Quasi-Linear Indistinguishability Obfuscation via Mathematical Proofs of Equivalence and Applications

Yaohua Ma[1], Chenxin Dai[2], Elaine Shi*[3]

[1]CMU and Tsinghua University
[2]CMU and Tsinghua University
[3]CMU, Oblivious Labs, and 0xPARC

Abstract

Indistinguishability obfuscation (iO) is a powerful cryptographic primitive and has been quoted as the "swiss army-knife of modern cryptography". Most prior works on iO focused on theoretical feasibility, and paid less attention to the efficiency of the constructions. As a result, all prior constructions stopped at achieving polynomial efficiency without worrying about how large the polynomial is. In fact, it has even been conjectured that a polynomial dependence on the input length is necessary.

In this work, we show that if the two circuits to be obfuscated enjoy a succinct propositional logic proof of equivalence, then we can create obfuscated versions of these programs that are computationally indistinguishable; and importantly, the obfuscated program's efficiency is quasi-linear in the circuit size and proof size. We show that our quasi-linear iO construction also leads to new applications. Specifically, we show how to achieve quasi-linear efficiency for 1) iO for Turing Machines with unbounded inputs, and 2) multi-input functional encryption, also assuming succinct proofs of equivalence.

*Author ordering is randomized.

Contents

1 Introduction

Indistinguishability obfuscation (iO) [GGH+13, JLS21] is a technique for hiding secrets in programs, such that given two functionally equivalent programs, their obfuscated counterparts are computationally indistinguishable. Because iO promises numerous theoretical and practical applications, the community has devoted much effort in the past decade to understand the *feasibility* of iO, culminating in recent ground-breaking results that showed how to construct iO from well-founded assumptions [JLS21]. In comparison, relatively little effort has been made to improve the *efficiency* of iO. To the best of our knowledge, all known *circuit* obfuscators incur *polynomial* overhead in the input length where overhead is measured in terms of the obfuscated program size and evaluation time. Not only so, since existing works largely focus on feasibility, they do not even care about quantifying the exact polynomial. Although there has been some progress in improving the efficiency of iO for Turing Machines (TMs) and Random Access Machines (RAMs) [AJS17, KLW15, BCG+18], these works mainly focus on making the obfuscated program size independent of the runtime of the original TM/RAM, and they do not remove the polynomial dependency on the input size. In this paper, we ask the following question:

Can we construct indistinguishability obfuscation whose cost is quasi-linear in the circuit size?

At first sight, there seems to be some inherent barrier towards achieving this. Specifically, it has been informally conjectured that there is an *input-length barrier* for any iO construction [JJ22]. As Jain and Jin described [JJ22], it seems that the security reduction for iO must check the equivalence of the two programs, and one way to achieve this is to iterate through all possible inputs, thus resulting in an exponential in input-length loss in the security proof. This loss requires that the underlying security primitives enjoy subexponential security such that they are secure against adversaries running in exponential in input-length time. The immediate efficiency implication is that the underlying cryptographic primitives must adopt a security parameter that is polynomial in the input length, and thus the resulting iO must suffer from polynomial in input-length efficiency loss.

Fortunately, as Jain and Jin observed [JJ22], in typical applications of iO, the programs to be obfuscated often have a *propositional logic proof of equivalence* that is not too much larger than the circuit size itself. Assuming that such a mathematical proof of equivalence is provided, Jain and Jin showed how to circumvent the conjectured "input-length barrier" and avoid the exponential in input-size blowup in the security reduction. Jain and Jin, however, still did not focus on optimizing the efficiency of the iO— like all prior works, they were satisfied with polynomial overhead. Inspired by Jain and Jin [JJ22], we refine our question to the following:

Assuming the existence of a propositional logic proof of equivalence, can we construct indistinguishability obfuscation whose cost is quasi-linear in the circuit size and the proof size?

1.1 Our Results and Contributions

Quasi-linear indistinguishability obfuscation with proofs of equivalence. We gave an affirmative answer to the above question. Specifically, we prove the following theorem:

Theorem 1.1 (Quasi-linear iO with proof of equivalence)**.** *Assume the following hard problems:*

- *polynomial (or subexponential resp.) hardness of Learning with Error (LWE);*
- *sub-exponentially secure one-way functions (OWF); and*

- *sub-exponentially secure indistinguishability obfuscation for circuits of size* $\widetilde{O}_\lambda(1)$.

Then, there exists a polynomially secure (or subexponentially secure resp.) indistinguishability obfuscator such that for circuit families of size at most N_{circ} *with propositional proof of size at most* N_{proof}*, the obfuscated program size and evaluation time*[1] *is upper bounded by* $\widetilde{O}_\lambda(N_{\mathrm{circ}} + N_{\mathrm{proof}})$*, where* $\widetilde{O}_\lambda(\cdot)$ *hides polylogarithmic terms and polynomial dependency on the security parameter* λ.

Clearly, the obfuscated program size and evaluation time has to be at least N_{circ}. Thus, our result enjoys nearly optimal dependency on the circuit size. As Jain and Jin [JJ22] showed, in numerous applications of iO, the circuits we care about have propositional proofs of equivalence whose size is roughly the size of the original circuit, i.e., $N_{\mathrm{proof}} = \widetilde{O}(N_{\mathrm{circ}})$. In such cases, the cost of our iO is simply $\widetilde{O}_\lambda(N_{\mathrm{circ}})$. Our work leaves open the natural question whether we can remove the quasi-linear dependency on N_{proof}.

Application: quasi-linear multi-input functional encryption. We show that our quasi-linear indistinguishability obfuscator (assuming proofs of equivalence) implies Multi-Input Functional Encryption (MIFE) [GGG+14] with quasi-linear efficiency (also assuming proof of equivalence for the family of functions supported). Previous work has shown how to get single-input functional encryption with near optimal efficiency [JLL23]; however, to the best of our knowledge, no existing work can achieve MIFE whose efficiency is quasi-linearly dependent on the circuit size. In all previous constructions of MIFE [GGG+14, GJO16], the decryption time is polynomially dependent on the circuit size, and no work has even attempted to quantify how large the polynomial is.

More concretely, we have the following theorem.

Theorem 1.2 (MIFE with quasi-linear efficiency)**.** *Assume the same assumptions as Theorem 1.1 (the subexponential security version). Fix some message length* $L = \mathsf{poly}(\lambda)$*, and consider a family of* t*-ary functions* $\mathcal{F}$ *which can be represented by circuits of size at most* N_{circ}*. Further, suppose that for any* $f \in \mathcal{F}$*, any subset of indices* $J \subset [t]$*, any two message vectors* $\mathbf{x}_J, \mathbf{x}'_J \in (\{0,1\}^L)^{|J|}$ *such that* $f(\mathbf{x}_J, \cdot) = f(\mathbf{x}'_J, \cdot)$*, the equivalence of* $f(\mathbf{x}_J, \cdot)$ *and* $f(\mathbf{x}'_J, \cdot)$ *can be shown with a length-*N_{proof} *proof in* $\mathcal{EF}$*. Then, there exists a subexponentially secure MIFE scheme for the function family* $\mathcal{F}$ *whose key generation and decryption times are bounded by* $\widetilde{O}_\lambda(N_{\mathrm{circ}} + N_{\mathrm{proof}} + L \cdot \mathsf{poly}(t))$.

The requirement that the function family $\mathcal{F}$ admits a succinct $\mathcal{EF}$ proof of equivalence also seems inherent if the conjectured input-length barrier for iO is true [JJ22]. Specifically, we can construct iO from even MIFE for only 2-ary functions, and the construction is efficiency-preserving.

Application: quasi-linear iO for Turing Machines. Until the work of Jain and Jin [JJ22], all prior constructions of iO for Turing Machines (TMs) [AJS17, BFK+19, JLL23] require knowledge of an a-priori bound on the input length at obfuscation time. In fact, as Jain and Jin explained [JJ22], there is a folklore belief that the bounded input length assumption is inherent due to reasons closely related to the aforementioned input-length barrier — see their work for a more detailed explanation [JJ22]. Interestingly, Jain and Jin [JJ22] showed, for the first time, how to circumvent this bounded input-length barrier by relying on the existence of proofs of equivalence between the TMs. Unfortunately, Jain and Jin's iO for TMs construction is not efficient, and the evaluator's running time is polynomially dependent on the original TM's running time (and also polynomially dependent on the $\mathcal{EF}$ proof size when translated from the PV proof for a given input length).

Using our quasi-linear circuit iO scheme (assuming proofs of equivalence), we show how to achieve iO for TMs with quasi-linear efficiency (also assuming proofs of equivalence). Like Jain

[1] The evaluation time of the obfuscated program is characterized in the Random Access Machine (RAM) model.

and Jin, our scheme also supports unbounded input length. Specifically, we prove the following theorem:

Theorem 1.3 (iO for TMs with quasi-linear efficiency)**.** *Assume the same assumptions as Theorem 1.1 (the subexponential security version). Let $\mathcal{M}$ be a family of Turing Machines (TMs) whose description sizes are bounded by desc, whose running times are bounded by $T(L)$ on length-L inputs. Further, suppose that $\mathcal{M}$ admits size-$N_{\text{proof}}(L)$ proofs of equivalence in $\mathcal{EF}$. Then, there exists a subexponentially secure indistinguishability obfuscator for the family $\mathcal{M}$ such that the obfuscated program size is $O(\mathsf{desc}) + \mathsf{poly}(\lambda)$, the evaluator's runtime on a length-L input is $\widetilde{O}_\lambda(T(L) \cdot \mathsf{poly}(\mathsf{desc}) + N_{\text{proof}}(L))$, and the obfuscator's runtime is $\mathsf{desc} \cdot \mathsf{poly}(\lambda)$. Further, the obfuscator need not know an a-priori bound on the input length.*

In the above, we say that $\mathcal{M}$ admits length-$N_{\text{proof}}(\cdot)$ $\mathcal{EF}$ proofs of equivalence, if for any two functionally equivalent TMs $M, M' \in \mathcal{M}$, there exists a uniform machine M^{prove} that can generate a length-$N_{\text{proof}}(L)$ proof of equivalence in $\mathcal{EF}$ for $[\![M]\!]_L$ and $[\![M']\!]_L$ which represent the induced (uniform) circuits that describe the computation of M and M' respectively on inputs of length L.

Note that the work of Jain and Jin [JJ22] uses PV proofs between the TMs. In our work, we switch to using $\mathcal{EF}$ proofs for the TM-induced circuits when fixing some input length — this is because the size of the $\mathcal{EF}$ proofs is what directly contributes to the cost of the resulting iO scheme. By contrast, Jain and Jin [JJ22] only care about achieving polynomial efficiency so they do not need such fine-grained accounting. Jain and Jin show that given a PV proof of equivalence, one can expand it into an $\mathcal{EF}$ proof (for the induced circuits) when fixing an input length; further, the resulting $\mathcal{EF}$ proof can be generated by a uniform machine. However, the PV to $\mathcal{EF}$ conversion may introduce some extra blowup in the proof size, so we directly use the $\mathcal{EF}$ proof size for more fine-grained accounting.

1.2 Technical Highlight

Background: Jain and Jin's blueprint. The elegant work of Jain and Jin [JJ22] showed how to avoid the input-length barrier for constructing iO, assuming mathematical proofs of equivalence. Their blueprint is as follows:

1. First, Jain and Jin define the notion of a pair of *s-equivalent* circuits. Two functionally and topologically equivalent circuits C and C' are said to be s-equivalent, iff they are otherwise identical except for a small subcircuit that contains at most s gates denoted S, and further, the subcircuits C_S and C'_S are functionally equivalent where C_S (or C'_S resp.) means the subcircuit of C (or C' resp.) induced by the gates S.

2. Jain and Jin then show that given two circuits C and C' of size at most N_{circ} with an Extended Frege ($\mathcal{EF}$) proof of equivalence of size at most N_{proof}, they can construct padded circuits $\overline{C} = \mathsf{Pad}(C, N_{\text{circ}}, N_{\text{proof}})$ and $\overline{C}' = \mathsf{Pad}(C', N_{\text{circ}}, N_{\text{proof}})$ with the following property:

   There exists a sequence of $m = \mathsf{poly}(N_{\text{circ}} + N_{\text{proof}})$ functionally and topologically equivalent circuits $C_1, C_2, \ldots, C_m$ beginning and ending at $C_1 = \overline{C}$ and $C_m = \overline{C}'$ respectively, such that any adjacent pair of circuits C_i and C_{i+1} are $O(\log(N_{\text{circ}} + N_{\text{proof}}))$-*equivalent*. ($\star$)

3. Additionally, Jain and Jin show how to construct a *locally indistinguishable obfuscator* (denoted LiO below) such that $\mathsf{LiO}(C_1)$ and $\mathsf{LiO}(C_2)$ only need to be indistinguishable if C_1 and C_2 are $O(\log(N_{\text{circ}} + N_{\text{proof}}))$-equivalent. This locally indistinguishable obfuscator adopts a

gate-by-gate obfuscation approach: it first augments each gate g with some extra cryptographic checks and operations, resulting in an augmented gate Gate^g, and then obfuscates the augmented gate Gate^g. The cryptographic augmentation of the gate ensures that the adversary can only evaluate the circuit based on the prescribed topology, i.e., it cannot perform a mix-and-match attack by altering the wiring of the gates at evaluation time.

4. Finally, the construction $\mathsf{iO}(\cdot) := \mathsf{LiO}(\mathsf{Pad}(\cdot, N_{\mathrm{circ}} + N_{\mathrm{proof}}))$ gives an obfuscation scheme that achieves indistinguishability for any two circuits of size at most N_{circ} and with an $\mathcal{EF}$ proof of equivalence of size at most N_{proof}.

Why Jain and Jin's construction is inefficient. Jain and Jin's construction suffers at least $(N_{\mathrm{circ}} + N_{\mathrm{proof}})^2$ obfuscated program size and at least $(N_{\mathrm{circ}} + N_{\mathrm{proof}})^{4+\varepsilon}$ evaluation time (where $\varepsilon > 0$ is an arbitrarily small constant) when instantiated with the state-of-the-art non-interactive batched argument (BARG) scheme by Waters and Wu [WW22]. The inefficiency is due to the following reasons:

1. First, the padded circuit's size is at least $N_{\mathrm{padded}} := (N_{\mathrm{circ}} + N_{\mathrm{proof}})^2$, which implies that their obfuscated circuit size is at least $N_{\mathrm{padded}} := (N_{\mathrm{circ}} + N_{\mathrm{proof}})^2$, and

2. An adversary may launch a mix-and-match-style attack by using intermediate results from one evaluation in another evaluation with different inputs to the circuit. To prevent such a mix-and-match attack, Jain and Jin require that for every gate g, for every input and output wire w of the gate g, the evaluator hashes (using a suitable hashing scheme called somewhere statistically extractable hash) all wire values that w depends on, and then computes a proof of consistency that all input and output wires of the gate have hash values that correspond to a consistent set of wire values. The augmented gate Gate^g expects to receive this consistency proof and always verifies its validity before continuing with the computation. To achieve this, for each of the up to $O(N_{\mathrm{padded}})$ gates, their evaluator needs to compute $O(1)$ non-interactive batched argument (BARG) proofs for up to N_{padded} circuits of size $\widetilde{O}(1)$. This implies that their evaluator's runtime is at least $O(N_{\mathrm{padded}}) \cdot T_{\mathrm{BARG}}(N_{\mathrm{padded}}, \widetilde{O}(1))$, which is at least $\widetilde{O}(N_{\mathrm{padded}}^2) = \widetilde{O}((N_{\mathrm{circ}} + N_{\mathrm{proof}})^4)$ if a BARG with ideal efficiency could be constructed. If we instantiate the BARG with the most efficient existing scheme [WW22], then $T_{\mathrm{BARG}}(n_{\mathrm{idx}}, n_{\mathrm{circ}}) = n_{\mathrm{idx}}^{1+\varepsilon} \mathsf{poly}(\lambda, n_{\mathrm{circ}})$ — consequently, the evaluator's runtime would be $\widetilde{O}((N_{\mathrm{circ}} + N_{\mathrm{proof}})^{4+\varepsilon})$.

Our novel ideas. To avoid the inefficiencies in Jain and Jin's construction, we devise the following new ideas.

1. *A quasi-linear padding algorithm.* We devise a new padding algorithm that achieves the same property ($\star$) achieved by Jain and Jin's padding algorithm; however we avoid the quadratic blowup of Jain and Jin, and our padded circuits are of size only $\widetilde{O}(N_{\mathrm{circ}} + N_{\mathrm{proof}})$.

2. *An efficient somewhere statistically extractable hash with prefix consistency proofs.* We construct an efficient somewhere statistically extractable hash (SEH) scheme with consistency proofs. While Jain and Jin's construction allows proving consistency on an arbitrary subset of indices, we observe that it is sufficient to be able to prove consistency on prefixes. This allows us to construct an SEH hash scheme without having to rely on generic BARGs to produce consistency proofs. In fact, we use the SEH construction of Hubacek and Wichs [HW15] in a non-black-box manner. Since their construction combines Fully Homomorphic Encryption

(FHE) and a Merkle-tree-type structure, we can construct consistency proofs on prefixes by giving out the "hashes" on only logarithmically many nodes in the tree. Further, we show that the evaluator can compute all hashes on all gates as well as their consistency proofs in quasi-linear time, using an incremental algorithm.

Combining the above ideas, we show that the obfuscated program and the evaluator runtime can both be reduced to $\widetilde{O}_\lambda(N_{\mathrm{circ}} + N_{\mathrm{proof}})$.

Quasi-linear iO for Turing Machines. Jain and Jin's iO for TM construction [JJ22] does not directly work for us. Informally speaking, in their construction, they obfuscate a universal gate, which, upon receiving an input length bound L and a gate index gidx, outputs the gidx-th augmented gate in the padded circuit, and then emulates the execution of the augmented gate. In other words, Jain and Jin implicitly assume that there is a uniform circuit that can generate the gidx-th gate of the padded circuit upon receiving the input length L and gate index gidx. Indeed, with Jain and Jin's (inefficient) padded circuit, this is possible. By contrast, our quasi-linearly sized padded circuit involves a routing network, and the routing decisions of each gate inside the routing network cannot be locally determined without having a global view.

We side-step this problem by adopting a different approach for obfuscating TMs. Specifically, we use a two-layered approach as described below. We consider a universal obfuscator UObf, which, upon receiving a specific input length L, computes the TM-induced circuit for input length L, pads the circuit, and then outputs a locally indistinguishable obfuscator LiO for the padded circuit. We then use a state-of-the-art RAM indistinguishability obfuscator [JLL23] denoted RAMObf to obfuscate UObf.

Although the underlying RAMObf's evaluation time has a polynomial dependence on the input length, it does not matter for us because the input length of UObf is small. Specifically, since the original TM's running time is polynomially bounded in λ, the input length of UObf cannot exceed $\log^2 \lambda$. On the other hand, underlying RAMObf's evaluation time is only quasi-linearly dependent on the original RAM's running time which is important for ensuring the evaluation efficiency — in our case, the program UObf's running time has a quasi-linear dependence on the original TM's running time T and the $\mathcal{EF}$ proof size N_{proof}.

We prove this two-layered construction secure in Section 7.4.

MIFE with quasi-linear efficiency. Our MIFE construction is inspired by that of Goyal et al. [GJO16]. The hope is to use LiO in place of standard iO to get the dependence on the circuit size to be quasi-linear. However, it turns out that directly replacing iO with LiO is not enough for getting our result. Specifically, Goyal et al. [GJO16] suffer additionally from a polynomial dependence on the message length L, and we have to make some non-trivial modifications to improve the dependence on L to be quasi-linear. In Goyal et al. [GJO16], due to a large exponential loss in their security reduction, they require that the security parameters of the underlying cryptographic building blocks should be as large as $\mathsf{poly}(t \cdot L)$ where t is the arity of the function and L is the length of each input message. To avoid this large blowup, one main new trick we used is to have a short PRF label (length-$\mathsf{poly}(\lambda)$ only) for each of the L bits and an extra short PRF label to assert the consistency of bits labels, instead of a long PRF label for whole message in Goyal et al.'s approach [GJO16].

Besides getting rid of the polynomial dependence on L, another technicality in our proofs is that in every step of the hybrid sequence that relies on the security of the obfuscator, we need to argue that the corresponding two programs to be obfuscated enjoy a succinct $\mathcal{EF}$ proof of equivalence.

1.3 Additional Related Work

We now review some additional related work.

Gate by gate obfuscation. Besides Jain and Jin and our work, the gate-by-gate obfuscation approach has also been used in a couple prior works. The work of Fernando et al. [FSS+23] also adopts this approach to get a quasi-linear non-interactive anonymous router (NIAR) scheme. From a technical perspective, their work is akin to constructing a function-hiding multi-input functional encryption (MIFE) scheme for a "permutation network" functionality. The reason why they take a gate-by-gate obfuscation approach is to get quasi-linear efficiency, since using prior iO schemes to directly obfuscate the permutation network would result in polynomial overhead. Although with our new quasi-linear iO with proof of equivalence, it might be possible to simplify the construction of Fernando et al. [FSS+23] by directly obfuscating the entire permutation network, Fernando et al. [FSS+23]'s result is still better from a theoretical perspective since they get their result from polynomially secure hardness assumptions (whereas our paper relies on subexponential assumptions). Partly, this is because Fernando et al.'s proof is tightly coupled with the special functionality of a permutation network whereas our approach works for generic functionalities.

Canetti et al. [CLTV15] also used a network of obfuscations, and their goal is to build leveled fully homomorphic encryption from iO. However, the proof techniques in Jain and Jin [JJ22] and our work are of a different nature from Canetti et al. [CLTV15] since Canetti et al. aims to build a different primitive.

Somewhere statistically binding hash. Our somewhat extractable hash with prefix consistency combines a Merkle tree with Fully Homomorphic Encryption (FHE), inspired by the elegant work of Hubacek and Wichs [HW15]. Brakerski et al. [BBK+23] and Freitag et al. [FWW23] propose a related notion called function-binding extractable hash functions. Specifically, these works allow running the setup with a function f, such that the resulting hash will be statistically binding to $f(x)$ where x is the input of the hash. By combining Merkle tree and FHE, the two works show how to construct function-binding extractable hash for 1) disjunctions of block functions; and 2) bit-fixing predicates. Despite the similarity in techniques, our extractable hash has a different abstraction from Brakerski et al. [BBK+23] and Freitag et al. [FWW23], since what we want is not to statistically bind to some function of the input, but to 1) statistically bind to a set of indices; and 2) support an efficient proof mechanism that the inputs of two hash digests share a common prefix. Importantly, we also need an efficient batched algorithm that generates the hashes, consistency proofs, and opening proofs on all wires of the circuit in time that is quasilinear in the circuit size.

2 Preliminaries

2.1 Circuits

In this work, we view circuits as directed acyclic graphs, where each vertex represents a gate, and each edge represents a wire. We say two circuits are topological equivalent if their graph representations are identical, while the operation of each gate may be different.

We say a circuit has d_{in} fan-in, if each gate has at most d_{in} input wires; and d_{out} fan-out if each gate has at most d_{out} output wires.

Let C be a circuit, a subcircuit S of C includes a subset of all gates. We can define the input and output wires of S in a similar way:

- **Input Wire:** We say a input wire of some gate $g \in S$ is an input wire of S, if it isn't an output wire of another gate $g' \in S$. We denote the set of input wires of S as $\mathsf{inp}(S)$.
- **Output Wire:** We say an output wire of some gate $g \in S$ is an output wire of S, if it isn't an input wire of another gate $g' \in S$. We denote the set of output wires of S as $\mathsf{out}(S)$.
- **Functionality:** Circuit C induces a circuit $C_S : \{0,1\}^{|\mathsf{inp}(S)|} \to \{0,1\}^{|\mathsf{out}(S)|}$ for subcircuit the S.

2.2 Extended Frege Proofs

An extended Frege proof system ($\mathcal{EF}$) is a propositional logic system defined as follows. The system can be described with *variables*, *formulas*, and *inference rules*. Variables are represented by letters, and they can take values from $\{\mathsf{True}, \mathsf{False}\}$. Formulas are defined inductively: any variable is a formula; and if u and v are formulas, then $u \to v$ and $\neg u$ are formulas where $\to$ denotes "implies" and $\neg$ denotes negation. With the operators $\to$ and $\neg$, we can also define other boolean operators such as $\wedge$, $\vee$, and $\oplus$. The inference rules include:

- *Axiom 1:* $\vdash P \to (Q \to P)$;
- *Axiom 2:* $\vdash (P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$;
- *Axiom 3:* $\vdash \neg\neg P \to P$;
- *Modus Ponens*: $P, P \to Q \vdash Q$.

Notation for substitution. 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 $A\sigma = (a \wedge b) \to ((a \vee b) \to (a \wedge b))$.

Definition of proof. An $\mathcal{EF}$ proof is a sequence of formulas (also called proof lines) $\theta_1, \ldots, \theta_{N_{\text{proof}}}$ where each θ_i is a formula of one of the following forms:

1. *Inference:* $\theta_i = B\sigma$ which is derived from form $A_1\sigma, \ldots, A_k\sigma \vdash B\sigma$ (or simply $\vdash B\sigma$) given some inference rule $A_1, \ldots, A_k \vdash B$ (or simply $\vdash B$) defined above and some substitution σ. Moreover, $A_1\sigma, \ldots, A_k\sigma$ must have appeared earlier in $\theta_1, \ldots, \theta_{i-1}$.
2. *Extension:* of the form $v \leftrightarrow A$, where A is some formula, and the variable v has not appeared in $\theta_1, \ldots, \theta_{i-1}$. Specifically, $a \leftrightarrow b$ is the abbreviation of $(a \to b) \wedge (b \to a)$.

If we do not allow the extension rule above, i.e., if we do not allow a proof line θ_i to be of the form $v \leftrightarrow A$, the resulting system is commonly called the Frege system. However, the Frege system has the limitation that each proof line must be a formula, and expressing computation as formulas may not be efficient. The extension $v \leftrightarrow A$ allowed by $\mathcal{EF}$ effectively allows us to represent each proof line as a circuit rather than a formula.

Assumption on the length of a proof line. Without loss of generality, we may assume that *in an $\mathcal{EF}$ proof, each proof line is of constant length.* If not, we can always use the extension rule to introduce intermediate variables and break up the proof into constant-sized proof lines, and this transformation causes only a constant blowup in the total proof size.

With this assumption, we can measure the *size* or *length* of an $\mathcal{EF}$ proof simply by the number of proof lines.

Expressing an $\mathcal{EF}$ proof as a circuit. Conversely, since the extension $v \leftrightarrow A$ is introduced such that we can express each proof line as a circuit rather than formula. In other words, given an $\mathcal{EF}$ proof, we can naturally encode the computation of the proof as a circuit, leading to the following lemma.

Lemma 2.1. *Given a proof $(\theta_1, \ldots, \theta_{N_{\text{proof}}})$ in $\mathcal{EF}$ of size N_{proof}, we can construct a circuit with fan-in 2 and unbounded fan-out, where the inputs represent the variables, and there are N_{proof} output wires, where the i-th output wire corresponds to θ_i in the following sense:*

- *if $\theta_i = B\sigma$ is obtained from some inference rule $A_1\sigma, \ldots, A_k\sigma \vdash B\sigma$, then the i-th output wire encodes $B\sigma$ (note also that $A_1\sigma, \ldots, A_k\sigma$ must correspond to some earlier output wires);*
- *else if θ_i is an extension of the form $v \leftrightarrow A$, then the i-th output wire constantly outputs 1.*

Further, the circuit satisfies the following:

1. *its circuit size is $O(N_{\text{proof}})$;*
2. *given any proof line $\theta_j = B\sigma$ obtained from[2] $A_1\sigma, \ldots, A_k\sigma \vdash B\sigma$ (or from $\vdash B\sigma$) where σ denotes the substitution $P_1 \mapsto Q_1, \ldots, P_\ell \mapsto Q_\ell$, let $w_{A_1\sigma}, \ldots, w_{A_k\sigma}, w_{B\sigma}$ denote the wires of the circuit that represent the formulas $A_1\sigma, \ldots, A_k\sigma, B\sigma$, respectively. Then, there is a subcircuit of constant size such that its inputs represent the formulas $Q_1, \ldots, Q_\ell$, and its outputs are the wires $w_{A_1\sigma}, \ldots, w_{A_k\sigma}$, and $w_{B\sigma}$. Further, while the input wires $Q_1, \ldots, Q_\ell$ may be consumed arbitrarily many times in this subcircuit, all gates in this subcircuit have fanout only 1.*

Proof. We can view an $\mathcal{EF}$ proof in the most natural manner as a constant fan-in, *unbounded fan-out* circuit that computes all the formulas in the proof. The circuit has constant fan-in, because all formulas are of constant size in the proof. However, the circuit may have unbounded fan-out, because each variable defined may be referenced an arbitrary number of times. Given a circuit is constant fan-in and unbounded fan-out, we can always convert it to a circuit with fan-in 2 and unbounded fan-out with only a constant blowup in the circuit size.

