Obfuscating Non-Evasive Programs: Making Witness Encryption Positional

Shafik Nassar
UT Austin
shafik@cs.utexas.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

Indistinguishability obfuscation ($i\mathcal{O}$) is a powerful cryptographic tool that enables many cryptographic capabilities. Due to its expressivity, constructing $i\mathcal{O}$ is challenging, and existing constructions based on well-founded assumptions all require multiple algebraic assumptions, including an assumption on bilinear groups. If we consider post-quantum constructions, existing candidates all rely on new heuristic assumptions. Due to the challenges in realizing $i\mathcal{O}$, a parallel line of work has aimed to build obfuscation for restricted classes of functionalities, such as point functions, compute-and-compare programs, and most broadly, null circuits (i.e., circuits that always output $\perp$) from weaker assumptions. Thus far, these techniques have all been limited to supporting *evasive* programs (i.e., programs where it is difficult for an evaluator to find an input where the program's output is not $\perp$).

In this work, we introduce a new notion called cutoff-$i\mathcal{O}$. In cutoff-$i\mathcal{O}$, we can obfuscate a circuit C together with a *secret* cutoff t. Then, given an input (w, i), the obfuscated program outputs $C(w, i)$ if $i > t$ and $\perp$ otherwise. Security essentially stipulates that the obfuscated program should hide the cutoff t. Cutoff-$i\mathcal{O}$ is an example of obfuscation for a *non-evasive* function class and implies notions like *positional witness encryption*, which was previously only known from $i\mathcal{O}$. Our main result is showing how to construct cutoff-$i\mathcal{O}$ from witness encryption and the learning with errors (LWE) assumption. Thus, our approach shows how to lift an obfuscation scheme for an evasive function class (e.g., witness encryption, and more broadly, null-$i\mathcal{O}$) to an obfuscation scheme for a non-evasive function class.

By relying on the implication to positional witness encryption, we obtain (from the same set of assumptions) a collusion-resistant broadcast-and-trace scheme with public tracing and ciphertext size $N^{\varepsilon} \cdot \mathsf{poly}(\lambda)$, where N is the number of users in the system and $\varepsilon > 0$ is an arbitrary constant. Previously, such an implication was only known from $i\mathcal{O}$ (or from bilinear groups for the special case where $\varepsilon = 1/2$). Our construction also extends to the registration-based setting where users can sample their own keys and there is no central key-issuing authority.

Additionally, we show that cutoff-$i\mathcal{O}$ and LWE can be used to obtain a somewhere-*statistically* correlation-intractable hash function for efficiently enumerable relations as well as a non-interactive batch argument for NP (BARG) with somewhere-*statistical* soundness.

Contents

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01, GGH+13] is a powerful tool in cryptography. Starting from the work of Sahai and Waters [SW14], the research community has now shown how to realize nearly all cryptographic primitives using indistinguishability obfuscation together with one-way functions; we refer to [JLS21, Corollary 1.1] for a non-exhaustive list. At the same time, constructing indistinguishability obfuscation is challenging, and despite over a decade of research, the only constructions of $i\mathcal{O}$ based on well-founded assumptions rely on a combination of multiple number-theoretic assumptions [JLS21, JLS22, RVV25]. Notably, all of these constructions critically rely on security assumptions over pairing groups, and thus, would become insecure against quantum adversaries. Existing approaches for post-quantum $i\mathcal{O}$ have typically relied on new heuristic assumptions [BDGM20, GP21, WW21, DQV+21, BDGM22, HJL25, CLW25]. For many of the earlier schemes in this line of work, subsequent works have identified counterexamples or attacks against their underlying hardness assumptions [HJL21, JLLS23, BDJ+25]. Thus, post-quantum $i\mathcal{O}$ still rests on shaky foundations.

Witness encryption and evasive functions. Given the challenges in building $i\mathcal{O}$, another line of work has focused on realizing the applications of $i\mathcal{O}$ from simpler tools. One such tool is witness encryption [GGSW13]. In a witness encryption scheme, one can use an arbitrary NP statement x (for an NP relation $\mathcal{R}$) as the public key to encrypt a message μ. The associated witness w serves as the decryption key. In other words, we can view a witness encryption ciphertext as an obfuscation of the program $P_{x,\mu}(w)$ which takes as input an NP witness w and outputs μ if $\mathcal{R}(x, w) = 1$ and $\perp$ otherwise. The statement x and the message μ are hard-wired inside the description of the program $P_{x,\mu}$. The security requirement says that if the statement x is false, then the ciphertext (i.e., the obfuscation of $P_{x,\mu}$) computationally hides the message μ. Observe that when the statement x is false, the program $P_{x,\mu}$ outputs $\perp$ on *all* inputs. In this case, we say that $P_{x,\mu}$ computes an *evasive* function [BBC+14]. This is a key difference between indistinguishability obfuscation and witness encryption: $i\mathcal{O}$ requires indistinguishability between the obfuscations of any pair of functionally equivalent programs, whereas witness encryption only requires indistinguishability for a restricted subset of evasive programs. Under the plain learning with errors (LWE) assumption, it is possible to upgrade witness encryption to null-$i\mathcal{O}$, which asks that the obfuscations of two *null* programs (i.e., programs that output $\perp$ on all inputs) be indistinguishable [WZ17, GKW17]. Again, this falls into the regime of obfuscating evasive functionalities.

Restricting our attention to obfuscating specific evasive functionalities simplifies the problem greatly.[1] For instance, the work of [GMM17] provides a formal black-box separation between witness encryption and $i\mathcal{O}$. From the constructions perspective, multiple works have proposed direct constructions of witness encryption from lattice-based assumptions [CVW18, Tsa22, VWW22], affine determinant programs [BIJ+20, SRG+26], or a hypothetical approach based on groups [BIOW20]. The lattice-based and affine-determinant-program-based approaches also provide a plausible route towards concretely efficient constructions. At the same time, the lattice-based techniques critically rely on the fact that witness encryption corresponds to an evasive functionality, and do not appear to extend to $i\mathcal{O}$.

Beyond evasive functions: positional witness encryption. Despite the progress on basing advanced cryptographic notions on witness encryption (cf. [GGSW13, BISW18, FWW23, WW24, JKLM25]), there still remains a large gap between the capabilities of $i\mathcal{O}$ and those of witness encryption. The fundamental reason lies in the fact that witness encryption and null-$i\mathcal{O}$ only support evasive functionalities, whereas the main techniques for unlocking the power of $i\mathcal{O}$ critically rely on the ability to obfuscate *non-evasive* programs. Thus, a natural goal towards $i\mathcal{O}$ is to design obfuscation schemes that can support interesting non-evasive function classes.

A natural target in this intermediate regime between witness encryption and full-fledged $i\mathcal{O}$ is *positional witness encryption* [GLW14, GVW19]. In this setting, one can encrypt a message μ with respect to an NP relation $\mathcal{R}$, an NP statement x *and* a *secret* cutoff index $t \in [0, 2^m]$. To decrypt, one must provide a valid witness w for the statement x, and moreover, we require that $w > t$. Thus, we can view the ciphertext as an obfuscation of the program $P_{x,\mu,t}(w)$ where

$$P_{x,\mu,t}(w) = \begin{cases} \mu & \mathcal{R}(x, w) = 1 \wedge w > t \\ \perp & \text{otherwise.} \end{cases}$$

[1] In fact, obfuscation schemes for special sub-classes of evasive functions like point functions [Can97, Wee05, CD08, Zha16], hyperplane membership [CRV10], or compute-and-compare programs [WZ17, GKW17] can be built from groups or lattices.

The security requirement is that if $\mathcal{R}(x,t)=0$, then an encryption of μ with respect to $(x,t-1)$ is computationally indistinguishable from an encryption with respect to (x,t). Or equivalently, the obfuscations of the programs $P_{x,\mu,t-1}$ and $P_{x,\mu,t}$ are computationally indistinguishable whenever $\mathcal{R}(x,t)=0$. This property is referred to as *index indistinguishability*. Observe that the programs $P_{x,\mu,t-1}$ and $P_{x,\mu,t}$ are *not* evasive (i.e., there can be many inputs where both programs output μ). Because the underlying functionality is non-evasive, positional witness encryption is generally considered to be a primitive that is closer in power to $i\mathcal{O}$ than to witness encryption (or null-$i\mathcal{O}$). The increased expressivity of positional witness encryption also enables new applications. For instance, the work of [GVW19] shows how to design a broadcast-and-trace scheme with public tracing from positional witness encryption. Such an implication is not known from plain witness encryption.

This work: obfuscating non-evasive circuits using witness encryption. Our goal in this work is to bridge the apparent gap between witness encryption and positional witness encryption. Namely, we show how to take a witness encryption scheme and combine it with the plain learning with errors (LWE) assumption to obtain a positional witness encryption scheme (with a polynomial-size index space; see Section 1.1). In other words, using LWE, we are able to lift an obfuscation scheme that only supports *evasive* functions to one that supports a large *non-evasive* function class. To our knowledge, this is the first example of building an obfuscation scheme that supports a non-evasive function class and which does not rely on $i\mathcal{O}$ (or something stronger). As immediate consequences of our result, we show how to obtain new (registered) broadcast-and-trace schemes as well as a statistically correlation-intractable hash function from plain witness encryption together with LWE.

1.1 Our Results

We start by defining a relaxation of positional witness encryption, which we call *indexed positional witness encryption*. Here, we consider NP relations where the witness is of the form $(w,i)\in\{0,1\}^m\times[N]$. We refer to m as the witness length and N as the size of the witness index space. In an indexed positional witness encryption scheme, we compare the secret cutoff index $t\in[0,N]$ with the witness index $i\in[N]$. The difference with standard positional witness encryption is that we only compare the index i with the cutoff t as opposed to the full witness. As we show in this work, this relaxed notion already suffices for the applications of positional witness encryption from [GVW19] (in fact, the applications from [GVW19] were implicitly using the indexed version). More precisely, the ciphertext of an indexed positional witness encryption for a relation $\mathcal{R}$ can be viewed as an obfuscation of the program $P_{x,\mu,t}(w,i)$ where

$$P_{x,\mu,t}(w,i)=\begin{cases}\mu & \mathcal{R}(x,(w,i))=1\wedge i>t\\ \bot & \text{otherwise.}\end{cases}$$

The two main properties we require on an indexed positional witness encryption scheme are as follows:

- **Index indistinguishability.** The analogous notion of index indistinguishability now asks that if $\mathcal{R}\big(x,(w,t)\big)=0$ for every $w\in\{0,1\}^m$, then the obfuscations of $P_{x,\mu,t}$ and $P_{x,\mu,t-1}$ are computationally indistinguishable.
- **Succinctness.** In our constructions, we allow the ciphertext size to depend on the index space size N. We say that an indexed positional witness encryption scheme is $\eta(N)$-succinct if it has ciphertexts of length $\eta(N)\cdot\mathsf{poly}(\lambda,|C|,|\mu|,\log N)$, where λ is the security parameter and C is the circuit implementing the relation $\mathcal{R}$ for the appropriate input length.

The succinctness requirement provides a mechanism to interpolate between witness encryption and positional witness encryption:

- First, $\mathsf{poly}(\log N)$-succinct indexed positional witness encryption scheme is equivalent to standard positional witness encryption. Namely, in this setting, we can take $N=2^m$ and just take the index $i\in[2^m]$ as the witness itself. This coincides with a positional witness encryption with a $\mathsf{poly}(\lambda,|C|,|\mu|,\log N)$-size ciphertext.
- On the other extreme, an N-succinct indexed positional witness encryption is equivalent to plain witness encryption. Specifically, we can take an encryption of μ with cutoff t to consist of N independent witness encryption ciphertexts, where the i^{th} ciphertext is a witness encryption under the relation $\mathcal{R}_i(x,w):=\mathcal{R}(x,(w,i))$

of μ if $i > t$ and an encryption of $\perp$ otherwise. Index indistinguishability follows from semantic security of the witness encryption scheme (which we can invoke whenever $\mathcal{R}(x, (w, t)) = 0$ for all $w \in \{0,1\}^m$).

In this work, we consider the intermediate regime and show how to use plain witness encryption *and* the plain LWE assumption to construct an N^ε-succinct indexed positional witness encryption for any constant $\varepsilon \in (0, 1)$.

Theorem 1.1 (Indexed Positional Witness Encryption, Informal)**.** *Assume the existence of a polynomially secure witness encryption scheme and polynomial hardness of LWE (with a sub-exponential modulus-to-noise ratio). Then there exists a polynomial* $\mathsf{poly}(\cdot)$ *and a constant* $d > 1$ *such that for every constant* $\varepsilon \in (0, 1)$*, there exists an indexed positional witness encryption scheme with ciphertext size* $N^\varepsilon \cdot \mathsf{poly}(\lambda, |C|, |\mu|, \log N)^{d^{1/\varepsilon}}$*, where* λ *is the security parameter,* C *is the circuit computing the* NP *relation,* μ *is the encrypted message, and* N *is the size of the witness index space.*

The factor $d^{1/\varepsilon}$ in the exponent is a common term that arises in recursive constructions. Alternatively, we can say that for every constant $\varepsilon \in (0, 1)$, there exists a fixed polynomial $\mathsf{poly}_\varepsilon(\cdot)$ such that the indexed positional witness encryption has ciphertexts of size $N^\varepsilon \cdot \mathsf{poly}_\varepsilon(\lambda, |C|, |\mu|, \log N)$.

Broadcast-and-trace systems. As mentioned above, the NP relation used in the work of [GVW19] to construct broadcast-and-trace schemes with public tracing is implicitly using an indexed positional witness encryption scheme. As such, we can leverage their work in conjunction with Theorem 1.1 to obtain a broadcast-and-trace scheme from plain witness encryption and LWE. Very briefly, broadcast-and-trace systems [NP00, NNL01] are a hybrid of broadcast encryption [FN93] and traitor tracing [CFN94]. In this system, there are N users (identified by the indices $[N]$), and each user has a decryption key sk_i. One can encrypt a message to an arbitrary subset of users $S \subseteq [N]$ with a ciphertext whose size is sublinear in $|S|$. Moreover, if an arbitrary set of users S collude to build a "pirate" decryption device (i.e., a box that takes a ciphertext as input and outputs the associated message), the tracing algorithm can successfully identify at least one of the secret keys used to build the box. We say the scheme supports public tracing if anyone can run the tracing algorithm; otherwise, if a secret key is needed, then we say the scheme only supports private tracing. By combining our indexed positional witness encryption with the approach from [GVW19], we obtain the following corollary (see also Appendix B):

Corollary 1.2 (Broadcast-and-Trace, Informal)**.** *Assume the existence of a polynomially secure witness encryption scheme and polynomial hardness of LWE (with a sub-exponential modulus-to-noise ratio). Then there exists a polynomial* $\mathsf{poly}(\cdot)$ *and a constant* $d > 1$ *such that for every constant* $\varepsilon \in (0, 1)$*, there exists a collusion-resistant broadcast-and-trace scheme with public tracing and ciphertexts of size* $N^\varepsilon \cdot \mathsf{poly}(\lambda, \log N)^{d^{1/\varepsilon}}$*, where* N *is the number of users in the system.*

Prior to this work, the only constructions of broadcast-and-trace with public tracing either relied on $i\mathcal{O}$ or multilinear maps [GLW14] or bilinear maps [BW06]. The construction from [GLW14] achieves optimal ciphertext size (i.e., $\mathsf{poly}(\lambda, \log N)$), but requires the full power of $i\mathcal{O}$ (or multilinear maps). The construction from [BW06] achieves $\sqrt{N}$-size ciphertexts from bilinear maps. Our construction achieves N^ε-size ciphertexts from witness encryption and LWE.

Registered broadcast-and-trace. As a bonus, our construction can be extended to obtain a *registered* broadcast-and-trace system with the same ciphertext size. In the registration-based model [GHMR18, GHM+19, HLWW23], the central key-issuer (who issues the secret keys sk_i to each user) is replaced by an untrusted key curator. In this model, users can choose their own public/private keys and there is a public aggregation algorithm that takes a collection of public keys and aggregates them together to form the public key for the broadcast-and-trace system. The aggregation algorithm is public and deterministic. Registered broadcast-and-trace can be viewed as a trustless variant of traditional broadcast-and-trace. To our knowledge, this is the first construction of registered broadcast-and-trace. Prior works have only considered trustless versions of broadcast encryption (i.e., distributed or flexible broadcast encryption [WQZD10, BZ14, FWW23, KMW23, CW24, GKPW24, CHW25, WW25, GY26b, AMR26, GY26a]) without tracing capabilities or registered traitor tracing without the ability to broadcast to arbitrary subsets of users [BLM+24]. The registered traitor tracing scheme from [BLM+24] based on bilinear maps supports public tracing with $\sqrt{N}$-size ciphertexts. However, it does not support the broadcast functionality of being able to encrypt to an arbitrary subset of users. The registered broadcast-and-trace scheme we obtain via our indexed positional witness encryption satisfies the same succinctness as Corollary 1.2. We refer to Appendices C and D for the full details.

Correlation-intractable hash functions and batch arguments. We also show an application to proof systems and correlation-intractable hash functions [CGH98]. Formally, we show that indexed positional witness encryption suffices to lift a somewhere-statistically correlation-intractable hash for *search* relations [CCH+19, PS19] to a somewhere-statistically correlation-intractable hash for *enumerable* relations [CCH+19, HW26]. Recall first that we say a hash function H is correlation-intractable for a relation $\mathcal{R}$ if it is hard to find an input x such that $\mathcal{R}(x, H(x))$ holds. We say a relation $\mathcal{R}$ is a search relation if for every x, there exists at most 1 value w where $\mathcal{R}(x, w) = 1$, and we say $\mathcal{R}$ is an enumerable relation if for every x, there exist at most N values $y_1, \ldots, y_N$ where $\mathcal{R}(x, y_i) = 1$. Previously, the work of [HW26] showed how to use $i\mathcal{O}$ to lift a correlation-intractable hash function for efficient search relations into one for efficiently enumerable relations. In this work, we show that the same can be done using an indexed positional witness encryption scheme. In fact, even a non-succinct indexed positional witness encryption scheme is sufficient for this transformation. Thus, combined with the classic construction of Peikert and Shiehian [PS19] who showed how to build a somewhere-statistically correlation-intractable hash function for efficient search relations from the plain LWE assumption, we obtain the following corollary:

Theorem 1.3 (Somewhere-Statistically Correlation-Intractable Hash Function, Informal)**.** *Assume the existence of a polynomially secure witness encryption scheme and the polynomial hardness of LWE (with a sub-exponential modulus-to-noise ratio). Then, there exists a somewhere-statistically correlation-intractable hash for any efficiently enumerable* NP *relation.*

A direct application of a somewhere-statistically correlation-intractable hash function for efficiently enumerable relations is that it can be used to build a batch argument for NP (BARG) that is somewhere *statistically* sound. A batch argument for NP allows a prover to prove the veracity of N NP statements $x_1, \ldots, x_N$ with a proof whose size scales with $o(N) \cdot \mathsf{poly}(\lambda, |\mathcal{R}|)$, where $|\mathcal{R}|$ is the size of the Boolean circuit computing the NP relation. We say the BARG is somewhere statistically sound if the common reference string for the BARG can be programmed with a hidden index $i \in [N]$ with the property that there do *not* exist accepting proofs for any tuple of statements $(x_1, \ldots, x_N)$ where x_i is false. By plugging in a somewhere-statistically correlation-intractable hash function for efficiently enumerable relations into the classic BARG construction of Choudhuri, Jain, and Jin [CJJ21], we obtain a somewhere-statistically sound BARG (from witness encryption and LWE). The original scheme from [CJJ21] only satisfied somewhere computational soundness since the underlying correlation-intractable hash function (from LWE) only provided computational correlation-intractability for efficiently enumerable relations.

Prior to this work, the only constructions of somewhere statistically sound BARGs either rely on $i\mathcal{O}$ [HW26, Nas26], circular-secure FHE [HW26], or bilinear maps [WW22, CEW25, BFN26]. BARGs with somewhere statistical soundness are useful when combined with notions like witness encryption (cf. [DJWW25]).

1.2 Technical Overview

In this section, we give an overview of our indexed positional witness encryption scheme. Throughout, we work with circuits that output a value in $\{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$. A circuit is called a *null circuit* if it outputs $\bot$ on all inputs.[2] We start by introducing a new notion of obfuscation that naturally implies indexed positional witness encryption (analogous to the relationship between vanilla $i\mathcal{O}$ and vanilla witness encryption [GGH+13]). We refer to the notion as cutoff indistinguishability obfuscation (*cutoff-$i\mathcal{O}$*). A cutoff indistinguishability obfuscator consists of a pair of efficient algorithms $(\mathsf{Obf}, \mathsf{Eval})$ with the following syntax:

- The obfuscate algorithm Obf takes as input the security parameter 1^λ, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ and a cutoff index $t \in [0, N]$, and outputs an obfuscated program P.
- The evaluation algorithm Eval takes as input an obfuscated program P, an input $(w, i) \in \{0,1\}^{\ell_{\mathsf{in}}} \times [N]$ and outputs a value in $\{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$. Correctness requires the following to hold for all programs P in the support of $\mathsf{Obf}(1^\lambda, C, t)$:

$$\mathsf{Eval}(P, w, i) = \begin{cases} C(w, i) & i > t \\ \bot & i \le t. \end{cases}$$

[2] This is slightly different from the usual notion where a circuit only outputs a binary value and is considered null if it outputs 0 on all inputs. Still, the notion we use is equivalent and more convenient to work with.

By design, the obfuscated program is fully functional at $t = 0$ and is a null program at $t = N$.

We say that (Obf, Eval) is $\eta(N)$-succinct if the size of the obfuscated program P is $\eta(N) \cdot \mathsf{poly}(\lambda, |C|)$. The security properties of cutoff-$i\mathcal{O}$ are inherited from positional witness encryption. At a high level, we require circuit hiding when $t = N$ and index indistinguishability between t and $t+1$ when $C(\cdot, t+1)$ is a null circuit. Formally, we have:

- **Circuit hiding:** For any two circuits C_0, C_1 with the same type signature and of the same size, we have $\mathsf{Obf}(1^\lambda, C_0, N)$ and $\mathsf{Obf}(1^\lambda, C_1, N)$ are computationally indistinguishable.
- **Index indistinguishability:** For any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ and $t \in [0, N-1]$ such that $C(w, t+1) = \bot$ for all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, we have $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ are computationally indistinguishable.

Trivial cutoff-$i\mathcal{O}$ from null-$i\mathcal{O}$. Our first observation is that cutoff-$i\mathcal{O}$ with trivial succinctness (i.e., N-succinct cutoff-$i\mathcal{O}$) is directly implied by a null-$i\mathcal{O}$ scheme. Recall that a null-$i\mathcal{O}$ scheme [GKW17, WZ17] is an obfuscator that is functionality-preserving for all circuits, but guarantees indistinguishability only for null circuits. On input a circuit C and a cutoff $t \in [0, N]$, the obfuscate algorithm Obf for the cutoff-$i\mathcal{O}$ scheme defines N different programs $P_1, \ldots, P_N\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ where program P_j is defined to be

$$P_j \equiv \begin{cases} C(\cdot, j) & j > t \\ C_\bot & j \le t. \end{cases} \tag{1.1}$$

Here, $C_\bot$ is a canonical null circuit of the same size as $C(\cdot, j)$. Specifically, the programs P_j where $j \le t$ implement the null circuit while the programs where $j > t$ implement the original circuit. The algorithm Obf then obfuscates each program $P'_j \leftarrow \mathsf{NiO}(P_j)$ using the null-$i\mathcal{O}$ scheme NiO and outputs $P = (P'_1, \ldots, P'_N)$. We now consider the two security requirements:

- **Circuit hiding:** First, *perfect* circuit hiding follows by construction. Namely, given any two circuits C_0, C_1 with the same type signature, the programs $P_1, \ldots, P_N$ constructed by Eq. (1.1) are identical when the cutoff is N. Namely, in both cases, $P_i \equiv C_\bot$ for all $i \in [N]$.
- **Index indistinguishability:** For index indistinguishability, consider the distributions $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ when $C(w, t+1) = \bot$ for all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$. The only difference between these two distributions is the distribution of the program $P'_{t+1} \leftarrow \mathsf{NiO}(P_{t+1})$. By construction, $\mathsf{Obf}(1^\lambda, C, t)$ defines $P_{t+1}(w) = C(w, t+1)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ defines $P_{t+1}(w) = C_\bot(w)$. Since $C(w, t+1) = \bot$ for all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, this means the circuit P_{t+1} is also a null circuit. By security of NiO, this means $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ are computationally indistinguishable, as required.

Although this construction only satisfies trivial succinctness, it is already sufficient for applications that do not require succinctness. As an example, we show in Section 6 how to boost a somewhere-statistically correlation-intractable hash function for search relations [CCH+19] to a somewhere-statistically correlation-intractable hash function for enumerable relations [HLR21, HW26] using cutoff-$i\mathcal{O}$ (with trivial succinctness). Our approach is similar to that of Holmgren and Waters [HW26], but we use cutoff-$i\mathcal{O}$ instead of full-fledged $i\mathcal{O}$. In combination with the techniques from [CJJ21], this yields a somewhere *statistically* sound batch argument for NP (see Appendix A).

Cutoff-$i\mathcal{O}$ with non-trivial succinctness. For other applications (e.g., broadcast-and-trace), we require cutoff-$i\mathcal{O}$ with non-trivial succinctness (i.e., a cutoff-$i\mathcal{O}$ scheme with $o(N)$-succinctness). In this overview, we focus on achieving $\sqrt{N}$-succinctness, which suffices to illustrate most of the ideas underlying our design.

High-level approach. The basic approach is to divide the index space $[N]$ into k blocks each of size k, where $k = \sqrt{N}$. Then, we have k *block programs* $P'_1, \ldots, P'_k$, where each block program is responsible for a single block of k indices. In more detail, we write each index $i \in [N]$ as $(i_1, i_2) \in [k]^2$ such that $i = (i_1 - 1)k + i_2$. Throughout this overview, we identify an index with its decomposition; in particular, for $i_1, i_2 \in [k]$, we write $C(w, (i_1, i_2))$ to denote

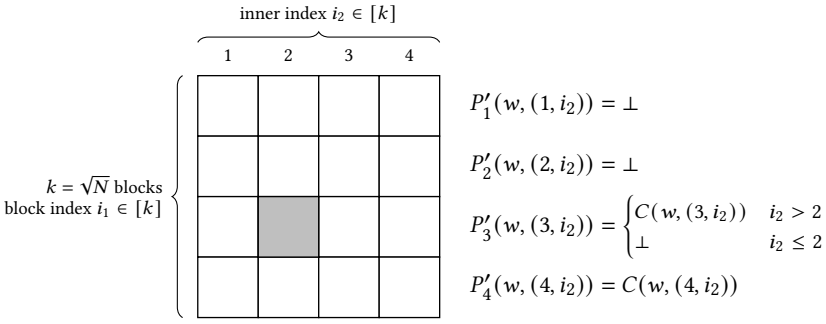

Figure 1: The index space $[N]$ viewed as a $k \times k$ grid, where $k = \sqrt{N}$. Here, $N = 16$ and $k = 4$. Each cell is a single index $i \in [N]$, written as $i = (i_1 - 1)k + i_2$ with $(i_1, i_2) \in [k]^2$. Each row $i_1 \in [k]$ is associated with a block program P'_{i_1} which implements evaluation for the inputs to that row. The shaded cell is the cutoff $t = (t_1 - 1)k + t_2$. In this example, $t_1 = 3$ and $t_2 = 2$.

$C(w, (i_1 - 1)k + i_2)$. To evaluate on an input $(w, (i_1, i_2))$, the evaluation algorithm computes $P'_{i_1}(w, (i_1, i_2))$. Every cutoff $t \in [0, N]$ can be uniquely written as $t = (t_1 - 1)k + t_2$ where $t_1 \in [k + 1]$ and $t_2 \in [0, k - 1]$. We refer to t_1 as the *target block* index. Our goal is to establish the following invariants:

- All block programs P'_{i_1} where $i_1 < t_1$ are *null programs*. Namely, $P'_{i_1}(w, (i_1, i_2)) = \bot$.
- All block programs P'_{i_1} where $i_1 > t_1$ are *fully functional programs*. Namely, for all w, i_2 we have $P'_{i_1}(w, (i_1, i_2)) = C(w, (i_1, i_2))$.
- Finally, the *target block program* P'_{t_1} should implement a cutoff-$i\mathcal{O}$ functionality for the index space $[(t_1 - 1) \cdot k + 1, t_1 \cdot k]$ with cutoff index t_2. Namely, for all w, i_2 we have

$$P'_{t_1}(w, (t_1, i_2)) = \begin{cases} C(w, (t_1, i_2)) & i_2 > t_2 \\ \bot & i_2 \le t_2 \end{cases}$$

We illustrate the general invariant in Fig. 1. Since there are k block programs, succinctness stipulates that the size of each block program P'_i cannot grow with k. Otherwise the total size of $P'_1, \ldots, P'_k$ would scale with $k^2 = N$, which is the same as the trivial scheme. At a high level, our construction uses a *single* auxiliary component H of size $O(k)$ that will allow the target block program P'_{t_1} to implement a trivial cutoff-$i\mathcal{O}$ for the target block. At the same time, the auxiliary component will not affect the functionality of the other block programs. Crucially, we show how to *reuse* the same auxiliary component for different block programs (which allows us to support different cutoffs).

The tension is that the auxiliary component is a *single* object of size k that is shared by all k block programs. At the same time, it should only induce the target block program P'_{t_1} to implement a cutoff-$i\mathcal{O}$ (with cutoff t_2). To implement this, we start by defining the default behavior of the block programs. Let $t = (t_1 - 1)k + t_2$ be the cutoff.

- For all $i_1 < t_1$, the block program $P'_{i_1}(w, (i_1, i_2))$ defaults to $\bot$.
- For all $i_1 \ge t_1$, the block program $P'_{i_1}(w, (i_1, i_2))$ defaults to $C(w, (i_1, i_2))$.

Observe that the size of each of the programs $P'_1, \ldots, P'_k$ is independent of k. Moreover, all of these programs compute the desired cutoff functionality except for P'_{t_1}. To fix this, we define the auxiliary component to include k auxiliary programs $D'_1, \ldots, D'_k$ designed as follows:

- Each of the auxiliary programs $D'_1, \ldots, D'_k$ takes the pair $(w, (i_1, i_2))$ as input and will either output $\bot$ to indicate that it is "inactive" or will output $C(w, (i_1, i_2))$ if it is "active."
- In the construction, each block program $P'_1, \ldots, P'_k$ first evaluates the auxiliary program $D'_{i_2}(w, (i_1, i_2))$ and outputs $\bot$ if D'_{i_2} is active. Otherwise, if D'_{i_2} is inactive on the input, the block program outputs its default value.

- We hard-code the target block index t_1 into *all* of the auxiliary programs $D'_1, \ldots, D'_k$ and set them to be inactive (i.e., output $\perp$) on all inputs $i_1 \neq t_1$. By construction, this means the behavior of the auxiliary programs only affects the output of the target block program P'_{t_1}.
- When $i_1 = t_1$, we use $D'_1, \ldots, D'_k$ to implement a trivial cutoff-$i\mathcal{O}$ with cutoff t_2. This ensures that P'_{t_1} implements a cutoff-$i\mathcal{O}$ with cutoff t_2. Concretely, this means D'_j should be inactive when $j > t_2$ (so P'_{t_1} outputs $C(w, (t_1, i_2))$) and active when $j \leq t_2$ (so P'_{t_1} outputs $\perp$).

Now, to ensure that the auxiliary programs hide the target block index t_1, we encrypt the auxiliary programs $D'_1, \ldots, D'_k$ under a symmetric key. We then hard-wire the decryption key inside the block programs $P'_1, \ldots, P_{k'}$.[3] Looking ahead to our recursive construction, we will adopt the convention where by default, we always encrypt the block programs under a symmetric key. This encryption key serves as an *evaluation key* for the encrypted program. The evaluation key for the top-level scheme is included as part of the obfuscated program, which enables program evaluation. The evaluation keys for the internal auxiliary schemes are not given out in the clear, but instead, hard-wired within the block programs of the outer scheme.

The main technical challenge. While the basic design is simple, implementing it securely is challenging. Specifically, in order to hide the cutoff index t, we encrypt the auxiliary programs $D'_1, \ldots, D'_k$ (recall the cutoff block index t_1 is hard-coded in these programs). However, in order for the target block program P'_{t_1} to make use of the auxiliary programs (which implement the underlying cutoff $i\mathcal{O}$ with cutoff t_2), it needs to be able to evaluate the auxiliary program and as such, it needs to know the decryption key used to encrypt $D'_1, \ldots, D'_k$. Embedding a secret key inside a program is straightforward using $i\mathcal{O}$ techniques (e.g., via punctured programming [SW14]), but the same techniques do *not* apply when we replace $i\mathcal{O}$ with null-$i\mathcal{O}$. Indeed, the power of punctured programming and $i\mathcal{O}$ hinges on the ability to obfuscate non-evasive programs. With null-$i\mathcal{O}$, we only have security when obfuscating null programs, but the target block program P'_{t_1} (which contains information about the decryption key) cannot be a null program because it is implementing a cutoff functionality. In the following, we describe a new technique that uses *nested* null-$i\mathcal{O}$ programs together with function-binding hash functions [FWW23] to bridge this gap. Nesting null-$i\mathcal{O}$ obfuscations gives us new degrees of freedom in the security analysis. For instance, an inner program could be a null program while the outer one is not or vice versa. In both cases, we are able to invoke null-$i\mathcal{O}$ security somewhere, even if the outer (or inner) functionality is itself not an evasive one. This mechanism enables us to lift a null-$i\mathcal{O}$ scheme into a cutoff-$i\mathcal{O}$ scheme.

Cutoff-$i\mathcal{O}$ construction. We now describe how to construct our auxiliary component and build a cutoff-$i\mathcal{O}$ scheme with $\sqrt{N}$-succinctness. Our construction of cutoff-$i\mathcal{O}$ will use the following building blocks:

- A null-$i\mathcal{O}$ scheme NiO.
- A CPA-secure symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where Gen is the key-generation algorithm, Enc is the encryption algorithm, and Dec is the decryption algorithm.
- A hash function $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Verify})$. Specifically, we consider a Merkle-style hash function[4] where the Setup algorithm outputs a succinct hash key hk and the Hash algorithm takes a vector of values $\mathbf{x} = (x_1, \ldots, x_k)$ and outputs a succinct digest dig together with succinct local openings $\pi_1, \ldots, \pi_k$ to the elements $x_1, \ldots, x_k$, respectively. The Verify algorithm checks the validity of a local opening π_i with respect to the claimed value x_i and the digest dig.

On input the security parameter λ, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\perp\}$, and a cutoff $t \in [0, k^2]$ decomposed as $t = (t_1 - 1)k + t_2$ with $t_1 \in [k+1]$ and $t_2 \in [0, k-1]$, the obfuscate algorithm Obf works as follows:

[3] Technically, we will only hard-wire it inside the block program P'_{t_1} associated with the target block t'_1. The behavior of the other block programs do not depend on the auxiliary programs.

[4] Technically, in our security analysis, we will need this hash function to be a function-binding hash function [FWW23], but for ease of exposition, it suffices to think about it as a Merkle hash for now.

1. **Auxiliary programs.** For each $j \in [k]$, let $D_{t_1,j} \colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ be the circuit C restricted to the j^{th} index of the target block:

$$D_{t_1,j}(w,i) = \begin{cases} C(w,i) & i = (t_1, j) \\ \bot & \text{otherwise.} \end{cases} \tag{1.2}$$

   Then, the obfuscate algorithm constructs the auxiliary programs $\widehat{D}_j$ as follows:

$$\widehat{D}_j \leftarrow \begin{cases} \mathsf{NiO}(D_{t_1,j}) & j \le t_2 \\ \mathsf{NiO}(C_\bot) & j > t_2. \end{cases}$$

   Note here that the indices j *greater than* t_2 correspond to null programs. As mentioned before, this means $\widehat{D}_j$ for $j > t_2$ is inactive whereas $\widehat{D}_j$ for $j \le t_2$ is active. Finally, it samples an *inner* encryption key $\mathsf{ek}' \leftarrow \mathsf{Gen}(1^\lambda)$ and for each $j \in [k]$, it computes the encrypted auxiliary program $D'_j \leftarrow \mathsf{Enc}(\mathsf{ek}', \widehat{D}_j)$.

2. **Auxiliary component.** Next, the obfuscate algorithm samples a hash key $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda)$ and computes a hash of the encrypted auxiliary programs:

$$(\mathsf{dig}, \pi_1, \ldots, \pi_k) = \mathsf{Hash}\big(\mathsf{hk}, (D'_1, \ldots, D'_k)\big).$$

   The auxiliary component H consists of the encrypted auxiliary programs together with their openings (with respect to dig):

$$H = \big((D'_1, \pi_1), \ldots, (D'_k, \pi_k)\big).$$

3. **Inner programs.** Before constructing the block programs, the obfuscate algorithm first constructs another sequence of inner programs. Specifically, for each $j \in [k]$, it computes

$$G'_j \leftarrow \begin{cases} \mathsf{NiO}(C_\bot) & j \ne t_1 \\ \mathsf{NiO}(G_{t_1}) & j = t_1, \end{cases}$$

   where G_{t_1} is the following program:

> **Constants:** A hash key hk, a digest dig, a block index $j \in [k]$, and an inner evaluation key ek'.
> **Inputs:** An input w, an index $i \in [N]$, an auxiliary program D', and an opening π.
>
> 1. Write $i = (i_1, i_2)$ where $i_1, i_2 \in [k]$ (i.e., $i = (i_1 - 1)k + i_2$).
> 2. If $i_1 \ne j$ or $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i_2, D', \pi) = 0$, output $\bot$.
> 3. Compute $D = \mathsf{Dec}(\mathsf{ek}', D')$ and output $D(w, i)$.

Figure 2: Program G_j.

   Crucially, the inner evaluation key ek' only appears inside G'_{t_1}.

4. **Block programs.** Next, for each $j < t_1$, the obfuscate algorithm computes $P'_j \leftarrow \mathsf{NiO}(C_\bot)$. For $j \ge t_1$, it computes $P'_j \leftarrow \mathsf{NiO}(P_j)$ for the following program P_j which has G'_j hard-wired into it:

Constants: A circuit C, a hash key hk, a digest dig, a block index $j \in [k]$, and an obfuscated program G'_j.
Inputs: An input w, an index $i \in [N]$, an auxiliary program D', and an opening π.

1. Write $i = (i_1, i_2)$ where $i_1, i_2 \in [k]$ (i.e., $i = (i_1 - 1)k + i_2$).
2. If $i_1 \neq j$ or $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i_2, D', \pi) = 0$, output $\perp$.
3. If $G'_j(w, i, D', \pi) = \perp$, output $C(w, i)$. Otherwise, output $\perp$.

Figure 3: Program P_j.

Finally, it samples an encryption key $\mathsf{ek} \leftarrow \mathsf{Gen}(1^\lambda)$ and encrypts each block program under its own encryption key:

$$\forall j \in [k] \ : \ \mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{ek}, P'_j).$$

5. Finally, it outputs the obfuscated program $P = \big(\mathsf{ek}, H, \mathsf{ct}_1, \ldots, \mathsf{ct}_k\big)$.

To evaluate on an input $(w, (i_1, i_2))$, the evaluation algorithm reads the pair (D'_{i_2}, π_{i_2}) from the auxiliary component $H = ((D'_1, \pi_1), \ldots, (D'_k, \pi_k))$, recovers $P'_{i_1} = \mathsf{Dec}(\mathsf{ek}, \mathsf{ct}_{i_1})$, and outputs $P'_{i_1}(w, i, D'_{i_2}, \pi_{i_2})$. Since ek is published as part of P, this last encryption layer can technically be omitted. However, including it will be conducive for *recursive* composition.

Correctness. Suppose we evaluate $P = (\mathsf{ek}, H, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ on an input (w, i) where $i = (i_1, i_2)$. Let t be the cutoff and as usual, write $t = (t_1 - 1)k + t_2 \in [0, k^2]$ with $t_2 \in [0, k-1]$ and $t_1 \in [k+1]$. The evaluation algorithm decrypts ct_{i_1} under ek to recover P'_{i_1} and outputs $P'_{i_1}(w, i, D'_{i_2}, \pi_{i_2})$. Since π_{i_2} is the honest opening of D'_{i_2} with respect to dig, neither the block check nor the opening check in P_{i_1} will reject. We now consider three cases:

- Suppose $i_1 < t_1$. In this case, P'_{i_1} implements a null program, so the evaluation outputs $\perp$. This is what correctness requires, since $i = (i_1 - 1)k + i_2 \leq i_1 \cdot k \leq (t_1 - 1)k \leq t$.
- Suppose $i_1 > t_1$. In this case, P'_{i_1} is an obfuscation of P_{i_1}, but its inner program G'_{i_1} is an obfuscation of a null program (since $i_1 \neq t_1$). Thus $G'_{i_1}(w, i, D'_{i_2}, \pi_{i_2}) = \perp$ and the evaluation outputs $C(w, i)$. This means P'_{i_1} is a fully functional block program. This is again what correctness requires, since $i \geq (i_1 - 1)k + 1 \geq t_1 \cdot k + 1 > t$.
- Suppose $i_1 = t_1$. As in the previous case, the output is $P_{i_1}(w, i, D'_{i_2}, \pi_{i_2})$. By construction, P_{i_1} evaluates $G'_{i_1}(w, i, D'_{i_2}, \pi_{i_2})$. Since $i_1 = t_1$, this means G'_{i_1} is an obfuscation of G_{i_1}. The program G_{i_1} decrypts the auxiliary program D'_{i_2} under ek' to obtain the obfuscated program $\widehat{D}_{i_2}$ and then outputs $\widehat{D}_{i_2}(w, i)$. The output is then

$$P_{i_1}(w, i, D'_{i_2}, \pi_{i_2}) = \begin{cases} C(w, i) & \widehat{D}_{i_2}(w, i) = \perp \\ \perp & \widehat{D}_{i_2}(w, i) \neq \perp. \end{cases}$$

  Since $i_1 = t_1$, we have $i > t$ exactly when $i_2 > t_2$, and there are two sub-cases to consider:

  - If $i_2 > t_2$, then $\widehat{D}_{i_2}$ is a null program, so $\widehat{D}_{i_2}(w, i) = \perp$ and P_{i_1} outputs $C(w, i)$, as required.
  - If $i_2 \leq t_2$, then $\widehat{D}_{i_2}$ is an obfuscation of D_{t_1, i_2}, and $D_{t_1, i_2}(w, i) = C(w, i)$ because $i = (t_1, i_2)$. Now we again consider two possibilities:
    * If $C(w, i) \neq \perp$, then $D_{t_1, i_2}(w, i) \neq \perp$ so P_{i_1} outputs $\perp$.
    * If $C(w, i) = \perp$, then $D_{t_1, i_2}(w, i) = \perp$, so P_{i_1} outputs $C(w, i) = \perp$.

    Taken together, we see that P_{i_1} always outputs $\perp$ when $i_1 = t_1$ and $i_2 \leq t_2$ (i.e., whenever $i \leq t$). This is precisely the cutoff functionality.

At a very high level, observe that the block programs defined above precisely instantiate our basic template:

- The block programs P'_{i_1} where $i_1 < t_1$ are null programs and those where $i_1 > t_1$ are fully functional programs.
- When $i_1 = t_1$, the behavior of the block program P'_{i_1} depends on the inner program G_{t_1}. The combination of the inner program G_{t_1} and the auxiliary programs $D'_1, \ldots, D'_k$ induces a trivial cutoff-$i\mathcal{O}$ for the inner index t_2.

Succinctness. Next, we argue that the above construction satisfies $\sqrt{N}$-succinctness. Recall that $k = \sqrt{N}$.

- The evaluation key ek is a symmetric encryption key, so $|\mathsf{ek}| = \mathsf{poly}(\lambda)$ and is independent of $|C|$ and N.
- The auxiliary component H consists of the auxiliary programs $D'_1, \ldots, D'_k$, where each D'_j encrypts an obfuscation of either $D_{t_1,j}$ or a (padded) canonical null circuit. The size of $D_{t_1,j}$ is $\mathsf{poly}(|C|)$, so each D'_j has size $\mathsf{poly}(\lambda, |C|)$. The openings $\pi_1, \ldots, \pi_k$ are also succinct so the size of H is $k \cdot \mathsf{poly}(\lambda, |C|)$. Note that we have $|C| > \log k$, so we can absorb $\mathsf{poly}(\log k)$ terms into $\mathsf{poly}(|C|)$.
- Next, consider the size of each obfuscated program P'_j. By design, $|P'_j| = \mathsf{poly}(\lambda, |C|, |G'_j|)$. By construction, $|G'_j| = \mathsf{poly}(\lambda, |G_j|) = \mathsf{poly}(\lambda, |C|)$. We conclude then that $|P'_j| = \mathsf{poly}(\lambda, |C|)$, and encrypting it under ek adds a further $\mathsf{poly}(\lambda)$.

Taken altogether, the size of the obfuscated program is then $k \cdot \mathsf{poly}(\lambda, |C|)$, so the cutoff-$i\mathcal{O}$ satisfies $\sqrt{N}$-succinctness.

Perfect circuit hiding. Recall that circuit hiding says that for any pair of equal-size circuits C_0, C_1, their obfuscations $\mathsf{Obf}(1^\lambda, C_0, N)$ and $\mathsf{Obf}(1^\lambda, C_1, N)$ are computationally indistinguishable. Much like the trivial cutoff-$i\mathcal{O}$ scheme, our scheme above satisfies perfect circuit hiding. Specifically, when $t = N$ we have $t_1 = k + 1$ and $t_2 = 0$. In this case, all of the block programs $P'_1, \ldots, P'_k$, all of the inner programs $G'_1, \ldots, G'_k$, and all of the obfuscated programs $\widehat{D}_1, \ldots, \widehat{D}_k$ underlying the auxiliary programs are obfuscations of the canonical null circuit $C_\perp$ and do *not* depend on the circuit C. The keys ek and ek′ are sampled independently of C as well. Therefore, perfect circuit hiding holds.

Intra-block index indistinguishability. The second security requirement for a cutoff-$i\mathcal{O}$ scheme is index indistinguishability which asks that $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ be computationally indistinguishable whenever the circuit $C(\cdot, t+1)$ is a null circuit. We argue this in two cases, based on whether the cutoffs t and $t+1$ share the same target block or not. We start with the easier case where the cutoffs share the same target block. Let $t = (t_1 - 1)k + t_2$ where $t_2 \leq k - 2$. Then $t + 1 = (t_1 - 1)k + (t_2 + 1)$ has the same target block t_1. Index indistinguishability essentially reduces to the same argument used to argue index hiding of the trivial construction (recall that for the target block t_1, block program P'_{t_1} together with the auxiliary component effectively implement a trivial cutoff-$i\mathcal{O}$ scheme with cutoff t_2).

More formally, consider $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$. By construction, everything in the two distributions is sampled identically, except for the program $\widehat{D}_{t_2+1}$ associated with D'_{t_2+1}:

- $\mathsf{Obf}(1^\lambda, C, t)$ samples $\widehat{D}_{t_2+1} \leftarrow \mathsf{NiO}(C_\perp)$.
- $\mathsf{Obf}(1^\lambda, C, t+1)$ samples $\widehat{D}_{t_2+1} \leftarrow \mathsf{NiO}(D_{t_1,t_2+1})$.

By construction $D_{t_1,t_2+1}(w, i) = \perp$ for every $i \neq (t_1, t_2 + 1) = t + 1$. Moreover, $C(w, t+1) = \perp$ for all w which means $D_{t_1,t_2+1}(w, t+1) = C(w, t+1) = \perp$ for all w. Taken together, this means D_{t_1,t_2+1} is a null circuit. We can now invoke null-$i\mathcal{O}$ security to conclude that $\mathsf{NiO}(C_\perp)$ and $\mathsf{NiO}(D_{t_1,t_2+1})$ are computationally indistinguishable. Correspondingly, this means $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ are also computationally indistinguishable.

Inter-block index indistinguishability. We now turn to the second case of index indistinguishability, where the cutoffs t and $t + 1$ are associated with different target blocks. As before, we need to show that $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ are computationally indistinguishable whenever $C(w, t+1) = \perp$ for all w. Formally, suppose the cutoff t falls on a block boundary and assume $t_1 < k + 1$ (i.e., $t < N$). This means

$$\begin{aligned} t &= (t_1 - 1) \cdot k + (k - 1) && \text{decomposes as } (t_1, k-1) \\ t + 1 &= t_1 \cdot k && \text{decomposes as } (t_1 + 1, 0). \end{aligned}$$

In this case, the target block index advances from t_1 to $t_1 + 1$, while the inner cutoff wraps from $k - 1$ (i.e., the block t_1 has been fully cut off) back to 0 (i.e., the block $t_1 + 1$ has no cutoff). Before describing the hybrid argument, it is instructive to write out the two distributions and see exactly where they differ. By construction, only three groups of circuits are obfuscated differently by $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$:

- **The auxiliary programs** $D'_1, \ldots, D'_{k-1}$: In both distributions, $D'_j \leftarrow \mathsf{Enc}(\mathsf{ek}', \widehat{D}_j)$ where

$$\underbrace{\widehat{D}_j \leftarrow \mathsf{NiO}(D_{t_1,j})}_{\mathsf{Obf}(1^\lambda, C, t)} \qquad \text{versus} \qquad \underbrace{\widehat{D}_j \leftarrow \mathsf{NiO}(C_\perp)}_{\mathsf{Obf}(1^\lambda, C, t+1)}$$

  Note that $\widehat{D}_k$ is an obfuscation of the null circuit $C_\perp$ in both cases since $t_2 = k - 1 < k$ on the left and $t_2 = 0 < k$ on the right.

