Monotone Policy BARGs from
BARGs and Additively Homomorphic Encryption

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

A monotone policy batch NP language $\mathcal{L}_{\mathcal{R},P}$ is parameterized by a monotone policy $P\colon \{0,1\}^k \to \{0,1\}$ and an NP relation $\mathcal{R}$. A statement $(x_1, \ldots, x_k)$ is a YES instance if there exists $w_1, \ldots, w_k$ where $P(\mathcal{R}(x_1, w_1), \ldots, \mathcal{R}(x_k, w_k)) = 1$. For example, we might say that an instance $(x_1, \ldots, x_k)$ is a YES instance if a majority of the statements are true. A monotone policy batch argument (BARG) for NP allows a prover to prove that $(x_1, \ldots, x_k) \in \mathcal{L}_{\mathcal{R},P}$ with a proof of size $\mathsf{poly}(\lambda, |\mathcal{R}|, \log k)$, where λ is the security parameter, $|\mathcal{R}|$ is the size of the Boolean circuit that computes $\mathcal{R}$, and k is the number of instances. Recently, Brakerski, Brodsky, Kalai, Lombardi, and Paneth (CRYPTO 2023) gave the first monotone policy BARG for NP from the learning with errors (LWE) assumption.

In this work, we describe a generic approach for constructing monotone policy BARGs from any BARG for NP together with an additively homomorphic encryption scheme. This yields the first constructions of monotone policy BARGs from the k-Lin assumption in prime-order pairing groups as well as the (subexponential) DDH assumption in *pairing-free* groups. Central to our construction is a notion of a zero-fixing hash function, which is a relaxed version of a predicate-extractable hash function from the work of Brakerski et al. Our relaxation enables a direct realization of zero-fixing hash functions from BARGs for NP and additively homomorphic encryption, whereas the previous notion relied on leveled homomorphic encryption, and by extension, the LWE assumption.

As an application, we also show how to combine a monotone policy BARG with a puncturable signature scheme to obtain a monotone policy aggregate signature scheme. Our work yields the first (statically-secure) monotone policy aggregate signatures that supports general monotone Boolean circuits from standard pairing-based assumptions. Previously, this was only known from LWE.

1 Introduction

A non-interactive batch argument (BARG) for NP allows a prover to convince a verifier that a collection of k statements $x_1, \ldots, x_k$ is true with a proof whose size scales *sublinearly* with k. Beyond the immediate application to amortizing the communication cost of NP verification, batch arguments for NP also play a key role in constructing delegation for RAM programs (also known as a succinct non-interactive argument (SNARG) for P) [KVZ21, CJJ21b, KLVW23] and incrementally verifiable computation [DGKV22, PP22]. These objects have received extensive study recently, and to date, we have constructions from most standard algebraic assumptions in cryptography such as the learning with errors (LWE) assumption [CJJ21b], the k-Lin assumption on groups with bilinear maps [WW22], the (sub-exponential) decisional Diffie-Hellman (DDH) assumption in pairing-free groups [CGJ+23], or combinations of quadratic residuosity and (sub-exponential) DDH in pairing-free groups [CJJ21a, HJKS22].

Beyond batch NP and P. The recent successes in constructing succinct arguments for batch NP and for P from standard cryptographic assumptions has motivated the study of other (sub)-classes of NP for which we can build succinct non-interactive arguments from standard (falsifiable) assumptions. Very recently, Brakerski, Brodsky, Kalai, Lombardi, and Paneth [BBK+23] showed how to construct SNARGs for monotone policy batch NP. At a high level, the monotone policy batch NP language $\mathcal{L}_{\mathcal{R},P}$ is defined with respect to an NP relation $\mathcal{R}$ together with a monotone policy $P\colon \{0,1\}^k \to \{0,1\}$ as follows:

$$\mathcal{L}_{\mathcal{R},P} = \{(x_1, \ldots, x_k) \mid \exists (w_1, \ldots, w_k) : P(\mathcal{R}(x_1, w_1), \ldots, \mathcal{R}(x_k, w_k)) = 1\}\,.$$

In words, an instance $(x_1, \ldots, x_k)$ is true as long as an acceptable subset of the statements are true (as determined by the policy P). Such "monotone policy batch arguments" capture policies like majority, general thresholds, and more. The standard batch argument corresponds to the special case where the policy P is a simple conjunction.

Brakerski et al. [BBK+23] provided two constructions of monotone policy BARGs for NP. The first construction only relies on standard (somewhere extractable) BARGs and collision-resistant hash functions, but could only support monotone policies of logarithmic depth (i.e., monotone NC^1). To extend to monotone policies of arbitrary polynomial depth, they combine standard BARGs with a new notion of a predicate-extractable hash function, which they then build from the LWE assumption (specifically, they rely on leveled homomorphic encryption). This yields a monotone policy batch argument for arbitrary monotone policies from the LWE assumption. Due to the current reliance on leveled homomorphic encryption to construct the predicate-extractable hash function, instantiations of monotone policy BARGs for arbitrary-depth policies rely on the LWE assumption.

1.1 Our Results

Our main result in this work is showing how to construct BARGs for monotone policies by combining a (standard) BARG with an *additively* homomorphic encryption scheme (which can in turn be built from most number-theoretic assumptions [Gam84, Pai99, Reg05]). Combined with the recent progress on constructing BARGs from pairing-based groups [WW22] and pairing-free groups [CGJ+23], we obtain the first monotone policy BARGs for NP from the k-Lin assumption over pairing groups and from the (sub-exponentially) DDH assumption in pairing-free groups. We provide an overview of our techniques in Section 1.2 and summarize our main results in the following theorem:

Theorem 1.1 (Informal)**.** *Assuming any of (1) the plain LWE assumption, (2) the k-Lin assumption over pairing groups for any constant $k \in \mathbb{N}$, or (3) the (sub-exponential) DDH assumption in pairing-free groups, there exists a monotone policy BARG for all polynomial-size monotone circuit policies. The monotone policy BARG satisfies non-adaptive soundness and the proof size is $\mathsf{poly}(\lambda + |C| + \log |P|)$, where $|C|$ denotes the size of the Boolean circuit computing the* NP *relation, and $|P|$ is the size of the monotone policy.*

Monotone policy aggregate signatures. A key difference between Theorem 1.1 and the previous LWE-based construction [BBK+23] is that we obtain a *non-adaptively-sound* BARG for monotone circuit policies whereas the [BBK+23] construction satisfied a stronger "somewhere extractability" notion. That is, in [BBK+23], the common reference string (CRS) can be sampled in a trapdoor mode and the trapdoor can be used to recover a witness for some x_i given a valid proof on statements $(x_1, \ldots, x_k)$. While extractability is often useful to have in a cryptographic primitive, it is not always essential.

As an illustrative example, we show how to use monotone policy BARGs in conjunction with (puncturable) signatures [GVW19] to construct a monotone policy aggregate multi-signature scheme. In an aggregate multi-signature scheme, there is a set of k signers, each with a signing/verification key-pair $(\mathsf{sk}_i, \mathsf{vk}_i)$. Given a policy P and a set of signatures σ_i for $i \in S$ (where σ_i verifies with respect to vk_i) on a common message m, if the set S satisfies the policy P, then it is possible to aggregate $\{\sigma_i\}_{i \in S}$ into a single short signature whose size is sublinear in $|S|$. For instance, P might encode a "threshold" policy that accepts all sets of size at least t. Crucially, static security of our monotone policy aggregate signature scheme only relies on *non-adaptive soundness* of the monotone policy BARG and security of the puncturable signature scheme. There is no need for an explicit extraction requirement. Very briefly, a puncturable signature scheme allows one to sample a "punctured" verification key vk (and associated signing key) for some message m^*. The punctured verification key is computationally indistinguishable from a normal verification key, but has the property that there does not *exist* any signatures on the punctured message m^* with respect to the punctured key. As shown in [GVW19], puncturable (or "all-but-one signatures") can be constructed from many standard number-theoretic assumptions. We summarize this result in the following theorem:

Theorem 1.2 (Informal)**.** *Assuming the existence of a non-adaptively sound monotone BARG and a puncturable signature scheme, there exists a monotone policy aggregate multi-signature scheme. The scheme satisfies static unforgeability and the size of the aggregate signature is $\mathsf{poly}(\lambda + \log |P|)$, where $|P|$ denotes the size of the circuit computing the monotone policy.*

Theorem 1.2 shows that in combination with puncturable signatures, soundness alone is sufficient for building aggregate signatures for general monotone policies. Notably, Theorem 1.2 also provides the *first* monotone policy aggregate signature from pairing-based assumptions (in the plain model). Previous work have shown how to build *vanilla* aggregate signatures using (vanilla) non-interactive batch arguments [WW22, DGKV22]. In an independent and concurrent work, [BCJP24] also show how to construct a monotone policy aggregate multi-signature. Their work provides two constructions of monotone policy aggregate (multi)-signatures. The first scheme supports monotone policies that can be implemented by a read-once, bounded-space Turing machine and is also *adaptively* secure. This scheme relies on somewhere extractable BARGs and a verifiable private information retrieval scheme [BKP22], and can be instantiated from standard pairing-based or lattice-based assumptions. The second scheme supports policies implemented by an arbitrary monotone Boolean circuit, but achieves a weaker security definition (closer to static security) and also relies on fully homomorphic encryption (which to date, is not known from pairing-based assumptions). Theorem 1.2 gives a statically-secure monotone policy aggregate signature scheme that supports all monotone Boolean circuits, and does not rely on fully homomorphic encryption. This enables a new instantiation from pairings.

Soundness vs. extraction. While Theorem 1.2 shows that extraction is *unnecessary* for all applications of monotone policy BARGs, our proof strategy for arguing soundness can nonetheless be extended to achieve a notion of extractability (see Section 8). The notion we achieve is similar to the somewhere extractability notion from [BBK+23], where for every monotone policy P, they define a notion of a "necessary set" associated with P (i.e., a set with the property that for every satisfying input $(x_1, \ldots, x_n)$ to P, there exists $i \in S$ where $x_i = 1$). The somewhere extractability notion from [BBK+23] programs S into the common reference string, and asserts that whenever the prover comes up with an accepting proof for statements $(x_1, \ldots, x_k)$ for an NP relation $\mathcal{R}$ and policy P, then the extractor will output w_i for $i \in S$ where $\mathcal{R}(x_i, w_i) = 1$. Our construction satisfies a looser variant of this property where the success probability of the extractor is smaller by a factor of $1/k$. We refer to this notion as semi-somewhere extractability. While our construction does achieve this notion of extraction with essentially no modification (see Section 8), we choose to focus on the simpler notion of non-adaptive soundness in the core part of this paper. Our rationale is twofold:

- First, there is a lack of consensus on what the "right" notion of extraction is when it comes to the setting of monotone policy BARGs. Notably, the recent and concurrent work of [BCJP24] that builds monotone policy aggregate signatures highlighted the *inadequacy* of the somewhere extractability notion from [BBK+23] for their particular application to constructing monotone policy aggregate signatures. Indeed, the work of [BCJP24] propose two different and seemingly incomparable notions of extraction for their application. This illustrates that the most useful or desirable notion of extraction for monotone policy BARGs may be application-dependent.