To complete the proof, we want to make sure that in the subcircuit that computes $A_1\sigma, \ldots, A_k\sigma, B\sigma$ from $Q_1, \ldots, Q_\ell$, all gates have fan-out 1 while the input wires $Q_1, \ldots, Q_\ell$ can still be consumed arbitrarily many times. Since this subcircuit is of constant size, we can always perform a transformation to make sure every gate has only a single fanout and fan-in 2, at the cost of a constant blowup in size. In particular, if any gate's output wire needs to be consumed again, we can copy the entire circuit gadget that led to this output wire again, at the cost of a higher fan-out for the input wires $Q_1, \ldots, Q_\ell$, and more gates in the subcircuit. □

Proof of equivalence for two circuits. Given a circuit C whose input wires are labeled $\mathbf{x} := (x_1, \ldots, x_n)$, for each gate in C whose input wires are labeled $a_1, \ldots, a_k$ and whose output wires are labeled $b_1, \ldots, b_m$, we can write down a set of extensions: for each $j \in [m]$, $b_j \leftrightarrow f_j(a_1, \ldots, a_k)$. Here we use the labels of the wires to directly name the variables in the extensions. We use the notation $\mathsf{Prop}[C](\mathbf{x})$ to denote all extensions obtained in this manner for circuit C and input $\mathbf{x}$. Given two circuits C and C' with the same number of inputs, a propositional proof of equivalence between C and C' is an $\mathcal{EF}$ proof that contains the following proof lines (we assume that all other wires in the two circuits have distinct labels except they share the same input wires):

- The proof includes the extensions $\mathsf{Prop}[C](\mathbf{x})$ and $\mathsf{Prop}[C'](\mathbf{x})$;

[2] k must be a constant by our formulation.

- Suppose that $o_1, \ldots, o_k$ and $o'_1, \ldots, o'_k$ denote the output wires of C and C' respectively, then the proof includes the lines $o_1 \leftrightarrow o'_1$, …, $o_k \leftrightarrow o'_k$.

The following lemma is implied by Lemma 2.1.

Lemma 2.2. *Let $c > 0$ be an appropriate universal constant. Given two circuits C and C' of maximum size N_{circ}, and an $\mathcal{EF}$ proof of equivalence denoted $(\theta_1, \ldots, \theta_{N_{\text{proof}}})$ of size N_{proof}, there is a circuit denoted C_{proof} with fan-in 2 and unbounded fanout that satisfies the following:*

- C_{proof} has exactly $c \cdot N_{\text{proof}}$ number of gates;
- C_{proof}'s inputs are the wires of C and C', padded with filler input wires to exactly $2N_{\text{circ}}$ input wires;
- C_{proof}'s outputs correspond to the proof lines in the same manner as Lemma 2.1. Without loss of generality, we may assume that if we sort the gates of C_{proof} in a topological order, the output wires of C_{proof} are the output wires of the last N_{proof} gates.
- Consider the union circuit $C_{\text{union}} = C \cup C' \cup C_{\text{proof}}$ where C and C' share the same input wires, and C_{proof}'s input wires are directly connected to the wires of C and C'. Then, property 2 of Lemma 2.1 holds for C_{union}, where the subcircuit may involve gates in C, C' and C_{proof}.

Given two identical circuits, we can construct a trivial $\mathcal{EF}$ proof of equivalence as stated in the lemma below.

Lemma 2.3 (Trivial proof of equivalence for identical circuits)**.** *Given two identical circuits C and C of size at most N_{circ}, there is an $\mathcal{EF}$ proof of equivalence of size $O(N_{\text{circ}})$.*

2.3 s-Equivalent Circuits

We define the notion of s-equivalence and transitive s-equivalence (a.k.a. δ-equivalence and $\Delta-$*equivalence* in Jain and Jin [JJ22][3]). Intuitively, two circuits are s-equivalent, if they are almost identical except a small subcircuit of size at most s:

Definition 1 (s-equivalence)**.** We say two topologically equivalent circuits C and C' are s-equivalent (via some subcircuit S), there is some small subcircuit S of size at most s, such that all gates outside S are identical in C and C', and moreover, C_S and C'_S are functionally equivalent, where C_S and C'_S denote the induced subcircuit on the set of gates S for C and C', respectively.

Two topologically identical circuits are transitively s-equivalent, if one can be transformed to another by a polynomially long sequence of s-equivalent transformations:

Definition 2 (Transitive s-equivalence)**.** We say two topological equivalent circuits C and C' (each of size n) are transitively s-equivalent via ℓ hybrids, if there exist ℓ circuits $C_1, C_2, \ldots, C_\ell$, such that $C_1 = C$, $C_\ell = C'$, and for each $i \in [\ell - 1]$, C_i and C_{i+1} are s-equivalent. In particular, if $\ell(n)$ is polynomially bounded in n, we abbreviate and say that C and C' are transitively s-equivalent.

Note that by Definition 1 and Definition 2, any two circuits that are s-equivalent or transitively s-equivalent must be functionally equivalent.

[3] We find it convenient to rename "δ-equivalence and Δ-equivalence parametrized by subcircuit size s" to simply s-equivalence and transitive s-equivalence because 1) δ and Δ are actually not a parameter in their definitions [JJ22], and 2) δ-equivalence and Δ-equivalence cannot be distinguished in spoken language.

2.4 Routing Network

A routing network is a circuit which receives n items as input, denoted $(x_1, \ldots, x_n)$, and outputs n items where each output is one of the inputs, and duplicate outputs are allowed. We consider a routing network where each gate receives two items denoted (y_0, y_1), and it has two outputs chosen among four possible configurations $\{(y_0, y_0), (y_0, y_1), (y_1, y_0), (y_1, y_1)\}$. We can imagine that the n inputs of a routing network correspond to n sources, and the n outputs correspond to n consumers. Each consumer wants to consume one of the n input items, and it is possible that multiple consumers want the same item. A routing network should satisfy the following property: no matter what the consumers want, it should be possible to configure the gates in the network such that all n consumers are satisfied.

Theorem 2.4 ([BCP15]). *There exist a routing network with n inputs and n outputs which satisfies following properties:*

- *The routing network is a circuit of fixed topology that only depends on n with size $O(n \log n)$ and depth $O(\log n)$ (namely, the item only traverses through $O(\log n)$ gates), and the topology can be determined in $O(n \log n)$ time without knowledge of consumers.*
- *Given demands of every consumer, the configuration of each gate can be determined in total time $O(n \log n)$.*

3 Transitive Logarithmic Equivalence from Propositional Proofs

3.1 Notations and Assumptions

Assumptions on the circuit. Without loss of generality, we may make the following assumptions on the circuit.

Without loss of generality, we may assume that in the input circuit C, each gate is of fan-in 2 and unbounded fan-out. Although a gate may have unbounded fan-out, we assume that each gate computes only a single output value which may later be consumed arbitrarily many times. We require the output circuit (i.e., the padded circuit) to have constant fan-in and constant fan-out.

We also assume that the input circuit C has exactly N_{circ} gates, and when sorted in topological order, the final output wire of the circuit C is the first output of the last gate. If C actually has fewer than N_{circ}, we can always pad it with some filler gates (whose output wires are unconsumed) to exactly N_{circ} gates.

Unless otherwise noted, we measure the size of the circuit by the number of wires it has (including input and output wires), since the number of gates is upper bounded by the number of wires. Sometimes if a circuit has some input wires that are not consumed by any gates, it is possible that the number of wires is much larger than the number of gates. Whenever we want to count the number of gates rather than wires, we will say so explicitly.

3.2 Padding Algorithm

Let $\mathcal{C}$ be a circuit family whose size is bounded by N_{circ}. Given two equivalent circuits $C_1, C_2 \in \mathcal{C}$ with an $\mathcal{EF}$ proof of equivalence of maximum size N_{proof}, we define a padding algorithm denoted Pad, such that 1) for $b \in \{1, 2\}$, C_b and $\mathsf{Pad}(C_b, N_{\text{circ}}, N_{\text{proof}})$ are functionally equivalent; and 2) $\mathsf{Pad}(C_1, N_{\text{circ}}, N_{\text{proof}})$ and $\mathsf{Pad}(C_2, N_{\text{circ}}, N_{\text{proof}})$ are transitively $O(\log(N_{\text{circ}} + N_{\text{proof}}))$-equivalent.

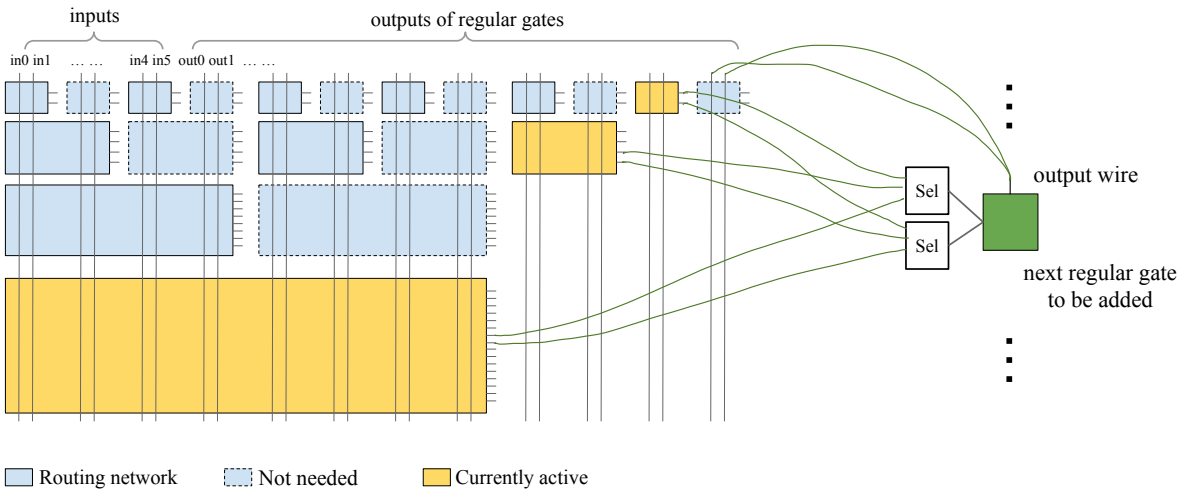


Figure 1: PadSingle **algorithm.** Each routing network's output wires on the right. The vertical wires simply pass through and are fed into logarithmically many routing networks. In the actual circuit, we will use a logarithmically sized binary tree (called a copy gadget) to copy each wire logarithmically many times such that the resulting circuit has constant fan-out.

3.2.1 Subroutine PadSingle

The PadSingle(C, N) subroutine takes a circuit C with an a-priori upper bound N on the number of gates, and converts C to a circuit of a fixed topology that depends only on N and the number of input wires of C. Further, the output circuit is $\widetilde{O}(N)$ in size.

Our PadSingle algorithm. We will sort all gates in C (henceforth called *regular gates* to distinguish them from the routing network gates) in a topological order, and add them one by one to the output circuit in a topological order like in Figure 1. Whenever we add the next gate, each of its two input wires can choose among 1) all inputs to the circuit, and 2) all outputs of previously added gates. The naïve approach, which is also Jain and Jin's approach [JJ22], is to have a selector gadget for each input wire, to choose among all the possible wires as mentioned above — this can be accomplished by building a binary tree of pairwise selector gates. However, since there can be up to N_{circ} wires to choose from, this naïve approach would result in a quadratic blowup in the output circuit size.

We will use the help of routing networks and an idea inspired by Bentley and Saxe's hierarchical data structure [BS80] to accomplish the same but more efficiently, resulting in a quasi-linear-sized padded circuit. The idea is illustrated in Figure 1. More concretely,

1. Imagine that we have a pool of wires — one can view a wire as a variable that is either an input to the circuit, or an output of a previous gate. Initially, this pool contains only input wires to the circuit C. As we add regular gates one by one in a topological order, we add each regular gate's output wires to the pool in order, and each output wire is replicated twice when being added to the pool.
2. Whenever the pool accumulates the next batch of 2^i wires, we will create a new routing network to route the correct signal to the input selectors of the next 2^{i-1} regular gates in order.

   In Figure 1, we show each wire to be fed into logarithmically many routing networks, but in the actual construction, we can use a binary-tree of logarithmic size (called a copy gadget) to

copy each wire logarithmically many times, so that the resulting circuit has constant fan-out.

3. For each gate that is being added next, its two input wires are connected to the output of a selector gadget, which selects from logarithmically many wires output from the routing networks. Such a selector gadget can be build using a binary tree of pairwise selector gates. Since the routing networks can route the input wires arbitrarily, essentially it is possible for each input wire to select from all wires in the pool so far.
4. Configure all routing networks gates such that all regular gates' input selectors receive the correct signal.
5. Finally, define the output of C to be the first output wire of the last regular gate in the padded circuit.

Note that since the routing network we use may replicate an input signal multiple times, this allows us to support an input circuit C with potentially unbounded fan-out.

When N is a power of 2, it is not hard to see that all output wires of routing networks will be consumed by the input selector of some regular gate (assuming that in the above Step 2, we stop creating new routing networks when there are no more gates to add).

Efficiency. Suppose we use a routing network that is $O(n \log n)$ in size for n inputs. In the above $\mathsf{PadSingle}(C, N)$ algorithm, we consume a routing network of size $O(2^i \log(2^i))$ every time we gain wires 2^i wires in the pool. Therefore, the total size of the padded circuit is upper bounded by

$$\sum_{i=1}^{\log N} 2^i \log(2^i) \cdot \frac{N}{2^i} = O(N \log^2 N)$$

3.2.2 The Pad Algorithm

We now describe the algorithm $\mathsf{Pad}(C, N_{\text{circ}}, N_{\text{proof}})$. The padded circuit is constructed as follows — see also Figure 2:

- Call $\widetilde{C}_1 := \mathsf{PadSingle}(C, N_{\text{circ}})$ to convert the circuit C to a fixed topology.
- Call $\widetilde{C}_2 := \mathsf{PadSingle}(\mathbf{1}, N_{\text{circ}})$ to create a placeholder circuit of a fixed topology. The circuits $\widetilde{C}_1$ and $\widetilde{C}_2$ share the same input wires. The notation $\mathbf{1}$ denotes a filler circuit of size at most N_{circ}, with the same number of inputs as C, and where all gates always output 1.
- Call $\widetilde{C}_{\text{proof}} = \mathsf{PadSingle}(\mathbf{1}, c \cdot (N_{\text{proof}} + N_{\text{circ}}))$ to create a placeholder circuit of a fixed topology, where the notation $\mathbf{1}$ denotes a filler circuit with at most $c \cdot (N_{\text{proof}} + N_{\text{circ}})$ gates, with $2N_{\text{circ}}$ number of input wires, and where all gates always output 1. Recall that c is a universal constant related to Lemma 2.2 and Lemma 2.3 — we can take c to be the products of the constants in Lemma 2.2 and Lemma 2.3.

  The input wires of $\widetilde{C}_{\text{proof}}$ contain 1) all input wires of $\widetilde{C}_1$ and $\widetilde{C}_2$ and 2) all output wires of regular gates in $\widetilde{C}_1$ and $\widetilde{C}_2$. The outputs of $\widetilde{C}_{\text{proof}}$ are defined as the outputs of the last $N_{\text{proof}} + N_{\text{circ}}$ regular gates added to $\widetilde{C}_{\text{proof}}$. At this moment, all these regular gates constantly output 1, and thus all outputs of $\widetilde{C}_{\text{proof}}$ must be 1.
- Then, we compute the AND (denoted $\mathsf{out}_{\text{proof}}$ in Figure 2) of all output wires of $\widetilde{C}_{\text{proof}}$. This can be accomplished by building a binary tree in which each node computes a pairwise AND.

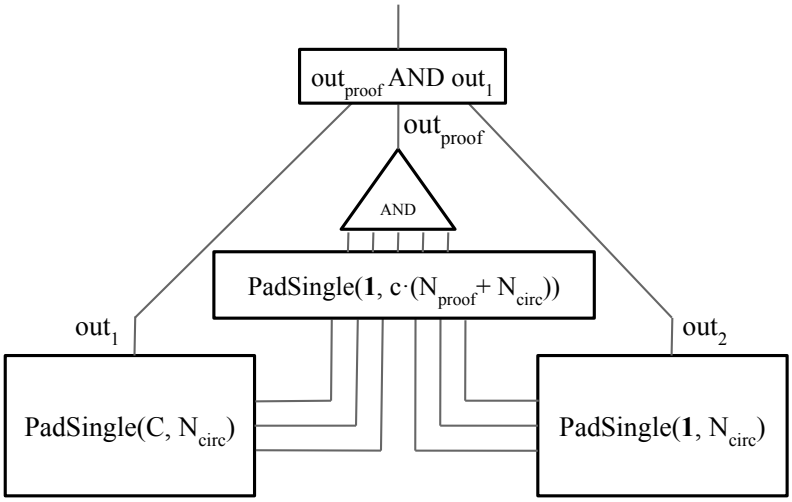


Figure 2: $\mathsf{Pad}(C, N_{\text{circ}}, N_{\text{proof}})$ **algorithm.** The notation $\mathbf{1}$ denotes a filler circuit of appropriate size in which all gates always output 1. The AND-tree is a binary tree of AND gates.

- Let out_1 and out_2 denote the output bits of $\widetilde{C}_1$ and $\widetilde{C}_2$ respectively, and let $\mathsf{out}_{\text{proof}}$ denote the output wire of the above step. The padded circuit has one final gate that takes out_1, out_2, and $\mathsf{out}_{\text{proof}}$ as inputs, and outputs ($\mathsf{out}_{\text{proof}}$ AND out_1). As mentioned earlier, since the outputs of $\widetilde{C}_{\text{proof}}$ must be 1, the padded circuit must output out_1 at this moment.

3.3 Proof of Transitive $O(\log(N_{\text{circ}} + N_{\text{proof}}))$-Equivalence

We want to show that $\mathsf{Pad}(C_1, N_{\text{circ}}, N_{\text{proof}})$ and $\mathsf{Pad}(C_2, N_{\text{circ}}, N_{\text{proof}})$ are transitively $O(\log(N_{\text{circ}} + N_{\text{proof}}))$-equivalent. This means that we need to construct a sequence of intermediate hybrid circuits that are all functionally equivalent and have the same topology. Moreover, the number of gates with different configurations in any two adjacent hybrid circuits is bounded by $O(\log(N_{\text{circ}} + N_{\text{proof}}))$, and the subcircuit formed by these differing gates are functionally equivalent in the two adjacent hybrid circuits.

Table 1 shows the sequence of outer hybrids. We now explain the notation. We can represent the initial circuit $\mathsf{Pad}(C_1, N_{\text{circ}}, N_{\text{proof}})$ using a tuple

$$\mathsf{Pad}(C_1, N_{\text{circ}}, N_{\text{proof}}) := \left(\widetilde{C}_1(C_1), \widetilde{C}_2(\mathbf{1}), \widetilde{C}_{\text{proof}}(\mathbf{1}), (\mathsf{out}_{\text{proof}} \text{ AND } \mathsf{out}_1)\right)$$

This means that the $\widetilde{C}_1$ part is obtained through $\mathsf{PadSingle}(C_1, N_{\text{circ}})$, the $\widetilde{C}_2$ part is obtained through $\mathsf{PadSingle}(\mathbf{1}, N_{\text{circ}})$, the $\widetilde{C}_{\text{proof}}$ part is obtained through $\mathsf{PadSingle}(\mathbf{1}, c \cdot N_{\text{proof}})$, and the final gate computes $\mathsf{out}_{\text{proof}}$ AND out_1. The AND-tree part of the circuit will be identical in all outer hybrids, so we omit it in the tuple. The notation for the rest of Table 1 is similarly defined, and the part that we modify in the next outer hybrid is highlighted in blue.

Table 1: Top-level sequence of hybrid circuits for transitively $O(\log n)$-equivalence proof.

$\mathsf{Pad}(C_1, N_{\text{circ}}, N_{\text{proof}})$:	$\widetilde{C}_1(C_1)$,	$\widetilde{C}_2(\mathbf{1})$,	$\widetilde{C}_{\text{proof}}(\mathbf{1})$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1)$
Grow C_2:			$\Downarrow$	
	$\widetilde{C}_1(C_1)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}(\mathbf{1})$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1)$
Grow proof for $C_1 = C_2$:			$\Downarrow$	
	$\widetilde{C}_1(C_1)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}([C_1 = C_2])$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1)$
Select second output:			$\Downarrow$	
	$\widetilde{C}_1(C_1)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}([C_1 = C_2])$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_2)$
Shrink proof:			$\Downarrow$	
	$\widetilde{C}_1(C_1)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}(\mathbf{1})$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_2)$
Replace C_1 with C_2:			$\Downarrow$	
	$\widetilde{C}_1(C_2)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}(\mathbf{1})$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_2)$
Grow proof for $C_2 = C_2$			$\Downarrow$	
	$\widetilde{C}_1(C_2)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}([C_2 = C_2])$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_2)$
Select first output			$\Downarrow$	
	$\widetilde{C}_1(C_2)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}([C_2 = C_2])$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1)$
Shrink proof			$\Downarrow$	
	$\widetilde{C}_1(C_2)$,	$\widetilde{C}_2(C_2)$,	$\widetilde{C}_{\text{proof}}(\mathbf{1})$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1)$
Shrink C_2			$\Downarrow$	
$\mathsf{Pad}(C_2, N_{\text{circ}}, N_{\text{proof}})$	$\widetilde{C}_1(C_2)$,	$\widetilde{C}_2(\mathbf{1})$,	$\widetilde{C}_{\text{proof}}(\mathbf{1})$,	$(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1)$

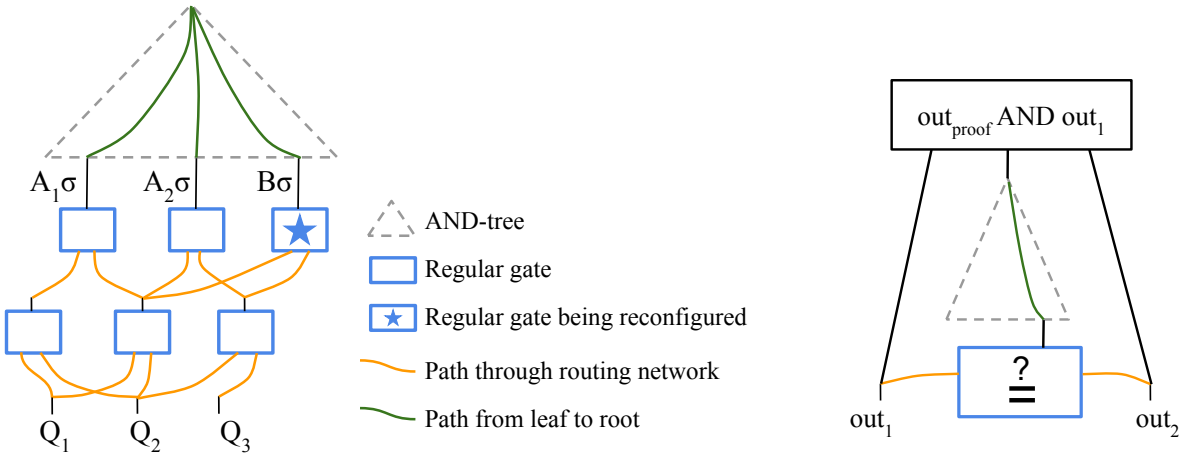
To complete the proof, we need a sequence of inner hybrids to switch from each outer hybrid to the next one, as described below.

Grow C_2. We want to change from $(\widetilde{C}_1(C_1), \widetilde{C}_2(\mathbf{1}), \widetilde{C}_{\text{proof}}(\mathbf{1}), (\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1))$ to $(\widetilde{C}_1(C_1)$, $\widetilde{C}_2(C_2)$, $\widetilde{C}_{\text{proof}}(\mathbf{1})$, $(\mathsf{out}_{\text{proof}}$ AND $\mathsf{out}_1))$. To do so, we need to reconfigure the following gates based on C_2 — we will first describe the goal, and the order in which we make the reconfiguration (i.e., the sequence of inner hybrids) will be described after that:

- configure all gates in the routing networks in $\widetilde{C}_2$ to encode the right routing decision;
- configure all gates in the input selector gadgets for all regular gates corresponding in $\widetilde{C}_2$ to select the right input wire; and
- configure all regular gates in $\widetilde{C}_2$ to compute the correct function.

We will accomplish the above reconfiguration through a sequence of inner hybrids as described below. Without loss of generality, we may assume the following:

- In $\mathsf{PadSingle}(\mathbf{1}, N_{\text{circ}})$, and $\mathsf{PadSingle}(\mathbf{1}, c \cdot N_{\text{proof}})$, all gates, including the ones in the routing networks and the input-selector gadgets of the regular gates, always output 1 on all output wires;
- N_{circ} is a power of 2, such that all wires output from routing networks in $\widetilde{C}_2$ are consumed by some regular gate in $\widetilde{C}_2$. Recall also that the output wires of all regular gates in $\widetilde{C}_2$ are fed into the $\widetilde{C}_{\text{proof}}$ part as input. This implies that when N_{circ} is a power of 2, all output wires of any gate in $\widetilde{C}_2$ must be consumed by some children gate, i.e., there is no loose output wire in $\widetilde{C}_2$.



(a) **Reconfiguring an output gate in $\widetilde{C}_{\text{proof}}$.** The output gate being configured is obtained from $A_1\sigma, A_2\sigma \vdash B\sigma$ where σ is the substitution $P_1 \mapsto Q_1, P_2 \mapsto Q_2, P_3 \mapsto Q_3$. The differing subcircuit consists of all gates on the depicted paths.

(b) **Select out_2 instead of out_1.** The differing subcircuit consists of all gates on the depicted paths and the final gate.

Figure 3: **The differing subcircuit in the "grow proof" and "select second output" steps.**

Specifically, we will sort all gates in $\widetilde{C}_2$ in a topological order. For each gate in a topological order, we will switch it to the right configuration. The number of inner-hybrids is equal to the number of gates in $\widetilde{C}_2$. The differing subcircuit S between any adjacent pair of inner-hybrids contains the gate G^* currently being reconfigured, and any immediate children gates of G^* (i.e., any gate which has an input wire connected to an output wire of G^*). It is easy to see that the differing subcircuit S is $O(1)$ in size. We argue that S is functionally equivalent between the adjacent pair of inner-hybrids. Specifically, the children nodes of G^* can only be

1. a gate inside $\widetilde{C}_2$ — in this case the gate must constantly output 1 since we are making the switch in topological order;
2. a filler gate that constantly outputs 1 in the $\widetilde{C}_{\text{proof}}$ part; or
3. the final ($\mathsf{out}_{\text{proof}}$ AND out_1) gate.

Therefore, each output wire of the subcircuit S either outputs 1 constantly, or its value is ($\mathsf{out}_{\text{proof}}$ AND out_1) which is not dependent on any output wire from the $\widetilde{C}_2$ part.

Grow proof for $C_1 = C_2$. Next, we want to grow the proof of equivalence denoted $\widetilde{C}_{\text{proof}}([C_1 = C_2])$ in the $\widetilde{C}_{\text{proof}}$ part of the circuit. We use the notation $\widetilde{C}_{\text{proof}}([C_1 = C_2])$ to denote the outcome of $\mathsf{PadSingle}(C_{\text{proof}}, c \cdot (N_{\text{proof}} + N_{\text{circ}}))$ where C_{proof} is the circuit corresponding to Lemma 2.2 that encodes the $\mathcal{EF}$ proof of C_1 and C_2's equivalence. We accomplish this through a sequence of inner hybrids.

- Let C'_{proof} be the same as C_{proof}, except that output gates constantly output 1. First, we will change the $\widetilde{C}_{\text{proof}}$ part to encode $\mathsf{PadSingle}(C'_{\text{proof}}, c \cdot (N_{\text{proof}} + N_{\text{circ}}))$. Without loss of generality, we may assume that initially, in $\widetilde{C}_{\text{proof}}(\mathbf{1}) := \mathsf{PadSingle}(\mathbf{1}, c \cdot (N_{\text{proof}} + N_{\text{circ}}))$, all gates constantly output 1.

To make this switch, we can topologically sort all gates except output gates in $\widetilde{C}_{\text{proof}}$, and in the topological order, we reconfigure each gate to the correct configuration. This would incur as many inner hybrids as the number of gates reconfigured. Using the same argument as the "Grow C_2" part, any pair of adjacent hybrids have a differing subcircuit of size $O(1)$ which is functionally equivalent across the two hybrids, since the all children gates to the gate currently being reconfigured constantly output 1.

- At this moment, all output gates in $\widetilde{C}_{\text{proof}}$ constantly output 1, since each output gate in $\widetilde{C}_{\text{proof}}$ corresponds to an output gate in C'_{proof}. Next, we want to switch them one by one to the correct output. Recall that each output wire of C_{proof} encodes the conclusion part (i.e., $B\sigma$) of some proof line of the form $A_1\sigma, \ldots, A_k\sigma \vdash B\sigma$ (or simply $\vdash B\sigma$). Now, in the order that the proof lines show up in the $\mathcal{EF}$ proof, we want to switch the corresponding output gate in $\widetilde{C}_{\text{proof}}$ to the correct configuration (rather than constantly outputting 1). Here, the number of inner hybrids is $N_{\text{proof}} + N_{\text{circ}}$.