- **The block program** P'_{t_1}:

$$\underbrace{P'_{t_1} \leftarrow \mathsf{NiO}(P_{t_1}) \text{ where } G'_{t_1} \leftarrow \mathsf{NiO}(G_{t_1})}_{\mathsf{Obf}(1^\lambda, C, t)} \qquad \text{versus} \qquad \underbrace{P'_{t_1} \leftarrow \mathsf{NiO}(C_\perp)}_{\mathsf{Obf}(1^\lambda, C, t+1)}$$

  On the left, block t_1 is the target block, so the block program P'_{t_1} has the inner program G'_{t_1} hard-wired inside it. On the right, since $t_1 < t_1 + 1$, the block program P'_{t_1} becomes an obfuscation of the null program.

- **The inner program** G'_{t_1+1}:

$$\underbrace{G'_{t_1+1} \leftarrow \mathsf{NiO}(C_\perp)}_{\mathsf{Obf}(1^\lambda, C, t)} \qquad \text{versus} \qquad \underbrace{G'_{t_1+1} \leftarrow \mathsf{NiO}(G_{t_1+1})}_{\mathsf{Obf}(1^\lambda, C, t+1)}$$

  In both cases, the obfuscate algorithm sets $P'_{t_1+1} \leftarrow \mathsf{NiO}(P_{t_1+1})$, and the program P_{t_1+1} has the inner program G'_{t_1+1} hard-wired within it. On the left, block $t_1 + 1 \neq t_1$, so $G'_{t_1+1} \leftarrow \mathsf{NiO}(C_\perp)$. In contrast, on the right, $t_1 + 1$ is the target block index, so G'_{t_1+1} is an obfuscation of the circuit G_{t_1+1}.

To argue that these two distributions are computationally indistinguishable, our high-level strategy is as follows:

- First, the auxiliary programs $D'_1, \ldots, D'_k$ are *encryptions* of the obfuscated programs $\widehat{D}_1, \ldots, \widehat{D}_k$. The obfuscated programs themselves are never given out. Thus, we would hope to appeal to CPA-security of the encryption scheme to argue that auxiliary programs $D'_1, \ldots, D'_k$ output by $\mathsf{Obf}(1^\lambda, C, t)$ and $\mathsf{Obf}(1^\lambda, C, t+1)$ are computationally indistinguishable.

- However, we cannot directly appeal to the security of the encryption scheme. This is because in $\mathsf{Obf}(1^\lambda, C, t)$, the inner program G'_{t_1} depends on the key ek', and this program is in turn hard-wired within the block program P'_{t_1}. Similarly, in $\mathsf{Obf}(1^\lambda, C, t+1)$, the inner program G'_{t_1+1} depends on ek'. Thus, before we can invoke security of the encryption scheme, we need to first argue that the inner evaluation key is computationally hidden. For this, we will need to rely on null-$i\mathcal{O}$ security. To use null-$i\mathcal{O}$ security, we must first ensure that the underlying obfuscated programs are indeed null programs.

Note that the *outer* evaluation key ek is handed to the adversary as part of the obfuscated program, so it plays no role here; only ek' has to be hidden, and only in the middle of the argument. We now illustrate our hybrid sequence. For ease of exposition, we start from the distribution in $\mathsf{Obf}(1^\lambda, C, t+1)$ and show how to transform it to the distribution $\mathsf{Obf}(1^\lambda, C, t)$. We assume that for all w, we have $C(w, t+1) = \perp$. Our hybrid structure proceeds as follows:

- **The inner program** G'_{t_1+1}**.** First, we switch G'_{t_1+1} from an obfuscation of G_{t_1+1} to an obfuscation of the null circuit $C_\perp$. To appeal to null-$i\mathcal{O}$ security, we need G_{t_1+1} to be a null circuit. Take any input (w, i, D', π) where $i = (t_1 + 1, i_2)$ and the opening verifies (on all other inputs, G_{t_1+1} outputs $\perp$). Since the digest dig is a hash of the auxiliary programs $D'_1, \ldots, D'_k$, and every $\widehat{D}_j$ in $\mathsf{Obf}(1^\lambda, C, t+1)$ is an obfuscation of $C_\perp$, binding of the hash function says that D' must decrypt to an obfuscation of $C_\perp$.[5] Thus, G_{t_1+1} outputs $\perp$ on all inputs.

  After this step, the block program P'_{t_1+1} is an obfuscation of P_{t_1+1} where the inner program G'_{t_1+1} is an obfuscation of $C_\perp$. This matches the distribution in $\mathsf{Obf}(1^\lambda, C, t)$. Moreover, the inner program G'_{t_1+1} no longer depends on the inner evaluation key ek'.

[5] If the hash function is just a plain Merkle hash, this is not true. However, as we discuss below, we can replace the Merkle hash with a stronger *function-binding hash function* [FWW23] to guarantee this property.

- **The auxiliary programs $D'_1, \ldots, D'_{k-1}$.** Now that we have erased the encryption key ek' from the inner program G'_{t_1+1}, we can appeal to CPA-security of the encryption scheme and switch each auxiliary program D'_j where $j < k$ from an encryption of $\mathsf{NiO}(C_\perp)$ to an encryption of $\mathsf{NiO}(D_{t_1,j})$. The last one, D'_k, is unchanged.

- **The block program P'_{t_1}.** Finally, we switch the block program P'_{t_1} from an obfuscation of $C_\perp$ to an obfuscation of P_{t_1}, where P_{t_1} has the inner program $G'_{t_1} \leftarrow \mathsf{NiO}(G_{t_1})$ hard-wired inside it. Here, we again rely on null-$i\mathcal{O}$ security. To do so, we need to make sure the program P_{t_1} we are obfuscating is a null circuit.

  Take any input (w, i, D', π) where $i = (t_1, i_2)$ and the opening verifies (on all other inputs, P_{t_1} outputs $\perp$). When the opening verifies, we appeal to the binding property of the hash function to conclude that D' must be the auxiliary program D'_{i_2}. In this case, the inner program G_{t_1} outputs $\widehat{D}_{i_2}(w, i)$. There are two cases:

  - If $i_2 < k$, then $\widehat{D}_{i_2}$ is an obfuscation of D_{t_1,i_2}, and $D_{t_1,i_2}(w, i) = C(w, i)$ because $i = (t_1, i_2)$. Then, by definition of G'_{t_1}, we have
    $$G'_{t_1}(w, i, D', \pi) = D_{t_1,i_2}(w, i) = C(w, i).$$
    We have two possibilities. If $C(w, i) = \perp$, then $G'_{t_1}(w, i, D', \pi) = \perp$, and P_{t_1} also outputs $\perp$. If $C(w, i) \neq \perp$, then P_{t_1} outputs $\perp$. Thus, in this case, P_{t_1} always outputs $\perp$.
  - If $i_2 = k$, then $\widehat{D}_k$ is an obfuscation of $C_\perp$, so $G'_{t_1}(w, i, D', \pi) = \perp$. In this case, P_{t_1} outputs $C(w, (t_1, k)) = C(w, t + 1) = \perp$ by assumption. Note that the latter holds because $t + 1 = (t_1 - 1) \cdot k + k$, which coincides with the index (t_1, k).

  In both cases P_{t_1} outputs $\perp$, so we conclude that P_{t_1} is a null circuit, and the claim now follows by security of the null-$i\mathcal{O}$ scheme. Observe that this step reintroduces the inner evaluation key ek' which is now hard-wired inside the inner program G'_{t_1}. The crux of our argument is that this step does *not* invoke null-$i\mathcal{O}$ security on the inner program G_{t_1}, which is not necessarily a null circuit. Instead, we invoke null-$i\mathcal{O}$ security for the *outer* block program P_{t_1}. This is the reason we decompose the block program into the program P_{t_1} and a separate inner program G_{t_1}.

After this sequence of steps, we obtain the distribution $\mathsf{Obf}(1^\lambda, C, t)$. The one gap in this overview is in how we leverage the binding property of the hash function. Specifically, in the first step, we are demanding that the (honestly generated) digest dig can *only* be locally opened to an auxiliary program D' that encrypts an obfuscation of $C_\perp$. Similarly, in the third step, we are demanding that the digest can only be locally opened to the honest auxiliary program D'_{i_2} at index i_2. This is a much stronger property than the standard computational binding property in a Merkle-style hash function which just says that the adversary cannot come up with openings to two different messages at any individual index. Here, we are asking for the stronger property that the *only* openings available at each index are to a value with certain structural properties. To ensure this, we instantiate the hash function with a function-binding hash function [FWW23]. We first recall the concept:

- A function-binding hash function inherits the interface of the Merkle-style hash function $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Verify})$. In addition, it has a binding setup algorithm which takes as input the description of an efficiently computable function $f\colon [k] \times \mathcal{X} \to \{0, 1\}$. Here k is the number of blocks of input we are hashing and each block is an element of $\mathcal{X}$. In our construction, each block is an auxiliary program D'_j.

- The first security requirement is mode indistinguishability which asks that the normal hash key be computationally indistinguishable from a binding hash key.

- When we use the binding hash key to hash a vector $(D'_1, \ldots, D'_k)$ where $f(j, D'_j) = 1$ for all $j \in [k]$, then the digest *perfectly* binds all blocks to satisfy the function. In particular, this means that whenever the hash key is binding for f, then for every index j^* and every block D' that the digest can be opened to at index j^*, it must be that $f(j^*, D') = 1$.

If we instantiate the hash function in our construction with a function-binding hash function, we can now carry out the proof strategy sketched above. We illustrate the approach with the full sequence of hybrids:

Stage 1: Switching the inner program G'_{t_1+1}.

- Hyb_0: This is the distribution $\mathsf{Obf}(1^\lambda, C, t+1)$.
- Hyb_1: Same as Hyb_0, except we generate the auxiliary programs D'_j using randomness derived from a pseudorandom function PRF. In more detail, in this experiment, the challenger samples a PRF key k_{PRF} and sets $D'_j \leftarrow \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(C_\perp); \mathsf{PRF}(k_{\mathsf{PRF}}, j)\big)$, where the pseudorandom coins supply *both* the obfuscation and the encryption randomness. Indistinguishability follows by security of PRF.

  This step allows the subsequent hybrid to *recompute* the auxiliary program D'_j just from the index j (and the keys $\mathsf{ek}', k_{\mathsf{PRF}}$). In conjunction with the function-binding hash function, this will bind the adversary to only opening the digest dig of $(D'_1, \ldots, D'_k)$ to auxiliary programs which encrypt an obfuscation of $C_\perp$.
- Hyb_2: Same as Hyb_1, except the hash key for the function-binding hash function is sampled in binding mode with respect to the function $f_{\mathsf{ek}', k_{\mathsf{PRF}}}$ where

$$f_{\mathsf{ek}', k_{\mathsf{PRF}}}(j, D') = 1 \quad \Longleftrightarrow \quad D' = \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(C_\perp); \mathsf{PRF}(k_{\mathsf{PRF}}, j)\big).$$

  Note that the key ek' and the PRF key k_{PRF} are hard-wired inside $f_{\mathsf{ek}', k_{\mathsf{PRF}}}$, and that since the coins are fixed by the PRF, the right-hand side is a single well-defined string.

  Indistinguishability follows by mode indistinguishability. In this experiment, each auxiliary program D'_j is by construction equal to $\mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(C_\perp); \mathsf{PRF}(k_{\mathsf{PRF}}, j)\big)$. This means $f_{\mathsf{ek}', k_{\mathsf{PRF}}}(j, D'_j) = 1$ for all $j \in [k]$. The function-binding property now ensures that for all indices $j \in [k]$, the digest dig can only be opened to the auxiliary program D'_j,[6] which encrypts an obfuscation of $C_\perp$.
- Hyb_3: Same as Hyb_2, except $G'_{t_1+1} \leftarrow \mathsf{NiO}(C_\perp)$. This follows from null-$i\mathcal{O}$ security as long as the circuit G_{t_1+1} is a null circuit, which we argued above. Specifically, we rely on the function-binding hash function to ensure that each auxiliary program D'_j decrypts to an obfuscation of $C_\perp$. This means G_{t_1+1} is a null circuit.

Stage 2: Switching the auxiliary programs $D'_1, \ldots, D'_{k-1}$.

- Hyb_4: Same as Hyb_3, except the hash key for the function-binding hash function is sampled in normal mode. Indistinguishability again follows by mode indistinguishability of the function-binding hash function. At this point, the auxiliary programs $D'_1, \ldots, D'_k$ are the only components of Hyb_4 that depend on the inner evaluation key ek'.
- Hyb_5: Same as Hyb_4, except $D'_j \leftarrow \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(D_{t_1,j}); \mathsf{PRF}(k_{\mathsf{PRF}}, j)\big)$ for each $j < k$. The last one, D'_k, still encrypts $\mathsf{NiO}(C_\perp)$. Indistinguishability now follows by CPA-security of the encryption scheme (since nothing else in this experiment depends on ek').

Stage 3: Switching the block program P'_{t_1}.

- Hyb_6: Same as Hyb_5, except the hash key for the function-binding hash function is sampled in binding mode with respect to the function $f_{\mathsf{ek}', k_{\mathsf{PRF}}, C, t_1}$ where

$$f_{\mathsf{ek}', k_{\mathsf{PRF}}, C, t_1}(j, D') = 1 \quad \Longleftrightarrow \quad D' = \begin{cases} \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(D_{t_1,j}); \mathsf{PRF}(k_{\mathsf{PRF}}, j)\big) & j < k \\ \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(C_\perp); \mathsf{PRF}(k_{\mathsf{PRF}}, j)\big) & j = k. \end{cases}$$

  The key ek', the PRF key k_{PRF}, the circuit C, and the target block index t_1 are hard-wired in $f_{\mathsf{ek}', k_{\mathsf{PRF}}, C, t_1}$. Note that $f_{\mathsf{ek}', k_{\mathsf{PRF}}, C, t_1}$ can construct $D_{t_1,j}$ from C and t_1 according to Eq. (1.2). Thus, the function $f_{\mathsf{ek}', k_{\mathsf{PRF}}, C, t_1}$ can be computed by a Boolean circuit of size $\mathsf{poly}(\lambda, |C|)$.

[6] It may seem surprising that a single succinct digest is *statistically* binding on all indices simultaneously. The reason this is possible is because the long string that we are binding on (i.e., the string $(D'_1, \ldots, D'_k)$) is a pseudorandom string with a succinct description (i.e., can be described by a PRF key). The digest is essentially binding to the key for the PRF that generates $(D'_1, \ldots, D'_k)$, which has a succinct description independent of k.

Indistinguishability follows by mode indistinguishability. In this experiment, each block hashed is exactly the auxiliary program D'_j. By construction, this means $f_{\mathsf{ek}',k_{\mathsf{PRF}},C,t_1}(j, D'_j) = 1$ for all $j \in [k]$. The function-binding property now ensures that the digest dig can only be opened at an index j to D'_j itself.

- Hyb_7: Same as Hyb_6, except $P'_{t_1} \leftarrow \mathsf{NiO}(P_{t_1})$, where the circuit P_{t_1} has the inner program $G'_{t_1} \leftarrow \mathsf{NiO}(G_{t_1})$ hard-wired inside it. This follows from null-$i\mathcal{O}$ security as long as the circuit P_{t_1} is a null circuit, which we argued above. Specifically, we rely on the function-binding hash function to ensure that the only opening at index i_2 is the auxiliary program D'_{i_2}, and the case analysis from before shows that P_{t_1} outputs $\perp$ on all inputs.

Stage 4: Cleaning up.

- Hyb_8: Same as Hyb_7, except the hash key for the function-binding hash function is sampled in normal mode. Indistinguishability again follows by mode indistinguishability of the function-binding hash function.

- Hyb_9: Same as Hyb_8, except we generate the auxiliary programs D'_j using fresh randomness instead of using randomness derived from the PRF. Indistinguishability follows by PRF security. At this point, the auxiliary programs satisfy $D'_j \leftarrow \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(D_{t_1,j})\big)$ for each $j < k$ and $D'_k \leftarrow \mathsf{Enc}\big(\mathsf{ek}', \mathsf{NiO}(C_\perp)\big)$, so the output of this experiment is distributed according to $\mathsf{Obf}(1^\lambda, C, t)$.

We conclude that $\mathsf{Obf}(1^\lambda, C, t+1)$ and $\mathsf{Obf}(1^\lambda, C, t)$ are computationally indistinguishable and index indistinguishability holds.

Achieving $N^{1/c}$-succinctness. The construction above shows how to obtain a cutoff-$i\mathcal{O}$ with $\sqrt{N}$-succinctness, but we can view it more generally as rebalancing the trivial cutoff-$i\mathcal{O}$. Namely, we start with the trivial N-succinct cutoff-$i\mathcal{O}$ scheme that consists of N block programs, each in charge of a single index. Our construction above effectively rebalances the index space $[N]$ into a $\sqrt{N}$-by-$\sqrt{N}$ grid, such that each of the $\sqrt{N}$ blocks can potentially implement a cutoff-$i\mathcal{O}$ for the $\sqrt{N}$ indices it controls. The crux of the construction is that we do not need $\sqrt{N}$ independent copies of the underlying scheme. Instead, we leverage the fact that for any given cutoff t, only one of the $\sqrt{N}$ block programs needs to be a cutoff-$i\mathcal{O}$. As such, we can use a single auxiliary component H to implement the cutoff-$i\mathcal{O}$ for that particular target block. Taken together, this yields a cutoff-$i\mathcal{O}$ with $\sqrt{N}$-succinctness.

More generally, we can use the same rebalancing technique to compile an $N^{1/c}$-succinct cutoff-$i\mathcal{O}$ and obtain an $N^{1/(c+1)}$-succinct cutoff-$i\mathcal{O}$. Namely, we divide the index space $[N]$ into $N^{1/(c+1)}$ blocks, each of size $N' = N^{c/(c+1)}$. We associate each of the $N^{1/(c+1)}$ blocks with a block program that can potentially implement an $N^{1/c}$-succinct cutoff-$i\mathcal{O}$ for its block of indices. As before, we show how to use a single auxiliary component H to implement all $N^{1/(c+1)}$ block programs. Since we are effectively applying a cutoff-$i\mathcal{O}$ with $N^{1/c}$-succinctness to a system with $N' = N^{c/(c+1)}$ indices, the auxiliary component has size $(N')^{1/c} = N^{1/(c+1)}$. As before, the obfuscated program also includes $N^{1/(c+1)}$ block programs. The total size is therefore $N^{1/(c+1)} \cdot \mathsf{poly}(\lambda, |C|)$. Thus, starting from the trivial cutoff-$i\mathcal{O}$ (i.e., $c = 1$) and applying this composition $c - 1$ times, we obtain $N^{1/c}$-succinctness for any constant $c \in \mathbb{N}$. Note that the resulting $N^{1/c}$-succinct cutoff-$i\mathcal{O}$ essentially contains c-nested null-$i\mathcal{O}$ obfuscations, so the size of the obfuscated program also scales with $\mathsf{poly}(\lambda, |C|)^{d^c}$, for some constant d. This is the reason we can only recursively compose a constant number of times.

More formally, to carry out the recursion generically, we require the underlying cutoff-$i\mathcal{O}$ to satisfy certain additional properties. We describe these below:

- **Decomposability.** The rebalancing step needs the underlying cutoff-$i\mathcal{O}$ scheme to be *decomposable*. Namely, rather than a single monolithic obfuscated program, the underlying scheme should also decompose into a collection of block programs, each in charge of one block of indices. This is what allows us to hash the inner block programs (using the function-binding hash function), hard-wire the digest dig into the outer block programs, and force the evaluator to give an opening π that *proves* it provided the honest inner block program (for the underlying scheme).

  In the security analysis, we additionally need the block function we bind on to be able to *derive* the j^{th} inner block program from the index j alone. As we saw in the hybrid description above, this is what ensures the

description of the block function (and correspondingly, the size of the hash key) remains independent of the number of blocks. The trivial cutoff-$i\mathcal{O}$ satisfies both requirements, as does the above construction (i.e., in the above scheme, the block programs $P'_1, \ldots, P'_k$ give a decomposition).

- **Evaluation hints.** Compared to the trivial cutoff-$i\mathcal{O}$, the construction above has an additional requirement in that evaluation additionally relies on a hint from the auxiliary component H. Specifically, to evaluate the block program P'_{i_1}, the evaluator must additionally provide the pair (D'_{i_2}, π_{i_2}) as input to the block program. In contrast, with the trivial cutoff-$i\mathcal{O}$, evaluation simply invokes the appropriate block program. Thus, in order to recurse, we need to ensure that the evaluation hints for the underlying level are provided to the block programs. To carry out the reduction, we require that the evaluation hints for the underlying level be succinct and moreover, that they can be computed by a preprocessing algorithm which only depends on the index of the block program and a (short) auxiliary component H.
- **Evaluation keys and functionality hiding.** Finally, the recursion needs the underlying scheme to export a short *evaluation key* ek' that is used to evaluate its block programs. Moreover, we also require a *functionality hiding* property that says that *without* the evaluation key ek', the auxiliary component, verification key, and the block programs together reveal nothing about the circuit or the associated cutoff. This is what enables Stage 2 of our hybrid argument. The outer level hard-wires ek' into the block program of its target block only, so that erasing the one block (via null-$i\mathcal{O}$ security, as in Stage 1) suffices to remove ek' from the adversary's view.

  For this to compose, the outer level must in turn provide an evaluation key of its own; it does so by sampling a fresh symmetric key and encrypting its own block programs under it. Since a symmetric key has size $\mathsf{poly}(\lambda)$ regardless of the size of the programs it encrypts, the evaluation key does not grow with the recursion depth, and publishing it in the final scheme preserves succinctness.

We describe our full recursive construction in Section 4.

The problem with full succinctness. As noted above, our current approach only supports $N^{1/c}$-succinctness for constant c because we need to recursively compose null obfuscations c times. A super-constant c would typically incur super-polynomial overhead in the size of the obfuscated program. One approach to mitigate this blow-up is to instantiate our compiler using a rate-1 null-$i\mathcal{O}$ scheme. In this case, the overhead from each nested obfuscation is additive rather than multiplicative. However, even if we were to use rate-1 null-$i\mathcal{O}$ in our current construction, we would still only be able to support a constant number of recursion levels. This is because the size of the function-binding hash key (as well as the openings) also incurs multiplicative overhead with the recursive depth. We believe there may be an alternative approach to directly nest the obfuscated programs together (and that avoids the function-binding hash functions altogether). If instantiated using rate-1 null-$i\mathcal{O}$, this could provide a path towards a fully succinct cutoff-$i\mathcal{O}$ scheme. However, we do not currently know how to build rate-1 null-$i\mathcal{O}$ without going through plain $i\mathcal{O}$, and so we did not pursue this further in the current work. We leave the problem of building fully succinct cutoff-$i\mathcal{O}$ from rate-1 null-$i\mathcal{O}$ as an open problem.

2 Preliminaries

We write λ for the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n] := \{1, \ldots, n\}$. For a binary string $x \in \{0,1\}^*$, we denote the length of x by $|x|$. We write $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all constants $c \in \mathbb{N}$ and $\mathsf{poly}(\lambda)$ to denote a function that is $O(\lambda^c)$ for some positive constant c. For any set S, we write $x \xleftarrow{R} S$ to denote a uniform random sample x from S. All circuits in this work take inputs in $\{0,1\}^{\ell_{\mathsf{in}}}$ and produce outputs in $\{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$, where the special symbol $\bot$ is encoded so as to be distinct from every string in $\{0,1\}^{\ell_{\mathsf{out}}}$. Since a circuit must read each of its input bits and write each of its output bits, we always have $\ell_{\mathsf{in}}, \ell_{\mathsf{out}} \le |C|$. Consequently, whenever we bound the size of an object by a polynomial in a size bound $s \ge |C|$, we suppress ℓ_{in} and ℓ_{out} from that polynomial, as they are already dominated by s. Given two circuits $C_0, C_1 \colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$, we say that C_0 and C_1 are functionally equivalent if $C_0(x) = C_1(x)$ for all $x \in \{0,1\}^{\ell_{\mathsf{in}}}$. We say that a circuit C is a *null* circuit if $C(x) = \bot$ for all $x \in \{0,1\}^{\ell_{\mathsf{in}}}$. We denote a trivial null circuit by $C_\bot$. Throughout this work, we say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input.

Cryptographic primitives. We now recall the standard definitions of the cryptographic building blocks we use in this work.

Definition 2.1 (Pseudorandom Function [GGM84])**.** A pseudorandom function (PRF) consists of a pair of efficient algorithms $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to K$: On input a security parameter λ, an input length ℓ_{in}, and an output length ℓ_{out}, the key-generation algorithm outputs a key K. We assume the key contains an implicit description of ℓ_{in} and ℓ_{out}.
- $\mathsf{Eval}(K, x) \to y$: On input a key K and an input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathsf{out}}}$.

In addition, we require that Π_{PRF} satisfies pseudorandomness. Formally, for a bit $b \in \{0,1\}$ and a security parameter λ, we define the pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:

1. On input the security parameter 1^λ, the adversary outputs an input length $1^{\ell_{\mathsf{in}}}$ and an output length $1^{\ell_{\mathsf{out}}}$.
2. The challenger samples a key $K \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
3. The adversary can make an arbitrary number of *distinct* evaluation queries:
   - On each evaluation query, the adversary chooses an input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.
   - The challenger samples $y_0 \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$ and computes $y_1 = \mathsf{Eval}(K, x)$. It replies to $\mathcal{A}$ with y_b.
4. At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{PRF} satisfies pseudorandomness if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition 2.2 (Secret-Key Encryption)**.** A secret-key encryption scheme consists of a triple of efficient algorithms $\Pi_{\mathsf{SKE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{sk}$: On input a security parameter $\lambda \in \mathbb{N}$, the key-generation algorithm outputs a secret key sk.
- $\mathsf{Enc}(\mathsf{sk}, \mu) \to \mathsf{ct}$: On input a secret key sk and a message $\mu \in \{0,1\}^*$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mu$: On input a secret key sk and a ciphertext ct, the decryption algorithm outputs a plaintext $\mu \in \{0,1\}^*$. The decryption algorithm is deterministic.

We require the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all messages $\mu \in \{0,1\}^*$, it holds that:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = \mu : \begin{array}{c} \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu) \end{array}\right] = 1.$$

- **CPA-security:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the CPA-security experiment as follows:
  1. On input the security parameter 1^λ, the challenger samples a key $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$.
  2. The adversary can now make (arbitrarily many) queries on pairs of messages (μ_0, μ_1) where $|\mu_0| = |\mu_1|$. On each query, the challenger replies with $\mathsf{Enc}(\mathsf{sk}, \mu_b)$.
  3. After the adversary $\mathcal{A}$ is done making queries, it outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{SKE} satisfies CPA-security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition 2.3 (Null Indistinguishability Obfuscation [GKW17, WZ17, adapted])**.** A null indistinguishability obfuscator NiO is an efficient algorithm that takes as input the security parameter 1^λ, a size bound 1^s, and a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s, and outputs a circuit C'. There exists a polynomial poly such that for all such λ, s, C we have $\left|\mathsf{NiO}(1^\lambda, 1^s, C)\right| = \mathsf{poly}(\lambda, s)$. The obfuscator NiO should satisfy the following properties:

- **Functionality-preserving:** For all security parameters $\lambda, s \in \mathbb{N}$, all circuits $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s, and all circuits C' in the support of $\mathsf{NiO}(1^\lambda, 1^s, C)$, the circuits C and C' are functionally equivalent.
- **Security for null circuits:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the null-$i\mathcal{O}$ security game as follows:
  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends 1^s and circuits $C_0, C_1\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s to the challenger.
  2. If C_0 or C_1 is not a null circuit, the experiment aborts and outputs 0.
  3. The challenger samples $C' \leftarrow \mathsf{NiO}(1^\lambda, 1^s, C_b)$ and sends C' to the adversary.
  4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that NiO satisfies security for null circuits if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Remark 2.4 (Equivalence to the Standard Formulation)**.** Definition 2.3 is implied by the standard definition of null-$i\mathcal{O}$ from [GKW17, WZ17], where the circuits being obfuscated have binary outputs, and security holds when obfuscating two circuits (of equal size) that output 0 on all inputs. Indeed, we can describe any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ as a collection of $\ell_{\mathsf{out}} + 1$ Boolean circuits $C_0, C_1, \ldots, C_{\ell_{\mathsf{out}}}$, where $C_i(x)$ outputs the i^{th} bit of $C(x)$ (and outputs 0 if $C(x) = \bot$) and $C_0\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ outputs 0 if $C(x) = \bot$ and 1 otherwise. To obfuscate C, we apply null-$i\mathcal{O}$ to $C_0, C_1, \ldots, C_{\ell_{\mathsf{out}}}$. This is functionality-preserving by construction. Moreover, if C, C' are null circuits, then the associated circuits $(C_0, C_1, \ldots, C_{\ell_{\mathsf{out}}})$ and $(C'_0, C'_1, \ldots, C'_{\ell_{\mathsf{out}}})$ are null circuits. Null-$i\mathcal{O}$ security now follows by a standard hybrid argument over the $\ell_{\mathsf{out}} + 1$ components.

Definition 2.5 (Function-Binding Hash for Conjunction of Block-Functions [FWW23])**.** A *function-binding hash* for conjunctions of block functions is a tuple of efficient algorithms $\Pi_{\mathsf{FBH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^s, 1^B, k) \to \mathsf{hk}$: On input a security parameter λ, a function size bound s, a block size B, and a number of blocks k (in binary), the setup algorithm outputs a hash key hk. We implicitly assume that hk includes $1^\lambda, 1^s, 1^B$, and k.
- $\mathsf{SetupBinding}(1^\lambda, 1^s, 1^B, k, f) \to \mathsf{hk}$: On input a security parameter λ, a function size bound s, a block size B, a number of blocks k (in binary), and a function $f\colon [k] \times \{0,1\}^B \to \{0,1\}$ of size at most s, the binding-mode setup outputs a hash key hk.
- $\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \to (\mathsf{dig}, \pi_1, \ldots, \pi_k)$: On input a hash key hk and a string $\mathbf{x} \in (\{0,1\}^B)^k$, the hash algorithm outputs a digest dig and k openings $\pi_1, \ldots, \pi_k$. This algorithm is deterministic.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x, \pi) \to b$: On input a hash key hk, a digest dig, an index $i \in [k]$, a block $x \in \{0,1\}^B$, and an opening π, the verification algorithm outputs a bit $b \in \{0,1\}$. The verification algorithm is deterministic.

We require that Π_{FBH} satisfies the following properties:

- **Succinctness:** For all parameters $\lambda, s, B, k \in \mathbb{N}$, all hk in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^B, k)$, all inputs $\mathbf{x} \in (\{0,1\}^B)^k$:
  - **Succinct digest:** The digest dig output by $\mathsf{Hash}(\mathsf{hk}, \mathbf{x})$ has size at most $\mathsf{poly}(\lambda, s, B, \log k)$.
  - **Succinct openings:** Each opening π_i output by $\mathsf{Hash}(\mathsf{hk}, \mathbf{x})$ has size at most $\mathsf{poly}(\lambda, s, B, \log k)$.
- **Correctness:** For all $\lambda, s, B, k \in \mathbb{N}$, every $\mathbf{x} = (x_1, \ldots, x_k) \in (\{0,1\}^B)^k$, and every $i \in [k]$:

$$\Pr\left[\ \mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x_i, \pi_i) = 1 \ \ : \ \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^B, k) \\ (\mathsf{dig}, \pi_1, \ldots, \pi_k) = \mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \end{array}\ \right] = 1.$$

- **Mode indistinguishability:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the mode indistinguishability game as follows:
  1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^B, k \in \mathbb{N}$ and $f\colon [k] \times \{0,1\}^B \to \{0,1\}$ of size at most s to the challenger.
  2. The challenger samples the hash keys $\mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^B, k)$ and $\mathsf{hk}_1 \leftarrow \mathsf{SetupBinding}(1^\lambda, 1^s, 1^B, k, f)$. The challenger gives hk_b to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{FBH} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Perfect function binding:** We say that Π_{FBH} satisfies perfect function binding if for all $\lambda, s, B, k \in \mathbb{N}$, all functions $f\colon [k] \times \{0,1\}^B \to \{0,1\}$ of size at most s, all hk in the support of $\mathsf{SetupBinding}(1^\lambda, 1^s, 1^B, k, f)$, and all $\mathbf{x} = (x_1, \ldots, x_k) \in (\{0,1\}^B)^k$ such that $f(i, x_i) = 1$ for all $i \in [k]$, there does not exist a valid opening π of dig for any index $i \in [k]$ to a block $x \in \{0,1\}^B$ that does not satisfy f, where $(\mathsf{dig}, \pi_1, \ldots, \pi_k) = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$. Namely, for all π, i, x if $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x, \pi) = 1$ then $f(i, x) = 1$.

3 Cutoff Indistinguishability Obfuscation

In this section, we introduce the main notion of a *cutoff indistinguishability obfuscation* (cutoff-$i\mathcal{O}$). A cutoff-$i\mathcal{O}$ scheme takes as input a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ and a cutoff threshold $t \in [0, N]$, and outputs a program that outputs $\bot$ for all indices up to t, but faithfully evaluates C for all indices beyond the threshold. Thus the obfuscated program is fully functional at $t = 0$ and is "turned off" at $t = N$. The security property states that if the circuit naturally outputs $\bot$ at index t, then programs computed with cutoff t are computationally indistinguishable from programs computed with cutoff $t - 1$.

Definition 3.1 (Cutoff Indistinguishability Obfuscation)**.** A cutoff indistinguishability obfuscation scheme Π_{CiO} consists of three efficient algorithms $\Pi_{\mathsf{CiO}} = (\mathsf{ObfNull}, \mathsf{Obf}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{ObfNull}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to P$: On input the security parameter λ, a size bound s, the size of the index space N, the input length ℓ_{in}, and the output length ℓ_{out}, the null-obfuscation algorithm outputs a program P.
- $\mathsf{Obf}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t) \to P$: On input the security parameter λ, a size bound s, the size of the index space N, the input length ℓ_{in}, the output length ℓ_{out}, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, and a cutoff index $t \in \{0, \ldots, N\}$, the obfuscation algorithm outputs a program P.
- $\mathsf{Eval}(P, w, i) \to y$: On input a program P, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$ and an index $i \in [N]$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$.

The cutoff-$i\mathcal{O}$ scheme Π_{CiO} should satisfy the following properties:

- **Correctness:** For all $\lambda, s, N, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, all circuits $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s, all cutoffs $t \in \{0,\ldots,N\}$, all $(w,i) \in \{0,1\}^{\ell_{\mathsf{in}}} \times [N]$, and all output programs P in the support of the obfuscation algorithm $\mathsf{Obf}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$, the following holds:

$$\mathsf{Eval}(P,w,i) = \begin{cases} C(w,i) & \text{if } i > t \\ \bot & \text{otherwise} \end{cases}.$$

- **Succinctness:** We say that Π_{CiO} is $\eta(N)$-succinct if for all $\lambda, s, N, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, all circuits $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s, all $t \in \{0,\ldots,N\}$, and any P in the support of the obfuscation algorithm $\mathsf{Obf}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$, the following holds:

$$|P| \le \eta(N) \cdot \mathsf{poly}(\lambda, s, \log N).$$

- **Index indistinguishability:** For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0,1\}$:

  1. On input 1^λ, $\mathcal{A}$ sends $1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C$, and an index $t \in \{1,\ldots,N\}$ to the challenger, where $|C| \le s$.
  2. If there exists $w \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $C(w,t) \neq \bot$, the experiment aborts and outputs $b' = 0$.
  3. The challenger samples $P \leftarrow \mathsf{Obf}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t-b)$ and sends P to $\mathcal{A}$.
  4. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{CiO} satisfies index indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Circuit hiding:** For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0,1\}$:

  1. On input 1^λ, $\mathcal{A}$ sends $1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, and a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$ to the challenger.
  2. If $b = 0$, the challenger samples $P \leftarrow \mathsf{Obf}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, N)$. If $b = 1$, the challenger samples $P \leftarrow \mathsf{ObfNull}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$. It sends P to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{CiO} satisfies *computational circuit hiding* if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

  We say that Π_{CiO} satisfies *perfect circuit hiding* if for every (potentially unbounded) adversary $\mathcal{A}$, we have

$$\Pr[b' = 1 : b = 0] = \Pr[b' = 1 : b = 1].$$

4 Leveled Cutoff Indistinguishability Obfuscation

In this section we develop a recursive approach to building cutoff-$i\mathcal{O}$. To that end, we first introduce a variant of Definition 3.1, which we call *c-leveled cutoff-$i\mathcal{O}$*, that is better suited for recursive composition. Specifically, suppose we want to handle an index space $N = k^c$. Our recursive composition relies on splitting the range into k blocks each of size k^{c-1}. For each of the k blocks, we have a separate block program P'_j that is in charge of evaluating on indices that fall into the block. Moreover, each block program should be efficiently computable on its own in time independent of k (and

without needing to recompute the other block programs). To capture this property, we split the obfuscation algorithm into a Setup algorithm that produces an initial *state*, and an ObfBlockProgram algorithm that takes as input a state and a block, and outputs the associated block program. Additionally, each block program will require an additional, succinct evaluation hint which is passed to it as input. Therefore, Setup will also output a *large* master auxiliary component of size up to k, from which the succinct evaluation hints can be derived (via a Preprocess algorithm). We also require an additional verification algorithm that checks whether an evaluation hint was derived honestly. We only require a mild soundness property on the verification algorithm that triggers for the fully functional program (when $t = 0$). If the evaluation hint verifies, then the evaluation output should coincide with that of the obfuscated circuit.

Additionally, Setup outputs a short *evaluation key* ek that is required in order to evaluate a block program, and which is published alongside the block programs (cf. Proposition 4.37). When the evaluation key is *withheld*, we require that the block programs reveal nothing about the obfuscated circuit or the cutoff. We call this *functionality hiding*.

Definition 4.1 (c-Leveled Cutoff Indistinguishability Obfuscation)**.** Let $c \geq 1$ be an integer. A *c-leveled cutoff indistinguishability obfuscation scheme* Π_{LCiO} consists of a tuple of efficient algorithms $\Pi_{\mathsf{LCiO}} = (\mathsf{SetupNull}, \mathsf{Setup}, \mathsf{ObfBlockProgram}, \mathsf{Preprocess}, \mathsf{Verify}, \mathsf{Eval})$, with the level parameter c hard-wired into the scheme, not an input to any of its algorithms. Throughout, we refer to k as the branching factor and denote by $N = k^c$ the size of the index space. The algorithms have the following syntax:

- $\mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to (\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$: On input the security parameter λ, a size bound s, a branching factor k, the input length ℓ_{in}, and the output length ℓ_{out}, the null-setup algorithm outputs a state st, an auxiliary component H, a verification key vk, and an evaluation key ek.
- $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t) \to (\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$: On input the security parameter λ, a size bound s, a branching factor k, the input length ℓ_{in}, the output length ℓ_{out}, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \leq s$, and a cutoff index $t \in \{0, \ldots, N\}$, the setup algorithm outputs a state st, an auxiliary component H, a verification key vk, and an evaluation key ek.
- $\mathsf{ObfBlockProgram}(\mathsf{st}, j) \to P_j$: On input a state st and a block index $j \in [k]$, the block-program obfuscation algorithm outputs a block program P_j.
- $\mathsf{Preprocess}(H, w, i) \to h$: On input an auxiliary component H, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [N]$, the preprocessing algorithm outputs an evaluation hint h. This algorithm is deterministic.
- $\mathsf{Verify}(\mathsf{vk}, h, i) \to b$: On input a verification key vk, an evaluation hint h, and an index $i \in [N]$, the verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.
- $\mathsf{Eval}(\mathsf{ek}, P, h, w, i) \to y$: On input an evaluation key ek, a block program P, an evaluation hint h, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [N]$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$. This algorithm is deterministic.

The c-leveled cutoff-*iO* scheme Π_{LCiO} should satisfy the following properties:

- **Correctness:** For all $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, all circuits $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ with $|C| \leq s$, all cutoffs $t \in \{0, \ldots, N\}$, all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, all $i \in [N]$, all $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$, and all P in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, \lceil i/k^{c-1} \rceil)$, we have

$$\mathsf{Eval}(\mathsf{ek}, P, \mathsf{Preprocess}(H, w, i), w, i) = \begin{cases} C(w, i) & \text{if } i > t, \\ \bot & \text{otherwise.} \end{cases}$$

- **Hint validity:** For all $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, all circuits $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ with $|C| \leq s$, all cutoffs $t \in \{0, \ldots, N\}$, all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, all $i \in [N]$, and all $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ in the support of either

$$\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t) \quad \text{or} \quad \mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}),$$

  we have $\mathsf{Verify}\big(\mathsf{vk}, \mathsf{Preprocess}(H, w, i), i\big) = 1$.

- **Endpoint soundness:** For all $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, all $i \in [N]$, and all $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ with $|C| \le s$, the following holds:
  - **Soundness at the all-off cutoff:** For all $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, N)$, all $j \in [k]$, and all P in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$, we have
  $$\forall h : \ \mathsf{Eval}(\mathsf{ek}, P, h, w, i) = \bot.$$
  - **Soundness at the all-on cutoff:** For all $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, 0)$ and all P in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, \lceil i/k^{c-1} \rceil)$, we have
  $$\forall h : \ \mathsf{Verify}(\mathsf{vk}, h, i) = 1 \implies \mathsf{Eval}(\mathsf{ek}, P, h, w, i) = C(w, i).$$
- **Succinctness:** There exist a universal polynomial $\mathsf{poly}_c(\cdot)$ (which may depend on the parameter c) and a universal polynomial $\mathsf{poly}_{\mathsf{ek}}(\cdot)$ (which may *not* depend on c) such that for all $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$ with $\lambda, k \ge 1$, all circuits $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s, all cutoffs $t \in \{0, \ldots, N\}$, all $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$ or of $\mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and all $i \in [N]$, the following bounds hold:
  - **Succinct evaluation key:** $|\mathsf{ek}| \le \mathsf{poly}_{\mathsf{ek}}(\lambda)$.
  - **Succinct verification key:** $|\mathsf{vk}| \le \mathsf{poly}_c(\lambda, s, \log k)$.
  - **Succinct evaluation hint:** $|h| \le \mathsf{poly}_c(\lambda, s, \log k)$, where $h = \mathsf{Preprocess}(H, w, i)$.
  - **Succinct state:** $|\mathsf{st}| \le \mathsf{poly}_c(\lambda, s, \log k)$.
  - **Succinct block programs:** For all $j \in [k]$, the running time of $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$ is bounded by $\mathsf{poly}_c(\lambda, s, \log k)$ and all P in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$ satisfy $|P| \le \mathsf{poly}_c(\lambda, s, \log k)$.
  - **Compact auxiliary component:** $|H| \le k \cdot \mathsf{poly}_c(\lambda, s, \log k)$.

  The auxiliary component is the only component permitted to grow with the branching factor k. In particular, each individual block program, and the time to generate it, is independent of k up to $\mathsf{poly}(\log k)$ factors.
- **Index indistinguishability:** We define the index indistinguishability game parameterized by a bit $b \in \{0,1\}$:
  1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, and a threshold $t \in \{1, \ldots, N\}$ to the challenger.
  2. If there exists $w \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $C(w, t) \ne \bot$, the experiment aborts and outputs $b' = 0$.
  3. The challenger samples $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t - b)$.
  4. For each $j \in [k]$, the challenger samples $P_j \leftarrow \mathsf{ObfBlockProgram}(\mathsf{st}, j)$.
  5. The challenger sends $(\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j \in [k]})$ to $\mathcal{A}$.
  6. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{LCiO} satisfies index indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
  $$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$
- **Functionality hiding:** We define the functionality hiding game parameterized by a bit $b \in \{0,1\}$:
  1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, a pair of circuits $C^{(0)}, C^{(1)}\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ bounded by size s, and a pair of cutoffs $t^{(0)}, t^{(1)} \in \{0, \ldots, N\}$ to the challenger.
  2. The challenger samples $(\mathsf{st}^{(b)}, H^{(b)}, \mathsf{vk}^{(b)}, \mathsf{ek}^{(b)}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C^{(b)}, t^{(b)})$.
  3. For each $j \in [k]$, the challenger samples $P_j^{(b)} \leftarrow \mathsf{ObfBlockProgram}(\mathsf{st}^{(b)}, j)$.

4. The challenger sends $(H^{(b)}, \mathsf{vk}^{(b)}, \{P_j^{(b)}\}_{j\in[k]})$ to the adversary. Note that the evaluation key $\mathsf{ek}^{(b)}$ is *not* given to the adversary.
5. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{LCiO} satisfies functionality hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Circuit hiding:** We define the circuit hiding game parameterized by a bit $b \in \{0,1\}$:
  1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, and a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$ to the challenger.
  2. If $b = 0$, the challenger samples $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, N)$. If $b = 1$, the challenger samples $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  3. For each $j \in [k]$, the challenger samples $P_j \leftarrow \mathsf{ObfBlockProgram}(\mathsf{st}, j)$.
  4. The challenger sends $(\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j\in[k]})$ to $\mathcal{A}$.
  5. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{LCiO} satisfies *computational circuit hiding* if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

  We say that Π_{LCiO} satisfies *perfect circuit hiding* if for every (potentially unbounded) adversary $\mathcal{A}$, we have

$$\Pr[b' = 1 : b = 0] = \Pr[b' = 1 : b = 1].$$

4.1 Constructing a 1-Leveled Cutoff-$i\mathcal{O}$

We start by building a trivial cutoff-$i\mathcal{O}$ scheme (i.e., with succinctness N). The construction relies on a null-$i\mathcal{O}$ together with a symmetric encryption scheme. Since $c = 1$, there is no inner level: the auxiliary component and the verification key are both empty, and Verify accepts unconditionally. The evaluation key is a symmetric encryption key under which the block programs are encrypted.

Construction 4.2 (1-Leveled Cutoff-$i\mathcal{O}$)**.** Our construction relies on the following primitives:

- Let NiO be a null indistinguishability obfuscation scheme.
- Let $\Pi_{\mathsf{SKE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme.

Let s_P be an upper bound on the size of the program described in Fig. 4 for any circuit C of size $|C| \le s$, and let $\hat{s}_P$ be the size of an NiO obfuscation of a circuit of size s_P. We construct a 1-leveled cutoff-$i\mathcal{O}$ scheme $\Pi_{\mathsf{LCiO}} = (\mathsf{SetupNull}, \mathsf{Setup}, \mathsf{ObfBlockProgram}, \mathsf{Preprocess}, \mathsf{Verify}, \mathsf{Eval})$ as follows:

- $\mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$: On input the security parameter λ, a size bound s, a branching factor k, the input length ℓ_{in}, and the output length ℓ_{out}, the null-setup algorithm samples an evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and outputs the state $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C_\bot, k, \mathsf{ek})$, the empty auxiliary component $H = \varnothing$, the verification key $\mathsf{vk} = \bot$, and the evaluation key ek.
- $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$: On input the security parameter λ, a size bound s, a branching factor k, the input length ℓ_{in}, the output length ℓ_{out}, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [k] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, and a cutoff index $t \in \{0,\dots,k\}$, the setup algorithm samples an evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and outputs the state $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C, t, \mathsf{ek})$, the empty auxiliary component $H = \varnothing$, the verification key $\mathsf{vk} = \bot$, and the evaluation key ek.

- $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$: On input a state $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C, t, \mathsf{ek})$ and a block index $j \in [k]$, the block-program obfuscation algorithm does the following:

  1. If $j \le t$, let P_{null} be a trivial null circuit of size s_P that outputs $\bot$ on every input, and sample $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathrm{null}})$.
  2. If $j > t$, construct the circuit $P_j = P[C, j]$ described in Fig. 4, and sample $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_j)$.

     Constants: A circuit C and a block index $j \in [k]$.
     Inputs: A witness $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, an index $i \in [k]$, and an evaluation hint h.

     1. If $i = j$, output $C(w, i)$. Otherwise, output $\bot$.

     Figure 4: Program $P[C, j]$.

  3. In either case, output the encrypted block program $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$.

- $\mathsf{Preprocess}(H, w, i)$: On input an auxiliary component H, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [k]$, the preprocessing algorithm outputs the empty evaluation hint $h = \bot$.

- $\mathsf{Verify}(\mathsf{vk}, h, i)$: On input a verification key vk, an evaluation hint h, and an index $i \in [k]$, the verification algorithm outputs $b = 1$.

- $\mathsf{Eval}(\mathsf{ek}, P, h, w, i)$: On input an evaluation key ek, a block program P, an evaluation hint h, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [k]$, the evaluation algorithm decrypts $P' = \mathsf{SKE.Dec}(\mathsf{ek}, P)$ and outputs $y = P'(w, i, h)$.