- Second, while it is straightforward to show that our construction satisfies some notion of extractability, proving this property does not appear to confer additional capabilities. For the main application to statically-secure aggregate signatures, we showed above that non-adaptive soundness already suffices. There is no need for extraction if this is the end goal. The main advantage of having some kind of extractability definition is we can apply this construction to compile any digital signature scheme into a monotone policy aggregate signature scheme, as opposed to restricting ourselves to puncturable signatures (and we show this in Section 8.1). While there is a qualitative benefit to this, we do not view it as strong evidence that semi-somewhere extractability is a clearly more powerful or more useful notion than non-adaptive soundness.

A new application: general-policy BARGs for NP $\cap$ coNP. We also highlight a simple application of BARGs for monotone policy batch NP to constructing a BARG that supports *arbitrary* policies over languages in NP $\cap$ coNP. Our observation essentially follows the similar strategy of extending monotone closure of SZK to non-monotone closure [Vad06]. Specifically, for a language $\mathcal{X} \in \mathsf{NP} \cap \mathsf{coNP}$ and an arbitrary policy $P \colon \{0,1\}^k \to \{0,1\}$, we define the language

$$\mathcal{L}_{\mathcal{X},P} = \{(x_1, \ldots, x_k) \mid P(b_1, \ldots, b_k) = 1 \text{ where } b_i = \mathbb{1}\{x_i \in \mathcal{X}\}\},$$

where $\mathbb{1}\{x_i \in \mathcal{X}\}$ is the indicator function that outputs 1 if $x_i \in \mathcal{X}$ and 0 otherwise. Importantly, in this context, we allow P to be any arbitrary (possibly non-monotone) Boolean circuit. It is not difficult to see that a BARG for monotone policy batch NP immediately implies a BARG for $\mathcal{L}_{\mathcal{X},P}$. Namely, we first re-express the circuit P on k inputs $b_1, \ldots, b_k$ as a new *monotone* circuit P' on $2k$ inputs corresponding to the original input bits $b_1, \ldots, b_k$ as well as their negations $\bar{b}_1, \ldots, \bar{b}_k$. We can then apply a BARG for monotone policy batch NP on the set of $2k$ inputs with the policy P'. For this transformation to work, it is important that for each statement x_i, the prover can either provide a proof of membership $x_i \in \mathcal{X}$ (which sets $b_i = 1$) or a proof of non-membership $x_i \notin \mathcal{X}$ (which sets $\bar{b}_i = 1$).

1.2 Technical Overview

The starting point of our BARG construction is the "canonical protocol" from [BBK+23, §2.1]. We recall this below. In our description, we will consider the NP relation of Boolean circuit satisfiability.

- Given a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \rightarrow \{0,1\}$, a monotone policy $P\colon \{0,1\}^k \rightarrow \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, witnesses $w_1, \ldots, w_k \in \{0,1\}^h$, the prover first computes $b_i \leftarrow C(x_i, w_i)$ for all $i \in [k]$.
- The prover then evaluates the circuit $P(b_1, \ldots, b_k)$. The prover commits to all of the wire values in $P(b_1, \ldots, b_k)$ using a succinct commitment com that supports local openings. We index the input wires with the integers $1, \ldots, k$, the output wire by s (where s is the number of wires in P), and the intermediate wires with $k+1, \ldots, s-1$.
- The prover uses a batch argument to prove the following statements with respect to the commitment com:
  - **Input wires:** For every input wire $j \in [k]$, it proves that there exists a local opening of com to a value $b_j \in \{0,1\}$ at index j, and moreover, $b_j = C(x_j, w_j)$.
  - **Gate computation:** For every gate g in P with input wires $j_{\mathrm{L}}, j_{\mathrm{R}}$ and output wire j, it proves that there exists a local opening of com to wire values $b_{j_{\mathrm{L}}}, b_{j_{\mathrm{R}}}, b_i \in \{0,1\}$ at indices $j_{\mathrm{L}}, j_{\mathrm{R}}, j \in [s]$, respectively, and moreover, $b_j = g(b_{j_{\mathrm{L}}}, b_{j_{\mathrm{R}}})$.
  - **Output wire**: It proves that there exists a local opening to the value 1 at index s for com.

  The proof consists of the commitment com together with the batch argument π.

When the policy circuit P has logarithmic depth, the authors of [BBK+23] describe a simple inductive argument to argue the security of this construction by relying on somewhere extractability of the underlying BARG. Somewhere extractability says that the common reference string of the BARG can be programmed at a small number of (hidden) indices $i_1, \ldots, i_\ell$. Given a valid proof π for $(x_1, \ldots, x_n)$ along with a trapdoor, one can extract witnesses for $x_{i_1}, \ldots, x_{i_\ell}$. However, when P has super-logarithmic depth, the basic inductive argument no longer suffices (specifically, the security loss of the reduction decays *exponentially* in the depth of P).

Predicate-extractable hash functions for bit-fixing constraints. To construct monotone policy BARGs for policies P of arbitrary depth, the authors of [BBK+23] replace the Merkle hash of the wire values with a more sophisticated "predicate-extractable" hash function for bit-fixing constraints.[1]

A predicate-extractable hash function for bit-fixing predicates is a hash function where the hash key can be programmed in one of two computationally indistinguishable modes: (1) a normal mode and (2) a bit-fixing mode. In bit-fixing mode, the setup algorithm takes as input a set of indices $S \subseteq [n]$ along with a collection of bits $\{(i, y_i)\}_{i \in S}$, where n is the input length. It outputs a hash key hk and an extraction trapdoor td. The correctness requirement says that if $\mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$ for an input $\mathbf{x}$ where $x_i = y_i$ for all $i \in S$, then $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = \mathsf{Matching}$. Alternatively, if dig is a digest for an input $\mathbf{x}$ where $x_i \neq y_i$ for some $i \in S$, then $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ should output $(\mathsf{NotMatching}, i^*)$ where $i^* \in S$ is an index where $x_{i^*} \neq y_{i^*}$. Essentially, the extractor is deciding whether dig corresponds to the hash of an input that is consistent with $\{(i, y_i)\}_{i \in S}$. If the hash is declared inconsistent, the extractor outputs one of the inconsistent indices. Finally, the hash function supports succinct local openings to individual bits of an input. The two key security properties are as follows:

- For a hash digest dig where $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = \mathsf{Matching}$, then it should be computationally difficult to construct an opening for dig to a value $x_i \neq y_i$ for any $i \in S$.

[1] This is conceptually similar to the notion of function-binding hash functions introduced concurrently in [FWW23].

- For a hash digest dig where $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = (\mathsf{NotMatching}, i^*)$, then it should be computationally difficult for the adversary to open index i^* to the value y_{i^*}.

In the monotone BARG construction, the prover takes the Boolean circuit C, the policy P, the statements $(x_1, \dots, x_k)$ and the witnesses $(w_1, \dots, w_k)$, and computes $b_i \leftarrow C(x_i, w_i)$ and $P(b_1, \dots, b_k)$. Let $(b_1, \dots, b_s)$ be the complete set of wire values in $P(b_1, \dots, b_k)$, arranged in topological order. The prover hashes the wire values $(b_1, \dots, b_s)$ using the predicate-extractable hash function. In fact, the prover computes two independent hashes $\mathsf{dig}_1, \mathsf{dig}_2$ of the wire values, and the BARG will check validity of the openings against *both* hashes. To argue non-adaptive soundness, the authors of [BBK+23] first define the zero-set J associated with a circuit C, policy P, and statement $(x_1, \dots, x_k)$:

- For each $i \in [k]$, let $\beta_i^* = 1$ if there exists w_i such that $C(x_i, w_i) = 1$ and let $\beta_i^* = 0$ otherwise.
- Let $\beta_1^*, \dots, \beta_s^* = P(\beta_1^*, \dots, \beta_k^*)$ be the wire values in $P(\beta_1^*, \dots, \beta_s^*)$, where the wires are ordered topologically.
- Let $J = \left\{ i \in [k] : \beta_i^* = 0 \right\}$. For a layer index t, define $J_t \subseteq J$ to just contain the indices of wires in layer t of P.

The proof of non-adaptive soundness now proceeds as follows:[2]

- Take any circuit C, monotone policy P, and statements $x_1, \dots, x_k$. The invariant they use roughly says the following: if $\mathsf{hk}_0, \mathsf{hk}_1$ are programmed to bind to the all-zeroes string on the zero-sets J_i, J_{i-1} for layers i and $i-1$ of P, and the digest associated with the upper layer is $\mathsf{NotMatching}$, then the digest associated with the lower layer is also $\mathsf{NotMatching}$.
- To establish this invariant, the proof critically relies on BARG security and security of the predicate-extractable hash function. Namely, if the extractor declares an index $j \in J_i$ in the upper layer to be $\mathsf{NotMatching}$ and the BARG is set to be extracting on wire j, then that means the adversary must have opened one of the input wires j' (to the gate computing wire j) to a 1 where $j' \in J_{i-1}$ (since the policy P is monotone). Security of the hash function then says that the extractor must declare the digest associated with the lower layer to be $\mathsf{NotMatching}$.
- To complete the proof, they argue that the output layer must be $\mathsf{NotMatching}$ (by programming the BARG to be extracting on the output wire). By propagating the invariant to the input wires, they conclude that the input layer must be $\mathsf{NotMatching}$ (when one of the hash keys is programmed to bind on the input layer). In this case, programming the BARG to be extracting on the wire identified by the $\mathsf{NotMatching}$ input (output by the extractor for the hash function) yields a contradiction (in this case, the BARG extractor would need to output a witness for a false NP statement).

The authors of [BBK+23] then show how to construct a predicate-extractable hash function for bit-fixing predicates using the learning with errors (LWE) assumption. Their construction specifically relies on leveled homomorphic encryption (similar to the construction of somewhere statistically binding hash functions [HW15]). In conjunction with BARGs for NP based on LWE [CJJ21b], this yields a monotone policy BARG for NP from LWE.

This work: zero-fixing hash functions. The starting point of our work is a relaxation of a predicate-extractable hash function for bit-fixing predicates we call a zero-fixing hash function. Like the predicate-extractable hash function, the zero-fixing hash function supports succinct local openings and moreover, the hash key for a zero-fixing hash function can be sampled in one of two computationally-indistinguishable modes: (1) a normal mode and (2) a zero-fixing mode. In zero-fixing mode, the setup algorithm takes as input a set $S \subseteq [n]$ of indices (that should be zero) and outputs a hash key hk along with a trapdoor td. There is also an extract algorithm $\mathsf{Extract}$ that takes as input the hash key hk and a digest dig, and outputs either $\mathsf{Matching}$ or $\mathsf{NotMatching}$. The key distinction with predicate-extractable hash functions is that $\mathsf{Extract}$ only outputs the flag; it does *not* output an index when it declares a digest $\mathsf{NotMatching}$. Correspondingly, the zero-fixing security requirement only imposes a requirement for matching digests:

[2] With a suitable strengthening of the notion of predicate-extractable hash functions, the authors of [BBK+23] also show how to obtain a somewhere *extractable* monotone policy BARG. In this work, we focus on achieving the core notion of non-adaptive soundness.

- **Zero-fixing:** Suppose $(\mathsf{hk}, \mathsf{td})$ are sampled in zero-fixing mode for a set S. Then, for any digest dig where $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ outputs Matching, it should be hard to find an opening to an index $i \in S$ to the value 1.