  It suffices to argue that between any pair of adjacent hybrids, there is a small differing subcircuit (henceforth denoted S) of size $O(\log N_{\text{proof}} + \log N_{\text{circ}})$ which preserves functionality across the two hybrids. We depict this differing subcircuit S in Figure 3a, which is constructed as follows. Suppose we are reconfiguring an output gate henceforth denoted G^* whose output wire should encode $B\sigma$ that is the conclusion of some proof line $A_1\sigma, \ldots, A_k\sigma \vdash B\sigma$, and σ is the substitution $P_1 \mapsto Q_1, \ldots, P_\ell \mapsto Q_\ell$. Due to Lemma 2.2, if we consider the union circuit $C_1 \cup C_2 \cup C_{\text{proof}}$, there is a constant-size subcircuit whose inputs correspond to $Q_1, \ldots, Q_\ell$ and whose outputs encode $A_1\sigma, \ldots, A_k\sigma, B\sigma$. For each gate in $C_1 \cup C_2 \cup C_{\text{proof}}$, there is a corresponding regular gate in the padded counterpart $\widetilde{C}_1 \cup \widetilde{C}_2 \cup \widetilde{C}_{\text{proof}}$ (depicted as the blue gates in Figure 3a). However, in the padded version $\widetilde{C}_1 \cup \widetilde{C}_2 \cup \widetilde{C}_{\text{proof}}$, a regular gate's output wires are not directly fed into the next regular gate as input. Instead, each regular gate's output wire needs to traverse 1) a copy gadget to make logarithmically many copies of the wire to feed into different routing networks; 2) a routing network; and 3) an input selector gadget for selecting a correct signal for the next regular gate. Therefore, in the padded $\widetilde{C}_1 \cup \widetilde{C}_2 \cup \widetilde{C}_{\text{proof}}$, the wiring between the blue regular gates in Figure 3a and $Q_1, \ldots, Q_\ell$ is now replaced with a path which traverses a copy gadget, a routing network, and an input selector (depicted as the orange paths in Figure 3a). Further, the output wires encoding $A_1\sigma, \ldots, A_k\sigma, B\sigma$ will travel through some path in the AND-tree (depicted as the green paths in Figure 3a). The differing subcircuit S is defined as all gates the orange and green paths traverse in Figure 3a. It is not hard to see that S has size at most $O(\log N_{\text{proof}} + \log N_{\text{circ}})$.

  We now argue that this differing subcircuit S is functionally equivalent across the two adjacent hybrids when we change the output of G^* from constantly 1 to the correct output. Note that earlier, we must have reconfigured the output gates that encode $A_1\sigma, \ldots, A_k\sigma$, and these values are already part of the AND-tree. Since the $\mathcal{EF}$ proof line $A_1\sigma, \ldots, A_k\sigma, B\sigma$ holds, when we change G^*'s output from 1 to the correct value $B\sigma$, it does not change the outcome of the AND-tree. Finally, S's output wires also include the *other* output wire of any gate that belongs to the copy gadget or the routing network that is not consumed by gates of the subcircuit S. It is easy to see that the values on these output wires are also functionally equivalent across the two adjacent hybrids. Note that the blue regular gates have a single fan-out only due to Lemma 2.1 and Lemma 2.2.

Select second output. Next, we want to change the final gate from ($\mathsf{out}_{\text{proof}}$ AND out_1) to ($\mathsf{out}_{\text{proof}}$ AND out_2). The differing subcircuit S is depicted in Figure 3b. The blue gate denotes

the regular gate that computes whether $\mathsf{out}_1 = \mathsf{out}_2$. Its output must be the conclusion of the final proof line, and thus it is fed into the AND tree. Just like Figure 3a, an orange path is a path through a routing network and input selector, and a green path is a path from leaf to root in the AND-tree. The differing subcircuit S consists of all gates that the orange and green paths travel through as well as the final gate ($\mathsf{out}_{\text{proof}}$ AND out_1) or ($\mathsf{out}_{\text{proof}}$ AND out_2). It is not hard to see that S is functionally equivalement before and after the switch, and moreover the size S is at most $O(\log N_{\text{proof}} + \log N_{\text{circ}})$.

Shrink proof. The proof is symmetric to the "grow proof" step, and we switch the relevant gates back to constantly outputting 1 in the reverse order of the "grow proof" step.

Replace C_1 with C_2. We can first switch gates in $\widetilde{C}_1$ to always outputting 1. This can be done with a sequence of inner hybrids in the reverse order as in the "grow C_2" step. Next, we can grow C_1 in the $\widetilde{C}_1$ part, and the proof is the same as the "grow C_2" step.

Grow proof for $C_2 = C_2$. At this moment, both the $\widetilde{C}_1$ and $\widetilde{C}_2$ parts encode the circuit C_2. Next, we want to grow a proof of equivalence for $C_2 = C_2$ in the $\widetilde{C}_{\text{proof}}$. Based on Lemma 2.3, such a proof of equivalence exists whose size is $O(N_{\text{circ}})$. This step can be accomplished in the same way as the "grow proof for $C_1 = C_2$" step.

Select first output. Symmetric to "select second output" step.

Shrink proof. The argument is the same as the previous "shrink proof" step.

Shrink C_2. Next, we want to shrink the C_2 in the $\widetilde{C}_2$ part. The argument is symmetric to "grow C_2" and we switch the gates in $\widetilde{C}_2$ to constantly outputting 1 in the reverse order of the "grow C_2" step.

In summary, we conclude:

Lemma 3.1. *The algorithm* Pad *satisfies:*

- *For any circuit C of size at most N_{circ}, $\mathsf{Pad}(C, N_{\text{circ}}, N_{\text{proof}})$ outputs an obfuscated circuit of size $\widetilde{O}(N_{\text{circ}} + N_{\text{proof}})$, and moreover, C and $\mathsf{Pad}(C, N_{\text{circ}}, N_{\text{proof}})$ are functionally equivalent.*
- *For any circuits C_1, C_2 with an $\mathcal{EF}$ equivalence proof, $\mathsf{Pad}(C_1, N_{\text{circ}}, N_{\text{proof}})$ and $\mathsf{Pad}(C_2, N_{\text{circ}}, N_{\text{proof}})$ are transitively $O(\log(N_{\text{circ}} + N_{\text{proof}}))$-equivalent.*

4 Somewhere Extractable Hash with Prefix Consistency Proof

We define a notion of somewhere extractable hash (SEH) with prefix consistency proof. Suppose two inputs $x_1, x_2 \in \{0,1\}^n$ share a common prefix of length j and they hash to y_1 and y_2 respectively. Then, we can produce an efficient proof π that attests to the fact that the preimages of y_1 and y_2 share a common length-j prefix. The hash function is somewhere extractable in the following sense. There is a trapdoor setup henceforth denoted TGen which additionallly takes in a set of indices denoted $E \subseteq \{0, 1, \ldots, n-1\}$. This trapdoor setup will only be used in the proofs but not in the actual cryptographic construction, and using the resulting hash key will cause the hash values to be statistically binding at the index set E. Further, TGen and the normal setup Gen (which is used in the actual cryptographic construction) output hash keys that are computationally indistinguishable.

Our notion of SEH differs from that of Jain and Jin [JJ22]. In particular, in their construction, the consistency proof can be for an arbitrary subset of indices $E' \subseteq [n]$. However, we observe that getting indistinguishability obfuscation for circuits with equivalence proofs requires only consistency proofs on a prefix. By constraining the consistency proofs to prefixes, we can greatly simplify the SEH construction and achieve better asymptotical efficiency. Notably, in Jain and Jin's SEH construction [JJ22], to compute a consistency proof, one would need to compute a batch argument (BARG) to prove that for all $i \in [n]$ some $\widetilde{O}(1)$-sized circuit $C(i, \cdot)$ is satisfiable. The proving time is $T_{\mathrm{barg}}(n, \widetilde{O}(1))$ where $T_{\mathrm{barg}}(n_{\mathrm{idx}}, n_{\mathrm{circ}})$ is the cost of computing a BARG for a circuit of size n_{circ} and an index set of size n_{idx}. Furthermore, in Jain and Jin's iO construction, the evaluator needs to compute $O(1)$ BARGs per gate in the padded circuit of size $N_{\mathrm{padded}} = \widetilde{O}((N_{\mathrm{circ}} + N_{\mathrm{proof}})^2)$. Thus, their evaluation time is at least $N_{\mathrm{padded}} \cdot T_{\mathrm{barg}}(N_{\mathrm{padded}}, \widetilde{O}(1))$ which is at least N_{circ}^4 (even when aggressively assuming a linear-time BARG).

By contrast, in our SEH construction, by exploiting the Merkle-tree-like structure of the underlying SEH hash, producing a prefix consistency proof requires only $\widetilde{O}_\lambda(1)$ time. We will also explain later in Section 5.4 that given the Merkle-tree-like structure of our SEH construction, our iO evaluator only needs to perform $\widetilde{O}(N_{\mathrm{circ}})$ work to compute all hashes needed during the iO evaluation.

4.1 Definition

Henceforth, given $x \in \{0,1\}^n$, we use $x[:j]$ to denote the length-j prefix of x. A somewhere extractable hash with *prefix* consistency proof (SEH with consistency proof) consists of PPT algorithms (Gen, TGen, Hash, Open, Verify, Ext, $\mathsf{ConsisP}, \mathsf{ConsisV}$):

- $\mathsf{Gen}(1^\lambda, 1^n, 1^{|E|})$: On input a security parameter λ, the length of the message n, and the size of a subset $E \subseteq \{0, 1, \ldots, n-1\}$, the "normal mode" key generation algorithm outputs a random hash key hk.
- $\mathsf{TGen}(1^\lambda, 1^n, E)$: On input a security parameter λ, the length of the message n and an extraction subset $E \subseteq \{0, 1, \ldots, n-1\}$, the "trapdoor mode" key generation algorithm outputs a random hash key hk and a trapdoor td.
- $\mathsf{Hash}(\mathsf{hk}, x \in \{0,1\}^n)$: On input a hash key hk and a vector $x \in \{0,1\}^n$, the algorithm outputs a hash value y.
- $\mathsf{Open}(\mathsf{hk}, x, j)$: On input a hash key hk, vector x and an index $j \in \{0, 1, \ldots, n-1\}$, the algorithm outputs an opening proof ρ.
- $\mathsf{Verify}(\mathsf{hk}, y, j, u, \rho)$: Given a hash key hk and hash value y; an integer index $j \in \{0, 1, \ldots, n-1\}$, a purported value $u \in \{0, 1\}$ and an opening proof ρ, the verification algorithm decides to accept (output 1) or reject (output 0) the opening.
- $\mathsf{Ext}(\mathsf{td}, y, i)$: Given hash value y, trapdoor td, and an index $i \in E$, the algorithm outputs an extracted bit at position i.
- $\mathsf{ConsisP}(\mathsf{hk}, x_1, x_2, j)$: Given hash key hk, two vectors $x_1, x_2 \in \{0,1\}^n$, and an index $j \in \{0, 1, \ldots, n-1\}$, the algorithm outputs a consistency proof π, proving that $x_1[:j] = x_2[:j]$.
- $\mathsf{ConsisV}(\mathsf{hk}, y_1, y_2, j, \pi)$: Given the hash key hk, two hash values y_1, y_2, an index $j \in \{0, 1, \ldots, n-1\}$ and a consistency proof π, the algorithm decides to accept or reject.

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

- **Succinctness.** The size of hash key, hash value, opening proof, and consistency proof are bounded by $\mathsf{poly}(\lambda, |E|, \log n)$.
- **Key Indistinguishability.** For any stateful PPT adversary $\mathcal{A}$ and any polynomial $n = n(\lambda)$, define the advantage of $\mathcal{A}$ in distinguishing Gen and TGen as:

$$\begin{aligned}&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{SEH}}(\lambda)\\ =&\Big|\Pr[E \leftarrow \mathcal{A}(1^\lambda, 1^n), \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{|E|}) : \mathcal{A}(\mathsf{hk}) = 1]\\ &- \Pr[E \leftarrow \mathcal{A}(1^\lambda, 1^n), \mathsf{hk} \leftarrow \mathsf{TGen}(1^\lambda, 1^n, E) : \mathcal{A}(\mathsf{hk}) = 1]\Big|.\end{aligned}$$

  Then we say SEH is polynomially secure iff for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{SEH}}(\lambda) \le \mathsf{negl}(\lambda)$, or 2^{λ^ϵ} sub-exponentially secure iff for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{SEH}}(\lambda) \le 2^{-\lambda^\epsilon}\mathsf{negl}(\lambda)$ for some $\epsilon > 0$.
- **Opening Correctness.** For any λ, n, $E \subseteq \{0, \ldots, n-1\}$, for any hash key hk in the output range of either $\mathsf{Gen}(1^\lambda, 1^n, 1^{|E|})$ or $\mathsf{TGen}(1^\lambda, 1^n, E)$, any vector $x \in \{0,1\}^n$, and any index $j \in \{0, 1, \ldots, n-1\}$, we have

$$\Pr[y \leftarrow \mathsf{Hash}(\mathsf{hk}, x), \rho \leftarrow \mathsf{Open}(\mathsf{hk}, x, j) : \mathsf{Verify}(\mathsf{hk}, y, j, x_j, \rho) = 1] = 1.$$

- **Extraction Correctness.** For any λ, n, $E \subseteq \{0, 1, \ldots, n-1\}$, for any $(\mathsf{hk}, \mathsf{td})$ in the output range of $\mathsf{TGen}(1^\lambda, 1^n, E)$, any $i \in E$, any hash value y, any $u \in \{0,1\}$, and any opening proof ρ, we have

$$\Pr[\mathsf{Verify}(\mathsf{hk}, y, i, u, \rho) = 1 \Rightarrow \mathsf{Ext}(\mathsf{td}, y, i) = u] = 1.$$

- **Consistency Proof Completeness.** For any λ, n, $E \subseteq \{0, 1, \ldots, n-1\}$, for any $j \in \{0, \ldots, n-1\}$, any $x_1 \in \{0,1\}^n$, $x_2 \in \{0,1\}^n$ such that $x_1[:j] = x_2[:j]$, then for any hk in the output range of either $\mathsf{Gen}(1^\lambda, 1^n, 1^{|E|})$ or $\mathsf{TGen}(1^\lambda, 1^n, E)$, we have

$$\begin{aligned}&\Pr[y_1 \leftarrow \mathsf{Hash}(\mathsf{hk}, x_1), y_2 \leftarrow \mathsf{Hash}(\mathsf{hk}, x_2), \pi \leftarrow \mathsf{ConsisP}(\mathsf{hk}, x_1, x_2, j) :\\ &\quad \mathsf{ConsisV}(\mathsf{hk}, y_1, y_2, j, \pi) = 1] = 1.\end{aligned}$$

- **Somewhere Statistical Soundness.** For any λ, n, any $E \subseteq \{0, 1, \ldots, n-1\}$, for any hash key $(\mathsf{hk}, \mathsf{td})$ in the output range of $\mathsf{TGen}(1^\lambda, 1^n, E)$, for any $0 \le j < n$, for any y_1, y_2,

$$\exists i \in E \cap \{0, \ldots, j\} : \mathsf{Ext}(\mathsf{td}, y_1, i) \neq \mathsf{Ext}(\mathsf{td}, y_2, i) \Longrightarrow \forall \pi, \mathsf{ConsisV}(\mathsf{hk}, y_1, y_2, j, \pi) = 0$$

4.2 Construction

Our construction, inspired by Hubacek and Wichs [HW15], combines a Merkle-tree-like structure and fully homomorphic encryption (FHE). It suffices to consider the case that $|E| = 1$, i.e. the extraction set is a singleton $\{i\}$. Based on such a scheme, we can use parallel repetition to support larger sets E [HW15].

Let $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a polynomially secure (or 2^{λ^ϵ} sub-exponentially secure) FHE scheme. We construct our SEH with consistency proof as follows:

- $\mathsf{Gen}(1^\lambda, 1^n, 1^{|E|=1})$: Simply run $\mathsf{TGen}(1^\lambda, 1^n, \{0\})$ and output hk, where TGen is defined below.
- $\mathsf{TGen}(1^\lambda, 1^n, \{i\})$: Without loss of generality assume $n = 2^\alpha$ for some integer α. For $j = 0, \ldots, \alpha$: create $\mathsf{pk}_j, \mathsf{sk}_j \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$. Let $b_\alpha b_{\alpha-1} \cdots b_1$ be the binary representation of i. For $j = 1, \ldots, \alpha$, compute $c_j \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_j}((\mathsf{sk}_{j-1}, b_j))$. Let $\mathsf{hk} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_\alpha, c_1, \ldots, c_\alpha)$, and $\mathsf{td} = \mathsf{sk}_\alpha$.

- $\mathsf{Hash}(\mathsf{hk}, x)$: Let $x = (x_0, \ldots, x_{n-1})$. Let T be a binary tree of height α. We think of the leaves as being at level 0 and the root of the tree as being at level α. We inductively and deterministically associate a ciphertext ct_v with each vertex $v \in T$:
  - If v is the j-th ($0 \le j < n$) leaf, then let $\mathsf{ct}_v = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(x_j; 0)$ (the FHE encryption of x_j using fixed randomness 0).
  - Let $v \in T$ be a non-leaf vertex at level $j \in [\alpha]$ with children v_0, v_1 having associated ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$, and let c_j be the ciphertext contained in hk for level j. Let $\mathsf{ct}_v = \mathsf{Eval}_{\mathsf{pk}_j}(F_{\mathsf{ct}_0,\mathsf{ct}_1}, c_j)$ where $F_{\mathsf{ct}_0,\mathsf{ct}_1}(\cdot,\cdot)$ is the following function:

$$F_{\mathsf{ct}_0,\mathsf{ct}_1}(\mathsf{sk}, b) : \{\text{Compute } x_0 = \mathsf{FHE.Dec}_{\mathsf{sk}}(\mathsf{ct}_0), x_1 = \mathsf{FHE.Dec}_{\mathsf{sk}}(\mathsf{ct}_1). \text{ Output } x_b.\}$$

- $\mathsf{Open}(\mathsf{hk}, x, j)$: Perform the computation of $\mathsf{Hash}(\mathsf{hk}, x)$ as described above and output the ciphertexts ct_v for each vertex v that is a sibling of some vertex along the path from the root to the leaf at position j.
- $\mathsf{Verify}(\mathsf{hk}, y, j, u, \rho)$: Compute the ciphertext associated with the root, using u as well as ρ which contains the ciphertexts along the path from leaf j to the root. Specifically, for each vertex v along the path from leaf j to the root, inductively compute a ciphertext ct_v. In the base case, when v is the j-th leaf, set $\mathsf{ct}_v = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(u; 0)$. Otherwise, if v is not a leaf, one of its children lies on the path from leaf j to the root in which case the corresponding ciphertext was computed in the previous step, and the sibling ciphertext is provided in the opening π. Therefore, we can compute ct_v using FHE evaluation just like in the Hash algorithm. Finally, compute the ciphertext ct_v associated with the root of the tree and check if $y = \mathsf{ct}_v$.
- $\mathsf{Ext}(\mathsf{td}, y, i)$: Parse $\mathsf{td} := \mathsf{sk}_\alpha$ and output the i-th bit of the decrypted message $\mathsf{FHE.Dec}_{\mathsf{sk}_\alpha}(y)$.
- $\mathsf{ConsisP}(\mathsf{hk}, x_1, x_2, j)$: Perform the computation of $\mathsf{Hash}(\mathsf{hk}, x_1)$ and $\mathsf{Hash}(\mathsf{hk}, x_2)$. For $b \in \{1, 2\}$, for any vertex $v \in T$, let $\mathsf{ct}_{v,b}$ be the ciphertext on v in the computation of $\mathsf{Hash}(\mathsf{hk}, x_b)$. In the tree T, the leaves indexed by $\{0, 1, \ldots, j-1\}$ can be covered by $L = O(\log n)$ subtrees — specifically, if the binary representation of j has the k-th bit set (where the least significant bit is the 0-th bit), then there is a subtree of size 2^k.

  Let vertices $u_1, \ldots, u_L$ be the roots of these subtrees. Output the proof π consisting of the following terms: for each $b \in \{1, 2\}$,
  - The ciphertexts $\mathsf{ct}_{u_k,b}$ for all $k \in [L]$.
  - The ciphertexts $\mathsf{ct}_{v,b}$ for each vertex v that is a sibling of some vertex along the path from the root to some u_k where $k \in [L]$.

  Note that π contains $O(\log n)$ ciphertexts.
- $\mathsf{ConsisV}(\mathsf{hk}, y_1, y_2, j, \pi)$: According to j, find the vertices $u_1, \ldots, u_L$ where $L = O(\log n)$ in tree T in the same way as $\mathsf{ConsisP}$. Then for $b \in \{1, 2\}$, in a similar way as Verify, compute the ciphertext associated with the root using the ciphertexts $\mathsf{ct}_{*,b}$ contained in π where $*$ stands for wildcard, and let the result be y'_b. $\mathsf{ConsisV}$ will check the following and output the result:
  - $y_1 = y'_1$ and $y_2 = y'_2$.
  - For each $k \in [L]$, $\mathsf{ct}_{u_k,1} = \mathsf{ct}_{u_k,2}$.

4.3 Security Proofs

We will show the above scheme satisfies the following properties:

- **Succinctness.** By the description of the algorithms, it is not hard to verify the succinctness holds.
- **Key Indistinguishability.** Since the hash key will not reveal sk_α, this property follows from the semantic security of the underlying FHE scheme. Formally, W.L.O.G. assume $|E|$ and $\log n$ are bounded by some polynomial of λ, since the secret key of SEH only consists of $O(|E| \log n)$ FHE ciphertexts, we know that the SEH scheme is polynomially (or 2^{λ^ϵ} sub-exponentially resp.) secure if the underlying FHE scheme satisfies the same level of security.
- **Opening Correctness.** Since Open outputs an opening proof ρ that is part of an honest execution of the Hash algorithm, the opening correctness holds.
- **Extraction Correctness.** Suppose hk is generated in the trapdoor mode on index i and the verification passes, i.e., $\mathsf{Verify}(\mathsf{hk}, y, i, u, \rho) = 1$. Let $\{v_0, v_1, \ldots, v_\alpha\}$ denote the path from the i-th leaf to the root of T. Let $\mathsf{ct}_{v_0} = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(u; 0), \mathsf{ct}_{v_1}, \ldots, \mathsf{ct}_{v_\alpha}$ denote the ciphertexts associated with these vertices during the Verify algorithm, and let $\mathsf{ct}'_{v_0}, \ldots, \mathsf{ct}'_{v_{\alpha-1}}$ be the corresponding sibling ciphertexts.
  - First, we have $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}_{v_0}) = u$ by definition.
  - Suppose $\mathsf{FHE.Dec}_{\mathsf{sk}_{j-1}}(\mathsf{ct}_{v_{j-1}}) = u$ and $\mathsf{FHE.Dec}_{\mathsf{sk}_{j-1}}(\mathsf{ct}'_{v_{j-1}}) = w$ — note that the decrypted value w is well-defined even if $\mathsf{ct}'_{v_{j-1}}$ is not an honestly generated ciphertext. Thus, the circuit $F_{\mathsf{ct}_{v_{j-1}}, \mathsf{ct}'_{v_{j-1}}}(\mathsf{sk}_{j-1}, b_j)$ chooses the b_j-th element of pair (u, w) which is exactly u. Therefore, by the correctness of FHE, we also have $\mathsf{FHE.Dec}_{\mathsf{sk}_j}(\mathsf{ct}_{v_j}) = u$.

  Since $y = \mathsf{ct}_{v_\alpha}$, we have $\mathsf{FHE.Dec}_{\mathsf{sk}_\alpha}(\mathsf{ct}_{v_\alpha}) = u$ which implies extraction correctness.
- **Consistency Proof Completeness.** Since $\mathsf{ConsisP}$ outputs a proof π that is part of an honest execution of the Hash algorithm, the consistency proof completeness holds.
- **Somewhere Statistical Soundness.** Suppose hk is generated by $\mathsf{TGen}(1^\lambda, 1^n, \{i\})$ and

$$\mathsf{ConsisV}(\mathsf{hk}, y_1, y_2, j, \pi) = 1$$

  for some $j > i$ and π. Let $u_1, \ldots, u_L$ where $L = O(\log n)$ be the roots of the subtrees that cover the leaves $0, 1, \ldots, j-1$ in T. Let u_k be the root of subtree that contains leaf i, and suppose it is in the β-th level of T. Using similar induction as the proof of **Extraction Correctness**, we can show that: $\mathsf{FHE.Dec}_{\mathsf{sk}_\alpha}(y_b) = \mathsf{FHE.Dec}_{\mathsf{sk}_\beta}(\mathsf{ct}_{u_k,b})$ for $b \in \{1, 2\}$. Since the consistency verification passes, $\mathsf{ct}_{u_k,1} = \mathsf{ct}_{u_k,2}$ holds, we have $\mathsf{FHE.Dec}_{\mathsf{sk}_\alpha}(y_1) = \mathsf{FHE.Dec}_{\mathsf{sk}_\alpha}(y_2)$. This implies that $\mathsf{Ext}(\mathsf{td}, y_1, i) = \mathsf{Ext}(\mathsf{td}, y_2, i)$.

5 Quasi-Linear Locally Indistinguishable Obfuscator for Circuits

Through this section, we use n to denote the maximum size of the circuit we want to obfuscate. We next construct a locally indistinguishable obfuscator henceforth denoted LiO for circuits that provides indistinguishability when obfuscating two $O(\log n)$-equivalent circuits. By a hybrid argument, the same obfuscator also provides indistinguishability when obfuscating two transitively $O(\log n)$-equivalent circuits.

5.1 Building Blocks and Parameters

We need following building blocks in our construction where $s = O(\log n)$ is the maximum size of the subcircuit that differ in appearance:

- Polynomially secure fully hormomorphic encryption $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with output length $L_{\mathsf{FHE}} = \mathsf{poly}(\lambda)$ per bit.
- $2^{\lambda^{\epsilon_{\mathsf{PRG}}}}$-secure pseudorandom generator $\mathsf{PRG} : \{0,1\}^{\lambda_{\mathsf{PRG}}} \to \{0,1\}^{2\lambda_{\mathsf{PRG}}}$ with security parameter $\lambda_{\mathsf{PRG}} = (sL_{\mathsf{FHE}})^{1/\epsilon_{\mathsf{PRG}}} = \mathsf{poly}(\lambda, s)$.
- $2^{\lambda^{\epsilon_{\mathsf{PRF}}}}$-secure puncturable pseudorandom functions $\mathsf{PRF}_m : \{0,1\}^{\lambda_{\mathsf{PRF}}} \times \{0,1\}^{sL_{\mathsf{FHE}}} \to \{0,1\}$ with security parameter $\lambda_{\mathsf{PRF}} = (sL_{\mathsf{FHE}})^{1/\epsilon_{\mathsf{PRF}}} = \mathsf{poly}(\lambda, s)$.
- $2^{\lambda^{\epsilon_{\mathsf{PRF}}}}$-secure puncturable pseudorandom functions $\mathsf{PRF}_\sigma : \{0,1\}^{\lambda_{\mathsf{PRF}}} \times \{0,1\}^{sL_{\mathsf{FHE}}+1} \to \{0,1\}^{\lambda_{\mathsf{PRG}}}$.
- Polynomially secure somewhere extractable hash with prefix consistency proof $\mathsf{SEH} = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext}, \mathsf{ConsisP}, \mathsf{ConsisV})$.
- $2^{\lambda^{\epsilon_{\mathsf{obf}}}}$-secure indistinguishability obfuscator denoted Obf with security parameter $\lambda_{\mathsf{obf}} = (sL_{\mathsf{FHE}})^{1/\epsilon_{\mathsf{obf}}} = \mathsf{poly}(\lambda, s)$.

5.2 Construction

Obfuscation algorithm. Our locally indistinguishability obfuscation for circuits denoted $\mathsf{LiO}(1^\lambda, s, C)$ below adopts a gate-by-gate obfuscation approach. On input circuit C, it will output an obfuscated circuit $\widetilde{\mathsf{Gate}}^g$ for each gate g. We formally present our LiO obfuscation algorithm and evaluation algorithm below:

$\mathsf{LiO}(1^\lambda, s, C)$

- Generate a somewhere extractable hash key $\mathsf{hk} \leftarrow \mathsf{SEH.Gen}(1^\lambda, 1^{|C|}, 1^s)$.
- For each wire $w \in \{0, 1, \ldots, |C| - 1\}$, generate the following:
  - Puncturable PRF key K_w^m of PRF_m for single-bit symmetric key encryption.
  - Puncturable PRF key K_w^σ of PRF_σ for MACs of the ciphertexts and shrunk hash values.
- Generate a FHE key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.
- Generate a fake trapdoor $\mathsf{td}' \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{|\mathsf{td}|})$.
- For each $j \in [s]$, generate a fake symmetric encryption key $K'_j \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{\lambda_{\mathsf{PRF}}})$.
- 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 obfuscate a circuit

$$\widetilde{\mathsf{Gate}}^g \leftarrow \mathsf{Obf}(1^\lambda, \mathsf{Gate}^g_{\mathsf{hk}, \{K_i^m, K_i^\sigma\}_{i \in \mathsf{inp}(g)}, \{K_o^m, K_o^\sigma\}_{o \in \mathsf{out}(g)}, \mathsf{pk}, \mathsf{td}', \{K'_j\}_{j \in [s]}}),$$

  where the circuit $\mathsf{Gate}^g = \mathsf{Gate}^g_{\mathsf{hk}, \{K_i^m, K_i^\sigma\}_{i \in \mathsf{inp}(g)}, \{K_o^m, K_o^\sigma\}_{o \in \mathsf{out}(g)}, \mathsf{pk}, \mathsf{td}', \{K'_j\}_{j \in [s]}}$ is defined below (henceforth also called the augmented circuit for gate g).