Theorem 4.3 (Correctness)**.** *Suppose* NiO *is correct and* Π_{SKE} *is correct. Then Construction 4.2 satisfies correctness.*

Proof. Take any $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [k] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, any cutoff $t \in \{0, \ldots, k\}$, any input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and any index $i \in [k]$. Take any tuple $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$ and let ct_i be in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, i)$. Since $c = 1$, we have $N = k$ and $\lceil i/k^{c-1} \rceil = i$. Write $\mathsf{ct}_i = \mathsf{SKE.Enc}(\mathsf{ek}, P'_i)$ where P'_i is the obfuscated program constructed by ObfBlockProgram; since ek is the key recorded in st, correctness of Π_{SKE} gives $\mathsf{SKE.Dec}(\mathsf{ek}, \mathsf{ct}_i) = P'_i$. By construction $\mathsf{Preprocess}(H, w, i) = \bot$, so

$$\mathsf{Eval}(\mathsf{ek}, \mathsf{ct}_i, \mathsf{Preprocess}(H, w, i), w, i) = P'_i(w, i, \bot).$$

We consider the two possibilities:

- Suppose $i \le t$. Then P'_i is an obfuscation of a trivial null circuit, so the output is $\bot$, as required.
- Suppose $i > t$. Then P'_i is an obfuscation of $P[C, i]$, which on index i outputs $C(w, i)$, again as required.

In both cases, we used the fact that NiO is functionality-preserving. □

Theorem 4.4 (Hint Validity)**.** *Construction 4.2 satisfies hint validity.*

Proof. This is immediate as Verify outputs 1 on every input. □

Theorem 4.5 (Endpoint Soundness)**.** *Suppose* NiO *is correct and* Π_{SKE} *is correct. Then Construction 4.2 satisfies endpoint soundness.*

Proof. Take any $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [k] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, any input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and any index $i \in [k]$; since $c = 1$, we have $N = k$ and $\lceil i/k^{c-1} \rceil = i$. In both cases below, ek is the key recorded in st, so by correctness of Π_{SKE} the evaluation algorithm recovers the obfuscated program that ObfBlockProgram encrypted. We consider the two endpoints separately:

- **Soundness at the all-off cutoff.** Let $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ be in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, k)$, let $j \in [k]$, and let ct_j be in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$, encrypting the obfuscated program P'_j. Since the cutoff is $t = k$, every $j \in [k]$ satisfies $j \le t$, so P'_j is an obfuscation of a trivial null circuit. Hence $\mathsf{Eval}(\mathsf{ek}, \mathsf{ct}_j, h, w, i) = P'_j(w, i, h) = \bot$ for every evaluation hint h, as required.

- **Soundness at the all-on cutoff.** Let $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ be in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, 0)$ and let ct_i be in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, i)$, encrypting the obfuscated program P'_i. Since the cutoff is $t = 0$ and $i \ge 1$, we have $i > t$, so P'_i is an obfuscation of $P[C, i]$, which on index i outputs $C(w, i)$. Now, for all h, we have $\mathsf{Eval}(\mathsf{ek}, \mathsf{ct}_i, h, w, i) = P'_i(w, i, h) = C(w, i)$, as required.

In both cases, we used the fact that NiO is functionality-preserving. □

Theorem 4.6 (Succinctness)**.** *Suppose* NiO *expands a circuit of size S to a circuit of size at most $p_{\mathsf{NiO}}(\lambda, S)$ for some fixed polynomial p_{NiO}. Then Construction 4.2 is succinct.*

Proof. The evaluation key is sampled as $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$, so $|\mathsf{ek}| \le \mathsf{poly}_{\mathsf{ek}}(\lambda)$. The verification key is $\bot$ and both H and every h are empty, so all three have size $O(1)$. The state st has the description of a circuit of size at most C, descriptions of the cutoff index and k which have size at most $\log k$, and the verification and evaluation keys which we already bounded. Therefore, the state has size at most $\mathsf{poly}(\lambda, s, \log k)$. The program $P[C, j]$ evaluates C and one equality check on $\log k$-bit indices. Thus, $|P[C, j]| \le s + O(\log k)$, and hence $\hat{s}_P \le p_{\mathsf{NiO}}(\lambda, s + O(\log k)) = \mathsf{poly}(\lambda, s, \log k)$. A block program is an encryption of such an obfuscated program, so its size is $\hat{s}_P + \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda, s, \log k)$ as well. The algorithm ObfBlockProgram only constructs this program P, obfuscates it, and encrypts it, so it runs in time $\mathsf{poly}(\lambda, s, \log k)$. All bounds of Definition 4.1 therefore hold with this polynomial. □

Theorem 4.7 (Index Indistinguishability)**.** *Assume* NiO *is a secure null-iO. Then Construction 4.2 satisfies index indistinguishability.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the index indistinguishability game that succeeds with non-negligible advantage $\varepsilon(\lambda)$. We construct an algorithm $\mathcal{B}$ for the NiO security game as follows:

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and receives $(1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$.
2. Algorithm $\mathcal{B}$ submits $(P_{\mathsf{null}}, P[C, t])$ and 1^{s_P} to the NiO challenger, where P_{null} is a trivial null program and $P[C, t]$ is described in Fig. 4. The program $P[C, t](w, i, h)$ outputs $C(w, i)$ if $i = t$ and $\bot$ otherwise, so $P[C, t]$ is a null program if and only if $C(w, t) = \bot$ for all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$. Since P_{null} is null, the NiO experiment aborts precisely when the index indistinguishability game does, and in that case, both experiments output 0. In particular, algorithm $\mathcal{B}$ does not have to decide whether such a w exists.
3. The null-iO challenger returns an obfuscated circuit P^*.
4. Algorithm $\mathcal{B}$ sets $P'_t = P^*$ and generates $\{P'_j\}_{j \ne t}$ as follows:
   - For any index $j < t$, algorithm $\mathcal{B}$ samples $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathsf{null}})$.
   - For any index $j > t$, algorithm $\mathcal{B}$ samples $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P[C, j])$, where $P[C, j]$ is described in Fig. 4.
5. Algorithm $\mathcal{B}$ samples an evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and encrypts each block program as $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$ for all $j \in [k]$.
6. Algorithm $\mathcal{B}$ gives the evaluation key ek, the empty auxiliary component $H = \varnothing$, the verification key $\mathsf{vk} = \bot$, and $\{\mathsf{ct}_j\}_{j \in [k]}$ to $\mathcal{A}$, and outputs $\mathcal{A}$'s guess $b' \in \{0, 1\}$.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger obfuscated P_{null}, $\mathcal{B}$ perfectly simulates the distribution for threshold t. If the challenger obfuscated $P[C, t]$, $\mathcal{B}$ perfectly simulates the distribution for threshold $t - 1$. Thus, algorithm $\mathcal{B}$ breaks security of NiO with the same non-negligible advantage $\varepsilon(\lambda)$. The claim holds. □

Theorem 4.8 (Functionality Hiding). *Assume that Π_{SKE} is CPA-secure. Then Construction 4.2 satisfies functionality hiding.*

Proof. The auxiliary component $H = \varnothing$ and the verification key $\mathsf{vk} = \bot$ are independent of the circuit and the cutoff, so the claim concerns only the block programs, each of which is an encryption under the withheld evaluation key ek. Fix an efficient adversary $\mathcal{A}$ for the functionality hiding game and let ε denote its advantage. We construct an algorithm $\mathcal{B}$ that breaks CPA-security with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, a pair of circuits $C^{(0)}, C^{(1)} \colon \{0,1\}^{\ell_{\mathsf{in}}} \times [k] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ bounded by size s, and a pair of cutoffs $t^{(0)}, t^{(1)} \in \{0, \ldots, k\}$ to $\mathcal{B}$.

2. For each $\beta \in \{0,1\}$ and each $j \in [k]$, algorithm $\mathcal{B}$ constructs the obfuscated program $P'^{(\beta)}_j$ exactly as ObfBlockProgram does on the state associated with $(C^{(\beta)}, t^{(\beta)})$: it obfuscates a trivial null circuit of size s_P if $j \le t^{(\beta)}$, and $P[C^{(\beta)}, j]$ from Fig. 4 otherwise.

3. For each $j \in [k]$, algorithm $\mathcal{B}$ makes the challenge query $\left(P'^{(0)}_j, P'^{(1)}_j\right)$, to which the CPA challenger responds with a ciphertext ct_j. Both messages have length $\hat{s}_P$, so the query is admissible.

4. Algorithm $\mathcal{B}$ sends the empty auxiliary component $H = \varnothing$, the verification key $\mathsf{vk} = \bot$, and $\{\mathsf{ct}_j\}_{j \in [k]}$ to $\mathcal{A}$, and outputs $\mathcal{A}$'s guess $b' \in \{0,1\}$.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$, and $\mathcal{B}$ never uses the encryption key. If the CPA challenger encrypts $P'^{(0)}_j$ for all j (bit $b = 0$), then $\mathcal{B}$ perfectly simulates the functionality hiding game with bit $b = 0$, and likewise for $b = 1$. Therefore $\mathcal{B}$ breaks CPA-security with the same advantage ε. □

Theorem 4.9 (Perfect Circuit Hiding). *Construction 4.2 satisfies perfect circuit hiding.*

Proof. Fix any $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$ and any circuit $C \colon \{0,1\}^{\ell_{\mathsf{in}}} \times [k] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s. We show that the following two distributions are identical:

$$\left\{ (\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j\in[k]}) : \begin{array}{c} (\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, k) \\ \forall j \in [k], P_j \leftarrow \mathsf{ObfBlockProgram}(\mathsf{st}, j) \end{array} \right\},$$

$$\left\{ (\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j\in[k]}) : \begin{array}{c} (\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \\ \forall j \in [k], P_j \leftarrow \mathsf{ObfBlockProgram}(\mathsf{st}, j) \end{array} \right\}.$$

In both cases $H = \varnothing$ and $\mathsf{vk} = \bot$, so the auxiliary component and the verification key in the two distributions are identically distributed, and in both cases the evaluation key is sampled as $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ independently of everything else, so it too is identically distributed. The state produced by Setup at threshold $t = k$ is $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C, k, \mathsf{ek})$, and the state produced by SetupNull is $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C_\bot, k, \mathsf{ek})$. In both, the associated cutoff is k. Hence for every $j \in [k]$ the condition $j \le t = k$ holds, so ObfBlockProgram always obfuscates the trivial null circuit as $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathsf{null}})$ in both cases, and then encrypts it under ek. Every block program is therefore an encryption, under the same key, of an independent NiO obfuscation of the same trivial null circuit at the same size bound, so the two distributions are identical. □

4.2 Composing Leveled Cutoff-$i\mathcal{O}$

Fix any positive integer $c \ge 2$. In this section we show how to turn a $(c-1)$-leveled cutoff-$i\mathcal{O}$ into a c-leveled cutoff-$i\mathcal{O}$, using a null indistinguishability obfuscator, a symmetric-key encryption scheme, and a function-binding hash function. The $(c-1)$-leveled scheme is used as a black box. We refer to Section 1.2 for a high-level description of this compiler for the special case of $c = 2$ (i.e., where we lift a trivial cutoff-$i\mathcal{O}$ scheme to one with $\sqrt{N}$-succinctness).

Approach overview. The high-level idea is to use the $(c-1)$-leveled scheme as the auxiliary component for the c-leveled scheme. When the cutoff index t is in a target block $[(t_1-1)\cdot k^{c-1}, t_1\cdot k^{c-1}-1]$, the auxiliary component implements a $(c-1)$-leveled cutoff-$i\mathcal{O}$ for that block. In addition, the block programs at level $(c-1)$ are hashed (using the function-binding hash function), and the associated inner evaluation key ek' lives in the block program corresponding to the target block. This allows the target block to implement the underlying cutoff-$i\mathcal{O}$ scheme without affecting the behavior of the other block programs.

Note that the inner block programs are *not* encrypted by this level: by Definition 4.1 they are already encrypted under ek'. What this level does encrypt is its *own* block programs $P_1, \ldots, P_k$, under a fresh symmetric key that it publishes as its evaluation key ek.

Notation. We now introduce some notation that we will use in our recursive construction.

- Let $N = k^c$ be the size of the index space and let $N' = k^{c-1}$ be the size of the inner index space. We will decompose an index $i \in [N]$ into a block index i_1 together with an inner index i_2. Specifically, define $\mathsf{decomp}_{k,c}\colon [N+1] \to [k+1] \times \left[k^{c-1}\right]$ by

$$\mathsf{decomp}_{k,c}(i) = (i_1, i_2) \text{ where } i_1 = \lceil i/k^{c-1} \rceil \text{ and } i_2 = i - (i_1-1)\cdot k^{c-1} \in [k^{c-1}].$$

  In particular, $i = (i_1-1)\cdot k^{c-1} + i_2$. For all $i \in [N]$, the decomposition (i_1, i_2) satisfies $i_1 \in [k]$. Lastly, $\mathsf{decomp}_{k,c}(N+1) = (k+1, 1)$.

- A cutoff t counts how many *leading* indices are switched off, so we decompose it by looking at the first index it leaves active (i.e., the index $t+1$), and decomposing that index. Specifically, we define the cutoff-decomposition function $\mathsf{decomp}^*_{k,c}\colon \{0,\ldots,N\} \to [k+1] \times \{0,\ldots,k^{c-1}-1\}$ where

$$\mathsf{decomp}^*_{k,c}(t) := \mathsf{decomp}_{k,c}(t+1) - (0,1).$$

  Writing $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, this amounts to $t_1 = \lfloor t/k^{c-1} \rfloor + 1$ and $t_2 = t - (t_1-1)\cdot k^{c-1}$ (i.e., $t = (t_1-1)\cdot k^{c-1} + t_2$). Thus t switches off the first $t_1 - 1$ blocks entirely, together with the first t_2 indices of block t_1, and t_1 is the block containing the first active index $t+1$. Note that $t = N$ is the unique cutoff with $t_1 = k+1$, in which case every block is switched off. For a cutoff t with $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, block t_1 is *fully active* when $t_2 = 0$ (i.e., t is an exact multiple of k^{c-1}) and is *partially active* when $t_2 > 0$. With this convention, for any index $i \in [N]$ the condition $i > t$ is equivalent to $(i_1 > t_1) \vee (i_1 = t_1 \wedge i_2 > t_2)$.

- Throughout, we use the following two mappings into the inner index space. For an index i with $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$, the *reversed inner index* is $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$, and the corresponding inner block index is $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$. For a cutoff with $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, the *reversed inner cutoff* is $\bar{t}_2 = k^{c-1} - t_2$.

Construction 4.10 (c-Leveled Cutoff-$i\mathcal{O}$ from $(c-1)$-Leveled Cutoff-$i\mathcal{O}$)**.** Our construction relies on the following primitives:

- Let $\Pi'_{\mathsf{LCiO}} = (\mathsf{SetupNull}', \mathsf{Setup}', \mathsf{ObfBlockProgram}', \mathsf{Preprocess}', \mathsf{Verify}', \mathsf{Eval}')$ be a $(c-1)$-leveled cutoff-$i\mathcal{O}$ scheme for index space $N' = k^{c-1}$. We write ek' for its evaluation key.
- Let NiO be a null indistinguishability obfuscation scheme.
- Let $\Pi_{\mathsf{SKE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme.
- Let $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{Eval})$ be a pseudorandom function. This will only appear in the security proof.
- Let $\Pi_{\mathsf{FBH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ be a function-binding hash for conjunctions of block functions.

Let s_D be an upper bound on the size of an inner block program produced by ObfBlockProgram′. By the succinctness of Π'_{LCiO} we may take $s_D = \mathrm{poly}'(\lambda, s, \log k)$, where poly′ is the polynomial guaranteed by Definition 4.1 for Π'_{LCiO}. Let s_f be the size of the block function described in Fig. 7. Let s_P and s_G be the sizes of the programs described in Figs. 5 and 6, parameterized by the circuit size bound s, and let $\hat{s}_P$ be the size of an NiO obfuscation of a circuit of size s_P. For any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ of size at most s, and any constant α, let $s' = s + r \cdot \mathrm{poly}(N)$ be a bound the circuit $C'(w,x) = C(w, \alpha - x)$. We construct a c-leveled cutoff-$i\mathcal{O}$ scheme $\Pi_{\mathsf{LCiO}} = (\mathsf{SetupNull}, \mathsf{Setup}, \mathsf{ObfBlockProgram}, \mathsf{Preprocess}, \mathsf{Verify}, \mathsf{Eval})$ as follows:

- $\mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$: On input the security parameter λ, a size bound s, a branching factor k, the input length ℓ_{in}, and the output length ℓ_{out}, the null-setup algorithm does the following:
  1. Sample keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$.
  2. Generate the inner level $(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{SetupNull}'(1^\lambda, 1^{s'}, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  3. For each inner block index $j \in [k]$, construct $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$.
  4. Compute the digest and openings $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$.
  5. Output the state $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C_\bot, k+1, 0, \mathsf{vk}, \mathsf{ek}, \mathsf{ek}')$, the auxiliary $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$, the verification key $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$, and the evaluation key ek.
- $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$: On input the security parameter λ, a size bound s, a branching factor k, the input length ℓ_{in}, the output length ℓ_{out}, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, and a cutoff index $t \in \{0, \ldots, N\}$, the setup algorithm does the following:
  1. Decompose the cutoff into $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$ and compute the reversed inner cutoff $\bar{t}_2 = k^{c-1} - t_2$.
  2. Sample keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$.
  3. Define the reversed inner circuit $\bar{C}_{t_1}\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N'] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ as
  $$\bar{C}_{t_1}(w,x) = C\big(w, (t_1 - 1)\cdot k^{c-1} + (k^{c-1} - x + 1)\big),$$
  where we adopt the convention that C outputs $\bot$ on any index outside $[N]$ (this only arises when $t_1 = k+1$, in which case every block is switched off and the inner level is never evaluated) and $\bar{C}_{t_1}$ has size at most s'.
  4. Generate the inner level $(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^\lambda, 1^{s'}, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, \bar{t}_2)$.
  5. For each inner block index $j \in [k]$, construct $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$.
  6. Compute the digest and openings $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$.
  7. Output the state $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C, t_1, t_2, \mathsf{vk}, \mathsf{ek}, \mathsf{ek}')$, the auxiliary $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$, the verification key $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$, and the evaluation key ek.
- $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$: On input a state $\mathsf{st} = (1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, k, C, t_1, t_2, \mathsf{vk}, \mathsf{ek}, \mathsf{ek}')$ with $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$, and a block index $j \in [k]$, the block-program obfuscation algorithm does the following:
  1. If $j < t_1$, let P_{null} be a trivial null circuit of size s_P, sample $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathrm{null}})$, and output the encrypted block program $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$.
  2. Otherwise, define the inner wrapper program G_j as follows:
     - If $j > t_1$, or $j = t_1$ with $t_2 = 0$ (a fully active block), let G_{null} be a trivial null circuit of size s_G and sample $G'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G_{\mathrm{null}})$.
     - If $j = t_1$ and $t_2 > 0$ (a partially active block), construct the circuit $G_j = G[\mathsf{hk}, \mathsf{dig}, j, \mathsf{ek}']$ described in Fig. 5 below and sample $G'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G_j)$:

Constants: A hash key hk, a digest dig, a block index $j \in [k]$, and an inner evaluation key ek'.
Inputs: An input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, an index $i \in [N]$, and an evaluation hint h.

1. Parse $h = ((D', \pi), h')$.
2. Decompose $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$ and compute the reversed inner index $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$ together with the inner block index $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$.
3. If $(i_1 = j) \wedge \mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{dig}, i'_1, D', \pi) = 1$, output $\mathsf{Eval}'(\mathsf{ek}', D', h', w, \bar{i}_2)$. Otherwise, output $\perp$.

Figure 5: Program $G[\mathsf{hk}, \mathsf{dig}, j, \mathsf{ek}']$.

3. Construct the circuit $P_j = P[C, \mathsf{vk}, j, G'_j]$ described in Fig. 6 below, sample $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_j)$, and output the encrypted block program $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$:

Constants: A circuit C, a verification key vk, a block index $j \in [k]$, and an obfuscated circuit G'_j.
Inputs: An input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, an index $i \in [N]$, and an evaluation hint h.

1. If $\mathsf{Verify}(\mathsf{vk}, h, i) = 0$, output $\perp$.
2. Decompose $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$.
3. If $(i_1 = j) \wedge (G'_j(w, i, h) = \perp)$, output $C(w, i)$. Otherwise, output $\perp$.

Figure 6: Program $P[C, \mathsf{vk}, j, G'_j]$.

- $\mathsf{Preprocess}(H, w, i)$: On input an auxiliary component $H = ((D'_j, \pi_j)_{j \in [k]}, H')$, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [N]$, the preprocessing algorithm does the following:
  1. Decompose $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$ and compute the reversed inner index $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$ together with its inner block index $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$.
  2. Compute the inner evaluation hint $h' = \mathsf{Preprocess}'(H', w, \bar{i}_2)$.
  3. Output $h = \big((D'_{i'_1}, \pi_{i'_1}), h'\big)$.
- $\mathsf{Verify}(\mathsf{vk}, h, i)$: On input a verification key $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$, an evaluation hint $h = ((D', \pi), h')$, and an index $i \in [N]$, the verification algorithm does the following:
  1. Decompose $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$ and compute the reversed inner index $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$ together with its inner block index $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$.
  2. Output $b = 1$ if $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{dig}, i'_1, D', \pi) = 1$ and $\mathsf{Verify}'(\mathsf{vk}', h', \bar{i}_2) = 1$; otherwise output $b = 0$.
- $\mathsf{Eval}(\mathsf{ek}, P, h, w, i)$: On input an evaluation key ek, a block program P, an evaluation hint h, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [N]$, the evaluation algorithm decrypts $P' = \mathsf{SKE.Dec}(\mathsf{ek}, P)$ and outputs $y = P'(w, i, h)$.

Finally, our security analysis will also use the following block function, which tests whether a value is the honestly generated inner block program at a given index. Its size bound is the parameter s_f fixed above.

Constants: A PRF key K and an inner state st'.
Inputs: A block index $j \in [k]$ and an inner block program D'.

1. If $D' = \mathsf{ObfBlockProgram}'(\mathsf{st}', j; \mathsf{PRF.Eval}(K, j))$, output 1. Otherwise, output 0.

Figure 7: Function $f_{K,\mathsf{st}'}$.

Theorem 4.11 (Correctness). *Suppose* Π'_{LCiO} *satisfies correctness and hint validity,* Π_{FBH} *is perfectly correct, and* NiO *and* Π_{SKE} *are correct. Then Construction 4.10 satisfies correctness.*

Proof. Take any $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, any cutoff $t \in \{0,\dots,N\}$, any input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and any index $i \in [N]$. Write $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$ and $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$, and, as in Construction 4.10, write $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$ for the reversed inner index, $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$ for its inner block index, and $\bar{t}_2 = k^{c-1} - t_2$ for the reversed inner cutoff. Let $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ be in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$ and let ct_{i_1} be in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, i_1)$, where $i_1 = \lceil i/k^{c-1} \rceil$ is the block containing i; write $\mathsf{ct}_{i_1} = \mathsf{SKE.Enc}(\mathsf{ek}, P'_{i_1})$ for the obfuscated program P'_{i_1} that it encrypts. Since ek is the key recorded in st, correctness of Π_{SKE} gives $\mathsf{SKE.Dec}(\mathsf{ek}, \mathsf{ct}_{i_1}) = P'_{i_1}$, so $\mathsf{Eval}(\mathsf{ek}, \mathsf{ct}_{i_1}, h, w, i) = P'_{i_1}(w, i, h)$. Parse $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$ and write $h = \mathsf{Preprocess}(H, w, i) = \big((D'_{i'_1}, \pi_{i'_1}), h'\big)$ with $h' = \mathsf{Preprocess}'(H', w, \bar{i}_2)$. By Theorem 4.12, $\mathsf{Verify}(\mathsf{vk}, h, i) = 1$, so the first step of Fig. 6 never rejects. Since NiO is functionality-preserving, we can reason about the behavior of the unobfuscated programs. Throughout, we use the fact that $i > t$ is equivalent to $(i_1 > t_1) \vee (i_1 = t_1 \wedge i_2 > t_2)$. We consider three cases, according to i_1:

- Suppose $i_1 < t_1$. Then $i \le t$ and P_{i_1} is a trivial null program, so the output is $\bot$, as required.
- Suppose $i_1 > t_1$, or $i_1 = t_1$ with $t_2 = 0$. Then $i > t$ and block i_1 is fully active, so G'_{i_1} is null. The check in Fig. 6 therefore passes and P_{i_1} outputs $C(w, i)$, as required.
- Suppose $i_1 = t_1$ and $t_2 > 0$. Then the program G_{t_1} is embedded in P_{t_1}, with the honest inner evaluation key ek′ hard-wired into it. By correctness of Π_{FBH}, FBH.Verify passes, so G'_{t_1} outputs $\mathsf{Eval}'(\mathsf{ek}', D'_{i'_1}, h', w, \bar{i}_2)$. Since the inner level was generated on $(\bar{C}_{t_1}, \bar{t}_2)$, correctness of Π'_{LCiO} gives that this equals $\bar{C}_{t_1}(w, \bar{i}_2)$ when $\bar{i}_2 > \bar{t}_2$ and $\bot$ otherwise. Now
$$\bar{i}_2 > \bar{t}_2 \iff k^{c-1} - i_2 + 1 > k^{c-1} - t_2 \iff i_2 \le t_2,$$
and $\bar{C}_{t_1}(w, \bar{i}_2) = C(w, (t_1 - 1)k^{c-1} + i_2) = C(w, i)$. Thus, G'_{t_1} outputs $C(w, i)$ when $i_2 \le t_2$ and $\bot$ when $i_2 > t_2$. Since P_{t_1} outputs $C(w, i)$ exactly when G'_{t_1} outputs $\bot$, it outputs $C(w, i)$ when $i_2 > t_2$ and $\bot$ when $i_2 \le t_2$ and $C(w, i) \ne \bot$. When $i_2 \le t_2$ and $C(w, i) = \bot$ both branches yield $\bot$. This is exactly the required behavior for the partially active block. □

Theorem 4.12 (Hint Validity). *Suppose* Π'_{LCiO} *satisfies hint validity and* Π_{FBH} *is perfectly correct. Then Construction 4.10 satisfies hint validity.*

Proof. Take any $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, any cutoff $t \in \{0,\dots,N\}$, any input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and any index $i \in [N]$. Write $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$ and, as in Construction 4.10, write $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$ for the reversed inner index and $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$ for its inner block index. Let $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ be in the support of either $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$ or $\mathsf{SetupNull}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$. Parse $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$ and $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$, and write $h = \mathsf{Preprocess}(H, w, i) = \big((D'_{i'_1}, \pi_{i'_1}), h'\big)$ with $h' = \mathsf{Preprocess}'(H', w, \bar{i}_2)$. In both cases the pairs $(D'_j, \pi_j)_{j\in[k]}$ are the honest digest openings computed by FBH.Hash, and H' is an honestly generated inner auxiliary component. By the perfect correctness of Π_{FBH}, the honest opening $\pi_{i'_1}$ verifies against dig at block i'_1, and by hint validity of Π'_{LCiO}, we have $\mathsf{Verify}'(\mathsf{vk}', h', \bar{i}_2) = 1$. Both checks of Verify therefore pass, so $\mathsf{Verify}(\mathsf{vk}, h, i) = 1$, as required. □

Theorem 4.13 (Endpoint Soundness). *Suppose* NiO *preserves functionality and* Π_{SKE} *is correct. Then Construction 4.10 satisfies endpoint soundness.*

Proof. Take any $\lambda, s, k, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \le s$, any input $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and any index $i \in [N]$. Write $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$, and note that $\lceil i/k^{c-1} \rceil = i_1$. Since NiO is functionality-preserving, we may argue about the unobfuscated programs. In both cases below, the evaluation key ek is the one output by the same execution of Setup that produced st, so by correctness of Π_{SKE} the evaluation algorithm recovers exactly the obfuscated program that ObfBlockProgram encrypted. We consider the two endpoints separately:

- **Soundness at the all-off cutoff.** Let $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ be in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, N)$, let $j \in [k]$, and let ct_j be in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, j)$, encrypting the obfuscated program P'_j. Since $\mathsf{decomp}^*_{k,c}(N) = (k+1, 0)$, we have $t_1 = k+1$, so every $j \in [k]$ satisfies $j < t_1$ and ObfBlockProgram takes the null branch. Hence P_j is a trivial null program and $\mathsf{Eval}(\mathsf{ek}, \mathsf{ct}_j, h, w, i) = P'_j(w, i, h) = \bot$ for every evaluation hint h.

- **Soundness at the all-on cutoff.** Let $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek})$ be in the support of $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, 0)$ and let ct_{i_1} be in the support of $\mathsf{ObfBlockProgram}(\mathsf{st}, i_1)$, encrypting the obfuscated program P'_{i_1}. Since $\mathsf{decomp}^*_{k,c}(0) = (1, 0)$, every block is fully active and every G_j is a trivial null circuit. The block program selected at index i is therefore $P_{i_1} = P[C, \mathsf{vk}, i_1, G'_{i_1}]$ with G'_{i_1} equal to the null program. Take any evaluation hint h with $\mathsf{Verify}(\mathsf{vk}, h, i) = 1$. Then the first step of Fig. 6 does not reject, and since $G'_{i_1}(w, i, h) = \bot$, we get $\mathsf{Eval}(\mathsf{ek}, \mathsf{ct}_{i_1}, h, w, i) = P'_{i_1}(w, i, h) = C(w, i)$.

□

Theorem 4.14 (Succinctness)**.** *Suppose that* NiO *expands a circuit of size S to a circuit of size at most $p_{\mathsf{NiO}}(\lambda, S)$ for a fixed polynomial p_{NiO}, that Π_{SKE} has keys of length at most $\mathsf{poly}_{\mathsf{ek}}(\lambda)$ and ciphertext expansion $\mathsf{poly}(\lambda)$, and that Π'_{LCiO} is succinct with polynomial poly'. Then Construction 4.10 is succinct, with evaluation key bound $\mathsf{poly}_{\mathsf{ek}}$ and with polynomial*

$$\mathsf{poly}(\lambda, s, \log k) = q\big(\lambda, s, \log k, \mathsf{poly}'(\lambda, s, \log k)\big)$$

for a fixed polynomial q that does not depend on c.

Proof. Write $S' = \mathsf{poly}'(\lambda, s, \log k)$. The values hashed at this level are the inner block programs themselves, so they have size $s_D \le S'$. The block function of Fig. 7 runs ObfBlockProgram′ and one equality test, so $s_f = \mathsf{poly}(\lambda, s, S')$ by the state and block program succinctness of Π'_{LCiO}, and a function-binding hash opening has size $\mathsf{poly}(\lambda, s_f, s_D, \log k)$. Each pair (D'_j, π_j) therefore has size $\mathsf{poly}(\lambda, s, \log k, S')$, and the same bound covers $(\mathsf{hk}, \mathsf{dig})$. We check the five bounds of Definition 4.1 in turn.

- **Evaluation key:** ek is sampled as $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$, so $|\mathsf{ek}| \le \mathsf{poly}_{\mathsf{ek}}(\lambda)$. Since KeyGen takes only the security parameter, this bound is independent of s, of k, and of S'.

- **Verification key:** $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ with $|\mathsf{vk}'| \le S'$, so $|\mathsf{vk}| \le \mathsf{poly}(\lambda, s, \log k, S')$.

- **Evaluation hint:** $h = ((D', \pi), h')$ with $|h'| \le S'$, giving the same bound.

- **State:** st has the description of a circuit of size at most $|C|$, descriptions of the cutoff index and k which have size at most $\log k$, the verification, and evaluation keys (for both the current level and the previous level) which we already bounded; in total, the state has size at most $\mathsf{poly}(\lambda, s, \log k)$.

- **Block programs:** The program G_j performs one function-binding hash verification and runs Eval′ on an inner block program of size at most S', with a hard-wired inner evaluation key of size $\mathsf{poly}_{\mathsf{ek}}(\lambda)$. Hence $\left|G_j\right|$, and therefore $\left|G'_j\right|$, is $\mathsf{poly}(\lambda, s, \log k, S')$. The program P_j hard-codes C, vk and G'_j, and runs Verify, which performs one function-binding hash verification and one call to Verify′ on inputs of size at most S'. So $\left|P_j\right| = \mathsf{poly}(\lambda, s, \log k, S')$; obfuscating preserves this up to p_{NiO}, and encrypting it under ek adds a further $\mathsf{poly}(\lambda)$. The algorithm ObfBlockProgram only constructs this program P_j, obfuscates it, and encrypts it, so it runs in time $\mathsf{poly}(\lambda, s, \log k)$.

- **Auxiliary component:** The auxiliary component H consists of k pairs (D'_j, π_j) together with H', and $|H'| \le k \cdot S'$; hence $|H| \le k \cdot \mathsf{poly}(\lambda, s, \log k, S')$.

Taking q to dominate the last five bounds gives the claim. □

4.3 Leveled Cutoff-$i\mathcal{O}$ Security

In this section we give the security analysis of Construction 4.10. Throughout, we keep the notation of Construction 4.10, where $N = k^c$ and $N' = k^{c-1}$. For a cutoff t we write $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$ for its block decomposition and $\bar{t}_2 = k^{c-1} - t_2$ for the reversed inner cutoff, and for a block index $a \in [k+1]$ we write

$$\bar{C}_a \colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N'] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}, \qquad \bar{C}_a(w, x) = C\big(w, (a-1)\cdot k^{c-1} + (k^{c-1} - x + 1)\big)$$

for the reversed inner circuit of block a, with the convention of Construction 4.10 that C outputs $\bot$ on any index outside $[N]$. For an index i we write $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$ for its block decomposition, $\bar{i}_2 = k^{c-1} - i_2 + 1 \in [N']$ for the reversed inner index, and $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil \in [k]$ for its inner block index. When two challenge circuits and cutoffs $(C^{(0)}, t^{(0)})$ and $(C^{(1)}, t^{(1)})$ are in play, the same notation is superscripted: $(t_1^{(b)}, t_2^{(b)}) = \mathsf{decomp}^*_{k,c}(t^{(b)})$, $\bar{t}_2^{(b)} = k^{c-1} - t_2^{(b)}$, and $\bar{C}_a^{(b)}$ is the reversed inner circuit of block a built from $C^{(b)}$. We write ek for the evaluation key of this level and ek' for that of the inner level Π'_{LCiO}. Finally, ρ denotes the number of random coins consumed by $\mathsf{ObfBlockProgram}'$, so that a PRF key used to derive them is sampled as $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{\lceil \log k \rceil}, 1^\rho)$. Here ρ also covers the coins that $\mathsf{ObfBlockProgram}'$ uses to encrypt its output under ek', so fixing the coins fixes the inner block program as a string.

Theorem 4.15 (Functionality Hiding)**.** *Assume that Π'_{LCiO} satisfies functionality hiding and that Π_{SKE} is CPA-secure. Then Construction 4.10 satisfies functionality hiding.*

Proof. We define the following sequence of hybrids:

- Hyb_0: This is the functionality hiding game with bit $b = 0$ and level $c > 1$.
  1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, a pair of circuits $C^{(0)}, C^{(1)} \colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ bounded by size s, and a pair of cutoffs $t^{(0)}, t^{(1)} \in \{0, \ldots, N\}$ to the challenger.
  2. The challenger samples the evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and the function-binding hash key $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$ in normal mode.
  3. The challenger decomposes $(t_1^{(0)}, t_2^{(0)}) = \mathsf{decomp}^*_{k,c}(t^{(0)})$ and computes the reversed inner cutoff $\bar{t}_2^{(0)} = k^{c-1} - t_2^{(0)}$ and the reversed inner circuit $\bar{C}^{(0)}_{t_1^{(0)}}$ of block $t_1^{(0)}$, both of which are efficiently and deterministically computable from $C^{(0)}$ and $t^{(0)}$, and recursively generates the inner level
  $$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'\Big(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}^{(0)}_{t_1^{(0)}}, \bar{t}_2^{(0)}\Big).$$
  4. For each $j \in [k]$, the challenger samples the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$.
  5. The challenger computes the digest and openings $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$, and assembles the verification key $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and the auxiliary component $H = \big((D'_j, \pi_j)_{j \in [k]}, H'\big)$.
  6. For each $j \in [k]$, the challenger constructs the obfuscated block program P'_j exactly as $\mathsf{ObfBlockProgram}$ does on the state associated with $(C^{(0)}, t^{(0)})$: a null program for $j < t_1^{(0)}$, and $P[C^{(0)}, \mathsf{vk}, j, G'_j]$ for $j \geq t_1^{(0)}$, where G'_j is the real wrapper $G[\mathsf{hk}, \mathsf{dig}, t_1^{(0)}, \mathsf{ek}']$ if $j = t_1^{(0)}$ and $t_2^{(0)} > 0$, and a null wrapper otherwise. It then encrypts $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$.
  7. The challenger sends $(H, \mathsf{vk}, \{\mathsf{ct}_j\}_{j \in [k]})$ to $\mathcal{A}$.
  8. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_1: This is the same as Hyb_0, except the challenger replaces the block programs with dummy ciphertexts.
  6. For each $j \in [k]$, the challenger encrypts a dummy message $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}\big(\mathsf{ek}, 0^{\hat{s}_P}\big)$.

  This hybrid is indistinguishable from the previous one by the CPA-security of Π_{SKE}.

- Hyb_2: This is the same as Hyb_1, except the challenger replaces the entire inner level with the one derived from $(C^{(1)}, t^{(1)})$.

  3. The challenger decomposes $(t_1^{(1)}, t_2^{(1)}) = \mathsf{decomp}^*_{k,c}(t^{(1)})$ and computes the reversed inner cutoff $\bar{t}_2^{(1)} = k^{c-1} - t_2^{(1)}$ and the reversed inner circuit $\bar{C}^{(1)}_{t_1^{(1)}}$ of block $t_1^{(1)}$, both of which are efficiently and deterministically computable from $C^{(1)}$ and $t^{(1)}$, and recursively generates the inner level

  $$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'\left(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}^{(1)}_{t_1^{(1)}}, \bar{t}_2^{(1)}\right)$$

  This hybrid is indistinguishable from the previous one by functionality hiding of Π'_{LCiO}.

- Hyb_3: This is the same as Hyb_2, except the challenger replaces the dummy ciphertexts with encryptions of the true block programs, now built from $(C^{(1)}, t^{(1)})$. This distribution corresponds exactly to the functionality hiding game with bit $b = 1$.

  6. For each $j \in [k]$, the challenger constructs the obfuscated block program P'_j as in Hyb_0 but with respect to $(C^{(1)}, t^{(1)})$, and encrypts $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$.

  This hybrid is indistinguishable from the previous one by the CPA-security of Π_{SKE}.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of Hyb_i with adversary $\mathcal{A}$. Below, we analyze each adjacent pair of hybrid experiments. Once we prove Claims 4.16 to 4.18, the main theorem follows by a standard hybrid argument.

Claim 4.16. *Assume that Π_{SKE} is CPA-secure. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. We construct an algorithm $\mathcal{B}$ that breaks CPA-security with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, a pair of circuits $C^{(0)}, C^{(1)} : \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\perp\}$ bounded by size s, and a pair of cutoffs $t^{(0)}, t^{(1)} \in \{0, \ldots, N\}$ to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ samples the function-binding hash key $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$ in normal mode.
3. Algorithm $\mathcal{B}$ computes the reversed cutoff $\bar{t}_2^{(0)}$ and inner circuit $\bar{C}^{(0)}_{t_1^{(0)}}$ as in Construction 4.10, where $(t_1^{(0)}, t_2^{(0)}) = \mathsf{decomp}^*_{k,c}(t^{(0)})$, and recursively generates the inner level

   $$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'\left(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}^{(0)}_{t_1^{(0)}}, \bar{t}_2^{(0)}\right).$$

4. Algorithm $\mathcal{B}$ samples the inner block programs $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$ for all $j \in [k]$, computes the digest and openings $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$, and assembles $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and $H = \left((D'_j, \pi_j)_{j \in [k]}, H'\right)$.
5. Algorithm $\mathcal{B}$ constructs the obfuscated block programs P'_j for all $j \in [k]$ exactly as in Hyb_0, which it can do since it generated the inner level itself and therefore knows both st' and ek'. For each $j \in [k]$, it makes a challenge query $(P'_j, 0^{\hat{s}_P})$, to which the challenger responds with a ciphertext ct_j. Both messages have length $\hat{s}_P$, so the query is admissible.
6. Algorithm $\mathcal{B}$ sends $(H, \mathsf{vk}, \{\mathsf{ct}_j\}_{j \in [k]})$ to $\mathcal{A}$.
7. Algorithm $\mathcal{A}$ outputs a bit b' which algorithm $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$, and $\mathcal{B}$ never uses the encryption key. In addition, if the CPA challenger samples ct_j as an encryption of P'_j for all $j \in [k]$, then algorithm $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$, whereas if the CPA challenger samples ct_j as an encryption of $0^{\hat{s}_P}$ for all $j \in [k]$, then algorithm $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Therefore we conclude that $\mathcal{B}$ breaks CPA-security with the same advantage ε. □

Claim 4.17. *Assume that Π'_{LCiO} satisfies functionality hiding. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right|$. We construct an adversary $\mathcal{B}$ for the functionality hiding game of Π'_{LCiO} that achieves the same advantage ε. Algorithm $\mathcal{B}$ interacts with the functionality hiding challenger for Π'_{LCiO} and simulates the experiment for $\mathcal{A}$ as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, a pair of circuits $C^{(0)}, C^{(1)} \colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ bounded by size s, and a pair of cutoffs $t^{(0)}, t^{(1)} \in \{0, \ldots, N\}$ to $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ samples the evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and the function-binding hash key $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_F}, 1^{s_D}, k)$ in normal mode.

3. Algorithm $\mathcal{B}$ computes the reversed cutoffs $\bar{t}_2^{(0)}, \bar{t}_2^{(1)}$ and inner circuits $\bar{C}^{(0)}_{t_1^{(0)}}, \bar{C}^{(1)}_{t_1^{(1)}}$ as in Construction 4.10, where $(t_1^{(0)}, t_2^{(0)}) = \mathsf{decomp}^*_{k,c}(t^{(0)})$ and $(t_1^{(1)}, t_2^{(1)}) = \mathsf{decomp}^*_{k,c}(t^{(1)})$. Algorithm $\mathcal{B}$ then forwards $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuits $\bar{C}^{(0)}_{t_1^{(0)}}, \bar{C}^{(1)}_{t_1^{(1)}}$ and the cutoffs $\bar{t}_2^{(0)}, \bar{t}_2^{(1)}$ as its challenge to the functionality hiding challenger for Π'_{LCiO}, which responds with the inner auxiliary component, verification key and block programs $(H', \mathsf{vk}', \{D'_j\}_{j\in[k]})$.

4. Algorithm $\mathcal{B}$ computes the digest and openings $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$, and assembles $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$.

5. Algorithm $\mathcal{B}$ samples the dummy ciphertexts $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}\big(\mathsf{ek}, 0^{\hat{s}_P}\big)$ for all $j \in [k]$, and sends $(H, \mathsf{vk}, \{\mathsf{ct}_j\}_{j\in[k]})$ to $\mathcal{A}$. Note that $\mathcal{B}$ requires neither the inner state st' nor the inner evaluation key ek' for this step (this is precisely why the block programs are first replaced by dummies in Hyb_1, since the functionality hiding challenger withholds ek' and returns no state).

6. Algorithm $\mathcal{A}$ outputs a bit b' which algorithm $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. In addition, if the functionality hiding challenger for Π'_{LCiO} generates the inner level by executing Setup' on $\bar{C}^{(0)}_{t_1^{(0)}}, \bar{t}_2^{(0)}$ (i.e., the bit $b = 0$ game), then algorithm $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$, whereas if it executes Setup' on $\bar{C}^{(1)}_{t_1^{(1)}}, \bar{t}_2^{(1)}$ (i.e., the bit $b = 1$ game), then algorithm $\mathcal{B}$ simulates $\mathsf{Hyb}_2(\mathcal{A})$. Therefore $\mathcal{B}$ breaks the functionality hiding of Π'_{LCiO} with the same advantage ε. □

Claim 4.18. *Assume that Π_{SKE} is CPA-secure. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

The proof of Claim 4.18 is completely analogous to that of Claim 4.16, with the inner level generated from $(\bar{C}^{(1)}_{t_1^{(1)}}, \bar{t}_2^{(1)})$ and the block programs built from $(C^{(1)}, t^{(1)})$. □

Theorem 4.19 (Index Indistinguishability)**.** *Assume that Π'_{LCiO} satisfies functionality hiding, index indistinguishability and endpoint soundness, NiO is a secure null-iO, Π_{FBH} satisfies mode indistinguishability and perfect function binding, and that there exists a secure pseudorandom function Π_{PRF}. Then Construction 4.10 satisfies index indistinguishability.*

Proof. We divide the adversaries for the index indistinguishability game into three types:

- **Intra-block adversaries:** This adversary submits k, t such that $\mathsf{decomp}^*_{k,c}(t) = (t_1, t_2)$ where $t_2 \in [2, k^{c-1} - 1]$ (i.e., all cases not covered by start-of-block and inter-block). This simply requires invoking the index indistinguishability of Π'_{LCiO}.
- **Start-of-block adversaries:** This adversary submits k, t such that $\mathsf{decomp}^*_{k,c}(t) = (t_1, 1)$ (i.e., t is the first index of block t_1). In this case we have $\mathsf{decomp}^*_{k,c}(t-1) = (t_1, 0)$, so block t_1 is fully active at cutoff $t-1$ and partially active at cutoff t. The transition from cutoff $t-1$ to t therefore requires changing the wrapper program G'_{t_1} from a null program to $G_{t_1} = G[\mathsf{hk}, \mathsf{dig}, t_1, \mathsf{ek}']$ described in Fig. 5, and then invoking the index indistinguishability of Π'_{LCiO} similar to the intra-block argument.
- **Inter-block adversaries:** This adversary submits k, t such that $\mathsf{decomp}^*_{k,c}(t) = (t_1, 0)$. In this case, we have $\mathsf{decomp}^*_{k,c}(t-1) = (t_1 - 1, k^{c-1} - 1)$ which means that the target block of t and of $t-1$ are different blocks. This requires switching the block program of block $t_1 - 1$ from a functional program to a null program. Note that because $\mathsf{decomp}^*_{k,c}(N) = (k+1, 0)$, the transition from $t = N-1$ to $t = N$ correctly triggers this inter-block logic.

Lemma 4.20 (Intra-Block Indistinguishability)**.** *Assume that Π'_{LCiO} satisfies index indistinguishability. Then for any efficient adversary $\mathcal{A}$ that sends a cutoff index $t \in [N]$ such that $\mathsf{decomp}^*_{k,c}(t) = (t_1, t_2)$ where $t_2 \in [2, k^{c-1} - 1]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Real}_0(\lambda) = 1] - \Pr[\mathsf{Real}_1(\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Real}_b(\lambda)$ is the output distribution of the index indistinguishability game with parameter b.

Proof. Consider the setup algorithm Setup for cutoffs t and $t-1$. Since $t_2 \in [2, k^{c-1}-1]$, we have $\mathsf{decomp}^*_{k,c}(t) = (t_1, t_2)$ and $\mathsf{decomp}^*_{k,c}(t-1) = (t_1, t_2 - 1)$ with $t_2, t_2 - 1 \in [1, k^{c-1} - 1]$, so in both executions block t_1 is partially active and carries the same real wrapper G'_{t_1}. In particular, the target block t_1 is identical in both distributions. The only difference between the two executions lies in the cutoff passed to the inner scheme Π'_{LCiO}.

- In Setup for cutoff t, the inner call is $\mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, \bar{t}_2)$, where $\bar{t}_2 = k^{c-1} - t_2$ and $\bar{C}_{t_1}(w, x) = C\big(w, (t_1 - 1) \cdot k^{c-1} + (k^{c-1} - x + 1)\big)$ is the reversed inner circuit of block t_1.
- In Setup for cutoff $t-1$, the inner call is $\mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, \bar{t}_2 + 1)$.

Because the top-level block t_1 is unchanged, the circuit $\bar{C}_{t_1}$ is identical in both executions. Furthermore, the logic assembling the auxiliary component H and all block programs P_j behaves strictly identically given the inner outputs.

We now describe the reduction algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ interacts with the Π'_{LCiO} challenger and simulates the index indistinguishability game for $\mathcal{A}$ as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and receives $(1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$.
2. Algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$ and defines the reversed inner circuit $\bar{C}_{t_1}(w, x) = C(w, (t_1 - 1) \cdot k^{c-1} + (k^{c-1} - x + 1))$.
3. Algorithm $\mathcal{B}$ sends $(1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1} - t_2 + 1)$ to the Π'_{LCiO} challenger. (Notice that the order of the hybrid step is inverted for the inner reversed circuit.) The challenger replies with the inner components $(\mathsf{ek}', H', \mathsf{vk}', \{D'_j\}_{j \in [k]})$.
4. Algorithm $\mathcal{B}$ generates the rest of the top-level components exactly as in Setup:
   - It samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$ and computes the function-binding hash digest dig and openings of $(D'_1, \ldots, D'_k)$.
   - It samples its own evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$.
   - It assembles the state st (which records ek and ek′), the auxiliary component $H = \big((D'_j, \pi_j)_{j \in [k]}, H'\big)$, and the verification key $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$.
   - It computes all top-level block programs $P_j \leftarrow \mathsf{ObfBlockProgram}(\mathsf{st}, j)$.

5. Finally, algorithm $\mathcal{B}$ outputs $(\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j\in[k]})$ to $\mathcal{A}$, and outputs $\mathcal{A}$'s guess b' to the $(c-1)$-leveled cutoff-$i\mathcal{O}$ challenger.