While this distinction of having the extractor output a mismatching index j or not might seem like a small difference, it has two significant implications:

- **Simpler to construct:** By only requiring the zero-fixing hash function declare whether a digest is Matching or NotMatching, we significantly simplify the construction of the hash function. Whereas computing and propagating an index of a "mismatching bit" (as in [BBK⁺23]) relies heavily on (leveled) homomorphic encryption, checking whether there *exists* a mismatching index or not can be realized from simpler tools. As we show in this work (and describe later on), we can construct zero-fixing hash functions generically from BARGs for NP together with any *additively* homomorphic encryption scheme (Section 5). If we prefer to avoid non-black-box techniques altogether, we also describe a direct algebraic construction using composite-order pairing groups (Section 6). This is the critical distinction that allows us to obtain monotone policy BARGs from group-based assumptions (which give additively homomorphic encryption [Gam84] but not leveled homomorphic encryption).

- **Sufficient for monotone policy BARGs:** A second important fact is that our notion of zero-fixing hash function still suffices to build monotone policy BARGs. As noted in the preceding sketch, the soundness analysis from [BBK⁺23] critically relied on the hash function extractor outputting an index of a mismatching bit. This is so that when the BARG is programmed to bind on the wire associated with the mismatching index, the NotMatching invariant propagates from the output layer to the input layer. In our setting, the zero-fixing extractor only outputs Matching or NotMatching, and in the case where the extractor outputs NotMatching, we *cannot* definitively declare an index to be "mismatching." This requires a new proof strategy as well as imposing additional security requirements on the zero-fixing hash function. We describe these properties as well as our new proof strategy in more detail below.

Monotone policy BARGs from zero-fixing hash functions. Our main construction is similar to the canonical protocol from [BBK⁺23] sketched above, except the prover commits to all of the wires of the policy circuit P using two zero-fixing hash functions (with hash keys hk_1 and hk_2). Our security analysis takes a different *bottom-up* approach rather than the previous top-down approach. The bottom-up approach is more natural when using our zero-fixing hash function. Here, we provide a sketch of our non-adaptive soundness analysis.

To argue non-adaptive soundness, fix a Boolean circuit C, a monotone policy P (assumed to be a layered Boolean circuit), and a false statement $(x_1, \ldots, x_k)$. Similar to [BBK⁺23], we define the zero-set J associated with C, P, and $(x_1, \ldots, x_k)$. The zero-set J contains the indices of the wires with value 0 in the computation $P(\beta_1^*, \ldots, \beta_k^*)$ where $\beta_i^* = 1$ if there exists w_i where $C(x_i, w_i) = 1$ and 0 otherwise. Since P is monotone, for all $w_1, \ldots, w_k$, the wire values of $P(C(x_1, w_1), \ldots, C(x_k, w_k))$ on the set J will be zero. As before, let $J_i \subset J$ be the subset of wires in layer i of P.

Our soundness argument proceeds layer-by-layer, starting from the input layer (i.e., layer 1) and progressing to the output layer (i.e., layer d, where d is the depth of P). Our goal establishes the following invariant: if the hash keys hk_1 and hk_2 are zero-fixing on J_i and J_{i+1} and the digest associated with the lower layer (i.e., layer i) is Matching, then the digest associated with the upper layer (i.e., layer $i + 1$) is also Matching. We provide a sketch of this step. For ease of exposition, suppose hk_1 is zero-fixing on J_i and the digest dig_1 is Matching. The goal is to show that hk_2 is zero-fixing on J_{i+1}, then the digest dig_2 is also Matching:[3]

- Initially, we set hk_2 to be binding on the empty set. We require in this case that dig_2 is always Matching. We refer to this property as an extractor validity property on the zero-fixing hash function.

- We now iteratively build up hk_2. Let $J_{i+1}[1]$ be the first element of J_{i+1}. We set hk_2 to be binding on the set $\{J_{i+1}[1]\}$. Our goal is to argue that dig_2 is still Matching. While it might seem like this property should follow

[3] This step is straightforward if we had a predicate-extractable hash function where the extractor outputs a mismatching index. Namely, if the upper layer digest is NotMatching, then the extractor outputs an index $j \in J_{i+1}$ that is mismatching (i.e., cannot be opened to a 0). This means the efficient adversary can only open wire j to the value 1. Now, if the BARG is extracting on the statement associated with wire j, then we either (1) obtain the opening of some index $j' \in J_i$ to a 1, which breaks security of the hash function (since the lower layer digest is Matching); or (2) the value of wire j is inconsistent with the input wires associated with the gate computing wire j, which breaks security of the BARG.

assuming a basic index hiding property on the zero-fixing hash function (i.e., that the hash key hk hides which set it is binding on), this is *insufficient*. The reason is that when hk_2 is binding on $\varnothing$, the adversary might output a Matching digest dig_2, but if hk_2 is binding on $\{J_{i+1}[1]\}$, the output digest dig_2 might be NotMatching. We cannot use such an adversary to construct an index hiding distinguisher, because in the index hiding security game, the distinguisher does *not* have the extraction trapdoor. As such, an attempted reduction algorithm cannot efficiently decide whether the adversary was successful or not. Indeed, this is a fundamental issue since knowledge of the extraction trapdoor would trivially break index hiding.

- To advance the proof, we introduce a *stronger* notion of index hiding security for zero-fixing hash functions, which essentially requires that no efficient adversary can output a digest dig that causes the output of Extract to differ depending on whether the hash key is binding on a set S or a set $S \setminus \{i\}$.[4] Of course, this is only meaningful when the digest is computed over an input that is 0 on index i.[5] Thus, we require this stronger index hiding with extracted guess property to hold only for digests dig where the adversary can provide an opening to index 0 for the target index i. We define this property formally in Definition 3.1.
- To leverage the index hiding with extracted guess property, we need to enforce the fact that dig_2 opens to a 0 on index $J_{i+1}[1]$. We ensure this by appealing to the somewhere extractability of the BARG along with zero-fixing security of the hash function. Specifically, suppose that the BARG is binding on wire $J_{i+1}[1]$. The BARG extractor then produces openings to the wire $J_{i+1}[1]$ with respect to dig_2 as well as opening to the wires $j_{\mathrm{L}}, j_{\mathrm{R}}$ with respect to dig_1 (corresponding to the input wires for the gate computing $J_{i+1}[1]$). Since dig_1 is zero-fixing on J_i and dig_1 is also Matching, if either $j_{\mathrm{L}}, j_{\mathrm{R}} \in J_i$, then the extracted openings must be openings to 0 (otherwise, we break zero-fixing of the hash function). But by monotonicity of P, this means the value of the output wire $J_{i+1}[1]$ must also be 0, and thus the BARG extractor produces an opening to 0 for wire $J_{i+1}[1]$. Now, by the index hiding with extracted guess property, we conclude that programming hk_2 to zero-fix on set $\{J_{i+1}[1]\}$ will still cause dig_2 to be Matching (except with a negligible loss in probability).
- We can now iteratively apply the argument and build up hk_2 until it is binding on all of J_{i+1}.

To complete the proof, we consider the input and output layers for P:

- **Handling the input layers:** The base case in our analysis is to show that if hk_1 is binding on J_1 (the input layer), then it is Matching. This follows using the same layer-wise strategy sketched above for proving our invariance, except for each index $J_1[i]$, we rely on the fact that the associated statement x_i is false (i.e., no witness exists) to argue that the only valid opening for dig_1 on index i is 0. Otherwise, we either break somewhere extractability of the BARG (i.e., extracting an invalid witness for index i) or the index hiding with extracted guess property.
- **Output layer:** Starting from the input layer, we now iteratively apply our basic invariant to argue that when the hash keys are binding to J_d (the output layer), the associated digests are also Matching. Now, if we have a valid proof, and the BARG is set to extract on the output layer, then the BARG extractor outputs an opening of the output wire to 1 with respect to the hash digests. However, since the output wire is contained in J_d (since the statement is false), and the digest is matching, this breaks zero-fixing security of the hash function.

Thus, the above analysis suffices to show non-adaptive soundness of our construction. The critical security requirement we require on our zero-fixing hash function is the strengthened index hiding with extracted guess property. This property allows us to complete the proof via an iterative approach without needing to rely on the extractor outputting a mismatching index as in previous work [BBK+23]. As we discuss below, this is an easier property to realize than full-fledged index extraction. We refer to Section 3 for the formal definition of zero-fixing hash functions and Section 4 for our construction of monotone policy BARGs.

[4] This type of property where the output of the extractor does not change for different choices of the CRS is often referred to as a "no-signaling" extraction property [PR17, KPY19, GZ21, KVZ21, CJJ21b].

[5] Otherwise, an honest digest on the input that is 1 at index i (and 0 everywhere else) would be declared Matching if the hash key was zero-fixing on a set S that contains i and NotMatching if the hash key was zero-fixing on the set $S \setminus \{i\}$

Constructing zero-fixing hash functions. Our second contribution in this work is a generic construction of zero-fixing hash functions from vanilla BARGs together with an additively homomorphic encryption scheme. We start with a basic construction that captures the key ideas underlying our construction and refer to Section 5 for the formal description and analysis:

- Let $n \in \mathbb{N}$ be the input length. For ease of exposition, we assume that $n = 2^k$ is a power-of-two. Suppose we want to zero-fix on a (possibly-empty) set $S \subseteq [n]$. The setup algorithm first samples a public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$ for an additively homomorphic encryption scheme. For each $i \in [n]$, the setup algorithm construct an encryption $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, 1)$ of 1 if $i \in S$ and an encryption of $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$ of 0 if $i \notin S$. It also constructs an encryption $\mathsf{ct}_{\mathsf{zero}} \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$ of 0. Finally, it constructs a commitment $\mathsf{com}_{\mathsf{hk}}$ to the ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$. The hash key is then $\mathsf{hk} = (\mathsf{pk}, \mathsf{ct}_{\mathsf{zero}}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n, \mathsf{com}_{\mathsf{hk}})$, and the extraction trapdoor is the decryption key sk.

- To hash an input $x \in \{0,1\}^n$, the user constructs a complete binary tree where each of the n leaves is associated with a ciphertext. If $x_i = 1$, then the user associates leaf i with ct_i, and if $x_i = 0$, then the user associates leaf i with $\mathsf{ct}_{\mathsf{zero}}$. The value of each internal node in the binary tree is defined to be the sum of the ciphertexts associated with its two children. By construction, the value of the root node is an encryption of the sum of the values associated with the n leaf nodes. We refer to the tree of ciphertexts as the "ciphertext-evaluation tree." The digest dig then consists of the ciphertext $\mathsf{ct}_{\mathsf{root}}$ associated with the root node along with a commitment $\mathsf{com}_{\mathsf{ct}}$ to all of the ciphertexts in the ciphertext-evaluation tree.

