of polynomial size. Hence, there exists a polynomial size proof in $\mathcal{EF}$ for $\vdash_{\mathcal{EF}} \overline{C}_n(x) \rightarrow \neg C_n(x,w)$, where $x = (c,f)$ and $w = (m,r)$. □

We now instantiate above public key encryption scheme in group-based cryptography and lattice-based cryptography, and show that they satisfies PV-proof of correctness.

Groups-based Instantiation. We first instantiate the above examples from group-based cryptography. As an example, we consider the ElGamal encryption scheme [ElG85].

Recall that, the public key of ElGamal encryption is a group element $\mathrm{pk} = h = g^s$, where s is the secret key. To encrypt a message $m \in \{0,1\}$, the encryption algorithm $\mathsf{Enc}(\mathrm{pk}, m; r)$ uses the randomness r to compute and output $c = (g^r, h^r \cdot m)$ as the ciphertext. To decrypt the ciphertext $c = (c_1, c_2)$, the decryption algorithm computes c_2/c_1^s to obtain m.

Lemma 11. *ElGamal encryption scheme has PV-proof of correctness.*

Proof Sketch. We now prove its correctness in PV via the following steps.

1. $(h,s) = \mathsf{Gen}(1^n; r') \rightarrow h = g^s$. (Note that here we use h to represent pk and s to represent sk. This line follows from the defining function of Gen.)

2. $h = g^s \rightarrow h^r \cdot m/(g^r)^s = (g^s)^r \cdot m/(g^r)^s$ (This follows from the substitution rule.)

3. $(g^s)^r = (g^r)^s$ (The basic properties of arithmetic can be proven in PV [Bus86].)

4. $(g^s)^r \cdot m/(g^r)^s = m$ (From 3 and basic properties of arithmetic, i.e. commutative law and associative law of multiplication.)

5. $h = g^s \rightarrow h^r \cdot m/(g^r)^s = m$ (Combine 4 with 2.)

6. $(h,s) = \mathsf{Gen}(1^n; r') \rightarrow \mathsf{Dec}(\mathsf{Enc}(\mathrm{pk}, m; r), \mathrm{sk}) = m$. (Follows from 5, 1, and the defining functions of Enc, Dec.)

We finish the proof. □

Lattice-based Instantiation. We next show how to instantiate the above example in lattice-based assumptions. We use Regev's public key encryption scheme [Reg05] as an example.

We first recall the construction of Regev's public key encryption scheme [Reg05]. Let n, m, q be positive integers. The private key is a vector $\mathbf{s} \in \mathbb{Z}_q^n$ and the public key is $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{1\times m}$, where $\mathbf{b} = \mathbf{s}^T \cdot \mathbf{A} + \mathbf{e}^T$, for some $\mathbf{e} \approx \mathbf{0}$. To encrypt a message $\mu \in \{0,1\}$, one uses the random string $\mathbf{r} \in \{0,1\}^m$, and computes

$(\mathbf{c}_1 = \mathbf{A}\cdot\mathbf{r}, c_2 = \mathbf{b}\cdot\mathbf{r} + \mu\cdot\lfloor q/2\rfloor)$. To decrypt the ciphertext $(\mathbf{c}_1, c_2)$, one computes $c_2 - \mathbf{s}^T\cdot\mathbf{c}_1$. If the value is close to $\lfloor q/2\rfloor$ then output 1. Otherwise, output 0. The correctness follows from the fact that

$$c_2 - \mathbf{s}^T\cdot\mathbf{c}_1 = \mathbf{b}\cdot\mathbf{r} + \mu\cdot\lfloor q/2\rfloor - \mathbf{s}^T\mathbf{A}\mathbf{r} = (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{r} + \mu\lfloor q/2\rfloor - \mathbf{s}^T\mathbf{A}\mathbf{r} = \mu\cdot\lfloor q/2\rfloor + \mathbf{e}^T\mathbf{r}. \qquad (2)$$

Since $\mathbf{e}\approx\mathbf{0}$, one can bound $|\mathbf{e}^T\mathbf{r}| \le \|\mathbf{e}\|_1\cdot\|\mathbf{r}\|_1 \le \|\mathbf{e}\|_1\cdot m$. Hence, if $\|\mathbf{e}\|_1\cdot m \le q/4$, then the decryption is correct.

Lemma 12. *Regev's public key encryption scheme has PV-proof of correctness.*

Proof. To show that the above proof of correctness can be formalized in *PV*, we need to slightly modify the Gen algorithm such that Gen only outputs a key-pair if $\|\mathbf{e}\|_1 < q/(4m)$. Otherwise, it outputs $\perp$. Then

$$((\mathbf{A},\mathbf{b}),\mathbf{s}) = \mathsf{Gen}(1^n; r') \rightarrow \left\|\mathbf{b} - \mathbf{s}^T\mathbf{A}\right\|_1 \le q/(4m)$$

can be proven in *PV* by the defining functions of Gen. Here, we use $(\mathbf{A},\mathbf{b})$ to represent pk and $\mathbf{s}$ to represent sk. $((\mathbf{A},\mathbf{b}),\mathbf{s}) = \mathsf{Gen}(1^n; r')$ implicitly expresses that $\mathsf{Gen}(1^n; r')$ doesn't output $\perp$. Next, we can prove Equation 2 in *PV*, since $+, -, \cdot, /, \lfloor\cdot\rfloor, \le$ can be introduced in *PV* as function symbols, and their basic laws such as commutative law, associative law, distributive law, etc. can be proven in *PV*. This is proven by the work [Bus86], where Buss introduces the *BASIC* axioms to formalize these laws and he showed that *BASIC* can be proven in *PV* (See Chapter 6 in [Bus86]). $\square$

8.2 Witness Encryption for Turing Machines

In this subsection, we show how to use our results of iO for Turing machines with *PV*-proof of equivalence. (Theorem 10).

Definition 17 (*PV* proof of Disjointness)**.** *We say an $\mathcal{NP}\cap co\mathcal{NP}$ language $L\subseteq\{0,1\}^*$ has a PV proof of disjointness, if there exists Turing machines M and $\overline{M}$ with the following properties.*

- *L can be decided by M: $L = \{x \mid \exists w : M(x,w) = 1\}$.*
- *$\overline{L} = \{0,1\}\setminus L$ can be decided by $\overline{M}$: $\overline{L} = \{x \mid \exists\overline{w} : \overline{M}(x,\overline{w}) = 1\}$.*
- *PV*-**proof of "**$L\cap\overline{L} = \phi$**":** $\vdash_{PV} \overline{M}(x,\overline{w}) = 1 \rightarrow M(x,w) = 0$, *which says that the statement*

  "For any x, if there exists a witness $\overline{w}$ for $\overline{L}$, then x can't have a witness w for L"

  can be proven in theory PV.

Remark 3. *The above definition of PV proof of disjointness for languages in $\mathcal{NP}\cap co\mathcal{NP}$ can be extended to promise languages $(L_{\mathsf{YES}}, L_{\mathsf{NO}})$, where we require M defines L_{YES} and $\overline{M}$ defines L_{NO} and "$L_{\mathsf{YES}}\cap L_{\mathsf{NO}} = \phi$" can be proven in PV.*

Theorem 11 (Witness Encryption for Turing machines)**.** *Under the same assumptions of Theorem 10, there exists a witness encryption scheme for any language $L\in\mathcal{NP}\cap co\mathcal{NP}$ with PV proof of disjointness. Moreover, let $\overline{M}(\cdot,\cdot)$ be the Turing machine that decides the relation of $\overline{L}$, and for any x, let $T(|x|)$ be the upper bound of the running time of the $\overline{M}(x,\cdot)$, then the ciphertext size for instance x is* $\mathsf{poly}(\lambda, T(|x|))$, *which is* independent *of the witness size.*

Remark 4. *We remark that although the ciphertext size grows with $T(|x|)$ in our witness encryption construction, the security parameter is independent of $T(|x|)$, and can be set to be smaller than the witness length for $\overline{L}$.*

Proof. Let M and $\overline{M}$ be the Turing machines in the Definition 17. Our construction of witness encryption follows from the construction in [SW21]. Namely, to encrypt a message m under an instance x_0, we output the obfuscation of the following Turing machine $f_{x_0,m}(w)$ as ciphertext. $f_{x_0,m}(w)$ outputs m, if $M(x_0, w) = 1$. Otherwise, it outputs 0.

To prove the security, we need to show that for any $x_0 \notin L$, we have $\vdash_{PV} f_{x_0,m}(w) = 0$. Once we have this, we can invoke Theorem 10 to argue that for any m_0, m_1, the obfuscation of $f_{x_0,m_0}(\cdot)$, $\Delta\mathsf{iO}(f_{x_0,m_0})$ is indistinguishable with $\Delta\mathsf{iO}(0)$, and thus is also indistinguishable with $\Delta\mathsf{iO}(f_{x_0,m_1})$ by hybrid argument. Hence the indistinguishable security follows.

We now show $\vdash_{PV} f_{x_0,m}(w) = 0$ for any $x_0 \notin L$. We first formalize $f_{x_0,m}(\cdot)$ in PV. Specifically, we formalize the "if" condition in PV as the following function symbol If.

$$\mathsf{If}(0, y, z) = z, \quad \mathsf{If}(x||1, y, z) = y, \quad \mathsf{If}(x||2, y, z) = 0$$

The proof of the bound on the output length in PV is straightforward. Now, $f_{x_0,m}(w)$ can be formalized in PV as

$$f_{x_0,m}(w) = \mathsf{If}(M(x_0, w), m, 0),$$

where x_0, m are expressed as *numerals* in PV (See Section 3.2). We prove $f_{x_0,m}(w) = 0$ in PV as follows.