The two experiments abort under the same condition: the Π'_{LCiO} game toggles between the reversed inner cutoffs $\bar{t}_2 + 1$ and $\bar{t}_2$ (i.e., the inner index $\bar{t} = k^{c-1} - t_2 + 1$), and it aborts if $\bar{C}_{t_1}(w, \bar{t}) \neq \bot$ for some $w \in \{0,1\}^{\ell_{\mathsf{in}}}$. By definition,

$$\bar{C}_{t_1}(w, \bar{t}) = C\big(w, (t_1 - 1) \cdot k^{c-1} + t_2\big) = C(w, t),$$

so the Π'_{LCiO} challenger aborts and outputs 0 precisely when the level-c game does, and $\mathcal{B}$ never has to decide whether such a w exists. If the Π'_{LCiO} challenger evaluated the higher reversed cutoff $\bar{t}_2 + 1$, $\mathcal{B}$ perfectly simulates the distribution for $t - 1$. If the challenger evaluated the lower reversed cutoff $\bar{t}_2$, $\mathcal{B}$ perfectly simulates t. By the inductive security assumption on Π'_{LCiO}, the advantage of $\mathcal{B}$ is bounded by $\mathsf{negl}(\lambda)$, which concludes the proof. □

Lemma 4.21 (Start-of-Block Indistinguishability). *Assume that Π'_{LCiO} satisfies index indistinguishability and endpoint soundness, NiO is a secure null-$i\mathcal{O}$, Π_{FBH} satisfies mode indistinguishability and perfect function binding, and that there exists a secure pseudorandom function Π_{PRF}. Then for any efficient adversary $\mathcal{A}$ that sends a cutoff index $t \in [N]$ such that $\mathsf{decomp}^*_{k,c}(t) = (t_1, 1)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Real}_0(\lambda) = 1] - \Pr[\mathsf{Real}_1(\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Real}_b(\lambda)$ is the output distribution of the index indistinguishability game with parameter b.

Proof. Let $\mathcal{A}$ be an adversary for the index indistinguishability game as described above. We define the following sequence of hybrids:

- Hyb_0: This is the distribution corresponding to the real index indistinguishability game with parameter $b = 1$, where $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t-1)$ is executed. By the assumption on t, we have that $\mathsf{decomp}^*_{k,c}(t-1) = (t_1, 0)$, so block t_1 is fully active at cutoff $t-1$. Specifically, the adversary starts by outputting $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C$ and the challenger responds as follows:

  1. The challenger samples keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$.
  2. The challenger sets the reversed inner cutoff $\bar{t}_2 = k^{c-1}$.
  3. The challenger defines the reversed inner circuit $\bar{C}_{t_1} : \{0,1\}^{\ell_{\mathsf{in}}} \times \left[k^{c-1}\right] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ as $\bar{C}_{t_1}(w, x) = C(w, (t_1 - 1) \cdot k^{c-1} + (k^{c-1} - x + 1))$.
  4. The challenger generates the inner level:
  $$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, \bar{t}_2)$$
  5. For each inner block index $j \in [k]$, the challenger constructs the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$.
  6. The challenger computes $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$.
  7. The challenger sets $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$.
  8. For each block index $j \in [k]$ with $j < t_1$, the challenger sets P_{null} to be a trivial null circuit of size s_P and sets $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathsf{null}})$.
  9. For each block index $j \in [k]$ with $j \geq t_1$, the challenger constructs the circuit $P_j = P[C, \mathsf{vk}, j, G'_j]$ from Fig. 6 where G_j is a trivial null circuit of size s_G, obfuscated as $G'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G_j)$, and samples $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_j)$.
  10. The challenger encrypts each block program as $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$ for all $j \in [k]$.
  11. The challenger outputs $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$.

  At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- Hyb_1: This is the same as Hyb_0, except that the challenger uses a pseudorandom function to derive the randomness for the generation of the inner programs D'_j. Specifically, the challenger makes the following changes:

  5. The challenger samples a PRF key $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{\lceil \log k \rceil}, 1^\rho)$ and, for each inner block index $j \in [k]$, constructs the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j; \mathsf{PRF.Eval}(K, j))$.

- Hyb_2: This is the same as Hyb_1, except that the function-binding hash key is generated in binding mode to ensure that the hashed values are the honestly generated inner block programs. Specifically, the challenger makes the following changes:

  1. The challenger samples keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, 1^{s_f}, 1^{s_D}, k, f_{K,\mathsf{st}'})$, where the function $f_{K,\mathsf{st}'}(j, D')$ is defined in Fig. 7.

- Hyb_3: This is the same as Hyb_2, except that the challenger replaces block t_1's wrapper circuit with the real wrapper program. Specifically, the challenger makes the following changes:

  9. For each block index $j \in [k]$ with $j \geq t_1$, the challenger constructs the circuit $P_j = P[C, \mathsf{vk}, j, G'_j]$ from Fig. 6 where G'_j is constructed as follows:
     - If $j = t_1$, the challenger constructs the circuit $G_j = G[\mathsf{hk}, \mathsf{dig}, j, \mathsf{ek}']$ from Fig. 5.
     - If $j > t_1$, the challenger sets G_j to be a trivial null circuit of size s_G.
     - In either case, the challenger obfuscates $G'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G_j)$.

  This transition relies on the security of NiO applied to G'_{t_1}.

- Hyb_4: This is the same as Hyb_3, except that the challenger returns the function-binding hash key to the normal setup mode. Specifically, the challenger makes the following changes:

  1. The challenger samples keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$.

- Hyb_5: This is the same as Hyb_4, except that the challenger replaces the pseudorandom coins used for generating the inner programs D'_j with true randomness, relying on the pseudorandomness of the PRF. Specifically, the challenger makes the following changes:

  5. For each $j \in [k]$, the challenger samples the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$. Specifically, $\mathsf{ObfBlockProgram}'$ uses true randomness.

- Hyb_6: This is the same as Hyb_5, except that the reversed inner cutoff is set to $k^{c-1} - 1$, so that the inner level is generated at cutoff $k^{c-1} - 1$. This transition relies on the index indistinguishability of Π'_{LCiO} and the promise that $C(w, t) = \bot$ for all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$.

  2. The challenger sets the reversed inner cutoff $\bar{t}_2 = k^{c-1} - 1$.

As usual, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of Hyb_i with adversary $\mathcal{A}$. The distribution in Hyb_6 is exactly the distribution generated by $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$.

Claim 4.22. *Assume that Π_{PRF} is a secure pseudorandom function. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. We construct an algorithm $\mathcal{B}$ that breaks the pseudorandomness of Π_{PRF} with the same advantage ε. Algorithm $\mathcal{B}$ participates in the pseudorandomness game and simulates the experiment for $\mathcal{A}$ as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$; algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, with $t_2 = 1$ by assumption.

2. Algorithm $\mathcal{B}$ outputs the input length $1^{\lceil \log k \rceil}$ and the output length 1^{ρ} to the pseudorandomness challenger, which samples a key $K \leftarrow \mathsf{PRF.Setup}(1^{\lambda}, 1^{\lceil \log k \rceil}, 1^{\rho})$.

3. Algorithm $\mathcal{B}$ samples $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^{\lambda})$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^{\lambda}, 1^{s_f}, 1^{s_D}, k)$, sets the reversed inner cutoff $\bar{t}_2 = k^{c-1}$, constructs the reversed inner circuit $\bar{C}_{t_1}$ of block t_1 as in Hyb_0, and generates $(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^{\lambda}, 1^{s}, 1^{k}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1})$.

4. For each inner block index $j \in [k]$, algorithm $\mathcal{B}$ makes the evaluation query j, receiving $r_j \in \{0,1\}^{\rho}$, and constructs $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j\,; r_j)$.

5. Algorithm $\mathcal{B}$ completes the experiment exactly as in Hyb_0 (computing the digest, openings, and auxiliary component, and constructing the top-level block programs with null wrappers in every block $j \geq t_1$ and null programs in every block $j < t_1$, then encrypting them under ek) and sends $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$.

6. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$, and the k evaluation queries are distinct. If the challenger answers with truly random r_j (bit $b = 0$), then each D'_j is distributed as a fresh $\mathsf{ObfBlockProgram}'(\mathsf{st}', j)$ and $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$. If it answers $r_j = \mathsf{PRF.Eval}(K, j)$ (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Thus algorithm $\mathcal{B}$ breaks pseudorandomness with the same advantage ε. □

Claim 4.23. *Assume that Π_{FBH} satisfies mode indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right|$. The only difference between Hyb_1 and Hyb_2 is that the function-binding hash key is sampled in normal mode via $\mathsf{FBH.Setup}$ in Hyb_1, whereas in Hyb_2 it is sampled in binding mode as $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^{\lambda}, 1^{s_f}, 1^{s_D}, k, f_{K,\mathsf{st}'})$. Algorithm $\mathcal{B}$ samples all other components itself and obtains the hash key from the mode indistinguishability challenger.

1. On input the security parameter 1^{λ}, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^{λ}. Algorithm $\mathcal{A}$ starts by sending $1^{s}, 1^{k}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$; algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^{*}_{k,c}(t)$, with $t_2 = 1$ by assumption.

2. Algorithm $\mathcal{B}$ samples $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^{\lambda})$ and a PRF key $K \leftarrow \mathsf{PRF.Setup}(1^{\lambda}, 1^{\lceil \log k \rceil}, 1^{\rho})$, sets $\bar{t}_2 = k^{c-1}$, constructs the reversed inner circuit $\bar{C}_{t_1}$ of block t_1 as in Hyb_0, and generates

$$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^{\lambda}, 1^{s}, 1^{k}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1}).$$

3. Algorithm $\mathcal{B}$ constructs the function $f_{K,\mathsf{st}'}$ from Fig. 7 and sends $1^{s_f}, 1^{s_D}, k$ and $f_{K,\mathsf{st}'}$ to the mode indistinguishability challenger, which replies with a hash key hk.

4. Using hk, algorithm $\mathcal{B}$ completes the experiment exactly as in Hyb_1: for each $j \in [k]$ it sets

$$D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j\,; \mathsf{PRF.Eval}(K, j)),$$

   then computes the digest, openings, and auxiliary component, constructs the top-level block programs with a null wrapper in every block $j \geq t_1$, encrypts them under ek, and sends $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If the challenger samples hk in normal mode (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$, whereas in binding mode (bit $b = 1$) it simulates $\mathsf{Hyb}_2(\mathcal{A})$. Therefore ε is negligible by the mode indistinguishability of Π_{FBH}. □

Claim 4.24. *Assume that NiO is a secure null-iO, that Π_{FBH} satisfies perfect function binding, and that Π'_{LCiO} satisfies endpoint soundness. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right|$. The only difference between Hyb_2 and Hyb_3 is the wrapper circuit embedded in block t_1: in Hyb_2 it is a trivial null circuit $C_\perp$, whereas in Hyb_3 it is $G_{t_1} = G[\mathsf{hk}, \mathsf{dig}, t_1, \mathsf{ek}']$ from Fig. 5 (both obfuscated under NiO at size s_G). We construct an algorithm $\mathcal{B}$ that breaks the security of NiO with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$; algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, with $t_2 = 1$ by assumption.

2. Algorithm $\mathcal{B}$ samples $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and a PRF key $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{\lceil \log k \rceil}, 1^\rho)$, sets $\bar{t}_2 = k^{c-1}$, constructs the reversed inner circuit $\bar{C}_{t_1}$ of block t_1 as in Hyb_0, and generates

$$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1}).$$

   It then constructs $f_{K,\mathsf{st}'}$ from Fig. 7 and samples the hash key in binding mode as

$$\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, 1^{s_f}, 1^{s_D}, k, f_{K,\mathsf{st}'}).$$

3. For each $j \in [k]$, algorithm $\mathcal{B}$ constructs $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j\,; \mathsf{PRF.Eval}(K, j))$ and computes the digest, openings, and auxiliary component H.

4. Algorithm $\mathcal{B}$ submits the two challenge circuits $C_0 = C_\perp$ and $C_1 = G[\mathsf{hk}, \mathsf{dig}, t_1, \mathsf{ek}']$, both of size s_G, to the null-$i\mathcal{O}$ challenger, which replies with $G'_{t_1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, C_b)$.

5. Algorithm $\mathcal{B}$ assembles the top-level block programs as in Hyb_2, embedding the challenge circuit in block t_1: for $j < t_1$ a null program; for $j > t_1$ a null wrapper G'_j and $P_j = P[C, \mathsf{vk}, j, G'_j]$; and for $j = t_1$, $P_{t_1} = P[C, \mathsf{vk}, t_1, G'_{t_1}]$ using the challenge. It encrypts each obfuscated block program under ek and sends $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$.

6. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If the challenger obfuscates $C_0 = C_\perp$ (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_2(\mathcal{A})$; if it obfuscates C_1 (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_3(\mathcal{A})$. It remains to verify that the NiO experiment does not abort (i.e., that C_0 and C_1 are null circuits). For C_0 this is immediate. For $C_1 = G_{t_1} = G[\mathsf{hk}, \mathsf{dig}, t_1, \mathsf{ek}']$, fix an arbitrary input (w, i, h) with $h = ((D', \pi), h')$, and let $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$, $\bar{i}_2 = k^{c-1} - i_2 + 1$, and $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil$. By the description of G in Fig. 5, if $i_1 \neq t_1$ or $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{dig}, i'_1, D', \pi) = 0$, then G_{t_1} outputs $\perp$; so it suffices to consider inputs with $i_1 = t_1$ and $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{dig}, i'_1, D', \pi) = 1$. Since hk is binding for $f_{K,\mathsf{st}'}$ and every honest block satisfies $f_{K,\mathsf{st}'}(j, D'_j) = 1$, the perfect function binding of Π_{FBH} forces any accepted opening at block i'_1 to satisfy $D' = \mathsf{ObfBlockProgram}'(\mathsf{st}', i'_1\,; \mathsf{PRF.Eval}(K, i'_1)) = D'_{i'_1}$, the honest inner block program. Since the inner level is generated at reversed cutoff k^{c-1}, which is the all-off cutoff for Π'_{LCiO}, and ek′ is the evaluation key output by that same execution of Setup′, the soundness of Π'_{LCiO} at the all-off cutoff gives $\mathsf{Eval}'(\mathsf{ek}', D'_{i'_1}, h', w, \bar{i}_2) = \perp$. Hence G_{t_1} outputs $\perp$ on every input, so C_1 is null and the experiment does not abort. We conclude that ε is negligible by the security of NiO. □

Claim 4.25. *Assume that Π_{FBH} satisfies mode indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. The proof is completely analogous to that of Claim 4.23. □

Claim 4.26. *Assume that Π_{PRF} is a secure pseudorandom function. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. The proof is completely analogous to that of Claim 4.22. □

Claim 4.27. *Assume that Π'_{LCiO} satisfies index indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\right|$. The only difference between Hyb_5 and Hyb_6 is the reversed inner cutoff used to generate the inner level: k^{c-1} in Hyb_5 and $k^{c-1} - 1$ in Hyb_6, both with the same inner circuit $\bar{C}_{t_1}$ and inner programs generated using true randomness. This is an index indistinguishability step on Π'_{LCiO}, following the same template as Lemma 4.20. We construct an adversary $\mathcal{B}$ for the index indistinguishability game of Π'_{LCiO} that achieves the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, where $t_2 = 1$ by the assumption on t, so that $t = (t_1 - 1) \cdot k^{c-1} + 1$.

2. Algorithm $\mathcal{B}$ defines the reversed inner circuit $\bar{C}_{t_1}(w, x) = C(w, (t_1 - 1) \cdot k^{c-1} + (k^{c-1} - x + 1))$ and sends $(1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1})$ to the index indistinguishability challenger for Π'_{LCiO}, which replies with the inner components $(\mathsf{ek}', H', \mathsf{vk}', \{D'_j\}_{j\in[k]})$, generated at reversed cutoff either k^{c-1} or $k^{c-1} - 1$.

3. Algorithm $\mathcal{B}$ generates the rest of the top-level components exactly as in Hyb_5:

   - First, it samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$ in normal mode, computes the digest and openings of $(D'_1, \ldots, D'_k)$, and assembles $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and the auxiliary component H.
   - Then, it samples its own evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$.
   - Finally, it constructs the top-level block programs: for $j < t_1$ a null program; for $j = t_1$ the program $P_{t_1} = P[C, \mathsf{vk}, t_1, G'_{t_1}]$ with the real wrapper $G'_{t_1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G[\mathsf{hk}, \mathsf{dig}, t_1, \mathsf{ek}'])$; and for $j > t_1$ the program $P_j = P[C, \mathsf{vk}, j, G'_j]$ with a null wrapper. It encrypts each obfuscated block program under ek.

4. Algorithm $\mathcal{B}$ outputs $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$, and forwards $\mathcal{A}$'s guess b' to the Π'_{LCiO} challenger.

The two experiments abort under the same condition: the Π'_{LCiO} game toggles between reversed inner cutoffs k^{c-1} and $k^{c-1} - 1$ (i.e., the inner index k^{c-1}), and it aborts if $\bar{C}_{t_1}(w, k^{c-1}) \neq \bot$ for some $w \in \{0,1\}^{\ell_{\mathsf{in}}}$. By definition,

$$\bar{C}_{t_1}(w, k^{c-1}) = C\big(w, (t_1 - 1) \cdot k^{c-1} + (k^{c-1} - k^{c-1} + 1)\big) = C\big(w, (t_1 - 1) \cdot k^{c-1} + 1\big) = C(w, t),$$

so the Π'_{LCiO} challenger aborts and outputs 0 precisely when the level-c game does. If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. Moreover, if the Π'_{LCiO} challenger uses reversed cutoff k^{c-1}, then $\mathcal{B}$ simulates $\mathsf{Hyb}_5(\mathcal{A})$, and if it uses reversed cutoff $k^{c-1} - 1$, then $\mathcal{B}$ simulates $\mathsf{Hyb}_6(\mathcal{A})$. Therefore $\mathcal{B}$ breaks the index indistinguishability of Π'_{LCiO} with the same advantage ε. □

By Claims 4.22 to 4.27, the lemma follows by a standard hybrid argument over $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_6$. □

Lemma 4.28 (Inter-Block Indistinguishability). *Assume that Π'_{LCiO} satisfies functionality hiding, index indistinguishability and endpoint soundness, NiO is a secure null-iO, Π_{FBH} satisfies mode indistinguishability and perfect function binding, and that there exists a secure pseudorandom function Π_{PRF}. Then for any efficient adversary $\mathcal{A}$ that sends a cutoff index $t \in [N]$ such that $\mathsf{decomp}^*_{k,c}(t) = (t_1, 0)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Real}_0(\lambda) = 1] - \Pr[\mathsf{Real}_1(\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Real}_b(\lambda)$ is the output distribution of the index indistinguishability game with parameter b.

Proof. Let $\mathcal{A}$ be an adversary for the index indistinguishability game as described above. We define the following sequence of hybrids:

- Hyb_0: This is the distribution corresponding to the real index indistinguishability game with parameter $b = 1$, where $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t - 1)$ is executed. By the assumption on t, we have that $\mathsf{decomp}^*_{k,c}(t - 1) = (t_1 - 1, k^{c-1} - 1)$, so block $t_1 - 1$ is partially active and every block $j \geq t_1$ is fully active. Specifically, the adversary starts by outputting $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C$ and the challenger responds as follows:

  1. The challenger samples keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$.
  2. The challenger sets the reversed inner cutoff $\bar{t}_2 = 1$.

3. The challenger defines the reversed inner circuit $\bar{C}_{t_1-1}\colon \{0,1\}^{\ell_{\mathsf{in}}} \times \left[k^{c-1}\right] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ as $\bar{C}_{t_1-1}(w,x) = C(w,(t_1-2)\cdot k^{c-1} + (k^{c-1}-x+1))$.
4. The challenger generates the inner level:

$$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1-1}, \bar{t}_2)$$

5. For each inner block index $j \in [k]$, the challenger constructs the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$.
6. The challenger computes $(\mathsf{dig}, \pi_1, \ldots, \pi_k) \leftarrow \mathsf{FBH.Hash}(\mathsf{hk}, (D'_1, \ldots, D'_k))$.
7. The challenger sets $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and $H = \left((D'_j, \pi_j)_{j\in[k]}, H'\right)$.
8. For each block index $j \in [k]$ with $j < t_1 - 1$, the challenger sets P_{null} to be a trivial null circuit of size s_P and sets $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathsf{null}})$.
9. For each block index $j \in [k]$ with $j \geq t_1 - 1$, the challenger constructs the circuit $P_j = P[C, \mathsf{vk}, j, G'_j]$ from Fig. 6 where G'_j is constructed as follows, and samples $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_j)$:
   - If $j = t_1 - 1$, the challenger constructs the circuit $G_j = G[\mathsf{hk}, \mathsf{dig}, j, \mathsf{ek}']$ from Fig. 5.
   - If $j > t_1 - 1$, the challenger sets G_j to be a trivial null circuit of size s_G.
   - In either case, the challenger obfuscates $G'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G_j)$.
10. The challenger encrypts each block program as $\mathsf{ct}_j \leftarrow \mathsf{SKE.Enc}(\mathsf{ek}, P'_j)$ for all $j \in [k]$.
11. The challenger outputs $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$.

- Hyb_1: This is the same as Hyb_0, except that the reversed inner cutoff is set to 0, so that the inner level is generated at cutoff 0. This transition relies on the index indistinguishability of Π'_{LCiO} and the promise that $C(w,t) = \bot$ for all $w \in \{0,1\}^{\ell_{\mathsf{in}}}$.

  2. The challenger sets the reversed inner cutoff $\bar{t}_2 = 0$.

- Hyb_2: This is the same as Hyb_1, except that the challenger uses a pseudorandom function to derive the randomness for the generation of the inner programs D'_j.

  5. The challenger samples a PRF key $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{\lceil \log k \rceil}, 1^\rho)$ and, for each inner block index $j \in [k]$, constructs the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j; \mathsf{PRF.Eval}(K, j))$.

- Hyb_3: This is the same as Hyb_2, except that the function-binding hash key is generated in binding mode to ensure that the hashed values are the honestly generated inner block programs.

  1. The challenger samples keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, 1^{s_f}, 1^{s_D}, k, f_{K,\mathsf{st}'})$, where the block function $f_{K,\mathsf{st}'}$ is the one from Fig. 7.

- Hyb_4: This is the same as Hyb_3, except that the challenger replaces the functional program for block $t_1 - 1$ with a trivial null program. This transition relies on the security of NiO applied to P'_{t_1-1}.

  8. For each block index $j \in [k]$ with $j < t_1$, the challenger sets P_{null} to be a trivial null circuit of size s_P and sets $P'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathsf{null}})$.
  9. For each block index $j \in [k]$ with $j \geq t_1$, the challenger constructs the circuit $P_j = P[C, \mathsf{vk}, j, G'_j]$ from Fig. 6 where G_j is a trivial null circuit of size s_G, obfuscated as $G'_j \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G_j)$.

- Hyb_5: This is the same as Hyb_4, except that the challenger returns the function-binding hash key to the normal setup mode.

  1. The challenger samples keys $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$.

- Hyb_6: This is the same as Hyb_5, except that the challenger replaces the pseudorandom coins used for generating the inner programs D'_j with true randomness, relying on the pseudorandomness of the PRF.

5. For each $j \in [k]$, the challenger samples the inner block program $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j)$. Specifically, ObfBlockProgram′ uses true randomness.

- Hyb_7: This is the same as Hyb_6, except that the inner level is generated for block t_1 at reversed cutoff k^{c-1} (instead of block $t_1 - 1$ at cutoff 0). This transition relies on the functionality hiding property of Π'_{LCiO}, which is applicable since by Hyb_4 the inner evaluation key ek' has left the adversary's view.

2. The challenger sets the reversed inner cutoff $\bar{t}_2 = k^{c-1}$.
3. The challenger defines the reversed inner circuit $\bar{C}_{t_1} : \{0,1\}^{\ell_{\mathsf{in}}} \times \left[k^{c-1}\right] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ as $\bar{C}_{t_1}(w, x) = C(w, (t_1 - 1) \cdot k^{c-1} + (k^{c-1} - x + 1))$.
4. The challenger generates the inner level:

$$(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1})$$

Note that this also changes the inner block programs $D'_1, \ldots, D'_k$, and hence the digest and the openings.

The distribution in Hyb_7 is exactly the distribution generated by $\mathsf{Setup}(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$. Once we prove Claims 4.29 to 4.35, the lemma follows by a standard hybrid argument over $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_7$.

Claim 4.29. *Assume that Π'_{LCiO} satisfies index indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. The only difference between Hyb_0 and Hyb_1 is the reversed inner cutoff used to generate the inner level: 1 in Hyb_0 and 0 in Hyb_1, both with the same inner circuit $\bar{C}_{t_1-1}$ and inner programs generated using true randomness. This is an index indistinguishability step on Π'_{LCiO}, following the same template as Lemma 4.20. We construct an adversary $\mathcal{B}$ for the index indistinguishability game of Π'_{LCiO} that achieves the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, where $t_2 = 0$ by the assumption on t, so that $t = (t_1 - 1) \cdot k^{c-1}$.
2. Algorithm $\mathcal{B}$ defines the reversed inner circuit $\bar{C}_{t_1-1}(w, x) = C(w, (t_1 - 2) \cdot k^{c-1} + (k^{c-1} - x + 1))$ and sends $(1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1-1}, 1)$ to the index indistinguishability challenger for Π'_{LCiO}, which replies with the inner components $(\mathsf{ek}', H', \mathsf{vk}', \{D'_j\}_{j\in[k]})$, generated at reversed cutoff either 1 or 0.
3. Algorithm $\mathcal{B}$ generates the rest of the top-level components exactly as in Hyb_0:
   - First, it samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$ in normal mode, computes the digest and openings of $(D'_1, \ldots, D'_k)$, and assembles $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and the auxiliary component H.
   - Then, it samples its own evaluation key $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$.
   - Finally, it constructs the top-level block programs: for $j < t_1 - 1$ a null program; for $j = t_1 - 1$ the program $P_{t_1-1} = P[C, \mathsf{vk}, t_1 - 1, G'_{t_1-1}]$ with the real wrapper $G'_{t_1-1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G[\mathsf{hk}, \mathsf{dig}, t_1 - 1, \mathsf{ek}'])$; and for $j > t_1 - 1$ the program $P_j = P[C, \mathsf{vk}, j, G'_j]$ with a null wrapper. It encrypts each obfuscated block program under ek.
4. Algorithm $\mathcal{B}$ outputs $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$, and forwards $\mathcal{A}$'s guess b' to the Π'_{LCiO} challenger.

The two experiments abort under the same condition: the Π'_{LCiO} game toggles between reversed inner cutoffs 1 and 0 (i.e., the inner index 1), and it aborts if $\bar{C}_{t_1-1}(w, 1) \neq \bot$ for some $w \in \{0,1\}^{\ell_{\mathsf{in}}}$. By definition,

$$\bar{C}_{t_1-1}(w, 1) = C\big(w, (t_1 - 2) \cdot k^{c-1} + (k^{c-1} - 1 + 1)\big) = C\big(w, (t_1 - 1) \cdot k^{c-1}\big) = C(w, t),$$

so the Π'_{LCiO} challenger aborts and outputs 0 precisely when the level-c game does. If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. Moreover, if the Π'_{LCiO} challenger uses reversed cutoff 1, then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$, and if it uses reversed cutoff 0, then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Therefore ε is negligible by the index indistinguishability of Π'_{LCiO}. □

Claim 4.30. *Assume that* Π_{PRF} *is a secure pseudorandom function. Then for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.

Proof. This proof is completely analogous to that of Claim 4.22, with the inner level generated from $(\bar{C}_{t_1-1}, 0)$ and with the real wrapper embedded in block $t_1 - 1$. □

Claim 4.31. *Assume that* Π_{FBH} *satisfies mode indistinguishability. Then for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.

Proof. This proof is completely analogous to that of Claim 4.23. □

Claim 4.32. *Assume that* NiO *is a secure null-i*$\mathcal{O}$*, that* Π_{FBH} *satisfies perfect function binding, and that* Π'_{LCiO} *satisfies endpoint soundness. Then for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right|$. The only difference between Hyb_3 and Hyb_4 is the top-level program at block t_1-1: in Hyb_3 it is $P'_{t_1-1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{t_1-1})$ with $P_{t_1-1} = P[C, \mathsf{vk}, t_1-1, G'_{t_1-1}]$ and $G'_{t_1-1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G[\mathsf{hk}, \mathsf{dig}, t_1 - 1, \mathsf{ek}'])$, whereas in Hyb_4 this block is null (i.e., $P'_{t_1-1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, P_{\mathrm{null}})$). We construct an algorithm $\mathcal{B}$ that breaks the security of NiO with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathrm{in}}}, 1^{\ell_{\mathrm{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$; algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, with $t_2 = 0$ by assumption.

2. Algorithm $\mathcal{B}$ samples $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and a PRF key $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{\lceil \log k \rceil}, 1^\rho)$, constructs $\bar{C}_{t_1-1}(w, x) = C(w, (t_1-2)\cdot k^{c-1} + (k^{c-1} - x + 1))$, and generates $(\mathsf{st}', H', \mathsf{vk}', \mathsf{ek}') \leftarrow \mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathrm{in}}}, 1^{\ell_{\mathrm{out}}}, \bar{C}_{t_1-1}, 0)$.

3. Algorithm $\mathcal{B}$ constructs $f_{K,\mathsf{st}'}$ from Fig. 7 and samples $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, 1^{s_f}, 1^{s_D}, k, f_{K,\mathsf{st}'})$ in binding mode. For each $j \in [k]$ it constructs $D'_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}', j\,; \mathsf{PRF.Eval}(K, j))$ and computes the digest, openings, and auxiliary component H.

4. Algorithm $\mathcal{B}$ constructs the real wrapper $G'_{t_1-1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_G}, G[\mathsf{hk}, \mathsf{dig}, t_1 - 1, \mathsf{ek}'])$ and the program $P_{t_1-1} = P[C, \mathsf{vk}, t_1 - 1, G'_{t_1-1}]$. It submits the two challenge circuits $C_0 = P_{t_1-1}$ and $C_1 = P_{\mathrm{null}}$, both of size s_P, to the null-i$\mathcal{O}$ challenger, which replies with $P'_{t_1-1} \leftarrow \mathsf{NiO}(1^\lambda, 1^{s_P}, C_b)$.

5. Algorithm $\mathcal{B}$ assembles the remaining top-level programs as in Hyb_3: for $j < t_1 - 1$ a null program; for $j > t_1 - 1$ a null wrapper G'_j and $P_j = P[C, \mathsf{vk}, j, G'_j]$, obfuscated to P'_j. It encrypts every obfuscated block program under ek and sends $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$.

6. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If the challenger obfuscates $C_0 = P_{t_1-1}$ (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_3(\mathcal{A})$; if it obfuscates $C_1 = P_{\mathrm{null}}$ (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_4(\mathcal{A})$. It remains to verify that the NiO experiment does not abort (i.e., that both challenge circuits are null). For $C_1 = P_{\mathrm{null}}$ this is immediate. For $C_0 = P_{t_1-1} = P[C, \mathsf{vk}, t_1 - 1, G'_{t_1-1}]$, fix an arbitrary input (w, i, h) with $h = ((D', \pi), h')$, and let $(i_1, i_2) = \mathsf{decomp}_{k,c}(i)$, $\bar{i}_2 = k^{c-1} - i_2 + 1$, and $i'_1 = \lceil \bar{i}_2 / k^{c-2} \rceil$. By the description of P in Fig. 6, P_{t_1-1} outputs $C(w, i)$ only if $i_1 = t_1 - 1$, $\mathsf{Verify}(\mathsf{vk}, h, i) = 1$, and $G'_{t_1-1}(w, i, h) = \bot$, and outputs $\bot$ otherwise; so it suffices to consider inputs with $i_1 = t_1 - 1$ and $\mathsf{Verify}(\mathsf{vk}, h, i) = 1$. In particular, the verification step of P guarantees both $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{dig}, i'_1, D', \pi) = 1$ and $\mathsf{Verify}'(\mathsf{vk}', h', \bar{i}_2) = 1$. Since hk is binding for $f_{K,\mathsf{st}'}$ and every honest block satisfies $f_{K,\mathsf{st}'}$, perfect function binding forces $D' = D'_{i'_1}$, the honest inner block program; hence, by the functionality-preserving property of NiO, $G'_{t_1-1}(w, i, h) = \mathsf{Eval}'(\mathsf{ek}', D'_{i'_1}, h', w, \bar{i}_2)$. The inner level is generated at reversed cutoff 0 with circuit $\bar{C}_{t_1-1}$, which is the all-on cutoff for Π'_{LCiO}, the key ek′ is the evaluation key output by that same execution of Setup′, and $\mathsf{Verify}'(\mathsf{vk}', h', \bar{i}_2) = 1$; hence, by the soundness of Π'_{LCiO} at the all-on cutoff,

$$G'_{t_1-1}(w, i, h) = \mathsf{Eval}'(\mathsf{ek}', D'_{i'_1}, h', w, \bar{i}_2) = \bar{C}_{t_1-1}(w, \bar{i}_2) = C(w, (t_1 - 2) \cdot k^{c-1} + i_2) = C(w, i),$$

where the last equalities substitute $\bar{i}_2 = k^{c-1} - i_2 + 1$ and use $i_1 = t_1 - 1$. Consequently, if $C(w,i) \neq \bot$, then $G'_{t_1-1}(w,i,h) \neq \bot$, so the condition $G'_{t_1-1}(w,i,h) = \bot$ fails and P_{t_1-1} outputs $\bot$; and if $C(w,i) = \bot$, then P_{t_1-1} outputs $C(w,i) = \bot$. In every case P_{t_1-1} outputs $\bot$, so C_0 is a null circuit and the experiment does not abort. Thus, algorithm $\mathcal{B}$ breaks security of NiO with the same probability ε. □

Claim 4.33. *Assume that Π_{FBH} satisfies mode indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. This proof is completely analogous to that of Claim 4.23. □

Claim 4.34. *Assume that Π_{PRF} is a secure pseudorandom function. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. This proof is completely analogous to that of Claim 4.22, with the inner level generated from $(\bar{C}_{t_1-1}, 0)$ and with a null wrapper in every block. □

Claim 4.35. *Assume that Π'_{LCiO} satisfies functionality hiding. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]\right|$. The only difference between Hyb_6 and Hyb_7 is the inner level: in Hyb_6 it is generated by $\mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1-1}, 0)$, whereas in Hyb_7 it is generated by $\mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{t_1}, k^{c-1})$; this changes H', vk', the inner block programs $D'_1, \ldots, D'_k$, and hence the digest and openings. In both hybrids every wrapper is null, so the reduction needs neither the inner state st' nor the inner evaluation key ek', and it obtains $(H', \mathsf{vk}', \{D'_j\}_{j\in[k]})$ directly from the functionality hiding challenger for Π'_{LCiO}. We construct an adversary $\mathcal{B}$ for the functionality hiding game of Π'_{LCiO} that achieves the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ starts by sending $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuit C and the cutoff t to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes $(t_1, t_2) = \mathsf{decomp}^*_{k,c}(t)$, where $t_2 = 0$ by the assumption on t.

2. Algorithm $\mathcal{B}$ samples $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{s_f}, 1^{s_D}, k)$ in normal mode.

3. Algorithm $\mathcal{B}$ constructs the two reversed inner circuits

$$\bar{C}_{t_1-1}(w,x) = C(w, (t_1-2)\cdot k^{c-1} + (k^{c-1} - x + 1)),$$
$$\bar{C}_{t_1}(w,x) = C(w, (t_1-1)\cdot k^{c-1} + (k^{c-1} - x + 1)),$$

   and forwards $1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, the circuits $\bar{C}_{t_1-1}, \bar{C}_{t_1}$ and the cutoffs $0, k^{c-1}$ as its challenge to the functionality hiding challenger for Π'_{LCiO}, which responds with the inner auxiliary component, verification key and block programs $(H', \mathsf{vk}', \{D'_j\}_{j\in[k]})$.

4. Algorithm $\mathcal{B}$ computes the digest and openings of $(D'_1, \ldots, D'_k)$, and assembles $\mathsf{vk} = ((\mathsf{hk}, \mathsf{dig}), \mathsf{vk}')$ and the auxiliary component $H = \big((D'_j, \pi_j)_{j\in[k]}, H'\big)$.

5. Algorithm $\mathcal{B}$ assembles the top-level block programs as in Hyb_6: for $j < t_1$ a null program; for $j \geq t_1$ a null wrapper G'_j and $P_j = P[C, \mathsf{vk}, j, G'_j]$, obfuscated to P'_j. It encrypts each obfuscated block program under ek and sends $(\mathsf{ek}, H, \mathsf{vk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ to $\mathcal{A}$.

6. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the functionality hiding challenger generates the inner level from $(\bar{C}_{t_1-1}, 0)$ (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_6(\mathcal{A})$; if it generates it from $(\bar{C}_{t_1}, k^{c-1})$ (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_7(\mathcal{A})$. Thus, $\mathcal{B}$ breaks functionality hiding of Π'_{LCiO} with the same advantage ε. □

By Claims 4.29 to 4.35, the lemma follows by a standard hybrid argument over $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_7$. □

Every cutoff $t \in [N]$ has $\mathrm{decomp}^*_{k,c}(t) = (t_1, t_2)$ with $t_2 \in \{0, \ldots, k^{c-1} - 1\}$, so the three cases $t_2 = 1$, $t_2 = 0$, and $t_2 \in [2, k^{c-1} - 1]$ are exhaustive and mutually exclusive. Lemmas 4.20, 4.21 and 4.28 therefore cover every adversary, which proves the theorem. □

Theorem 4.36 (Circuit Hiding). *If Π'_{LCiO} satisfies perfect circuit hiding then Construction 4.10 satisfies perfect circuit hiding.*

Proof. We show that for every circuit C, the joint distribution of $(\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j\in[k]})$ generated by Setup at cutoff k^c together with ObfBlockProgram is perfectly identical to the one generated by SetupNull together with ObfBlockProgram. The adversary's advantage is then exactly 0.

- First consider the inner level. In Setup, the decomposition of $t = k^c$ yields $(t_1, t_2) = (k + 1, 0)$, so the reversed inner cutoff is $\bar{t}_2 = k^{c-1} - 0 = k^{c-1}$, which is exactly the all-off cutoff for Π'_{LCiO}. The inner call is therefore $\mathsf{Setup}'(1^\lambda, 1^s, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \bar{C}_{k+1}, k^{c-1})$, where $\bar{C}_{k+1}$ is the reversed inner circuit of block $t_1 = k + 1$, whereas in SetupNull it is $\mathsf{SetupNull}'(1^\lambda, 1^{s'}, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$. By the perfect circuit hiding of Π'_{LCiO} (which holds for every inner circuit, so the particular choice of $\bar{C}_{k+1}$ is immaterial), the resulting tuples $(\mathsf{ek}', H', \mathsf{vk}', \{D'_j\}_{j\in[k]})$ are identically distributed in the two branches.
- Now consider the level-c components. The evaluation key ek is sampled as $\mathsf{ek} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ in both branches, independently of everything else, so it is identically distributed. The auxiliary component H and verification key vk are obtained from $(H', \mathsf{vk}', \{D'_j\})$ by hashing the inner block programs and pairing the result with H' and vk' respectively; since $(H', \mathsf{vk}', \{D'_j\})$ is identically distributed in both branches and $(\mathsf{ek}, \mathsf{hk})$ are sampled identically, so are H and vk. For the block programs, both executions store $t_1 = k + 1$ in the state, so for every $j \in [k]$ the condition $j < t_1$ holds and ObfBlockProgram encrypts, under the same key ek, an NiO obfuscation of the same trivial null circuit, independently of the stored circuit. Hence the two joint distributions are identically distributed. □

4.4 Putting It Together

We now combine the pieces in two steps. We first show that every property of Definition 4.1 *other than succinctness* transfers directly to the original notion of Definition 3.1: a c-leveled cutoff-$i\mathcal{O}$ yields a plain cutoff-$i\mathcal{O}$ by simply publishing the evaluation key, the auxiliary component and the verification key together with all k block programs, and correctness, index indistinguishability, and perfect circuit hiding then carry over verbatim. We deliberately leave succinctness out of that step, since the size of the resulting program is determined by the sizes of the leveled scheme's components; we therefore state that size explicitly instead. We then show that a c-leveled cutoff-$i\mathcal{O}$ exists for every constant level c, again with explicit bounds on the sizes of its components. Combining the two yields the $N^{1/c}$-succinct cutoff-$i\mathcal{O}$ we are after.

Proposition 4.37 (From Leveled to Plain Cutoff-$i\mathcal{O}$). *Let $c \geq 1$ and let Π'_{LCiO} be a c-leveled cutoff-$i\mathcal{O}$ (Definition 4.1) satisfying correctness, index indistinguishability, and perfect circuit hiding. Then there is a cutoff-$i\mathcal{O}$ scheme $\Pi_{\mathsf{CiO}} = (\mathsf{ObfNull}, \mathsf{Obf}, \mathsf{Eval})$ (Definition 3.1) for index space $N = k^c$ satisfying correctness, index indistinguishability, and perfect circuit hiding. Moreover, every program it outputs is of the form $P = (\mathsf{ek}, H, \mathsf{vk}, P_1, \ldots, P_k)$, where* ek, H, vk *and $P_1, \ldots, P_k$ are respectively the evaluation key, the auxiliary component, the verification key, and the block programs of Π'_{LCiO}. In particular,*

$$|P| \leq |\mathsf{ek}| + |H| + |\mathsf{vk}| + k \cdot \max_{j\in[k]} |P_j|.$$

Proof. Denote the algorithms of Π'_{LCiO} by $(\mathsf{SetupNull}', \mathsf{Setup}', \mathsf{ObfBlockProgram}', \mathsf{Preprocess}', \mathsf{Verify}', \mathsf{Eval}')$, and let $k = N^{1/c}$. We construct a cutoff-$i\mathcal{O}$ scheme $\Pi_{\mathsf{CiO}} = (\mathsf{ObfNull}, \mathsf{Obf}, \mathsf{Eval})$ as follows:

- $\mathsf{Obf}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$: On input the security parameter λ, a size bound s, the size of the index space N, the input length ℓ_{in}, the output length ℓ_{out}, a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ such that $|C| \leq s$, and a cutoff index $t \in \{0, \ldots, N\}$, the obfuscation algorithm does the following:
  1. Sample $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{Setup}'(1^\lambda, 1^{s'}, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$.

2. Compute $P_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}, j)$ for every $j \in [k]$.
3. Output $P = (\mathsf{ek}, H, \mathsf{vk}, P_1, \ldots, P_k)$.

- $\mathsf{ObfNull}(1^\lambda, 1^s, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$: On input the security parameter λ, a size bound s, the size of the index space N, the input length ℓ_{in}, and the output length ℓ_{out}, the null-obfuscation algorithm does the following:
  1. Sample $(\mathsf{st}, H, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{SetupNull}'(1^\lambda, 1^{s'}, 1^k, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  2. Compute $P_j \leftarrow \mathsf{ObfBlockProgram}'(\mathsf{st}, j)$ for every $j \in [k]$.
  3. Output $P = (\mathsf{ek}, H, \mathsf{vk}, P_1, \ldots, P_k)$.
- $\mathsf{Eval}(P, w, i)$: On input a program $P = (\mathsf{ek}, H, \mathsf{vk}, P_1, \ldots, P_k)$, a string $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, and an index $i \in [N]$, the evaluation algorithm computes the evaluation hint $h = \mathsf{Preprocess}'(H, w, i)$ and outputs $\mathsf{Eval}'\big(\mathsf{ek}, P_{\lceil i/k^{c-1} \rceil}, h, w, i\big)$.

The claimed form of P, and hence the displayed size bound, are immediate from the description of Obf. Note that ObfNull does not depend on C, as Definition 3.1 requires, since $\mathsf{SetupNull}'$ does not.

Correctness. Fix C, a cutoff t, an input w, and an index $i \in [N]$, and let $P = (\mathsf{ek}, H, \mathsf{vk}, P_1, \ldots, P_k)$ be in the support of Obf on (C, t). Write $j = \lceil i/k^{c-1} \rceil$. The block program selected by Eval is P_j, which by construction lies in the support of $\mathsf{ObfBlockProgram}'(\mathsf{st}, j)$, the evaluation hint it is given is $\mathsf{Preprocess}'(H, w, i)$, and the evaluation key it is given is the one output by the same execution of Setup'. These are exactly the quantifiers of the correctness property of Definition 4.1, which therefore gives

$$\mathsf{Eval}(P, w, i) = \mathsf{Eval}'\big(\mathsf{ek}, P_j, \mathsf{Preprocess}'(H, w, i), w, i\big) = \begin{cases} C(w, i) & \text{if } i > t, \\ \bot & \text{otherwise,} \end{cases}$$

which is the correctness requirement of Definition 3.1.

Index indistinguishability and perfect circuit hiding. Both games of Definition 4.1 hand the adversary exactly the tuple $(\mathsf{ek}, H, \mathsf{vk}, \{P_j\}_{j \in [k]})$, which by the description of Obf and ObfNull above is precisely the program P that the corresponding games of Definition 3.1 hand the adversary. The two pairs of games are moreover identical in every other respect: the index indistinguishability games agree on the abort condition and on sampling the challenge at cutoff $t-b$, and the circuit hiding games agree that $b = 0$ samples at cutoff N while $b = 1$ calls the null algorithm. Hence for each property the two experiments are the same, and any adversary against Π_{CiO} is an adversary against Π'_{LCiO} with the same advantage. The two properties therefore transfer from Π'_{LCiO}, which satisfies them by assumption. □

It remains to build a leveled scheme of the desired level.

Theorem 4.38 (Existence of Leveled Cutoff-$i\mathcal{O}$)**.** *Let $c \geq 1$ be a constant. Assume the existence of a null indistinguishability obfuscator* NiO *with polynomial size overhead, a CPA-secure symmetric-key encryption scheme Π_{SKE}, a function-binding hash Π_{FBH} for conjunctions of block functions satisfying mode indistinguishability and perfect function binding, and a secure pseudorandom function Π_{PRF}. Then for every k there exists a c-leveled cutoff-$i\mathcal{O}$ for index space $N = k^c$ satisfying correctness, hint validity, endpoint soundness, functionality hiding, index indistinguishability, and perfect circuit hiding. Moreover, there are a polynomial* poly *and a constant $d \geq 1$, both independent of c, such that the verification key, every evaluation hint, and every block program of this scheme have size at most $\mathsf{poly}(\lambda, s, \log k)^{d^c}$, and its auxiliary component has size at most $k \cdot \mathsf{poly}(\lambda, s, \log k)^{d^c}$. The evaluation key is exempt from this recursion: it has size at most $\mathsf{poly}_{\mathsf{ek}}(\lambda)$ for a polynomial $\mathsf{poly}_{\mathsf{ek}}$ that is independent of c, s and k.*

Proof. Define a family of schemes $\Pi^{(1)}_{\mathsf{LCiO}}, \ldots, \Pi^{(c)}_{\mathsf{LCiO}}$ as follows: $\Pi^{(1)}_{\mathsf{LCiO}}$ is Construction 4.2, and for $2 \leq r \leq c$, the scheme $\Pi^{(r)}_{\mathsf{LCiO}}$ is the result of instantiating Construction 4.10 at level r with $\Pi^{(r-1)}_{\mathsf{LCiO}}$ as its underlying $(r-1)$-leveled scheme. Each level instantiates NiO, Π_{SKE}, Π_{FBH} and Π_{PRF} with a fresh set of parameters, which are polynomially bounded by Theorem 4.14.

We claim that for every $r \in [c]$, the scheme $\Pi^{(r)}_{\mathsf{LCiO}}$ is an r-leveled cutoff-$i\mathcal{O}$ satisfying correctness, hint validity, endpoint soundness, succinctness, index indistinguishability, functionality hiding, and perfect circuit hiding. We prove the claim by induction on r.

1. **Base case ($r = 1$).** The scheme $\Pi^{(1)}_{\mathsf{LCiO}}$ is Construction 4.2, and the seven properties are established by Theorems 4.3 to 4.9, respectively.

2. **Inductive step ($2 \le r \le c$).** Suppose the claim holds for $\Pi^{(r-1)}_{\mathsf{LCiO}}$. Every composition result of this section treats its underlying scheme as a black box, so each of them applies with $\Pi'_{\mathsf{LCiO}} = \Pi^{(r-1)}_{\mathsf{LCiO}}$, and the induction hypothesis supplies exactly the hypotheses they require. We invoke them in the following order, since correctness at level r appeals to hint validity at level r: hint validity by Theorem 4.12, correctness by Theorem 4.11, endpoint soundness by Theorem 4.13, succinctness by Theorem 4.14, functionality hiding by Theorem 4.15, index indistinguishability by Theorem 4.19, and perfect circuit hiding by Theorem 4.36.

In particular $\Pi^{(c)}_{\mathsf{LCiO}}$ is a c-leveled cutoff-$i\mathcal{O}$ with all seven properties.

It remains to make the size bounds explicit. For each r, let poly_r be the bound on the parameter sizes associated with $\Pi^{(r)}_{\mathsf{LCiO}}$, and let q be the polynomial from Theorem 4.14 of total degree d. Both q and the polynomial of Theorem 4.6 are the same at every level, so we may take a single polynomial poly, independent of c, that dominates them both; then $\mathsf{poly}_1 \le \mathsf{poly}$ and $\mathsf{poly}_r \le \left(\mathsf{poly} \cdot \mathsf{poly}_{r-1}\right)^d$ for every $2 \le r \le c$. Unrolling this recursion gives $\mathsf{poly}_c \le \mathsf{poly}^{d^c}$, after absorbing the resulting constant factors into d. By the succinctness of $\Pi^{(c)}_{\mathsf{LCiO}}$, this is exactly the claimed bound on the verification key, the evaluation hints, and the block programs, and k times it on the auxiliary component.