- A local opening for index i^* and value $b_{i^*} \in \{0,1\}$ for the digest $\mathsf{dig} = (\mathsf{ct}_{\mathsf{root}}, \mathsf{com}_{\mathsf{ct}})$ is a BARG proof. The BARG statements correspond to the indices of the nodes in the ciphertext-evaluation tree. The associated relation is parameterized by the target index i^*, the root ciphertext $\mathsf{ct}_{\mathsf{root}}$, the encryption $\mathsf{ct}_{\mathsf{zero}}$ of 0 from the hash key, and the commitment to the input ciphertexts $\mathsf{com}_{\mathsf{hk}}$. The BARG relation then checks the following:
  - **Leaf nodes:** For each leaf node i, $\mathsf{com}_{\mathsf{ct}}$ opens to either $\mathsf{ct}_{\mathsf{zero}}$ or ct_i at index i. For the particular index i^*, it checks that $\mathsf{com}_{\mathsf{ct}}$ opens to $\mathsf{ct}_{\mathsf{zero}}$ if $b_{i^*} = 0$ and $\mathsf{com}_{\mathsf{ct}}$ opens to ct_{i^*} if $b_{i^*} = 1$. Since the BARG relation only has $\mathsf{com}_{\mathsf{hk}}$ and not ct_i itself, the prover provides ct_i as part of its witness along with a proof of opening for ct_i with respect to $\mathsf{com}_{\mathsf{hk}}$. The proof of opening ensures that the correct ct_i is provided.
  - **Internal nodes:** For an internal node i (with children indexed j_{L}, j_{R}), the BARG checks that $\mathsf{com}_{\mathsf{ct}}$ opens to ciphertexts ct_i, $\mathsf{ct}_{j_{\mathrm{L}}}$, $\mathsf{ct}_{j_{\mathrm{R}}}$ where ct_i is the sum of ciphertexts $\mathsf{ct}_{j_{\mathrm{L}}}$ and $\mathsf{ct}_{j_{\mathrm{R}}}$.
  - **Root node:** For the root node, the BARG checks that $\mathsf{com}_{\mathsf{ct}}$ opens to $\mathsf{ct}_{\mathsf{root}}$.

- To test whether a digest $\mathsf{dig} = (\mathsf{ct}_{\mathsf{root}}, \mathsf{com}_{\mathsf{ct}})$ is matching or not, the Extract algorithm outputs Matching if $\mathsf{ct}_{\mathsf{root}}$ decrypts to 0 and NotMatching otherwise.

By definition, the ciphertext $\mathsf{ct}_{\mathsf{root}}$ in any (honestly-generated) hash digest is the sum of the ciphertexts associated with the leaves of the ciphertext-evaluation tree. On an input x, if $x_i = 0$, then the associated ciphertext is an encryption of 0 and does not contribute to the sum. If $x_i = 1$, then the ciphertext associated with the leaf is an encryption of 1 if $i \in S$ and encryption of 0 otherwise. Thus, the sum is only incremented if $x_i = 1$ for some $i \in S$. This is precisely when Extract outputs NotMatching (i.e., the digest is for an input x where $x_i = 1$ for $i \in S$).

To argue that it is hard to open a Matching, but possibly-malformed digest to a 1 at an index $i \in S$, we appeal to soundness of the BARG. In this case, the root ciphertext $\mathsf{ct}_{\mathsf{root}}$ in dig decrypts to a non-zero value, and yet the user constructed a valid BARG proof of opening for an index $i \in S$. The key observation is that the structure of the BARG used in the above construction is very similar to the structure of the canonical protocol from [BBK+23] described at the beginning of Section 1.2 for demonstrating correct evaluation of a monotone circuit. Moreover, because the ciphertext-evaluation tree is *perfectly* balanced, it has depth $\log n$, where $n = \mathsf{poly}(\lambda)$ is the input length. As such, we are able to adapt the proof strategy for arguing soundness of the monotone policy BARGs for *log-depth circuits* to directly argue zero-fixing security of our hash function. Specifically, we rely on BARG security to ensure that if the adversary uses an encryption of 1 as one of the leaves to the ciphertext (which it must if it opens an index $i \in S$ to a 1), then the root ciphertext necessarily is an encryption of a non-zero value. We provide the full details in Section 5.1.3.

While the core construction described here satisfies zero-fixing security, we need to augment the construction to satisfy the additional security requirements we impose on a zero-fixing hash function. We summarize these here, and defer to the technical sections (Sections 5, 5.1.4 and 5.1.5) for the full details:

- **Extractor validity:** Recall that this property says that when the hash function is zero-fixing on the empty set, it should be hard for an adversary to come up with a "valid" digest that is NotMatching. To satisfy this property, we simply include a BARG proof of validity to the digest, where the BARG proof of validity simply checks that the ciphertext-evaluation tree was correctly constructed. When the hash key is binding to the empty set, all of the ciphertexts ct_i are an encryption of 0, so the root of a properly computed ciphertext-evaluation tree will also be an encryption of 0. We provide the details in Section 5.1.4.

- **Index hiding with extracted guess:** Recall that this property says that the adversary cannot produce a digest dig where the extractor output disagrees depending on whether the hash key is zero-fixing on a set S or a set $S \setminus \{i\}$ (provided that the adversary provides an opening to 0 for index i). The only difference between the hash keys in these two cases is ct_i in the CRS changes from an encryption of 0 to an encryption of 1, which we could in principle show using semantic security. However, the reduction algorithm would have no way of checking whether a digest dig output by the adversary is Matching or NotMatching (since it does *not* and cannot know the decryption key). Thus, to argue this we adopt a Naor-Yung type of strategy [NY90] and *encrypt* twice. Namely, we introduce two *parallel* copies of the scheme (i.e., two independent public keys and two independent sets of ciphertexts). The digest now consists of two ciphertexts $\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}$ for the roots of the two ciphertext-evaluation trees. The same BARG would validate both roots. The key idea now is we can switch $\mathsf{ct}_i^{(0)}$ from an encryption of 0 (i.e., zero-fixing at $S \setminus \{i\}$) to an encryption of 1 (i.e., zero-fixing at S) while being able to decrypt (i.e., extract) for the parallel encryption scheme. We can leverage soundness of the BARG to argue that for a valid digest/opening, both $\mathsf{ct}_{\mathsf{root}}^{(0)}$ and $\mathsf{ct}_{\mathsf{root}}^{(1)}$ encrypt *identical* values. This allows us to leverage semantic security to switch the ciphertexts for one scheme while being able to detect whether the output of Extract changed or not (using knowledge of the secret key for the parallel scheme). We provide the full details in Section 5.1.5.

Taken together, we obtain a zero-fixing hash function from any standard BARG together with an additively-homomorphic encryption scheme. By instantiating with BARGs from the k-Lin assumption over pairing groups [WW22] or the (sub-exponential) DDH assumption over pairing-free groups [CGJ+23], we obtain zero-fixing hash functions from the same underlying assumptions. In conjunction with our generic construction from above, this yields Theorem 1.1.

An algebraic construction of zero-fixing hash functions. As another contribution, we also describe an algebraic approach to construct zero-fixing hash functions directly from (composite-order) bilinear maps. This construction has the advantage that it only makes black-box use of cryptography. We give a brief sketch of the construction here, but defer the details to Section 6. The basic version is an adaptation of the Catalano-Fiore vector commitment [CF13]:

- Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be a composite-order bilinear group of order N, generator g, and an efficiently-computable non-degenerate bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. In the actual construction, we will require that N be a product of six primes. In the description here, we will just describe the basic scheme that operates primarily in just two subgroups. Let g_1 and g_2 be generators of two orthogonal subgroups of $\mathbb{G}$.

- To sample a hash key for a set $S \subseteq [n]$, the setup algorithm samples exponents $\alpha_i, \beta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. If $i \in S$, it sets $A_i \leftarrow (g_1 g_2)^{\alpha_i}$ and if $i \notin S$, it sets $A_i \leftarrow g_1^{\alpha_i}$. It sets $B_i \leftarrow g_1^{\beta_i}$ and for $i \neq j$, it computes the cross term $C_{i,j} \leftarrow g_1^{\alpha_i \beta_j}$. The hash key then contains A_i, B_i for $i \in [n]$ and $C_{i,j}$ for all $i \neq j$.

- The hash of an input $x \in \{0,1\}^n$ is then $\mathsf{dig} = \prod_{i \in [n]} A_i^{x_i}$. The opening to an index i is $V = \prod_{j \neq i} C_{j,i}^{x_j}$. To verify an opening to a bit b at index i, the verifier checks
$$e(\mathsf{dig}, B_i) = e(A_i, B_i)^b \cdot e(g_1, V).$$

- To check whether a digest dig is Matching or not, the extraction algorithm output Matching if $e(\mathsf{dig}, g_2) = 1$ and NotMatching otherwise.

The basic principle is to move the "encoding elements" A_i for $i \in S$ to have a component in the span of g_2. The components A_i for $i \notin S$ are only in the span of g_1. Then, any digest that includes an index $i \in S$ will contain a non-zero element in the span of g_2, and thus, be declared $\mathsf{NotMatching}$. Arguing the security of this scheme is more delicate and will require introducing a number of additional randomizing components (and subgroups). We refer to Section 6 for the details.

Constructing monotone policy aggregate multi-signatures Our final contribution is a construction of monotone policy aggregate multi-signatures. While previous construction of aggregate signatures relied on extractable BARGs [WW22, DGKV22], a similar implication is possible by combining a non-adaptively-sound BARG together with a "puncturable signature" scheme (also called an all-but-one signature scheme) [GVW19]. We sketch our construction below, and provide the full details in Section 7.

In a puncturable signature scheme, it is possible to puncture a verification key on a message m^*. The property is that there does not *exist* signatures on m^* that verify with respect to the punctured verification key. Moreover, a punctured verification key is computationally indistinguishable from an honestly-generated verification key, even if the adversary is able to see signatures on arbitrary messages $m \neq m^*$. Goyal, Vusirikala, and Waters [GVW19] showed how to construct puncturable signatures from most standard number-theoretic assumptions (e.g., RSA, pairing-based assumptions, and LWE). We can use a non-adaptively-sound monotone policy BARG together with a puncturable signature scheme to construct a (statically-secure)[6] aggregate multi-signature scheme for any policy computed by a monotone Boolean circuit. We provide a sketch below:

- **Setup:** Consider a scheme with k signers. Each signer $i \in [k]$ has a signing key sk_i and a verification key vk_i for the punctured signature scheme. The public parameters of the aggregation scheme contain the common reference string for a monotone policy BARG.

- **Signing:** To sign a message m, each user signs with their individual signing key.

- **Aggregation:** Given a set of signatures $\{\sigma_i\}_{i \in S}$ on the same message m and a (monotone) aggregation policy P, a user can aggregate the signatures by giving a monotone policy BARG proof for the policy P with respect to the natural relation $\mathcal{R}[m] = \{(\mathsf{vk}, \sigma) : \mathsf{Verify}(\mathsf{vk}, m, \sigma)\}$. The aggregate signature is simply the BARG proof for the statements $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$ with the witness $(\sigma_1, \ldots, \sigma_k)$.

- **Verification:** To verify an aggregate multi-signature with respect to a policy P, the verifier just checks the BARG proof.

Note that one could also construct an aggregate multi-signature by sending the set S where $P(S) = 1$ and then use a vanilla BARG to prove knowledge of a signature σ_i for every $i \in S$. However, this approach would require communicating the set S as part of the aggregate signature. Using monotone policy BARGs, the aggregate signature only consists of the BARG proof, and thus has size, $\mathsf{poly}(\lambda, \log |P|)$. It is straightforward to prove static security of the above multi-signature scheme just assuming *non-adaptive-soundness* on the underlying BARG. We sketch the reduction below:

- In the static security game, the adversary has to pre-commit to the message m^* it wants to forge on, the set of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$ it wants to use (which can be a mix of honest verification keys chosen by the challenger and verification keys chosen adversarially), and the aggregation policy P *before* seeing the aggregation parameters.