- Output the obfuscated circuit which is a tuple

$$\left(\{\widetilde{\mathsf{Gate}}^{g}\}_{g\in C}, \{K_w^m, K_w^\sigma\}_{w\in \mathsf{inp}(C)\cup \mathsf{out}(C)}, \mathsf{hk}, \mathsf{pk}, \mathsf{td}', \{K'_j\}_{j\in[s]}\right).$$

The augmented circuit Gate^g for the gate g is defined below:

$$\mathsf{Gate}_{\mathsf{hk},\{K_i^m,K_i^\sigma\}_{i\in\mathsf{inp}(g)},\{K_o^m,K_o^\sigma\}_{o\in\mathsf{out}(g)},\mathsf{pk},\mathsf{td}',\{K'_j\}_{j\in[s]}}(H, \{H_i, \mathsf{ct}_i, \sigma_i, \rho_i, \pi_i\}_{i\in\mathsf{inp}(g)})$$

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

$$h \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K'_j\}_{j\in[s]})),$$
$$h_i \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).$$

  Here, the length of $h, \{h_i\}_{i\in\mathsf{inp}(g)}$ is exactly sL_{FHE} since the circuit Shrink outputs s bits.
- 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 is consistent with (ct_i, h_i):

$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$

  - Proof of consistency for H and H_i with proof π_i:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$

  - MAC σ_i for (ct_i, h_i): $\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i)))$.
- Decrypt the symmetric key ciphertexts ct_i: let

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

- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$.
- Encrypt m_o for each $o \in \mathsf{out}(g)$ under K_o^m, and MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

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

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

The $\mathsf{Shrink}_H(\mathsf{td}, \{K_j\}_{j\in[s]})$ subroutine is defined below:

$$\mathsf{Shrink}_H(\mathsf{td}, \{K_j\}_{j\in[s]})$$

- Let E be the subset of indices that SEH can extract, it is guaranteed $|E| = s$. Extract the ciphertexts and hash values by SEH: $\{(\mathsf{ct}_w^*, h_w^*)\}_{w\in E} \leftarrow \mathsf{SEH.Ext}(\mathsf{td}, H)$.
- For each $w \in E$, suppose w is the j-th $(1 \le j \le s)$ element in E, decrypt the message $m_w^* \leftarrow \mathsf{ct}_w^* \oplus \mathsf{PRF}_m(K_j, h_w^*)$. If the extraction of SEH fails, simply let $m_w^* = 0$.
- Output $\{m_w^*\}_{w\in E}$.

Evaluation algorithm. We now describe how to evaluate the obfuscated circuit. Given an obfuscated circuit

$$\left(\{\widetilde{\mathsf{Gate}}^g\}_{g\in C}, \{K_w^m, K_w^\sigma\}_{w\in \mathsf{inp}(C)\cup \mathsf{out}(C)}, \mathsf{hk}, \mathsf{pk}, \mathsf{td}', \{K'_j\}_{j\in[s]}\right)$$

and an input $x \in \{0,1\}^{|\mathsf{inp}(C)|}$, we now compute $C(x)$ as follows (assuming the topology of C is public):

Evaluation of obfuscated circuit

Suppose that the $n := |C|$ wires are indexed according to topological order. For each wire $w \in \{0, 1, \ldots, n-1\}$:

- Define the string of all existing ciphertexts and shrink hash values as follows: let $s_w = (s_{w,0}, s_{w,1}, \ldots, s_{w,n-1})$, where for each wire $w' \in \{0, 1, \ldots, n-1\}$,

$$s_{w,w'} = \begin{cases} (\mathsf{ct}_{w'}, h_{w'}) & \text{for} \quad w' < w \\ (0, 0^{|h_{w'}|}) & \text{for} \quad w' \geq w \end{cases}.$$

  Here $|h_{w'}| = sL_{\mathsf{FHE}}$ is the length of the shrunk hash value.

- Prepare the hash value $H_w = \mathsf{SEH.Hash}(\mathsf{hk}, s_w)$.

- Shrink the hash value: let

$$h_w \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_w}, (\mathsf{td}', \{K'_j\}_{j\in[s]})),$$

  where Shrink is the circuit defined above.

- If w is an input wire of C, compute

$$\mathsf{ct}_w = x_w \oplus \mathsf{PRF}_m(K_w^m, h_w), \sigma_w = \mathsf{PRF}_\sigma(K_w^\sigma, (\mathsf{ct}_w, h_w)),$$

  where x_w denotes the value of wire w in input x.

- Otherwise, suppose w is the output wire of gate g.
  - Prepare the local openings for the input wires $\mathsf{inp}(g)$ of hash value H_w: for each $i \in \mathsf{inp}(g)$, let $\rho_i \leftarrow \mathsf{SEH.Open}(\mathsf{hk}, s_w, i)$.
  - Prepare the prefix consistency proofs for H_w and each of $\{H_i\}_{i\in\mathsf{inp}(g)}$: for each $i \in \mathsf{inp}(g)$, let $\pi_i \leftarrow \mathsf{SEH.ConsisP}(\mathsf{hk}, s_w, s_i, i)$.
  - Compute the gate g. Let

$$(\mathsf{ct}_w, \sigma_w) \leftarrow \widetilde{\mathsf{Gate}^g}(H_w, \{H_i, \mathsf{ct}_i, \sigma_i, \rho_i, \pi_i\}_{i\in\mathsf{inp}(g)}).$$

Finally, use the keys $\{K_o^m\}_{o\in\mathsf{out}(C)}$ to decrypt the output wires $\{\mathsf{ct}_o\}_{o\in\mathsf{out}(C)}$ and output the wire values $\{\mathsf{ct}_o \oplus \mathsf{PRF}_m(K_o^m, h_o)\}_{o\in\mathsf{out}(C)}$.

The correctness of our construction can be checked mechanically, so we omit the proof here. We defer the security proofs to Section 5.3.

5.3 Security Proofs

We will prove the following lemma:

Lemma 5.1 (Security of LiO)**.** *Suppose $s = O(\log n)$. The algorithm $\mathsf{LiO}(1^\lambda, s, \cdot)$ is an s-equivalent obfuscator: for any two size-n circuits $C, \widetilde{C}$ that are s-equivalent via the subcircuit S with $|S| = s$, and for any PPT adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathcal{A}(\mathsf{LiO}(1^\lambda, s, C)) = 1] - \Pr[\mathcal{A}(\mathsf{LiO}(1^\lambda, s, \widetilde{C})) = 1]\right| = \mathsf{negl}(\lambda).$$

By the definition of transitively s-equivalent circuits, we immediately obtain the following corollary from a standard hybrid argument:

Corollary 5.2. *Suppose $s = O(\log n)$. The algorithm $\mathsf{LiO}(1^\lambda, s, \cdot)$ is also an transitively s-equivalent obfuscator: for any two transitively s-equivalent circuits $C, \widetilde{C}$ and any PPT adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathcal{A}(\mathsf{LiO}(1^\lambda, s, C)) = 1] - \Pr[\mathcal{A}(\mathsf{LiO}(1^\lambda, s, \widetilde{C})) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof of Lemma 5.1. We prove the lemma via a series of hybrids. Let $C, \widetilde{C}$ be two s-equivalent circuits. This means that the two circuits are identical except for a subcircuit S of size at most s; and moreover the subcircuit S in both circuits has the same functionality.

Hybrid Real**.** This hybrid is the same as $\mathsf{LiO}(1^\lambda, s, C)$.

Hybrid Trap**.** This hybrid is almost the same as Real, except that we replace the SEH key hk to the trapdoor mode that is extractable for the indices $E = \mathsf{inp}(S)$ with trapdoor td. Then we replace the FHE ciphertexts td' and $\{K'_j\}_{j\in[s]}$ as

$$\begin{aligned}&\mathsf{td}' \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{td}),\\ &K'_j \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, K^m_{\mathsf{inp}_j(S)}), j \in [s],\end{aligned}$$

where $\mathsf{inp}_j(S)$ denotes the j-th input wire of S. By the key indistinguishability of SEH and the semantic security of FHE, Real and Trap are computationally indistinguishable.

Hybrid Hyb**.** This hybrid is almost the same as Trap, except that for each gate $g \in S$, instead of obfuscating the original Gate^g, we obfuscate the following circuit:

Modified Gate^g circuit

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

$$\begin{aligned}h &\leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K'_j\}_{j\in[s]})),\\ h_i &\leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).\end{aligned}$$

- 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 is consistent with (ct_i, h_i):

$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$

  - Proof of consistency for H and H_i with proof π_i:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$

– MAC σ_i for (ct_i, h_i): $\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i)))$.

- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w \in \mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

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

$$m_o = C_S^o(\{m_w^*\}_{w \in \mathsf{inp}(S)}),$$

where C_S^o is the function that directly calculates m_o from the inputs of S.

- Encrypt m_o for each $o \in \mathsf{out}(g)$ under K_o^m, and MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

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

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

Remark 5.3. *Recall that in the* Shrink *circuit, if the extraction for some m_j^* fails, we simply set it to 0. So we assume*

$$\{m_w^*\}_{w \in \mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h)$$

will always outputs a valid $m_w^* \in \{0, 1\}$ for each $w \in \mathsf{inp}(S)$.

To show the indistinguishability of Trap and Hyb, we will use a series of intermediate hybrids. Notice that Trap and Hyb only differ by the gates inside S, henceforth we may label all gates in S from 0 to $s - 1$ in topological order.

Hybrid Hyb_g. This hybrid is indexed by a gate g: for each $g' \leq g$, g' is obfuscated in the same way as Hyb; otherwise, g' is obfuscated in the same way as Trap. We can see that $\mathsf{Hyb}_{-1} \equiv \mathsf{Trap}$ and $\mathsf{Hyb}_{s-1} \equiv \mathsf{Hyb}$. where $\equiv$ means identically distributed.

Next we will use a sequence of intermediate hybrids to show that for each g,

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{g-1}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g) = 1]\right| = \mathsf{negl}(\lambda),$$

and therefore

$$|\Pr[\mathcal{A}(\mathsf{Trap}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}) = 1]| = \mathsf{negl}(\lambda).$$

Below, without loss of generality, we assume the gate g has two input wires denoted $\mathsf{inp}(g) = \{l, r\}$. The proof for any constant number of input wires is similar by iterating over all input wires. We have the following sequence of hybrids:

Hybrid Hyb_g^η. This hybrid is almost the same as Hyb_{g-1}, except that for gate g, we additionally add a checking for the decrypted values m_l of the ciphertext (ct_l, h_l), depending on whether $h_l \leq \eta$ or not. There are $2^{sL_{\mathsf{FHE}}}$ possibility of h_ls, thus η is bounded by $2^{sL_{\mathsf{FHE}}}$.

Modified Gate^g circuit

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

$$h \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K'_j\}_{j\in[s]})),$$
$$h_i \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).$$

- 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 is consistent with (ct_i, h_i):

$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$

  - Proof of consistency for H and H_i with proof π_i:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$

  - MAC σ_i for (ct_i, h_i): $\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K^\sigma_i, (\mathsf{ct}_i, h_i)))$.
- Decrypt the symmetric key ciphertexts ct_i: let

$$m_i = \mathsf{ct}_i \oplus \mathsf{PRF}_m(K^m_i, h_i), \forall i \in \mathsf{inp}(g).$$

- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m^*_w\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- If $h_l \le \eta$ and l is not an input wire of S, then check

$$m_l \stackrel{?}{=} C^l_S(\{m^*_w\}_{w\in\mathsf{inp}(S)}).$$

  Recall that C^l_S is the function that directly calculates the value of wire l from the inputs of subcircuit S.
- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$.
- Encrypt m_o for each $o \in \mathsf{out}(g)$ under K^m_o, and MAC (ct_o, h) under K^σ_o to obtain σ_o, i.e.

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

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

We have $\mathsf{Hyb}^{-1}_g \approx_c \mathsf{Hyb}_{g-1}$ from the security of the underlying indistinguishability obfuscation. Next, we will use a sequence of hybrids to argue that,

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}^\eta_g) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}^{\eta+1}_g) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).$$

Since the case is trivial when l is an input wire of S, we assume l is not an input wire of S without loss of generality.

Hybrid $\mathsf{Hyb}_g^{\eta,0}$. This hybrid is same as $\mathsf{Hyb}_g^{\eta-1}$.

Hybrid $\mathsf{Hyb}_g^{\eta,1}$. This hybrid is almost the same as $\mathsf{Hyb}_g^{\eta,0}$, except that we make the following changes:

Slightly abusing notation, let l also denote the gate that has output wire l. Since $l < g$ according to topological order, it has been already changed into the form in Hyb. So if in the execution of gate l, $h = \eta$, then the ciphertext of output wire l is fixed. Let it be CT_l:

$$\mathsf{CT}_l = C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)}) \oplus \mathsf{PRF}_m(K_l^m, \eta), \{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, \eta).$$

Let $\overline{\mathsf{CT}_l} = \mathsf{CT}_l \oplus 1$. We replace the obfuscation corresponding to gate l with an obfuscation of the following circuit:

Modified Gate^l circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

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

$$m_o = C_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)}), \forall o \in \mathsf{out}(l).$$

- For each $o \in \mathsf{out}(l)$, encrypt m_o under K_o^m:

$$\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m, h).$$

- For each $o \in \mathsf{out}(l)$, MAC (ct_o, h) under K_o^σ to obtain σ_o:
  - If $o \neq l$, then calculate
  $$\sigma_o = \mathsf{PRF}_\sigma(K_o^\sigma, (\mathsf{ct}_o, h)).$$
  - Otherwise, $o = l$, calculate
  $$\sigma_l = \mathsf{PRF}_\sigma(K_l^\sigma\{(\overline{\mathsf{CT}_l}, \eta)\}, (\mathsf{ct}_l, h)).$$
  Here $K_l^\sigma\{(\overline{\mathsf{CT}_l}, \eta)\}$ is hard-coded in the circuit.
- Output $\{\mathsf{ct}_o, \sigma_o\}_{o\in\mathsf{out}(l)}$.

Moreover, we modify the Gate^g circuit as follows:

Modified Gate^g circuit

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

$$h \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K_j'\}_{j\in[s]})),$$
$$h_i \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K_j'\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).$$

- 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 is consistent with (ct_i, h_i):
$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$
  - Proof of consistency for H and H_i with proof π_i:
$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$
  - MAC σ_i for (ct_i, h_i): If $i \neq l$, then check
$$\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i))).$$
    Otherwise, if $i = l$:
    * If $h_l = \eta$ and $\mathsf{ct}_l = \overline{\mathsf{CT}_l}$, then verify
$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\underbrace{\mathsf{PRF}_\sigma(K_l^\sigma, (\overline{\mathsf{CT}_l}, \eta))}_{\text{hard-wired}}).$$
    * Otherwise, verify
$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(\underbrace{K_l^\sigma\{(\overline{\mathsf{CT}_l}, \eta)\}}_{\text{hard-wired}}, (\mathsf{ct}_l, h_l))).$$
    In the above, $K_l^\sigma\{(\overline{\mathsf{CT}_l}, \eta)\}$ and $\mathsf{PRF}_\sigma(K_l^\sigma, (\overline{\mathsf{CT}_l}, \eta))$ are hard-wired in the circuit.
- Decrypt the symmetric key ciphertexts ct_i: let
$$m_i = \mathsf{ct}_i \oplus \mathsf{PRF}_m(K_i^m, h_i), \forall i \in \mathsf{inp}(g).$$
- Extract the inputs to the subcircuit S via the decryption of FHE:
$$\{m_w^*\}_{w \in \mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$
- If $h_l < \eta$, then check
$$m_l \stackrel{?}{=} C_S^l(\{m_w^*\}_{w \in \mathsf{inp}(S)}).$$
- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i \in \mathsf{inp}(g)})$.
- Encrypt m_o for each $o \in \mathsf{out}(g)$ under K_o^m, and MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.
$$\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m, h), \sigma_o = \mathsf{PRF}_\sigma(K_o^\sigma, (\mathsf{ct}_o, h)), \forall o \in \mathsf{out}(g).$$
- Output $\{\mathsf{ct}_o, \sigma_o\}_{o \in \mathsf{out}(g)}$.

We now argue the indistinguishability of $\mathsf{Hyb}_g^{\eta,0}$ and $\mathsf{Hyb}_g^{\eta,1}$.

First by the definition of $\overline{\mathsf{CT}_l}$, $(\overline{\mathsf{CT}_l}, \eta)$ will never occur in the MAC step of l, so our modification to gate l does not change its functionality. By the security of the underlying indistinguishability

obfuscation, the obfuscation of l is indistinguishable in these two hybrids.

Our modification of the gate g also does not change its functionality, so by the security of the underlying indistinguishability obfuscation, we have

$$\begin{aligned}&\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,0})=1]-\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,1})=1]\right|\\ =&2^{-\lambda_{\mathsf{obf}}{}^{\epsilon_{\mathsf{obf}}}}\mathsf{negl}(\lambda)\\ \leq&2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).\end{aligned}$$

Hybrid $\mathsf{Hyb}_g^{\eta,2}$. This hybrid is almost the same as $\mathsf{Hyb}_g^{\eta,1}$, except that we replace Gate^g with the following:

Modified Gate^g circuit

- $\cdots$
- If $i \neq l$, then check
$$\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i))).$$
Otherwise, if $i = l$:
  - If $h_l = \eta$ and $\mathsf{ct}_l = \overline{\mathsf{CT}_l}$, verify
$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} r,$$
where r is a random string of length $2\lambda_{\mathsf{PRG}}$ as output of PRG, hard-coded in the circuit.
  - Otherwise, verify
$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_l^\sigma\{(\overline{\mathsf{CT}_l}, \eta)\}, (\mathsf{ct}_l, h_l))).$$
- $\cdots$

Since the master key K_l^σ does not appear anywhere else, by the security of puncturable PRF and PRG, we have

$$\begin{aligned}&\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,1})=1]-\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,2})=1]\right|\\ =&(2^{-\lambda_{\mathsf{PRG}}{}^{\epsilon_{\mathsf{PRG}}}}+2^{-\lambda_{\mathsf{PRF}}{}^{\epsilon_{\mathsf{PRF}}}})\mathsf{negl}(\lambda)\\ \leq&2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).\end{aligned}$$

Hybrid $\mathsf{Hyb}_g^{\eta,3}$. This hybrid is almost the same as $\mathsf{Hyb}_g^{\eta,2}$, except that we replace Gate^g with the following:

Modified Gate^g circuit

- $\cdots$
- If $i \neq l$, then check
$$\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i))).$$
Otherwise, if $i = l$:

– If $h_l = \eta$ and $\mathsf{ct}_l = \overline{\mathsf{CT}_l}$, abort.

– Otherwise, verify

$$\mathsf{PRG}(\sigma_l) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_l^\sigma\{(\overline{\mathsf{CT}_l}, \eta)\}, (\mathsf{ct}_l, h_l))).$$

- ...

For gate l, we undo its changes in $\mathsf{Hyb}_g^{\eta,1}$ (which does not affect its functionality).

If we set λ_{PRG} to be large enough, then with overwhelming probability (at least $1 - 2^{-\lambda_{\mathsf{PRG}}}$) a random r does not lie in the domain of the PRG, so the functionality of gate g does not change. By the security of the underlying indistinguishability obfuscation, we have

$$\begin{aligned}
&\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,2}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,3}) = 1]\right| \\
\leq & 2^{-\lambda_{\mathsf{PRG}}} + 2^{-\lambda_{\mathsf{obf}}\epsilon_{\mathsf{obf}}}\mathsf{negl}(\lambda) \\
\leq & 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).
\end{aligned}$$

Hybrid $\mathsf{Hyb}_g^{\eta,4}$. This hybrid is almost the same as $\mathsf{Hyb}_g^{\eta,3}$, except that we replace Gate^g with the following:

Modified Gate^g circuit

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

$$\begin{aligned}
h &\leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \\
h_i &\leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).
\end{aligned}$$

- 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 is consistent with (ct_i, h_i):

$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$

  – Proof of consistency for H and H_i with proof π_i:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$

  – MAC σ_i for (ct_i, h_i): $\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i)))$.

- Decrypt the symmetric key ciphertexts ct_i: let

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

- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- (Recall that we assume l is not an input wire of S.) If $h_l < \eta$, then check

$$m_l \stackrel{?}{=} C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)}).$$

- If $h_l = \eta$, then check

$$m_l \stackrel{?}{=} \mathsf{CT}_l \oplus \mathsf{PRF}_m(K_l^m, \eta).$$

- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$.
- Encrypt m_o for each $o \in \mathsf{out}(g)$ under K_o^m, and MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

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

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

The difference is: when $h_l = \eta$, in $\mathsf{Hyb}_g^{\eta,3}$, we check $\mathsf{ct}_l \stackrel{?}{=} \mathsf{CT}_l$; but in $\mathsf{Hyb}_g^{\eta,4}$, we set $m_l = \mathsf{ct}_l \oplus \mathsf{PRF}_m(K_l^m, \eta)$ and check $m_l \stackrel{?}{=} \mathsf{CT}_l \oplus \mathsf{PRF}_m(K_l^m, \eta)$.

Since the functionality of g does not change, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,3}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,4}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda)$$

by the security of the underlying indistinguishability obfuscation.

Hybrid $\mathsf{Hyb}_g^{\eta,5}$. This hybrid is the same as Hyb_g^{η}. The difference of Gate^g in $\mathsf{Hyb}_g^{\eta,4}$ and $\mathsf{Hyb}_g^{\eta,5}$ is:

- In $\mathsf{Hyb}_g^{\eta,4}$, when $h_l = \eta$, we check whether $m_l = \mathsf{CT}_l \oplus \mathsf{PRF}_m(K_l^m, \eta)$. Note that by the definition of CT_l:

$$\mathsf{CT}_l = C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)}) \oplus \mathsf{PRF}_m(K_l^m, \eta), \{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, \eta),$$

  it is equivalent to check whether $m_l = C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)}), \{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, \eta)$.
- In $\mathsf{Hyb}_g^{\eta,5}$, when $h_l = \eta$, we check whether $m_l = C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)}), \{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h)$.

Note that the only difference is $\mathsf{FHE.Dec}(\mathsf{sk}, \eta)$ and $\mathsf{FHE.Dec}(\mathsf{sk}, h)$.

Recall that in gate g we do a consistency verification:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_l, l, \pi_l) \stackrel{?}{=} 1,$$

and h, h_l are extracted from H, H_l, so when $h_l = \eta$, by the somewhere statistical soundness of SEH, $\mathsf{FHE.Dec}(\mathsf{sk}, \eta)$ and $\mathsf{FHE.Dec}(\mathsf{sk}, h)$ must agree on the set $\mathsf{inp}(S) \cap [: l]$.

Let $\mathsf{dep}(l)$ denote the wires that l depends on. The topological order of these wires must be smaller than l, so $\mathsf{dep}(l) \subseteq [: l]$, and thus $\mathsf{FHE.Dec}(\mathsf{sk}, \eta)$ and $\mathsf{FHE.Dec}(\mathsf{sk}, h)$ must agree on $\mathsf{inp}(S) \cap \mathsf{dep}(l)$.

Note that when we calculate $C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, its value fully depends on $\{m_w^*\}_{w\in\mathsf{inp}(S)\cap\mathsf{dep}(l)}$. Hence the functionality of gate g does not change in $\mathsf{Hyb}_g^{\eta,5}$ and $\mathsf{Hyb}_g^{\eta,4}$, and from the security of the underlying indistinguishability obfuscation, we get

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,4}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta,5}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).$$

From previous intermediate hybrids, we prove

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g^{\eta+1}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda),$$

therefore,

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{g-1}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g^{2^{sL_{\mathsf{FHE}}}}) = 1]\right| = \mathsf{negl}(\lambda),$$

Hybrid $\mathsf{HybCheckL}_g$. This hybrid is almost the same as $\mathsf{Hyb}_g^{2^{sL_{\mathsf{FHE}}}}$, except that we replace gate G^g with:

Modified Gate^g circuit

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

$$h \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K'_j\}_{j\in[s]})),$$
$$h_i \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).$$

- 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 is consistent with (ct_i, h_i):

$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$

  - Proof of consistency for H and H_i with proof π_i:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$

  - MAC σ_i for (ct_i, h_i): $\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K_i^\sigma, (\mathsf{ct}_i, h_i)))$.
- Decrypt the symmetric key ciphertexts ct_i: let

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

- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- If l is not an input wire of S, check $m_l \stackrel{?}{=} C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$.
- Encrypt m_o for each $o \in \mathsf{out}(g)$ under K_o^m, and MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

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

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

The functionality of gate g does not change, hence by the security of the underlying indistinguishability obfuscation, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_g^{2^{sL_{\mathsf{FHE}}}}) = 1] - \Pr[\mathcal{A}(\mathsf{HybCheckL}_g) = 1]\right| = \mathsf{negl}(\lambda).$$

Hybrid $\mathsf{HybCheckLR}_g$. This hybrid is almost the same as $\mathsf{HybCheckL}_g$, except that we replace Gate^g with:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- If l is not an input wire of S, check $m_l \stackrel{?}{=} C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- If r is not an input wire of S, check $m_r \stackrel{?}{=} C_S^r(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$.
- $\cdots$

Similarly, we can prove

$$\left|\Pr[\mathcal{A}(\mathsf{HybCheckL}_g) = 1] - \Pr[\mathcal{A}(\mathsf{HybCheckLR}_g) = 1]\right| = \mathsf{negl}(\lambda).$$

Hybrid $\mathsf{HybCheck}_g$. This hybrid is almost the same as $\mathsf{HybCheckLR}_g$, except that we replace Gate^g with:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- If l is not an input wire of S, check $m_l \stackrel{?}{=} C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- If r is not an input wire of S, check $m_r \stackrel{?}{=} C_S^r(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- If l is an input wire of S, check $m_l \stackrel{?}{=} m_l^*$.
- If r is an input wire of S, check $m_r \stackrel{?}{=} m_r^*$.
- Compute the wire value m_o for each output wire $o \in \mathsf{out}(g)$ as $m_o = g_o(\{m_i\}_{i\in\mathsf{inp}(g)})$.
- $\cdots$

Since we also do a local opening verification in gate g, by the extraction correctness of SEH, the functionality of g does not change, so

$$\left|\Pr[\mathcal{A}(\mathsf{HybCheckLR}_g) = 1] - \Pr[\mathcal{A}(\mathsf{HybCheck}_g) = 1]\right| = \mathsf{negl}(\lambda)$$

by the security of the underlying indistinguishability obfuscation.

Hybrid $\mathsf{HybCheckDirect}_g$. This hybrid is almost the same as $\mathsf{HybCheck}_g$, except that we directly calculate the outputs of g by the decryption values of FHE:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- If l is not an input wire of S, check $m_l \stackrel{?}{=} C_S^l(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- If r is not an input wire of S, check $m_r \stackrel{?}{=} C_S^r(\{m_w^*\}_{w\in\mathsf{inp}(S)})$.
- If l is an input wire of S, check $m_l \stackrel{?}{=} m_l^*$.
- If r is an input wire of S, check $m_r \stackrel{?}{=} m_r^*$.
- Compute the output wire values $\{m_o\}_{o\in\mathsf{out}(g)}$:

$$m_o = C_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)}).$$

- $\cdots$

Since we have the checkings for m_l and m_r, the functionality of g does not change. By the security of the underlying indistinguishability obfuscation, we have

$$\left|\Pr[\mathcal{A}(\mathsf{HybCheck}_g) = 1] - \Pr[\mathcal{A}(\mathsf{HybCheckDirect}_g) = 1]\right| = \mathsf{negl}(\lambda).$$

Finally, note that the difference of $\mathsf{HybCheckDirect}_g$ and Hyb_g is that $\mathsf{HybCheckDirect}_g$ additionally contains the checkings. We can undo these checkings and use similar hybrid argument to show that

$$\left|\Pr[\mathcal{A}(\mathsf{HybCheckDirect}_g) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g) = 1]\right| = \mathsf{negl}(\lambda).$$

In conclusion, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{g-1}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_g) = 1]\right| = \mathsf{negl}(\lambda).$$

Indeed Trap and Hyb are indistinguishable.