The evaluation key does not enter this recursion: at every level r, the evaluation key of $\Pi^{(r)}_{\mathsf{LCiO}}$ is a freshly sampled Π_{SKE} key, so by Theorems 4.6 and 4.14 its size is bounded by the same polynomial $\mathsf{poly}_{\mathsf{ek}}(\lambda)$ at every level. □

Corollary 4.39 ($N^{1/c}$-Succinct Cutoff-$i\mathcal{O}$)**.** *Let $c \ge 1$ be a constant, and assume the existence of witness encryption and the hardness of the learning with errors assumption with sub-exponential approximation factors. Then there exists an $N^{1/c}$-succinct cutoff-$i\mathcal{O}$ scheme (Definition 3.1).*

Proof. First, the ingredients can be instantiated under the assumptions:

- Null-$i\mathcal{O}$ can be constructed from witness encryption and the hardness of LWE with sub-exponential approximation factors [GKW17, WZ17].
- A function-binding hash can be constructed from LWE [FWW23].
- Pseudorandom functions and secret-key encryption can also be constructed from LWE.

From this point, the corollary follows immediately by applying Proposition 4.37 to the scheme of Theorem 4.38 with $k = N^{1/c}$, using $\log k = \frac{1}{c} \log N$ and absorbing the constant factors into d. The obfuscated program additionally publishes the evaluation key, which by Theorem 4.38 contributes only $\mathsf{poly}_{\mathsf{ek}}(\lambda)$ to its size. □

5 Indexed Positional Witness Encryption

In this section we show how to obtain indexed positional witness encryption from cutoff-$i\mathcal{O}$. An indexed positional witness encryption scheme encrypts a message μ to a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$ together with a cutoff $t \in [0, N]$, and a witness (w, i) recovers μ exactly when $C(w, i) = 1$ and $i > t$. The scheme is therefore an ordinary witness encryption for C at $t = 0$, and hides the message entirely at $t = N$. Security requires that the cutoffs t and $t - 1$ be indistinguishable whenever C has no witness at index t. These are analogous to the guarantees of Definition 3.1, so the two notions map onto each other directly. We give the definition in Definition 5.1, and then obtain it in Construction 5.2 by obfuscating the circuit that outputs μ when $C(w, i) = 1$ and $\perp$ otherwise.

Definition 5.1 (Indexed Positional Witness Encryption [GLW14, adapted])**.** An indexed positional witness encryption scheme for circuit satisfiability consists of two efficient algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following syntax. Throughout, we write m for the witness length and N for the size of the index space.

- $\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu) \to \mathsf{ct}$: On input a security parameter λ, an index space size N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, a cutoff $t \in [0, N]$, and a message $\mu \in \{0,1\}^*$, the encryption algorithm outputs a ciphertext ct.

- $\mathsf{Dec}(\mathsf{ct}, w, i) \to y$: On input a ciphertext ct and a witness $(w, i) \in \{0,1\}^m \times [N]$, the decryption algorithm outputs a value $y \in \{0,1\}^* \cup \{\bot\}$. This algorithm is deterministic.

The positional witness encryption scheme should satisfy the following properties:

- **Correctness:** For all $\lambda, m, N \in \mathbb{N}$, all $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, all $t \in [0, N]$, all $(w, i) \in \{0,1\}^m \times [N]$ such that $C(w, i) = 1 \wedge i > t$, all messages $\mu \in \{0,1\}^*$, and all ciphertexts ct in the support of $\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu)$, we have $\mathsf{Dec}(\mathsf{ct}, w, i) = \mu$.

- **Succinctness:** We say that the positional witness encryption scheme is $\eta(N)$-succinct if for all $\lambda, m, N \in \mathbb{N}$, all circuits $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, all cutoffs $t \in [0, N]$, all messages $\mu \in \{0,1\}^*$, and any ciphertext ct in the support of $\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu)$, the following holds:

$$|\mathsf{ct}| \leq \eta(N) \cdot \mathsf{poly}(\lambda, |C|, |\mu|, \log N).$$

- **Message hiding:** For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0,1\}$:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends 1^N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, and a pair of messages $\mu_0, \mu_1 \in \{0,1\}^*$ of the same length to the challenger.
  2. The challenger computes a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, 1^N, C, N, \mu_b)$ and sends ct to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the positional witness encryption scheme satisfies message hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Index indistinguishability:** For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0,1\}$:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends 1^N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, a cutoff index $t \in [N]$ and a message $\mu \in \{0,1\}^*$ to the challenger.
  2. If there exists $w \in \{0,1\}^m$ such that $C(w, t) = 1$, the experiment aborts and outputs 0.
  3. The challenger computes a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, 1^N, C, t - b, \mu)$ and sends ct to $\mathcal{A}$.
  4. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the positional witness encryption scheme satisfies index indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

5.1 Constructing Indexed Positional Witness Encryption From Cutoff-$i\mathcal{O}$

Construction 5.2 (Indexed Positional Witness Encryption)**.** Let $\Pi_{\mathsf{CiO}} = (\mathsf{ObfNull}, \mathsf{Obf}, \mathsf{Eval})$ be a cutoff-$i\mathcal{O}$ scheme as per Definition 3.1. We construct an indexed positional witness encryption scheme $\Pi_{\mathsf{PWE}} = (\mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu)$: On input a security parameter λ, an index space size N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, a cutoff $t \in [0, N]$, and a message $\mu \in \{0,1\}^*$, the encryption algorithm does the following:

  1. Set the cutoff-$i\mathcal{O}$ output length $\ell_{\mathsf{out}} = |\mu|$.

2. Construct the circuit $D = D[C, \mu]$ described in Fig. 8 below, and let $s = |D|$:

> **Constants:** A circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$ and a message $\mu \in \{0,1\}^*$.
> **Inputs:** A witness $w \in \{0,1\}^m$ and an index $i \in [N]$.
>
> 1. If $C(w, i) = 1$, output μ.
> 2. Otherwise, output $\perp$.

Figure 8: Circuit $D[C, \mu]$.

3. Compute $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\text{out}}}, D, t)$.
4. Output the ciphertext $\mathsf{ct} = P$.

- $\mathsf{Dec}(\mathsf{ct}, w, i)$: On input a ciphertext ct and a witness $(w, i) \in \{0,1\}^m \times [N]$, the decryption algorithm outputs $\mathsf{CiO.Eval}(\mathsf{ct}, w, i)$.

Theorem 5.3 (Correctness). *If Π_{CiO} is correct then Construction 5.2 is correct.*

Proof. Let $\lambda, m, N \in \mathbb{N}$, let $C\colon \{0,1\}^m \times [N] \to \{0,1\}$ be a circuit, $t \in [0, N]$ a cutoff, $\mu \in \{0,1\}^*$ a message, and let $(w, i) \in \{0,1\}^m \times [N]$ be a witness such that $C(w, i) = 1$ and $i > t$. Let ct be a ciphertext in the support of $\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu)$. Then the following hold:

- By construction, ct is an obfuscated program generated by $\mathsf{CiO.Obf}$ for the circuit $D = D[C, \mu]$ with cutoff t, and the decryption algorithm $\mathsf{Dec}(\mathsf{ct}, w, i)$ outputs $\mathsf{CiO.Eval}(\mathsf{ct}, w, i)$.
- By the correctness of Π_{CiO}, the obfuscated program evaluates to exactly $D(w, i)$ whenever $i > t$. Since we are guaranteed by the correctness condition that $i > t$, the evaluation algorithm outputs exactly $D(w, i)$.
- We now analyze the execution of $D[C, \mu]$ on input (w, i). It evaluates $C(w, i)$, and since $C(w, i) = 1$, the circuit hits the accepting branch and outputs the message μ. Thus, $\mathsf{Dec}(\mathsf{ct}, w, i) = \mu$, satisfying correctness. □

Theorem 5.4 (Succinctness). *If Π_{CiO} is $\eta(N)$-succinct then Construction 5.2 is $\eta(N)$-succinct.*

Proof. Let $\lambda, m, N \in \mathbb{N}$, let $C\colon \{0,1\}^m \times [N] \to \{0,1\}$ be a circuit, $t \in [0, N]$ a cutoff, and $\mu \in \{0,1\}^*$ a message. Let ct be a ciphertext generated by $\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu)$. By construction, ct is output by $\mathsf{CiO.Obf}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\text{out}}}, D, t)$, where $D = D[C, \mu]$ and $s = |D|$. By the $\eta(N)$-succinctness property of the underlying Π_{CiO} scheme, there exists a polynomial $\mathsf{poly}_{\mathsf{CiO}}$ such that the size of the generated program is bounded by:

$$|\mathsf{ct}| \le \eta(N) \cdot \mathsf{poly}_{\mathsf{CiO}}(\lambda, s, \log N).$$

It remains to bound s: the circuit $D[C, \mu]$ evaluates C and outputs μ or $\perp$, hence $s = |D| \le |C| + \mathsf{poly}(|\mu|, \log N)$. Substituting this bound back into the cutoff-$i\mathcal{O}$ succinctness guarantee and using that the composition of polynomials is a polynomial, there exists a single fixed polynomial $\mathsf{poly}_{\mathsf{PWE}}$ such that:

$$|\mathsf{ct}| \le \eta(N) \cdot \mathsf{poly}_{\mathsf{PWE}}(\lambda, |C|, |\mu|, \log N).$$ □

Theorem 5.5 (Message Hiding). *Assume that Π_{CiO} satisfies circuit hiding. Then Construction 5.2 satisfies message hiding.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the message hiding game of the positional witness encryption scheme. We describe the following hybrids:

- Real_b: This corresponds to the message hiding game with bit b.

1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends an index space size 1^N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, and a pair of messages $\mu_0, \mu_1 \in \{0,1\}^{\ell_{\text{out}}}$ to the challenger.

2. The challenger constructs the circuit $D = D[C, \mu_b]$ from Fig. 8, with $s = |D|$. It then samples the program $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}}, D, N)$ and sends P to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

- Hyb: This is the same as Real_b, except that the program P is sampled using ObfNull. In particular, the parameter b is never used.

1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends an index space size 1^N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, and a pair of messages $\mu_0, \mu_1 \in \{0,1\}^{\ell_{\mathsf{out}}}$ to the challenger.
2. The challenger samples the program $P \leftarrow \mathsf{CiO.ObfNull}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}})$ and sends P to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

We prove that $|\Pr[\mathsf{Real}_b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}(\mathcal{A}) = 1]| \le \mathsf{negl}(\lambda)$ for all $b \in \{0,1\}$, and thus message hiding follows. Fix $b \in \{0,1\}$ and suppose $|\Pr[\mathsf{Real}_b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}(\mathcal{A}) = 1]| \ge \varepsilon$ for some non-negligible ε. We construct an algorithm $\mathcal{B}$ that breaks the circuit hiding property of the underlying Π_{CiO} scheme. Denote the circuit hiding challenge bit by b_{CiO}. Algorithm $\mathcal{B}$ interacts with the cutoff-$i\mathcal{O}$ challenger and simulates the positional witness encryption game for $\mathcal{A}$ as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and gets an index space size 1^N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, and a pair of messages $\mu_0, \mu_1 \in \{0,1\}^{\ell_{\mathsf{out}}}$.
2. Algorithm $\mathcal{B}$ constructs the circuit $D = D[C, \mu_b]$ from Fig. 8, with $s = |D|$, and submits $(1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}}, D)$ to the cutoff-$i\mathcal{O}$ circuit hiding challenger.
3. The cutoff-$i\mathcal{O}$ challenger samples a program P as follows:
   - If $b_{\mathsf{CiO}} = 0$, the challenger outputs $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}}, D, N)$.
   - If $b_{\mathsf{CiO}} = 1$, the challenger outputs $P \leftarrow \mathsf{CiO.ObfNull}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}})$.

   The challenger sends P to $\mathcal{B}$.
4. Algorithm $\mathcal{B}$ forwards P to $\mathcal{A}$, receives a guess b' and outputs it as its guess.

Note that if $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If $b_{\mathsf{CiO}} = 0$, then $\mathcal{B}$ perfectly simulates $\mathsf{Real}_b(\mathcal{A})$. If $b_{\mathsf{CiO}} = 1$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}(\mathcal{A})$. Therefore we conclude that the advantage of $\mathcal{B}$ in the circuit hiding game is exactly

$$|\Pr[\mathsf{Real}_b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}(\mathcal{A}) = 1]| = \varepsilon,$$

which is non-negligible. □

Theorem 5.6 (Index Indistinguishability)**.** *Assume that Π_{CiO} satisfies index indistinguishability. Then Construction 5.2 satisfies index indistinguishability.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the index indistinguishability game of the positional witness encryption scheme. We construct an algorithm $\mathcal{B}$ that breaks the index indistinguishability of the underlying Π_{CiO} scheme. Let b be the challenge bit of the underlying index indistinguishability game of Π_{CiO}. Algorithm $\mathcal{B}$ interacts with the cutoff-$i\mathcal{O}$ challenger and simulates the positional witness encryption game for $\mathcal{A}$ as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and gets an index space size 1^N, a circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$, a cutoff $t \in [N]$, and a message $\mu \in \{0,1\}^{\ell_{\mathsf{out}}}$.
2. Algorithm $\mathcal{B}$ constructs the circuit $D = D[C, \mu]$ from Fig. 8, with $s = |D|$, and submits $(1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}}, D, t)$ to the cutoff-$i\mathcal{O}$ index indistinguishability challenger.
3. The cutoff-$i\mathcal{O}$ challenger samples a program $P \xleftarrow{R} \mathsf{CiO.Obf}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}}, D, t - b)$, and sends P to $\mathcal{B}$.
4. Algorithm $\mathcal{B}$ forwards P to $\mathcal{A}$, receives a guess b' and outputs b' as its guess.

The two games abort under exactly the same condition. Indeed, $D(w,t) = \mu$ if $C(w,t) = 1$ and $D(w,t) = \bot$ otherwise, so there exists $w \in \{0,1\}^m$ with $D(w,t) \neq \bot$ if and only if there exists $w \in \{0,1\}^m$ with $C(w,t) = 1$. The cutoff-$i\mathcal{O}$ challenger therefore aborts and outputs 0 precisely when the positional witness encryption game does, and $\mathcal{B}$ need not decide the condition itself. Next, if $\mathcal{A}$ is efficient then so is $\mathcal{B}$.

Finally, we verify that $\mathcal{B}$ perfectly simulates the positional witness encryption index indistinguishability game. For each $b \in \{0,1\}$, the Π_{CiO} challenger samples $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^s, 1^N, 1^m, 1^{\ell_{\mathsf{out}}}, D, t-b)$, and by construction, $\mathsf{Enc}(1^\lambda, 1^N, C, t-b, \mu)$ obfuscates exactly the circuit D with cutoff $t-b$. Thus for each $b \in \{0,1\}$, the program P is distributed exactly as the challenge ciphertext in the positional witness encryption index indistinguishability game with challenge bit b. Therefore, $\mathcal{B}$ perfectly simulates the positional witness encryption game with the same challenge bit b. Thus, if $\mathcal{A}$ breaks index indistinguishability with advantage ε, then $\mathcal{B}$ breaks index indistinguishability of Π_{CiO} with the same advantage ε. □

6 Somewhere-Statistically Correlation-Intractable Hash Function

In this section we use cutoff-$i\mathcal{O}$ and the learning with errors assumption to build a somewhere-statistically correlation-intractable hash function for efficiently enumerable relations. In turn, we can use a somewhere-statistically correlation-intractable hash function in conjunction with the work of [CJJ21] to obtain a somewhere *statistically* sound batch argument for NP (BARG). For completeness, we show this implication in Appendix A.

Our construction of somewhere-statistically correlation-intractable hash functions follows the approach of Holmgren and Waters [HW26] of boosting a somewhere-statistically correlation-intractable hash function for search relations to one for enumerable relations using obfuscation. While Holmgren and Waters [HW26] use (full-fledged) $i\mathcal{O}$ for their transformation, we show that cutoff-$i\mathcal{O}$ suffices. We start by defining somewhere-statistically correlation-intractable hash functions [CGH98, CCH+19] for a family of relations.

Definition 6.1 (Somewhere-Statistically Correlation-Intractable Hash)**.** Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of relations, where each $R \in \mathcal{R}_\lambda$ is a relation over $\{0,1\}^{\ell_{\mathsf{in}}} \times \{0,1\}^{\ell_{\mathsf{out}}}$ specified by a *description* $\langle R\rangle$. A somewhere-statistically correlation-intractable hash for $\mathcal{R}$ consists of three efficient algorithms $\Pi_{\mathsf{CI}} = (\mathsf{Gen}, \mathsf{GenAvoid}, \mathsf{Hash})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to \mathsf{hk}$: On input a security parameter λ, a description size bound s, an input length ℓ_{in}, and an output length ℓ_{out}, the generation algorithm outputs a hash key hk.
- $\mathsf{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, d) \to \mathsf{hk}$: On input a security parameter λ, a description size bound s, an input length ℓ_{in}, an output length ℓ_{out}, and a description d of size at most s, the avoiding generation algorithm outputs a hash key hk.
- $\mathsf{Hash}(\mathsf{hk}, x) \to y$: On input a hash key hk and a string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$.

The Π_{CI} scheme should satisfy the following two properties:

- **Statistical correlation intractability:** For any (potentially) unbounded adversary $\mathcal{A}$, we define the following experiment:
  1. On input a security parameter, the adversary $\mathcal{A}$ sends $s, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$ as well as a description $\langle R\rangle$ of size at most s, where $R \in \mathcal{R}_\lambda$.
  2. The challenger samples an avoiding hash key $\mathsf{hk} \leftarrow \mathsf{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \langle R\rangle)$ and sends hk to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

  We say that Π_{CI} is statistically correlation intractable if for any $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, in the experiment above, we have $\Pr[(x, \mathsf{Hash}(\mathsf{hk}, x)) \in R] = \mathsf{negl}(\lambda)$. We say that Π_{CI} is perfectly correlation intractable if the probability is 0.

- **Mode indistinguishability:** For any efficient adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0,1\}$:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, and the description $\langle R \rangle$ of a relation $R \in \mathcal{R}_\lambda$ to the challenger.
  2. The challenger computes a hash key hk as follows:
     - If $b = 0$, the challenger computes $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
     - If $b = 1$, the challenger computes $\mathsf{hk} \leftarrow \mathsf{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \langle R \rangle)$.

     The challenger sends hk to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{CI} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

We instantiate this notion for the relation families used in this work, calling out the description in each case.

Definition 6.2 (Search Relations [CCH+19])**.** For any function $f\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$, the associated *search relation* is $R_f = \left\{(x, f(x)) : x \in \{0,1\}^{\ell_{\mathsf{in}}}\right\}$, whose description $\langle R_f \rangle$ is the circuit computing f. The family of *search relations*, which we denote by $\mathcal{F}$, is the set of all search relations. We say that a hash function is somewhere-statistically correlation-intractable for *search relations* if it is somewhere-statistically correlation-intractable (as per Definition 6.1) for $\mathcal{F}$. Note that for a search relation, the condition $(x, \mathsf{Hash}(\mathsf{hk}, x)) \notin R_f$ is equivalent to $\mathsf{Hash}(\mathsf{hk}, x) \neq f(x)$.

We only use the learning with errors assumption to invoke the following result:

Fact 6.3 (Correlation-Intractable Hash Functions from LWE [CCH+19, PS19])**.** Assume the polynomial hardness of the learning with errors assumption. Then there exists a somewhere-statistically correlation-intractable hash function with perfect correlation intractability for search relations.

Definition 6.4 (Enumerable Relations [HW26])**.** Let $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ be a circuit that on input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, outputs a list of N elements $(y_1, \ldots, y_N) \in (\{0,1\}^{\ell_{\mathsf{out}}})^N$. The associated *$N$-enumerable relation* is defined as $R_E = \{(x, y) : y \in E(x)\}$, and the description of R_E is the circuit E. The family of *(s, N)-enumerable relations*, which we denote by $\mathcal{E}$, consists of all N-enumerable relations with descriptions of size at most s. We say that a hash function is somewhere-statistically correlation-intractable for *(s, N)-enumerable relations* if it is somewhere-statistically correlation-intractable (as per Definition 6.1) for $\mathcal{E}$. Note that by definition of R_E, the correlation intractability condition $(x, \mathsf{Hash}(\mathsf{hk}, x)) \notin R_E$ is equivalent to $\mathsf{Hash}(\mathsf{hk}, x) \notin E(x)$.

6.1 Boosting Somewhere-Statistical Correlation Intractability Using Cutoff-$i\mathcal{O}$

Our construction of a somewhere-statistically correlation-intractable hash for enumerable relations follows the outline of Holmgren and Waters [HW26]. The ingredients are a somewhere-statistically correlation-intractable hash for search relations $\Pi'_{\mathsf{CI}} = (\mathsf{Gen}', \mathsf{GenAvoid}', \mathsf{Hash}')$, and a cutoff-$i\mathcal{O}$ $(\mathsf{Obf}, \mathsf{ObfNull}, \mathsf{Eval})$:

- The hash key hk consists of a hash key hk' and an obfuscated program $P\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$.
- To hash a string x, we first compute $y_i = P(x, i)$ for each $i \in [N]$. If there exists $y_i \neq \bot$, then P overrides the hash key, and the output is y_i. Otherwise, the output is $\mathsf{Hash}'(\mathsf{hk}', x)$.

In normal mode, we sample the program P as $P \leftarrow \mathsf{ObfNull}$. In the avoiding mode on an enumerable relation R_E associated with $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$, we define a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \times [N] \to \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$ that on input (x, i) first computes $y' = \mathsf{Hash}'(\mathsf{hk}', x)$ and $(y_1, \ldots, y_N) = E(x)$. Then, if $y' \neq y_i$, it outputs $\bot$, and otherwise, it "vetoes" the value y' and outputs some fixed string outside of the entire enumeration. The program P is sampled as $\mathsf{Obf}(C, 0)$. This is a fully functional program that implements C. Note that perfect correlation intractability follows

immediately: if $\mathsf{Hash}'(\mathsf{hk}', x)$ is not in the enumeration, then the output is $z = \mathsf{Hash}((\mathsf{hk}', P), x) = \mathsf{Hash}'(\mathsf{hk}', x)$, which by definition of the enumerable relation satisfies $(x, z) \notin R_E$. If $\mathsf{Hash}'(\mathsf{hk}', x)$ is in the enumeration (i.e., is equal to some y_i in the vector $E(x)$), then we have $P(x, i) = z \neq \bot$, which causes the output of $\mathsf{Hash}((\mathsf{hk}', P), x)$ to be z. By the choice of z, we have that z is not in the enumeration and again $(x, z) \notin R_E$.

Security sketch. We prove that the program we described is indistinguishable from a null program using a hybrid argument:

1. First, we sample $P \leftarrow \mathsf{Obf}(C, N)$ meaning that P is a trivial null program. This is indistinguishable from normal mode by circuit hiding.
2. To get from $\mathsf{Obf}(C, t)$ to $\mathsf{Obf}(C, t-1)$, we first switch hk' to statistically avoid the function $E_t(x) := E(x, t)$ that outputs the t^{th} value in the enumeration. This follows by mode indistinguishability of Π'_{CI}.
3. Next, we argue that $C(x, t) = \bot$ for all $x \in \{0,1\}^{\ell_{\mathsf{in}}}$: indeed, by the statistical correlation intractability of Π'_{CI}, the hash key hk' avoids $E_t(x)$. This allows us to invoke index indistinguishability of the cutoff-$i\mathcal{O}$ in order to switch $P \leftarrow \mathsf{Obf}(C, t-1)$.

We give the formal construction below.

Construction 6.5 (Somewhere-Statistically Correlation-Intractable Hash for Enumerable Relations)**.** Our construction relies on the following primitives:

- A base somewhere-statistically correlation-intractable hash $\Pi'_{\mathsf{CI}} = (\mathsf{Gen}', \mathsf{GenAvoid}', \mathsf{Hash}')$ for search relations. Such a hash function with *perfect correlation intractability* exists assuming LWE [PS19].
- A cutoff indistinguishability obfuscation scheme $\Pi_{\mathsf{CiO}} = (\mathsf{ObfNull}, \mathsf{Obf}, \mathsf{Eval})$.

We will construct a somewhere-statistically correlation-intractable hash for all (s, N)-enumerable relations $\mathcal{R} \subseteq \{0,1\}^{\ell_{\mathsf{in}}} \times \{0,1\}^{\ell_{\mathsf{out}}}$ where $N \leq 2^{\ell_{\mathsf{out}}} - 1$. For any enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ of size at most s, let s_{CiO} be an upper bound on the size of the circuit described in Fig. 9 and let s' be an upper bound on the size of the circuit described in Fig. 10. We construct the boosted hash scheme $\Pi_{\mathsf{CI}} = (\mathsf{Gen}, \mathsf{GenAvoid}, \mathsf{Hash})$ as follows:

- $\mathsf{Gen}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$: On input a security parameter λ, a description size bound s, an input length ℓ_{in}, and an output length ℓ_{out}, the generation algorithm does the following:
  1. Sample a base hash key $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  2. Sample a null program $P \leftarrow \mathsf{CiO.ObfNull}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  3. Output the hash key $\mathsf{hk} = (\mathsf{hk}', P)$.
- $\mathsf{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E)$: On input a security parameter λ, a description size bound s, an input length ℓ_{in}, an output length ℓ_{out}, and an enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ of size at most s, the algorithm does the following:
  1. Sample a base hash key $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  2. Construct the circuit $C = C[\mathsf{hk}', E]$ described in Fig. 9 below. Since $N \leq 2^{\ell_{\mathsf{out}}} - 1$, the circuit is well-defined; namely, it is always able to find a string $z \in \{0,1\}^{\ell_{\mathsf{out}}}$ outside a set of size at most N.

Constants: A base hash key hk' and an enumerator circuit E.
Inputs: An input $x \in \{0,1\}^{\ell_{\text{in}}}$ and an index $i \in [N]$.

1. Compute the list of elements $(y_1, \ldots, y_N) = E(x)$.
2. Compute the base evaluation $y' = \mathsf{Hash}'(\mathsf{hk}', x)$.
3. If $i \le N$ and $y' = y_i$:
   - Find the lexicographically first string $z \in \{0,1\}^{\ell_{\text{out}}} \setminus \{y_1, \ldots, y_N\}$.
   - Output z.
4. Otherwise, output $\bot$.

Figure 9: Circuit $C[\mathsf{hk}', E]$.

3. Construct the program $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\text{CiO}}}, 1^N, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, C, 0)$.
4. Output the hash key $\mathsf{hk} = (\mathsf{hk}', P)$.

- $\mathsf{Hash}(\mathsf{hk}, x)$: On input a hash key $\mathsf{hk} = (\mathsf{hk}', P)$ and an input $x \in \{0,1\}^{\ell_{\text{in}}}$, the evaluation algorithm does the following:

  1. For $i = 1$ to N:
     - Compute $y \leftarrow \mathsf{CiO.Eval}(P, x, i)$.
     - If $y \neq \bot$, output y and terminate.
  2. If no evaluation overrides, output $\mathsf{Hash}'(\mathsf{hk}', x)$.

Theorem 6.6 (Perfect Correlation Intractability)**.** *If Π_{CiO} is correct, then Construction 6.5 is perfectly correlation intractable for all (s, N)-enumerable relations.*

Proof. Let $\lambda, s, N, \ell_{\text{in}}, \ell_{\text{out}} \in \mathbb{N}$. Let $\mathcal{R} \subseteq \{0,1\}^{\ell_{\text{in}}} \times \{0,1\}^{\ell_{\text{out}}}$ be an (s, N)-enumerable relation with enumerator circuit $E\colon \{0,1\}^{\ell_{\text{in}}} \to (\{0,1\}^{\ell_{\text{out}}})^N$. Let $x \in \{0,1\}^{\ell_{\text{in}}}$, and let $\mathsf{hk} = (\mathsf{hk}', P)$ be a hash key in the support of $\mathsf{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, E)$. By construction, P is generated by CiO.Obf for the circuit $C = C[\mathsf{hk}', E]$ with threshold $t = 0$. We analyze the execution of $\mathsf{Hash}(\mathsf{hk}, x)$. The evaluation algorithm computes $(y_1, \ldots, y_N) = E(x)$. Let $y' = \mathsf{Hash}'(\mathsf{hk}', x)$. We proceed by a case analysis:

- Suppose $y' \in \{y_1, \ldots, y_N\}$. Then, there exists some index $j \le N$ such that $y' = y_j$. During the evaluation loop in $\mathsf{Hash}(\mathsf{hk}, x)$, the algorithm queries P on (x, j). Because $j \ge 1 > 0$, the cutoff-$i\mathcal{O}$ correctness property guarantees that $\mathsf{CiO.Eval}(P, x, j) = C(x, j)$. Evaluating $C(x, j)$ satisfies the conditional branch $(j \le N \wedge y' = y_j)$, so the circuit computes the lexicographically first string $z \in \{0,1\}^{\ell_{\text{out}}} \setminus \{y_1, \ldots, y_N\}$ and outputs z. Since $z \in \{0,1\}^{\ell_{\text{out}}}$, we have $z \neq \bot$, so the Hash loop immediately terminates and outputs z. Because $z \notin \{y_1, \ldots, y_N\}$, we have $(x, z) \notin \mathcal{R}$ by definition of $E(x)$.

- Suppose $y' \notin \{y_1, \ldots, y_N\}$. Then, for all indices $i \in [N]$, either $i > N$ or $y' \neq y_i$. By the correctness of Π_{CiO}, for every $i \in [N]$ we have $i > 0$ and hence $\mathsf{CiO.Eval}(P, x, i) = C(x, i) = \bot$. The loop runs to completion without triggering an override, and the Hash algorithm outputs y'. Because $y' \notin \{y_1, \ldots, y_N\}$, we have $y' \notin E(x)$, which implies $(x, y') \notin \mathcal{R}$.

In all cases, $(x, \mathsf{Hash}(\mathsf{hk}, x)) \notin \mathcal{R}$ and perfect correlation intractability holds. □

Theorem 6.7 (Mode Indistinguishability)**.** *Assume that Π'_{CI} is a somewhere-statistically correlation-intractable hash for search relations and Π_{CiO} satisfies index indistinguishability and circuit hiding. Then Construction 6.5 satisfies mode indistinguishability.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the mode indistinguishability game of Π_{CI}. For any $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ and any $t \in [N]$, let E_t be the circuit defined in Fig. 10.

Constants: An enumerator $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ and an index $t \in [N]$.
Inputs: A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. Compute $E(x) = (y_1, \ldots, y_N)$.
2. If $t \le N$, output y_t. Otherwise, if $t > N$, output $0^{\ell_{\mathsf{out}}}$.

Figure 10: Circuit E_t.

We define the following sequence of hybrids:

- Real_0: This is the real mode indistinguishability game with a normal hash key (not avoiding).
  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends $1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$, and an enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ of size at most s to the challenger.
  2. The challenger samples a base hash key $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  3. The challenger samples a null program $P \leftarrow \mathsf{CiO.ObfNull}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$.
  4. The challenger sets $\mathsf{hk} = (\mathsf{hk}', P)$ and sends hk to $\mathcal{A}$.
  5. The adversary $\mathcal{A}$ outputs a guess b' which is the output of the experiment.
- Hyb_t for each $t \in \{0, \ldots, N\}$: This is the same as Real_0, except that the challenger samples a hash key in avoiding mode, with the cutoff for cutoff-$i\mathcal{O}$ set to t instead of 0. Note that the cutoff now runs *downwards*: Hyb_N is the fully switched-off program and Hyb_0 is the fully functional one.
  3. The challenger samples $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$ where $C = C[\mathsf{hk}', E]$ is the circuit from Fig. 9.
- Real_1: The real mode indistinguishability game with a fully avoiding hash key. This is exactly Hyb_0.

For each $t \in [N]$, we define an inner sequence of hybrids:

- $\mathsf{Hyb}_{t,1}$: This is the same as Hyb_t, except that the underlying somewhere-statistically correlation-intractable hash avoids the function E_t.
  2. The challenger samples a base hash key $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$.
- $\mathsf{Hyb}_{t,2}$: This is the same as $\mathsf{Hyb}_{t,1}$, except that the experiment aborts if there exists a correlation under hk' for the relation E_t.
  2. The challenger samples a base hash key $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$. If there exists $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $\mathsf{Hash}'(\mathsf{hk}', x) = E_t(x)$, the experiment aborts and outputs $b' = 0$.
- $\mathsf{Hyb}_{t,3}$: This is the same as $\mathsf{Hyb}_{t,2}$, except that the cutoff-$i\mathcal{O}$ threshold decreases to $t - 1$.
  3. The challenger samples a program $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t-1)$ where $C = C[\mathsf{hk}', E]$ is the circuit from Fig. 9.
- $\mathsf{Hyb}_{t,4}$: This is the same as $\mathsf{Hyb}_{t,3}$, except that the abort condition is removed.
  2. The challenger samples a base hash key $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$, with no abort condition.

Claim 6.8. *Assume that Π_{CiO} satisfies circuit hiding. Then for any efficient $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_N(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_N(\mathcal{A}) = 1]\right|$. The only difference between Real_0 and Hyb_N is how the program P is generated: in Real_0 it is a null program $P \leftarrow \mathsf{CiO.ObfNull}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, whereas in Hyb_N it is an obfuscation $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, N)$ of the circuit $C = C[\mathsf{hk}', E]$ at cutoff N; the base hash key $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ is sampled identically in both. We construct an algorithm $\mathcal{B}$ that breaks the circuit hiding of Π_{CiO} with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and an enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ samples a base hash key $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ and constructs the circuit $C = C[\mathsf{hk}', E]$ from Fig. 9.
3. Algorithm $\mathcal{B}$ sends $1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and the circuit C to the circuit hiding challenger, which replies with a program P.
4. Algorithm $\mathcal{B}$ sets $\mathsf{hk} = (\mathsf{hk}', P)$ and sends hk to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. Moreover, if the circuit hiding challenger samples $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, N)$ (i.e., the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_N(\mathcal{A})$, whereas if it samples $P \leftarrow \mathsf{CiO.ObfNull}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ (i.e., the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Real}_0(\mathcal{A})$. Therefore $\mathcal{B}$ breaks the circuit hiding of Π_{CiO} with the same advantage ε. □

Claim 6.9. *Assume that Π'_{Cl} satisfies mode indistinguishability. Then for any efficient $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $t \in [N]$*

$$\left|\Pr[\mathsf{Hyb}_t(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,1}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Fix an efficient $\mathcal{A}$ and $t \in [N]$, and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_t(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,1}(\mathcal{A}) = 1]\right|$. The only difference between Hyb_t and $\mathsf{Hyb}_{t,1}$ is how the base hash key is sampled: in Hyb_t it is a normal key $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, whereas in $\mathsf{Hyb}_{t,1}$ it is an avoiding key $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$ for the search relation described by E_t; in both hybrids the program is generated as $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$ from the circuit $C = C[\mathsf{hk}', E]$. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π'_{Cl} with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and an enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ constructs the circuit E_t from Fig. 10, which describes the search relation R_{E_t} and has size at most s', and sends $1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and E_t to the mode indistinguishability challenger of Π'_{Cl}, which replies with a base hash key hk'.
3. Algorithm $\mathcal{B}$ constructs the circuit $C = C[\mathsf{hk}', E]$ from Fig. 9 and samples $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t)$.
4. Algorithm $\mathcal{B}$ sets $\mathsf{hk} = (\mathsf{hk}', P)$ and sends hk to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. Moreover, if the challenger samples $\mathsf{hk}' \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ (i.e., the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_t(\mathcal{A})$, whereas if it samples $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$ (i.e., the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_{t,1}(\mathcal{A})$. Therefore $\mathcal{B}$ breaks the mode indistinguishability of Π'_{Cl} with the same advantage ε. □

Claim 6.10. *Assume that Π'_{Cl} is statistically correlation intractable for search relations. Then for any (potentially) unbounded $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $t \in [N]$*

$$\left|\Pr[\mathsf{Hyb}_{t,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,2}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Fix an adversary $\mathcal{A}$ and $t \in [N]$, and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_{t,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,2}(\mathcal{A}) = 1]\right|$. The only difference between $\mathsf{Hyb}_{t,1}$ and $\mathsf{Hyb}_{t,2}$ is that $\mathsf{Hyb}_{t,2}$ aborts and outputs 0 if there exists $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $\mathsf{Hash}'(\mathsf{hk}', x) = E_t(x)$; denote this event by A. Since the two experiments are identical unless A occurs, we have $\varepsilon \le \Pr[A]$. We build an (unbounded) adversary $\mathcal{B}$ for the statistical correlation intractability game of Π'_{CI} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and an enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ constructs the circuit E_t from Fig. 10, which describes the search relation R_{E_t} and has size at most s', and sends $1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and E_t to the statistical correlation intractability challenger of Π'_{CI}, which replies with an avoiding hash key $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$.
3. Algorithm $\mathcal{B}$ searches over every $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ for some x such that $\mathsf{Hash}'(\mathsf{hk}', x) = E_t(x)$. If such an x exists, it outputs that x; otherwise it outputs an arbitrary $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

Although $\mathcal{B}$ is not efficient, this is admissible since statistical correlation intractability holds against unbounded adversaries. Since $\mathcal{B}$ runs $\mathcal{A}$ to generate the circuit E_t and finds a colliding x exactly when A occurs, the probability that $\mathcal{B}$ breaks statistical correlation intractability is exactly $\Pr[A] \ge \varepsilon$. By the statistical correlation intractability of Π'_{CI}, this probability is negligible, and therefore so is ε. □

Claim 6.11. *Assume that Π_{CiO} satisfies index indistinguishability. Then for any efficient $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $t \in [N]$*

$$\left|\Pr[\mathsf{Hyb}_{t,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,3}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Fix an efficient $\mathcal{A}$ and $t \in [N]$, and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_{t,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,3}(\mathcal{A}) = 1]\right|$. The only difference between $\mathsf{Hyb}_{t,2}$ and $\mathsf{Hyb}_{t,3}$ is the cutoff passed to the obfuscator: t in $\mathsf{Hyb}_{t,2}$ and $t-1$ in $\mathsf{Hyb}_{t,3}$, both with the same circuit $C = C[\mathsf{hk}', E]$, the same avoiding base key $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$, and the same abort condition. We construct an algorithm $\mathcal{B}$ that breaks the index indistinguishability of Π_{CiO} with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $1^s, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and an enumerator circuit $E\colon \{0,1\}^{\ell_{\mathsf{in}}} \to (\{0,1\}^{\ell_{\mathsf{out}}})^N$ to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ constructs the circuit E_t from Fig. 10 and samples an avoiding base hash key with respect to that circuit $\mathsf{hk}' \leftarrow \mathsf{GenAvoid}'(1^\lambda, 1^{s'}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, E_t)$.
3. Algorithm $\mathcal{B}$ constructs the circuit $C = C[\mathsf{hk}', E]$ from Fig. 9 and sends $1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C$ and the index t to the index indistinguishability challenger, which replies with a program $P \leftarrow \mathsf{CiO.Obf}(1^\lambda, 1^{s_{\mathsf{CiO}}}, 1^N, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C, t-b)$.
4. Algorithm $\mathcal{B}$ sets $\mathsf{hk} = (\mathsf{hk}', P)$ and sends hk to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

We argue that $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{t,2}$ and $\mathsf{Hyb}_{t,3}$ depending on how the Π_{CiO} challenger samples the obfuscated program. First note that $\mathcal{B}$ perfectly simulates the abort condition without explicitly checking for it: by construction of C in Fig. 9 and since $t \le N$, there exists $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $C(x,t) \ne \bot$ if and only if there exists $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $\mathsf{Hash}'(\mathsf{hk}', x) = E_t(x)$. By the abort condition of index indistinguishability in Definition 3.1, the challenger aborts (and outputs 0) if there exists $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $C(x,t) \ne \bot$, which is exactly the abort condition of $\mathsf{Hyb}_{t,2}$ and $\mathsf{Hyb}_{t,3}$; in both cases the experiment then outputs 0. Next, if $\mathcal{A}$ is efficient then so is $\mathcal{B}$; in particular $\mathcal{B}$ never itself decides the abort. Finally, if the challenger samples P at cutoff t (i.e., the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_{t,2}(\mathcal{A})$, whereas if it samples P at cutoff $t-1$ (i.e., the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_{t,3}(\mathcal{A})$. Therefore $\mathcal{B}$ breaks the index indistinguishability of Π_{CiO} with the same advantage ε. □

Claim 6.12. *Assume that Π'_{CI} is statistically correlation intractable for search relations. Then for any (potentially) unbounded $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $t \in [N]$*

$$\left|\Pr[\mathsf{Hyb}_{t,3}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t,4}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. The proof is completely analogous to that of Claim 6.10. □

Claim 6.13. *Assume that* Π'_{Cl} *satisfies mode indistinguishability. Then for any efficient* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and all* $t \in [N]$

$$\left|\Pr[\mathsf{Hyb}_{t,4}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{t-1}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. The proof is completely analogous to that of Claim 6.9. □

By applying Claims 6.9 to 6.13 a total of N times (for $t = N, N-1, \ldots, 1$), we get that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathsf{Hyb}_{N}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

By Claim 6.8, we have that Real_0 and Hyb_N are computationally indistinguishable. Finally, Real_1 and Hyb_0 are identical. The theorem follows. □

Acknowledgments

Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator Award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, an Amazon Research Award, and a gift from the Stellar Development Foundation.

Statement on AI use. All ideas in this work were developed without the use of AI. The use of AI was restricted to preparing initial drafts of the write-up, as well as copy editing. The authors takes full responsibility of the correctness of this paper.

References

[AMR26] Damiano Abram, Giulio Malavolta, and Lawrence Roy. How to authenticate a non-deterministic computation: Shift-hiding functions, compressed LWE sampling, broadcast encryption, and obfuscation. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/741.pdf.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *TCC*, 2014.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013.

[BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In *EUROCRYPT*, 2020.

[BDGM22] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: circular-secure LWE suffices. In *ICALP*, 2022.

[BDJ+25] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. In *CRYPTO*, 2025.

[BFN26] David Balbás, Dario Fiore, and Duy Nguyen. Splitting bilinear groups: New translations from composite-to prime-order with applications to batch arguments for NP. In *CRYPTO*, 2026.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BIJ+20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In *ITCS*, 2020.

[BIOW20] Ohad Barta, Yuval Ishai, Rafail Ostrovsky, and David J Wu. On succinct arguments and witness encryption from groups. In *CRYPTO*, 2020.

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. In *EUROCRYPT*, 2018.

[BLM+24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor tracing without trusted authority from registered functional encryption. In *ASIACRYPT*, 2024.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *ACM CCS*, 2006.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO*, 1997.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N Rothblum, Ron D Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In *STOC*, 2019.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Obfuscating point functions with multibit output. In *EUROCRYPT*, 2008.

[CEW25] Binyi Chen, Noel Elias, and David J Wu. Pairing-based batch arguments for NP with a linear-size CRS. In *ASIACRYPT*, 2025.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, 1994.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, 1998.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, 2021.

[CLW25] Valerio Cini, Russell W. F. Lai, and Ivy K. Y. Woo. Lattice-based obfuscation from NTRU and equivocal LWE. In *CRYPTO*, 2025.

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *TCC*, 2010.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *CRYPTO*, 2018.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J Wu. Succinct witness encryption for batch languages and applications. In *ASIACRYPT*, 2025.

[DQV+21] Lalita Devadas, Willy Quach, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Succinct LWE sampling, random polynomials, and obfuscation. In *TCC*, 2021.

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO*, 1993.

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

[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.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *FOCS*, 2017.

[GLW14] Craig Gentry, Allison Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, 2014.

[GMM17] Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. Lower bounds on obfuscation from all-or-nothing encryption primitives. In *CRYPTO*, 2017.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *STOC*, 2021.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *PKC*, 2019.

[GY26a] Rishab Goyal and Saikumar Yadugiri. Adaptively-secure flexible and identity-based broadcast encryption from decomposed LWE. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/862.pdf.

[GY26b] Rishab Goyal and Saikumar Yadugiri. Equivocal broadcast encryption: Adaptively-secure optimal distributed broadcast encryption from lattices. In *CRYPTO*, 2026.

[HJL21] Samuel B. Hopkins, Aayush Jain, and Huijia Lin. Counterexamples to new circular security assumptions underlying iO. In *CRYPTO*, 2021.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum iO from circular security with random opening assumption. In *CRYPTO*, 2025.

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D Rothblum. Fiat–Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In *STOC*, 2021.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

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

[HW26] Justin Holmgren and Brent Waters. Separating selective opening security from standard security, assuming indistinguishability obfuscation. In *PKC*, 2026.

[JKLM25] Zhengzhong Jin, Yael Tauman Kalai, Alex Lombardi, and Surya Mathialagan. Universal SNARGs for NP from proofs of correctness. In *STOC*, 2025.

[JLLS23] Aayush Jain, Huijia Lin, Paul Lou, and Amit Sahai. Polynomial-time cryptanalysis of the subspace flooding assumption for post-quantum iO. In *EUROCRYPT*, 2023.

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

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In *EUROCRYPT*, 2022.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[Nas26] Shafik Nassar. The power of rerandomization in Obfustopia: Collision-resistant hash, somewhere-extractable BARGs, and more. Cryptology ePrint Archive, Paper 2026/1785, 2026.

[NNL01] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, 2001.

[NP00] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In *International Conference on Financial Cryptography*, 2000.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO*, 2019.

[RVV25] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In *TCC*, 2025.

[SRG+26] Lev Soukhanov, Yaroslav Rebenko, Muhammad El Gebali, Mikhail Komarov, Handan Kilinc-Alper, Michel Abdalla, and Patrick Towa. Implementable witness encryption from arithmetic affine determinant programs. *IACR Cryptol. ePrint Arch.*, 2026.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In *CRYPTO*, 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In *ASIACRYPT*, 2022.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, 2005.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In *EUROCRYPT*, 2021.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, 2022.

[WW24] Brent Waters and Daniel Wichs. Adaptively secure attribute-based encryption from witness encryption. In *TCC*, 2024.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *FOCS*, 2017.

[Zha16] Mark Zhandry. The magic of ELFs. In *CRYPTO*, 2016.

A Somewhere Statistically Sound BARGs

This section recalls the BARG construction of Choudhuri, Jain, and Jin [CJJ21] and instantiates it with the somewhere-statistically correlation-intractable hash function from Section 6 (Construction 6.5). This yields a somewhere *statistically* sound BARG. Previously, the work of [CJJ21] showed somewhere computational soundness for their BARG (since the underlying correlation-intractable hash function only provided computational correlation intractability for enumerable relations). The construction and its analysis are essentially theirs; the only change is the building block, and it is what upgrades the guarantee to somewhere *statistical* extraction. We give the argument in full for completeness.

A.1 Building Blocks

We start by recalling the main building blocks underlying the construction of [CJJ21].

Definition A.1 (Somewhere-Extractable Hash)**.** Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of block spaces where each block in $\mathcal{X}_\lambda$ has a polynomial size description. When λ is clear from context, we abuse notation and write $x \in \mathcal{X}$ to refer to $x \in \mathcal{X}_\lambda$. A somewhere-extractable hash function for block space $\mathcal{X}$ is a tuple of efficient algorithms $\Pi_{\mathsf{SEH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{hk}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a hash key hk.
- $\mathsf{SetupBinding}(1^\lambda, i^*) \to (\mathsf{hk}, \mathsf{td})$: On input the security parameter λ and an index $i^* \in \mathbb{N}$, the binding setup algorithm outputs a hash key hk and a trapdoor td.
- $\mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_k)) \to (\mathsf{dig}, \sigma_1, \ldots, \sigma_k)$: On input a hash key hk and a vector of blocks $(x_1, \ldots, x_k) \in \mathcal{X}^k$, the hash algorithm outputs a digest dig and openings $\sigma_1, \ldots, \sigma_k$.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, y, \sigma) \to b'$: On input a hash key hk, a digest dig, an index $i \in \mathbb{N}$, a block $y \in \mathcal{X}$, and an opening σ, the verification algorithm outputs a bit $b' \in \{0, 1\}$.
- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) \to y$: On input a trapdoor td and a digest dig, the extraction algorithm outputs a block $y \in \mathcal{X}$. This algorithm is deterministic.

Moreover, Π_{SEH} should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, all $k \in \mathbb{N}$ with $k < 2^\lambda$, all vectors $(x_1, \ldots, x_k) \in \mathcal{X}^k$, and all indices $i \in [k]$, we have:

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x_i, \sigma_i) = 1 \;\middle|\; \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{dig}, \sigma_1, \ldots, \sigma_k) \leftarrow \mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_k)) \end{array}\right] = 1.$$

- **Mode indistinguishability:** For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends an index $i^* \in \mathbb{N}$ to the challenger.
  2. The challenger computes the public parameters as follows:
     - If $b = 0$, the challenger computes $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda)$.
     - If $b = 1$, the challenger computes $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SetupBinding}(1^\lambda, i^*)$.

     The challenger sends hk to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

We say that Π_{SEH} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Somewhere extraction:** For all $\lambda \in \mathbb{N}$, all indices $i^* \in \mathbb{N}$, and all $(\mathsf{hk}, \mathsf{td})$ in the support of $\mathsf{SetupBinding}(1^\lambda, i^*)$, the following holds: for any digest dig, any block $y \in \mathcal{X}$, and any opening σ, if $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i^*, y, \sigma) = 1$, then $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = y$.
- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $k \in \mathbb{N}$ with $k < 2^\lambda$, all vectors $\mathbf{x} \in \mathcal{X}^k$, all hk in the support of $\mathsf{Setup}(1^\lambda)$, and $(\mathsf{dig}, \sigma_1, \ldots, \sigma_k) = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$, the following bounds hold:
  - **Succinct hash key:** $|\mathsf{hk}| \le \mathsf{poly}(\lambda)$.
  - **Succinct digest:** $|\mathsf{dig}| \le \mathsf{poly}(\lambda)$.
  - **Succinct local opening:** For all $i \in [k]$, $|\sigma_i| \le \mathsf{poly}(\lambda)$.