- Let $S \subseteq [k]$ be the set of indices i where the chosen key vk_i^* is uncorrupted (i.e., chosen by the challenger). The admissibility requirement is that $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise; this is saying that the adversary cannot satisfy the policy P just by providing signatures under keys it controls.

- In the security reduction, we first puncture the honest users' verification keys vk_i on the challenge message m^*. This means that there does not exist valid signatures on the challenge message m^* with respect to the honest users' verification keys vk_i

[6] In the static security model, we require that the adversary declare the set of corrupted verification keys, its challenge message, and the aggregation policy at the beginning of the security game.

- Consider the relation $\mathcal{R}[m^*]$ used for verification. By definition of the set S and the fact that the honest verification keys are punctured at m^*, the statement $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$ is *false* for the policy P with respect to the relation $\mathcal{R}[m^*]$. By non-adaptive soundness of the monotone policy BARG, the probability that the adversary can produce a valid aggregate signature (i.e., a valid proof on a false statement) is negligible.

Observe that in the above sketch, the verification time is linear in k. However, using a RAM delegation scheme, we can achieve fast verification. We refer to Remark 7.8 for additional details.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. For $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For $a, b \in \mathbb{N}$ we write $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$. We write $\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in λ, and $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$. We say an event happens with overwhelming probability if its complement occurs with negligible probability. For a finite set S, we write $x \xleftarrow{\text{R}} S$ to denote that x is a uniformly random element of S. For a distribution $\mathcal{D}$ we write $x \leftarrow \mathcal{D}$ to denote that x is a random draw from $\mathcal{D}$.

We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. A non-uniform algorithm $\mathcal{A}$ consists of a pair of algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ is a (possibly-unbounded) algorithm that takes as input 1^λ and outputs an advice string ρ_λ of $\mathsf{poly}(\lambda)$ size. Algorithm $\mathcal{A}_2$ is an efficient algorithm. The output of $\mathcal{A}$ on an input $x \in \{0,1\}^\lambda$ is defined as first computing the advice string $\rho_\lambda \leftarrow \mathcal{A}_1(1^\lambda)$ and then outputting $\mathcal{A}_2(x, \rho_\lambda)$. We say two ensembles of distributions $\mathcal{D}_1 = \left\{\mathcal{D}_{1,\lambda}\right\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_2 = \left\{\mathcal{D}_{2,\lambda}\right\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if no efficient adversary can distinguish them with non-negligible probability. We say they are statistically indistinguishable if their statistical distance is bounded by $\mathsf{negl}(\lambda)$.

2.1 Cryptographic Building Blocks

In this section, we recall the definition of a few standard cryptographic building blocks we use in this work.

Additively-homomorphic encryption. We start by reviewing the notion of an additively homomorphic encryption. For our applications, it suffices to consider constructions that only support decryption of values residing in a bounded message space. Such additively homomorphic encryption schemes can be built from most standard number-theoretic assumptions that imply public-key encryption such as the decisional Diffie-Hellman (DDH) assumption [Gam84], decisional composite residuosity (DCR) [Pai99], or the learning with errors (LWE) assumption [Reg05].

Definition 2.1 (Additively Homomorphic Encryption)**.** An additively homomorphic encryption with bounded support is a tuple of polynomial time algorithms $\Pi_{\mathsf{HE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^n) \to (\mathsf{sk}, \mathsf{pk})$: On input a security parameter $\lambda \in \mathbb{N}$ and a range parameter $n \in \mathbb{N}$, the key-generation algorithm outputs a secret key sk and a public key pk. We assume that the secret key and the public key includes an implicit description of the range parameter 1^n.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg}) \to \mathsf{ct}$: On input a public key pk and an integer $\mathsf{msg} \in \{0, \ldots, n\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{msg}$: On input a secret key sk and a ciphertext ct, the decryption algorithm either outputs a plaintext $\mathsf{msg} \in \{0, \ldots, n\}$ or a special symbol $\mathsf{msg} = \bot$. The decryption algorithm is deterministic.
- $\mathsf{Add}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}'$: On input a public key pk and two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, the homomorphic addition algorithm outputs a new ciphertext ct'. The addition algorithm is *deterministic*.

We require the following properties:

- **Correctness:** For all $\lambda, n \in \mathbb{N}$ and all messages $\mathsf{msg} \in \{0, \ldots, n\}$, it holds that:

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

- **Evaluation correctness:** For all $\lambda, n \in \mathbb{N}$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{Gen}(1^\lambda, 1^n)$ and all ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ where $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) \neq \perp$, $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_2) \neq \perp$, and $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) + \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_2) \in \{0, \ldots, n\}$, it holds that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Add}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) + \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_2).$$

- **Compactness:** There exists a polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, n \in \mathbb{N}$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{Gen}(1^\lambda, 1^n)$, all ciphertexts ct in the support of $\mathsf{Enc}(\mathsf{pk}, \cdot)$ and $\mathsf{Add}(\mathsf{pk}, \cdot, \cdot)$, it holds that $|\mathsf{pk}| \leq \mathsf{poly}(\lambda + \log n)$ and $|\mathsf{ct}| \leq \mathsf{poly}(\lambda + \log n)$.

- **CPA-security:** For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, define the CPA-security experiment $\mathsf{ExptSS}_{\mathcal{A}}(\lambda, b)$ as follows:

  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ starts by outputting a range parameter 1^n.
  2. The challenger samples a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$ and sends pk to the adversary.
  3. The adversary can now make (arbitrarily many) queries on pairs of messages $(\mathsf{msg}_0, \mathsf{msg}_1)$. On each query, the challenger replies with $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg}_b)$.
  4. After the adversary $\mathcal{A}$ is done making queries, it outputs a guess $b' \in \{0, 1\}$.

  We say that Π_{HE} is semantically secure if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|\Pr[\mathsf{ExptSS}_{\mathcal{A}}(\lambda, 1) = 1] - \Pr[\mathsf{ExptSS}_{\mathcal{A}}(\lambda, 0) = 1]\right| = \mathsf{negl}(\lambda)$.

Fact 2.2 (Additively Homomorphic Encryption [Gam84, Pai99, Reg05])**.** Assuming any of (1) the decisional Diffie-Hellman assumption (DDH), (2) the decisional composite residuosity assumption (DCR), or (3) the learning with errors (LWE) assumption, there exists an additively homomorphic encryption scheme with a bounded support.

Vector commitments. Next, we recall the notion of a vector commitment scheme with succinct local openings. Such commitments can be built from any collision-resistant hash function [Mer87].

Definition 2.3 (Vector Commitment)**.** A vector commitment with local openings is a tuple of efficient algorithms $\Pi_{\mathsf{Com}} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^n, \ell) \rightarrow \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the block length $n \in \mathbb{N}$, and the vector length $\ell \in \mathbb{N}$ (in *binary*), the setup algorithm outputs a common reference string crs. We assume the common reference string implicitly contains the parameters 1^n and ℓ.

- $\mathsf{Commit}(\mathsf{crs}, (x_1, \ldots, x_t)) \rightarrow (\mathsf{com}, \sigma_1, \ldots, \sigma_t)$: On input the common reference string crs and a vector of $t \leq \ell$ messages $x_1, \ldots, x_t \in \{0, 1\}^n$, the commit algorithm outputs a commitment com and openings $\sigma_1, \ldots, \sigma_t$.

- $\mathsf{Verify}(\mathsf{crs}, \mathsf{com}, i, y, \sigma) \rightarrow b'$: On input the common reference string crs, the commitment com, an index $i \in [\ell]$, a message $y \in \{0, 1\}^n$, and an opening σ, the verification algorithm outputs a bit $b' \in \{0, 1\}$.

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

- **Correctness:** For all $\lambda, n, \ell \in \mathbb{N}$, and all positive $t \leq \ell$, all $x_1, \ldots, x_t \in \{0, 1\}^n$, and indices $i \in [t]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{com}, i, x_i, \sigma_i) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^\ell), \\ (\mathsf{com}, \sigma_1, \ldots, \sigma_t) \leftarrow \mathsf{Commit}(\mathsf{crs}, (x_1, \ldots, x_t)) \end{array}\right] = 1.$$

- **Computational binding:** For an adversary $\mathcal{A}$, define the computational binding experiment as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the block length 1^n and vector length ℓ.
  2. The challenger responds with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \ell)$.
  3. Algorithm $\mathcal{A}$ outputs a commitment com, an index $i \in [\ell]$, and openings (y_0, σ_0) and (y_1, σ_1).

4. The output of the experiment is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}, \mathsf{com}, i, y_0, \sigma_0) = 1 = \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, i, y_1, \sigma_1)$ and $y_0 \neq y_1$. Otherwise, the output is $b = 0$.

The commitment scheme is binding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the binding experiment.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, n, \ell \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^n, \ell)$, all $t \leq \ell$, and all $(\mathsf{com}, \sigma_1, \ldots, \sigma_t)$ in the support of $\mathsf{Commit}(\mathsf{crs}, \cdot)$, the following holds:
  - **Succinct CRS:** $|\mathsf{crs}| = \mathsf{poly}(\lambda + \log n + \log \ell)$.
  - **Succinct commitment:** $|\mathsf{com}| = \mathsf{poly}(\lambda + \log n + \log \ell)$.
  - **Succinct local opening:** For all $i \in [\ell]$, $|\sigma_i| = \mathsf{poly}(\lambda + \log n + \log \ell)$.

Fact 2.4 (Vector Commitments from Collision-Resistant Hash Functions [Mer87])**.** Assuming the existence of collision-resistant hash functions, there exists a vector commitment scheme with local openings.

2.2 Batch Arguments for NP

In this section, we recall the notion of a non-interactive batch argument (BARG) for NP, the special case of a BARG for index languages (i.e., an "index BARG" [CJJ21b]) and the notion of a BARG for monotone policy batch NP [BBK+23].

Batch arguments for NP. We begin with the notion of a somewhere extractable batch argument for NP. Our presentation is adapted from [CJJ21b, WW22]. Here, we provide a more general syntax where the batch arguments supports extraction on up to ℓ indices.

Definition 2.5 (Boolean Circuit Satisfiability)**.** We define the circuit satisfiability language $\mathcal{L}_{\mathsf{CSAT}}$ as

$$\mathcal{L}_{\mathsf{CSAT}} = \left\{ (C, x) \;\middle|\; \begin{array}{c} C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}, x \in \{0,1\}^n \\ \exists w \in \{0,1\}^* : C(x, w) = 1 \end{array} \right\}.$$

Definition 2.6 (Non-Interactive Batch Argument)**.** A somewhere-extractable non-interactive batch argument (BARG) for Boolean circuit satisfiability is a tuple of efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{TrapGen}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell) \to (\mathsf{crs}, \mathsf{vk})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$, the instance length $n \in \mathbb{N}$, a bound on the size of the Boolean circuit $s \in \mathbb{N}$, and a bound on the size of the extraction set $\ell \in \mathbb{N}$, the generator algorithm outputs a common reference string crs and a verification key vk.

- $\mathsf{Prove}(\mathsf{crs}, C, (x_1, \ldots, x_k), (w_1, \ldots, w_k)) \to \pi$: On input the common reference string crs, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^k$, and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$, the prove algorithm outputs a proof π.