1. $\overline{M}(x, \overline{w}) = 1 \rightarrow M(x, w) = 0$ (Follows from the PV proof of disjointness.)
2. $M(x, w) = 0 \rightarrow \mathsf{If}(M(x, w), m, 0) = \mathsf{If}(0, m, 0)$ (Axiom E_4)
3. $\overline{M}(x, \overline{w}) = 1 \rightarrow \mathsf{If}(M(x, w), m, 0) = \mathsf{If}(0, m, 0)$ (Combine 1 and 2 by the implication rule.)
4. $\overline{M}(x, \overline{w}) = 1 \rightarrow (\mathsf{If}(M(x, w), m, 0) = \mathsf{If}(0, m, 0) \wedge \mathsf{If}(0, m, 0) = 0)$ (Combine 3 and defining equation of If by the implication rule)
5. $(\mathsf{If}(M(x, w), m, 0) = \mathsf{If}(0, m, 0) \wedge \mathsf{If}(0, m, 0) = 0) \rightarrow \mathsf{If}(M(x, w), m, 0) = 0$ (Axiom E_3)
6. $\overline{M}(x, \overline{w}) = 1 \rightarrow \mathsf{If}(M(x, w), m, 0) = 0$ (Combine 4 and 5 by the implication rule similar to 3)
7. $\overline{M}(x, \overline{w}) = 1 \rightarrow (\mathsf{If}(M(x, w), m, 0) = 0 \wedge f_{x,m}(w) = \mathsf{If}(M(x, w), m, 0))$ (Combine 6 and the defining equation of $f_{x,m}(w)$ by the implication rule similar to 4.)
8. $(\mathsf{If}(M(x, w), m, 0) = 0 \wedge f_{x,m}(w) = \mathsf{If}(M(x, w), m, 0)) \rightarrow f_{x,m}(w) = 0$ (Axiom E_3)
9. $\overline{M}(x, \overline{w}) = 1 \rightarrow f_{x,m}(w) = 0$ (Combine 7 and 8 by the implication rule similar to 3)
10. $\overline{M}(x_0, \overline{w}_0) = 1$, since $x_0 \in \overline{L}$ and $\overline{M}$ decides $\overline{L}$, there must be a $\overline{w}_0$ such that $\overline{M}(x_0, \overline{w}_0) = 1$, and $\overline{M}(x_0, \overline{w}_0) = 1$ can be proven in PV by following the computation of $\overline{M}$ in a natural way.
11. $\overline{M}(x_0, \overline{w}_0) = 1 \rightarrow f_{x_0,m}(w) = 0$ (Substitution rule applied to 9 with $x \mapsto x_0, \overline{w} \mapsto \overline{w}_0$)
12. $f_{x_0,m}(w) = 0$ (Combine 10 and 11 by implication rule.)

This concludes a proof in PV such that $\vdash_{PV} f_{x_0,m}(w) = 0$.

Bounding the size of $|\Pi|$. Line 1 contributes $O(1)$ in $\mathsf{Desc}[\Pi]$. For other lines except Line 10, they are either axioms or followed by inference rules. Hence they contribute $O(1)$ in $|\Pi|$. Line 10 costs a polynomial in the running time of $\overline{M}(x_0, \overline{w}_0)$. Hence, $|\Pi| = \mathsf{poly}(T(|x_0|))$, and by Corollary 2, we obtain a witness encryption of ciphertext size $\mathsf{poly}(T(|x_0|))$.

This finish the proof. We remark that although we need M, $\overline{M}$, and the existence of a $\overline{L}$-witness $\overline{w}_0$ in the *PV proof*, the *construction* of our witness encryption (and $\Delta\mathsf{iO}$) only needs Turing machine M. □

Example: The Language TAUT**.** As a concrete example of Theorem 11, we will show that TAUT, which contains all tautologies, has *PV* proof of disjointness, and hence Theorem 11 implies a witness encryption for TAUT. TAUT is an important language in complexity since it's $co\mathcal{NP}$-complete [AB09].

In this work, we define TAUT as the following promise language $(L_{\mathsf{YES}}, L_{\mathsf{NO}})$ so that the honest prover can take a polynomial size witness as input.

- **Yes Instance:** It contains all formula ϕ such that there exists a *polynomial-size* proof of size at most $p(|\phi|)$ for ϕ in the extended Frege system $\mathcal{EF}$. Let $p(\cdot)$ be a polynomial.

$$L_{\mathsf{YES}} = \{\phi \mid \exists \text{Propositional logic proof of size at most } p(|\phi|) \text{ for } \phi\}.$$

- **No Instance:** It contains all formula ϕ such that there exists an assignment x that $\phi(x) = 0$.

$$L_{\mathsf{NO}} = \{\phi \mid \exists x \text{ s.t. } \phi(x) = 0\}.$$

Note that size of the witness for L_{YES} is *unbounded* in $|\phi|$, since the size of the propositional proof can be arbitrary (here we allow $p(\cdot)$ to be *any* polynomial). However, the size of the witness for L_{NO} is bounded by $|\phi|$, since any assignment to ϕ can be described as a binary string of length no longer than $|\phi|$.

We will show in Lemma 13 that TAUT has *PV* proof of disjointness. Then by Theorem 11, there exists a witness encryption for TAUT with ciphertext size *independent* of the witness size.

Lemma 13. *The promise language* TAUT *has PV proof of disjointness.*

Proof. We first describe the construction of the Turing machines M and $\overline{M}$ deciding L_{YES} and L_{NO}, respectively. Let $F(\ulcorner\pi\urcorner)$ be the following function: if $\ulcorner\pi\urcorner$ is the Gödel number of a valid proof π in $\mathcal{EF}$, then F outputs the tautology that π proves. Otherwise F outputs 1. Clearly, F is a polynomial time function. Since any polynomial function can be defined in *PV*, so does F. Cook in [Coo75] further showed that the soundness of $\mathcal{EF}$ can be proven in *PV*. Specifically, let TRUTH be the following function.

$$\mathsf{TRUTH}(\ulcorner\phi\urcorner, \ulcorner v\urcorner) = \begin{cases} 1 & \text{if } \phi \text{ is a formula, and } v \in \{0,1\}^* \text{ is an assignment with } \phi(v) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Cook [Coo75] showed that $\vdash_{PV} \mathsf{TRUTH}(F(x), y) = 1$, which means that any formula that F outputs must evaluate to 1 for any assignment.

We will use F and TRUTH to define L_{YES} and L_{NO}, respectively. For L_{YES}, we build M as $M(\ulcorner\phi\urcorner, \ulcorner\pi\urcorner) = \mathsf{Eq}(\ulcorner\phi\urcorner, F(\ulcorner\pi\urcorner))$, where Eq is a function symbol such that $\mathsf{Eq}(x, y)$ outputs 1, if $x = y$, and it outputs 0 otherwise. Formally, we define $\mathsf{Eq}(x, y)$ recursively as follows.

$$\mathsf{Eq}(0,0) = 1$$
$$\mathsf{Eq}(0, y||j) = 0, \mathsf{Eq}(x||i, 0) = 0, \quad i, j \in \{1,2\}$$
$$\mathsf{Eq}(x||i, y||j) = \begin{cases} \mathsf{Eq}(x,y) & i = j \\ 0 & \text{Otherwise} \end{cases}, \quad i, j \in \{1,2\}$$

For L_{NO}, we build $\overline{M}(\ulcorner\phi\urcorner, \ulcorner v\urcorner)$ as $\overline{M}(\ulcorner\phi\urcorner, \ulcorner v\urcorner) = \neg\mathsf{TRUTH}(\ulcorner\phi\urcorner, \ulcorner v\urcorner)$. Next, we prove $\overline{M}(x, \overline{w}) = 1 \longrightarrow M(x, w) = 0$ in *PV*, where x and w represents $\ulcorner\phi\urcorner$ and $\ulcorner v\urcorner$, respectively. Here we only give a high level sketch.

1. Show that $\overline{M}(x, \overline{w}) = 1 \longrightarrow \mathsf{TRUTH}(x, \overline{w}) = 0$ by using defining function of $\overline{M}$.
2. Show that $\mathsf{Eq}(x, F(w)) = 1 \longrightarrow \mathsf{TRUTH}(x, \overline{w}) = 1$ via the following steps.

(a) Show that $\mathsf{Eq}(x, y) = 1 \to x = y$ for variables x and y, by an induction on the digits in x and y.

(b) $\mathsf{Eq}(x, F(w)) = 1 \to x = F(w)$. (Applying the substitution $y \mapsto F(w)$ to 2a.)

(c) $\mathsf{TRUTH}(F(x), y) = 1$ for the variables x, y (proven by Cook in [Coo75]).

Then combine 2b and 2c, and substitute y with $\overline{w}$ in 2c.

3. $\overline{M}(x, \overline{w}) = 1 \to \neg\mathsf{Eq}(x, F(w)) = 1$ (From 1 and 2, using implication rule.)

4. Show that $\overline{M}(x, \overline{w}) = 1 \to \mathsf{Eq}(x, F(w)) = 0$ from 3, by first proving that $\neg\mathsf{Eq}(x, y) = 1 \to \mathsf{Eq}(x, y) = 0$ and substitute y with $F(w)$.