Fact A.2 (Somewhere-Extractable Hash from FHE [HW15])**.** Assuming the existence of fully homomorphic encryption (FHE), there exists a somewhere-extractable hash function.

PCP with fast online verification. We recall the definition of PCPs with a *fast online verification property* as written in [CJJ21], and implicitly considered by Bitansky et al. [BCI+13]. At a high level, such a property requires that for any PCP for the circuit satisfiability language

$$\text{Circuit-SAT} = \{x \mid \exists w : C(x, w) = 1\},$$

the verification algorithm can be split into two parts: (i) a query algorithm Query which generates the PCP queries that depend on C but are independent of x; and (ii) an online verification algorithm Verify, which depends on x but its running time grows only polylogarithmically in $|C|$ and polynomially in n.

Definition A.3 (PCP with Fast Online Verification [CJJ21])**.** Let $C\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be a Boolean circuit. A *PCP with fast online verification* for Circuit-SAT is a tuple of polynomial-time algorithms $\Pi_{\mathsf{PCP}} = (\mathsf{Prove}, \mathsf{Query}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Prove}(1^\lambda, C, x, w) \to \pi$: The prover algorithm takes as input a security parameter λ, the circuit C, an instance x, and its witness w, and outputs a PCP proof $\pi \in \{0,1\}^*$.
- $\mathsf{Query}(1^\lambda, C) \to (Q, \mathsf{st})$: On input the security parameter λ and the circuit C, the randomized query algorithm outputs a subset of indices $Q \subseteq [|\pi|]$ and a state st.
- $\mathsf{Verify}(x, \mathsf{st}, \tau) \to b$: On input an instance x, a state st, and a binary string $\tau \in \{0,1\}^{|Q|}$, the online verification algorithm deterministically decides to accept (outputs 1) or reject (outputs 0).

Furthermore, we require the following properties of the PCP:

- **Completeness:** For any circuit C, any instance $x \in$ Circuit-SAT, and any witness w for x, we have:

$$\Pr\left[\mathsf{Verify}(x, \mathsf{st}, \pi|_Q) = 1 : \begin{array}{c} \pi \leftarrow \mathsf{Prove}(1^\lambda, C, x, w) \\ (Q, \mathsf{st}) \leftarrow \mathsf{Query}(1^\lambda, C) \end{array}\right] = 1,$$

  where $\pi|_Q$ denotes the subset of the bits of π indexed by Q.
- **Polynomial proof size:** The size of the proof π is bounded by $\mathsf{poly}(\lambda, |C|)$.
- **Small query complexity:** The size of the set Q is bounded by $\mathsf{poly}(\lambda, \log |C|)$.

- **Succinct verification:** The state st can be represented in $\mathsf{poly}(\lambda, n, \log|C|)$ bits, and the online verification algorithm Verify runs in time $\mathsf{poly}(\lambda, n, \log|C|)$. The query algorithm Query runs in time $\mathsf{poly}(\lambda, |C|)$.
- **Proof of knowledge:** There exists a deterministic polynomial-time extractor algorithm E and a negligible function $\rho(\lambda)$. We define the following experiment for any unbounded adversary $\mathcal{A}$:
  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ sends a circuit C, an instance x, and a proof $\pi^* \in \{0,1\}^*$ to the challenger.
  2. The challenger computes the exact success probability of the proof:
  $$\varepsilon = \Pr\left[\mathsf{Verify}(x, \mathsf{st}, \pi^*|_Q) = 1 : (Q, \mathsf{st}) \leftarrow \mathsf{Query}(1^\lambda, C)\right].$$
  3. If $\varepsilon \le \rho(\lambda)$, the experiment aborts and outputs 0.
  4. The challenger extracts the witness by computing $w \leftarrow E(\pi^*)$.
  5. The experiment outputs 1 if $C(x, w) = 0$, and 0 otherwise.

  We say that the PCP is a proof of knowledge if for any unbounded adversary, the experiment always outputs 0.

Fact A.4 (PCP with Fast Online Verification [CJJ21])**.** There exists a PCP with fast online verification for Circuit-SAT that is a proof of knowledge.

Relations and correlation intractability. We recall the definition of PCP bad relations [CJJ21].

Definition A.5 (Bad Relation for PCP [CJJ21])**.** Let Π_{SEH} be a somewhere-extractable hash (Definition A.1), and let Π_{PCP} be a PCP with fast online verification (Definition A.3) and proof of length $\ell = \ell(\lambda) \in \mathbb{N}$. For a security parameter $\lambda \in \mathbb{N}$, an instance length $n = n(\lambda) \in \mathbb{N}$, a witness length $m = m(\lambda) \in \mathbb{N}$, a number of instances $k = k(\lambda) \in \mathbb{N}$, an instance $x \in \{0,1\}^n$, a target index $i^* \in [k]$, and a somewhere-extractable hash trapdoor $\mathsf{td}_{\mathsf{SEH}}$ output by $\mathsf{SEH.SetupBinding}(1^\lambda, i^*)$, we define the bad relation $\mathsf{BAD}_{\lambda,x,\mathsf{td}_{\mathsf{SEH}}}$ as follows:

- For any circuit $C\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ and any sequence of ℓ digests $\mathsf{dig} = (\mathsf{dig}_1, \ldots, \mathsf{dig}_\ell)$, define the extracted PCP proof $\pi^* \in \{0,1\}^\ell$ bit-by-bit as $\pi^*[j] = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}_j)$ for all $j \in [\ell]$.
- Let $w = E(\pi^*)$, where E is the deterministic proof of knowledge extractor for the PCP.
- The relation $\mathsf{BAD}_{\lambda,x,\mathsf{td}_{\mathsf{SEH}}}$ consists of pairs $\big((C, \mathsf{dig}), r\big)$, where we parse the randomness as a tuple of blocks $r = (r_1, \ldots, r_t) \in (\{0,1\}^m)^t$, such that:
  $$C(x, w) \neq 1 \quad \wedge \quad \mathsf{PCP.Verify}(x, \mathsf{st}, \pi^*|_Q) = 1$$
  where $(Q, \mathsf{st}) = \mathsf{PCP.Query}(1^\lambda, C; r)$.

We denote the family of relations by $\mathsf{BAD}_\lambda = \left\{\mathsf{BAD}_{\lambda,x,\mathsf{td}_{\mathsf{SEH}}}\right\}$. Following Definition 6.1, we take the description $\langle \mathsf{BAD}_{\lambda,x,\mathsf{td}_{\mathsf{SEH}}} \rangle$ to be the circuit that, on input $\big((C, \mathsf{dig}), r\big)$, decides membership in $\mathsf{BAD}_{\lambda,x,\mathsf{td}_{\mathsf{SEH}}}$ as above, with the instance x, the trapdoor $\mathsf{td}_{\mathsf{SEH}}$, the PCP query and verification algorithms, and the proof-of-knowledge extractor E hard-coded. This circuit has size $\mathsf{poly}(\lambda, \log k, |C|)$.

Holmgren, Lombardi, and Rothblum [HLR21] proved that a correlation-intractable hash function for efficiently enumerable relations can be upgraded to a correlation-intractable hash function for efficiently verifiable product relations that satisfy a certain sparsity constraint. Holmgren and Waters [HW26] proved that the transformation preserves statistical correlation intractability. Finally, Choudhuri, Jain, and Jin [CJJ21] observed that via parallel repetition, they can get a PCP for which the relation BAD from Definition A.5 is sufficiently sparse, and moreover, is essentially an efficiently verifiable product relation. Therefore, the combination of all of those works proves the following corollary:

Corollary A.6 (Correlation Intractability for the PCP Bad Relation [HLR21, CJJ21, HW26])**.** *Assume there exists a hash scheme that is somewhere-statistically correlation-intractable (as per Definition 6.1) for efficiently enumerable relations (Definition 6.4). Then, for any PCP with fast online verification Π_{PCP} that is a proof of knowledge, there exists a hash scheme Π_{CI} that is somewhere-statistically correlation-intractable for the family of relations $\mathsf{BAD} = \{\mathsf{BAD}_\lambda\}$ as defined in Definition A.5.*

Non-interactive batch arguments with statistical extraction. We recall the definition of BARGs for index languages with statistical extraction. As shown by [CJJ21], such arguments can be bootstrapped to BARGs for NP. Waters and Wu [WW22] proved that the transformation preserves statistical extraction.

Definition A.7 (Index BARG)**.** An index batch argument (BARG) is a special case of a BARG for the batch index language. Specifically, it proves that for a given circuit $C\colon [k] \times \{0,1\}^m \to \{0,1\}$, there exist witnesses $w_1, \ldots, w_k \in \{0,1\}^m$ such that $C(i, w_i) = 1$ for all $i \in [k]$. A somewhere-extractable index BARG is a tuple of efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{TrapGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, k, 1^s) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$ (in binary), and a circuit size bound $s \in \mathbb{N}$, the setup algorithm outputs a common reference string crs.
- $\mathsf{TrapGen}(1^\lambda, k, 1^s, i^*) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$ (in binary), a circuit size bound $s \in \mathbb{N}$, and a target extraction index $i^* \in [k]$, the trapdoor generator outputs a common reference string crs and an extraction trapdoor td.
- $\mathsf{Prove}(\mathsf{crs}, C, (w_1, \ldots, w_k)) \to \Pi$: On input a common reference string crs, a circuit C of size at most s, and a vector of k witnesses $w_1, \ldots, w_k \in \{0,1\}^m$, the prover algorithm outputs a proof Π.
- $\mathsf{Verify}(\mathsf{crs}, C, k, \Pi) \to b'$: On input a common reference string crs, a circuit C, the number of instances $k \in \mathbb{N}$ (in binary), and a proof Π, the verification algorithm outputs a bit $b' \in \{0,1\}$.
- $\mathsf{Extract}(\mathsf{td}, \Pi) \to w$: On input the trapdoor td and a proof Π, the deterministic extraction algorithm outputs a single witness $w \in \{0,1\}^m$.

Moreover, Π_{BARG} must satisfy the following properties:

- **Completeness:** For all $\lambda, k, s \in \mathbb{N}$, all circuits C of size at most s, and all witnesses $w_1, \ldots, w_k \in \{0,1\}^m$ such that $C(i, w_i) = 1$ for all $i \in [k]$, we have:

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, k, \Pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s) \\ \Pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (w_1, \ldots, w_k)) \end{array}\right] = 1.$$

- **Mode indistinguishability:** For any efficient adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0,1\}$:

  1. On input 1^λ, algorithm $\mathcal{A}$ sends the number of instances k, the circuit size 1^s and an index $i^* \in [k]$ to the challenger.
  2. The challenger computes the parameters:
     - If $b = 0$, samples $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s)$.
     - If $b = 1$, samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, k, 1^s, i^*)$.

     The challenger sends crs to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{BARG} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Somewhere proof of knowledge:** For any (potentially unbounded) adversary $\mathcal{A}$, we define the following game:

  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ sends $k, 1^s$ and an index $i^* \in [k]$ to the challenger.
  2. The challenger computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, k, 1^s, i^*)$.

3. The challenger sends crs to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs a circuit $C\colon [k] \times \{0,1\}^m \to \{0,1\}$ of size at most s, and a proof Π.
5. The challenger extracts a witness $w \leftarrow \mathsf{Extract}(\mathsf{td}, \Pi)$.
6. The experiment outputs 1 if $\mathsf{Verify}(\mathsf{crs}, C, k, \Pi) = 1$ and $C(i^*, w) \neq 1$. Otherwise, it outputs 0.

The index BARG satisfies somewhere proof of knowledge if for all adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the probability that the experiment outputs 1 is at most $\mathsf{negl}(\lambda)$.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, k, s \in \mathbb{N}$, all $C\colon [k] \times \{0,1\}^m \to \{0,1\}$ of size at most s, and all witnesses $w_1, \ldots, w_k \in \{0,1\}^m$:
  - **Compact CRS:** For all crs in the support of $\mathsf{Gen}(1^\lambda, k, 1^s)$, we have $|\mathsf{crs}| \leq \mathsf{poly}(\lambda, \log k, s)$.
  - **Succinct proof:** For all Π in the support of $\mathsf{Prove}(\mathsf{crs}, C, (w_1, \ldots, w_k))$, we have $|\Pi| \leq \mathsf{poly}(\lambda, \log k, s)$.
  - **Verification time:** The running time of $\mathsf{Verify}(\mathsf{crs}, C, k, \Pi)$ is bounded by $\mathsf{poly}(\lambda, \log k, s)$.

A.2 Constructing Somewhere-Extractable BARGs

We now recall the construction of [CJJ21], instantiated using a somewhere-statistically correlation-intractable hash function for enumerable relations.

Construction A.8 (Recursive Index BARG [CJJ21, adapted])**.** Let k be the number of instances, and without loss of generality, suppose that k is a power of 2. Our construction relies on the following primitives:

- A base somewhere-extractable index BARG $\Pi'_{\mathsf{BARG}} = (\mathsf{Gen}', \mathsf{TrapGen}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{Extract}')$ for $k/2$ instances.
- A PCP with fast online verification $\Pi_{\mathsf{PCP}} = (\mathsf{PCP.Prove}, \mathsf{PCP.Query}, \mathsf{PCP.Verify})$ as per Definition A.3, with proof length ℓ, query complexity t_{PCP}, and deterministic extractor E.
- A somewhere-extractable hash $\Pi_{\mathsf{SEH}} = (\mathsf{SEH.Setup}, \mathsf{SEH.SetupBinding}, \mathsf{SEH.Hash}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract})$ for single bit blocks.
- A somewhere-statistically correlation-intractable hash $\Pi_{\mathsf{CI}} = (\mathsf{CI.Gen}, \mathsf{CI.GenAvoid}, \mathsf{CI.Hash})$ for the relation ensemble BAD from Definition A.5.

Let s be the size bound of the index circuit $C\colon [k] \times \{0,1\}^m \to \{0,1\}$. Let s_{new} be the maximum size of the intermediate circuit described in Fig. 11. Let s_{CI} be an upper bound on the size of the verifier circuit associated with the correlation-intractable hash scheme (which we will explicitly construct in the security proof). We construct the index BARG scheme $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{TrapGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract})$ as follows:

- $\mathsf{Gen}(1^\lambda, k, 1^s)$: On input the security parameter λ, the number of instances k, and a circuit size bound s, the setup algorithm does the following:
  1. Sample the somewhere-extractable hash key: $\mathsf{hk}_{\mathsf{SEH}} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.
  2. Sample the correlation-intractable hash key: $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.Gen}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, where $\ell_{\mathsf{in}} = |C| + \ell \cdot |\mathsf{dig}|$ and the output length provides sufficient randomness for PCP.Query.
  3. Recursively generate the base BARG for $k' = k/2$ instances:
  $$\mathsf{crs}' \leftarrow \mathsf{Gen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}})$$
  4. Output the common reference string $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$.
- $\mathsf{TrapGen}(1^\lambda, k, 1^s, i^*)$: On input the security parameter λ, the number of instances k, a circuit size bound s, and a target extraction index $i^* \in [k]$, the trapdoor generator does the following:

1. Compute the target index for the base BARG: $i' = \lceil i^*/2 \rceil$.
2. Sample the binding somewhere-extractable hash key for the target index:
$$(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$$
3. Sample the correlation-intractable hash key: $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}})$.
4. Recursively generate the base BARG trapdoor parameters:
$$(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$$
5. Output the common reference string $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$ and the trapdoor $\mathsf{td} = \mathsf{td}_{\mathsf{SEH}}$.

- $\mathsf{Prove}(\mathsf{crs}, C, (w_1, \ldots, w_k))$: On input a common reference string $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$, a circuit C of size at most s, and witnesses $w_1, \ldots, w_k \in \{0,1\}^m$, the prover algorithm does the following:
  1. For each $i \in [k]$, generate the PCP proof: $\pi_i \leftarrow \mathsf{PCP.Prove}(1^\lambda, C, i, w_i)$.
  2. Commit to the proofs column-wise: for each bit-position $j \in [\ell]$, define the vector $x_j = (\pi_1[j], \ldots, \pi_k[j]) \in \{0,1\}^k$, and compute:
$$(\mathsf{dig}_j, \sigma_{1,j}, \ldots, \sigma_{k,j}) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}_{\mathsf{SEH}}, x_j)$$
  3. Let $\mathsf{dig} = (\mathsf{dig}_1, \ldots, \mathsf{dig}_\ell)$. Apply the correlation-intractable hash to derive the PCP randomness: $r \leftarrow \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))$.
  4. Derive the PCP queries and state: $(Q, \mathsf{st}) = \mathsf{PCP.Query}(1^\lambda, C; r)$.
  5. For each $i \in [k]$, prepare the projected PCP proof $\tau_i = \pi_i|_Q$ and the corresponding openings $\alpha_i = \{\sigma_{i,q}\}_{q \in Q}$.
  6. Construct the witnesses for the base BARG: for each $i' \in [k/2]$, set $w'_{i'} = (\tau_{2i'-1}, \alpha_{2i'-1}, \tau_{2i'}, \alpha_{2i'})$.
  7. Construct the intermediate circuit $\mathsf{NewRel} = \mathsf{NewRel}[C, \mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}, Q, \mathsf{st}]$ described in Fig. 11 below:

     Constants: A circuit C, a somewhere-extractable hash key $\mathsf{hk}_{\mathsf{SEH}}$, digests dig, queries Q, and a PCP state st.
     Inputs: An index $i' \in [k/2]$ and a grouped witness $w' = (\tau_0, \alpha_0, \tau_1, \alpha_1)$.
     1. Define the original indices $i_0 = 2i' - 1$ and $i_1 = 2i'$.
     2. Parse $Q = (q_1, \ldots, q_{|Q|})$.
     3. For $b \in \{0,1\}$:
        - For each $j \in [|Q|]$, verify the local opening: check $\mathsf{SEH.Verify}(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}_{q_j}, i_b, \tau_b[j], \alpha_b[j]) = 1$.
        - Verify the PCP online: check $\mathsf{PCP.Verify}(i_b, \mathsf{st}, \tau_b) = 1$.
     4. If all checks pass, output 1. Otherwise, output 0.

  Figure 11: Circuit $\mathsf{NewRel}[C, \mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}, Q, \mathsf{st}]$.

  8. Compute the base BARG proof: $\Pi' \leftarrow \mathsf{Prove}'(\mathsf{crs}', \mathsf{NewRel}, (w'_1, \ldots, w'_{k/2}))$.
  9. Output the proof $\Pi = (\mathsf{dig}, \Pi')$.
- $\mathsf{Verify}(\mathsf{crs}, C, k, \Pi)$: On input a common reference string $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$, a circuit C, the number of instances k, and a proof $\Pi = (\mathsf{dig}, \Pi')$, the verification algorithm does the following:
  1. Apply the correlation-intractable hash: $r \leftarrow \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))$.
  2. Derive the PCP queries: $(Q, \mathsf{st}) = \mathsf{PCP.Query}(1^\lambda, C; r)$.
  3. Construct the intermediate circuit $\mathsf{NewRel} = \mathsf{NewRel}[C, \mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}, Q, \mathsf{st}]$.
  4. Delegate the bulk of the verification to the base BARG by outputting $\mathsf{Verify}'(\mathsf{crs}', \mathsf{NewRel}, k/2, \Pi')$.

- $\mathsf{Extract}(\mathsf{td}, \Pi)$: On input a trapdoor $\mathsf{td} = \mathsf{td}_{\mathsf{SEH}}$ and a proof $\Pi = (\mathsf{dig}, \Pi')$, the extraction algorithm does the following:

  1. Recover the full PCP proof for the target index i^* bit-by-bit: for each $j \in [\ell]$, compute:
  $$b_j = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}_j)$$
  2. Reconstruct the target PCP proof $\pi^* = (b_1, \ldots, b_\ell)$.
  3. Apply the deterministic PCP extractor: output $w \leftarrow E(\pi^*)$.

Remark A.9 (Completeness and Succinctness)**.** The completeness and succinctness properties of the index BARG scheme Π_{BARG} follow identically from the construction of Choudhuri, Jain, and Jin [CJJ21].

Theorem A.10 (Somewhere Proof of Knowledge)**.** *Assume that the base index BARG scheme Π'_{BARG} satisfies somewhere proof of knowledge, the hash scheme Π_{SEH} satisfies somewhere extraction, and the hash scheme Π_{CI} is somewhere-statistically correlation-intractable for the family of relations* BAD *(as defined in Definition A.5) defined for the PCP scheme Π_{PCP} with fast verification. Then, the index BARG scheme Π_{BARG} from Construction A.8 satisfies somewhere proof of knowledge.*

Proof. Fix any unbounded adversary $\mathcal{A}$. We define the following sequence of hybrids:

- Hyb_0: This is the real somewhere proof of knowledge game:

  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ sends $k, 1^s$ and an index $i^* \in [k]$ to the challenger.
  2. The challenger computes the trapdoor parameters as follows:
     - Computes the target index for the base BARG: $i' = \lceil i^*/2 \rceil$.
     - Samples the binding somewhere-extractable hash key: $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$.
     - Samples the avoiding correlation-intractable hash key: $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}})$, where $\ell_{\mathsf{in}} = |C| + \ell \cdot |\mathsf{dig}|$.
     - Recursively generates the base BARG trapdoor parameters: $(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$.
     - Sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$, and $\mathsf{td} = \mathsf{td}_{\mathsf{SEH}}$.
  3. The challenger sends crs to $\mathcal{A}$.
  4. The adversary $\mathcal{A}$ outputs a circuit $C\colon [k] \times \{0,1\}^m \to \{0,1\}$ of size at most s, and a proof $\Pi = (\mathsf{dig}, \Pi')$, where $\mathsf{dig} = (\mathsf{dig}_1, \ldots, \mathsf{dig}_\ell)$.
  5. The challenger extracts the witness w as follows:
     - For each $j \in [\ell]$, extracts the j^{th} bit of the target PCP proof: $b_j = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}_j)$.
     - Reconstructs the target PCP proof: $\pi^* = (b_1, \ldots, b_\ell)$.
     - Applies the PCP extractor: $w = E(\pi^*)$.
  6. The challenger evaluates the verification logic as follows:
     - Applies the correlation-intractable hash to derive the PCP randomness: $r \leftarrow \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))$.
     - Derives the PCP queries and state: $(Q, \mathsf{st}) = \mathsf{PCP.Query}(1^\lambda, C; r)$.
     - Constructs the intermediate circuit: $\mathsf{NewRel} = \mathsf{NewRel}[C, \mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}, Q, \mathsf{st}]$.
     - Delegates verification to the base BARG by computing the bit: $v = \mathsf{Verify}'(\mathsf{crs}', \mathsf{NewRel}, k/2, \Pi')$.
  7. The experiment outputs 1 if $v = 1$ and $C(i^*, w) \neq 1$. Otherwise, it outputs 0.

- Hyb_1: Same as Hyb_0, except that the challenger extracts the witness using the underlying BARG, and additionally checks that it satisfies the relation NewRel:

  7. The challenger computes the output as follows:
     - The challenger extracts $w' = \mathsf{Extract}'(\mathsf{td}', \Pi')$.

– The output of the experiment is 1 if and only if $v = 1$ and $C(i^*, w) \neq 1$ and $\mathsf{NewRel}(i', w') = 1$.

This hybrid is indistinguishable for unbounded adversaries by the somewhere proof of knowledge property of Π'_{BARG}.

- Hyb_2: Same as Hyb_1, except that the challenger extracts the projected PCP proof using the somewhere-extractable hash, and makes sure it is consistent with the witness extracted from the underlying BARG:

  7. The challenger computes the output as follows:
     – The challenger extracts $w' = \mathsf{Extract}'(\mathsf{td}', \Pi')$ and parses $w' = (\tau_{2i'-1}, \alpha_{2i'-1}, \tau_{2i'}, \alpha_{2i'})$. Note that the target index $i^* \in \{2i', 2i' - 1\}$.
     – The challenger extracts $\pi^*_q = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}_q)$ for all $q \in Q$, and assembles the projection proof $\pi^*|_Q = (\pi^*_q)_{q \in Q}$.
     – The output of the experiment is 1 if and only if $v = 1$ and $C(i^*, w) \neq 1$ and $\mathsf{NewRel}(i', w') = 1$ and $\tau_{i^*} = \pi^*|_Q$.

  This hybrid is indistinguishable for unbounded adversaries by the somewhere extraction property of Π_{SEH}. Finally, for any unbounded adversary, this hybrid outputs 1 with negligible probability by the statistical correlation intractability of Π_{CI}.

We now formalize the three steps above. Throughout, note that all three experiments hand $\mathcal{A}$ the same view: namely, the trapdoor common reference string $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$ generated as in Hyb_0, and differ only in how the output bit is computed. In particular, $\mathcal{A}$'s circuit and proof (C, Π) are identically distributed in all three experiments.

Claim A.11. *Assume that Π'_{BARG} satisfies somewhere proof of knowledge. Then for any unbounded adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an unbounded $\mathcal{A}$. Since Hyb_1 differs from Hyb_0 only by imposing the additional conjunct $\mathsf{NewRel}(i', w') = 1$ on the output, where $w' = \mathsf{Extract}'(\mathsf{td}', \Pi')$ and $i' = \lceil i^*/2 \rceil$, we have

$$\begin{aligned}\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] &= \Pr\left[v = 1 \wedge C(i^*, w) \neq 1 \wedge \mathsf{NewRel}(i', w') \neq 1\right] \\ &\leq \Pr\left[v = 1 \wedge \mathsf{NewRel}(i', w') \neq 1\right].\end{aligned}$$

We construct an unbounded adversary $\mathcal{B}$ against the somewhere proof of knowledge game of Π'_{BARG} whose experiment outputs 1 exactly on the event $v = 1 \wedge \mathsf{NewRel}(i', w') \neq 1$:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $k, 1^s$ and an index $i^* \in [k]$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes $i' = \lceil i^*/2 \rceil$ and sends $k/2, 1^{s_{\mathsf{new}}}, i'$ to the somewhere proof of knowledge challenger, which computes $(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$ and returns crs'.
2. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$ to $\mathcal{A}$, where $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$ and $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}})$.
3. Algorithm $\mathcal{A}$ outputs a circuit C and a proof $\Pi = (\mathsf{dig}, \Pi')$.
4. Algorithm $\mathcal{B}$ computes $r \leftarrow \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))$, derives $(Q, \mathsf{st}) = \mathsf{PCP.Query}(1^\lambda, C; r)$, constructs $\mathsf{NewRel} = \mathsf{NewRel}[C, \mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}, Q, \mathsf{st}]$ from Fig. 11, and outputs the circuit NewRel and the proof Π' to the challenger.

The common reference string crs that $\mathcal{B}$ hands to $\mathcal{A}$ is distributed exactly as in Hyb_0 and Hyb_1 (the challenger samples $(\mathsf{crs}', \mathsf{td}')$ via $\mathsf{TrapGen}'$, and $\mathcal{B}$ samples $\mathsf{hk}_{\mathsf{SEH}}$ and $\mathsf{hk}_{\mathsf{CI}}$ as in the construction), so $\mathcal{A}$'s output (C, Π) is identically distributed. The somewhere proof of knowledge experiment for Π'_{BARG} then extracts $w' = \mathsf{Extract}'(\mathsf{td}', \Pi')$ and outputs 1 if and only if $\mathsf{Verify}'(\mathsf{crs}', \mathsf{NewRel}, k/2, \Pi') = 1$ and $\mathsf{NewRel}(i', w') \neq 1$ (i.e., exactly when $v = 1 \wedge \mathsf{NewRel}(i', w') \neq 1$). By the somewhere proof of knowledge of Π'_{BARG}, this probability is bounded by a negligible function $\mathsf{negl}(\lambda)$, which therefore also bounds $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. □

Claim A.12. *Assume that* Π_{SEH} *satisfies somewhere extraction. Then for any unbounded adversary* $\mathcal{A}$*, the following holds:* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

Proof. The experiments Hyb_1 and Hyb_2 are identical except that Hyb_2 imposes the additional conjunct $\tau_{i^*} = \pi^*|_Q$ on the output, where $w' = (\tau_{2i'-1}, \alpha_{2i'-1}, \tau_{2i'}, \alpha_{2i'}) = \mathsf{Extract}'(\mathsf{td}', \Pi')$ and $i^* \in \{2i'-1, 2i'\}$. It suffices to show that whenever Hyb_1 outputs 1, the added conjunct already holds; the two output bits then coincide on every execution. Suppose $\mathsf{Hyb}_1(\mathcal{A})$ outputs 1; in particular $\mathsf{NewRel}(i', w') = 1$. Let $b^* \in \{0,1\}$ be the branch with $i_{b^*} = i^*$, which exists because NewRel sets $i_0 = 2i'-1$ and $i_1 = 2i'$ (Fig. 11) and $i^* \in \{2i'-1, 2i'\}$. Since $\mathsf{NewRel}(i', w') = 1$, all local opening checks of branch b^* pass: for every $j \in [|Q|]$,

$$\mathsf{SEH.Verify}(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{dig}_{q_j}, i^*, \tau_{i^*}[j], \alpha_{i^*}[j]) = 1.$$

Because $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}})$ is in the support of $\mathsf{SEH.SetupBinding}(1^\lambda, i^*)$, the perfect somewhere extraction of Π_{SEH} gives $\mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}_{q_j}) = \tau_{i^*}[j]$ for every j. By the definition of the extracted PCP proof (Definition A.5), $\pi^*[q_j] = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}_{q_j})$, so $\tau_{i^*}[j] = \pi^*[q_j]$ for every $j \in [|Q|]$ (i.e., $\tau_{i^*} = \pi^*|_Q$). Hence $\mathsf{Hyb}_2(\mathcal{A})$ also outputs 1, and the two experiments output 1 with equal probability. □

Claim A.13. *Assume that* Π_{CI} *is somewhere-statistically correlation-intractable for the family* BAD*. Then for any unbounded adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \le \mathsf{negl}(\lambda)$.

Proof. Fix an unbounded $\mathcal{A}$. We first argue that whenever $\mathsf{Hyb}_2(\mathcal{A})$ outputs 1, the value derived by the correlation-intractable hash lands in the bad relation. Consider any execution of $\mathsf{Hyb}_2(\mathcal{A})$ in which $\mathcal{A}$ outputs C and $\Pi = (\mathsf{dig}, \Pi')$, and let $r = \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))$ and $(Q, \mathsf{st}) = \mathsf{PCP.Query}(1^\lambda, C; r)$. If the execution outputs 1, then

- $C(i^*, w) \neq 1$, where $w = E(\pi^*)$ is the witness extracted from the target PCP proof π^*; and
- $\mathsf{NewRel}(i', w') = 1$ and $\tau_{i^*} = \pi^*|_Q$.

Let b^* be the branch with $i_{b^*} = i^*$. Since $\mathsf{NewRel}(i', w') = 1$, the online PCP check of branch b^* passes (Fig. 11): $\mathsf{PCP.Verify}(i^*, \mathsf{st}, \tau_{i^*}) = 1$. Substituting $\tau_{i^*} = \pi^*|_Q$ gives $\mathsf{PCP.Verify}(i^*, \mathsf{st}, \pi^*|_Q) = 1$. Together with $C(i^*, w) \neq 1$, this is precisely the membership condition of $\mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}}$ (Definition A.5) for the pair $\big((C, \mathsf{dig}), r\big)$. Therefore

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \le \Pr\Big[\big((C, \mathsf{dig}), \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))\big) \in \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}}\Big],$$

where the probability is over the sampling of $\mathsf{hk}_{\mathsf{CI}}$ (and $\mathcal{A}$'s coins) in Hyb_2. We bound the right-hand side by the statistical correlation intractability of Π_{CI}. Consider the (unbounded) correlation intractability adversary $\mathcal{B}$ that proceeds as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain its first message $k, 1^s, i^*$, and samples the binding somewhere-extractable hash key $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$.
2. Algorithm $\mathcal{B}$ submits $s_{\mathsf{CI}}, \ell_{\mathsf{in}}, \ell_{\mathsf{out}}$ together with the description $\mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}}$ to the correlation intractability challenger, which replies with an avoiding key $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}})$.
3. Algorithm $\mathcal{B}$ computes $i' = \lceil i^*/2 \rceil$, generates the base BARG parameters $(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$, sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$, and runs $\mathcal{A}$ on crs to obtain C and $\Pi = (\mathsf{dig}, \Pi')$.
4. Algorithm $\mathcal{B}$ outputs $x = (C, \mathsf{dig})$.

The common reference string crs that $\mathcal{B}$ feeds $\mathcal{A}$ is distributed exactly as in Hyb_2, so the pair (C, dig) output by $\mathcal{B}$ is identically distributed. Hence

$$\Pr\Big[(x, \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, x)) \in \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}}\Big] = \Pr\Big[\big((C, \mathsf{dig}), \mathsf{CI.Hash}(\mathsf{hk}_{\mathsf{CI}}, (C, \mathsf{dig}))\big) \in \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}}\Big].$$

Since Π_{CI} is somewhere-statistically correlation-intractable for BAD (Definition 6.1) and $\mathcal{B}$ is a valid (unbounded) adversary, the left-hand side is bounded by a negligible function $\mathsf{negl}(\lambda)$. Combining with the earlier inequality yields $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \le \mathsf{negl}(\lambda)$. □

Combining Claims A.11 to A.13, we get that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda)$. Since Hyb_0 is exactly the somewhere proof of knowledge experiment for Π_{BARG}, the theorem follows. □

Theorem A.14 (Mode Indistinguishability). *Assume that the base index BARG scheme Π'_{BARG}, the somewhere-extractable hash scheme Π_{SEH}, and the correlation intractable hash scheme Π_{CI} all satisfy mode indistinguishability. Then, the index BARG scheme Π_{BARG} from Construction A.8 satisfies mode indistinguishability.*

Proof. Fix any efficient adversary $\mathcal{A}$. We define the following sequence of hybrids:

- Hyb_0: This is the mode indistinguishability game with bit $b = 0$, where the challenger samples the common reference string in normal mode.

  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ sends $k, 1^s$, and an index $i^* \in [k]$ to the challenger.
  2. The challenger samples the standard somewhere-extractable hash key $\mathsf{hk}_{\mathsf{SEH}} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.
  3. The challenger samples the standard correlation-intractable hash key $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.Gen}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, where $\ell_{\mathsf{in}} = |C| + \ell \cdot |\mathsf{dig}|$.
  4. The challenger recursively generates the standard base BARG parameters $\mathsf{crs}' \leftarrow \mathsf{Gen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}})$.
  5. The challenger sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$ and sends crs to $\mathcal{A}$.
  6. The adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

- Hyb_1: This is the same as Hyb_0, except the challenger samples the somewhere-extractable hash key in binding mode at the target index.

  2. The challenger samples $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$.

  This hybrid is indistinguishable from the previous one by the mode indistinguishability of Π_{SEH}.

- Hyb_2: This is the same as Hyb_1, except the challenger samples the correlation-intractable hash key in avoiding mode, which is now possible since $\mathsf{td}_{\mathsf{SEH}}$ is defined.

  3. The challenger samples $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda, i^*, \mathsf{td}_{\mathsf{SEH}}})$.

  This hybrid is indistinguishable from the previous one by the mode indistinguishability of Π_{CI}.

- Hyb_3: This is the same as Hyb_2, except the challenger generates the base BARG parameters in trapdoor mode.

  4. The challenger computes the target index $i' = \lceil i^*/2 \rceil$ for the base BARG and recursively generates the base BARG trapdoor parameters as $(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$.

  This hybrid is indistinguishable from the previous one by the mode indistinguishability of Π'_{BARG}.

We now formalize the three transitions above.

Claim A.15. *Assume that Π_{SEH} satisfies mode indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. The only difference between Hyb_0 and Hyb_1 is how $\mathsf{hk}_{\mathsf{SEH}}$ is sampled: via $\mathsf{SEH.Setup}(1^\lambda)$ in Hyb_0 and via $\mathsf{SEH.SetupBinding}(1^\lambda, i^*)$ in Hyb_1. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π_{SEH} with the same advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $k, 1^s$ and an index $i^* \in [k]$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ sends i^* to the mode indistinguishability challenger of Π_{SEH}, which replies with a hash key $\mathsf{hk}_{\mathsf{SEH}}$.
2. Algorithm $\mathcal{B}$ samples $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.Gen}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ and $\mathsf{crs}' \leftarrow \mathsf{Gen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}})$, sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$, and sends crs to $\mathcal{A}$.

3. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger samples $\mathsf{hk}_{\mathsf{SEH}} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$ (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$; if it samples $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$ (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Note that $\mathcal{B}$ does not require the trapdoor $\mathsf{td}_{\mathsf{SEH}}$ here, since $\mathsf{hk}_{\mathsf{CI}}$ is sampled in normal mode. This is why the somewhere-extractable hash key is switched to binding mode before the correlation-intractable hash key is switched to avoiding mode. We conclude that $\mathcal{B}$ breaks mode indistinguishability with the same advantage ε. □

Claim A.16. *Assume that Π_{CI} satisfies mode indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right|$. The only difference between Hyb_1 and Hyb_2 is how $\mathsf{hk}_{\mathsf{CI}}$ is sampled: via $\mathsf{CI.Gen}$ in Hyb_1 and via $\mathsf{CI.GenAvoid}(\cdots, \mathsf{BAD}_{\lambda,i^*,\mathsf{td}_{\mathsf{SEH}}})$ in Hyb_2. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π_{CI} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $k, 1^s$ and an index $i^* \in [k]$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ samples $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$.
2. Algorithm $\mathcal{B}$ constructs the description $\mathsf{BAD}_{\lambda,i^*,\mathsf{td}_{\mathsf{SEH}}}$ of the bad relation (Definition A.5) and sends $1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}$ and $\mathsf{BAD}_{\lambda,i^*,\mathsf{td}_{\mathsf{SEH}}}$ to the mode indistinguishability challenger of Π_{CI}, which replies with a hash key $\mathsf{hk}_{\mathsf{CI}}$.
3. Algorithm $\mathcal{B}$ samples $\mathsf{crs}' \leftarrow \mathsf{Gen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}})$, sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$, and sends crs to $\mathcal{A}$.
4. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger samples $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.Gen}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$; if it samples $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda,i^*,\mathsf{td}_{\mathsf{SEH}}})$ (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_2(\mathcal{A})$. We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability of Π_{CI} with the same advantage ε. □

Claim A.17. *Assume that Π'_{BARG} satisfies mode indistinguishability. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right|$. The only difference between Hyb_2 and Hyb_3 is how the base parameters crs' are sampled: via $\mathsf{Gen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}})$ in Hyb_2 and via $\mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$ with $i' = \lceil i^*/2 \rceil$ in Hyb_3. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π'_{BARG} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ sends $k, 1^s$ and an index $i^* \in [k]$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes $i' = \lceil i^*/2 \rceil$ and sends $k/2, 1^{s_{\mathsf{new}}}, i'$ to the mode indistinguishability challenger of Π'_{BARG}, which replies with a common reference string crs'.
2. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SEH}}, \mathsf{hk}_{\mathsf{CI}}, \mathsf{crs}')$ to $\mathcal{A}$, where $(\mathsf{hk}_{\mathsf{SEH}}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, i^*)$ and $\mathsf{hk}_{\mathsf{CI}} \leftarrow \mathsf{CI.GenAvoid}(1^\lambda, 1^{s_{\mathsf{CI}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \mathsf{BAD}_{\lambda,i^*,\mathsf{td}_{\mathsf{SEH}}})$.
3. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger samples $\mathsf{crs}' \leftarrow \mathsf{Gen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}})$ (bit $b = 0$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_2(\mathcal{A})$; if it samples $(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{TrapGen}'(1^\lambda, k/2, 1^{s_{\mathsf{new}}}, i')$ (bit $b = 1$), then $\mathcal{B}$ simulates $\mathsf{Hyb}_3(\mathcal{A})$. We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability of Π'_{BARG} with the same advantage ε. □

By Claims A.15 to A.17, for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$. Since Hyb_0 is exactly the mode indistinguishability game in normal mode ($b = 0$) and Hyb_3 is exactly the mode indistinguishability game in trapdoor mode ($b = 1$), the theorem follows. □

B Augmented Broadcast Encryption

In this section, we show how to use indexed positional witness encryption to build augmented broadcast encryption [BW06]. The work of Boneh and Waters [BW06] then shows how to use augmented broadcast encryption to construct a broadcast-and-trace system. The construction is almost identical to that of Goyal et al. [GVW19] that uses standard positional witness encryption instead. The observation is that Goyal et al. [GVW19] implicitly use the indexed version in their security proof.

B.1 Definitions and Building Blocks

We start by recalling the definition of augmented broadcast encryption from [BW06]. At a high level, an augmented broadcast encryption scheme encrypts a message to a set of users $S \subseteq [N]$ together with an index $t \in \{0, 1, \ldots, N\}$, and a user i can recover the message only if $i \in S$ and $i > t$.

Definition B.1 (Augmented Broadcast Encryption)**.** An augmented broadcast encryption scheme over a message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{AugBE}} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to (\mathsf{mpk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$: On input a security parameter λ and a number of users N, the setup algorithm outputs a master public key mpk and secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$, where sk_i is the secret key of user i.
- $\mathsf{Enc}(\mathsf{mpk}, S, \mu, t) \to \mathsf{ct}$: On input a master public key mpk, a set of users $S \subseteq [N]$, a message $\mu \in \mathcal{M}$, and an index $t \in \{0, 1, \ldots, N\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{mpk}, S, \mathsf{ct}) \to \mu$: On input a user index $i \in [N]$, the secret key sk_i of user i, a master public key mpk, a set of users $S \subseteq [N]$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}$ or the special symbol $\perp$. This algorithm is deterministic.

The augmented broadcast encryption scheme Π_{AugBE} should satisfy the following properties:

- **Correctness:** For all $\lambda, N \in \mathbb{N}$, all messages $\mu \in \mathcal{M}$, all sets $S \subseteq [N]$, all indices $t \in \{0, 1, \ldots, N\}$, all users $i \in S \cap \{t+1, t+2, \ldots, N\}$, all $(\mathsf{mpk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N))$ in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, and all ct in the support of $\mathsf{Enc}(\mathsf{mpk}, S, \mu, t)$, the following holds:

$$\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{mpk}, S, \mathsf{ct}) = \mu.$$

- **Succinctness:** We say that the augmented broadcast encryption scheme is succinct if there exists a polynomial poly such that for all $\lambda, N \in \mathbb{N}$, all $(\mathsf{mpk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N))$ in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all sets $S \subseteq [N]$, all messages $\mu \in \mathcal{M}$, all indices $t \in \{0, 1, \ldots, N\}$, and all ct in the support of $\mathsf{Enc}(\mathsf{mpk}, S, \mu, t)$, the following hold:
  - **Succinct public key:** $|\mathsf{mpk}| \le \mathsf{poly}(\lambda, \log N)$.
  - **Succinct secret keys:** $|\mathsf{sk}_i| \le \mathsf{poly}(\lambda)$ for all $i \in [N]$.
  - $\eta(N)$**-succinct ciphertexts:** $|\mathsf{ct}| \le \eta(N) \cdot \mathsf{poly}(\lambda, \log N)$.
- **Message hiding:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the message hiding game as follows:
  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a number of users 1^N.
  2. The challenger samples $(\mathsf{mpk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N)) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and sends the master public key mpk as well as all the secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ outputs a set $S \subseteq [N]$ and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
  4. The challenger samples $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \mu_b, N)$ and sends ct to $\mathcal{A}$.
  5. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

We say that the augmented broadcast encryption scheme satisfies message hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Index indistinguishability:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the index indistinguishability game as follows:

  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a number of users 1^N and an index $t \in [N]$.
  2. The challenger samples $(\mathsf{mpk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N)) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and sends mpk to $\mathcal{A}$.
  3. The adversary can now issue any polynomial number of *pre-challenge corruption queries*. For each query, the adversary sends an index $i \in [N]$. The challenger responds with sk_i.
  4. The adversary $\mathcal{A}$ outputs a set $S \subseteq [N]$ and a message $\mu \in \mathcal{M}$.
  5. The challenger samples $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \mu, t - b)$ and sends ct to $\mathcal{A}$.
  6. The adversary can now issue any polynomial number of *post-challenge corruption queries*. For each query, the adversary sends an index $i \in [N]$. The challenger responds with sk_i.
  7. If $t \in S$ and the adversary queried the secret key of user t at any point, the experiment aborts and outputs $b' = 0$.
  8. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the augmented broadcast encryption scheme satisfies index indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

All-but-one signatures. We additionally rely on *all-but-one* signatures, introduced by Goyal, Vusirikala, and Waters [GVW19]. An all-but-one signature scheme is a signature scheme augmented with a punctured setup algorithm that produces a verification key under which *no* signature verifies at a single punctured message μ^*, while behaving as usual on every other message. For this reason, all-but-one signatures are also commonly referred to as "puncturable signatures."

Definition B.2 (All-But-One Signatures [GVW19])**.** An all-but-one signature scheme with message space $\{0,1\}^n$ and signature space $\{0,1\}^\ell$, where $n = n(\lambda)$ and $\ell = \ell(\lambda)$ are fixed polynomials, is a tuple of efficient algorithms $\Pi_{\mathsf{ABO}} = (\mathsf{Setup}, \mathsf{SetupPunc}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$: On input a security parameter λ, the setup algorithm outputs a signing key sk and a verification key vk.
- $\mathsf{SetupPunc}(1^\lambda, \mu^*) \to (\mathsf{sk}, \mathsf{vk})$: On input a security parameter λ and a message $\mu^* \in \{0,1\}^n$, the punctured setup algorithm outputs a signing key sk and a verification key vk that is punctured at μ^*.
- $\mathsf{Sign}(\mathsf{sk}, \mu) \to \sigma$: On input a signing key sk and a message $\mu \in \{0,1\}^n$, the signing algorithm outputs a signature $\sigma \in \{0,1\}^\ell$.
- $\mathsf{Verify}(\mathsf{vk}, \mu, \sigma) \to b$: On input a verification key vk, a message $\mu \in \{0,1\}^n$, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.

The all-but-one signature scheme Π_{ABO} should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, all messages $\mu \in \{0,1\}^n$, all $(\mathsf{sk}, \mathsf{vk})$ in the support of $\mathsf{Setup}(1^\lambda)$, and all σ in the support of $\mathsf{Sign}(\mathsf{sk}, \mu)$, the following holds:

$$\mathsf{Verify}(\mathsf{vk}, \mu, \sigma) = 1.$$

- **Punctured correctness:** For all $\lambda \in \mathbb{N}$, all messages $\mu^* \in \{0,1\}^n$, all $(\mathsf{sk}, \mathsf{vk})$ in the support of the punctured setup $\mathsf{SetupPunc}(1^\lambda, \mu^*)$, and all signatures $\sigma \in \{0,1\}^\ell$, the following holds:

$$\mathsf{Verify}(\mathsf{vk}, \mu^*, \sigma) = 0.$$

- **Verification key indistinguishability:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the verification key indistinguishability game as follows:

  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a message $\mu^* \in \{0,1\}^n$.
  2. The challenger computes the keys as follows:
     - If $b = 0$, it samples $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
     - If $b = 1$, it samples $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SetupPunc}(1^\lambda, \mu^*)$.

     It sends vk to $\mathcal{A}$.
  3. Throughout the game, the adversary may adaptively make signing queries: on a query $\mu \in \{0,1\}^n$ with $\mu \neq \mu^*$, the challenger replies with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mu)$.
  4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the all-but-one signature scheme satisfies verification key indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

A standard hybrid argument shows that verification key indistinguishability implies the usual notion of selective unforgeability for signatures.

B.2 Construction of Augmented Broadcast Encryption

We now show how to construct an augmented broadcast encryption scheme from an indexed positional witness encryption scheme, a somewhere-extractable hash, and an all-but-one signature scheme. The construction follows Goyal, Vusirikala, and Waters [GVW19], but uses our indexed positional witness encryption definition instead of the general positional witness encryption. Because the syntax of those two primitives is different, we rewrite the construction and prove its security from scratch.

Construction overview. At a high level, a user's secret key is an all-but-one signature on its index $i \in [N]$, and we encode a broadcast set S by a somewhere-extractable hash of the indicator vector associated with the set (i.e., the string $\mathbf{1}_S \in \{0,1\}^N$ that is 1 in position i if and only if $i \in S$). Then, a ciphertext encrypts the message under a positional witness encryption instance where the statement is the hash of the broadcast set and where a valid witness consists of an opening of the hash to 1 at position i together with a signature on i. We give the full construction below:

Construction B.3 (Augmented Broadcast Encryption [GVW19, adapted])**.** Our construction relies on the following primitives:

- A somewhere-extractable hash $\Pi_{\mathsf{SEH}} = (\mathsf{SEH.Setup}, \mathsf{SEH.SetupBinding}, \mathsf{SEH.Hash}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract})$ as per Definition A.1 for single-bit blocks $\mathcal{X}_\lambda = \{0,1\}$. We assume that $\mathsf{SEH.Hash}$ is deterministic.
- An all-but-one signature scheme $\Pi_{\mathsf{ABO}} = (\mathsf{ABO.Setup}, \mathsf{ABO.SetupPunc}, \mathsf{ABO.Sign}, \mathsf{ABO.Verify})$ as per Definition B.2, with message space $\{0,1\}^\lambda$. We identify each index $i \in [N]$ with its binary encoding as a λ-bit string (recall $N \leq 2^\lambda$).
- An indexed positional witness encryption scheme $\Pi_{\mathsf{PWE}} = (\mathsf{PWE.Enc}, \mathsf{PWE.Dec})$ as per Definition 5.1.