- $\mathsf{Verify}(\mathsf{vk}, C, (x_1, \ldots, x_k), \pi) \to b$: On input the verification key vk, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$ and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

- $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell, S) \to (\mathsf{crs}, \mathsf{vk}, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$, the instance size $n \in \mathbb{N}$, a bound on the size of the Boolean circuit $s \in \mathbb{N}$, a bound on the size of the extraction set $\ell \in \mathbb{N}$, and a set $S \subseteq [k]$ of size at most ℓ, the trapdoor generator algorithm outputs a common reference string crs, a verification key vk and an extraction trapdoor td.

- $\mathsf{Extract}(\mathsf{td}, C, (x_1, \ldots, x_k), \pi, i) \to w$. On input the trapdoor td, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a collection of statements $x_1, \ldots, x_k \in \{0,1\}^n$, a proof π and an index $i \in [k]$, the extraction algorithm outputs a witness w.

For notational convenience, when $\ell = 1$, we omit the final input 1^ℓ and instead, write $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s)$ to denote $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^1)$. Similarly, we also write $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, i)$ to denote $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{i\})$. Finally, we require that Π_{BARG} satisfy the following properties:

- **Completeness:** For all $\lambda, k, n, s, \ell \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, all statements $x_1, \ldots, x_k \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$ where $C(x_i, w_i) = 1$ for all $i \in [k]$,
$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, C, (x_1, \ldots, x_k), \pi) = 1 : \begin{array}{c} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell), \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (x_1, \ldots, x_k), (w_1, \ldots, w_k)) \end{array}\right] = 1.$$

- **Set hiding:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, define the set hiding experiment $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, b)$ as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the number of instances 1^k, the instance size 1^n, the bound on the circuit size 1^s, the bound on the size of the extraction set 1^ℓ, and a set $S \subseteq [k]$ of size at most ℓ.
  2. If $b = 0$, the challenger gives $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell)$ to $\mathcal{A}$. If $b = 1$, the challenger samples $(\mathsf{crs}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell, S)$ and gives $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  Then, Π_{BARG} satisfies set hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that
  $$\left|\Pr[\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 1) = 1]\right| = \mathsf{negl}(\lambda).$$
  When $\ell = 1$, we might refer to this property as *index hiding*.

- **Somewhere extractable in trapdoor mode:** For an adversary $\mathcal{A}$, define the somewhere extractable security game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the number of instances 1^k, the instance size 1^n, the bound on the circuit size 1^s, a bound on the size of the extraction set 1^ℓ, and a nonempty set $S \subseteq [k]$ of size at most ℓ.
  2. The challenger samples $(\mathsf{crs}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell, S)$ and gives $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, statements $x_1, \ldots, x_m \in \{0,1\}^n$, and a proof π.
  4. The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{vk}, C, (x_1, \ldots, x_m), \pi) = 1$ and there exists an index $i \in S$ for which $C(x_i, w_i) \neq 1$ where $w_i \leftarrow \mathsf{Extract}(\mathsf{td}, C, (x_1, \ldots, x_k), \pi, i)$. Otherwise, the output is $b = 0$.

  Then Π_{BARG} is somewhere extractable in trapdoor mode if for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the somewhere extractable game.

- **Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, k, n, s, \ell \in \mathbb{N}$, all crs in the support of $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell)$, and all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, the following properties hold:

  - **Succinct proofs:** The proof π output by $\mathsf{Prove}(\mathsf{crs}, C, \cdot, \cdot)$ satisfies $|\pi| \le \mathsf{poly}(\lambda + \log k + s + \ell)$.
  - **Succinct CRS:** $|\mathsf{crs}| \le \mathsf{poly}(\lambda + k + n + \ell) + \mathsf{poly}(\lambda + \log k + s + \ell)$.
  - **Succinct verification key:** $|\mathsf{vk}| \le \mathsf{poly}(\lambda + \log k + s + \ell)$.

Fact 2.7 (Batch Arguments for NP [CJJ21b, WW22, KLVW23, CGJ+23]). Assuming any of (1) the plain LWE assumption, (2) the k-Lin assumption over pairing groups for any constant $k \in \mathbb{N}$, or (3) the (sub-exponential) DDH assumption in pairing-free groups, there exists a non-interactive batch argument for NP.

Set hiding with extraction. For our main construction (Section 5), we require a slight strengthening of the somewhere extractability property from Definition 2.6. Our stronger set-hiding property essentially says that if the extraction key is programmed to extract either on $S_0 \subseteq [k]$ or $S_1 \subseteq [k]$, then the extracted witness on "common indices" $i^* \in S_0 \cap S_1$ is computationally indistinguishable in the two cases. This type of property is often referred to as a "no-signaling" extraction property [PR17, KPY19, GZ21, KVZ21, CJJ21b]. We define this formally below and show that it follows generically from the standard vanilla extractability in Appendix A.

Definition 2.8 (Set Hiding with Extraction)**.** Let $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{TrapGen}, \mathsf{Extract})$ be a somewhere extractable batch argument for Boolean circuit satisfiability (Definition 2.6). For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, define the set hiding with extraction experiment $\mathsf{ExptSHwE}_{\mathcal{A}}(\lambda, b)$ as follows:

1. On input the security parameter λ, algorithm $\mathcal{A}$ starts by outputting the number of instances 1^k, the instance length 1^n, the bound on the circuit size 1^s, the bound on the extraction set 1^ℓ, a set $S \subseteq [k]$ of size at most ℓ, and an index $i^* \in S$.
2. If $b = 0$, the challenger samples $(\mathsf{crs}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell, S)$. If $b = 1$, the challenger samples $(\mathsf{crs}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell, \{i^*\})$. The challenger replies to $\mathcal{A}$ with $(\mathsf{crs}, \mathsf{vk})$.
3. Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof π.
4. If $\mathsf{Verify}(\mathsf{vk}, C, (x_1, \ldots, x_k), \pi) \neq 1$, then the experiment halts with output 0. Otherwise, the challenger replies with $w^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (x_1, \ldots, x_k), \pi, i^*)$.
5. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

Then, Π_{BARG} satisfies set hiding with extraction if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

Index BARGs. An index BARG [CJJ21b] is a batch argument for the batch index language where the instance is *always* the tuple $(1, \ldots, k)$. Since the statements are the integers, they have a succinct description, so we can impose a stronger requirement on the running time of the Verify algorithm. We define this below:

Definition 2.9 (Index BARG [CJJ21b])**.** An index BARG is a special case of a BARG where the instances $(x_1, \ldots, x_k)$ are restricted to the integers $(1, \ldots, k)$. In this setting, the Gen algorithm to the index BARG does *not* separately take in the instance length n as a separate input. Moreover, instead of providing $x_1, \ldots, x_k$ as input to the Prove, Verify, and Extract algorithms, we just give the single index k (in binary). Moreover, we require the *additional* succinctness property on the running time of Verify:

- **Succinct verification time:** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, k, n, s, \ell \in \mathbb{N}$, all $(\mathsf{crs}, \mathsf{vk})$ in the support of $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell)$ and all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, the running time of $\mathsf{Verify}(\mathsf{vk}, C, k, \cdot)$ is bounded by $\mathsf{poly}(\lambda + \log k + s + \ell)$.

Monotone policy BARG. Next, we recall the notion of a SNARG for monotone policy BatchNP [BBK+23], which we refer to more succinctly as a "monotone policy BARG." In this work, we just focus on the simplest notion of non-adaptive soundness.

Definition 2.10 (Monotone Policy BatchNP)**.** A Boolean circuit $P \colon \{0,1\}^k \to \{0,1\}$ is a monotone Boolean policy if P is a Boolean circuit comprised entirely of AND and OR gates. Let $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be a Boolean circuit and $P \colon \{0,1\}^k \to \{0,1\}$ be a monotone Boolean policy. We define the monotone policy BatchNP language $\mathcal{L}_{\mathsf{MP\text{-}CSAT}}$ to be

$$\mathcal{L}_{\mathsf{MP\text{-}CSAT}} = \left\{ (C, P, x_1, \ldots, x_k) \;\middle|\; \begin{array}{c} C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}, P \colon \{0,1\}^k \to \{0,1\}, x_1, \ldots, x_k \in \{0,1\}^n \\ \exists w_1, \ldots, w_k \in \{0,1\}^h : P\big(C(x_1, w_1), \ldots, C(x_k, w_k)\big) = 1 \end{array} \right\}. \tag{2.1}$$

Definition 2.11 (Monotone Policy BARG [BBK+23, adapted])**.** A monotone policy BARG is a triple of efficient algorithms $\Pi_{\mathsf{MP\text{-}BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the instance size $n \in \mathbb{N}$, a bound on the size of the Boolean circuit $s_c \in \mathbb{N}$, and a bound on the size of the policy $s_p \in \mathbb{N}$, the generator algorithm outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), (w_1, \ldots, w_k)) \to \pi$: On input the common reference string crs, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a monotone Boolean policy $P \colon \{0,1\}^k \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), \pi) \to b$: On input the common reference string crs, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a monotone Boolean policy $P \colon \{0,1\}^k \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

Moreover, $\Pi_{\mathsf{MP\text{-}BARG}}$ should satisfy the following properties:

- **Completeness:** For all $\lambda, n, s_c, s_p \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, all monotone Boolean policies $P \colon \{0,1\}^k \to \{0,1\}$ of size at most s_p, all statements $x_1, \ldots, x_k \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$ where $P(C(x_1, w_1), \ldots, C(x_k, w_k)) = 1$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), (w_1, \ldots, w_k)) \end{array}\right] = 1.$$

- **Non-adaptive soundness:** For any adversary $\mathcal{A}$, define the non-adaptive soundness game as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the instance size 1^n, the bound on the size of the NP relation 1^{s_c}, the bound on the size of the policy 1^{s_p}, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, a monotone Boolean circuit $P \colon \{0,1\}^k \to \{0,1\}$ of size at most s_p, and statements $x_1, \ldots, x_k \in \{0,1\}^n$.
  2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$ and gives it to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a proof π.
  4. The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), \pi) = 1$ and $(C, P, (x_1, \ldots, x_k)) \notin \mathcal{L}_{\mathsf{MP\text{-}CSAT}}$.

  We say that $\Pi_{\mathsf{MP\text{-}BARG}}$ is non-adaptively sound if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the non-adaptive soundness game.

- **Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, n, s_c, s_p \in \mathbb{N}$, all crs in the support of $\mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, and all monotone Boolean policies $P \colon \{0,1\}^k \to \{0,1\}$ of size $|P| \le s_p$, the following properties hold:
  - **Slightly succinct proofs:** The proof π output by $\mathsf{Prove}(\mathsf{crs}, C, P, \cdot, \cdot)$ satisfies $|\pi| \le \mathsf{poly}(\lambda + s_c + \log s_p)$.
  - **Succinct proofs:** The proof π output by $\mathsf{Prove}(\mathsf{crs}, C, P, \cdot, \cdot)$ satisfies $|\pi| \le \mathsf{poly}(\lambda + s_c + \log |P|)$.