5. Show that $\overline{M}(x, \overline{w}) = 1 \to M(x, w) = 0$, from the defining equation of $M(x, w)$ and 4.

Since we reach $\overline{M}(x, \overline{w}) = 1 \to M(x, w) = 0$ at the last line, we prove $\vdash_{PV} \overline{M}(x, \overline{w}) = 1 \to M(x, w) = 0$, and thus TAUT has *PV* proof of disjointness. □

Combining Theorem 11 and Lemma 13, we obtain a witness encryption for TAUT. Note that for any formula ϕ, the running time of the Turing machine $\overline{M}(\phi, \cdot)$ is $\mathsf{poly}(|\phi|)$, which is *independent* of the size of the propositional proof proving ϕ. Hence, we have the following corollary.

Corollary 4 (Witness Encryption for TAUT)**.** *Assuming the same assumptions as Theorem 11, there exists a witness encryption for* TAUT *with ciphertext size independent of the witness size.*

8.3 SNARGs for Turing Machines

In this section, we build SNARGs for Turing machines, following the construction in [SW21]. We first recall the definition of SNARGs [GW11].

Definition 18 (SNARGs)**.** *A succinct non-interactive argument (SNARG) for a language L is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *with the following syntax.*

- $\mathsf{Gen}(1^\lambda)$: *It takes as input the security parameter, and output a common reference string* crs.
- $\mathsf{P}(\mathsf{crs}, x, w)$: *It takes as input the* crs, *an instance x and its witness w, and outputs a proof π.*
- $\mathsf{V}(\mathsf{crs}, x, \pi)$: *It takes as input the* crs, *instance x, and a proof π, and decides to accept or reject.*

Furthermore, they need to satisfy the following properties.

- **Succinctness:** *The proof size $|\pi|$ is sub-linear in $|w|$.*
- **Completeness:** *For any $x \in L$ and any witness w of x,*

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \pi \leftarrow \mathsf{P}(\mathsf{crs}, x, w) : \mathsf{V}(\mathsf{crs}, x, \pi) \textit{ accepts}\right] = 1.$$

- **Soundness:** *For any $x^* \notin L$, and any non-uniform PPT adversary* P^*, *there exists a negligible function $\nu(\lambda)$ such that*

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \pi^* \leftarrow \mathsf{P}^*(\mathsf{crs}, x^*) : \mathsf{V}(\mathsf{crs}, x^*, \pi^*) \textit{ accepts}\right] \leq \nu(\lambda).$$

Theorem 12 (SNARGs for languages with *PV* proof of disjointness)**.** *Under the same assumptions of Theorem 10, there exists SNARGs for any language L with PV proof of disjointness. Moreover, let the relation Turing machine of $\overline{L}$ be $\overline{M}$ and let $T(|x|)$ be an upper bound of the running time for $\overline{M}(x, \cdot)$. Then the CRS size of the SNARG is bounded by* $\mathsf{poly}(T(|x|))$.

Before we prove the theorem, we introduce the following notion of PV proof of functionality preservation.

Definition 19 (PV-proof of Functionality Preservation). *We say a puncturable PRF has PV-proof of functionality preservation, if* $\mathsf{PRF}, \mathsf{PRFPunc}, \mathsf{PRF}_{punc}$ *can be formalized as function symbols in PV, and*

$$\vdash_{PV} \left(\neg x = x^* \wedge \mathsf{EQLEN}(x, x^*) = 1\right) \to \mathsf{PRF}_{punc}(\mathsf{PRFPunc}(K, x^*), x) = \mathsf{PRF}(K, x),$$

where $\mathsf{EQLEN}(a, b)$ *is a function that outputs* 1 *if* a, b *have the same lengths, and it outputs* 0 *otherwise.* EQLEN *can be defined in PV by limited recursion on the notations of a and b.*

Lemma 14. *There exists a construction of puncturable PRF with PV-proof of functionality preservation.*

Proof. We will show that the GGM construction of puncturable PRF in [GGM84] has PV-proof of functionality preservation property.

Formalization. We first formalize GGM puncturable PRF construction in PV.

- $\underline{\mathsf{PRF}(K, x)}$: Let $\mathsf{PRG} : \{0,1\}^{|K|} \to \{0,1\}^{2|K|}$ be a pseudorandom generator, $\pi_1 : \{0,1\}^{2|K|} \to \{0,1\}^{|K|}$ be a function that outputs the least significant half digits of its input, and $\pi_2 : \{0,1\}^{2|K|} \to \{0,1\}^{|K|}$ be a function that outputs the most significant half digits of its input. Since $\mathsf{PRG}, \pi_1, \pi_2$ are polynomial time functions, they can be defined in PV as function symbols. Next, we define $\mathsf{PRF}(K, x)$ via limited recursion on the notations of x.

$$\mathsf{PRF}(K, 0) = K,$$
$$\mathsf{PRF}(K, x||i) = \pi_i(\mathsf{PRG}(\mathsf{PRF}(K, x))), \text{for } i = 1, 2$$

  The upper bound on its output length is provable in PV, since the output length of π_i is bounded by $|K|$.

- $\underline{\mathsf{PRFPunc}(K, x^*)}$: This algorithm outputs $K^* = (x^*, f^*)$, where $f^* = (f_1, \ldots, f_{|x^*|})$ is a list of PRF values on the root-to-leaf path for x^*. Specifically, $f_i^* = \mathsf{PRF}_{punc}(K, x_i^*)$, where x_i^* contains the first i digits of x^* with the last digit flipped. To define $\mathsf{PRFPunc}$ in PV, we introduce function symbols $\mathsf{Cons}, \mathsf{Car}, \mathsf{Cdr}$ to manipulate a list of elements, in a similar way as LISP [Ste90]. Specifically, they are function symbols in PV such that $\mathsf{Car}(\mathsf{Cons}(x, y)) = x$ and $\mathsf{Cdr}(\mathsf{Cons}(x, y)) = y$ can be proven in PV. Then we define $\mathsf{PRFPunc}$ as follows.

$$\mathsf{PRFPunc}(K, 0) = \mathsf{Cons}(0, 0),$$
$$\mathsf{PRFPunc}(K, x^*||i^*) = \mathsf{Cons}\Big(x^*||i^*, \mathsf{Cons}\big(\mathsf{Cdr}(\mathsf{PRFPunc}(K, x^*)), \mathsf{PRF}(K, x^*||\overline{i^*})\big)\Big),$$

  where $\overline{i^*} \in \{1, 2\} \setminus i^*$ is the “flip” of i^*.

- $\underline{\mathsf{PRF}_{punc}(K^*, x)}$: Since K^* is in the form $\mathsf{Cons}(x^*, f^*)$, we will define the following function symbol $\overline{\mathsf{PPRF}(K^*, x^*, x)}$ first and then define $\mathsf{PRF}_{punc}(K^*, x) = \mathsf{PPRF}(K^*, \mathsf{Car}(K^*), x)$.

$$\mathsf{PPRF}(K^*, 0, 0) = 0, \quad \mathsf{PPRF}(K^*, 0, x||i) = 0, \quad \mathsf{PPRF}(K^*, x^*||i^*, 0) = 0,$$
$$\begin{aligned}\mathsf{PPRF}(K^*, x^*||i^*, x||i) = \mathsf{If}(&\mathsf{Eq}(x^*, x),\\ &\mathsf{Nth}(x^*||i, K^*),\\ &\pi_i(\mathsf{PRG}(\mathsf{PPRF}(K^*, x^*, x)))), \quad i, i^* \in \{1, 2\},\end{aligned}$$

Where $\mathsf{Nth}(x^*, K^*)$ will output $f_{|x^*|}$. Formally, Nth is defined as

$$\mathsf{Nth}(x, L) = \mathsf{Cdr}(\mathsf{Strip}(\mathsf{LESS}(\mathsf{Car}(L), x), \mathsf{Cdr}(L))),$$

where $\mathsf{Strip}(y, L)$ is a function that deletes the last $|y|$ elements from the list L. Then our construction of $\mathsf{Nth}(y, L)$ first deletes $|L| - |y|$ elements in the list L and then outputs the last elements, which is the $|y|$-th element in the list. Formally, $\mathsf{Strip}(y, L)$ is defined via the following limited recursion on the notations of y.

$$\mathsf{Strip}(0, L) = L, \quad \mathsf{Strip}(y||i, L) = \mathsf{Car}(\mathsf{Strip}(y, L)).$$

Proof of Functionality Preservation in *PV*. To show above formalization of puncturable PRFs has PV proof of functionality preservation, we will argue inductively on the variable x and x^*. Specifically, let $A(x, x^*, K)$ be the formula we want to prove

$$\left(\neg x = x^* \land \mathsf{EQLEN}(x, x^*) = 1\right) \to \mathsf{PRF}_{punc}(\mathsf{PRFPunc}(K, x^*), x) = \mathsf{PRF}(K, x).$$

We will show that $A(0, 0, K)$ holds and $\vdash_{PV} A(x, x^*, K) \to A(x||i, x^*||i^*, K)$ for every $i, i^* \in \{1, 2\}$. Then by 2-induction rule (See our preliminary Section 3.3), we can prove $A(x, x^*, K)$. Since $A(0, 0, K)$ holds trivially, we only need to prove $A(x, x^*, K) \to A(x||i, x^*||i^*, K)$ in PV. We prove this in the following two cases.