Hybrid $\widetilde{\mathsf{Hyb}}$. In this hybrid, for each gate in subcircuit S, instead of computing each output wires by C, we compute it by the circuit $\widetilde{C}$. Namely, for each gate $g \in S$, we obfuscate the following circuit:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

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

$$m_o = \widetilde{C}^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)}).$$

- ...

To show that Hyb and $\widetilde{\mathsf{Hyb}}$ are indistinguishable, we use a series of intermediate hybrids.

Hybrid $\widetilde{\mathsf{Hyb}}_g$. For each gate g' where $g' \le g$, it computes its output using $\widetilde{C}$. But for each gate g' with $g' > g$, it still uses C to compute its output wires.

We will use a series of hybrids to show that for each $g \in S$, $\widetilde{\mathsf{Hyb}}_{g-1}$ and $\widetilde{\mathsf{Hyb}}_g$ are indistinguishable. For the simplicity of notation, we assume g only has one output wire o (it is not hard to generalize the proof for multiple output wires — see Remark 5.4). **If o is an output wire of S, since the functionality of the subcircuit S is preserved, the case is trivial. So without loss of generality we assume $o \notin \mathsf{out}(S)$.**

Hybrid $\widetilde{\mathsf{Hyb}}^{\eta}_g$. This hybrid is almost the same as $\widetilde{\mathsf{Hyb}}_{g-1}$, except that for gate g, we compute the wire value m_o and its ciphertext ct_o as the following (again η is bounded by $2^{sL_{\mathsf{FHE}}}$):

Modified Gate^g circuit

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

$$h \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_H, (\mathsf{td}', \{K'_j\}_{j\in[s]})),$$
$$h_i \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{Shrink}_{H_i}, (\mathsf{td}', \{K'_j\}_{j\in[s]})), \forall i \in \mathsf{inp}(g).$$

- 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 is consistent with (ct_i, h_i):

$$\mathsf{SEH.Verify}(\mathsf{hk}, H, i, (\mathsf{ct}_i, h_i), \rho_i) \stackrel{?}{=} 1.$$

  - Proof of consistency for H and H_i with proof π_i:

$$\mathsf{SEH.ConsisV}(\mathsf{hk}, H, H_i, i, \pi_i) \stackrel{?}{=} 1.$$

  - MAC σ_i for (ct_i, h_i): $\mathsf{PRG}(\sigma_i) \stackrel{?}{=} \mathsf{PRG}(\mathsf{PRF}_\sigma(K^\sigma_i, (\mathsf{ct}_i, h_i)))$.
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m^*_w\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- Compute the output wire value m_o:
  - If $h \le \eta$, compute $m_o = \widetilde{C}^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$.
  - If $h > \eta$, compute $m_o = C^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$.

- Encrypt m_o under K_o^m, and MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

$$\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m, h), \sigma_o = \mathsf{PRF}_\sigma(K_o^\sigma, (\mathsf{ct}_o, h)).$$

- Output $(\mathsf{ct}_o, \sigma_o)$.

We will use a series of hybrids to argue that

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta-1}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).$$

Hybrid $\widetilde{\mathsf{Hyb}}_g^{\eta,0}$. This hybrid is the same as $\widetilde{\mathsf{Hyb}}_g^{\eta-1}$.

Hybrid $\widetilde{\mathsf{Hyb}}_g^{\eta,1}$. This hybrid is almost the same as $\widetilde{\mathsf{Hyb}}_g^{\eta,0}$, except that we replace Gate^g with the following:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- Compute the output wire value m_o and corresponding ciphertext ct_o:
  - If $h < \eta$, compute $m_o = \widetilde{C}_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m\{\eta\}, h)$.
  - If $h = \eta$, compute $m_o = C_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m, \eta)$.
  - If $h > \eta$, compute $m_o = C_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m\{\eta\}, h)$.

  Here $K_o^m\{\eta\}$ and $\mathsf{PRF}_m(K_o^m, \eta)$ are hard-coded in the circuit.
- MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

$$\sigma_o = \mathsf{PRF}_\sigma(K_o^\sigma, (\mathsf{ct}_o, h)).$$

- Output $(\mathsf{ct}_o, \sigma_o)$.

Since the functionality of g does not change,

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,0}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,1}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda)$$

from the security of the underlying indistinguishability obfuscation.

Hybrid $\widetilde{\mathsf{Hyb}}_g^{\eta,2}$. This hybrid is almost the same as $\widetilde{\mathsf{Hyb}}_g^{\eta,1}$, except that we replace Gate^g with the following where r a random bit:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m^*_w\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- Compute the output wire value m_o and corresponding ciphertext ct_o:
  - If $h < \eta$, compute $m_o = \widetilde{C}^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K^m_o\{\eta\}, h)$.
  - If $h = \eta$, compute $m_o = C^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus r$.
  - If $h > \eta$, compute $m_o = C^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K^m_o\{\eta\}, h)$.

  Here $K^m_o\{\eta\}$ and random value r are hard-coded in the circuit.
- MAC (ct_o, h) under K^σ_o to obtain σ_o, i.e.

$$\sigma_o = \mathsf{PRF}_\sigma(K^\sigma_o, (\mathsf{ct}_o, h)).$$

- Output $(\mathsf{ct}_o, \sigma_o)$.

Since o is not an output wire of S, let g_o be the gate that has input wire o, then $g_o \in S$. In this hybrid, g_o will compute its outputs directly from C_S, and it will not use K^m_o to decrypt the input ciphertext ct_o (see Hyb).

So the unpunctured version of K^m_o does not occur anywhere else. By the security of puncturable PRF, we have

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}^{\eta,1}_g) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}^{\eta,2}_g) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).$$

Hybrid $\widetilde{\mathsf{Hyb}}^{\eta,3}_g$. This hybrid is almost the same as $\widetilde{\mathsf{Hyb}}^{\eta,2}_g$, except that we replace Gate^g with the following:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m^*_w\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- Compute the output wire value m_o and corresponding ciphertext ct_o:
  - If $h < \eta$, compute $m_o = \widetilde{C}^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K^m_o\{\eta\}, h)$.
  - If $h = \eta$, compute $m_o = \widetilde{C}^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus r$.
  - If $h > \eta$, compute $m_o = C^o_S(\{m^*_w\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K^m_o\{\eta\}, h)$.

  Here $K^m_o\{\eta\}$ and random value r are hard-coded in the circuit.
- MAC (ct_o, h) under K^σ_o to obtain σ_o, i.e.

$$\sigma_o = \mathsf{PRF}_\sigma(K^\sigma_o, (\mathsf{ct}_o, h)).$$

- Output $(\mathsf{ct}_o, \sigma_o)$.

Since r is uniformly random, the distribution of ct_o does not change, so we have

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,2}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,3}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda)$$

by the security of the underlying indistinguishability obfuscation.

Hybrid $\widetilde{\mathsf{Hyb}}_g^{\eta,4}$. This hybrid is almost the same as $\widetilde{\mathsf{Hyb}}_g^{\eta,3}$, except that we replace Gate^g with the following:

Modified Gate^g circuit

- $\cdots$
- Extract the inputs to the subcircuit S via the decryption of FHE:

$$\{m_w^*\}_{w\in\mathsf{inp}(S)} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, h).$$

- Compute the output wire value m_o and corresponding ciphertext ct_o:
  - If $h < \eta$, compute $m_o = \widetilde{C}_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m\{\eta\}, h)$.
  - If $h = \eta$, compute $m_o = \widetilde{C}_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m, \eta)$.
  - If $h > \eta$, compute $m_o = C_S^o(\{m_w^*\}_{w\in\mathsf{inp}(S)})$, and $\mathsf{ct}_o = m_o \oplus \mathsf{PRF}_m(K_o^m\{\eta\}, h)$.

  Here $K_o^m\{\eta\}$ and $\mathsf{PRF}_m(K_o^m, \eta)$ are hard-coded in the circuit.
- MAC (ct_o, h) under K_o^σ to obtain σ_o, i.e.

$$\sigma_o = \mathsf{PRF}_\sigma(K_o^\sigma, (\mathsf{ct}_o, h)).$$

- Output $(\mathsf{ct}_o, \sigma_o)$.

By the security of puncturable PRF, we have

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,3}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,4}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).$$

Hybrid $\widetilde{\mathsf{Hyb}}_g^{\eta,5}$. This hybrid is the same as $\widetilde{\mathsf{Hyb}}_g^{\eta}$. Since the functionality of g does not change in $\widetilde{\mathsf{Hyb}}_g^{\eta,5}$ and $\widetilde{\mathsf{Hyb}}_g^{\eta,4}$, we have

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,4}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta,5}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda)$$

by the security of the underlying indistinguishability obfuscation.

From above hybrid argument, we have

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta-1}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g^{\eta}) = 1]\right| = 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda).$$

Further,

$$\left|\Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_{g-1}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathsf{Hyb}}_g) = 1]\right| = \mathsf{negl}(\lambda),$$

hence Hyb and $\widetilde{\mathsf{Hyb}}$ are indistinguishable.

Remark 5.4. *If the gate g has multiple outputs, we can extend the proof by going over the above hybrid sequence and change each output wire one by one.*

Hybrid $\widetilde{\mathsf{Real}}$. This hybrid is the same as $\mathsf{LiO}(1^\lambda, s, \widetilde{C})$. By a reverse hybrid argument, we can show that $\widetilde{\mathsf{Hyb}}$ and $\widetilde{\mathsf{Real}}$ are indistinguishable.

In conclusion, Real and $\widetilde{\mathsf{Real}}$ are indistinguishable, i.e.,

$$\left|\Pr[\mathcal{A}(\mathsf{LiO}(1^\lambda, s, C)) = 1] - \Pr[\mathcal{A}(\mathsf{LiO}(1^\lambda, s, \widetilde{C})) = 1]\right| = \mathsf{negl}(\lambda).$$

□

5.4 Efficiency

Below we assume that the size of the circuit to be obfuscated is bounded by n and the size of the differing (but functionally equivalent) subcircuit is bounded by $s = O(\log n)$.

Size of obfuscated circuit. Since $s = O(\log n)$, and due to the succinctness of the underlying SEH, we know that the size of the augmented circuit for each gate g is upper bounded by $\widetilde{O}_\lambda(1)$. Therefore, applying a polynomial-overhead indistinguishability obfuscation to this augmented circuit results in an obfuscated program of size $\widetilde{O}_\lambda(1)$. Recall that we go through a gate-by-gate obfuscation for the entire circuit, and the obfuscated program size is dominated by the total size of the obfuscated programs corresponding to all gates. Thus, the size of the obfuscated circuit is upper bounded by $\widetilde{O}_\lambda(n)$.

Time for evaluating the obfuscated circuit. The time for evaluating each obfuscated gate is $\widetilde{O}_\lambda(1)$. However, evaluating the obfuscated circuit also requires evaluating the SEH hashes for all wires, and computing the opening and consistency proofs for all gates. If done naïvely, evaluating all n wires' hash values will take at least quadratic in n time since each hash involves a length-n input. Thus, to make the evaluation quasi-linear time, we need to show that it takes only $\widetilde{O}_\lambda(n)$ time to evaluate all n hashes and all the opening and consistency proofs. Fortunately, all n hash values can indeed be computed incrementally in $\widetilde{O}_\lambda(n)$ time due to the tree structure, each time modifying only $O(\log n)$ hash computations. With some additional tricks (described in the proof of Claim 5.5), we can compute all the opening and consistency proofs in $\widetilde{O}_\lambda(n)$ time too. Thus we have the following claim:

Claim 5.5. *The evaluation algorithm can be carefully implemented such that for a size-n circuit C, it runs in $\widetilde{O}_\lambda(n)$ time.*

Proof. The performance bottleneck of the evaluation algorithm is the following SEH operations:

- For each $w \in \{0, 1, \ldots, n-1\}$, the evaluation algorithm should prepare the hash value H_w. Recall that $s_w = (s_{w,0}, s_{w,1}, \ldots, s_{w,n-1})$ where

$$s_{w,w'} = \begin{cases} (\mathsf{ct}_{w'}, h_{w'}) & \text{for} \quad w' < w \\ (0, 0^{|h_{w'}|}) & \text{for} \quad w' \geq w \end{cases}$$

  and

$$H_w \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, s_w).$$

- For each $w \in \{0, 1, \ldots, n-1\}$, suppose it is the output wire of g, then the evaluation algorithm needs to prepare local openings $\{\mathsf{SEH.Open}(\mathsf{hk}, s_w, i)\}_{i \in \mathsf{inp}(g)}$. There are totally $O(n)$ such operations.
- For each $w \in \{0, 1, \ldots, n-1\}$, suppose it is the output wire of g, then the evaluation algorithms needs to prepare consistency proofs $\{\mathsf{SEH.ConsisP}(\mathsf{hk}, s_w, s_i, i)\}_{i \in \mathsf{inp}(g)}$. There are totally $O(n)$ such operations.

We will show that, the above operations can be done in $\widetilde{O}_\lambda(n)$ time.

For each w, we maintain the Merkle-tree structure $\mathsf{MKT}_w = \{\mathsf{ct}_v\}_{v \in T}$ when calculating the hash value $\mathsf{SEH.Hash}(\mathsf{hk}, s_w)$.

At first, we set $s_{-1} = (0, 0, \ldots, 0)$ and maintain the structure of MKT_{-1}.

- For each $w \in \{0, 1, \ldots, n-1\}$, when calculating H_w, we have already maintained the Merkle-tree structure MKT_{w-1}. If we want to update MKT_{w-1} to MKT_w, we only need to modify $O(\log n)$ vertices: the path from leaf w to the root. So the first type of operations can be done in $\widetilde{O}_\lambda(n)$ time.
- With the structure MKT_w, we only need to access $O(\log n)$ vertices to calculate a local opening $\mathsf{SEH.Open}(\mathsf{hk}, s_w, i)$. So the second type of operations can also be done in $\widetilde{O}_\lambda(n)$ time.
- For the third type of operations, again we only need to access $O(\log n)$ vertices to calculate a consistency proof $\mathsf{SEH.ConsisP}(\mathsf{hk}, s_w, s_i, i)$ with MKT_w and MKT_i. But now the problem is, when we see MKT_w, the structure of MKT_i has already been modified, i.e., we cannot access the two structures at the same time. Specifically, for any pair of (w, i), the consistency proof will always be for prefix $s_w[:i]$. The $b = 2$ part of the consistency proof (see Section 4.2) can be determined by only MKT_i. Therefore, for each $i \in \{0, 1, \ldots, n-1\}$, when the MKT_i structure is about to be modified, we can store a hint hint_i which is the $b = 2$ part of $\mathsf{SEH.ConsisP}(\mathsf{hk}, s_w, s_i, i)$ for any $w > i$. Later, when the evaluation algorithm needs to calculate $\mathsf{SEH.ConsisP}(\mathsf{hk}, s_w, s_i, i)$, it can use hint_i as the $b = 2$ part, and calculate the $b = 1$ part based on MKT_w. So the third type of operations can also be done in $\widetilde{O}_\lambda(n)$ time.

□

In conclusion, combining Corollary 5.2, Claim 5.5 and Lemma 3.1, we obtain the following theorem:

Theorem 5.6 (Indistinguishability obfuscation for circuits with proof of equivalence)**.** *Assume the following hardness assumptions:*

- *polynomial hardness of* LWE*,*
- *sub-exponentially secure one-way functions, and*
- *sub-exponentially secure indistinguishability obfuscation for circuits of size* $\mathsf{poly}(\lambda, \log N_{\mathrm{circ}}, \log N_{\mathrm{proof}})$*.*

Then, there exists a (polynomially secure) indistinguishability obfuscator, and for circuit families of size at most N_{circ} *with propositional equivalence proofs of size at most* N_{proof}*, the obfuscated program size, obfuscation time, and evaluator time is bounded by* $\widetilde{O}_\lambda(N_{\mathrm{circ}} + N_{\mathrm{proof}})$*.*

5.5 Extension to Sub-Exponentially Secure Obfuscator

The obfuscation construction in Section 5.2 only achieves polynomial security while we have assumed the sub-exponential security of one-way functions and the underlying indistinguishability obfuscator Obf. This asymmetry naturally leads us to explore a quasi-linear obfuscator with sub-exponential security. In fact, if we additionally assume the existence of sub-exponentially secure FHE, by fine-tuning parameters, we can derive the following theorem:

Theorem 5.7 (Sub-exponentially secure indistinguishability obfuscation for circuits with proof of equivalence)**.** *Assume the following hard problems:*

- *sub-exponential hardness of the* LWE *assumption,*
- *sub-exponentially secure one-way functions, and*
- *sub-exponentially secure indistinguishability obfuscation for circuits of size* $\mathsf{poly}(\lambda, \log N_{\mathrm{circ}}, \log N_{\mathrm{proof}})$.

For any $\epsilon > 0$, there exists 2^{λ^ϵ} sub-exponential secure indistinguishability obfuscation and evaluation algorithm for circuit families of size at most N_{circ} with propositional proof of maximum size N_{proof}, and runs in time $\widetilde{O}_\lambda(N_{\mathrm{circ}} + N_{\mathrm{proof}})$.

Proof. We will use essentially same construction as Section 5.2 but with different parameter regimes. Specifically, we require following building blocks including sub-exponential secure FHE and sub-exponential secure SEH with prefix consistency proof (recall that these two primitives are both implied by sub-exponential hardness of LWE):

- $2^{\lambda^{\epsilon_{\mathsf{FHE}}}}$ secure fully hormomorphic encryption $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with security parameter $\lambda_{\mathsf{FHE}} = \lambda^{\epsilon/\epsilon_{\mathsf{FHE}}}$ and output length $\lambda_{\mathsf{FHE}} = \mathsf{poly}(\lambda)$ per bit.
- $2^{\lambda^{\epsilon_{\mathsf{PRG}}}}$ secure pseudorandom generator $\mathsf{PRG} : \{0,1\}^{\lambda_{\mathsf{PRG}}} \to \{0,1\}^{2\lambda_{\mathsf{PRG}}}$ with security parameter $\lambda_{\mathsf{PRG}} = (\lambda^\epsilon + sL_{\mathsf{FHE}})^{1/\epsilon_{\mathsf{PRG}}} = \mathsf{poly}(\lambda, s)$.
- $2^{\lambda^{\epsilon_{\mathsf{PRF}}}}$ secure puncturable pseudorandom functions $\mathsf{PRF}_m : \{0,1\}^{\lambda_{\mathsf{PRF}}} \times \{0,1\}^{sL_{\mathsf{FHE}}} \to \{0,1\}$ with security parameter $\lambda_{\mathsf{PRF}} = (\lambda^\epsilon + sL_{\mathsf{FHE}})^{1/\epsilon_{\mathsf{PRF}}} = \mathsf{poly}(\lambda, s)$.
- $2^{\lambda^{\epsilon_{\mathsf{PRF}}}}$ secure puncturable pseudorandom functions $\mathsf{PRF}_\sigma : \{0,1\}^{\lambda_{\mathsf{PRF}}} \times \{0,1\}^{sL_{\mathsf{FHE}}+1} \to \{0,1\}^{\lambda_{\mathsf{PRG}}}$.
- $2^{\lambda^{\epsilon_{\mathsf{FHE}}}}$ secure somewhere extractable hash with prefix consistency proof $\mathsf{SEH} = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext}, \mathsf{ConsisP}, \mathsf{ConsisV})$, with security parameter λ_{FHE} and maintained by a Merkle-tree structure MKT.
- $2^{\lambda^{\epsilon_{\mathsf{obf}}}}$ secure indistinguishability obfuscator Obf with security parameter $\lambda_{\mathsf{obf}} = (\lambda^\epsilon + sL_{\mathsf{FHE}})^{1/\epsilon_{\mathsf{obf}}} = \mathsf{poly}(\lambda, s)$.

One can verify a security proof analogous to Section 5.3 shows the 2^{λ^ϵ}-security of our LiO scheme, because for each hybrid step, if it is based on security of the underlying FHE and SEH scheme, the advantage of adversary can be bounded by

$$2^{-\lambda_{\mathsf{FHE}}{}^{\epsilon_{\mathsf{FHE}}}}\mathsf{negl}(\lambda) = 2^{-\lambda^\epsilon}\mathsf{negl}(\lambda);$$

and for the remaining hybrid steps that are based on security of the underlying PRG, PRF and Obf, the advantage of adversary is bounded by

$$\begin{aligned} 2^{-\lambda_{\mathsf{PRG}}{}^{\epsilon_{\mathsf{PRG}}}} &= 2^{-\lambda^\epsilon} \cdot 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda) \\ 2^{-\lambda_{\mathsf{PRF}}{}^{\epsilon_{\mathsf{PRF}}}} &= 2^{-\lambda^\epsilon} \cdot 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda) \\ 2^{-\lambda_{\mathsf{obf}}{}^{\epsilon_{\mathsf{obf}}}} &= 2^{-\lambda^\epsilon} \cdot 2^{-sL_{\mathsf{FHE}}}\mathsf{negl}(\lambda) \end{aligned}$$

□

6 Multi-Input Functional Encryption with Quasi-Linear Efficiency

In this section, we show how to construct efficient multi-input functional encryption as an application of our quasi-linear locally indistinguishability obfuscator LiO. Specifically, our technique achieves quasi-linear efficiency in following settings: 1) single-input (public-key) functional encryption; and 2) $\mathsf{poly}(\lambda)$-input secret-key functional encryption.

6.1 Definition: Multi-Input Functional Encryption

We first review the notion of multi-input functional encryption (MIFE) adapting from [GGG+14].

Syntax. Let λ denote the security parameter, $t = t(\lambda)$ denote the function arity where $t(\lambda)$ is some polynomial over λ. Let $L(\lambda)$ denote the bit-length of each message. Let $\mathcal{F}_\lambda$ denote a family of of PPT functions that map inputs from $(\{0,1\}^L)^t$ to an output in $\{0,1\}^{L_{\mathrm{out}}(\lambda)}$. A multi-input functional encryption scheme w.r.t. the family $\mathcal{F}_\lambda$ consists of following PPT algorithms:

- $(\{\mathsf{ek}_i\}_{i\in[t]}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^t, 1^L)$: On input a security parameter λ, the function arity t and message length L, the setup algorithm outputs t encryption keys $\{\mathsf{ek}_i\}_{i\in[t]}$ and a master secret key msk.
- $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ek}_i, x_i)$: Takes in an encryption key ek_i and a message $x_i \in \{0,1\}^L$ as inputs, and outputs a ciphertext ct_i.
- $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$: On input a master secret key msk and a description of a t-arity function f, the key generation algorithm outputs a functional key sk_f.
- $y \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_t)$: On inputs a functional key sk_f and t ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_t$, the decryption algorithm outputs a decrypted message y.

Correctness. For any λ, t, L, for any t-arity function f and input messages $x_1, \ldots, x_t$ where $x_i \in \{0,1\}^L$, the following must hold:

$$\Pr\left[\begin{array}{ll}(\{\mathsf{ek}_i\}_{i\in[t]}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^t, 1^L), & \\ \forall i \in [t] : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ek}_i, x_i), & : \mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_t) = f(x_1, \ldots, x_t) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) & \end{array}\right] = 1.$$

Security. An adversary $\mathcal{A}$ can hold a subset of encryption keys $\{\mathsf{ek}_i\}_{i\in I}$ where $I \subseteq [t]$ where I denotes the corrupted subset. We consider an indistinguishability-based security notion, where the adversary is allowed to submit challenge messages after receiving encryption keys $\{\mathsf{ek}_i\}_{i\in I}$, but before making any key-generation query — such a notion is often referred to as "semi-adaptive" [GKW16, CW14]. Formally, the MIFE scheme is said to be secure w.r.t. to $I \subseteq [t]$, iff for any t, L polynomially bounded in the security parameter λ, for any admissible PPT *stateful* adversary $\mathcal{A}$, the experiments $\mathsf{INDFE}^{0,\mathcal{A}}(\lambda)$ and $\mathsf{INDFE}^{1,\mathcal{A}}(\lambda)$ are computationally indistinguishable to $\mathcal{A}$, where $\mathsf{INDFE}^{b,\mathcal{A}}(\lambda)$ is defined as follows:

Experiment $\mathsf{INDFE}^{b,\mathcal{A}}(\lambda)$

- $(\{\mathsf{ek}_i\}_{i\in[t]}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^t, 1^L)$,

- $\{\mathbf{X}_q^{(0)}, \mathbf{X}_q^{(1)}\}_q \leftarrow \mathcal{A}(1^\lambda, \{\mathsf{ek}_i\}_{i\in I})$ where for $b \in \{0,1\}$, $\mathbf{X}_q^{(b)} := (\mathbf{x}_{q,1}^{(b)}, \ldots \mathbf{x}_{q,t}^{(b)}) \in \left(\{0,1\}^L\right)^t$
- for all q, let $\mathbf{CT}_q := \left(\mathsf{Enc}(\mathsf{ek}_1, \mathbf{x}_{q,1}^{(b)}), \ldots, \mathsf{Enc}(\mathsf{ek}_t, \mathbf{x}_{q,t}^{(b)})\right)$
- $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\{\mathbf{CT}_q\}_q)$

In the above, the stateful adversary $\mathcal{A}$ can pass state to later invocations, and we do not write its state explicitly. We refer to $(\mathbf{X}_q^{(0)}, \mathbf{X}_q^{(1)})$ as the q-th *challenge pair*.

The adversary $\mathcal{A}$ is said to be *admissible*, if the functions f submitted to the $\mathbf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must be from the family $\mathcal{F}_\lambda$, and moreover, the submitted challenge messages cannot be trivially distinguished by evaluating functional keys. Formally, for any key-generation query f made by $\mathcal{A}$, any challenge pair $(\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_t^{(0)})$ and $(\mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_t^{(1)})$, we require $f(\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_t^{(0)}) = f(\mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_t^{(1)})$ if for any $i \in [t]$, either of the followings holds:

- $\mathbf{x}_i^{(0)} = \mathbf{x}_i^{(1)}$ and $i \in I$.
- for some index q, $\mathbf{x}_i^{(0)} = \mathbf{x}_{q,i}^{(0)}$, and $\mathbf{x}_i^{(1)} = \mathbf{x}_{q,i}^{(1)}$, where $\{\mathbf{x}_{q,i}^{(0)}\}_{i\in[t]}$ and $\{\mathbf{x}_{q,i}^{(1)}\}_{i\in[t]}$ correspond to the q-th challenge pair.

Depending on the choice of I, the above gives a unified definition that captures all of the following special cases:

1. *Secret-key setting*: $I = \emptyset$.
2. *Public-key setting*: $I = [t]$.
3. *More general case*: I can be any subset of $[t]$.

Definition 3 ($\mathcal{F}_\lambda$ admits a succinct equivalence proof)**.** Fix L and t to be polynomially bounded functions in λ. We say that the family $\mathcal{F}_\lambda$ of t-ary functions on L-bit inputs *admits an equivalence proof of size at most* $N(\lambda)$, iff for any λ, for any $f \in \mathcal{F}_\lambda$, for any subset $J \subseteq [t]$, for any $\mathbf{x}_J \in (\{0,1\}^L)^{|J|}$ and $\mathbf{x}'_J \in (\{0,1\}^L)^{|J|}$ such that $f(\mathbf{x}_J, \cdot) = f(\mathbf{x}'_J, \cdot)$, there exists an $\mathcal{EF}$ proof of size at most N that proves the functional equivalence of $f(\mathbf{x}_J, \cdot)$ and $f(\mathbf{x}'_J, \cdot)$.

In the above, we use the notation $f(\mathbf{x}_J, \cdot)$ to denote the $(t - |J|)$-ary function derived from f by restricting the subset J of indices to $\mathbf{x}_J$.

6.2 Building Blocks

We require following building blocks in our construction:

- polynomially secure public-key bit-encryption $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ instantiated by Regev's construction [Reg05] with output length $L_{\mathsf{PKE}} = \mathsf{poly}(\lambda)$ per bit.
- $2^{\lambda^{\epsilon_{\mathsf{COM}}}}$-secure, perfectly binding non-interactive bit-commitment $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ instantiated by Regev's construction [Reg05] with security parameter $\lambda_{\mathsf{COM}} = ((t+2)L_{\mathsf{PKE}})^{1/\epsilon_{\mathsf{COM}}}$ and commitment randomness length $r_{\mathsf{COM}} = \mathsf{poly}(\lambda)$.
- $2^{\lambda^{\epsilon_{\mathsf{PRF}}}}$-secure puncturable pseudorandom functions $\mathsf{PRF} : \{0,1\}^{\lambda_{\mathsf{PRF}}} \times \{0,1\}^{3L_{\mathsf{PKE}}} \to \{0,1\}^{r_{\mathsf{COM}}}$ with security parameter $\lambda_{\mathsf{PRF}} = ((t+2)L_{\mathsf{PKE}})^{1/\epsilon_{\mathsf{PRF}}}$.
- $2^{\lambda^{\epsilon_{\mathsf{LiO}}}}$-secure quasi-linear locally indistinguishable obfuscator LiO for $O(\log n)$-equivalent circuits (described in Section 5) with security parameter $\lambda_{\mathsf{LiO}} = ((t+2)L_{\mathsf{PKE}})^{1/\epsilon_{\mathsf{LiO}}}$.