Remark 2.12 (Slightly Succinct Proofs to Succinct Proofs)**.** In a "slightly succinct" proof system, the size of the proof scales logarithmically with the bound s_p on the size of the policy circuit, rather than the size of the policy circuit itself. It is straightforward to transform a scheme with slightly succinct proofs into one that satisfies the standard notion of succinctness. We use a "powers-of-two" construction. Namely, we generate $\ell = \lceil \log s_p \rceil$ different common reference strings, where the i^{th} CRS supports policies of size at most 2^i. The prover and verifier will use the CRS for scheme i when proving or verifying statements with respect to policies of size between 2^{i-1} and 2^i. In this case, the size of the proof scales polylogarithmically with the size of the policy P rather than the bound s_p. This approach only incurs logarithmic overhead in the CRS size. In the rest of this work, we will focus on constructions satisfying the simpler requirement of having slightly succinct proofs.

Remark 2.13 (Short Verification Key via RAM Delegation)**.** In Definition 2.11, the setup algorithm outputs a single CRS that is used both for generating proofs and for verifying proofs. The size of the CRS is allowed to grow with the size of both the circuit C and the size of the monotone policy P. It is possible to obtain a construction with a short verification key (that grows polylogarithmically with $|C|$ and $|P|$) by "delegating" the verification process using a RAM delegation scheme [CJJ21b, WW22, KLVW23, CGJ+23]. In this case, the verification key would be a succinct hash of the actual CRS. Since this provides a generic approach for realizing a short verification key, in our constructions, we will not explicitly decompose the CRS into a proving key and a separate short verification key. A similar approach was also used in [BBK+23] in their construction of predicate-extractable hash functions.

3 Zero-Fixing Hash Functions

In this section, we formally introduce the notion of a zero-fixing hash function. As we show in Section 4, we can combine a zero-fixing hash function with a vanilla BARG to obtain a monotone policy BARG. Recall from Section 1.2 that a zero-fixing hash function is a keyed hash function that supports succinct local openings. Moreover, the hash key is associated with a set of indices $S \subseteq [n]$, where n is the input length. Moreover, there is a trapdoor td associated with the hash key hk that can be used to decide whether a hash digest dig is $\mathsf{Matching}$ or $\mathsf{NotMatching}$ on the set S. The zero-fixing security requirement then says that if the extractor outputs $\mathsf{Matching}$ for a digest dig, it must be computationally hard to open dig to a 1 on any index $i \in S$.

As discussed in Section 1.2, our zero-fixing hash function is similar to the predicate-extractable hash function for bit-fixing predicates from [BBK+23]. A key distinction is that when the extraction algorithm outputs $\mathsf{NotMatching}$, the predicate-extractable hash function also outputs an index $i \in [n]$ where it is computationally infeasible to open the digest to a 1. In contrast, with our zero-fixing hash function, the extraction algorithm only outputs a single $\mathsf{Matching}$ or $\mathsf{NotMatching}$ flag. At the same time, we require our zero-fixing hash functions to satisfy additional security requirements that were not required in [BBK+23]. These additional security properties are necessary for our construction of monotone policy BARGs (Section 4). We now give the formal definition:

Definition 3.1 (Zero-Fixing Hash Function)**.** A *zero-fixing hash function* is a tuple of polynomial-time algorithms $\Pi_{\mathsf{H}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{ProveOpen}, \mathsf{VerOpen}, \mathsf{Extract}, \mathsf{ValidateDigest})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n, S) \to (\mathsf{hk}, \mathsf{vk}, \mathsf{td})$: On input a security parameter λ, an input length n, and a set $S \subseteq [n]$, the setup algorithm outputs a hash key hk, a verification key vk and a trapdoor td. We implicitly assume that hk includes λ and n.

- $\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{dig}$: On input a hash key hk and a string $x \in \{0,1\}^n$, the hash algorithm outputs a digest dig. This algorithm is deterministic.

- $\mathsf{ValidateDigest}(\mathsf{vk}, \mathsf{dig}) \to b$: On input a hash key vk and a digest dig, the digest validation algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.

- $\mathsf{ProveOpen}(\mathsf{hk}, x, i) \to \sigma$: On input a hash key hk, a string $x \in \{0,1\}^n$ and an index $i \in [n]$, the prove algorithm outputs an opening σ.

- $\mathsf{VerOpen}(\mathsf{vk}, \mathsf{dig}, i, b, \sigma) \to b'$: On input a hash key vk, a digest dig, an index $i \in [n]$, a bit $b \in \{0,1\}$ and an opening σ, the verification algorithm outputs a bit $b' \in \{0,1\}$. The verification algorithm is deterministic.

- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) \to m$: On input a trapdoor td and a digest dig, the extraction algorithm outputs a value $m \in \{\mathsf{Matching}, \mathsf{NotMatching}\}$. This algorithm is deterministic.

We require Π_{H} satisfy the following efficiency and correctness properties:

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all parameters $\lambda, n \in \mathbb{N}$, all $(\mathsf{hk}, \mathsf{vk}, \mathsf{td})$ in the support of $\mathsf{Setup}(1^\lambda, 1^n, \cdot)$, all inputs $x \in \{0,1\}^n$ and all indices $i \in [n]$, the following properties hold:
  - **Succinct verification key:** $|\mathsf{vk}| \le \mathsf{poly}(\lambda + \log n)$.

- **Succinct digest:** The digest dig output by $\mathsf{Hash}(\mathsf{hk}, x)$ satisfies $|\mathsf{dig}| \leq \mathsf{poly}(\lambda + \log n)$.
  - **Succinct openings:** The opening σ output by $\mathsf{ProveOpen}(\mathsf{hk}, x, i)$ satisfies $|\sigma| \leq \mathsf{poly}(\lambda + \log n)$.
  - **Succinct verification:** The running time of $\mathsf{VerOpen}(\mathsf{vk}, \cdot, \cdot, \cdot, \cdot)$ is $\mathsf{poly}(\lambda + \log n)$.

- **Correctness:** For all $\lambda, n \in \mathbb{N}$, every $x \in \{0,1\}^n$, and every $i \in [n]$, the following properties hold:
  - **Opening correctness:**

$$\Pr\left[\ \mathsf{VerOpen}(\mathsf{vk}, \mathsf{dig}, i, x_i, \sigma) = 1\ :\ \begin{array}{c}(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \varnothing) \\ \mathsf{dig} \leftarrow \mathsf{Hash}(\mathsf{hk}, x), \sigma \leftarrow \mathsf{ProveOpen}(\mathsf{hk}, x, i)\end{array}\ \right] = 1.$$

  - **Digest correctness:**

$$\Pr\left[\mathsf{ValidateDigest}(\mathsf{vk}, \mathsf{dig}) = 1 : (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \varnothing), \mathsf{dig} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)\right] = 1.$$

We additionally require the following security properties:

- **Set hiding:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the set hiding game $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, b)$ as follows:
  1. On input 1^λ, the adversary $\mathcal{A}$ outputs 1^n and a set $S \subseteq [n]$.
  2. If $b = 0$, the challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \varnothing)$ and if $b = 1$, the challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, S)$. It gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

  The hash function satisfies set binding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

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

- **Index hiding with extracted guess:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the index hiding with extracted guess game $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, b)$ as follows:
  1. On input 1^λ, algorithm $\mathcal{A}$ outputs 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$.
  2. If $b = 0$, the challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, S \setminus \{i^*\})$. Otherwise, it samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, S)$. The challenger sends $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a digest dig and an opening σ.
  4. The output of the experiment is 1 if $\mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, i^*, 0, \sigma) = 1$ and $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ outputs $\mathsf{Matching}$. Otherwise, the output is 0.

  The hash function satisfies index hiding with extracted guess if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

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

- **Zero fixing:** For an adversary $\mathcal{A}$, we define the adaptive zero-fixing game $\mathsf{ExptZF}_{\mathcal{A}}(\lambda)$ as follows:
  1. On input 1^λ, algorithm $\mathcal{A}$ outputs 1^n and a set $S \subseteq [n]$.
  2. The challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, S)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a digest dig, an index $i \in S$ and an opening σ.
  4. The output of the experiment is 1 if $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ outputs $\mathsf{Matching}$ and $\mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, i, 1, \sigma) = 1$. Otherwise, the output is 0.

  The hash function satisfies zero-fixing if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathsf{ExptZF}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda)$.

- **Extractor validity:** For an adversary $\mathcal{A}$, we define the extractor validity game $\mathsf{ExptEV}_{\mathcal{A}}(\lambda)$ as follows:
  1. On input 1^λ, the adversary $\mathcal{A}$ outputs 1^n.
  2. The challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \varnothing)$ and sends $(\mathsf{hk}, \mathsf{vk})$ to the adversary.
  3. Algorithm $\mathcal{A}$ outputs a digest dig.
  4. The output of the experiment is 1 if $\mathsf{ValidateDigest}(\mathsf{hk}, \mathsf{dig}) = 1$ and $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = \mathsf{NotMatching}$. Otherwise, the output is 0.

  The hash function satisfies the extractor validity property if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathsf{ExptEV}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda)$.

Remark 3.2 (Selective Zero-Fixing Security)**.** We can define a weaker selective notion of zero-fixing security where the adversary outputs the index $i \in S$ at the beginning of the security game (i.e., before seeing hk and vk). Note that the selective zero-fixing security definition is equivalent to the zero-fixing definition in Definition 3.1. To see that selective zero fixing implies standard zero-fixing, consider a reduction algorithm that guesses the index $i \xleftarrow{\text{R}} S$ at the beginning of the security reduction and aborts whenever the guess is incorrect. This reduction succeeds with probability $1/|S|$; since $|S| = \mathsf{poly}(\lambda)$, this incurs only a polynomial loss in advantage. In our construction (Construction 4.4), we will work with the adaptive notion of security, but in our constructions (Constructions 5.2 and 6.3), we will work with the simpler selective definition.

One-sided index hiding. For our application, it suffices to consider a weaker notion of "one-sided" index hiding where we only require that the adversary's advantage cannot increase (but could decrease). Proving one-sided security is often easier than proving two-sided security, so we define the simpler notion here:

Definition 3.3 (One-Sided Index-Hiding with Extracted Guess)**.** We say a zero-fixing hash function Π_{H} satisfies *one-sided* index-hiding with extracted guess security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 1) = 1] \geq \Pr[\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 0) = 1] - \mathsf{negl}(\lambda).$$

4 Constructing Monotone Policy BARGs

In this section, we describe how to construct monotone policy BARGs from a standard batch argument for NP together with a zero-fixing hash function. We start by defining the conventions we use for describing Boolean circuits.

Definition 4.1 (Monotone Circuit Wire Indexing)**.** Let $P \colon \{0,1\}^k \to \{0,1\}$ be a monotone Boolean circuit consisting exclusively of AND and OR gates with fan-in two. Let s be the size of P (i.e., the number of wires in P). A topological indexing of the wires of C is an assignment of an index $i \in [s]$ to each wire in P with the following properties:

- **Input wire:** For $i \in [k]$, the i^{th} input to P is associated with the index i.
- **Output wire:** The output wire is associated with the index s.
- **Intermediate wires:** The intermediate wires are associated with an index $i \in \{k+1, \ldots, s-1\}$ with the property that the value of index i is completely determined by the values of the wires with indices $j_{i,1}, j_{i,2} \in \{1, \ldots, i-1\}$.

Every monotone circuit P has a canonical topological indexing that can be computed efficiently (e.g., by applying a deterministic topological sort to the wires of P).

Definition 4.2 (Layered Monotone Circuit)**.** Let $P \colon \{0,1\}^k \to \{0,1\}$ be a (monotone) Boolean circuit of size s. We denote by $L_P(i)$ the *layer* of the wire i and define it as follows:

- If $i \in [k]$ (i.e., an input wire), then $L_P(i) = 1$.