- **Case I:** $x^* = x$**:** In this case we will prove that $A(x, x^*, K) \land x^* = x \to A(x||i, x^*||i^*, K)$. To show this, we prove that $x^* = x \to \mathsf{PRF}_{punc}(\mathsf{PRFPunc}(K, x^*||i^*), x||i) = \mathsf{Nth}(x^*||i, K^*)$ by the defining equations of PRF_{punc} and PPRF. Then we can prove that $\neg x||i = x^*||i^* \land x = x^* \to i = \overline{i^*}$, and hence

  $$x^* = x \to (\neg x||i = x^*||i^* \to \mathsf{Nth}(x^*||i, K^*) = \mathsf{PRF}(K, x^*||\overline{i^*})).$$

  Hence, we have $x^* = x \to (\neg x||i = x^*||i^* \to \mathsf{PRFPunc}(K, x^*||i^*), x||i) = \mathsf{PRF}(K, x^*||\overline{i^*}))$, which implies that $A(x, x^*, K) \land x^* \to A(x||i, x^*||i, K)$.

- **Case II:** $x^* \neq x$**:** In this case we will prove $A(x, x^*, K) \land \neg x^* = x \to A(x||i, x^*||i^*, K)$. We start with the induction hypothesis, since $\mathsf{EQLEN}(x||i, x^*||i^*) \to \mathsf{EQLEN}(x, x^*) = 1$ can be proven by the defining equations of EQLEN, we can prove

  $$A(x, x^*, K) \land \neg x^* = x \land \mathsf{EQLEN}(x||i, x^*||i^*) = 1 \to \mathsf{PRF}_{punc}(\mathsf{PRFPunc}(K, x^*), x) = \mathsf{PRF}(K, x).$$

  By defining equations of PRF_{punc}, we have $\mathsf{PRF}_{punc}(K \setminus \{x^*\}, x) = \mathsf{PPRF}(K \setminus \{x^*\}, x^*, x)$, where we denote $\mathsf{PRFPunc}(K, x^*)$ as $K \setminus \{x^*\}$ for simplicity. Hence, under the premise of $A(x, x^*, K), \neg x^* = x, \mathsf{EQLEN}(x||i, x^*||i^*) = 1$, we have $\mathsf{PPRF}(K \setminus \{x^*\}, x^*, x) = \mathsf{PRF}(K, x)$.

  Next, also by the defining equations of PRF_{punc} and PPRF, we have

  $$\neg x = x^* \to \mathsf{PRF}_{punc}(K \setminus \{x^*||i^*\}, x||i) = \pi_i(\mathsf{PRG}(\mathsf{PPRF}(K \setminus \{x^*||i^*\}, x^*, x))).$$

  On the other hand, from the defining equations of PRF, we have $\mathsf{PRF}(K, x||i) = \pi_i(\mathsf{PRG}(\mathsf{PRF}(K, x)))$. Now, once we have $\mathsf{PPRF}(K \setminus \{x^*\}, x^*, x) = \mathsf{PPRF}(K \setminus \{x^*||i^*\}, x^*, x)$, then we can derive $\mathsf{PRF}_{punc}(K \setminus \{x^*||i^*\}, x||i) = \mathsf{PRF}(K, x||i)$, and hence we finish the proof. To show this, we let $B(a, a^*, K, x^*)$ be the following formula.

  $$\mathsf{LESS}(a, x^*) = 0 \land \mathsf{EQLEN}(a, a^*) = 1 \to \mathsf{PPRF}(K \setminus \{x^*\}, a^*, a) = \mathsf{PPRF}(K \setminus \{x^*||i^*\}, a^*, a).$$

  Then we can prove $B(a, a^*, K, x^*)$ holds by a 2-induction on the notations of a and a^*. Finally, we apply substitution rule to replace a and a^* with x and x^* in the formula B. Then we finish the proof.

□

Using puncturable PRF with PV-proof of functionality preservation, we are ready to prove Theorem 12.

Proof of Theorem 12. We use the same construction of NIZKs in [SW21]. We recall the construction first. Then we formalize the algorithms used in the construction as function symbols in theory PV. Next, we show that all functional equivalence arguments used in the original soundness proof in [SW21] can be formalized in PV.

Construction. Let M be the Turing machine deciding the relation of the language $L \in \mathcal{NP}$. Let OWF be a one-way function.

- $\mathsf{Gen}(1^\lambda)$: Let $\mathsf{crs} = (\Delta\mathsf{iO}(\mathsf{PK}(\cdot,\cdot)), \Delta\mathsf{iO}(\mathsf{VK}(\cdot,\cdot))))$ be the obfuscation of the following Turing machines.
  - $\mathsf{PK}(x, w)$: It takes as input an instance x and a string w. If $M(x, w) = 1$, then output $\pi \leftarrow \mathsf{PRF}(K, x)$, where K is a secret key hardwired. Otherwise it output 0.
  - $\mathsf{VK}(x, \sigma)$: It takes as input an instance x and a proof π. if $\mathsf{OWF}(\pi) = \mathsf{OWF}(\mathsf{PRF}(K, x))$, then it outputs 1 to indicate "accept". Otherwise it outputs 0 to represent "reject".
- $\mathsf{P}(\mathsf{crs}, x, w)$: It parses $\mathsf{crs} = (\widetilde{\mathsf{PK}}(\cdot,\cdot), \widetilde{\mathsf{VK}}(\cdot,\cdot))$, then it executes and outputs $\pi \leftarrow \widetilde{\mathsf{PK}}(x, w)$.
- $\mathsf{V}(\mathsf{crs}, x, \pi)$: It parses $\mathsf{crs} = (\widetilde{\mathsf{PK}}(\cdot,\cdot), \widetilde{\mathsf{VK}}(\cdot,\cdot))$. If $\widetilde{\mathsf{VK}}(x, \pi) = 1$, then accept. Otherwise reject.

Formalizing PK, VK in PV. We formalize PK, VK as function symbols in PV as follows.

- $\mathsf{PK}(x, w)$: Let If be the condition function symbol we constructed in the proof of Theorem 11. We define
$$\mathsf{PK}(x, w) = \mathsf{If}(M(x, w), \mathsf{PRF}(K, x), 0),$$
where the PRF key k is represented as *numerals* in PV (See Section 3.2).
- $\mathsf{VK}(x, \pi)$: Since any polynomial-time algorithm can be defined in PV, we can define OWF as a function symbol in PV. We define
$$\mathsf{VK}(x, \pi) = \mathsf{If}(\mathsf{Eq}(\mathsf{OWF}(\pi), \mathsf{OWF}(\mathsf{PRF}(K, x))), 1, 0).$$

Completeness follows from the functional equivalence between $\widetilde{\mathsf{PK}}, \mathsf{PK}$ and $\widetilde{\mathsf{VK}}, \mathsf{VK}$.

Soundness. We prove the soundness via the following hybrids. For any $x_0^* \notin L$, we represent x_0^* as numerals in PV.

- H_0: In this hybrid, the crs is the same as in the honest CRS generation algorithm.
- H_1: We modify $\mathsf{PK}(\cdot,\cdot)$ as the following $\mathsf{PK}_1(\cdot,\cdot)$:
$$\begin{aligned}\mathsf{PK}_1(x, w) = &\mathsf{If}(\mathsf{Eq}(x, x_0^*),\\ &\qquad \mathsf{If}(M(x_0^*, w), \mathsf{PRF}(K, x_0^*), 0)\\ &\qquad \mathsf{If}(M(x, w), \mathsf{PRF}(K, x), 0)).\end{aligned}$$
which is almost the same as PK, except that we add a branch for $x = x_0^*$. We can prove $\vdash_{PV} \mathsf{PK}(x, w) = \mathsf{PK}_1(x, w)$ by proving $x = x_0^* \to \mathsf{PK}(x, w) = \mathsf{PK}_1(x, w)$ and $(\neg x = x_0) \to \mathsf{PK}(x, w) = \mathsf{PK}_1(x, w)$, and then combine them by the implication rule. Then the indistinguishability from H_0 and H_1 follows from the security of $\Delta\mathsf{iO}$.

- H_2: We further replace $\mathsf{PK}_1(x,w)$ with the following function symbol $\mathsf{PK}_2(\cdot,\cdot)$.

$$\begin{aligned}\mathsf{PK}_2(x,w) = & \mathsf{If}(\mathsf{Eq}(x,x_0^*),\\ & \quad \mathsf{If}(0, \mathsf{PRF}(K,x_0^*),0)\\ & \quad \mathsf{If}(M(x,w),\mathsf{PRF}(K,x),0)).\end{aligned}$$

  We can prove $\vdash_{PV} \mathsf{PK}_1(x,w) = \mathsf{PK}_2(x,w)$ by using the fact that $x_0^* \notin L$, and hence there exists $\overline{w}_0$ such that $\overline{M}(x_0^*, \overline{w}_0) = 1$. Since L has PV proof of disjointness, we have $\vdash_{PV} \overline{M}(x,\overline{w}) = 1 \rightarrow M(x,w) = 0$. Then $\mathsf{PK}_1(x,w) = \mathsf{PK}_2(x,w)$ follows.