6.3 Vector Operations for Bit-by-Bit Encryption

We will use vector operations because in our scheme we will encrypt an L-bit message bit-by-bit. Our usage of vector operations will be restricted to performing L-fold SIMD operations arising from the message length L.

General notation for vector operations. In general, for any binary function op, given a scalar x and a vector $\mathbf{y} := (y_1, \ldots, y_L)$, we use the following notation:

$$\mathsf{op}(x, \mathbf{y}) := (\mathsf{op}(x, y_1), \ldots, \mathsf{op}(x, y_L)) .$$

For two vectors $\mathbf{x} := (x_1, \ldots, x_L)$ and $\mathbf{y} := (y_1, \ldots, y_L)$, we define

$$\mathsf{op}(\mathbf{x}, \mathbf{y}) := (\mathsf{op}(x_1, y_1), \ldots, \mathsf{op}(x_L, y_L)) .$$

Examples. We will be use the following examples of vector operations in our scheme description:

1. *Bit-by-bit encryption.* We will use bit-by-bit encryption to encrypt an L-bit message. All L bits will share the same encryption keys. Given some PKE key pair $(\mathsf{pk}, \mathsf{sk})$ of a bit-encryption scheme, we use the following vector notation:
   - $\mathbf{c} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{x})$: given an L-bit message $\mathbf{x} = (x_1, \ldots, x_L) \in \{0,1\}^L$, we use this notation to mean for $\ell \in [L]$, let $c_\ell := \mathsf{PKE.Enc}(\mathsf{pk}, x_\ell)$, and output $\mathbf{c} := (c_1, \ldots, c_L)$.
   - $\mathbf{x} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \mathbf{c})$: given $\mathbf{c} := (c_1, \ldots, c_L)$, we use this notation to mean for $\ell \in [L]$, let $x_\ell := \mathsf{PKE.Dec}(\mathsf{sk}, c_\ell)$, and output $\mathbf{x} := (x_1, \ldots, x_L)$.
2. *PRF and commitment vector operations.* Given L PRF keys $\mathbf{k} := (k_1, \ldots, k_L)$, and an L messages $\mathbf{x} := (x_1, \ldots, x_L) \in \{\{0,1\}^{L_{\mathsf{PKE}}}\}^L$ each of $3L_{\mathsf{PKE}}$ bits long,

   $$\mathsf{PRF}(\mathbf{k}, \mathbf{x}) := (\mathsf{PRF}(k_1, x_1), \ldots, \mathsf{PRF}(k_L, x_L))$$

   Given a vector $\mathbf{r} := (r_1, \ldots, r_L)$ of random strings of appropriate length,

   $$\mathsf{COM.Commit}(\mathsf{crs}, 0; \mathbf{r}) := (\mathsf{COM.Commit}(\mathsf{crs}, 0; r_1), \ldots, \mathsf{COM.Commit}(\mathsf{crs}, 0; r_L)) ,$$

3. *Other operators.* Given a scalar v, vectors $\mathbf{x} := (x_1, \ldots, x_L)$ and $\mathbf{y} := (y_1, \ldots, y_L)$,

   $$v \| \mathbf{x} \| \mathbf{y} := (v \| x_1 \| y_1, \ldots, v \| x_L \| y_L)$$

6.4 Construction

We present our MIFE construction in this section. Henceforth, we will use the boldface notation $\mathbf{x}_i \in \{0,1\}^L$ to denote an L-bit message to make explicit that we treat it as a length-L vector, since we will use the vector notation of Section 6.3.

Parameters. Our scheme uses the following parameters where the CNIZK_i and the G_f circuits will be defined later in the scheme.

- *For obfuscating the* CNIZK_i *circuit:*
  - let $N_{\text{circ}} = tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}) = L \cdot \mathsf{poly}'(\lambda, t)$;

- let $N_{\text{proof}} = tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = L \cdot \mathsf{poly}'(\lambda, t)$; and
- let $s = O(\log(N_{\text{circ}} + N_{\text{proof}}))$.

- *For obfuscating the G_f circuit:*
  - let $N'_{\text{circ}} = N^{\mathcal{F}}_{\text{circ}} + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = N^{\mathcal{F}}_{\text{circ}} + L \cdot \mathsf{poly}'(\lambda, t)$ where we use the notation $N^{\mathcal{F}}_{\text{circ}}(\lambda)$ to denote an upper bound on the circuit size of any function $f \in \mathcal{F}_\lambda$;
  - for the single-input and secret-key settings, let $N'_{\text{proof}} = O(N^{\mathcal{F}}_{\text{circ}} + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}})) = O(N^{\mathcal{F}}_{\text{circ}}) + L \cdot \mathsf{poly}'(\lambda, t)$;
  - for the setting where $\mathcal{F}_\lambda$ admits a size-$N^{\mathcal{F}}_{\text{proof}}(\lambda)$ equivalence proof, let $N'_{\text{proof}} = O(N^{\mathcal{F}}_{\text{circ}} + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) + N^{\mathcal{F}}_{\text{proof}}) = O(N^{\mathcal{F}}_{\text{circ}} + N^{\mathcal{F}}_{\text{proof}}) + L \cdot \mathsf{poly}'(\lambda, t)$;
  - let $s' = O(\log(N'_{\text{circ}} + N'_{\text{proof}}))$.

Construction. We now describe our MIFE construction.

- $\mathsf{Gen}(1^\lambda, 1^t, 1^L)$: Run the corresponding setup algorithms to generate the following keys:
  - for each $i \in [t]$, $b \in \{0,1\}$: let $(\mathsf{pk}_i^b, \mathsf{sk}_i^b) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$;
  - for $i \in [t]$, let $(\mathsf{pk}_i^{\mathsf{ext}}, \mathsf{sk}_i^{\mathsf{ext}}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$;
  - for $i \in [t]$, sample L puncturable PRF keys $\mathbf{k}_i := (k_{i,1}, \ldots, k_{i,L})$ each of length λ_{PRF};
  - let $\mathsf{crs} \leftarrow \mathsf{COM.Gen}(1^{\lambda_{\mathsf{COM}}})$; and
  - for $i \in [t]$, let $\widetilde{\mathsf{CNIZK}}_i \leftarrow \mathsf{LiO}(1^\lambda, s, \mathsf{Pad}(\mathsf{CNIZK}_i, N_{\text{circ}}, N_{\text{proof}}))$ where CNIZK_i is the following circuit:

$\mathsf{CNIZK}_i(v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, r_i^{\mathsf{ext}}, \mathbf{x}, \mathbf{r}_i^0, \mathbf{r}_i^1)$

- Abort outputting $\perp$ if any of the following checks fail:
  - v_i is an encryption of 0, i.e.,
$$\mathsf{PKE.Enc}(\mathsf{pk}_i^{\mathsf{ext}}, 0; r_i^{\mathsf{ext}}) \stackrel{?}{=} v_i$$
  - $\mathbf{c}_i^0$ and $\mathbf{c}_i^1$ are encryptions of same message $\mathbf{x} \in \{0,1\}^L$, i.e.,
$$\forall b \in \{0,1\} : \ \mathsf{PKE.Enc}(\mathsf{pk}_i^b, \mathbf{x}; \mathbf{r}_i^b) \stackrel{?}{=} \mathbf{c}_i^b,$$
    where $\mathbf{r}_i^b := (r_{i,1}^b, \ldots, r_{i,L}^b)$ denotes the random coins consumed by the L encryption instances.
- Output a NIZK proof $\boldsymbol{\pi}_i := \mathsf{PRF}(\mathbf{k}_i, v_i \| \mathbf{c}_i^0 \| \mathbf{c}_i^1)$.

// See Section 6.3 regarding vector notation used for $\mathsf{PKE.Enc}$, $\stackrel{?}{=}$ *and* $\|$ *functions.*

Figure 4: Circuit CNIZK_i.

Then, output encryption keys $\mathsf{ek}_i = \{\mathsf{pk}_i^{\mathsf{ext}}, \mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{\mathsf{CNIZK}}_i\}$ for $i \in [t]$ and a master secret key $\mathsf{msk} = \{\mathsf{sk}_i^{\mathsf{ext}}, \mathsf{sk}_i^0, \mathsf{sk}_i^1, \mathbf{k}_i\}_{i \in [t]} \cup \{\mathsf{crs}\}$.

In the above each $\widetilde{\mathsf{CNIZK}}_i$ is the NIZK construction due to Sahai and Waters [SW14] but now using our LiO instead.

- $\mathsf{Enc}(\mathsf{ek}_i, \mathbf{x}_i)$: Uniformly sample randomness r_i^{ext} and $\mathbf{r}_i^b := (r_{i,1}^b, \ldots, r_{i,L}^b)$ for $b \in \{0,1\}$, then compute a single label

$$v_i \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^{\mathsf{ext}}, 0; r_i^{\mathsf{ext}}),$$

two ciphertext vectors

$$\mathbf{c}_i^b \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^b, \mathbf{x}_i; \mathbf{r}_i^b), \quad \forall b \in \{0,1\},$$

and an auxiliary proof vector

$$\boldsymbol{\pi}_i \leftarrow \widetilde{\mathsf{CNIZK}}_i(v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, r_i^{\mathsf{ext}}, x_i, \mathbf{r}_i^0, \mathbf{r}_i^1).$$

Output the resulting ciphertext $\mathsf{ct}_i := (v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i)$.

- $\mathsf{KeyGen}(\mathsf{msk}, f)$: Output the obfuscated program

$$\mathsf{sk}_f \leftarrow \mathsf{LiO}(1^\lambda, s', \mathsf{Pad}(G_f, N'_{\mathrm{circ}}, N'_{\mathrm{proof}}))$$

where G_f is the circuit described in Figure 5.

$G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^0 = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, \mathbf{c}_i^0)$ for each $i \in [t]$.
- For each $i \in [t]$, check the validity of $\boldsymbol{\pi}_i$ and abort if false:

$$\mathsf{COM.Commit}\left(\mathsf{crs}, 0; \boldsymbol{\pi}_i\right) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\mathbf{k}_i, v_i \| \mathbf{c}_i^0 \| \mathbf{c}_i^1)\right)$$

- Output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

// See Section 6.3 regarding vector notation used for $\mathsf{PKE.Dec}$, $\mathsf{COM.Commit}$, *and* $\stackrel{?}{=}$ *functions.*

Figure 5: Circuit G_f.

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_t)$: parse each $\mathsf{ct}_i := (v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i)$, then output $\mathsf{sk}_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$.

6.5 Security Proofs

We will use a sequence of hybrids to prove the security of our MIFE scheme. In the first hybrid the challenger encrypts $\{\mathbf{X}_q^{(0)}\}_q = \{\mathbf{x}_{q,1}^{(0)}, \ldots, \mathbf{x}_{q,t}^{(0)}\}_q$, and in the last hybrid the challenger encrypts $\{\mathbf{X}_q^{(1)}\}_q = \{\mathbf{x}_{q,1}^{(1)}, \ldots, \mathbf{x}_{q,t}^{(1)}\}_q$.

Experiment Real_0. This experiment is identically distributed as $\mathsf{INDFE}^{0,\mathcal{A}}(\lambda)$. However, upon encryption queries, instead of computing the NIZK proofs using the obfuscated $\widetilde{\mathsf{CNIZK}}_i$ programs, the challenger instead computes the NIZK proofs directly using the PRF and the master secret key. Specifically, for the q-th challenge pair $(\{\mathbf{x}_{q,i}^{(0)}\}_{i\in[t]}, \{\mathbf{x}_{q,i}^{(1)}\}_{i\in[t]})$, the challenger responds with

$$\begin{aligned} v_{q,i} &\leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^{\mathsf{ext}}, 0) \\ \mathbf{c}_{q,i}^{(0)} &\leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^0, \mathbf{x}_{q,i}^{(0)}) \\ \mathbf{c}_{q,i}^{(1)} &\leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^1, \mathbf{x}_{q,i}^{(0)}) \end{aligned}$$

and corresponding NIZK proof $\boldsymbol{\pi}_{q,i} = \mathsf{PRF}(\mathbf{k}_i, v_{q,i} \| \mathbf{c}_{q,i}^{(0)} \| \mathbf{c}_{q,i}^{(1)})$.

Experiment Real_0^*. This hybrid is the same as Real_0, except that the challenger computes $v_{q,i}$ as encryption of 1:

$$v_{q,i} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^{\mathsf{ext}}, 1),$$

and changes the second component of the ciphertexts to encrypt $\mathbf{x}_{q,i}^{(1)}$ instead:

$$\mathbf{c}_{q,i}^{(0)} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^0, \mathbf{x}_{q,i}^{(0)})$$
$$\mathbf{c}_{q,i}^{(1)} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i^1, \mathbf{x}_{q,i}^{(1)}).$$

Further, as before, it computes the NIZK proof as $\boldsymbol{\pi}_{q,i} = \mathsf{PRF}(\mathbf{k}_i, v_{q,i} \| \mathbf{c}_{q,i}^{(0)} \| \mathbf{c}_{q,i}^{(1)})$.

Since the secret keys $\mathsf{sk}_i^{\mathsf{ext}}$ and sk_i^1 are never used anywhere, it follows from the semantic security of PKE that no PPT adversary can distinguish Real_0 and Real_0^* with more than negligible probability.

Observe that due to the semantic security of the PKE scheme, it holds that except with negligible probability, the labels v_i of all challenge queries are distinct. Therefore, in Real_0^* and all future hybrids, we may assume that the v_i's for all challenge queries are distinct — in case not, the experiment can simply abort outputting $\perp$, and this introduces only a negligibly small statistical distance.

Experiment Hyb. This hybrid is the same as Real_0^*, except that each functional key sk_f decrypts the ciphertext $\mathbf{c}_i^1$ instead of $\mathbf{c}_i^0$ for $i \in [t]$. To show the indistinguishability of Hyb and Real_0^*, we need to go through a sequence of intermediate hybrids.

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}$. This hybrid is indexed by a label $\widehat{\mathbf{v}} = (\widehat{v}_1, \ldots, \widehat{v}_t)$, and the label $\widehat{\mathbf{v}}$ (when viewed as an integer) is upper bounded by $2^{tL_{\mathsf{PKE}}}$. If G_f's input label $\mathbf{v} = (v_1, \ldots, v_t) < \widehat{\mathbf{v}}$ then for each functional key sk_f, the inputs of f are decrypted from $\mathbf{c}_i^1$ instead of $\mathbf{c}_i^0$:

Modified $G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^b = \mathsf{PKE.Dec}(\mathsf{sk}_i^b, \mathbf{c}_i^b)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t]$, check the validity of $\boldsymbol{\pi}_i$ and abort if false:

$$\mathsf{COM.Commit}\left(\mathsf{crs}, 0; \boldsymbol{\pi}_i\right) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\mathbf{k}_i, v_i \| \mathbf{c}_i^0 \| \mathbf{c}_i^1)\right)$$

- If $\mathbf{v} < \widehat{\mathbf{v}}$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

Clearly Hyb_0 is functionally equivalent to Real_0^* while $\mathsf{Hyb}_{2^{tL_{\mathsf{PKE}}}}$ is functionally equivalent to Hyb with an $\mathcal{EF}$ proof of size at most $O(N_{\mathrm{circ}}^{\mathcal{F}} + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}})) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$. Therefore, it suffices to prove that

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}+1}) = 1]\right| = 2^{-tL_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Observe that $\mathsf{Hyb}_{\widehat{\mathbf{v}}}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}+1}$ can only differ by outputs on $\mathbf{v} = \widehat{\mathbf{v}} = (\widehat{v}_1, \ldots, \widehat{v}_t)$. Recall that all the encryption queries are made upfront before the **KeyGen** queries. Let $\{v_{q,i}, \mathbf{c}_{q,i}^{(0)}, \mathbf{c}_{q,i}^{(1)}\}_{i\in[t]}$

be the q-th challenge ciphertext given to $\mathcal{A}$, let $\{\mathbf{x}_{q,i}^{(0)}, \mathbf{x}_{q,i}^{(1)}\}_{i\in[t]}$ be the corresponding q-th plaintext pair queried. We can define $(\widehat{\mathbf{c}}_i^0, \widehat{\mathbf{c}}_i^1)$ as follows:

$$(\widehat{\mathbf{c}}_i^0, \widehat{\mathbf{c}}_i^1) = \begin{cases} \mathbf{c}_{q,i}^{(0)}, \mathbf{c}_{q,i}^{(1)} & \text{if } \widehat{v}_i = v_{q,i} \text{ for some index } q \\ \bot, \bot & \text{otherwise} \end{cases} \tag{1}$$

Similarly, we define $(\widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1)$ as follows:

$$(\widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1) = \begin{cases} \mathbf{x}_{q,i}^{(0)}, \mathbf{x}_{q,i}^{(1)} & \text{if } \widehat{v}_i = v_{q,i} \text{ for some index } q \\ \bot, \bot & \text{otherwise} \end{cases} \tag{2}$$

As we remarked at the end of Real_0^*, we may assume that if the experiment does not abort, then all the v_i labels are distinct, and thus the above definition of $\widehat{\mathbf{c}}_i^0, \widehat{\mathbf{c}}_i^1, \widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1$ does not have ambiguity.

Now we use a sequence of inner hybrids to show the indistinguishability of $\mathsf{Hyb}_{\widehat{\mathbf{v}}}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}+1}$.

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^c$. This hybrid is indexed by a pair of PKE ciphertexts $c = (c^0, c^1)$ corresponding to the encryption of a single bit (notice that c is bounded by $2^{2L_{\mathsf{PKE}}}$). $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^c$ is almost the same as $\mathsf{Hyb}_{\widehat{v}}$, except that we add an extra validation check for any input ciphertext where $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) < c$ in the G_f circuit to be obfuscated when computing a functional key sk_f:

Modified $G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^b = \mathsf{PKE.Dec}(\mathsf{sk}_i^b, \mathbf{c}_i^b)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t]$: check the following and abort if false:
  - Check the validity of $\boldsymbol{\pi}_i$:

$$\mathsf{COM.Commit}\left(\mathsf{crs}, 0; \boldsymbol{\pi}_i\right) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\mathbf{k}_i, v_i \| \mathbf{c}_i^0 \| \mathbf{c}_i^1)\right)$$

  - For each $\ell \in [L]$, if $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) < c$, check if at least one of following conditions holds where $\widehat{\mathbf{c}}_i^0$ and $\widehat{\mathbf{c}}_i^1$ are defined in Equation (1):
    - $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, $i \in I$ and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$.
    - $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$.
- If $\mathbf{v} < \widehat{\mathbf{v}}$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

Since the extra validation check is not executed when $c = 0$, $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^0$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}}$ are indistinguishable due to the security of underlying LiO, and the fact that the two circuits have a trivial equivalence proof in $\mathcal{EF}$ of size $O(N_{\mathrm{circ}}^{\mathcal{F}} + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathrm{COM}})) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$. We now argue that

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^c) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c+1}) = 1]\right| = 2^{-(t+2)L_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

through a sequence of hybrids.

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,0}$. This hybrid is same as $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^c$.

Experiment $\mathsf{Hyb}^{c,1}_{\widehat{\mathbf{v}}}$. This hybrid is almost same as $\mathsf{Hyb}^{c,0}_{\widehat{\mathbf{v}}}$, except that for $i \in I$, if $\mathsf{PKE.Dec}(\mathsf{sk}^{\mathsf{ext}}_i, \widehat{v}_i) \neq 0$ or $\mathsf{PKE.Dec}(\mathsf{sk}^0_i, c^0) \neq \mathsf{PKE.Dec}(\mathsf{sk}^1_i, c^1)$, then we use the following modified CNIZK_i circuit when computing ek_i.

Modified $\mathsf{CNIZK}_i(v_i, \mathbf{c}^0_i, \mathbf{c}^1_i, r^{\mathsf{ext}}_i, \mathbf{x}, \mathbf{r}^0_i, \mathbf{r}^1_i)$

- Abort outputting $\perp$ if either of the following checks fails: (▲)
  - $\mathsf{PKE.Enc}(\mathsf{pk}^{\mathsf{ext}}_i, 0; r^{\mathsf{ext}}_i) = v_i$;
  - $\forall b \in \{0,1\} : \mathsf{PKE.Enc}(\mathsf{pk}^b_i, \mathbf{x}; \mathbf{r}^b_i) = \mathbf{c}^b_i$.
- Output a NIZK proof $\boldsymbol{\pi}_i$ consisting of $(\boldsymbol{\pi}_i)_\ell$ for each bit $\ell \in [L]$ defined as follows: let

$$(\boldsymbol{\pi}_i)_\ell = \mathsf{PRF}\left(\underbrace{k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}}_{\text{hard-wired}}, v_i \| (\mathbf{c}^0_i)_\ell \| (\mathbf{c}^1_i)_\ell\right)$$

  where the hard-wired value $k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}$ denotes the PRF key $k_{i,\ell}$ punctured at the point $\widehat{v}_i \| c^0 \| c^1$.

Further, under the same condition $\mathsf{PKE.Dec}(\mathsf{sk}^{\mathsf{ext}}_i, \widehat{v}_i) \neq 0$ or $\mathsf{PKE.Dec}(\mathsf{sk}^0_i, c^0) \neq \mathsf{PKE.Dec}(\mathsf{sk}^1_i, c^1)$, whenever the challenger needs to compute the NIZK proofs for answering encryption queries for $i \in I$, it uses the punctured key $k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}$ if the ciphertext is not equal to $(\widehat{v}_i \| c^0 \| c^1)$; else, it uses the hard-wired value $\mathsf{PRF}(k_{i,\ell}, \widehat{v}_i \| c^0 \| c^1)$ as the proof.

Observe that the CNIZK_i checks the conditions (▲) on v_i, $\{(\mathbf{c}^0_i)_\ell, (\mathbf{c}^1_i)_\ell\}_{\ell \in [L]}$. Due to the correctness of the PKE scheme, we know that the same conditions cannot hold for $\widehat{v}_i$, c^0 and c^1. Therfore, if the CNIZK_i circuit actually reaches the step of generating the proof $\boldsymbol{\pi}_i$, then the input to the PRF cannot be $\widehat{v}_i || c^0 || c^1$. Due to the puncturing correctness of the PRF, the modified CNIZK_i circuit must be functionally equivalent to the original one. Moreover, by Lemma A.1 and Lemma A.2, the equivalence of CNIZK_i and the above modified one can be proven in $\mathcal{EF}$ with proof length at most $tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}) = L \cdot \mathsf{poly}'(\lambda, t)$. So the indistinguishability of $\mathsf{Hyb}^{c,1}_{\widehat{\mathbf{v}}}$ and $\mathsf{Hyb}^{c,0}_{\widehat{\mathbf{v}}}$ follows from the security of underlying LiO, that is,

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}^{c,1}_{\widehat{\mathbf{v}}}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}^{c,2}_{\widehat{\mathbf{v}}}) = 1]\right| = 2^{-\lambda_{\mathsf{LiO}}\epsilon_{\mathsf{LiO}}}\mathsf{negl}(\lambda) = 2^{-(t+2)L_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Experiment $\mathsf{Hyb}^{c,2}_{\widehat{\mathbf{v}}}$. This hybrid is almost same as $\mathsf{Hyb}^{c,1}_{\widehat{\mathbf{v}}}$, except that we modify the G_f circuit when computing a functional key sk_f as:

Modified $G_f(\{v_i, \mathbf{c}^0_i, \mathbf{c}^1_i, \boldsymbol{\pi}_i\}_{i \in [t]})$

- Decrypt $\mathbf{x}^b_i = \mathsf{PKE.Dec}(\mathsf{sk}^b_i, \mathbf{c}^b_i)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t], \ell \in [L]$:
  - If $v_i \neq \widehat{v}_i$ or $((\mathbf{c}^0_i)_\ell, (\mathbf{c}^1_i)_\ell) \neq c$, check the validity of $(\boldsymbol{\pi}_i)_\ell$ using the punctured PRF key:

$$\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\underbrace{k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}}_{\text{hard-wired}}, v_i \| (\mathbf{c}^0_i)_\ell \| (\mathbf{c}^1_i)_\ell)\right)$$

– Else if $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = c$, check the following:

$$\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \underbrace{\mathsf{PRF}(k_{i,\ell}, \widehat{v}_i \| c^0 \| c^1)}_{\text{hard-wired}}\right) \qquad (\star)$$

– If $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) < c$, check if at least one of following conditions holds:

* $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, $i \in I$ and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$.
* $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$.

- If $\mathbf{v} < \widehat{\mathbf{v}}$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

Observe that the modification does not the affect the functionality of the circuit, and there exists some $\mathcal{EF}$ proof of equivalence of size $O(N_{\mathrm{circ}}^{\mathcal{F}}) + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$ due to Lemma A.1. Therefore, by the security of underlying LiO, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,1}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,2}) = 1]\right| = 2^{-\lambda_{\mathsf{LiO}}\epsilon_{\mathsf{LiO}}}\mathsf{negl}(\lambda) = 2^{-(t+2)L_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}$. This hybrid is almost same as $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,2}$, except that we further modify the G_f circuit when computing a functional key sk_f as follows:

Modified $G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^b = \mathsf{PKE.Dec}(\mathsf{sk}_i^b, \mathbf{c}_i^b)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t], \ell \in [L]$:

– If $v_i \neq \widehat{v}_i$ or $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) \neq c$, check validity of $(\boldsymbol{\pi}_i)_\ell$ using punctured PRF key:

$$\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\underbrace{k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}}_{\text{hard-wired}}, v_i \| (\mathbf{c}_i^0)_\ell \| (\mathbf{c}_i^1)_\ell)\right)$$

– Else, we have $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = c$.

* If $c \neq ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$ and meanwhile not all of the following hold: $i \in I$, $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$, then check the following:

$$\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{com}_{i,\ell}^* \qquad (\diamond)$$

where each $\mathsf{com}_{i,\ell}^*$ is chosen to be a commitment of 1 using true random coins.

* Else, check $\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \underbrace{\mathsf{PRF}(k_{i,\ell}, \widehat{v}_i \| c^0 \| c^1)}_{\text{hard-wired}}\right)$

– If $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) < c$, check if at least one of following conditions holds:

* $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, $i \in I$ and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$.

$*$ $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$.

- If $\mathbf{v} < \widehat{\mathbf{v}}$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

The difference between $\mathsf{Hyb}_{\widehat{v}}^{c,2}$ and $\mathsf{Hyb}_{\widehat{v}}^{c,3}$ is that we changed the branch marked $(\star)$ in $\mathsf{Hyb}_{\widehat{v}}^{c,2}$ into two separate cases in $\mathsf{Hyb}_{\widehat{v}}^{c,3}$. To argue the indistinguishability of these experiments, observe that in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,2}$,

- If either $i \notin I$ or $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, \widehat{v}_i) \neq 0$, or $\mathsf{PKE.Dec}(\mathsf{sk}_i^0, c^0) \neq \mathsf{PKE.Dec}(\mathsf{sk}_i^1, c^1)$, then the key $k_{i,\ell}$ is not needed in the experiment since either $i \notin I$ is honest and the challenger need not give $\mathcal{A}$ the obfuscated $\widetilde{\mathsf{CNIZK}}_i$; or the CNIZK_i circuit uses the punctured key $k_{i,\ell}\{\widehat{v}_i \| c^0 \| c^1\}$ instead. Recall also that the challenger always uses the punctured key $k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}$ or a hard-wired value $\mathsf{PRF}(k_{i,\ell}, \widehat{v}_i \| c^0 \| c^1)$ for answering encryption queries, so the key $k_{i,\ell}$ is also never used in answering encryption queries.
- If additionally, $c \neq ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$, then the PRF evaluation outcome $\mathsf{PRF}(k_{i,\ell}, \widehat{v}_i \| c^0 \| c^1)$ is never disclosed to the adversary $\mathcal{A}$ in any encryption query. The only place this evaluation outcome is used is in the $(\star)$ line of the (modified) G_f circuit in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,2}$.

Due to the security of the puncturable PRF, under these conditions, we can replace the right-hand-side of line $(\star)$ with a commitment of 0 under fresh random coins. Due to the security of the commitment, we then replace each commitment of 0 with a commitment of 1 which leads to $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}$. Thus, we have

$$\begin{aligned}
&\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,2}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}) = 1]\right| \\
\leq & 2^{-\lambda_{\mathsf{LiO}}{}^{\epsilon^{\mathsf{LiO}}}} \mathsf{negl}(\lambda) + 2^{-\lambda_{\mathsf{PRF}}{}^{\epsilon_{\mathsf{PRF}}}} \mathsf{negl}(\lambda) + 2^{-\lambda_{\mathsf{COM}}{}^{\epsilon_{\mathsf{COM}}}} \mathsf{negl}(\lambda) \\
= & 2^{-(t+2)L_{\mathsf{PKE}}} \mathsf{negl}(\lambda).
\end{aligned}$$

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,4}$. This hybrid is almost same as $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}$, except that we further modify the G_f circuit when computing a functional key sk_f as follows:

Modified $G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^b = \mathsf{PKE.Dec}(\mathsf{sk}_i^b, \mathbf{c}_i^b)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t], \ell \in [L]$:
  - If $v_i \neq \widehat{v}_i$ or $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) \neq c$, check that

    $$\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(k_{i,\ell}\{(\widehat{v}_i \| c^0 \| c^1)\}, v_i \| (\mathbf{c}_i^0)_\ell \| (\mathbf{c}_i^1)_\ell)\right)$$

  - Else, we have $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = c$.
    - Check that either $c = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$, or all of the following hold: $i \in I$, $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$.
    - Check $\mathsf{COM.Commit}(\mathsf{crs}, 0; (\boldsymbol{\pi}_i)_\ell) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(k_{i,\ell}, \widehat{v}_i \| c^0 \| c^1)\right)$
  - If $v_i = \widehat{v}_i$ and $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) < c$, check if at least one of following conditions holds:

$*$ $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, $i \in I$ and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$.
$*$ $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$.