For a set $S \subseteq [N]$, let $\mathbf{1}_S \in \{0,1\}^N$ denote its indicator vector (i.e., $\mathbf{1}_S[i] = 1$ if $i \in S$ and $\mathbf{1}_S[i] = 0$ otherwise). We construct the scheme $\Pi_{\mathsf{AugBE}} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$, over the message space $\mathcal{M}$ of Π_{PWE}, as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input a security parameter λ and the number of users N, the setup algorithm does the following:

  1. Sample an all-but-one signature key $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$ and a somewhere-extractable hash key $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.
  2. For each $i \in [N]$, compute a signature $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$.
  3. Output the master public key $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ and the secret keys $\mathsf{sk}_i = \sigma_i$ for each $i \in [N]$.

- $\mathsf{Enc}(\mathsf{mpk}, S, \mu, t)$: On input a master public key $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$, a set of users $S \subseteq [N]$, a message $\mu \in \mathcal{M}$, and an index $t \in \{0, 1, \ldots, N\}$, the encryption algorithm does the following:

  1. Compute the hash of the indicator vector $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$.
  2. Construct the circuit $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ described in Fig. 12 below:

     Constants: A verification key $\mathsf{vk}_{\mathsf{ABO}}$, a somewhere-extractable hash key hk, and a digest dig.
     Inputs: A witness (σ, π) and an index $i \in [N]$.

     1. Check $\mathsf{ABO.Verify}(\mathsf{vk}_{\mathsf{ABO}}, i, \sigma) = 1$.
     2. Check $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, i, 1, \pi) = 1$.
     3. If both checks pass, output 1. Otherwise, output 0.

     Figure 12: Circuit $C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$.

  3. Compute $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$.
  4. Output the ciphertext ct.

- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{mpk}, S, \mathsf{ct})$: On input a user index $i \in [N]$, a secret key $\mathsf{sk}_i = \sigma_i$, a master public key $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$, a set of users $S \subseteq [N]$, and a ciphertext ct, the decryption algorithm does the following:

  1. Recompute $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$.
  2. Output $\mathsf{PWE.Dec}\big(\mathsf{ct}, (\sigma_i, \pi_i), i\big)$.

Remark B.4 (Dynamic Key Generation)**.** We note that this construction supports a more general key-generation functionality: the setup can just sample a master public key $\mathsf{mpk} = \mathsf{vk}_{\mathsf{ABO}}$ and a master secret key $\mathsf{msk} = \mathsf{sk}_{\mathsf{ABO}}$. Then, we can dynamically generate secret keys for new users by just signing indices. This allows for efficient setup and dynamic user key generation.

Remark B.5 (Identity-Based Augmented Broadcast)**.** If the underlying indexed positional witness encryption scheme is fully succinct (i.e., ciphertexts have size $\mathrm{polylog}(N)$ instead of N^ϵ), then this construction would additionally be *identity-based*. We could interpret each identity as an index in $[N]$. Since the dependence on N is only polylogarithmic, we can support identities of polynomial length.

Theorem B.6 (Correctness)**.** *If Π_{SEH}, Π_{ABO}, and Π_{PWE} are correct, then Construction B.3 is correct.*

Proof. Fix $\lambda, N \in \mathbb{N}$, a message $\mu \in \mathcal{M}$, a set $S \subseteq [N]$, an index $t \in \{0, 1, \ldots, N\}$, and a user index $i \in S \cap \{t+1, t+2, \ldots, N\}$. Note that $i \in S$ and $i > t$. Let $(\mathsf{mpk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N))$ be in the support of $\mathsf{Setup}(1^\lambda, 1^N)$ and let ct be in the support of $\mathsf{Enc}(\mathsf{mpk}, S, \mu, t)$. By construction, $\mathsf{sk}_i = \sigma_i$ where $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$, and $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, where $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ is the circuit of Fig. 12 and $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$. Since SEH.Hash is deterministic, the decryption algorithm $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{mpk}, S, \mathsf{ct})$ recomputes the same digest dig and openings $\pi_1, \ldots, \pi_N$, and outputs $\mathsf{PWE.Dec}(\mathsf{ct}, (\sigma_i, \pi_i), i)$. We claim that (σ_i, π_i) is an accepting witness for index i (i.e., $C\big((\sigma_i, \pi_i), i\big) = 1$):

- By correctness of Π_{ABO}, we have $\mathsf{ABO.Verify}(\mathsf{vk}_{\mathsf{ABO}}, i, \sigma_i) = 1$.

- Since $i \in S$, the indicator bit is $\mathbf{1}_S[i] = 1$, so by correctness of Π_{SEH}, we have $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, i, 1, \pi_i) = 1$.

By Fig. 12, both checks passing gives $C\big((\sigma_i, \pi_i), i\big) = 1$. Moreover, $i > t$. By correctness of Π_{PWE}, since $C\big((\sigma_i, \pi_i), i\big) = 1$ and $i > t$, we have $\mathsf{PWE.Dec}(\mathsf{ct}, (\sigma_i, \pi_i), i) = \mu$. Therefore $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{mpk}, S, \mathsf{ct}) = \mu$, as required. □

Theorem B.7 (Succinctness)**.** *Suppose* Π_{SEH} *has succinct hash keys,* Π_{ABO} *has verification keys and signatures of length* $\mathsf{poly}(\lambda)$*, and* Π_{PWE} *is* $\eta(N)$*-succinct. Then Construction B.3 is* $\eta(N)$*-succinct for messages of length* $\mathsf{poly}(\lambda)$*.*

Proof. The master public key is $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$. Since $|\mathsf{vk}_{\mathsf{ABO}}| = \mathsf{poly}(\lambda)$ and $|\mathsf{hk}| = \mathsf{poly}(\lambda, \log N)$, we have $|\mathsf{mpk}| = \mathsf{poly}(\lambda, \log N)$. Each secret key $\mathsf{sk}_i = \sigma_i$ is an all-but-one signature, so $|\mathsf{sk}_i| = \mathsf{poly}(\lambda)$. Finally, $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$ for the circuit $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ of Fig. 12. This circuit runs ABO.Verify and SEH.Verify on the hard-coded $\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}$. Since $|\mathsf{vk}_{\mathsf{ABO}}| = \mathsf{poly}(\lambda)$ and $|\mathsf{hk}|, |\mathsf{dig}| = \mathsf{poly}(\lambda, \log N)$ (using the succinct somewhere-extractable hash key and digest), its size is $|C| = \mathsf{poly}(\lambda, \log N)$. By the $\eta(N)$-succinctness of Π_{PWE}, it follows that $|\mathsf{ct}| \le \eta(N) \cdot \mathsf{poly}(\lambda, |C|, |\mu|, \log N) = \eta(N) \cdot \mathsf{poly}(\lambda, |\mu|, \log N)$, which is $\eta(N) \cdot \mathsf{poly}(\lambda, \log N)$ for messages of length $\mathsf{poly}(\lambda)$. □

Theorem B.8 (Message Hiding)**.** *Assume* Π_{PWE} *satisfies message hiding. Then Construction B.3 satisfies message hiding.*

Proof. Fix any efficient adversary $\mathcal{A}$, and let ε be its advantage in the message hiding game of Construction B.3. We construct an algorithm $\mathcal{B}$ for the message hiding game of Π_{PWE} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to get a number of users 1^N.
2. Algorithm $\mathcal{B}$ computes the parameters as follows:
   - Samples all-but-one signature keys $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$ and a somewhere-extractable hash key $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.
   - For each $i \in [N]$, computes $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$.
   - Sends the master public key $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ together with all the secret keys $\sigma_1, \ldots, \sigma_N$ to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs a set $S \subseteq [N]$ and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
4. Algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$, constructs the circuit $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ from Fig. 12, and sends 1^N, the circuit C, and the messages μ_0, μ_1 to the Π_{PWE} challenger.
5. The challenger sends a ciphertext ct, which $\mathcal{B}$ forwards to $\mathcal{A}$.
6. Algorithm $\mathcal{A}$ outputs a guess b', which $\mathcal{B}$ forwards as its own output.

First note that if $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. Next, for any $b \in \{0, 1\}$, if the Π_{PWE} challenger samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, N, \mu_b)$, then $\mathcal{B}$ perfectly simulates the message hiding game of Construction B.3 where the challenger encrypts the message μ_b. Finally, algorithm $\mathcal{B}$ outputs the same guess as $\mathcal{A}$, therefore we conclude that $\mathcal{B}$ has advantage exactly ε in breaking the message hiding security of Π_{PWE} and the claim follows. □

Theorem B.9 (Index Indistinguishability)**.** *Assume* Π_{SEH} *satisfies mode indistinguishability and somewhere extraction,* Π_{ABO} *satisfies verification key indistinguishability, and* Π_{PWE} *satisfies index indistinguishability. Then Construction B.3 satisfies index indistinguishability.*

Proof. Fix any efficient adversary $\mathcal{A}$. Assume $\mathcal{A}$ never causes the experiment to abort (by making an illegal query). We refer to such an adversary as *admissible*. Then $\mathcal{A}$ never both places t in the challenge set and corrupts user t. Hence, every (non-aborting) execution satisfies $t \notin S$ or $t \notin Q$, where Q denotes the set of corrupted indices. We classify $\mathcal{A}$ into two types and bound the advantage of each type of adversary:

- *Type 1* adversaries always output a challenge set with $t \notin S$.
- *Type 2* adversaries always output $t \in S$ (and hence, by admissibility, never corrupt user t, i.e., $t \notin Q$).

Let Adv_1 and Adv_2 be the maximum advantages among Type 1 and Type 2 adversaries, respectively. Since every admissible adversarial strategy must be one of these two types, we can bound the advantage of $\mathcal{A}$ by $\Pr[t \notin S] \cdot \mathsf{Adv}_1 + \Pr[t \in S] \cdot \mathsf{Adv}_2$. Thus, it suffices to bound Adv_1 and Adv_2 separately. We begin from the following description of the real game, which is common to both cases.

- Real_0: This is the index indistinguishability game where the challenger plays with bit $b = 0$:
  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a number of users 1^N and an index $t \in [N]$.
  2. The challenger computes the parameters as follows:
     - Samples all-but-one signature keys $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$ and a somewhere-extractable hash key $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.
     - For each $i \in [N]$, computes $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$.
     - Sends the master public key $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ to $\mathcal{A}$.
  3. On each pre-challenge corruption query $i \in [N]$, the challenger responds with σ_i.
  4. The adversary $\mathcal{A}$ outputs a set $S \subseteq [N]$ and a message $\mu \in \mathcal{M}$.
  5. The challenger computes the challenge ciphertext as follows:
     - Computes $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$.
     - Constructs the circuit $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ from Fig. 12.
     - Samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$ and sends ct to $\mathcal{A}$.
  6. On each post-challenge corruption query $i \in [N]$, the challenger responds with σ_i. Let $Q \subseteq [N]$ be the set of all corrupted indices.
  7. If $t \in S$ and $t \in Q$, the experiment outputs 0 (this never occurs for admissible adversaries).
  8. The adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

We now consider the analysis for Type 1 adversaries and for Type 2 adversaries. We use a hybrid argument in each case.

Type 1 adversaries ($t \notin S$). We consider the following sequence of hybrid experiments:

- Hyb_0: Same as Real_0, except that we switch the somewhere-extractable hash setup to binding mode at the target index. We change how the parameters are computed:
  - The challenger samples $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$.

  This hybrid is indistinguishable by the mode indistinguishability of Π_{SEH}.
- Hyb_1: Same as Hyb_0, except that the challenge ciphertext is generated at threshold $t - 1$:
  - The challenger samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t-1, \mu)$ and sends ct to $\mathcal{A}$.

  This hybrid is indistinguishable by the index indistinguishability of Π_{PWE}. To see that the Π_{PWE} challenger does not abort at index t, recall that for a Type 1 adversary $t \notin S$, so $\mathbf{1}_S[t] = 0$. By correctness of Π_{SEH}, the honest opening π_t satisfies $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, t, 0, \pi_t) = 1$, so by the somewhere extraction property $\mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}) = 0$. Consequently, no opening π satisfies $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, t, 1, \pi) = 1$, and therefore no witness (σ, π) satisfies $C((\sigma, \pi), t) = 1$; that is, the circuit C has no witness at index t.
- Real_1: Same as Hyb_1, except that we switch the somewhere-extractable hash setup back to normal mode:
  - The challenger samples $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.

  This hybrid is indistinguishable by the mode indistinguishability of Π_{SEH}.

We now formalize the three transitions for Type 1 adversaries.

Claim B.10. *Assume that* Π_{SEH} *satisfies mode indistinguishability. Then for any efficient Type 1 adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Fix an efficient Type 1 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right|$. The only difference between Real_0 and Hyb_0 is how the somewhere-extractable hash key is sampled: via $\mathsf{SEH.Setup}(1^\lambda)$ in Real_0 and via $\mathsf{SEH.SetupBinding}(1^\lambda, t)$ in Hyb_0; in both, the challenge ciphertext is generated at threshold t. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π_{SEH} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain the number of users 1^N and the index $t \in [N]$, and sends t to the mode indistinguishability challenger of Π_{SEH}, which replies with a hash key hk.
2. Algorithm $\mathcal{B}$ samples an all-but-one signature key $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$, computes the signature $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$ for each $i \in [N]$, and sends $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ to $\mathcal{A}$.
3. On each corruption query $i \in [N]$ (pre- or post-challenge), algorithm $\mathcal{B}$ responds with σ_i.
4. When $\mathcal{A}$ outputs a set $S \subseteq [N]$ (with $t \notin S$) and a message μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$, constructs $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ from Fig. 12, samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, and sends ct to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$, and since $\mathcal{B}$ holds $\mathsf{sk}_{\mathsf{ABO}}$ it answers every corruption query. If the challenger samples $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$ (the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Real}_0(\mathcal{A})$; if it samples $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$ (the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$. Thus, algorithm $\mathcal{B}$ breaks mode indistinguishability of Π_{SEH} with the same advantage ε. □

Claim B.11. *Assume that* Π_{SEH} *satisfies somewhere extraction and* Π_{PWE} *satisfies index indistinguishability. Then for any efficient Type 1 adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Fix an efficient Type 1 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. The only difference between Hyb_0 and Hyb_1 is the threshold of the challenge ciphertext: t in Hyb_0 and $t - 1$ in Hyb_1; both sample the binding somewhere-extractable hash key $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$. We construct an algorithm $\mathcal{B}$ that breaks the index indistinguishability of Π_{PWE} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain 1^N and the index $t \in [N]$.
2. Algorithm $\mathcal{B}$ sends $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ to $\mathcal{A}$ where $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$ and $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$.
3. Algorithm $\mathcal{B}$ computes $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$ for each $i \in [N]$.
4. On each corruption query $i \in [N]$, algorithm $\mathcal{B}$ responds with σ_i.
5. When $\mathcal{A}$ outputs $S \subseteq [N]$ (with $t \notin S$) and μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$, constructs $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ from Fig. 12, and sends $(1^N, C, t, \mu)$ to the index indistinguishability challenger of Π_{PWE}; the challenger replies with a ciphertext ct, which $\mathcal{B}$ forwards to $\mathcal{A}$.
6. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

We verify that the Π_{PWE} challenger does not abort (i.e., that C has no witness at index t). Since $\mathcal{A}$ is Type 1, $t \notin S$, so $\mathbf{1}_S[t] = 0$. By correctness of Π_{SEH}, the honest opening satisfies $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, t, 0, \pi_t) = 1$ and since $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}})$ is in the support of $\mathsf{SEH.SetupBinding}(1^\lambda, t)$, the somewhere extraction property gives $\mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}) = 0$. Hence no opening π can satisfy $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, t, 1, \pi) = 1$, as this would force $\mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}) = 1$; the second check of C therefore fails for every witness at index t, so $C((\sigma, \pi), t) = 0$ for all (σ, π). Consequently, the challenger does not abort, and $\mathcal{B}$ perfectly simulates. If the challenger encrypts at cutoff t (the bit $b = 0$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$, and if it encrypts at cutoff $t - 1$ (the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Therefore $\mathcal{B}$ breaks index indistinguishability of Π_{PWE} with the same advantage ε. □

Claim B.12. *Assume that Π_{SEH} satisfies mode indistinguishability. Then for any efficient Type 1 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. The proof is completely analogous to that of Claim B.10, with the challenge ciphertext generated at threshold $t - 1$ in both experiments. □

Combining Claims B.10 to B.12, for every efficient Type 1 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda)$.

Type 2 adversaries ($t \in S$, hence $t \notin Q$). We consider the following sequence of hybrid experiments:

- Hyb_0: Same as Real_0, except that we switch the all-but-one signature setup to punctured mode at the target index. We change how the parameters are computed:
  - The challenger samples $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.SetupPunc}(1^\lambda, t)$.

  This hybrid is indistinguishable by the verification key indistinguishability of Π_{ABO}. The reduction is admissible because a Type 2 adversary never corrupts user t (i.e., $t \notin Q$), so the challenger only ever produces signatures σ_i for $i \neq t$, respecting the restriction of the verification key indistinguishability game that no signing query be made on the punctured message t.
- Hyb_1: Same as Hyb_0, except that the challenge ciphertext is generated at threshold $t - 1$:
  - The challenger samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t - 1, \mu)$ and sends ct to $\mathcal{A}$.

  This hybrid is indistinguishable by the index indistinguishability of Π_{PWE}. To see that the reduction is admissible, recall that under the punctured verification key $\mathsf{vk}_{\mathsf{ABO}}$, by punctured correctness of Π_{ABO} we have $\mathsf{ABO.Verify}(\mathsf{vk}_{\mathsf{ABO}}, t, \sigma) = 0$ for *every* signature σ. Hence no witness (σ, π) satisfies $C((\sigma, \pi), t) = 1$, so the circuit C has no witness at index t and the Π_{PWE} challenger does not abort.
- Real_1: Same as Hyb_1, except that we switch the all-but-one signature setup back to normal mode:
  - The challenger samples $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$.

  This hybrid is indistinguishable by the verification key indistinguishability of Π_{ABO}.

We now formalize the three transitions for Type 2 adversaries.

Claim B.13. *Assume that Π_{ABO} satisfies verification key indistinguishability. Then for any efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Fix an efficient Type 2 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right|$. The only difference between Real_0 and Hyb_0 is how the all-but-one signature keys are sampled: via $\mathsf{ABO.Setup}(1^\lambda)$ in Real_0 and via $\mathsf{ABO.SetupPunc}(1^\lambda, t)$ in Hyb_0. In both, the challenge ciphertext is generated at threshold t. We construct an algorithm $\mathcal{B}$ that breaks the verification key indistinguishability of Π_{ABO} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain 1^N and the index $t \in [N]$, and sends the punctured message $\mu^* = t$ to the verification key indistinguishability challenger of Π_{ABO}, which replies with a verification key $\mathsf{vk}_{\mathsf{ABO}}$.

2. Algorithm $\mathcal{B}$ samples a somewhere-extractable hash key $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$ and sends $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ to $\mathcal{A}$.

3. On each corruption query $i \in [N]$ (pre- or post-challenge), which necessarily satisfies $i \neq t$ since $\mathcal{A}$ is Type 2, algorithm $\mathcal{B}$ obtains σ_i by making a signing query on i to the Π_{ABO} challenger (caching the response so that repeated queries on i are answered consistently) and responds with σ_i.

4. When $\mathcal{A}$ outputs a set $S \subseteq [N]$ (with $t \in S$) and a message μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$, constructs $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ from Fig. 12, samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, and sends ct to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

Every signing query $\mathcal{B}$ makes is on an index $i \neq t = \mu^*$, so $\mathcal{B}$ is admissible in the verification key indistinguishability game; if $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger samples $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.Setup}(1^\lambda)$ (the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Real}_0(\mathcal{A})$; if it samples $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.SetupPunc}(1^\lambda, t)$ (the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$. Therefore, algorithm $\mathcal{B}$ breaks the verification key indistinguishability of Π_{ABO} with the same advantage ε. □

Claim B.14. *Assume that Π_{ABO} satisfies punctured correctness and Π_{PWE} satisfies index indistinguishability. Then for any efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Fix an efficient Type 2 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. The only difference between Hyb_0 and Hyb_1 is the threshold of the challenge ciphertext: t in Hyb_0 and $t-1$ in Hyb_1. Both sample the punctured all-but-one signature keys $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.SetupPunc}(1^\lambda, t)$. We construct an algorithm $\mathcal{B}$ that breaks the index indistinguishability of Π_{PWE} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain 1^N and the index $t \in [N]$.

2. Algorithm $\mathcal{B}$ samples the punctured all-but-one signature keys $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}}) \leftarrow \mathsf{ABO.SetupPunc}(1^\lambda, t)$, a somewhere-extractable hash key $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$, and $\sigma_i \leftarrow \mathsf{ABO.Sign}(\mathsf{sk}_{\mathsf{ABO}}, i)$ for each $i \neq t$, and sends $\mathsf{mpk} = (1^\lambda, N, \mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk})$ to $\mathcal{A}$.

3. On each corruption query $i \in [N]$ (necessarily $i \neq t$ since $\mathcal{A}$ is Type 2), algorithm $\mathcal{B}$ responds with σ_i.

4. When $\mathcal{A}$ outputs $S \subseteq [N]$ and μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \pi_1, \ldots, \pi_N) \leftarrow \mathsf{SEH.Hash}(\mathsf{hk}, \mathbf{1}_S)$, constructs $C = C[\mathsf{vk}_{\mathsf{ABO}}, \mathsf{hk}, \mathsf{dig}]$ from Fig. 12, and sends $(1^N, C, t, \mu)$ to the index indistinguishability challenger of Π_{PWE}; the challenger replies with a ciphertext ct, which $\mathcal{B}$ forwards to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

We verify that the Π_{PWE} challenger does not abort (i.e., that C has no witness at index t). Since $(\mathsf{sk}_{\mathsf{ABO}}, \mathsf{vk}_{\mathsf{ABO}})$ is in the support of $\mathsf{ABO.SetupPunc}(1^\lambda, t)$, punctured correctness gives $\mathsf{ABO.Verify}(\mathsf{vk}_{\mathsf{ABO}}, t, \sigma) = 0$ for every signature σ; hence the first check of C fails for every witness at index t, so $C((\sigma, \pi), t) = 0$ for all (σ, π) and the challenger does not abort. Consequently $\mathcal{B}$ perfectly simulates: if the challenger encrypts at cutoff t (the bit $b = 0$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$, and if it encrypts at cutoff $t-1$ (the bit $b = 1$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Thus, algorithm $\mathcal{B}$ breaks the index indistinguishability of Π_{PWE} with the same advantage ε. □

Claim B.15. *Assume that Π_{ABO} satisfies verification key indistinguishability. Then for any efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. The proof is completely analogous to that of Claim B.13, with the challenge ciphertext generated at threshold $t - 1$ in both experiments. □

Combining Claims B.13 to B.15, for every efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda)$.

Completing the proof. In both cases, Real_0 and Real_1 exactly coincide with the index indistinguishability games of Construction B.3 with bits $b = 0$ and $b = 1$, respectively, while the intermediate distributions are computationally indistinguishable. Since every admissible adversary is either of Type 1 or Type 2, the theorem follows. □

C Registered Augmented Broadcast Encryption

Appendix B considers the traditional setting of centralized (augmented) broadcast encryption, where a trusted authority is responsible for generating secret keys for each user. In this section, we consider a registered variant where users choose their own public/secret keys, and there is a public, deterministic aggregation algorithm that takes the public keys of the users and aggregates them into the master public key for the augmented broadcast encryption scheme. Using the master public key, one can encrypt to an arbitrary subset of the registered users (together with a cutoff t). This is the analog of registration-based encryption [GHMR18] applied to the setting of augmented broadcast encryption.

Registered augmented broadcast encryption is similar to notions like distributed broadcast encryption [WQZD10, BZ14] and flexible broadcast encryption [FWW23]. Both notions generalize broadcast encryption to the setting where individual users choose their own public keys. The main difference in the registration-based setting is that we require an explicit aggregation algorithm that aggregates the public keys for a set of users into a succinct master public key that can be used for encryption. In contrast, with distributed or flexible broadcast encryption, the encryption algorithm itself takes all of the individual public keys as input. In this section, we show how to adapt the flexible broadcast encryption scheme from [FWW23] to obtain a registered augmented broadcast encryption scheme from indexed positional witness encryption. In Appendix D, we show how to use registered augmented broadcast encryption to obtain a registered broadcast-and-trace scheme with public tracing.

C.1 Definition of Registered Augmented Broadcast Encryption

We now define registered augmented broadcast encryption. In this setting, users sample their own public and private keys. Then, there is a deterministic aggregation algorithm that takes as input N public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ and outputs a master public key mpk together with N helper decryption keys. A message can be encrypted to a subset $I \subset [N]$ and a cutoff $t \in [0, N]$ such that only users in $I \cap [t + 1, N]$ can decrypt. We require message hiding and index indistinguishability, which are analogous to the centralized version of Definition B.1.

Definition C.1 (Registered Augmented Broadcast Encryption)**.** A registered augmented broadcast encryption scheme over a message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{RegAugBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: On input a security parameter λ, the setup algorithm outputs a common reference string crs.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$: On input the common reference string crs, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{Aggregate}\left(\mathsf{crs}, \left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N\right)\right) \to (\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N))$: On input the common reference string crs and a list of public keys $\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N\right)$, the aggregation algorithm outputs a master public key mpk and a list of helper decryption keys $(\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)$. Assume that mpk and each hsk_i implicitly contain the number of aggregated keys N. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, t) \to \mathsf{ct}$: On input the common reference string crs, a master public key mpk, a message $\mu \in \mathcal{M}$, a subset of indices $I \subseteq [N]$, and an index $t \in \{0, 1, \ldots, N\}$, the encryption algorithm outputs a ciphertext ct.

- $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}, \mathsf{hsk}, I, \mathsf{ct}) \to \mu$: On input the common reference string crs, a secret key sk, a helper decryption key hsk, a subset of indices $I \subseteq [N]$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}$ or the special symbol $\perp$. This algorithm is deterministic.

The Π_{RegAugBE} scheme should satisfy the following correctness and succinctness properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, all $N \in [2^\lambda]$, all messages $\mu \in \mathcal{M}$, all subsets $I \subseteq [N]$, all indices $t \in \{0, 1, \ldots, N\}$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for $i \in [N]$, the master public key and helper decryption keys $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\big)$, and all ct in the support of $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, t)$, the following holds for all $j \in I \cap \{t+1, t+2, \ldots, N\}$:

$$\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}_j, \mathsf{hsk}_j, I, \mathsf{ct}) = \mu.$$

- **Succinctness:** We say that the registered augmented broadcast encryption scheme is $\eta(N)$-succinct if there exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, all $N \in [2^\lambda]$, all messages $\mu \in \mathcal{M}$, all subsets $I \subseteq [N]$, all indices $t \in \{0, 1, \ldots, N\}$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}_j, \mathsf{sk}_j)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for $j \in [N]$, the master public key and helper decryption keys $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\big)$, and all ct in the support of $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, t)$, the following holds:
  - **Succinct master public key:** $|\mathsf{mpk}| \le \mathsf{poly}(\lambda)$.
  - **Succinct helper decryption keys:** $\big|\mathsf{hsk}_j\big| \le \mathsf{poly}(\lambda)$ for all $j \in [N]$.
  - $\eta(N)$**-succinct ciphertexts:** $|\mathsf{ct}| \le \eta(N) \cdot \mathsf{poly}(\lambda)$.

Security games. In the two security games below, the challenger maintains a list K of users in the system, and an index ind. Initially, K is empty and $\mathsf{ind} = 0$. During a *query phase*, the adversary may adaptively and repeatedly issue the following three kinds of queries:

- **Key generation query:** On a key-generation query, the challenger increments $\mathsf{ind} \leftarrow \mathsf{ind} + 1$, samples $(\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, sets $K[\mathsf{ind}] \leftarrow (\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}, \mathsf{honest})$, and returns $\mathsf{pk}_{\mathsf{ind}}$ to the adversary.
- **Corruption query:** On a corruption query, the adversary specifies an index $i \le \mathsf{ind}$. The challenger retrieves $(\mathsf{pk}_i, \mathsf{sk}_i, \cdot) \leftarrow K[i]$, overwrites $K[i] \leftarrow (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{corrupt})$ and sends sk_i to the adversary.
- **Malicious key registration query:** On a malicious key registration query, the adversary sends a public key pk. The challenger increments $\mathsf{ind} \leftarrow \mathsf{ind} + 1$, and sets $K[\mathsf{ind}] \leftarrow (\mathsf{pk}, \perp, \mathsf{corrupt})$.

The Π_{RegAugBE} scheme should satisfy the following security properties:

- **Message hiding:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the message hiding game as follows:
  1. On input the security parameter 1^λ, the challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{A}$.
  2. The adversary adaptively issues key generation, corruption, and malicious key registration queries.
  3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
  4. The challenger retrieves the entry $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, and computes $(\mathsf{mpk}, \cdot) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I, N)$, and sends ct to $\mathcal{A}$.
  5. The adversary adaptively issues corruption queries.
  6. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that the registered augmented broadcast encryption scheme satisfies message hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Index indistinguishability:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the index indistinguishability game as follows:

  1. On input the security parameter 1^λ, the challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{A}$.
  2. The adversary adaptively issues key generation, corruption, and malicious key registration queries.
  3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, a challenge index $t \in [N]$, and a message $\mu \in \mathcal{M}$.
  4. The challenger retrieves the entry $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, and computes $(\mathsf{mpk}, \cdot) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, t-b)$, and sends ct to $\mathcal{A}$.
  5. The adversary adaptively issues corruption queries.
  6. Let $i^* = i_t$ be the index at position t in the challenge list, and let $\mathsf{pk}^* = \mathsf{pk}_{i^*}$ be its public key, so that $K[i^*] = (\mathsf{pk}^*, \cdot, \cdot)$. If $t \in I$ and $K[i^*]$ is marked as corrupt, then the experiment aborts and outputs $b' = 0$.
  7. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the registered augmented broadcast encryption scheme satisfies index indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Remark C.2 (Selective Index Indistinguishability)**.** We can define a selective version of index indistinguishability in which, during setup, the adversary commits to both the target player index i^* and the challenge position $t \in [N]$ at which it will place i^* in the challenge list (so that $i_t = i^*$). This notion is equivalent to the (adaptive) index indistinguishability security defined above: a reduction guesses the pair (i^*, t) in advance and aborts with output 0 on a wrong guess. Since there are only polynomially many indices, the guess is correct with probability $1/\mathsf{poly}(\lambda)$, so the security loss is polynomial. As such, when building this primitive, we focus on achieving selective index indistinguishability.

Bit commitments. Our construction additionally relies on a (perfectly binding) bit commitment scheme in the common reference string model. We give the definition below.

Definition C.3 (Perfectly Binding Bit Commitment)**.** A perfectly binding bit commitment scheme is a tuple of efficient algorithms $\Pi_{\mathsf{Com}} = (\mathsf{Com.Setup}, \mathsf{Com.Commit})$ with the following syntax:

- $\mathsf{Com.Setup}(1^\lambda) \to \mathsf{ck}$: On input a security parameter λ, the setup algorithm outputs a commitment key ck.
- $\mathsf{Com.Commit}(\mathsf{ck}, b; r) \to c$: On input a commitment key ck, a bit $b \in \{0,1\}$, and randomness $r \in \{0,1\}^\rho$, the commitment algorithm outputs a commitment $c \in \{0,1\}^{\ell_c}$, where $\rho = \rho(\lambda)$ and $\ell_c = \ell_c(\lambda)$ are fixed polynomials. This algorithm is deterministic.

The bit commitment scheme Π_{Com} should satisfy the following properties:

- **Perfect binding:** For all $\lambda \in \mathbb{N}$ and all ck in the support of $\mathsf{Com.Setup}(1^\lambda)$, there is no commitment $c \in \{0,1\}^{\ell_c}$ together with randomness $r_0, r_1 \in \{0,1\}^\rho$ such that $\mathsf{Com.Commit}(\mathsf{ck}, 0; r_0) = \mathsf{Com.Commit}(\mathsf{ck}, 1; r_1) = c$.
- **Computational hiding:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the hiding game as follows:

  1. On input the security parameter 1^λ, the challenger samples $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$ and $r \xleftarrow{R} \{0,1\}^\rho$, computes $c = \mathsf{Com.Commit}(\mathsf{ck}, b; r)$, and sends (ck, c) to $\mathcal{A}$.
  2. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the Π_{Com} scheme satisfies computational hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

C.2 Construction of Registered Augmented Broadcast Encryption

We construct a registered augmented broadcast encryption scheme from an indexed positional witness encryption scheme, a somewhere-extractable hash function, and a perfectly binding bit commitment scheme. The construction closely follows the structure of the flexible broadcast encryption scheme from Freitag et al. [FWW23]. The main difference is that we introduce a threshold t (by substituting an indexed positional witness encryption scheme in place of the vanilla witness encryption scheme). We recall the basic structure of the scheme:

- A user's public key is a commitment to the bit 1, and its secret key is the commitment randomness.
- The master public key is a somewhere-extractable hash digest of the public keys of the set of users, and the helper decryption keys are the corresponding local openings. At aggregation time, we associate each public key with an index $i \in [N]$, where N is the total number of public keys being aggregated.
- To encrypt to a subset $I \subseteq [N]$ of the aggregated users, the encrypter hashes the indicator string of I under a second somewhere-extractable hash whose key is included in the common reference string.
- Encryption is a positional witness encryption to the circuit that checks, at index i, that the digest opens to a commitment to 1 whose randomness the decryptor knows and that the indicator digest opens to 1 at position i (i.e., that $i \in I$), and that the index is above the cutoff (i.e., that $i > t$).

We now give the formal construction:

Construction C.4 (Registered Augmented Broadcast Encryption)**.** Our construction relies on the following primitives:

- A perfectly binding bit commitment scheme $\Pi_{\mathsf{Com}} = (\mathsf{Com.Setup}, \mathsf{Com.Commit})$, with randomness length $\rho = \rho(\lambda)$ and commitment length $\ell_c = \ell_c(\lambda)$.
- A somewhere-extractable hash $\Pi_{\mathsf{SEH}} = (\mathsf{SEH.Setup}, \mathsf{SEH.SetupBinding}, \mathsf{SEH.Hash}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract})$ as per Definition A.1 for blocks $\mathcal{X}_\lambda = \{0,1\}^{\ell_c}$, and another hash Π'_{SEH} for blocks $\mathcal{X}'_\lambda = \{0,1\}$.
- An indexed positional witness encryption scheme $\Pi_{\mathsf{PWE}} = (\mathsf{PWE.Enc}, \mathsf{PWE.Dec})$.

We construct the scheme $\Pi_{\mathsf{RegAugBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$, over the message space $\mathcal{M}$ of Π_{PWE}, as follows:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, the setup algorithm does the following:
  1. Sample somewhere-extractable hash keys $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$ and $\mathsf{hk}' \leftarrow \mathsf{SEH'.Setup}(1^\lambda)$, and a commitment key $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$.
  2. Output the common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$.
- $\mathsf{KeyGen}(\mathsf{crs})$: On input a common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$, the key-generation algorithm does the following:
  1. Sample randomness $r \xleftarrow{R} \{0,1\}^\rho$ and compute the commitment $c = \mathsf{Com.Commit}(\mathsf{ck}, 1; r)$.
  2. Output the public key $\mathsf{pk} = c$ and the secret key $\mathsf{sk} = r$.
- $\mathsf{Aggregate}\left(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\right)$: On input a common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ and a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, the aggregation algorithm does the following:
  1. Compute $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH.Hash}\left(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\right)$.
  2. Output the master public key $\mathsf{mpk} = (\mathsf{dig}, N)$ and, for each $j \in [N]$, the helper decryption key $\mathsf{hsk}_j = (j, \sigma_j)$.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, t)$: On input a common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$, a master public key $\mathsf{mpk} = (\mathsf{dig}, N)$, a message $\mu \in \mathcal{M}$, a subset of indices $I \subseteq [N]$, and an index $t \in \{0, 1, \ldots, N\}$, the encryption algorithm does the following:

1. Hash the indicator string of the set $(\mathsf{dig}', \cdot) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$.
2. Construct the circuit $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ described in Fig. 13 below:

> **Constants:** Somewhere-extractable hash keys hk and hk', a commitment key ck, and digests dig and dig'.
> **Inputs:** A commitment c, hash openings (σ, σ'), a commitment randomness r, and an index $i \in [N]$.
>
> 1. Check $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, i, c, \sigma) = 1$.
> 2. Check $\mathsf{SEH}'.\mathsf{Verify}(\mathsf{hk}', \mathsf{dig}', i, 1, \sigma') = 1$.
> 3. Check $c = \mathsf{Com.Commit}(\mathsf{ck}, 1; r)$.
> 4. If all checks pass, output 1. Otherwise, output 0.

Figure 13: Circuit $C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$.

3. Compute $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$.
4. Output the ciphertext ct.

- $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}, \mathsf{hsk}, I, \mathsf{ct})$: On input a common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$, a secret key $\mathsf{sk} = r$, a helper decryption key $\mathsf{hsk} = (i, \sigma)$, a subset of indices $I \subseteq [N]$, and a ciphertext ct, the decryption algorithm does the following:

  1. Recompute the commitment $c = \mathsf{Com.Commit}(\mathsf{ck}, 1; r)$.
  2. Compute openings $\big(\cdot, (\sigma'_1, \ldots, \sigma'_N)\big) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$.
  3. Output $\mathsf{PWE.Dec}(\mathsf{ct}, (c, \sigma, \sigma'_i, r), i)$.

Theorem C.5 (Correctness). *If Π_{SEH} and Π'_{SEH} satisfy opening correctness and Π_{PWE} is correct, then Construction C.4 is correct.*

Proof. Fix $\lambda \in \mathbb{N}$, $N \in [2^\lambda]$, a message $\mu \in \mathcal{M}$, a subset $I \subseteq [N]$, an index $t \in \{0, 1, \ldots, N\}$, and an index $i \in I \cap \{t+1, t+2, \ldots, N\}$; note that $i \in I$ and $i > t$. Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ be in the support of $\mathsf{Setup}(1^\lambda)$, let $(\mathsf{pk}_j, \mathsf{sk}_j)$ be in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for each $j \in [N]$ (so $\mathsf{sk}_j = r_j$ and $\mathsf{pk}_j = \mathsf{Com.Commit}(\mathsf{ck}, 1; r_j)$), let $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\big)$, and let ct be in the support of $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, t)$. By construction, $\mathsf{mpk} = (\mathsf{dig}, N)$ and $\mathsf{hsk}_i = (i, \sigma_i)$, where $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH.Hash}\big(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\big)$. Moreover, $(\mathsf{dig}', \cdot) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$ and $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$ for the circuit $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ of Fig. 13.

The decryptor parses $\mathsf{hsk}_i = (i, \sigma_i)$ and $\mathsf{sk}_i = r_i$. Then, it recomputes $c = \mathsf{Com.Commit}(\mathsf{ck}, 1; r_i)$, which equals pk_i. Next, it recomputes the openings $\big(\cdot, (\sigma'_1, \ldots, \sigma'_N)\big) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, which, since $\mathsf{SEH}'.\mathsf{Hash}$ is deterministic, are openings of the same digest dig' hard-coded in C. Finally, it outputs $\mathsf{PWE.Dec}(\mathsf{ct}, (c, \sigma_i, \sigma'_i, r_i), i)$. We claim that $(c, \sigma_i, \sigma'_i, r_i)$ is an accepting witness for index i (i.e., $C\big((c, \sigma_i, \sigma'_i, r_i), i\big) = 1$):

- The i^{th} block of $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ equals $\mathsf{pk}_i = c$. By opening correctness of Π_{SEH}, the honest opening σ_i of position i verifies, so $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, i, c, \sigma_i) = 1$.
- Since $i \in I$, the i^{th} bit of $\mathbf{1}_I$ equals 1. By opening correctness of Π'_{SEH}, the honest opening σ'_i of position i verifies, so $\mathsf{SEH}'.\mathsf{Verify}(\mathsf{hk}', \mathsf{dig}', i, 1, \sigma'_i) = 1$.
- By construction $c = \mathsf{Com.Commit}(\mathsf{ck}, 1; r_i)$.

By construction of the circuit in Fig. 13, all checks pass, so $C\big((c, \sigma_i, \sigma'_i, r_i), i\big) = 1$. Moreover, $i > t$. By correctness of Π_{PWE}, since this witness is accepting and $i > t$, we have $\mathsf{PWE.Dec}(\mathsf{ct}, (c, \sigma_i, \sigma'_i, r_i), i) = \mu$. Therefore $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}_i, \mathsf{hsk}_i, I, \mathsf{ct}) = \mu$, as required. □

Theorem C.6 (Succinctness). *Suppose Π_{SEH} and Π'_{SEH} have succinct hash keys, digests, and openings, the commitment scheme has commitments and randomness of length $\mathsf{poly}(\lambda)$, and Π_{PWE} is $\eta(N)$-succinct. Then Construction C.4 is $\eta(N)$-succinct for messages of length $\mathsf{poly}(\lambda)$.*

Proof. Throughout, recall that the commitment length is $\ell_c = \mathrm{poly}(\lambda)$, that SEH.Hash is applied to the N blocks $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, each in $\{0,1\}^{\ell_c}$, that SEH′.Hash is applied to the N-bit indicator string $\mathbf{1}_I$, and that $N \le 2^\lambda$, so $\log N \le \lambda$ and hence $\mathrm{poly}(\lambda, \log N) = \mathrm{poly}(\lambda)$. The common reference string is $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$. By the succinctness of Π_{SEH} and Π'_{SEH}, $|\mathsf{hk}|, |\mathsf{hk}'| = \mathrm{poly}(\lambda, \log N) = \mathrm{poly}(\lambda)$, and $|\mathsf{ck}| = \mathrm{poly}(\lambda)$. Each public key $\mathsf{pk}_i = c \in \{0,1\}^{\ell_c}$ has $|\mathsf{pk}_i| = \mathrm{poly}(\lambda)$, and each secret key $\mathsf{sk}_i = r \in \{0,1\}^\rho$ has $|\mathsf{sk}_i| = \mathrm{poly}(\lambda)$. The master public key $\mathsf{mpk} = (\mathsf{dig}, N)$ consists of the digest dig and the recipient count N, so $|\mathsf{mpk}| = \mathrm{poly}(\lambda, \log N) = \mathrm{poly}(\lambda)$, and each helper decryption key $\mathsf{hsk}_j = (j, \sigma_j)$ consists of an index $j \in [N]$ and a single somewhere-extractable hash opening of size $\mathrm{poly}(\lambda, \log N)$, so $|\mathsf{hsk}_j| = \mathrm{poly}(\lambda)$. Finally, the ciphertext is $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, where the digest dig' of the N-bit string $\mathbf{1}_I$ satisfies $|\mathsf{dig}'| = \mathrm{poly}(\lambda, \log N) = \mathrm{poly}(\lambda)$ by the succinctness of Π'_{SEH}. The circuit $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ performs one invocation each of SEH.Verify, SEH′.Verify, and Com.Commit, and hard-codes $\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}'$; since these have size $\mathrm{poly}(\lambda)$ and SEH.Verify, SEH′.Verify, Com.Commit are computable by circuits of size $\mathrm{poly}(\lambda, \log N) = \mathrm{poly}(\lambda)$, its size is $|C| = \mathrm{poly}(\lambda)$. By the $\eta(N)$-succinctness of Π_{PWE}, $|\mathsf{ct}| \le \eta(N) \cdot \mathrm{poly}(\lambda, |C|, |\mu|, \log N) = \eta(N) \cdot \mathrm{poly}(\lambda, |\mu|)$, which is $\eta(N) \cdot \mathrm{poly}(\lambda)$ for messages of length $\mathrm{poly}(\lambda)$. □

Theorem C.7 (Message Hiding). *Assume Π_{PWE} satisfies message hiding. Then Construction C.4 satisfies message hiding.*

Proof. Fix any efficient adversary $\mathcal{A}$, and let ε be its advantage in the message hiding game of Construction C.4. We construct an algorithm $\mathcal{B}$ for the message hiding game of Π_{PWE} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ, samples somewhere-extractable hash keys $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$ and $\mathsf{hk}' \leftarrow \mathsf{SEH'.Setup}(1^\lambda)$, and a commitment key $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends the common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.

2. Algorithm $\mathcal{B}$ answers $\mathcal{A}$'s pre-challenge queries itself, exactly as the challenger of Definition C.1 would:

   - **Key generation query:** On a key-generation query, algorithm $\mathcal{B}$ increments $\mathsf{ind} \leftarrow \mathsf{ind} + 1$, samples $(\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, sets $K[\mathsf{ind}] \leftarrow (\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}, \mathsf{honest})$, and returns $\mathsf{pk}_{\mathsf{ind}}$.
   - **Corruption query:** On a corruption query on index i, algorithm $\mathcal{B}$ retrieves $(\mathsf{pk}_i, \mathsf{sk}_i, \cdot) \leftarrow K[i]$, overwrites $K[i] \leftarrow (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{corrupt})$, and returns sk_i.
   - **Malicious key registration:** On a malicious key registration query on the public key pk, algorithm $\mathcal{B}$ increments $\mathsf{ind} \leftarrow \mathsf{ind} + 1$ and sets $K[\mathsf{ind}] \leftarrow (\mathsf{pk}, \perp, \mathsf{corrupt})$.

3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.

4. Algorithm $\mathcal{B}$ retrieves the entries $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, and computes the master public key and helper decryption keys $\big(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)\big) = \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$ with $\mathsf{mpk} = (\mathsf{dig}, N)$, computes the indicator digest and openings $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH'.Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs the circuit $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, and sends 1^N, the circuit C, and the messages μ_0, μ_1 to the Π_{PWE} challenger.

5. The challenger sends a ciphertext ct, which $\mathcal{B}$ forwards to $\mathcal{A}$.

6. Algorithm $\mathcal{B}$ continues to answer $\mathcal{A}$'s post-challenge corruption queries exactly as in the pre-challenge phase.

7. Algorithm $\mathcal{A}$ outputs a guess b', which $\mathcal{B}$ forwards as its own output.

First note that if $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. Next, for any $b \in \{0,1\}$, if the Π_{PWE} challenger samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, N, \mu_b)$, then $\mathcal{B}$ perfectly simulates the message hiding game of Construction C.4: it answers every query exactly as the challenger would, and the ciphertext ct is distributed exactly as $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I, N)$, the encryption of μ_b at cutoff N. Finally, algorithm $\mathcal{B}$ outputs the same guess as $\mathcal{A}$; we conclude that $\mathcal{B}$ has advantage exactly ε in breaking the message hiding security of Π_{PWE}, and the claim follows. □

Theorem C.8 (Index Indistinguishability). *Assume Π_{Com} satisfies computational hiding and perfect binding, Π_{SEH} and Π'_{SEH} satisfy mode indistinguishability and somewhere extraction, and Π_{PWE} satisfies index indistinguishability. Then Construction C.4 satisfies selective index indistinguishability as defined in Remark C.2.*

Proof. Fix any efficient adversary $\mathcal{A}$ for the selective game. Recall that $\mathcal{A}$ commits at setup to a target player i^* together with the position t at which it will place pk_{i^*} in the challenge list. By admissibility, the experiment outputs 0 whenever $t \in I$ and the player i^* at position t is corrupted. Hence in every execution that contributes to $\mathcal{A}$'s advantage we have either that $t \notin I$ or that i^* is honest (i.e., it was produced by a key generation query and never corrupted, so $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 1; r^*)$ for randomness r^* that is never revealed to $\mathcal{A}$). Similar to the proof of Theorem B.9, we classify $\mathcal{A}$ into two types and bound the advantage of each type of adversarial strategy:

- *Type 1* adversaries always output a challenge with $t \notin I$.
- *Type 2* adversaries always output $t \in I$ (and hence, by admissibility, never corrupt i^*).

We begin from the following description of the real game with bit $b = 0$, which is common to both cases.

- Real_0: This is the selective index indistinguishability game where the challenger plays with bit $b = 0$:
  1. On input 1^λ, the adversary commits to a target player i^* and a position t. The challenger samples $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$, $\mathsf{hk}' \leftarrow \mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$, and $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, initializes $K \leftarrow \emptyset$ and $\mathsf{ind} \leftarrow 0$, and sends $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.
  2. The challenger answers $\mathcal{A}$'s pre-challenge key generation, corruption, and malicious key registration queries as in Definition C.1.
  3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$ with $i_t = i^*$, a subset $I \subseteq [N]$, and a message $\mu \in \mathcal{M}$.
  4. The challenger retrieves the entries $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, computes a hash of the public keys $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH.Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$, computes a hash of the set $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, and sends ct to $\mathcal{A}$.
  5. The challenger continues to answer $\mathcal{A}$'s post-challenge corruption queries.
  6. If $t \in I$ and the player i^* at position t is corrupted, the experiment outputs 0 (this never occurs for admissible adversaries).
  7. The adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

We now consider the analysis for Type 1 adversaries and for Type 2 adversaries. We use a hybrid argument in each case.