- H_3: We further modify $\mathsf{PK}_2(\cdot,\cdot)$ to the following function symbol $\mathsf{PK}_3(\cdot,\cdot)$:

$$\mathsf{PK}_3(x,w) = \mathsf{If}(\mathsf{Eq}(x,x_0^*), 0, \mathsf{If}(M(x,w),\mathsf{PRF}(K,x),0)$$

  Then $\vdash_{PV} \mathsf{PK}_2(x,w) = \mathsf{PK}_3(x,w)$ follows from the defining function of If.

- H_4: We replace PRF with PPRF and build $\mathsf{PK}_4(\cdot,\cdot)$ as follows.

$$\mathsf{PK}_4(x,w) = \mathsf{If}(\mathsf{Eq}(x,x_0^*), 0, \mathsf{If}(M(x,w), \mathsf{PPRF}_{punc}(K \setminus \{x_0^*\}, x), 0)).$$

  We prove $\vdash_{PV} \mathsf{PK}_3(x,w) = \mathsf{PK}_4(x,w)$ by firstly proving $x = x_0^* \rightarrow \mathsf{PK}_3(x,w) = \mathsf{PK}_4(x,w)$ using the defining function of If, and then proving $(\neg x = x_0^*) \rightarrow \mathsf{PK}_3(x,w) = \mathsf{PK}_4(x,w)$ using PV-proof of functionality preservation of PPRF. Finally, we combine them to obtain $\mathsf{PK}_3(x,w) = \mathsf{PK}_4(x,w)$ using the implication rule similar to H_1.

- H_5: This hybrid is almost the same as H_4. The only difference is that we also replace $\mathsf{VK}(\cdot,\cdot)$ with the following function symbol $\mathsf{VK}'(\cdot,\cdot)$.

$$\begin{aligned}\mathsf{VK}'(x,\pi) = & \mathsf{If}(\mathsf{Eq}(x,x_0^*),\\ & \quad \mathsf{If}(\mathsf{Eq}(\mathsf{OWF}(\pi),\mathsf{OWF}(\mathsf{PRF}(K,x_0^*))),1,0)\\ & \quad \mathsf{If}(\mathsf{Eq}(\mathsf{OWF}(\pi),\mathsf{OWF}(\mathsf{PPRF}_{punc}(K\setminus\{x_0^*\},x))),1,0))\end{aligned}$$

  We can prove $\vdash_{PV} \mathsf{VK}(x,\pi) = \mathsf{VK}'(x,\pi)$ following the similar idea from H_1 to H_4. Then the indistinguishability from H_5 and H_4 follows from the security of ΔiO.

- H_6: In this hybrid, we modify $\mathsf{VK}'(x,\mathsf{pk})$ as the following $\mathsf{VK}''(\cdot,\cdot)$:

$$\begin{aligned}\mathsf{VK}'(x,\pi) = & \mathsf{If}(\mathsf{Eq}(x,x_0^*),\\ & \quad \mathsf{If}(\mathsf{Eq}(\mathsf{OWF}(\pi),y),1,0)\\ & \quad \mathsf{If}(\mathsf{Eq}(\mathsf{OWF}(\pi),\mathsf{OWF}(\mathsf{PPRF}_{punc}(K\setminus\{x_0^*\},x))),1,0)),\end{aligned}$$

  where $y = \mathsf{OWF}(s)$ is a numeral computed from a random numeral s. This hybrid is indistinguishable with H_5 from the pseudorandomness at the puncturable point x_0^* of PPRF.

  Now we can argue the soundness of the SNARGs in hybrid H_6 using the security of the one-way function OWF.

By the hybrid argument, we prove the soundness. □

Combining Theorem 12 and Lemma 13, we obtain the following corollary.

Corollary 5 (SNARGs for TAUT)**.** *Under the same assumption as Theorem 10, there exists SNARGs for* TAUT *with* unbounded witness length.

Acknowledgement

The authors were supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. The first author was additionally supported in part by AFOSR Award FA9550-19-1-0200 and the Office of Naval Research Grant N00014-19-1-2294. This work was done while the authors were visiting University of California Berkeley.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009. 67

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. https://eprint.iacr.org/2013/689. 7

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 191–225, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 4

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. 4, 16

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 284–332, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. 4

[AJS17] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Robust transforming combiners from indistinguishability obfuscation to functional encryption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 91–121, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 12

[AP20] Shweta Agrawal and Alice Pellet-Mary. Indistinguishability obfuscation without maps: Attacks and fixes for noisy linear FE. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 110–140, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 4

[AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 152–181, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 4

[BCG+18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018. 4

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 52–73, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany. 7

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 505–514, New York, NY, USA, May 31 – June 3, 2014. ACM Press. 7

[BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 79–109, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 4

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. 4, 34

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany. 9, 33

[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 Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press. 7, 16

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. 32

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 1480–1498, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press. 4

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 792–821, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 7

[Bus] Samuel R. Buss. Propositional proof complexity, an introduction. `https://mathweb.ucsd.edu/sbuss/ResearchWeb/marktoberdorf97/paper.pdf`. 84

[Bus86] Samuel Buss. *Bounded Arithmetic*. Bibliopolis, Naples, Italy, 1986. 7, 10, 11, 24, 64, 65

[Bus98] Samuel R. Buss. Chapter i - an introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 1–78. Elsevier, 1998. 6, 8, 10, 23

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany. 4

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 171–190, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press. 4, 16

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. 9, 33

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. 4

[BZ16] Mark Bun and Mark Zhandry. Order-revealing encryption and the hardness of private learning. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 176–206, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany. 4

[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 Annual ACM Symposium on Theory of Computing*, pages 429–437, Portland, OR, USA, June 14–17, 2015. ACM Press. 7, 16

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 1115–1127, Cambridge, MA, USA, June 18–21, 2016. ACM Press. 4

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from lwe. In *FOCS*, 2021. `https://ia.cr/2021/808`. 7, 18, 37

[CK07] Stephen Cook and Jan Krajíček. Consequences of the provability of NP $\subseteq$ P/poly. *Journal of Symbolic Logic*, 72(4):1353 – 1371, 2007. 11

[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. 11, 24, 25

[Coo] Stephen Cook. Connecting complexity classes, weak formal theories, and propositional proof systems. https://www.cs.toronto.edu/sacook/slidesPrint.pdf. page 10-15. 11

[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. 7, 8, 10, 11, 20, 21, 24, 25, 26, 52, 54, 67, 68, 83

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979. 11, 20, 26

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985. 9, 10, 11, 63, 64

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. 33

[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 Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press. 4, 34

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 518–535, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. 7

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press. 33, 69

[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 Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. 4, 5

[GJLS21] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 97–126, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. 4

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 151–170, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press. 4

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing*, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press. 34

[GMM$^+$16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 241–268, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. 4

[GP17] Sanjam Garg and Omkant Pandey. Incremental program obfuscation. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 193–223, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 12

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 736–749. ACM, 2021. 4

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 579–604, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. 5, 12

[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, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 156–181, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 5, 12

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 419–442, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. 5, 12

[GS18] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 425–454, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. 16

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 99–108, San Jose, CA, USA, June 6–8, 2011. ACM Press. 5, 8, 68

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 443–457, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany. 4

[HLR21] Justin Holmgren, A. Lombardi, and R. Rothblum. Fiat-shamir via list-recoverable codes (or: Parallel repetition of gmw is not zero-knowledge). *STOC*, 2021. 37

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations *in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery. 7, 18, 34, 35

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st Annual ACM Symposium on Theory of Computing*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press. 33

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 668–697, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. 7

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 251–281, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 4

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021. 4, 8

[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 Annual ACM Symposium on Theory of Computing*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press. 7, 16

[KP90] Jan Krajíček and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Mathematical Logic Quarterly*, 36(1):29–46, 1990. 11

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press. 9, 33

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 28–57, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 4

[Lin17] Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 599–629, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 4

[LPST16a] 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: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 447–462, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. 4

[LPST16b] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 96–124, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany. 4, 7

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 630–660, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 4

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th Annual Symposium on Foundations of Computer Science*, pages 11–20, New Brunswick, NJ, USA, October 9–11, 2016. IEEE Computer Society Press. 4

[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: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 138–169, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 4, 5, 12

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. 5

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. 5

[Par71] Rohit Parikh. Existence and feasibility in arithmetic. *The Journal of Symbolic Logic*, 36(3):494–508, 1971. 7, 24

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 333–342, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. 32

[Pic15] Ján Pich. Logical strength of complexity theory and a formalization of the PCP theorem in bounded arithmetic. *Logical Methods in Computer Science*, Volume 11, Issue 2, June 2015. 11

[PRS17] Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 461–473, Montreal, QC, Canada, June 19–23, 2017. ACM Press. 32

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 89–114, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. 37

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. 4

[PW85] J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In Carlos Augusto Di Prisco, editor, *Methods in Mathematical Logic*, pages 317–340, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. 11

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press. 9, 10, 11, 63, 64

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009. 32

[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). 11

[Ste90] Guy Steele. *Common LISP: the language*. Elsevier, 1990. 69

[SW21] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *SIAM J. Comput.*, 50(3):857–908, 2021. 4, 5, 8, 9, 12, 63, 66, 68, 71

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 127–156, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. 4

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. 16

A Proof of Theorem 9

To present our succinct description of Cook's translation, we first define the following operations on the succinct description of small formula derivation (See Definition 13).