- If $\mathbf{v} < \widehat{\mathbf{v}}$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

Since the commitment is perfectly binding, in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}$, the check in line ($\diamond$) never succeeds. Therefore, the modified G_f circuits of $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,4}$ are equivalent, and they enjoy an $\mathcal{EF}$ equivalence proof of size at most $O(N_{\mathrm{circ}}^{\mathcal{F}}) + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L\mathsf{poly}'(\lambda, t)$ due to Lemma A.2, assuming we use Regev's encryption [Reg05] to instantiate the commitment scheme. Now, by the security of LiO, we have that

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,3}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,4}) = 1]\right| = 2^{-\lambda_{\mathsf{LiO}}\epsilon_{\mathsf{LiO}}}\mathsf{negl}(\lambda) = 2^{-(t+2)L_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

We next argue that $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,4}$ is indistinguishable from $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c+1}$. Due to the puncturing correctness of the PRF, the modified G_f circuits in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,4}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c+1}$ are equivalent. Further, due to Lemma A.1, there is an $\mathcal{EF}$ proof of equivalence of size at most $O(N_{\mathrm{circ}}^{\mathcal{F}}) + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$. Therefore, by the security of the underlying LiO, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c,4}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{c+1}) = 1]\right| = 2^{-\lambda_{\mathsf{LiO}}\epsilon_{\mathsf{LiO}}}\mathsf{negl}(\lambda) = 2^{-(t+2)L_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}$. This experiment is otherwise identical to $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}$ except that the G_f circuit is further modified to the following:

Modified $G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^b = \mathsf{PKE.Dec}(\mathsf{sk}_i^b, \mathbf{c}_i^b)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t]$, check the validity of $\boldsymbol{\pi}_i$:

$$\mathsf{COM.Commit}\left(\mathsf{crs}, 0; \boldsymbol{\pi}_i\right) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\mathbf{k}_i, v_i \| \mathbf{c}_i^0 \| \mathbf{c}_i^1)\right)$$

- For every $i \in [t]$: if $v_i = \widehat{v}_i$, check that
  - if $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, \widehat{v}_i) = 0$, then $i \in I$ and $\mathbf{x}_i^0 = \mathbf{x}_i^1$,
  - else, $(\mathbf{c}_i^0, \mathbf{c}_i^1) = (\widehat{\mathbf{c}}_i^0, \widehat{\mathbf{c}}_i^1)$ and $(\mathbf{x}_i^0, \mathbf{x}_i^1) = (\widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1)$ where $(\widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1)$ are defined in Equation (2).
- If $\mathbf{v} < \widehat{\mathbf{v}}$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

Recall that in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}$, if $v_i = \widehat{v}_i$, we check that for each $\ell \in [L]$, either $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$ (henceforth called the first condition), or all of the following hold: $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, $i \in I$ and $(\mathbf{x}_i^0)_\ell = (\mathbf{x}_i^1)_\ell$ (henceforth called the second condition). The difference between $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}$ is that in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}$, we check that for each coordinate $\ell \in [L]$, either the first condition or the second condition holds, whereas in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}$, we check that the first or the second condition simultaneously holds for all coordinates $\ell \in [L]$.

Note that if $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, \widehat{v}_i) = 0$, then $(\widehat{\mathbf{c}}_i^0)_\ell = (\widehat{\mathbf{c}}_i^1)_\ell = \perp$ for every $\ell \in [L]$. Therefore, assume that for each coordinate, either the first or second condition holds like in $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}$. Then, it must be that

- if $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, v_i) = 0$, then the second condition simultaneously holds for all $\ell \in [L]$;
- otherwise, the first condition simultaneously holds for all $\ell \in [L]$.

This is why we can switch from $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}$ to $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}$. Proving the equivalence of the modified G_f circuits of the two experiments would require proving the correctness of the PKE scheme, since we need to prove that ciphertext equality $((\mathbf{c}_i^0)_\ell, (\mathbf{c}_i^1)_\ell) = ((\widehat{\mathbf{c}}_i^0)_\ell, (\widehat{\mathbf{c}}_i^1)_\ell)$ implies decrypted plaintext equality $((\mathbf{x}_i^0)_\ell, (\mathbf{x}_i^1)_\ell) = ((\widehat{\mathbf{x}}_i^0)_\ell, (\widehat{\mathbf{x}}_i^1)_\ell)$. Due to Lemma A.2, proving the equivalence of the modified G_f circuits of the two experiments would involve an $\mathcal{EF}$ proof of size $O(N_{\mathrm{circ}}^{\mathcal{F}}) + tL\mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$. Hence, by the security of the underlying LiO, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{2^{2L_{\mathsf{PKE}}}}) = 1]\right| = 2^{-\lambda_{\mathsf{LiO}}\epsilon_{\mathsf{LiO}}}\mathsf{negl}(\lambda) = 2^{-(t+2)L_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Experiment $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{\mathsf{switch}}$. This hybrid is almost same as $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}$, except that we further modify the circuit G_f when generating the functional key sk_f as follows:

Modified $G_f(\{v_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \boldsymbol{\pi}_i\}_{i\in[t]})$

- Decrypt $\mathbf{x}_i^b = \mathsf{PKE.Dec}(\mathsf{sk}_i^b, \mathbf{c}_i^b)$ for each $i \in [t], b \in \{0,1\}$.
- Let $\mathbf{v} = (v_1, \ldots, v_t)$.
- For each $i \in [t]$, check that

$$\mathsf{COM.Commit}\left(\mathsf{crs}, 0; \boldsymbol{\pi}_i\right) \stackrel{?}{=} \mathsf{COM.Commit}\left(\mathsf{crs}, 0; \mathsf{PRF}(\mathbf{k}_i, v_i \| \mathbf{c}_i^0 \| \mathbf{c}_i^1)\right)$$

- For every $i \in [t]$: if $v_i = \widehat{v}_i$, check that (♣)
  - if $\mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{ext}}, \widehat{v}_i) = 0$, then $i \in I$ and $\mathbf{x}_i^0 = \mathbf{x}_i^1$,
  - else, $(\mathbf{c}_i^0, \mathbf{c}_i^1) = (\widehat{\mathbf{c}}_i^0, \widehat{\mathbf{c}}_i^1)$ and $(\mathbf{x}_i^0, \mathbf{x}_i^1) = (\widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1)$.
- If $\mathbf{v} < \widehat{\mathbf{v}} + 1$, output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. Otherwise, output $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$.

The functionality of sk_f can only differ by the output of the case $\mathbf{v} = \widehat{\mathbf{v}}$, in which we now output $f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$ instead of $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0)$. When $\mathbf{v} = \widehat{\mathbf{v}}$, the condition (♣) must hold for all $i \in [t]$, so the admissibility of adversary guarantees $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0) = f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$.

Therefore, if we want to prove in $\mathcal{EF}$ that the modified G_f circuits of $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{\mathsf{switch}}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{*}$ are equivalent, we mainly need to prove in $\mathcal{EF}$ that if (♣) holds for all $i \in [t]$, then $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_t^0) = f(\mathbf{x}_1^1, \ldots, \mathbf{x}_t^1)$. In the most general case, we may need to iterate through all possible messages for $i \in I$ for proving this statement, which introduces an exponential in input-length blowup in the security reduction, which implies that we will suffer from polynomial in input-length overhead in the efficiency of the scheme. However, for the following special cases, we can avoid this blowup:

- **Single-input.** This is the case when $t = 1$. If the condition (♣) holds, then $\mathbf{x}_1^0$ and $\mathbf{x}_1^1$ are either identical or equal to the challenge pair $(\widehat{\mathbf{x}}_1^0, \widehat{\mathbf{x}}_1^1)$. In both cases, the fact that $f(\mathbf{x}_1^0, \ldots, \mathbf{x}_1^t) = f(\mathbf{x}_1^1, \ldots, \mathbf{x}_1^t)$ can be proven simply by evaluating the function f on the two inputs. Therefore, the functional equivalence of the modified G_f circuits can be proven in $\mathcal{EF}$ with proof length at most $O(N_{\mathrm{circ}}^{\mathcal{F}}) + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = O(N_{\mathrm{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$.

- **Secret-key.** In this case $I = \emptyset$. If the condition (♣) holds, then each $(\mathbf{x}_i^0, \mathbf{x}_i^1)$ can only be the challenge message $(\widehat{\mathbf{x}}_i^0, \widehat{\mathbf{x}}_i^1)$. So the functional equivalence can be efficiently verified by evaluating on both inputs. Therefore, the modified G_f circuits of the two experiments enjoy an $\mathcal{EF}$ proof of equivalence of size at most $O(N_{\text{circ}}^{\mathcal{F}}) + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) = O(N_{\text{circ}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$.
- **Proof of equivalence.** Suppose that the function family $\mathcal{F}_\lambda$ admits an efficient $\mathcal{EF}$ proof of equivalence of size $N_{\text{proof}}^{\mathcal{F}}(\lambda, t, L)$. Therefore, the two equivalence of the modified G_f circuits of the two experiments can be proven in $\mathcal{EF}$ with proof length at most $O(N_{\text{circ}}^{\mathcal{F}} + tL \cdot \mathsf{poly}(\lambda, \lambda_{\mathsf{PRF}}, \lambda_{\mathsf{COM}}) + N_{\text{proof}}^{\mathcal{F}}) = O(N_{\text{circ}}^{\mathcal{F}} + N_{\text{proof}}^{\mathcal{F}}) + L \cdot \mathsf{poly}'(\lambda, t)$ which is polynomially bounded in λ as long as $N_{\text{proof}}^{\mathcal{F}}$ is polynomially bounded.

In all of the above cases, we have the following due to the security of the underlying LiO:

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^*) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{\mathsf{switch}}) = 1]\right| = 2^{-tL_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Notice that the only difference between $\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{\mathsf{switch}}$ and $\mathsf{Hyb}_{\widehat{\mathbf{v}}+1}$ is the extra validation check (♣). Relying on a reverse sequence of hybrid experiments, we can undo the validation check (♣) and prove that

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}^{\mathsf{switch}}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}+1}) = 1]\right| = 2^{-tL_{\mathsf{PKE}}}\mathsf{negl}(\lambda).$$

Therefore, we have

$$\left|\Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}}) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{\widehat{\mathbf{v}}+1}) = 1]\right| = 2^{-tL_{\mathsf{PKE}}}\mathsf{negl}(\lambda),$$

and $|\Pr[\mathcal{A}(\mathsf{Real}_0^*) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}) = 1]| = |\Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_{2^{tL_{\mathsf{PKE}}}}) = 1]| = \mathsf{negl}(\lambda)$.

Experiment Real_1. This hybrid is the experiment $\mathsf{INDFE}^{1,\mathcal{A}}(\lambda)$. By a reverse hybrid sequence, we can prove Hyb and Real_1 are indistinguishable.

In conclusion, Real_0 and Real_1 are indistinguishable.

Theorem 6.1 (Multi-input functional encryption)**.** *The construction in Section 6.4 is a polynomially secure multi-input functional encryption scheme for the following settings:*

1. *single-input;*
2. *secret-key settings; and*
3. *the family $\mathcal{F}_\lambda$ admits a polynomially-sized equivalence proof.*

Further, the scheme enjoys the following performance bounds:

- **Single-input or secret-key settings:** *the encryption time is $\widetilde{O}_\lambda(L \cdot \mathsf{poly}(t))$, the key size, key generation time, and decryption time are bounded by $\widetilde{O}_\lambda(N_{\text{circ}}^{\mathcal{F}} + L \cdot \mathsf{poly}(t))$.*
- **$\mathcal{F}_\lambda$ admits size-$N_{\text{proof}}^{\mathcal{F}}$ equivalence proof:** *the encryption time is $\widetilde{O}_\lambda(L \cdot \mathsf{poly}(t))$, the key size, key generation time, and decryption time are bounded by $\widetilde{O}_\lambda(N_{\text{circ}}^{\mathcal{F}} + L \cdot \mathsf{poly}(t) + N_{\text{proof}}^{\mathcal{F}})$.*

In all of the above cases, assuming that $t = \mathsf{poly}(\lambda)$, then the costs have only quasi-linear dependency on the message length L, and the maximum circuit size of the function family $\mathcal{F}$. For the case when $\mathcal{F}_\lambda$ admits size-$N_{\text{proof}}^{\mathcal{F}}$ equivalence proof, there is also a quasi-linear dependence on the proof size $N_{\text{proof}}^{\mathcal{F}}$.

6.6 Barriers for Further Improvement

Necessity of equivalence proof for MIFE. Our results raises the following natural question: for the more general case (not just restricted to single-input or secret-key settings), can we achieve quasi-linear-cost MIFE without relying on proof of equivalence? There seems to be an inherent barrier towards this goal. In particular, if we could indeed have an arity-2 public-key MIFE with quasi-linear efficiency, then it would lead to an iO scheme with quasi-linear efficiency (without proofs of equivalence). However, this would violate the conjectured input-length barrier for iO without proof of equivalence [JJ22].

Theorem 6.2. *Suppose there is an arity-2, public-key* MIFE *scheme for 2-input circuits of maximum size N, and the scheme enjoys $T(\lambda, N)$ decryption time. Then, there exists an indistinguishability obfuscation scheme for (single-input) circuits of size N whose evaluation time is bounded by $T(\lambda, \widetilde{O}(N))$.*

The construction is very similar to what Goldwasser et al. [GGG+14] described. To obfuscate a circuit $f \in \mathcal{F}$ with L-bit inputs, we run $\mathsf{ek}_1, \mathsf{ek}_2, \mathsf{msk} \leftarrow \mathsf{MIFE.Gen}(1^\lambda, L)$, $\mathsf{sk}_U \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{msk}, U)$ where U is the universal circuit defined as $U(f, x) = f(x)$, and $\mathsf{ct}_1 \leftarrow \mathsf{MIFE.Enc}(\mathsf{ek}_1, f)$. We then output the obfuscated circuit which includes $(\mathsf{ek}_2, \mathsf{sk}_U, \mathsf{ct}_1)$. To evaluate the obfuscated circuit on the input $x \in \{0,1\}^L$, simply call $\mathsf{ct}_2 \leftarrow \mathsf{MIFE.Enc}(\mathsf{ek}_2, x)$, and then output $\mathsf{MIFE.Dec}(\mathsf{sk}_U, \mathsf{ct}_1, \mathsf{ct}_2)$. Therefore, if the MIFE scheme's evaluation cost is quasi-linear in the size of U which is quasi-linear in the maximum size of any circuit $f \in \mathcal{F}_\lambda$ [Val76], then the resulting iO scheme would enjoy evaluation time that is quasi-linear in the maximum circuit size of $\mathcal{F}_\lambda$. Security of the iO scheme follows in a straightforward manner from the security of the underlying MIFE.

Necessity of polynomial dependence on t for secret-key MIFE. Another interesting question is whether the polynomial dependence on the function arity t is necessary. Below, we prove that for the secret-key setting without proofs of equivalence, this polynomial dependency on t is inherent, assuming the input-length barrier of iO [JJ22]. Specifically, we show that a secret-key MIFE scheme with quasi-linear efficiency would imply an iO with quasi-linear efficiency without proofs of equivalence. An interesting direction for future research is to understand whether the polynomial dependence on t is necessary if we have a succinct proof of equivalence.

Theorem 6.3. *Suppose there exists a secret-key* MIFE *for arbitrary-arity circuits of size at most N and per-message length $L = 1$, and the scheme enjoys decryption time $T(\lambda, N)$. Then, there exists an indistinguishability obfuscatior for arbitrary circuits of size N whose evaluation time is bounded by $T(\lambda, \widetilde{O}(N))$.*

Proof. The construction is similar to what earlier works described [GGG+14, AJ15]. Given a secret-key MIFE scheme $\mathsf{MIFE} = (\mathsf{MIFE.Gen}, \mathsf{MIFE.Enc}, \mathsf{MIFE.KeyGen}, \mathsf{MIFE.Dec})$, and a circuit C whose size is bounded by some a-priori bound N and whose input length is L, we can obfuscate C as follows, Below, let $C = (C_1, \ldots, C_N)$ be the bit-by-bit encoding of the circuit C, and let $x = (x_1, \ldots, x_L)$ be the bit-by-bit encoding of the string x. Run $(\mathsf{ek}_1, \ldots, \mathsf{ek}_{N+L+1} \leftarrow \mathsf{MIFE.Gen}(1^\lambda, 1^{N+L}, 1)$. The obfuscated circuit consists of the following terms:

- $\mathsf{sk} \leftarrow \mathsf{MIFE.KeyGen}(U)$, where U is a universal circuit such that $U(C_1, \ldots, C_N, x_1, \ldots, x_L) = C(x)$.
- $\mathsf{ct}_C \leftarrow (\mathsf{MIFE.Enc}(\mathsf{ek}_1, C_1), \ldots, \mathsf{MIFE.Enc}(\mathsf{ek}_N, C_N))$.

- $\mathsf{ct}_{x,0} = \{c_{i,0}\}_{i\in[L]}$ where $c_{i,0} = \mathsf{MIFE.Enc}(\mathsf{ek}_{N+i}, 0)$.
- $\mathsf{ct}_{x,1} = \{c_{i,1}\}_{i\in[L]}$ where $c_{i,1} = \mathsf{MIFE.Enc}(\mathsf{ek}_{N+i}, 1)$.

This information is sufficient to reconstruct $C(x)$ for an arbitrary L-bit input $x = (x_1, \ldots, x_L)$: to reconstruct $C(x)$, simply call $\mathsf{MIFE.Dec}(\mathsf{sk}, \mathsf{ct}_C, \{c_{\ell,x_\ell}\}_{\ell\in[L]})$. Security of the resulting iO construction follows from the security of the MIFE scheme: consider two functionally equivalent circuits C and C'. The security of the secret-key MIFE ensures that $(\mathsf{ct}_C, \mathsf{ct}_{x,0}, \mathsf{ct}_{x,1})$ and $(\mathsf{ct}_{C'}, \mathsf{ct}_{x,0}, \mathsf{ct}_{x,1})$ are indistinguishable because $U(C_1, \ldots, C_N, y_1, \ldots, y_L) = U(C'_1, \ldots, C'_N, y_1, \ldots, y_L)$ for all inputs $y = (y_1, \ldots, y_L)$. Since the size of the universal circuit is quasi-linear in N, we get that the resulting iO's evaluation time is bounded by $T(\widetilde{O}(N))$. □

7 Quasi-Linear Indistinguishable Obfuscator for Turing Machines

7.1 Turing Machines with Succinct Proof of Equivalence

Given a Turing Machine M that runs in time at most $T(L)$ on length-L inputs, we can convert M to a circuit when fixing some input length L [PF79]. The resulting circuit is uniform in the sense that it can be generated by a uniform machine, and the circuit size is bounded by $O(T(L) \cdot \log T(L))$. Henceforth, we will use the notation $[\![M]\!]_L$ to denote the circuit induced by M for input length L.

Succinct proof of equivalence for TMs. Like Jain and Jin [JJ22], we construct an obfuscator for TMs that achieves indistinguishability for any two TMs with a proof of equivalence in the PV system. Specifically, we will need the following property that is implied by the existence of a PV proof.

Definition 4 (TM family admits $\mathcal{EF}$ proof of equivalence)**.** We say that two functionally equivalent Turing Machines M and M' with running time $T(\cdot)$ admit an $\mathcal{EF}$ proof of equivalence of size $N_{\text{proof}}(\cdot)$, iff for every input length L, there is a succinct RAM machine $M^{\text{prove}}(\cdot)$ with description size at most $\mathsf{poly}(\mathsf{desc}(M), \lambda)$ (here $\mathsf{desc}(M)$ denotes the description size of M) which, on receiving the input length L, outputs an $\mathcal{EF}$ proof of equivalence of length $N_{\text{proof}}(L)$ between $[\![M]\!]_L$ and $[\![M']\!]_L$.

We say that a family of Turing Machines admits $\mathcal{EF}$ proofs of equivalence of size at most $N_{\text{proof}}(\cdot)$, iff any two functionally equivalent TMs from the family admit an $\mathcal{EF}$ proof of equivalence of size $N_{\text{proof}}(\cdot)$.

Definition 5 (iO for TMs with proof of equivalence)**.** Let $\mathcal{M}$ be a family of TMs whose running time is polynomially bounded in the input length, and suppose that $\mathcal{M}$ admits $\mathcal{EF}$ proofs of equivalence with proof length polynomially bounded by the input length. A randomized algorithm $\mathsf{TMObf}(1^\lambda, \cdot)$ is said to be a secure indistinguishability obfuscator for the Turing Machine family $\mathcal{M}$, iff for any two functionally equivalent $M, M' \in \mathcal{M}$, $\mathsf{TMObf}(1^\lambda, M) \approx \mathsf{TMObf}(1^\lambda, M')$. Further, for any M and λ, the obfuscated program $\mathsf{TMObf}(1^\lambda, M)$ must be functionally equivalent to the original M.

Fact 7.1. *Let M and M' be two Turing Machines whose description size is bounded by desc and whose running time is bounded by $T(\cdot)$. Suppose that M and M' admit an $\mathcal{EF}$ proof of equivalence of length $N_{\text{proof}}(\cdot)$, then, there exists m which is polynomially bounded in desc and T, and a succinct RAM machine M^{gen} of size $\mathsf{poly}(\mathsf{desc}(M), \lambda)$, such that on receiving the input length L and an index $i \in [m]$, $M^{\text{gen}}(L, i)$ outputs a padded circuit C_i, such that the circuits $C_1, \ldots, C_m$ satisfy the following properties:*

- $C_1 = \mathsf{Pad}(\llbracket M \rrbracket_L, T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}), N_{\mathrm{proof}}(L))$, *and* $C_m = \mathsf{Pad}(\llbracket M \rrbracket_L, T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}), N_{\mathrm{proof}}(L))$.
- C_i *and* C_{i+1} *are* s*-equivalent for* $i \in [m-1]$*, where* $s = O(\log(T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}) + N_{\mathrm{proof}}(L)))$.

Proof. By the definition of admitting $\mathcal{EF}$ proof of equivalence, we know that there exists some succinct RAM machine $M^{\mathrm{prove}}(\cdot)$ efficiently generates an $\mathcal{EF}$ equivalence proof of size $N_{\mathrm{proof}}(L)$ for $\llbracket M \rrbracket_L$ and $\llbracket M' \rrbracket_L$, therefore by Lemma 3.1, we know that $\mathsf{Pad}(\llbracket M \rrbracket_L, T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}), N_{\mathrm{proof}}(L))$ and $\mathsf{Pad}(\llbracket M \rrbracket_L, T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}), N_{\mathrm{proof}}(L))$ are transitively s-equivalent via some sequence $C_1, \ldots, C_m$ for $m = \mathsf{poly}(\mathsf{desc}, T)$.

It remains to show that all the intermediate hybrid circuits C_i can be succinctly and uniformly generated by some RAM machine M^{gen}. Recall that each of the padded circuits and hybrid circuits consists of the following components: $\widetilde{C}_1(\cdot), \widetilde{C}_2(\cdot), \widetilde{C}_{\mathrm{proof}}(\cdot)$ and the AND circuit, where the detailed components of each phases can be found in Table 1. Note that given the circuits $\llbracket M \rrbracket_L, \llbracket M' \rrbracket_L$ and $\mathcal{EF}$ equivalence proof, the RAM program M^{gen} can be designed to output each hybrid circuits on Table 1 in quasi-linear time $\widetilde{O}(T\mathsf{poly}(\mathsf{desc}) + N_{\mathrm{proof}}(L))$ by calling the subroutine $\mathsf{PadSingle}(\cdot, \cdot)$. However, M^{gen} should also support outputting a sequence of inner-hybrid circuits showing the transitively s-equivalence between the adjacent hybrid circuits on Table 1. Here we present two typical instances:

- **Initial circuit and grow C_2.** The only difference between these two hybrid circuits is that, $\widetilde{C}_2(\mathbf{1})$ is replaced by $\widetilde{C}_2(\llbracket M' \rrbracket_L)$. Recall that in order to prove the transitively s-equivalence, we iteratively switch all gates in $\widetilde{C}_2$ to the right configuration. M^{gen} can just faithfully simulate this procedure: given an index i, M^{gen} first determines how many gates have been switched, then outputs circuit $C_1 = \mathsf{Pad}(\llbracket M \rrbracket_L, T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}), N_{\mathrm{proof}}(L))$ except that the first few gates of $\widetilde{C}_2(\mathbf{1})$ is reconfigured to corresponding ones of $\widetilde{C}_2(\llbracket M' \rrbracket_L)$.
- **Grow C_2 and grow proof for $C_1 = C_2$.** The only difference between these two hybrid circuits is that, $\widetilde{C}_{\mathrm{proof}}(\mathbf{1})$ is replaced by $\widetilde{C}_{\mathrm{proof}}([\llbracket M \rrbracket_L = \llbracket M' \rrbracket_L])$. In the proof we split this step into two sub-steps: switch $\widetilde{C}_{\mathrm{proof}}(\mathbf{1})$ to $\widetilde{C}_{\mathrm{proof}}(C'_{\mathrm{proof}})$, where C'_{proof} is almost same as $[\llbracket M \rrbracket_L = \llbracket M' \rrbracket_L]$ except that output gates constantly output 1; switch the output gates to the correct outputs. The first sub-step is analogous to the previous scheme "Initial circuit and grow C_2", we focus on how M^{gen} can simulate the second sub-step. As we done in the proof, M^{gen} also gradually switch the output gates. Formally, given an index i, M^{gen} first determines how many output gates should be switched, then output a hybrid circuit where the first few output gates of $\widetilde{C}_{\mathrm{proof}}$ correctly output values, and the rest of them constantly output 1.

□

7.2 Building Blocks

Preliminary: iO for RAM. We rely on the iO for RAM construction of Jain et al. [JLL23]. Although their iO for RAM construction suffers from polynomial in input-length cost for the evaluator, the evaluator's running time has only quasi-linear dependence on T, the running time of the original RAM. More specifically, Jain et al. [JLL23] proved the following:

Lemma 7.2 (iO for RAM)**.** *Assuming the existence of sub-exponentially secure* iO *for circuits and sub-exponentially secure one-way functions, there exists sub-exponentially secure* iO *for RAM (for bounded-length inputs), where the obfuscated program has size* $O(\mathsf{desc}(M)) + \mathsf{poly}(\lambda)$*, the evaluator's runtime is* $\widetilde{O}_\lambda(T) \cdot \mathsf{poly}(\mathsf{desc}(M), L)$*, and the obfuscator's runtime is* $\mathsf{desc}(M) \cdot \mathsf{poly}(\lambda, L)$ *where* $\mathsf{desc}(M)$ *is the size of TM and* L *is the input length.*

Building blocks. We use following ingredients in our construction:

- Sub-exponentially secure puncturable pseudorandom functions kPRF and oPRF.
- Sub-exponentially secure constant-overhead RAM indistinguishability obfuscator RAMObf.
- Sub-exponentially secure quasi-linear locally indistinguishability obfuscator LiO for $O(\lambda)$-equivalent circuits. In particular, we will use the construction of Section 5.

7.3 Construction

Next, we present our indistinguishability obfuscator for TMs assuming proofs of equivalence.

Obfuscation algorithm. The obfuscation algorithm first samples PRF keys kSK and oSK corresponding to kPRF and oPRF respectively. Then, it outputs

$$\widetilde{\mathsf{UObf}} \leftarrow \mathsf{RAMObf}(1^\lambda, \mathsf{UObf}_{1^\lambda, M, N_{\mathrm{proof}}(\cdot), T(\cdot), \mathsf{kSK}, \mathsf{oSK}}).$$

In particular, $\mathsf{UObf}_{\lambda, M, N_{\mathrm{proof}}(\cdot), T(\cdot), \mathsf{kSK}, \mathsf{oSK}}(L)$ is the following machine which computes an obfuscated circuit for M given an input length L:

$\mathsf{UObf}_{\lambda, M, N_{\mathrm{proof}}(\cdot), T(\cdot), \mathsf{kSK}, \mathsf{oSK}}(L)$

// Let $s = c \log(T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}(M)) + N_{\mathrm{proof}}(L))$ for some universal constant c.