Type 1 adversaries ($t \notin I$). We consider the following sequence of hybrid experiments:

- Hyb_0: Same as Real_0, except that the indicator hash key is set up in binding mode at position t:
  - The challenger samples the binding key $(\mathsf{hk}', \mathsf{td}_{\mathsf{SEH}'}) \leftarrow \mathsf{SEH}'.\mathsf{SetupBinding}(1^\lambda, t)$.

  This hybrid is indistinguishable from Real_0 by the mode indistinguishability of Π'_{SEH}. Since $t \notin I$, the t^{th} bit of $\mathbf{1}_I$ is 0; by correctness of Π'_{SEH} the honest opening satisfies $\mathsf{SEH}'.\mathsf{Verify}(\mathsf{hk}', \mathsf{dig}', t, 0, \cdot) = 1$, so by the somewhere extraction property $\mathsf{SEH}'.\mathsf{Extract}(\mathsf{td}_{\mathsf{SEH}'}, \mathsf{dig}') = 0$. Consequently no opening σ' satisfies $\mathsf{SEH}'.\mathsf{Verify}(\mathsf{hk}', \mathsf{dig}', t, 1, \sigma') = 1$, so the second check of C fails at index t and the circuit C has no accepting witness at index t.
- Hyb_1: Same as Hyb_0, except that the challenge ciphertext is generated at cutoff $t - 1$:
  - The challenger samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t - 1, \mu)$.

  This hybrid is indistinguishable from Hyb_0 by the index indistinguishability of Π_{PWE}; the Π_{PWE} challenger does not abort at index t because, as argued above, C has no accepting witness at index t.
- Real_1: Same as Hyb_1, except that the indicator hash key is set up in normal mode:
  - The challenger samples the standard key $\mathsf{hk}' \leftarrow \mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$.

This hybrid is indistinguishable from Hyb_1 by the mode indistinguishability of Π'_{SEH}. This is exactly the index indistinguishability game with bit $b = 1$, in which the challenge is encrypted at cutoff $t - 1$.

We now formalize the three transitions for Type 1 adversaries.

Claim C.9. *Assume that Π'_{SEH} satisfies mode indistinguishability. Then for any efficient Type 1 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.*

Proof. Fix an efficient Type 1 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right|$. The only difference between Real_0 and Hyb_0 is how the indicator hash key is sampled: via $\mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$ in Real_0 and via $\mathsf{SEH}'.\mathsf{SetupBinding}(1^\lambda, t)$ in Hyb_0; in both, the challenge ciphertext is generated at cutoff t. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π'_{SEH} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ, which commits to a target player i^* and a position t, and sends t to the mode indistinguishability challenger of Π'_{SEH}, which replies with a hash key hk'.

2. Algorithm $\mathcal{B}$ samples $\mathsf{hk} \leftarrow \mathsf{SEH}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{ck} \leftarrow \mathsf{Com}.\mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.

3. Throughout, algorithm $\mathcal{B}$ answers $\mathcal{A}$'s key generation, corruption, and malicious key registration queries exactly as the challenger of Definition C.1 would; it can do so since it holds ck and generates all key pairs itself.

4. When $\mathcal{A}$ outputs a list $(i_1, \ldots, i_N)$ with $i_t = i^*$, a subset $I \subseteq [N]$ with $t \notin I$, and a message μ, algorithm $\mathcal{B}$ retrieves the public keys $\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}$, computes $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH}.\mathsf{Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$ and $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, samples $\mathsf{ct} \leftarrow \mathsf{PWE}.\mathsf{Enc}(1^\lambda, 1^N, C, t, \mu)$, and sends ct to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger samples $\mathsf{hk}' \leftarrow \mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$ (the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Real}_0(\mathcal{A})$; if it samples $(\mathsf{hk}', \mathsf{td}_{\mathsf{SEH}'}) \leftarrow \mathsf{SEH}'.\mathsf{SetupBinding}(1^\lambda, t)$ (the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$. Therefore, algorithm $\mathcal{B}$ breaks the mode indistinguishability of Π'_{SEH} with the same advantage ε. □

Claim C.10. *Assume that Π'_{SEH} satisfies somewhere extraction and Π_{PWE} satisfies index indistinguishability. Then for any efficient Type 1 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Fix an efficient Type 1 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. The only difference between Hyb_0 and Hyb_1 is the cutoff of the challenge ciphertext: t in Hyb_0 and $t - 1$ in Hyb_1; both sample the binding indicator key $(\mathsf{hk}', \mathsf{td}_{\mathsf{SEH}'}) \leftarrow \mathsf{SEH}'.\mathsf{SetupBinding}(1^\lambda, t)$. We construct an algorithm $\mathcal{B}$ that breaks the index indistinguishability of Π_{PWE} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ in order to get i^* and t.

2. Algorithm $\mathcal{B}$ samples the hash key for public keys $\mathsf{hk} \leftarrow \mathsf{SEH}.\mathsf{Setup}(1^\lambda)$, the hash key for sets $(\mathsf{hk}', \mathsf{td}_{\mathsf{SEH}'}) \leftarrow \mathsf{SEH}'.\mathsf{SetupBinding}(1^\lambda, t)$, and commitment key $\mathsf{ck} \leftarrow \mathsf{Com}.\mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.

3. Algorithm $\mathcal{B}$ answers all queries exactly as the index indistinguishability challenger of Definition C.1.

4. When $\mathcal{A}$ outputs $(i_1, \ldots, i_N)$ with $i_t = i^*$, a subset $I \subseteq [N]$ with $t \notin I$, and a message μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH}.\mathsf{Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$ and $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, and sends $(1^N, C, t, \mu)$ to the index indistinguishability challenger of Π_{PWE}; the challenger replies with ct, which $\mathcal{B}$ forwards to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

We verify that the Π_{PWE} challenger does not abort (i.e., that C has no accepting witness at index t). Since $\mathcal{A}$ is Type 1, $t \notin I$, so the t^{th} bit of $\mathbf{1}_I$ is 0; by correctness of Π'_{SEH} the honest opening satisfies $\mathsf{SEH}'.\mathsf{Verify}(\mathsf{hk}', \mathsf{dig}', t, 0, \cdot) = 1$, and since $(\mathsf{hk}', \mathsf{td}_{\mathsf{SEH}'})$ is in the support of $\mathsf{SEH}'.\mathsf{SetupBinding}(1^\lambda, t)$, the somewhere extraction property gives $\mathsf{SEH}'.\mathsf{Extract}(\mathsf{td}_{\mathsf{SEH}'}, \mathsf{dig}') = 0$. Hence no opening σ' satisfies $\mathsf{SEH}'.\mathsf{Verify}(\mathsf{hk}', \mathsf{dig}', t, 1, \sigma') = 1$ (this would force $\mathsf{SEH}'.\mathsf{Extract}(\mathsf{td}_{\mathsf{SEH}'}, \mathsf{dig}') = 1$), so the second check of C fails for every witness at index t; that is, C has no accepting witness at index t and the challenger does not abort. Consequently $\mathcal{B}$ perfectly simulates: if the challenger encrypts at cutoff t (the bit $b = 0$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$, and if it encrypts at cutoff $t-1$ (the bit $b = 1$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Therefore, algorithm $\mathcal{B}$ breaks the index indistinguishability of Π_{PWE} with the same advantage ε. □

Claim C.11. *Assume that Π'_{SEH} satisfies mode indistinguishability. Then for any efficient Type 1 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.*

The proof of Claim C.11 is completely analogous to that of Claim C.9, with the challenge ciphertext generated at cutoff $t - 1$ in both experiments.
Combining Claims C.9 to C.11 via the triangle inequality, for every efficient Type 1 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| \le \mathsf{negl}(\lambda)$.

Type 2 adversaries ($t \in I$, hence i^* honest). We consider the following sequence of hybrid experiments:

- Hyb_0: Same as Real_0, except that the master public key hash is set up in binding mode at position t:
  - The challenger samples the binding key $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$.

  This hybrid is indistinguishable from Real_0 by the mode indistinguishability of Π_{SEH}.
- Hyb_1: Same as Hyb_0, except that the public key of the target player i^* is replaced by a commitment to 0:
  - When answering the key generation query that produces i^*, the challenger sets $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 0; r^*)$.

  This hybrid is indistinguishable from Hyb_0 by the computational hiding of Π_{Com}; the reduction is admissible because i^* is honest, so the opening r^* is never needed to answer a corruption query. Since the t^{th} block of the hashed list is $\mathsf{pk}_{i_t} = \mathsf{pk}_{i^*}$, by correctness of Π_{SEH} its honest opening verifies, so $\mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}) = \mathsf{pk}_{i^*}$, and by the somewhere extraction property $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, t, c, \sigma) = 1$ implies $c = \mathsf{pk}_{i^*}$. By the perfect binding of Π_{Com}, $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 0; r^*)$ is not a commitment to 1, so no r satisfies $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 1; r)$ and the third check of C fails at index t. Hence the circuit C has no accepting witness at index t.
- Hyb_2: Same as Hyb_1, except that the challenge ciphertext is generated at cutoff $t - 1$:
  - The challenger samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t - 1, \mu)$.

  This hybrid is indistinguishable from Hyb_1 by the index indistinguishability of Π_{PWE}; the Π_{PWE} challenger does not abort at index t because, as argued above, C has no accepting witness at index t.
- Hyb_3: Same as Hyb_2, except that the public key of i^* is restored to a commitment to 1:
  - When answering the key generation query that produces i^*, the challenger sets $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 1; r^*)$.

  This hybrid is indistinguishable from Hyb_2 by the computational hiding of Π_{Com}.
- Real_1: Same as Hyb_3, except that the master public key hash is set up in normal mode:
  - The challenger samples the standard key $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$.

  This hybrid is indistinguishable from Hyb_3 by the mode indistinguishability of Π_{SEH}. This is exactly the index indistinguishability game with bit $b = 1$, in which the challenge is encrypted at cutoff $t - 1$.

We now formalize the five transitions for Type 2 adversaries.

Claim C.12. *Assume that* Π_{SEH} *satisfies mode indistinguishability. Then for any efficient Type 2 adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.

Proof. Fix an efficient Type 2 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right|$. The only difference between Real_0 and Hyb_0 is how the master hash key is sampled: via $\mathsf{SEH.Setup}(1^\lambda)$ in Real_0 and via $\mathsf{SEH.SetupBinding}(1^\lambda, t)$ in Hyb_0; in both, the target public key is a commitment to 1 and the challenge ciphertext is generated at cutoff t. We construct an algorithm $\mathcal{B}$ that breaks the mode indistinguishability of Π_{SEH} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ, which commits to i^* and t, and sends t to the mode indistinguishability challenger of Π_{SEH}, which replies with a hash key hk.
2. Algorithm $\mathcal{B}$ samples the hash key for sets $\mathsf{hk}' \leftarrow \mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$ and $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.
3. Algorithm $\mathcal{B}$ answers $\mathcal{A}$'s queries exactly as the index indistinguishability challenger of Definition C.1.
4. When $\mathcal{A}$ outputs a list $(i_1, \ldots, i_N)$ with $i_t = i^*$, a subset $I \subseteq [N]$ with $t \in I$, and a message μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH.Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$ and $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, and sends ct to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger samples $\mathsf{hk} \leftarrow \mathsf{SEH.Setup}(1^\lambda)$ (the bit $b = 0$ game), then $\mathcal{B}$ simulates $\mathsf{Real}_0(\mathcal{A})$; if it samples $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$ (the bit $b = 1$ game), then $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$. Therefore, algorithm $\mathcal{B}$ breaks the mode indistinguishability of Π_{SEH} with the same advantage ε. □

Claim C.13. *Assume that* Π_{Com} *satisfies computational hiding. Then for any efficient Type 2 adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.

Proof. Fix an efficient Type 2 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$. The only difference between Hyb_0 and Hyb_1 is the public key of the target player i^*: a commitment to 1 in Hyb_0 and a commitment to 0 in Hyb_1; both sample the binding master hash key $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$ and encrypt at cutoff t. We construct an algorithm $\mathcal{B}$ that breaks the computational hiding of Π_{Com} with advantage ε:

1. The computational hiding challenger samples $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, $r^* \xleftarrow{R} \{0,1\}^\rho$, and a challenge commitment $c^* = \mathsf{Com.Commit}(\mathsf{ck}, \beta; r^*)$ for its bit β, and sends (ck, c^*) to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ, which commits to i^* and t, samples $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$ and $\mathsf{hk}' \leftarrow \mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.
3. Algorithm $\mathcal{B}$ answers $\mathcal{A}$'s queries as the challenger of Definition C.1, with one exception: on the key generation query that produces the target player i^*, it sets $\mathsf{pk}_{i^*} = c^*$ instead of sampling a fresh commitment. Since $\mathcal{A}$ is Type 2 it never corrupts i^*, so $\mathcal{B}$ is never asked to reveal i^*'s secret key (which it does not know).
4. When $\mathcal{A}$ outputs $(i_1, \ldots, i_N)$ with $i_t = i^*$, a subset $I \subseteq [N]$ with $t \in I$, and a message μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH.Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$ and $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, samples $\mathsf{ct} \leftarrow \mathsf{PWE.Enc}(1^\lambda, 1^N, C, t, \mu)$, and sends ct to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If the challenger's bit is $\beta = 1$, then c^* is a commitment to 1 and $\mathcal{B}$ simulates $\mathsf{Hyb}_0(\mathcal{A})$; if $\beta = 0$, then c^* is a commitment to 0 and $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$. Therefore, algorithm $\mathcal{B}$ breaks the computational hiding of Π_{Com} with the same advantage ε. □

Claim C.14. *Assume that* Π_{SEH} *satisfies somewhere extraction,* Π_{Com} *satisfies perfect binding, and* Π_{PWE} *satisfies index indistinguishability. Then for any efficient Type 2 adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda)$.

Proof. Fix an efficient Type 2 adversary $\mathcal{A}$ and let $\varepsilon = \left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right|$. The only difference between Hyb_1 and Hyb_2 is the cutoff of the challenge ciphertext: t in Hyb_1 and $t-1$ in Hyb_2; both sample the binding master hash key $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$ and set the target public key to $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 0; r^*)$. We construct an algorithm $\mathcal{B}$ that breaks the index indistinguishability of Π_{PWE} with advantage ε:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ in order to get i^* and t.
2. Algorithm $\mathcal{B}$ samples the hash key for public keys $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}}) \leftarrow \mathsf{SEH.SetupBinding}(1^\lambda, t)$, the hash key for sets $\mathsf{hk}' \leftarrow \mathsf{SEH}'.\mathsf{Setup}(1^\lambda)$, and commitment key $\mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends $\mathsf{crs} = (\mathsf{hk}, \mathsf{hk}', \mathsf{ck})$ to $\mathcal{A}$.
3. Algorithm $\mathcal{B}$ answers $\mathcal{A}$'s queries as the challenger of Definition C.1, except that on the key generation query producing i^* it samples $r^* \xleftarrow{R} \{0,1\}^\rho$ and sets $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 0; r^*)$. As $\mathcal{A}$ is Type 2 it never corrupts i^*.
4. When $\mathcal{A}$ outputs $(i_1, \ldots, i_N)$ with $i_t = i^*$, a subset $I \subseteq [N]$ with $t \in I$, and a message μ, algorithm $\mathcal{B}$ computes $(\mathsf{dig}, \sigma_1, \ldots, \sigma_N) = \mathsf{SEH.Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N}))$ and $(\mathsf{dig}', \sigma'_1, \ldots, \sigma'_N) = \mathsf{SEH}'.\mathsf{Hash}(\mathsf{hk}', \mathbf{1}_I)$, constructs $C = C[\mathsf{hk}, \mathsf{hk}', \mathsf{ck}, \mathsf{dig}, \mathsf{dig}']$ from Fig. 13, and sends $(1^N, C, t, \mu)$ to the index indistinguishability challenger of Π_{PWE}; the challenger replies with ct, which $\mathcal{B}$ forwards to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ forwards as its own output.

We verify that the Π_{PWE} challenger does not abort (i.e., that C has no accepting witness at index t). The t^{th} block of the hashed list is $\mathsf{pk}_{i_t} = \mathsf{pk}_{i^*}$; by correctness of Π_{SEH} its honest opening verifies, and since $(\mathsf{hk}, \mathsf{td}_{\mathsf{SEH}})$ is in the support of $\mathsf{SEH.SetupBinding}(1^\lambda, t)$, the somewhere extraction property gives $\mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{SEH}}, \mathsf{dig}) = \mathsf{pk}_{i^*}$, so any (c, σ) with $\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{dig}, t, c, \sigma) = 1$ satisfies $c = \mathsf{pk}_{i^*}$. Since $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 0; r^*)$, by the perfect binding of Π_{Com} there is no randomness r with $\mathsf{pk}_{i^*} = \mathsf{Com.Commit}(\mathsf{ck}, 1; r)$, so the third check of C fails for every witness at index t; that is, C has no accepting witness at index t and the challenger does not abort. Consequently $\mathcal{B}$ perfectly simulates: if the challenger encrypts at cutoff t (the bit $b = 0$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_1(\mathcal{A})$, and if it encrypts at cutoff $t-1$ (the bit $b = 1$ game) then $\mathcal{B}$ simulates $\mathsf{Hyb}_2(\mathcal{A})$. Therefore, algorithm $\mathcal{B}$ breaks the index indistinguishability of Π_{PWE} with the same advantage ε. □

Claim C.15. *Assume that Π_{Com} satisfies computational hiding. Then for any efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.*

Proof. This proof is completely analogous to that of Claim C.13, with the challenge ciphertext generated at cutoff $t-1$ in both experiments and the roles of the commitments to 0 and to 1 exchanged. □

Claim C.16. *Assume that Π_{SEH} satisfies mode indistinguishability. Then for any efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.*

Proof. This proof is completely analogous to that of Claim C.12, with the target public key a commitment to 1 and the challenge ciphertext generated at cutoff $t-1$ in both experiments. □

Combining Claims C.12 to C.16 via the triangle inequality, for every efficient Type 2 adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| \le \mathsf{negl}(\lambda)$.

Completing the proof. In both cases, Real_0 and Real_1 are exactly the selective index indistinguishability games of Construction C.4 with bits $b = 0$ and $b = 1$, respectively, while the intermediate distributions are computationally indistinguishable. Since every admissible adversary is of Type 1 or Type 2, the theorem follows. □

D Registered Broadcast-and-Trace

In this section, we define registered broadcast-and-trace, then show how to construct it from registered augmented broadcast encryption. The construction and the security proofs are completely analogous to the centralized case from [GVW19].

D.1 Registered Broadcast-and-Trace Definition

We start by defining the notion of registered broadcast-and-trace, which is a registration-based version of standard broadcast-and-trace. As usual, in this setting, users can sample their own public/private key-pairs and register their public key with an (untrusted) key curator. The key curator aggregates the public keys for all the users together into a succinct master public key that functions as the public parameters for a broadcast-and-trace scheme. At aggregation time, the key curator associates a different index with each user (e.g., by taking the lexicographic ordering of the users' public keys). This way, if there are N users, then each public key is associated with an index $i \in [N]$. Messages can be encrypted to an arbitrary subset $I \subseteq [N]$ of users. In addition, the scheme provides a public tracing algorithm that, given black-box access to a decoder D that distinguishes encryptions to a set I_D, outputs a non-empty set of positions in I_D whose secret keys the adversary controls. We require the usual CPA-security together with secure tracing.

Definition D.1 (Registered Broadcast-and-Trace)**.** A registered broadcast-and-trace scheme over a message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{RegBT}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: On input a security parameter λ, the setup algorithm outputs a common reference string crs.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$: On input the common reference string crs, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{Aggregate}\left(\mathsf{crs}, \left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N\right)\right) \to (\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N))$: On input the common reference string crs and a list of public keys $\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N\right)$, the aggregation algorithm outputs a master public key mpk and a list of helper decryption keys $(\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)$. Assume that mpk and each hsk_i implicitly contain the number of aggregated keys N. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I) \to \mathsf{ct}$: On input the common reference string crs, a master public key mpk, a message $\mu \in \mathcal{M}$, and a subset of indices $I \subseteq [N]$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}, \mathsf{hsk}, I, \mathsf{ct}) \to \mu$: On input the common reference string crs, a secret key sk, a helper decryption key hsk, a subset of indices $I \subseteq [N]$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}$ or the special symbol $\perp$. This algorithm is deterministic.
- $\mathsf{Trace}^D\left(\mathsf{crs}, \mathsf{mpk}, I_D, \mu_0, \mu_1, 1^{1/\varepsilon}\right) \to T$: On input the common reference string crs, a master public key mpk, a subset of indices $I_D \subseteq [N]$, two messages $\mu_0, \mu_1 \in \mathcal{M}$, and a distinguishing parameter $\varepsilon \in (0, 1)$, and given black-box access to a decoder D, the tracing algorithm outputs a set of indices $T \subseteq [N]$. Since the tracing algorithm takes only public information as input, it can be run by anyone, and the scheme is *publicly* traceable.

The Π_{RegBT} scheme should satisfy the following correctness and succinctness properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, all $N \in [2^\lambda]$, all messages $\mu \in \mathcal{M}$, all subsets $I \subseteq [N]$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for $i \in [N]$, the master public key and helper decryption keys $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{Aggregate}\left(\mathsf{crs}, \left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N\right)\right)$, and all ct in the support of $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I)$, the following holds for all $j \in I$:
$$\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}_j, \mathsf{hsk}_j, I, \mathsf{ct}) = \mu.$$
- **Succinctness:** We say that the registered broadcast-and-trace scheme is $\eta(N)$-succinct if there exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, all $N \in [2^\lambda]$, all messages $\mu \in \mathcal{M}$, all subsets $I \subseteq [N]$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}_j, \mathsf{sk}_j)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for $j \in [N]$, the master public key and helper decryption keys $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{Aggregate}\left(\mathsf{crs}, \left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N\right)\right)$, and all ct in the support of $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I)$, the following holds:
  - **Succinct master public key:** $|\mathsf{mpk}| \le \mathsf{poly}(\lambda)$.
  - **Succinct helper decryption keys:** $\left|\mathsf{hsk}_j\right| \le \mathsf{poly}(\lambda)$ for all $j \in [N]$.
  - $\eta(N)$**-succinct ciphertext:** $|\mathsf{ct}| \le \eta(N) \cdot \mathsf{poly}(\lambda)$.

Security games. In the two security games below, the challenger maintains a list K of users in the system, and an index ind. Initially, K is empty and ind = 0. During a *query phase*, the adversary may adaptively and repeatedly issue the following three kinds of queries:

- **Key generation query:** On a key-generation query, the challenger increments $\mathsf{ind} \leftarrow \mathsf{ind} + 1$, samples $(\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, sets $K[\mathsf{ind}] \leftarrow (\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}, \mathsf{honest})$, and returns $\mathsf{pk}_{\mathsf{ind}}$ to the adversary.
- **Corruption query:** On a corruption query, the adversary specifies an index $i \le \mathsf{ind}$. The challenger retrieves $(\mathsf{pk}_i, \mathsf{sk}_i, \cdot) \leftarrow K[i]$, overwrites $K[i] \leftarrow (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{corrupt})$ and sends sk_i to the adversary.
- **Malicious key registration query:** On a malicious key registration query, the adversary sends a public key pk. The challenger increments $\mathsf{ind} \leftarrow \mathsf{ind} + 1$, and sets $K[\mathsf{ind}] \leftarrow (\mathsf{pk}, \perp, \mathsf{corrupt})$.

The Π_{RegBT} scheme should satisfy the following security properties:

- **CPA-security:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the CPA game as follows:
  1. On input the security parameter 1^λ, the challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{A}$.
  2. The adversary adaptively issues key generation, corruption, and malicious key registration queries.
  3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
  4. The challenger retrieves the entry $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, and computes $(\mathsf{mpk}, \cdot) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I)$, and sends ct to $\mathcal{A}$.
  5. The adversary adaptively issues corruption queries.
  6. If there exists a position $j \in I$ such that $K[i_j]$ is marked as corrupt, then the experiment aborts and outputs $b' = 0$.
  7. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that the registered broadcast-and-trace scheme satisfies CPA-security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Secure tracing:** For a non-negligible function $\varepsilon(\cdot)$ and an adversary $\mathcal{A}$, we define the tracing game as follows:
  1. On input the security parameter 1^λ, the challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{A}$.
  2. The adversary adaptively issues key generation, corruption, and malicious key registration queries.
  3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I_D \subseteq [N]$, a decoder D, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
  4. The challenger retrieves the entry $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, and computes $(\mathsf{mpk}, \cdot) = \mathsf{Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$.
  5. The challenger computes $T \leftarrow \mathsf{Trace}^D\big(\mathsf{crs}, \mathsf{mpk}, I_D, \mu_0, \mu_1, 1^{1/\varepsilon(\lambda)}\big)$, which is the output of the experiment.

  Let $C \subseteq [N]$ be the set of positions $j \in [N]$ in the challenge list for which the entry $K[i_j]$ is marked as corrupt at the end of the experiment. We define the following events, whose probabilities are taken over the randomness of the experiment above:

  – Good-Dec: the decoder D output by $\mathcal{A}$ satisfies

$$\Pr\left[D(\mathsf{ct}) = b \;:\; b \xleftarrow{\mathrm{R}} \{0,1\},\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I_D)\right] \ge \frac{1}{2} + \varepsilon(\lambda),$$

  where the probability is over the choice of b and the randomness of Enc and D.

– Correct-Tr: $|T| > 0$ and $T \subseteq C \cap I_D$.

– False-Tr: $T \not\subseteq C \cap I_D$.

We say that the registered broadcast-and-trace scheme satisfies secure tracing if for every efficient adversary $\mathcal{A}$, every polynomial $q(\cdot)$, and every non-negligible function $\varepsilon(\cdot)$, there exist negligible functions $\mathsf{negl}_1(\cdot)$ and $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\varepsilon(\lambda) > 1/q(\lambda)$,

$$\Pr[\text{Correct-Tr}] \geq \Pr[\text{Good-Dec}] - \mathsf{negl}_1(\lambda) \quad \text{and} \quad \Pr[\text{False-Tr}] \leq \mathsf{negl}_2(\lambda).$$

Remark D.2 (Tracing Positions Rather Than Identities)**.** The tracing algorithm outputs a set T of *positions* in the aggregated list $(\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})$ rather than a set of user indices in the challenger's list K. Since the map $j \mapsto i_j$ is public, a position $j \in T$ identifies the user i_j whose key occupies that position. We note that the same user may register several public keys, in which case each of its keys occupies a separate position.

D.2 Registered Broadcast-and-Trace Construction

We construct a registered broadcast-and-trace scheme from a registered augmented broadcast encryption scheme, following Goyal, Vusirikala, and Waters [GVW19]. We recall the basic structure of the scheme:

- Setup, key generation, aggregation, and decryption are inherited unchanged from the registered augmented broadcast encryption scheme.
- Encryption is registered augmented broadcast encryption at cutoff $t = 0$, so that every position in the recipient set I can decrypt.
- Tracing estimates the decoder's success probability at each cutoff $t \in \{0, 1, \ldots, N\}$ and accuses every position at which consecutive estimates drop by more than a certain threshold.

We now give the formal construction:

Construction D.3 (Registered Broadcast-and-Trace)**.** Let $\Pi_{\mathsf{RegAugBE}} = (\mathsf{RegAugBE.Setup}, \mathsf{RegAugBE.KeyGen}, \mathsf{RegAugBE.Aggregate}, \mathsf{RegAugBE.Enc}, \mathsf{RegAugBE.Dec})$ be a registered augmented broadcast encryption scheme over message space $\mathcal{M}$. We construct a broadcast-and-trace scheme $\Pi_{\mathsf{RegBT}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ over $\mathcal{M}$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, the setup algorithm outputs the common reference string $\mathsf{crs} \leftarrow \mathsf{RegAugBE.Setup}(1^\lambda)$.
- $\mathsf{KeyGen}(\mathsf{crs})$: On input a common reference string crs, the key-generation algorithm outputs the key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RegAugBE.KeyGen}(\mathsf{crs})$.
- $\mathsf{Aggregate}\left(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\right)$: On input a common reference string crs and a list of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, the aggregation algorithm outputs $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{RegAugBE.Aggregate}\left(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\right)$.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I)$: On input a common reference string crs, a master public key mpk, a message $\mu \in \mathcal{M}$, and a subset of indices $I \subseteq [N]$, the encryption algorithm outputs $\mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, 0)$.
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}, \mathsf{hsk}, I, \mathsf{ct})$: On input a common reference string crs, a secret key sk, a helper decryption key hsk, a subset of indices $I \subseteq [N]$, and a ciphertext ct, the decryption algorithm outputs $\mathsf{RegAugBE.Dec}(\mathsf{crs}, \mathsf{sk}, \mathsf{hsk}, I, \mathsf{ct})$.
- $\mathsf{Trace}^D\left(\mathsf{crs}, \mathsf{mpk}, I_D, \mu_0, \mu_1, 1^{1/\varepsilon}\right)$: On input a common reference string crs, a master public key mpk (which implicitly specifies the number of aggregated keys N), a subset of indices $I_D \subseteq [N]$, two messages $\mu_0, \mu_1 \in \mathcal{M}$, and a distinguishing parameter $\varepsilon \in (0, 1)$, and given black-box access to a decoder D, the tracing algorithm does the following:

  1. Set the sample count $M = 8\lambda\,(N/\varepsilon)^2$.

2. For each $t \in \{0, 1, \ldots, N\}$, set $\mathsf{cnt}_t = 0$ and, for each $k \in [M]$, sample $b \xleftarrow{R} \{0,1\}$ and construct the ciphertext $\mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I_D, t)$. Increment cnt_t if $D(\mathsf{ct}) = b$. Set $\hat{p}_t = \mathsf{cnt}_t / M$.
3. Output the set of accused indices $T = \left\{ j \in [N] : \hat{p}_{j-1} - \hat{p}_j \geq \varepsilon / 4N \right\}$.

Since ε is given in unary and $N \leq \mathsf{poly}(\lambda)$ for any efficient caller, the tracing algorithm makes $(N+1) \cdot M = \mathsf{poly}(\lambda)$ calls to RegAugBE.Enc and to D, and is therefore efficient.

Theorem D.4 (Correctness). *If Π_{RegAugBE} is correct, then Construction D.3 is correct.*

Proof. Fix $\lambda \in \mathbb{N}$, $N \in [2^\lambda]$, a message $\mu \in \mathcal{M}$, a subset $I \subseteq [N]$, and a position $j \in I$, and let crs, $\left\{(\mathsf{pk}_i, \mathsf{sk}_i)\right\}_{i \in [N]}$, $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N))$, and ct be as in the correctness requirement of Definition D.1. By construction, crs, each $(\mathsf{pk}_i, \mathsf{sk}_i)$, and $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N))$ are in the support of the corresponding Π_{RegAugBE} algorithms, and ct is in the support of $\mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, 0)$. Since $I \cap \{1, 2, \ldots, N\} = I$, we have $j \in I \cap \{1, 2, \ldots, N\}$, so correctness of Π_{RegAugBE} at cutoff $t = 0$ gives $\mathsf{RegAugBE.Dec}(\mathsf{crs}, \mathsf{sk}_j, \mathsf{hsk}_j, I, \mathsf{ct}) = \mu$. By construction, this is exactly $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}_j, \mathsf{hsk}_j, I, \mathsf{ct})$. □

Theorem D.5 (Succinctness). *If Π_{RegAugBE} is $\eta(N)$-succinct, then Construction D.3 is $\eta(N)$-succinct.*

Proof. Fix $\lambda \in \mathbb{N}$, $N \in [2^\lambda]$, $\mu \in \mathcal{M}$, and a subset $I \subseteq [N]$, and let crs, $\left\{(\mathsf{pk}_j, \mathsf{sk}_j)\right\}_{j \in [N]}$, $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N))$, and ct be as in the succinctness requirement of Definition D.1. By construction, we have $(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)) = \mathsf{RegAugBE.Aggregate}\left(\mathsf{crs}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)\right)$ and ct is in the support of $\mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu, I, 0)$. The three required bounds are therefore exactly those guaranteed by the $\eta(N)$-succinctness of Π_{RegAugBE} at cutoff $t = 0$: $|\mathsf{mpk}| \leq \mathsf{poly}(\lambda)$, $\left|\mathsf{hsk}_j\right| \leq \mathsf{poly}(\lambda)$ for all $j \in [N]$, and $|\mathsf{ct}| \leq \eta(N) \cdot \mathsf{poly}(\lambda)$. □

Theorem D.6 (CPA-Security). *If Π_{RegAugBE} satisfies message hiding and index indistinguishability, then Construction D.3 satisfies CPA-security.*

Proof. Fix an efficient adversary $\mathcal{A}$ and let $N_{\max} = N_{\max}(\lambda)$ be a polynomial bound on the length of the list it outputs. For $b \in \{0,1\}$ and $t \in \{0, 1, \ldots, N_{\max}\}$, let $\mathsf{Hyb}_{b,t}$ be the following game, obtained from the CPA-security game of Definition D.1 with bit b by substituting the algorithms of Construction D.3 with their Π_{RegAugBE} implementations; the queries are those of Definition D.1.

1. On input the security parameter 1^λ, the challenger samples $\mathsf{crs} \leftarrow \mathsf{RegAugBE.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{A}$.
2. The adversary adaptively issues key generation, corruption, and malicious key registration queries.
3. The adversary $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
4. The challenger retrieves the entry $K[i_j] = (\mathsf{pk}_{i_j}, \cdot, \cdot)$ for each $j \in [N]$, and computes

$$(\mathsf{mpk}, \cdot) = \mathsf{RegAugBE.Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$$
$$\mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}\big(\mathsf{crs}, \mathsf{mpk}, \mu_b, I, \min(t, N)\big).$$

   The challenger sends ct to $\mathcal{A}$.
5. The adversary adaptively issues corruption queries.
6. If there exists a position $j \in I$ such that $K[i_j]$ is marked as corrupt, then the experiment aborts and outputs $b' = 0$.
7. The adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

By construction $\mathsf{Hyb}_{b,0}$ is the CPA-security game with bit b, so it suffices to show, for each $b \in \{0,1\}$ and all $i \in [N_{\max}]$, that $\mathsf{Hyb}_{b,i-1}$ and $\mathsf{Hyb}_{b,i}$ are computationally indistinguishable, and in addition that $\mathsf{Hyb}_{0,N_{\max}}$ and $\mathsf{Hyb}_{1,N_{\max}}$ are also computationally indistinguishable. Let B denote the abort condition of Step 6, namely that $K[i_j]$ is marked corrupt for some $j \in I$ at the end of the game.

Claim D.7. *Assume* Π_{RegAugBE} *satisfies index indistinguishability. Then for all* $b \in \{0,1\}$ *and* $t \in [N_{\max}]$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_{b,t-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{b,t}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda)$.

Proof. If $t > N$ the two games are identical, so assume $t \in [N]$. Let $\varepsilon = \left|\Pr[\mathsf{Hyb}_{b,t-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{b,t}(\mathcal{A}) = 1]\right|$. We construct an algorithm $\mathcal{B}$ for the index indistinguishability game of Definition C.1 with bit β:

1. The index indistinguishability challenger samples $\mathsf{crs} \leftarrow \mathsf{RegAugBE.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{B}$, which runs $\mathcal{A}$ on input 1^λ and forwards crs to it.

2. Algorithm $\mathcal{B}$ forwards each of $\mathcal{A}$'s key generation, corruption, and malicious key registration queries to its challenger and returns the response to $\mathcal{A}$, maintaining its own copy of the marks of K.

3. When $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$, algorithm $\mathcal{B}$ forwards the list $(i_1, \ldots, i_N)$, the subset I, the challenge index t, and the single message μ_b.

4. The challenger computes $\big(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)\big) = \mathsf{RegAugBE.Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$ and replies with the ciphertext $\mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I, t - \beta)$. Algorithm $\mathcal{B}$ forwards ct to $\mathcal{A}$.

5. Algorithm $\mathcal{B}$ forwards $\mathcal{A}$'s corruption queries as in Step 2.

6. Finally, algorithm $\mathcal{B}$ outputs $\mathcal{A}$'s guess b', except that it outputs $b' = 0$ if B occurs.

Algorithm $\mathcal{B}$ is efficient whenever $\mathcal{A}$ is, and it can decide B from its copy of K. Since the query interfaces of Definitions C.1 and D.1 coincide, every query is answered exactly as in $\mathsf{Hyb}_{b,t-\beta}$, and since $t \le N$, the challenge ciphertext is generated at cutoff $t - \beta = \min(t - \beta, N)$, again exactly as in $\mathsf{Hyb}_{b,t-\beta}$. The abort conventions do not conflict: Step 6 of the reduction reproduces the abort of Step 6 of $\mathsf{Hyb}_{b,t-\beta}$, while the index indistinguishability game aborts only if $t \in I$ and $K[i_t]$ is marked corrupt, which implies B, in which case $\mathcal{B}$ already outputs 0. Hence $\mathcal{B}$ simulates $\mathsf{Hyb}_{b,t}$ when $\beta = 0$ and $\mathsf{Hyb}_{b,t-1}$ when $\beta = 1$. Therefore, algorithm $\mathcal{B}$ breaks the index indistinguishability of Π_{RegAugBE} with the same advantage ε. □

Claim D.8. *Assume* Π_{RegAugBE} *satisfies message hiding. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathsf{Hyb}_{0,N_{\max}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,N_{\max}}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Let $\varepsilon = \left|\Pr[\mathsf{Hyb}_{0,N_{\max}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,N_{\max}}(\mathcal{A}) = 1]\right|$. We construct an algorithm $\mathcal{B}$ for the message hiding game of Definition C.1 with bit β:

1. The message hiding challenger samples $\mathsf{crs} \leftarrow \mathsf{RegAugBE.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{B}$, which runs $\mathcal{A}$ on input 1^λ and forwards crs to it.

2. Algorithm $\mathcal{B}$ forwards each of $\mathcal{A}$'s key generation, corruption, and malicious key registration queries to its challenger and returns the response to $\mathcal{A}$, maintaining its own copy of the marks of K.

3. When $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I \subseteq [N]$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$, algorithm $\mathcal{B}$ forwards them exactly as is. Its challenger replies with $\mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_\beta, I, N)$, where $\big(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)\big) = \mathsf{RegAugBE.Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$, and $\mathcal{B}$ forwards ct to $\mathcal{A}$.

4. Algorithm $\mathcal{B}$ forwards $\mathcal{A}$'s corruption queries as in Step 2.

5. Finally, algorithm $\mathcal{B}$ outputs $\mathcal{A}$'s guess b', except that it outputs $b' = 0$ if B occurs.

Algorithm $\mathcal{B}$ is efficient whenever $\mathcal{A}$ is, and it can decide B from its copy of K. Since the query interfaces of Definitions C.1 and D.1 coincide, every query is answered exactly as in $\mathsf{Hyb}_{\beta,N_{\max}}$. Next, since $\min(N_{\max}, N) = N$, the challenge ciphertext is distributed exactly as in $\mathsf{Hyb}_{\beta,N_{\max}}$. Finally, Step 5 reproduces the abort of Step 6 of that game, which the message hiding game itself does not perform. Hence $\mathcal{B}$ simulates $\mathsf{Hyb}_{\beta,N_{\max}}$. Therefore, algorithm $\mathcal{B}$ breaks the message hiding of Π_{RegAugBE} with the same advantage ε. □

The theorem follows from Claims D.7 and D.8 and a standard hybrid argument. □

Theorem D.9 (Secure Tracing). *If Π_{RegAugBE} satisfies message hiding and index indistinguishability, then Construction D.3 satisfies secure tracing.*

Proof. Fix an efficient adversary $\mathcal{A}$, a polynomial $q(\cdot)$, and a non-negligible $\varepsilon(\cdot)$ with $\varepsilon(\lambda) > 1/q(\lambda)$, and consider the tracing game of Definition D.1. Once mpk and $\mathcal{A}$'s output (D, I_D, μ_0, μ_1) are fixed, define for $t \in \{0, 1, \ldots, N\}$

$$p_t = \Pr\left[D(\mathsf{ct}) = b \;:\; b \xleftarrow{R} \{0,1\},\ \mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I_D, t)\right],$$

so that Good-Dec is the event $p_0 \geq 1/2 + \varepsilon(\lambda)$, and $\hat{p}_t$ as computed by Trace is the mean of $M = 8\lambda(N/\varepsilon)^2$ independent samples of an indicator with expectation p_t, the $N+1$ sampling loops being mutually independent. Note that $M = \mathsf{poly}(\lambda)$ since ε is given in unary and $\varepsilon(\lambda) > 1/q(\lambda)$.

Claim D.10. *Assume Π_{RegAugBE} satisfies index indistinguishability. Then there exists a negligible function $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{False\text{-}Tr}] \leq \mathsf{negl}_2(\lambda)$.*

Proof. For $j \in [N]$ let Gap_j be the event $p_{j-1} - p_j \geq \varepsilon/8N$, and let $\mathsf{Gap} = \bigvee_{j \notin C \cap I_D} \mathsf{Gap}_j$. Then

$$\Pr[\mathsf{False\text{-}Tr}] \leq \Pr\left[\mathsf{False\text{-}Tr} \wedge \overline{\mathsf{Gap}}\right] + \sum_{j \in [N]} \Pr\left[j \notin C \cap I_D \wedge \mathsf{Gap}_j\right].$$

For the first term, fix $j \notin C \cap I_D$ with $p_{j-1} - p_j < \varepsilon/8N$. Accusing j requires $\hat{p}_{j-1} - \hat{p}_j \geq \varepsilon/4N$, a deviation of at least $\varepsilon/8N$ from the mean of the independent estimates $\hat{p}_{j-1}$ and $\hat{p}_j$; since $M = 8\lambda(N/\varepsilon)^2$, a Chernoff bound bounds this by $2^{-\Omega(\lambda)}$, and a union bound over $j \in [N]$ bounds the first term by $N \cdot 2^{-\Omega(\lambda)}$.

For the second term, suppose towards a contradiction that $\Pr[j^* \notin C \cap I_D \wedge \mathsf{Gap}_{j^*}] \geq \delta(\lambda)$ for some $j^* \in [N]$ and non-negligible $\delta(\cdot)$. We construct an algorithm $\mathcal{B}$ for the index indistinguishability game of Definition C.1 with bit β:

1. The index indistinguishability challenger samples $\mathsf{crs} \leftarrow \mathsf{RegAugBE.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{B}$, which runs $\mathcal{A}$ on input 1^λ and forwards crs to it.

2. Algorithm $\mathcal{B}$ forwards each of $\mathcal{A}$'s key generation, corruption, and malicious key registration queries to its challenger and returns the response to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I_D \subseteq [N]$, a decoder D, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$, algorithm $\mathcal{B}$ samples $j \xleftarrow{R} [N]$ and $\gamma \xleftarrow{R} \{0,1\}$, and forwards the list $(i_1, \ldots, i_N)$, the subset I_D, the challenge index j, and the single message μ_γ. Its challenger replies with $\mathsf{ct}_1 \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_\gamma, I_D, j - \beta)$, where $\big(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)\big) = \mathsf{RegAugBE.Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$.

4. Algorithm $\mathcal{B}$ samples $\beta' \xleftarrow{R} \{0,1\}$, computes mpk itself, and computes $\mathsf{ct}_2 \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_\gamma, I_D, j - \beta')$.

5. Finally, algorithm $\mathcal{B}$ outputs β' if $D(\mathsf{ct}_1) = D(\mathsf{ct}_2)$, and $1 - \beta'$ otherwise.

Algorithm $\mathcal{B}$ is efficient whenever $\mathcal{A}$ is, invoking D twice. Note that $\mathcal{B}$ never needs to test whether $j \in C \cap I_D$: the index indistinguishability challenger aborts precisely when $j \in I_D$ and $K[i_j]$ is marked corrupt at the end of the game, which, since the tracing game has no post-challenge query phase, is precisely the condition $j \in C \cap I_D$; and on those runs the experiment outputs 0 regardless of $\mathcal{B}$'s output.

Writing $p_{t,\gamma} = \Pr[D(\mathsf{ct}) = \gamma : \mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_\gamma, I_D, t)]$, so that $p_t = (p_{t,0} + p_{t,1})/2$, a direct calculation over the four choices of (β, β') gives that, conditioned on the challenger not aborting and on γ, algorithm $\mathcal{B}$ outputs β with probability $1/2 + (p_{j-1,\gamma} - p_{j,\gamma})^2/2$. This is at least $1/2$ always; and with probability at least δ/N we have $j = j^*$, $j \notin C \cap I_D$, and $p_{j-1} - p_j \geq \varepsilon/8N$, in which case $p_{j-1,\gamma} - p_{j,\gamma} \geq \varepsilon/8N$ for at least one of the two values of γ. Averaging over γ, algorithm $\mathcal{B}$ outputs β with probability at least $1/2 + \delta\varepsilon^2/256N^3$. Therefore, algorithm $\mathcal{B}$ breaks the index indistinguishability of Π_{RegAugBE} with advantage at least $\delta\varepsilon^2/256N^3$, which is non-negligible since $\varepsilon(\lambda) > 1/q(\lambda)$. This is a contradiction. □

Claim D.11. *Assume* Π_{RegAugBE} *satisfies message hiding. Then* $\Pr[p_N > 1/2 + \varepsilon/4] \leq \mathsf{negl}(\lambda)$ *for some negligible function* $\mathsf{negl}(\lambda)$.

Proof. Let $\nu(\lambda)$ denote this probability. We construct an algorithm $\mathcal{B}$ for the message hiding game of Definition C.1 with bit β:

1. The message hiding challenger samples $\mathsf{crs} \leftarrow \mathsf{RegAugBE.Setup}(1^\lambda)$, initializes $K \leftarrow \varnothing$ and $\mathsf{ind} \leftarrow 0$, and sends crs to $\mathcal{B}$, which runs $\mathcal{A}$ on input 1^λ and forwards crs to it.

2. Algorithm $\mathcal{B}$ forwards each of $\mathcal{A}$'s key generation, corruption, and malicious key registration queries to its challenger and returns the response to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a list of indices $(i_1, \ldots, i_N)$, a subset $I_D \subseteq [N]$, a decoder D, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$, algorithm $\mathcal{B}$ computes $\big(\mathsf{mpk}, (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)\big) = \mathsf{RegAugBE.Aggregate}\big(\mathsf{crs}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_N})\big)$ itself and estimates p_N by the mean $\hat{p}$ of M independent samples of $D(\mathsf{ct})$, where $b \xleftarrow{R} \{0,1\}$ and $\mathsf{ct} \leftarrow \mathsf{RegAugBE.Enc}(\mathsf{crs}, \mathsf{mpk}, \mu_b, I_D, N)$; it can generate these on its own.

4. If $\hat{p} \leq 1/2 + \varepsilon/8$, algorithm $\mathcal{B}$ outputs a uniformly random bit without submitting a challenge. Otherwise, algorithm $\mathcal{B}$ forwards the list $(i_1, \ldots, i_N)$, the subset I_D, and the pair of messages μ_0, μ_1, and outputs $D(\mathsf{ct})$ for the ciphertext ct it receives.

Algorithm $\mathcal{B}$ is efficient whenever $\mathcal{A}$ is, and it is admissible, as message hiding imposes no restriction on the corrupted positions, which is essential here, since $\mathcal{A}$ may corrupt positions in I_D. By a Chernoff bound, $|\hat{p} - p_N| < \varepsilon/16$ except with probability $2^{-\Omega(\lambda)}$; hence $\mathcal{B}$ takes the second branch with probability at least $\nu - 2^{-\Omega(\lambda)}$, and on that branch $p_N > 1/2 + \varepsilon/16$, so $\mathcal{B}$ outputs the challenger's bit with probability at least p_N. Overall $\mathcal{B}$ outputs the challenger's bit with probability at least $1/2 + (\nu - 2^{-\Omega(\lambda)})\varepsilon/16$. Therefore, algorithm $\mathcal{B}$ breaks the message hiding of Π_{RegAugBE} with advantage at least $(\nu - 2^{-\Omega(\lambda)})\varepsilon/16$, and we conclude that ν is negligible. □

Claim D.12. *Assume* Π_{RegAugBE} *satisfies message hiding. Then* $\Pr[T \neq \varnothing] \geq \Pr[\text{Good-Dec}] - \mathsf{negl}(\lambda)$ *for some negligible function* $\mathsf{negl}(\lambda)$.

Proof. Condition on Good-Dec, so that $p_0 \geq 1/2 + \varepsilon$, and on $p_N \leq 1/2 + \varepsilon/4$, which holds up to a negligible loss by Claim D.11. Then $\sum_{j \in [N]}(p_{j-1} - p_j) = p_0 - p_N \geq \varepsilon/2$, so there is some $j \in [N]$ with $p_{j-1} - p_j \geq \varepsilon/2N$. For this j, accusing it requires only $\hat{p}_{j-1} - \hat{p}_j \geq \varepsilon/4N$, a deviation of at most $\varepsilon/4N$ from the mean; since $M = 8\lambda(N/\varepsilon)^2$, a Chernoff bound gives $j \in T$ except with probability $2^{-\Omega(\lambda)}$. Hence $\Pr[T = \varnothing \wedge \text{Good-Dec}] \leq \mathsf{negl}(\lambda)$, and the claim follows. □

Since Correct-Tr is the event that $T \neq \varnothing$ and $\overline{\text{False-Tr}}$, combining Claims D.10 and D.12 gives

$$\Pr[\text{Correct-Tr}] \geq \Pr[T \neq \varnothing] - \Pr[\text{False-Tr}] \geq \Pr[\text{Good-Dec}] - \mathsf{negl}_1(\lambda)$$

for some negligible function $\mathsf{negl}_1(\lambda)$, which together with Claim D.10 establishes secure tracing. □