Concatenation. Given two succinct descriptions $\Pi_1 = (\mathsf{Get}_1, \mathsf{Where}_1), \Pi_2 = (\mathsf{Get}_2, \mathsf{Where}_2)$, let the number of lines in the extension phase and the reasoning phase of the proof described by Π_i be E_i, R_i, respectively ($i = 1, 2$). Then we define the following succinct description $\Pi = (\mathsf{Gen}, \mathsf{Where})$ as the concatenation of Π_1, Π_2, and denote it as $\Pi = \Pi_1 \circ \Pi_2$.

Π describes the following concatenated proof, where the first $E_1 + E_2$ lines are the concatenation of the extension phase of Π_1, Π_2, and the rest $R_1 + R_2$ lines are the concatenation of the reasoning phase of Π_1, Π_2. Specifically,

- $\mathsf{Get}(i)$: If $i \in [E_1]$, output $\mathsf{Get}_1(i)$. If $i \in (E_1, E_1+E_2]$, output $\mathsf{Get}_2(i-E_1)$. If $i \in (E_1+E_2, E_1+E_2+R_1]$, then output $\mathsf{Get}_1(i - E_1)$. Finally, $i > E_1 + E_2 + R_1$, and we output $\mathsf{Get}_2(i - (E_1 + R_1))$.
- $\mathsf{Where}(\ulcorner v \urcorner)$: It invokes $\mathsf{Where}_1(\ulcorner v \urcorner)$ and $\mathsf{Where}_2(\ulcorner v \urcorner)$ and shifts the returned indices to the corresponding indices in the concatenated proof, and finally outputs the union of them.

Substitution. For any substitution σ in PV, we naturally extend σ to the propositional variables in extended Frege system $\mathcal{EF}$. Namely, we define $\sigma(P_i[t]) := P_i[t\sigma]$, and also $\sigma(Q_i[t]) := Q_i[t\sigma]$. We also extend the substitution to $\mathsf{ATOM}[f]$ for any formula f in $\mathcal{EF}$, i.e. we define $\sigma(\mathsf{ATOM}[f]) := \mathsf{ATOM}[f\sigma]$. (See $\mathsf{ATOM}[\cdot]$ defined for intermediate variables in Section 7.2).

Let σ be a substitution in PV. Then given any succinct description $\Pi = (\mathsf{Get}, \mathsf{Where})$, we define the succinct description $\Pi\sigma$ through one of the following cases.

- **Direct Construction:** We will explicitly specify the construction of $\Pi\sigma$ for any σ after our construction of Π. Our construction will satisfy the following property.
  - $\mathsf{Get}'(i)$: It invokes $\mathsf{Get}(i)$, and applies σ to it.
  - $\mathsf{Where}'(\ulcorner v \urcorner)$: We use $\sigma^{-1}(v)$ to denote the set of all variables in Π such that its image under σ is v. Then this circuit will output $\bigcup_{v' \in \sigma^{-1}(v)} \mathsf{Where}(\ulcorner v' \urcorner)$.
- **Inductive Definition:** If $\Pi = \Pi_1 \circ \Pi_2$, and $\Pi_1\sigma, \Pi_2\sigma$ have been already defined, then we define $\Pi\sigma := (\Pi_1\sigma) \circ (\Pi_2\sigma)$. Also, if $\Pi = \Pi'\sigma'$, then we define $\Pi\sigma := \Pi'(\sigma' \cdot \sigma)$, where $\sigma' \cdot \sigma$ is the composition of σ' and σ.

On the Sizes. For concatenation operation, if $\Pi = \Pi_1 \circ \Pi_2$, then Get and Where circuits of Π have sizes $|\mathsf{Get}_1| + |\mathsf{Get}_2| + \mathsf{poly}(\log m)$, and $|\mathsf{Where}_1| + |\mathsf{Where}_2| + \mathsf{poly}(\log m)$, respectively. Hence, we have $|\Pi| = |\Pi_1| + |\Pi_2| + \mathsf{poly}(\log m)$.

Similarly, for substitution operation, if $\Pi\sigma$ is defined through direct construction, then we have $|\Pi\sigma| = |\Pi| + |\sigma| + |\sigma^{-1}|$, where $|\sigma|$ is the size of the circuit computing σ, and $|\sigma^{-1}|$ is the size of the circuit computing σ^{-1}. In our construction, we are interested in those σ such that both $|\sigma|$ and $|\sigma^{-1}|$ can be bounded by $\mathsf{poly}(\log m)$.

A.1 Succinct Description of Translated Formulas

In this section, we will succinctly describe the propositional formula $[\![t = u]\!]^n_m$ in Cook's translation. Namely, we will break the formula $[\![t = u]\!]^n_m$ via a series of extension rules, such that each rule only contain a single connective (See Section 7.2).

Towards this, we first define a succinct description $\Phi_m[t]$ for any term t. Intuitively, $\Phi_m[t]$ specifies how $\mathsf{Val}_m[t]$ is computed from the variables appeared in t via a sequence of extension rules. (See Preliminary 3.2 for the definition of $\mathsf{Val}_m[t]$.) If the bounding value m is clear from the context, we suppress it for representation simplicity. More specifically, $\Phi_m[t]$ is defined inductively as follows.

- If t is a variable or function symbols of arity 0, then we define $\Phi_m[t]$ as the empty proof.
- If t is a term $f(t_1, \ldots, t_k)$, where f is function symbols and $t_1, \ldots, t_k$ are terms, we define

$$\Phi[f(t_1, \ldots, t_k)] := \Phi[t_1] \circ \Phi[t_2] \circ \ldots \circ \Phi[t_k] \circ \Phi[f]\sigma,$$

  where f is a function symbol originality defined as $f(x_1, \ldots, x_k)$ and x_i's are variables, and $\sigma : x_1 \mapsto t_1, \ldots, x_k \mapsto t_k$ is a substitution, and $\Phi[f]$ for any function symbol f is defined as follows.
  - If f is an initial function, then we define $\Phi[f]$ in a natural way.
  - If f is defined as $f(x_1, \ldots, x_k) = t'$, where t' is a term. Then we define $\Phi[f] := \Phi[t'] \circ \Phi^{eq}_f$ inductively, where Φ^{eq}_f consists of the lines setting $\mathsf{Val}_m[f(x_1, \ldots, x_k)]$ as the same truth values of $\mathsf{Val}_m[t']$.

– If f is defined in Cobham's limited recursion on notation, i.e.

$$f(0,\mathbf{y}) = g(\mathbf{y}), f(x||i,\mathbf{y}) = h_i(x,\mathbf{y},f(x,\mathbf{y})), \quad i = 1,2.$$

Then we define

$$\Phi[f] := \Phi[g(\mathbf{y})] \circ \Phi^{\mathsf{CP}}_{\mathbf{y}} \circ \Phi^{\mathsf{CP}}_{g(\mathbf{y})} \circ \Phi_x \circ \Phi_f,$$

where $\Phi^{\mathsf{CP}}_{\mathbf{y}}, \Phi^{\mathsf{CP}}_{g(\mathbf{y})}, \Phi_x, \Phi_f$ are defined as follows. Here, we use $\Phi^{\mathsf{CP}}_{\mathbf{y}}, \Phi^{\mathsf{CP}}_{g(\mathbf{y})}$ to "copy" the values of the term $\mathbf{y}$ and $g(\mathbf{y})$ for m times, and when we compute f at each level of the recursion, we use "one copy" of $\mathbf{y}, g(\mathbf{y})$. In this way, each propositional variable in $\mathsf{Val}_m[\mathbf{y}]$ and $\mathsf{Val}_m[g(\mathbf{y})]$ only appears a constant number of times in the translated propositional logic proof, and hence our 'Where' circuit can have an output length independent of m, which is necessary for 'Where' circuit to be succinct.

– $\Phi^{\mathsf{CP}}_{\mathbf{y}}$: We introduce the function symbols $\mathsf{CP}^1(\mathbf{y}), \mathsf{CP}^2(\mathbf{y}), \dots, \mathsf{CP}^m(\mathbf{y})$ to "copy" the value of the term $\mathbf{y}$, i.e. we firstly set $\mathsf{CP}^0(\mathbf{y}) = \mathbf{y}$, and then for each $j = 1,2,\dots,m$, we set $\mathsf{CP}^j(\mathbf{y}) = \mathsf{CP}^{j-1}(\mathbf{y})$. $\Phi^{\mathsf{CP}}_{\mathbf{y}}$ is then defined as the following extension rules. For $\mathbf{V} \in \{P_i, Q_i \mid i = 0,1,\dots,m\}$,

$$\mathbf{V}[\mathsf{CP}^0(\mathbf{y})] \leftrightarrow \mathbf{V}[\mathbf{y}] \quad \mathbf{V}[\mathsf{CP}^j(\mathbf{y})] \leftrightarrow \mathbf{V}[\mathsf{CP}^{j-1}(\mathbf{y})],\ \forall j \in [m].$$

Next, we define the 'Where' circuits of $\Phi^{\mathsf{CP}}_{\mathbf{y}}$ and $\Phi^{\mathsf{CP}}_{\mathbf{y}}\sigma$ via direct construction. Since $\Phi^{\mathsf{CP}}_{\mathbf{y}}$ can also be regarded as $\Phi^{\mathsf{CP}}_{\mathbf{y}}\sigma$ with a trivial σ, we only need to define 'Where' for $\Phi^{\mathsf{CP}}_{\mathbf{y}}\sigma$, where σ is a general substitution in PV. For input variable v in $\mathcal{EF}$, if v is in the form $\mathbf{V}[\mathsf{CP}^j(\sigma(\mathbf{y}))]$ or $\mathbf{V}[\sigma(\mathbf{y})]$, where $\mathbf{V} \in \{P_i, Q_i \mid i = 0,1,\dots,m\}$ then 'Where' outputs the indices of the lines that involves $\mathbf{V}[\mathsf{CP}^j(\mathbf{y})]$ and $\mathbf{V}[\mathbf{y}]$.