- Let $C = \mathsf{Pad}([\![M]\!]_L, T \log T \cdot \mathsf{poly}(\mathsf{desc}(M)), N_{\mathrm{proof}}(L))$.
- Output an obfuscated circuit $\mathsf{LiO}(1^\lambda, s, C)$ seeded with the PRF keys kSK and oSK. Specifically, $\mathsf{kPRF}(\mathsf{kSK}, \cdot)$ is used to generate the random coins used for sampling keys in the construction of LiO, and $\mathsf{oPRF}(\mathsf{oSK}, \cdot)$ is used to generate the random coins needed by the internal circuit obfuscator Obf called by LiO.

In the above, we assume that the description of M comes with an a-priori bound on the maximum description length of the family of TMs we care about obfuscating. For convenience, we use $\mathsf{desc}(M)$ to denote this a-priori bound.

Evaluation algorithm. We now describe the evaluation algorithm.

Evaluation of an obfuscated TM $\widetilde{\mathsf{UObf}}$

Given an input $x \in \{0,1\}^L$:

- Obtain the obfuscated program $\widetilde{C} = \widetilde{\mathsf{UObf}}(L)$.
- Output $\widetilde{C}(x)$ using the evaluation algorithm described in Section 5.

Efficiency. Since $L \leq 2^\lambda$ the description length of L (i.e., input length of UObf) is bounded by λ. Further, due to the quasi-linear efficiency of the LiO scheme, the $\widetilde{\mathsf{UObf}}(\cdot)$ program runs in time $\widetilde{O}_\lambda(T \cdot \mathsf{poly}(\mathsf{desc}(M)) + N_{\mathrm{proof}}(L))$. By Lemma 7.2, the obfuscated program size is $O(\mathsf{desc}(M)) + \mathsf{poly}(\lambda)$, and runs in time $\widetilde{O}_\lambda(T \cdot \mathsf{poly}(\mathsf{desc}(M)) + N_{\mathrm{proof}}(L))$. Moreover, the obfuscator's runtime is $\mathsf{desc}(M) \cdot \mathsf{poly}(\lambda)$.

7.4 Security Proofs

Given two TMs M, M' which admit a succinct $\mathcal{EF}$ proof of equivalence, we want to show that an obfuscation of M is indistinguishable from an obfuscation of M'. Specifically, we define the following experiments Real and Real'.

Experiment Real**.** Output $\mathsf{RAMObf}(1^\lambda, \mathsf{UObf}_{\lambda,M,N_{\mathrm{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}})$ where kSK and oSK are sampled at random.

Experiment Real'**.** Output $\mathsf{RAMObf}(1^\lambda, \mathsf{UObf}_{\lambda,M',N_{\mathrm{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}})$ where kSK and oSK are sampled at random.

We want to prove that Real is indistinguishable from Real'. We prove this via a sequence of hybrid experiments defined below.

Hybrid Hyb_ℓ**.** This hybrid is indexed by ℓ (upper bounded by 2^λ). For $L < \ell$, we obfuscate the $\mathsf{UObf}_{\lambda,M,N_{\mathrm{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}}$ program just like in Real; otherwise we obfuscate the $\mathsf{UObf}_{\lambda,M',N_{\mathrm{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}}$ program like in Real'. Clearly $\mathsf{Hyb}_0 \equiv \mathsf{Real}$ and $\mathsf{Hyb}_{2^\lambda} \equiv \mathsf{Real}'$. We will use a sequence of intermediate hybrids to prove $\mathsf{Hyb}_\ell \approx \mathsf{Hyb}_{\ell+1}$, which implies that $\mathsf{Real} \approx \mathsf{Real}'$.

Let M^{gen} be the RAM that outputs the intermediate circuits $C_1, \ldots, C_m$ given M and M' (see Fact 7.1).

Hybrid Hyb_ℓ^i**.** In this hybrid UObf is modified to:

Modified $\mathsf{UObf}(L)$

// Let $s = c \log(T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}(M)) + N_{\mathrm{proof}}(L))$

- If $L < \ell$, return $\mathsf{UObf}_{\lambda,M,N_{\mathrm{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}}(L)$
- Else if $L > \ell$, return $\mathsf{UObf}_{\lambda,M',N_{\mathrm{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}}(L)$
- Otherwise, let $C_i \leftarrow M^{\mathrm{gen}}(L, i)$, and output an obfuscated circuit $\mathsf{LiO}(1^\lambda, s, C_i)$ seeded with PRF keys kSK and oSK.

Remark 7.3 (How LiO generates pseudorandom coins)**.** *In the above, we use the key* kSK *to seed the coins needed for sampling the keys needed by* LiO*, and use* oSK *to seed the coins needed for sampling the coins consumed by the inner obfuscator* Obf *called by* LiO*. In particular, recall that the* LiO *algorithm needs to sample the following keys:*

- *global: somewhere extractable hash key* hk*;*
- *global:* FHE *key pair* $(\mathsf{sk}, \mathsf{pk})$ *and trapdoor* td'*;*
- *global: symmetric encryption keys* $\{K'_j\}_{j\in[s]}$*; and*
- *per-wire: for each wire* w *in the circuit: puncturable* PRF *keys* K^m_w, K^σ_w*.*

We may assume that each key can be uniquely identified by a name in the format of $(w, \text{type of key})$*, where* $w = \perp$ *for a global key, and* w *refers to the wire identifier for a per-wire key; and the field "type of key" states which type of key. Each key can be generated using the template* $\mathsf{kPRF}(\mathsf{kSK}, \text{name of key})$*. Henceforth, let* names_S *denote the names of following keys: 1) all global keys; and 2) per-wire keys for wires attached to gates in* S*. Further, we may assume that when* LiO *obfuscates the* Gate^g

circuit corresponding to gate g, it uses $\mathsf{oPRF}(\mathsf{oSK}, g)$ *to generate the random coins needed by the inner obfuscator* Obf.

We now continue with the proof. Since the modified UObf program of Hyb_ℓ^1 is functionally equivalent to that of Hyb_ℓ, and the modified UObf program of Hyb_ℓ^k is functionally equivalent to that of $\mathsf{Hyb}_{\ell+1}$, we have $\mathsf{Hyb}_\ell^1 \approx \mathsf{Hyb}_\ell$ and $\mathsf{Hyb}_\ell^k \approx \mathsf{Hyb}_{\ell+1}$ by the security of RAMObf. Next, we will show that $\mathsf{Hyb}_\ell^i \approx \mathsf{Hyb}_\ell^{i+1}$ which implies that $\mathsf{Hyb}_\ell \approx \mathsf{Hyb}_{\ell+1}$.

Hybrid $\mathsf{Hyb}_\ell^{i,0}$. Henceforth, let S be the subcircuit of size s where C_i and C_{i+1} differ (but without affecting the functionality). The experiment $\mathsf{Hyb}_\ell^{i,0}$ is almost identical to Hyb_ℓ^i except that we further modify the UObf program as follows:

Modified UObf

// Let $s = c\log(T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}(M)) + N_{\mathrm{proof}}(L))$

- If $L < \ell$, return $\mathsf{UObf}_{\lambda, M, N_{\mathrm{proof}}(\cdot), T(\cdot), \mathsf{kSK}, \mathsf{oSK}}(L)$;
- Else if $L > \ell$, return $\mathsf{UObf}_{\lambda, M', N_{\mathrm{proof}}(\cdot), T(\cdot), \mathsf{kSK}, \mathsf{oSK}}(L)$
- Otherwise,
  - Let $C_i = M^{\mathrm{gen}}(L, i)$.
  - Generate all keys as follows: each key identified by some name is generated by

$$\begin{cases} \mathsf{kPRF}(\underbrace{\mathsf{kSK}/\{\mathsf{names}_S\}}_{\text{hard-wired}}, \mathsf{name}), & \mathsf{name} \notin \mathsf{names}_S; \\ \underbrace{\mathsf{kPRF}(\mathsf{kSK}, \mathsf{name})}_{\text{hard-wired}}, & \text{otherwise.} \end{cases}$$

  - For each gate g, generate the random coins needed by the internal obfuscator Obf as follows:

$$\begin{cases} \mathsf{oPRF}(\underbrace{\mathsf{oSK}/\{S\}}_{\text{hard-wired}}, g) & g \notin S; \\ \underbrace{\mathsf{oPRF}(\mathsf{oSK}, g)}_{\text{hard-wired}} & \text{otherwise.} \end{cases}$$

  - Output $\mathsf{LiO}(1^\lambda, s, C_i)$ except that we generate the coins needed by LiO as above.

Due to the puncturing correctness of the punctured PRF, the modified UObf program of Hyb_ℓ^i is functionally equivalent to that of $\mathsf{Hyb}_\ell^{i,0}$. Therefore, we have that $\mathsf{Hyb}_\ell^i \approx \mathsf{Hyb}_\ell^{i,0}$ from the security of RAMObf.

Hybrid $\mathsf{Hyb}_\ell^{i,1}$. $\mathsf{Hyb}_\ell^{i,1}$ is almost the same as $\mathsf{Hyb}_\ell^{i,0}$, except that in the case of $L = \ell$, all keys related to gates in S are hard-wired random values (sampled honestly using the relevant key generation algorithms):

Modified UObf

// Let $s = c\log(T \cdot \log T \cdot \mathsf{poly}(\mathsf{desc}(M)) + N_{\mathrm{proof}}(L))$

- If $L < \ell$, return $\mathsf{UObf}_{\lambda,M,N_{\text{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}}(L)$;
- Else if $L > \ell$, return $\mathsf{UObf}_{\lambda,M',N_{\text{proof}}(\cdot),T(\cdot),\mathsf{kSK},\mathsf{oSK}}(L)$
- Otherwise,
  - Let $C_i = M^{\text{gen}}(L, i)$.
  - Generate all keys as follows: each key identified by some name is generated by

$$\begin{cases} \mathsf{kPRF}(\underbrace{\mathsf{kSK}/\{\mathsf{names}_S\}}_{\text{hard-wired}}, \mathsf{name}), & \mathsf{name} \notin \mathsf{names}_S; \\ \text{hard-wired random value,} & \text{otherwise.} \end{cases}$$

  - For each gate g, generate the random coins needed by the internal obfuscator Obf as follows:

$$\begin{cases} \mathsf{oPRF}(\underbrace{\mathsf{oSK}/\{S\}}_{\text{hard-wired}}, g) & g \notin S; \\ \text{hard-wired random coins,} & \text{otherwise.} \end{cases}$$

  - Output $\mathsf{LiO}(1^\lambda, s, C_i)$ except that we generate the coins needed by LiO as above.

Observe that the original PRF keys kSK and oSK are never used. Therefore, due to the security of the puncturable PRF, $\mathsf{Hyb}_\ell^{i,1}$ and $\mathsf{Hyb}_\ell^{i,0}$ are computationally indistinguishable.

Hybrid $\mathsf{Hyb}_\ell^{i,2}$. $\mathsf{Hyb}_\ell^{i,2}$ is otherwise the same as $\mathsf{Hyb}_\ell^{i,1}$, except that we modify the UObf program such that for the case $L = \ell$, we precompute the obfuscated $\widetilde{\mathsf{Gate}}^g$ gates for all $g \in S$, hardwire these results inside UObf. Now, for each gate $g \notin S$, UObf computes the obfuscated $\widetilde{\mathsf{Gate}}^g$ just like in $\mathsf{Hyb}_\ell^{i,1}$, and for each gate $g \in S$, it simply outputs the precomputed outcome.

Since this modification does not change the functionality of UObf, due to the security of RAMObf, $\mathsf{Hyb}_\ell^{i,2}$ is computationally indistinguishable from $\mathsf{Hyb}_\ell^{i,1}$.

Hybrid $\mathsf{Hyb}_\ell^{i,3}$. $\mathsf{Hyb}_\ell^{i,3}$ is otherwise identical as $\mathsf{Hyb}_\ell^{i,2}$, except that for the case $L = \ell$, instead of obfuscating C_i, the modified UObf program now obfuscates C_{i+1} instead.

In both $\mathsf{Hyb}_\ell^{i,2}$ and $\mathsf{Hyb}_\ell^{i,3}$, the obfuscated gates for the subcircuit S are hard-wired in UObf, and they are for C_i and C_{i+1} respectively in the two experiments. For any gate not in S, the corresponding obfuscated gates output by LiO are identically distributed in the two experiments. Therefore, to argue the computational indistinguishability of $\mathsf{Hyb}_\ell^{i,2}$ and $\mathsf{Hyb}_\ell^{i,3}$, it suffices to argue that the obfuscated gates belonging to S are computationally indistinguishable in the two experiments. This can be shown using the same proof as that of Lemma 5.1. In particular, it is easy to check that the proof of Lemma 5.1 would still hold as long as the coins associated with S are chosen randomly, including the coins for generating the keys for coins in name_S, and the coins for the inner Obf for gates in S.

Now, using a symmetric reverse sequence of hybrids, we can argue that $\mathsf{Hyb}_\ell^{i,3} \approx \mathsf{Hyb}_{\ell+1}$. This is sufficient for arguing that $\mathsf{Hyb}_\ell \approx \mathsf{Hyb}_{\ell+1}$.

In summary, there are only polynomially many hybrid experiments that go from Real to Real', so if the underlying primitives kPRF, oPRF, RAMObf, and LiO enjoy subexponential security, then the distinguishing probability of Real and Real' is also subexponentially small. In conclusion, we have the following theorem.

Theorem 7.4 (Sub-exponentially secure indistinguishability obfuscation for TMs with succinct proof of equivalence)**.** *Assume the following hard problems:*

- *sub-exponential hardness of the* LWE *assumption,*
- *sub-exponentially secure one-way functions, and*
- *sub-exponentially secure indistinguishability obfuscation for circuits.*

Let $T(\cdot)$ and $N_{\text{proof}}(\cdot)$ be polynomially bounded functions. Let $\mathcal{M}$ be a family of TMs with description size at most desc*, running time at most $T(L)$ on length-L inputs, and suppose that $\mathcal{M}$ admits $\mathcal{EF}$ proofs of equivalence with proof length at most $N_{\text{proof}}(L)$ on length-L inputs.*

Then, for any $\epsilon > 0$, there exists a 2^{λ^ϵ}- sub-exponential secure indistinguishability obfuscator for $\mathcal{M}$, such that the obfuscator and evaluator's running times are upper bounded by $\widetilde{O}_\lambda(T(L) \cdot \mathsf{poly}(\mathsf{desc}) + N_{\text{proof}}(L))$ on length-L inputs, and the size of obfuscated program is bounded by $\mathsf{poly}(\mathsf{desc}, \lambda)$.

Acknowledgments

This work is in part supported by NSF awards 2128519 and 2044679, an ONR grant, and a DARPA SIEVE grant under a subcontract from SRI.

References

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In *CRYPTO*, 2017.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, 2023.

[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 Journal on Computing*, 2018.

[BCP15] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel ram. In *TCC*, 2015.

[BFK+19] Saikrishna Badrinarayanan, Rex Fernando, Venkata Koppula, Amit Sahai, and Brent Waters. Output compression, mpc, and io for turing machines. In *ASIACRYPT*, 2019.

[BS80] Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of Cryptography*, pages 468–497, 2015.

[CW14] Jie Chen and Hoeteck Wee. Semi-adaptive attribute-based encryption and improved delegation for boolean formula. In *SCN*, 2014.

[FSS+23] Rex Fernando, Elaine Shi, Pratik Soni, Nikhil Vanjani, and Brent Waters. Non-interactive anonymous router with quasi-linear router computation. In *TCC*, 2023.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In *CRYPTO*, 2023.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Eurocrypt*, 2014.

[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 *FOCS*, 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 1986.

[GJO16] Vipul Goyal, Aayush Jain, and Adam O'Neill. Multi-input functional encryption with unbounded-message security. In *ASIACRYPT*, 2016.

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In *TCC*, 2016.

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 1999.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[JJ22] A. Jain and Z. Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1023–1034, nov 2022.

[JLL23] Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 479–510. Springer, 2023.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 2021.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 419–428, 2015.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, April 1979.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 475–484, New York, NY, USA, 2014. Association for Computing Machinery.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 196–203. ACM, 1976.

[WW22] Brent Waters and David J. Wu. Batch arguments for np and more from standard bilinear group assumptions. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, page 433–463, 2022.

A Additional Preliminaries

In this section, we review the notation and definition of some cryptographic primitives used in this work.

We use the notation $x \xleftarrow{\$} D$ denotes uniformly sampling a value x from distribution D. Similarly, for a (possibly randomized) algorithm A, $y \leftarrow A(x)$ denotes algorithm A outputs y on input x. Sometimes we write $y \leftarrow A(x; r)$ to specify the random coins r used in A on input x.

In this paper, we use λ denotes the overall security parameter, and for some specific primitive P, we may denote λ_P as the security parameter of P, it is always the case that $\lambda_P = \mathsf{poly}(\lambda)$. We slightly abuse statement "PPT adversary" to cover non-uniform probabilistic polynomial-time adversary (i.e. family of polynomial-time probabilistic circuits). A positive function $\epsilon(\lambda)$ is said to be negligible in λ if for any $c > 0$, there exists λ_c where $\epsilon(\lambda) < n^{-c}$ whenever $\lambda > \lambda_c$. Moreover, $\mathsf{negl}(\lambda)$ denotes some (unspecified) negligible function in λ.

A.1 Pseudorandom Generators

A pseudorandom generator (PRG) stretches some truly random inputs to a longer sequence which is computationally indistinguishable to uniformly random string.

Definition 6. For an given function $\mathsf{PRG} : \{0,1\}^l \to \{0,1\}^n$ where $l < n$ is polynomially bounded in λ and any PPT adversary $\mathcal{A}$, define the advantage of $\mathcal{A}$ as

$$\begin{aligned}&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PRG}}(\lambda)\\ =&\left|\Pr[\mathsf{seed} \xleftarrow{\$} \{0,1\}^l, r \leftarrow \mathsf{PRG}(\mathsf{seed}) : \mathcal{A}(r) = 1] - \Pr[r \xleftarrow{\$} \{0,1\}^n : \mathcal{A}(r) = 1]\right|,\end{aligned}$$

then PRG is said to be a polynomially secure pseudorandom generator if $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PRG}}(\lambda)$ is bounded by some negligible function $\mathsf{negl}(\lambda)$, and is $2^{-\lambda^\epsilon}$ sub-exponentially secure if $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PRG}}(\lambda) \leq 2^{\lambda^\epsilon}\mathsf{negl}(\lambda)$.

It is known that polynomially (or sub-exponentially resp.) secure PRG exists if and only if polynomially (or sub-exponentially resp.) secure one-way functions exists [HILL99].

A.2 Puncturable Pseudorandom Functions

We formally define the notion of puncturable pseudorandom functions (puncturable PRF). In this work, we restrict the power of puncturable PRF to only support puncturing at one point.

Definition 7. A puncturable PRF family consists of deterministic polynomial time algorithms both take as PRF key K and input x: evaluation algorithm $\mathsf{PRF}(K, x)$ outputs the evaluation at given point x, and puncturing algorithm $K\{x\} = \mathsf{PRF}_{\mathrm{punc}}(K, x)$ outputs a punctured PRF key. It satisfies following properties:

- **Correctness.** For any PRF key K and input x, the punctured PRF key preserves the functionality except at point x, that is, for all $x' \neq x$,

$$\mathsf{PRF}(K, x') = \mathsf{PRF}(K\{x\}, x').$$

- **Pseudorandomness.** The punctured PRF key doesn't leak nontrivial information of $\mathsf{PRF}(K, x)$. Formally, for each PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(1^\lambda)$ outputs an input x and private state st, define the advantage of $\mathcal{A}$ as:

$$\begin{aligned}&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PRF}}(\lambda)\\ =&\left|\Pr[\mathcal{A}_2(\mathsf{st}, K/\{x\}, x, \mathsf{PRF}(K, x)) = 1] - \Pr[r \xleftarrow{\$} U_\lambda : \mathcal{A}(\mathsf{st}, K\{x\}, x, r) = 1]\right|,\end{aligned}$$

  where U_λ is the range of PRF. We say the puncturable PRF is polynomially (or 2^{λ^ϵ} sub-exponentially resp.) secure if $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PRF}}(\lambda) \leq \mathsf{negl}(\lambda)$ (or $2^{-\lambda^\epsilon}\mathsf{negl}(\lambda)$ resp.).

As observed by [BW13], the GGM-style PRFs from PRG [GGM86] (and thus, from one-way functions) yields punturable PRFs, moreover, the obtained punturable PRF is polynomially (or sub-exponentially resp.) secure if and only if the PRG building block shares same level of security.

We also observe that for any given K, x, the correctness of puncturable PRF key $K/\{x\}$ can be proved succinctly in $\mathcal{EF}$ system, which is useful for our later construction:

Lemma A.1 ([JJ22]). *For puncturable* PRF *scheme* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$*, the correctness of puncturing at point* x *has a* $\mathsf{poly}(\lambda, n, m)$*-size* $\mathcal{EF}$ *proof:*

$$(x' \neq x) \to (\mathsf{PRF}(K, x') = \mathsf{PRF}(K\{x\}, x')).$$

A.3 Public Key Encryption

Here we review the notion of public key encryption.

Definition 8. A public key encryption (PKE) scheme consists of following PPT algorithms (Gen,Enc,Dec):

- $\mathsf{Gen}(1^\lambda)$: On input a security parameter λ, the key generation algorithm outputs randomly generated public key pk and secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, m)$: On input a public key pk and a message $m \in \{0,1\}^n$, the encryption algorithm outputs a ciphertext $\mathsf{ct} \in \{0,1\}^l$ of m.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Given secret key sk and a ciphertext ct, the decryption algorithm decodes ct and outputs a plaintext message m.

It satisfies the following properties:

- **Correctness.** For any message $m \in \{0,1\}^n$, we have

$$\Pr\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) : \mathsf{Dec}(\mathsf{sk},\mathsf{ct}) = m\right] = 1.$$

  Hence, without loss of generality we may assume Dec is a deterministic algorithm.

- **CPA Security.** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, denote the advantage of $\mathcal{A}$ in CPA game as:

$$\begin{aligned}&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda)\\ =&\left|\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda),\\ (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pk})\\ b \xleftarrow{\$} \{0,1\}\\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array} : \mathcal{A}_2(\mathsf{st}, \mathsf{ct}_b) = b\right] - \frac{1}{2}\right|\end{aligned}$$

We say the PKE scheme is polynomially (or 2^{λ^ϵ} sub-exponentially) IND-CPA secure if the advantage $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda) \leq \mathsf{negl}(\lambda)$ (or $2^{-\lambda^\epsilon}\mathsf{negl}(\lambda)$ resp.).

A IND-CPA secure PKE scheme can be built from various assumptions, including DDH, QR, RSA, LWE and even indistinguishability obfuscation with one-way functions. In this work we choose the LWE-based instantiation, i.e. Regev's PKE scheme [Reg05] as candidate, not only for minimizing assumptions, but also because we can easily obtain a succinct correctness proof in $\mathcal{EF}$ system:

Lemma A.2 ([JJ22]). *Regev's* PKE *scheme [Reg05] has a* $\mathsf{poly}(\lambda, n, l)$*-size* $\mathcal{EF}$ *proof of correctness:*

$$((\mathsf{pk},\mathsf{sk}) = \mathsf{Gen}(1^\lambda; r_1)) \rightarrow (\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m; r_2)) = m).$$

A.4 Non-Interactive Bit Commitment

We also review the definition of perfectly binding non-interactive bit commitment scheme.

Definition 9. A perfectly binding non-interactive bit commitment scheme consists of PPT algorithms $(\mathsf{Setup}, \mathsf{Commit})$:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, the setup algorithm outputs common reference string CRS.
- $\mathsf{Commit}(\mathsf{CRS}, b)$: On input CRS and a bit $b \in \{0,1\}$, the commitment generation algorithm outputs a commitment com.

The scheme should satisfy following properties:

- **Perfect Binding.** For any randomness r, r', we have

$$\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda) : \mathsf{Commit}(\mathsf{CRS}, 0; r) = \mathsf{Commit}(\mathsf{CRS}, 1; r')] = 0$$

- **Computationally Hiding.** For all PPT adversary $\mathcal{A}$, denote the advantage of $\mathcal{A}$ in CPA game as:

$$\begin{aligned}&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{COM}}(\lambda)\\ =&\left|\Pr\left[\begin{array}{l}\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda),\\ b \xleftarrow{R} \{0,1\}\\ \mathsf{com}^* \leftarrow \mathsf{Commit}(\mathsf{CRS}, b)\end{array} : \mathcal{A}(\mathsf{CRS}, \mathsf{com}^*) = b\right] - \frac{1}{2}\right|\end{aligned}$$

The bit commitment scheme is polynomial (or 2^{λ^ϵ} sub-exponential resp.) hiding if $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{COM}}(\lambda) \leq \mathsf{negl}(\lambda)$ (or $2^{-\lambda^\epsilon}\mathsf{negl}(\lambda)$ resp.).

A perfectly binding non-interactive bit commitment scheme can be trivially implemented by a PKE scheme:

- $\mathsf{Setup}(1^\lambda)$: Output $\mathsf{PKE.Gen}(1^\lambda)$.
- $\mathsf{Commit}(\mathsf{CRS}, b)$: Output $\mathsf{PKE.Enc}(\mathsf{CRS}, b)$.

Notice that the bit commitment scheme shares same level of security as underlying PKE. Specifically, if we instantiate the PKE by Regev's construction with sub-exponential security, then the commitment scheme is sub-exponential hiding and has a polynomial-size binding proof by Lemma A.2.

A.5 Fully Homomorphic Encryption

A fully homomorphic encryption (FHE) scheme is a generalized version of PKE, where additionally support homomorphic evaluation.

Definition 10. A FHE scheme for a function family $\mathcal{F}$ consists of following PPT algorithms:

- $\mathsf{Gen}(1^\lambda)$: On input a security parameter λ, the key generation algorithm randomly generates public key pk and secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, m)$: On input a public key pk and a message m, the encryption algorithm outputs ciphertext ct of m.
- $\mathsf{Eval}(\mathsf{pk}, \mathsf{ct}, f)$: On input a public key pk, a ciphertext ct and a function $f \in \mathcal{F}$, the evaluation algorithm outputs a ciphertext ct_f.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Given secret key sk and ciphertext ct, the decryption algorithm decrypts ct and output a plaintext m.

We require the FHE scheme to satisfy following properties:

- **Succinctness.** The output length of evaluated ciphertext $\mathsf{Eval}(\mathsf{pk}, \mathsf{ct}, f)$ should not depend on descryption of function f. Formally, there exists some $L_{\mathsf{FHE}} = \mathsf{poly}(\lambda)$ such that FHE scheme stretches a single bit of plaintext to L_{FHE} bits ciphertext, thus, the output length of Eval algorithm is exactly $|f(m)|L_{\mathsf{FHE}}$ if input ct is ciphertext of m.
- **Correctness.** For any message $m \in \{0,1\}^n$ and function $f \in \mathcal{F}$ with input length n, we have

$$\Pr\begin{bmatrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), & \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m), & : \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_f) = f(m) \\ \mathsf{ct}_f \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{ct}, f) & \end{bmatrix} = 1.$$

- **CPA Security.** The CPA game is define in the same standard way as public key encryption. For all adversary $\mathcal{A}$, denote the advantage of $\mathcal{A}$ in CPA game as $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{FHE}}(\lambda)$, we say the FHE scheme is polynomially (or 2^{λ^ϵ} sub-exponentially resp.) IND-CPA secure if $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{FHE}}(\lambda) \leq \mathsf{negl}(\lambda)$ (or $2^{-\lambda^\epsilon}\mathsf{negl}(\lambda)$ resp.).

It is known that polynomially (or sub-exponentially resp.) secure FHE scheme exists assuming polynomial (or sub-exponential resp.) hardness of LWE assumption [Gen09].

A.6 Indistinguishability Obfuscation

We formally define the notion of indistinguishability obfuscation (iO) for circuits:

Definition 11. An indistinguishability obfuscator iO for circuit family $\mathcal{C}$ takes a circuit $C \in \mathcal{C}$ as input, and outputs an obfuscated circuit. It should satisfy the following:

- **Completeness.** For any $C \in \mathcal{C}$ and any input x of C, it holds that

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

- **Indistinguishability.** For any PPT adversary $\mathcal{A}$, denote the advantage of $\mathcal{A}$ in distinguishing functionally equivalent circuits $C_0, C_1 \in \mathcal{C}$ where $C_0(x) = C_1(x)$ for all input x as:

$$\begin{aligned}&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{iO},C_0,C_1}(\lambda)\\ =&\left|\Pr[\mathcal{A}(\mathsf{iO}(1^\lambda, C_0)) = 1] - \Pr[\mathcal{A}(\mathsf{iO}(1^\lambda, C_1)) = 1]\right|,\end{aligned}$$

  then we say the iO scheme is polynomially (or 2^{λ^ϵ} sub-exponentially resp.) secure if there exists some negligible function $\epsilon_{\mathcal{A}}(\lambda)$ such that $\mathsf{Adv}_{\mathsf{iO},C_0,C_1}(\lambda) \leq \epsilon_{\mathcal{A}}(\lambda)$ (or $2^{-\lambda^\epsilon}\epsilon_{\mathcal{A}}(\lambda)$ resp.) for any functionally equivalent circuits $C_0, C_1 \in \mathcal{C}$.