– $\Phi^{\mathsf{CP}}_{g(\mathbf{y})}$: Similar to $\Phi^{\mathsf{CP}}_{\mathbf{y}}$, we introduce the terms $\mathsf{CP}^0(g(\mathbf{y})) = g(\mathbf{y}), \mathsf{CP}^j(g(\mathbf{y})) = \mathsf{CP}^{j-1}(g(\mathbf{y})), j \in [m]$ to copy the term $g(\mathbf{y})$.
The 'Where' circuit of $\Phi^{\mathsf{CP}}_{g(\mathbf{y})}$ is constructed in a similar way as above.

– Φ_x: We introduce the variables x^j for $j = 0,1,\dots,m-1$, and informally speaking, let "$x^j = \mathsf{TR}(x^{j+1})$ for any $j \in [m]$, and $x^m = x$". Formally, the lines in Φ_x is as follows.

$$\begin{gathered}\mathsf{Val}_m[x^m] \leftrightarrow \mathsf{Val}_m[x] \\ P_m[x^j] \leftrightarrow \mathsf{F}, \quad P_i[x^j] \leftrightarrow P_{i+1}[x^{j+1}], \quad \text{For } i,j = 0,\dots,m-1 \\ Q_m[x^j] \leftrightarrow \mathsf{F}, \quad Q_i[x^j] \leftrightarrow Q_{i+1}[x^{j+1}], \quad \text{For } i,j = 0,\dots,m-1\end{gathered}$$

Since we introduce new variables x^j, we need to define how they interact with substitutions over $x, \mathbf{y}$. For any substitution σ, we define $\sigma(x^j) := \sigma(x)^j$. We construct 'Where' circuit of $\Phi_x\sigma$ for a general substitution σ via direct construction as follows. For input propositional variable v, if v is in the form $\mathbf{V}[\sigma(x)^j]$, then it outputs the indices that $\mathbf{V}[x^j]$ appears in the above extension rules, where $\mathbf{V} \in \{P_i, Q_i \mid i = 0,1,\dots,m\}$.

– Φ_f : Finally, we introduce the terms $f(x^j,\mathbf{y})$ for $j = 0,1,\dots,m$, and informally speaking, we shall define them as "$f(x^j,\mathbf{y}) = \mathsf{CP}^j(g(\mathbf{y}))$, if $x^j = 0$, and $f(x^j,\mathbf{y}) = h_i(x^{j-1}, \mathsf{CP}^{j-1}(\mathbf{y}), f(x^{j-1},\mathbf{y}))$ otherwise", where i is the last digit of x^j. Formally, Φ_f consists of the following lines. Let $\alpha_i, \beta_i, \gamma_i$ be the variables in the original definition of h_i, $i = 1,2$.
We first introduce $\mathsf{Val}_m[h_i(x^{j-1}, \mathsf{CP}^{j-1}(\mathbf{y}), f(x^{j-1},\mathbf{y}))]$, $i = 1,2$ by

$$\Phi_m[h_i]\sigma_i^j, \text{where } \sigma_i^j : \alpha_i \mapsto x^{j-1}, \beta_i \mapsto \mathsf{CP}^{j-1}(\mathbf{y}), \gamma_i \mapsto f(x^{j-1},\mathbf{y}), i = 1,2, j \in [m],$$

Then we introduce $\mathsf{Val}_m[f(x^j,\mathbf{y})]$ by selecting between $\mathsf{Val}_m[h_i(x^{j-1},\mathsf{CP}^{j-1}(\mathbf{y}),f(x^{j-1},\mathbf{y}))](i=1,2)$ and $\mathsf{Val}_m[\mathsf{CP}^j(g(\mathbf{y}))]$, depending on the last digit of x^j and $Q_0[x^j]$.

$$\begin{aligned}\mathbf{V}[f(x^j,\mathbf{y})] \leftrightarrow \Bigg(Q_0[x^j] \wedge \Bigg[&(\neg P_0[x^j] \wedge \mathbf{V}[h_1(x^{j-1},\mathsf{CP}^{j-1}(\mathbf{y}),f(x^{j-1},\mathbf{y}))])\\ &\vee(P_0[x^j] \wedge \mathbf{V}[h_2(x^{j-1},\mathsf{CP}^{j-1}(\mathbf{y}),f(x^{j-1},\mathbf{y}))])\Bigg]\Bigg)\vee\\ &(\neg Q_0[x^j]) \wedge \mathbf{V}[\mathsf{CP}^j(g(\mathbf{y}))],\ \text{where } \mathbf{V}\in\{P_i,Q_i \mid i=0,1,\dots,m\}.\end{aligned}$$

The above lines of extensions can be broken down by introducing intermediate propositional variables. To avoid using $Q_0[x^j]$ in every lines of $\mathbf{V}=Q_i, i=0,1,\dots,m$, we "copy" it to $Q_0^i[x^j]$, similar to what we did for $\mathsf{CP}(\mathbf{y})$. For any substitution σ, we define $f(x^j,\mathbf{y})\sigma := f(\sigma(x)^j,\mathbf{y}\sigma)$ and $\sigma(Q_0^i[x^j]) = Q_0^i[\sigma(x)^j]$. Then we define 'Where' circuit for $\Phi_f\sigma$ correspondingly via direct construction.

Describe $\mathsf{Prop}_m[\cdot]$ **Succinctly.** Next, we will succinctly describe $\mathsf{Prop}_m[\cdot]$ in [Coo75] by small description of small formula derivation $\Psi_m^n[\cdot]$. For a brief overview of $\mathsf{Prop}_m[\cdot]$, the readers can refer to our preliminary (Section 3.2). We define $\Psi_m^n[t]$ for any term t inductively as follows.

- If t is a variable, recall that $\mathsf{Prop}_m[t] = \bigwedge_{i=1}^m Q_i[t] \to Q_{i-1}[t]$. Ψ_m^n consists of the following lines introducing intermediate propositional variables (See Section 7.2).

  We first introduce $\mathsf{ATOM}[\mathbf{V}[t]] \leftrightarrow \mathbf{V}[t]$, for all $\mathbf{V}\in\{P_i,Q_i \mid i=0,1,\dots,m\}$. Then for $j=1,2,\dots,m$, we introduce a new variable $R_j[t]$ for the term t, and write the following lines and setting $R_0[t]\leftrightarrow \mathsf{T}$.

$$\begin{aligned}\mathsf{ATOM}\Big[Q_j[t]\to Q_{j-1}[t]\Big] &\leftrightarrow \Big(\mathsf{ATOM}\Big[Q_i[t]\Big] \to \mathsf{ATOM}\Big[Q_{i-1}[t]\Big]\Big)\\ R_j[t] &\leftrightarrow R_{j-1}[t] \wedge \mathsf{ATOM}\Big[Q_j[t]\to Q_{j-1}[t]\Big]\end{aligned}$$

  Then $\mathsf{Prop}_m[t]$ can be represented as $R_m[t]$. For any substitution σ in PV, we define $R_i[t]\sigma := R_i[t\sigma]$, and construct Where circuit for $\Psi_m^n[t]$ accordingly.

- If t is in the form $t=f(t_1,\dots,t_k)$, where f is a function symbol, and $t_1,\dots,t_k$ are terms. Then we define $\Psi_m^n[t]$ to contain the following lines.

$$\Psi_m^n[f(t_1,\dots,t_k)] := \Phi_m[f(t_1,\dots,t_k)] \circ \Psi_m[x_1] \circ \Psi_m[x_1] \circ \dots \circ \Psi_m[x_{k'}],$$

  where $x_1,x_2,\dots,x_{k'}$ are the variables appears in t.

- For any function symbol $f(x_1,\dots,x_k)$, we define $\Psi_m^n[f]$ naturally to describe $\mathsf{Prop}_m[f(x_1,\dots,x_k)]$ in a similar way as above.

Finally, we define $\Psi_m^n[\cdot]$ for equations. For any equation $t=u$, where both t and u are terms, we define

$$\Psi_m^n[t=u] := \Psi_m^n[t]\circ\Psi_m^n[u]\circ\Psi^{Eq},$$

where Ψ^{Eq} is the succinct description of the following formula $[\![t=u]\!]_m^n$ that represents "if the variables in t and u are in a certain range, then $\mathsf{Val}_m[t]$ is the same as $\mathsf{Val}_m[u]$". We use a new propositional variable $\mathsf{EQ}[t,u]$ to represent $[\![t=u]\!]_m^n$.

$$\begin{aligned}\mathsf{EQ}_m^n[t,u] \leftrightarrow \Big\{\Big(&R_m[x_1]\wedge R_m[x_2]\dots\wedge R_m[x_k]\wedge\neg Q_n[x_1]\wedge\neg Q_n[x_2]\wedge\dots\wedge\neg Q_n[x_k]\Big)\to\\ &\bigwedge_{i=0}^m (Q_i[t]\leftrightarrow Q_i[u]) \wedge \Big[Q_i[t]\to(P_i[t]\leftrightarrow P_i[u])\Big]\Big\}.\end{aligned}$$

The above extension for $\mathrm{EQ}^n_m[t,u]$ can be broken down to small formulas via a series of extension rules that can be described succinctly. Finally, we set the reasoning phase of $\Psi_m[t=u]$ as $\mathrm{EQ}^n_m[t,u]$.

On the Sizes. From the analysis on the sizes of concatenation and substitution operations in the previous section, we bound the size of $\Psi^n_m[t]$ for any term t as follows. For a term $t = f(t_1,t_2,\dots,t_k)$ with variables $x_1,\dots,x_{k'}$, we have $|\Psi^n_m[t]| \le |\Phi_m[t]| + O(k')\cdot \mathrm{poly}(\log m)$, where $\Phi_m[f(t_1,t_2,\dots,t_k)]$ can be bounded inductively as $|\Phi_m[f(t_1,\dots,t_k)]| \le |\Phi_m[t_1]| + |\Phi_m[t_2]| + \dots + |\Phi_m[t_k]| + |\Phi_m[f]|\cdot|\sigma| + O(k)\cdot\mathrm{poly}(\log m)$, and $|\sigma| \le O(|t_1| + |t_2| + \dots + |t_k|)$. The sizes of other succinct descriptions can be argued similarly in an inductive way. Finally, we have $|\Psi^n_m[t]| \le \mathrm{poly}(\mathrm{Desc}[t], \log m)$ for any term t.

A.2 Succinct Description of Cook's Translation

Finally, we slightly modify Cook's translation from theory PV to propositional logic to make the translation succinct.

Let $(\mathrm{eq}_1, \mathrm{eq}_2, \dots, \mathrm{eq}_\ell)$ be a proof in theory PV. To translate it to propositional logic proof, we need to translate each step of the derivation to propositional logic. However, the inference rule R_4 of PV is essentially a substitution rule. But the extended Frege system in our current definition does not natively support substitutions rules, although it is known how to use extended Frege to simulate substitution Frege. Here we incorporate this kind of simulation presented in [Bus, Theorem 9] to deal with R_4 rule.

Our first step is to define a substitution $\sigma_{i,j}$, which maps any propositional variable x to a new propositional variable $x^{(i,j)}$. Then we use $\sigma_{i,j}$ to replace all propositional variables in $\Psi_m[\mathrm{eq}_1], \Psi_m[\mathrm{eq}_2], \dots, \Psi_m[\mathrm{eq}_\ell]$. For simplicity, let's denote $\Psi^{i,j}_k := \Psi_m[\mathrm{eq}_k]\sigma_{i,j}$, and we introduce

$$\phi_j := \Psi^{j,1}_1 \circ \Psi^{j,2}_2 \circ \dots \circ \Psi^{j,j}_j$$

Next, to prove $\Psi_m[\mathrm{eq}_\ell]$ in extended Frege system, we will firstly prove ϕ_1 is a tautology, then we prove $\phi_1 \vdash \phi_2, \phi_2 \vdash \phi_3, \dots, \phi_{\ell-1} \vdash \phi_\ell$ in extended Frege system and then $\Psi_m[\mathrm{eq}_\ell]$ follows from ϕ_ℓ. Since $\phi_1 = \Psi[\mathrm{eq}_1]\sigma_{1,1}$ must follow from an axiom, it can be proven directly. Next, we will use a series of succinct descriptions of proofs to show that $\phi_j \vdash \phi_{j+1}$. We will prove this by introducing the propositional variables $x^{(j,\cdot)}$ in ϕ_j using the extension rule, in terms of propositional variables $x^{(j+1,\cdot)}$ in ϕ_{j+1}. Namely, for each $j \in [\ell-1]$, there are two cases.

Case I: Handling Rules Except R_4. If eq_j is derived from an inference rule other than R_4 applied to $\mathrm{eq}_{i_1}, \mathrm{eq}_{i_2}, \dots, \mathrm{eq}_{i_c}$, where $i_1, i_2, \dots, i_c < i$ and c is a constant, then we introduce the propositional variables $x^{(j,i)}$ as follows.

- If $i \in \{i_1, i_2, \dots, i_c\}$, then let $x^{(j,i)}$ be $x^{(j+1,j+1)}$, if $\Psi^{(j+1,i)}_i$ holds. Otherwise, let $x^{(i,j)}$ be $x^{(j+1,i+1)}$. Formally, we introduce $x^{(j,i)}$ via the following extension.

$$x^{(j,i)} \leftrightarrow \left[(\llbracket \mathrm{eq}_i \rrbracket \sigma_{j+1,i} \land x^{(j+1,j+1)}) \lor (\neg \llbracket \mathrm{eq}_i \rrbracket \sigma_{j+1,i} \land x^{(j+1,i)}) \right].$$

- Otherwise, we set $x^{(j,i)} \leftrightarrow x^{(j+1,i)}$.

After we add above extension rules for $x^{(j,\cdot)}$, we next construct the following proofs and concatenate them.

1. $\underline{\text{Show that for any } i \in \{i_1, i_2, \dots, i_c\}, \Psi^{j,i}_i \vdash \Psi^{j+1,i}_i \circ \Psi^{j+1,j+1}_i}$. Intuitively, this is true because under the premise of $\Psi^{j,i}_i$, if $\llbracket \mathrm{eq}_i \rrbracket \sigma_{j+1,i}$ is false, then $x^{(j,i)} \leftrightarrow x^{(j+1,i)}$, and we can show that $\Psi^{j,i}_i \to \Psi^{j+1,i}_i$, which is a contradiction. Hence, under the premise of $\Psi^{j,i}_i$ we must have $\llbracket \mathrm{eq}_i \rrbracket \sigma_{j+1,i}$ and thus $x^{(j,i)} \leftrightarrow x^{(j+1,j+1)}$, which implies that $\Psi^{j,i}_i \vdash \Psi^{j+1,j+1}_i$.

2. Show that for any $i \notin \{i_1, i_2, \ldots, i_c\}$, $\Psi_i^{(j,i)} \vdash \Psi_i^{(j+1,i+1)}$. This follows from $x^{(j,i)} \leftrightarrow x^{(j+1,i)}$. Both this proof and the previous one can be described succinctly. To describe them, we can use substitution for succinct descriptions, and leverage the fact that the map $\sigma_{i,j} : x \mapsto x^{(i,j)}$ and its inverse can be computed by a succinct circuit.

3. Show that

$$\Psi_1^{j,1} \circ \Psi_2^{j,2} \circ \ldots \circ \Psi_j^{j,j} \vdash \Psi_1^{j+1,1} \circ \ldots \circ \Psi_{j+1}^{j+1,j+1} \circ (\Psi_{i_1}^{j+1,j+1} \circ \Psi_{i_2}^{j+1,j+1} \circ \ldots \circ \Psi_{i_c}^{j+1,j+1}),$$

by combing the 1 and 2.

This step can be done by $\mathsf{poly}(\ell)$ lines, and thus can be described succinctly by a $\mathsf{poly}(\ell)$-size circuit.

4. Show that $\Psi_{i_1}^{j+1,j+1} \circ \Psi_{i_2}^{j+1,j+1} \circ \ldots \circ \Psi_{i_c}^{j+1,j+1} \vdash \Psi_{j+1}^{j+1,j+1}$. This step needs to take care of inference rules R_1, R_2, R_3 and R_5 and axioms in PV, separately.

5. Prove that $\phi_i \vdash \phi_{i+1}$. This can be proven by combining 3 and 4.

Case II: Handling Substitution Rule R_4. If eq_j is derived from a substitution rule, i.e. $\mathrm{eq}_j = \mathrm{eq}_{i'}(v/y)$ for some $i' < j$, where v is a term and y is a variable in PV, then we introduce the variables $x^{(j,i)}$ as follows.

1. If $i = i'$, then similar to the argument above, we set

$$x^{(j,i)} \leftrightarrow \Big[(\llbracket \mathrm{eq}_i \rrbracket \sigma_{j+1,i} \wedge \sigma(x)^{(j+1,j+1)}) \vee (\neg \llbracket \mathrm{eq}_i \rrbracket \sigma_{j+1,i} \wedge x^{(j+1,i)}) \Big],$$

where σ is the substitution $y \mapsto v$ extended to propositional variables.

2. Otherwise, for $i \neq i'$, we set $x^{(j,i)} \leftrightarrow x^{(j+1,i)}$.

Then similar to the Case I, we can succinctly describe the following proofs and concatenate them.

1. Show that $\Psi_{i'}^{j,i'} \vdash \Psi_{i'}^{j+1,i'} \circ \Psi_{j+1}^{j+1,j+1}$. We first use the same idea as in Case I to show that $\Psi_{i'}^{j,i'} \to \Psi_{i'}^{j+1,i'} \wedge (\Psi_{i'}^{j+1,j+1}\sigma)$. Note that $\Psi_{i'}^{j+1,j+1}\sigma$ is the same as $\Psi_{j+1}^{j+1,j+1}$, since $\mathrm{eq}_j = \mathrm{eq}_{i'}\sigma$. Hence, we already have the desired proof.

2. Show that $\Psi_i^{j,i} \vdash \Psi_i^{j+1,i}$ for each $i \neq i'$. This can be shown in a similar way as we did in Case I.

3. Prove that $\phi_i \vdash \phi_{i+1}$. This can be obtained by combining the above two proofs via $\mathsf{poly}(\ell)$ lines, and hence can also be described succinctly.