- If $i > k$ then $L_P(i) = 1 + \max\{L_P(j_{i,1}), L_P(j_{i,2})\}$, where $j_{i,1}, j_{i,2} \in \{1, \ldots, i-1\}$ are the indices of the wires to the gate that computes the value of wire i.

The depth of the circuit is defined to be the layer associated with the output wire: $d = L_P(s)$. A circuit is *layered* if for every $i \in \{k+1, \ldots, s\}$, it holds that $L_P(j_{i,1}) = L_P(j_{i,2})$. For a layer index $\ell \in [d]$, we define $\mathsf{layer}_\ell(P) = \{i \in [s] : L_P(i) = \ell\}$ to be the set of wire indices in layer ℓ of the circuit.

Remark 4.3 (Layered Monotone Circuit)**.** Every monotone circuit $P\colon \{0,1\}^k \to \{0,1\}$ of size s can be converted into a layered monotone circuit of size $\mathsf{poly}(s)$. Thus, without loss of generality, we exclusively consider layered monotone circuits in the remainder of this work.

4.1 Monotone Policy BARG Construction

We now describe our construction of a monotone policy BARG for NP.

Construction 4.4 (Monotone Policy BARG)**.** Let $\Pi'_{\mathsf{BARG}} = (\mathsf{Gen}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{TrapGen}', \mathsf{Extract}')$ be a somewhere extractable BARG for Boolean circuit satisfiability. Let $\Pi_{\mathsf{H}} = (\mathsf{H.Setup}, \mathsf{H.Hash}, \mathsf{H.ProveOpen}, \mathsf{H.VerOpen}, \mathsf{H.Extract}, \mathsf{H.ValidateDigest})$ be a zero-fixing hash function. We construct a monotone policy BARG $\Pi_{\mathsf{MP\text{-}BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$: On input the security parameter λ, the input length n, the bound on the size of the Boolean circuit s_c, and the bound on the size of the monotone policy s_p, the setup algorithm proceeds as follows:
  - Sample two hash keys $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$ and $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.
  - Let s' be a bound on the size of the circuit that computes the relation $\mathcal{R}[C, k, s_p, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$ from Fig. 1 when instantiated with an arbitrary Boolean circuit C of size at most s_c, an input length $k \le s_p$ and digests $\mathsf{dig}_0, \mathsf{dig}_1$ associated with the hash and verification keys $(\mathsf{hk}_0, \mathsf{vk}_0)$ and $(\mathsf{hk}_1, \mathsf{vk}_1)$. Let $n' = 3 \cdot \lceil \log s_p \rceil + 1$ be the bound on the statement length. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.[7]

  It outputs the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$.

- $\mathsf{Prove}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$: On input $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a monotone layered Boolean policy circuit $P\colon \{0,1\}^k \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$, the prove algorithm does the following:
  - Let s be the size of P. Index the wires of P under a canonical topological ordering (Definition 4.1). For each wire $i \in \{k+1, \ldots, s\}$, let $g_i \in \{\textsc{and}, \textsc{or}\}$ be its type. Let $j_{i,1}, j_{i,2} \in \{1, \ldots, i-1\}$ be the indices of the input wires to the gate i.
  - For each $i \in [s]$, let $\beta_i \in \{0,1\}$ be the value of wire i in the evaluation of P on input $(C(x_1, w_1), \ldots, C(x_k, w_k))$. For $i \in \big\{s+1, \ldots, s_p\big\}$, let $\beta_i = 0$. (This corresponds to "padding" the $s_p - s$ unused slots).
  - Compute the digest $\mathsf{dig}_0 \leftarrow \mathsf{H.Hash}(\mathsf{hk}_0, (\beta_1, \ldots, \beta_{s_p}))$ and $\mathsf{dig}_1 \leftarrow \mathsf{H.Hash}(\mathsf{hk}_1, (\beta_1, \ldots, \beta_{s_p}))$.
  - For each $i \in [s]$ and each $b \in \{0,1\}$, compute the opening $\sigma_i^{(b)} \leftarrow \mathsf{H.ProveOpen}(\mathsf{hk}_b, (\beta_1, \ldots, \beta_{s_p}), i)$.
  - Let C_{aug} be the circuit that computes the relation $\mathcal{R}[C, k, s, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$ shown in Fig. 1.
  - For each $i \in [s_p]$, construct the statement $\hat{x}_i$ and witness $\hat{w}_i$ as follows:
    - If $i \in [k]$, let $\hat{x}_i = (i, x_i)$ and $\hat{w}_i = \big(\beta_i, \sigma_i^{(0)}, \sigma_i^{(1)}, w_i\big)$.
    - If $i \in \{k+1, \ldots, s\}$, let $\hat{x}_i = (i, (g_i, j_{i,1}, j_{i,2}))$ and $\hat{w}_i = \big(\beta_i, \sigma_i^{(0)}, \sigma_i^{(1)}, \big(\beta_{j_{i,1}}, \sigma_{j_{i,1}}^{(0)}, \sigma_{j_{i,1}}^{(1)}, \beta_{j_{i,2}}, \sigma_{j_{i,2}}^{(0)}, \sigma_{j_{i,2}}^{(1)}\big)\big)$.
    - If $i > s$, let $\hat{x}_i = \bot$ and $\hat{w}_i = \bot$.

    Essentially, there is an instance $\hat{x}_i$ associated with each wire i of P.
  - Compute $\pi_{\mathsf{BARG}} \leftarrow \mathsf{Prove}'(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{aug}}, (\hat{x}_1, \ldots, \hat{x}_{s_p}), (\hat{w}_1, \ldots, \hat{w}_{s_p}))$ and output $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$.

Statement: index i and auxiliary statement x
Witness: value b, openings $(\sigma^{(0)}, \sigma^{(1)})$ and auxiliary witness w
Hard-Coded: circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the number of inputs k, the policy size s, the hash keys $\mathsf{hk}_0, \mathsf{hk}_1$, and the digests $\mathsf{dig}_0, \mathsf{dig}_1$

On input a statement (i, x) and a witness $(b, \sigma^{(0)}, \sigma^{(1)}, w)$:

- If $i \le k$, output 1 if all of the following conditions are met, otherwise output 0:
  - **Opening validity:** For all $\alpha \in \{0,1\}$, $\mathsf{H.VerOpen}\left(\mathsf{vk}_\alpha, \mathsf{dig}_\alpha, i, b, \sigma^{(\alpha)}\right) = 1$.
  - **Wire consistency:** $C(x, w) = b$.
- If $i \in \{k+1, \dots, s\}$, parse $x = (g, j_1, j_2)$ where $g \in \{\textsc{and}, \textsc{or}\}$ and $j_1, j_2 \in \{1, \dots, i-1\}$. Parse $w = (b_1, \sigma_1^{(0)}, \sigma_1^{(1)}, b_2, \sigma_2^{(0)}, \sigma_2^{(1)})$. Check each of the following conditions for $\alpha \in \{0,1\}$: Output 1 if all of the following conditions are met, otherwise output 0:
  - **Opening validity:** For all $\alpha \in \{0,1\}$, all of the following holds:
    - $\mathsf{H.VerOpen}\left(\mathsf{vk}_\alpha, \mathsf{dig}_\alpha, j_1, b_1, \sigma_1^{(\alpha)}\right) = 1$;
    - $\mathsf{H.VerOpen}\left(\mathsf{vk}_\alpha, \mathsf{dig}_\alpha, j_2, b_2, \sigma_2^{(\alpha)}\right) = 1$;
    - $\mathsf{H.VerOpen}\left(\mathsf{vk}_\alpha, \mathsf{dig}_\alpha, i, b, \sigma^{(\alpha)}\right) = 1$.
  - **Wire consistency:** $b = g(b_1, b_2)$.
  - **Output gate:** If $i = s$, check that $b = 1$.
- If $i > s$, then output 1.

Figure 1: The relation $\mathcal{R}[C, k, s, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$.

- $\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \dots, x_k), \pi)$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a layered monotone Boolean policy $P\colon \{0,1\}^k \to \{0,1\}$, statements $x_1, \dots, x_k \in \{0,1\}^n$, and a proof $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$, the verification algorithm does the following:
  - If $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = 0$ or $\mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 0$, then output 0.
  - Let s be the size of P. Index the wires of P under a canonical topological ordering (Definition 4.1). For each wire $i \in \{k+1, \dots, s\}$, let $j_{i,1}, j_{i,2} \in \{1, \dots, i-1\}$ be the indices of the input wires of the gate $g_i \in \{\textsc{and}, \textsc{or}\}$ that computes wire i. For each $i \in [s_p]$, construct the statement $\hat{x}_i$ as follows:
    - If $i \in [k]$, let $\hat{x}_i = (i, x_i)$.
    - If $i \in \{k+1, \dots, s\}$, let $\hat{x}_i = (i, (g_i, j_{i,1}, j_{i,2}))$.
    - If $i > s$, let $\hat{x}_i = \perp$.
  - Let C_{aug} be the circuit that computes the relation $\mathcal{R}[C, k, s, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$ from Fig. 1.
  - Output $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, (\hat{x}_1, \dots, \hat{x}_{s_p}), \pi_{\mathsf{BARG}})$.

Theorem 4.5 (Completeness). *If Π'_{BARG} is complete and Π_{H} is correct, then Construction 4.4 is complete.*

Proof. Take any $\lambda, n, s_c, s_p \in \mathbb{N}$, any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, and any monotone Boolean policy $P\colon \{0,1\}^k \to \{0,1\}$ of size $s \le s_p$. Let $x_1, \dots, x_k \in \{0,1\}^n$ and $w_1, \dots, w_k \in \{0,1\}^h$ be a collection of statements and witnesses such that $P(C(x_1, w_1), \dots, C(x_t, w_t)) = 1$. We start by defining the following quantities:

[7] Recall that when the bound on the extraction set parameter ℓ is not given, it defaults to the value 1.

- For each $i \in [s_p]$, let $\beta_i \in \{0,1\}$ be the value of wire i for predicate P on input $(C(x_1, w_t), \ldots, C(x_t, w_t))$.
- Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$. By construction, the hash keys are sampled as $(\mathsf{hk}_b, \mathsf{vk}_b, \mathsf{td}_b) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$ for each $b \in \{0,1\}$.
- Let $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$. Let

  $$\mathsf{dig}_0 = \mathsf{H.Hash}(\mathsf{hk}_0, (\beta_1, \ldots, \beta_{s_p})) \quad \text{and} \quad \mathsf{dig}_1 = \mathsf{H.Hash}(\mathsf{hk}_1, (\beta_1, \ldots, \beta_{s_p}))$$

  be the digests computed by the Prove algorithm. Moreover, by correctness of Π_{H}, $\mathsf{H.ValidateDigest}(\mathsf{vk}_b, \mathsf{dig}_b) = 1$ for $b \in \{0,1\}$.

Consider now the relation $\mathcal{R}[C, k, s, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$ defined in Fig. 1. We show that for all $i \in [s_p]$, the statement $(\hat{x}_i, \hat{w}_i)$ defined in Prove satisfies the relation. First, for all $i \in [s_p]$ and $b \in \{0,1\}$, the Prove algorithm computes $\sigma_i^{(b)} \leftarrow \mathsf{H.ProveOpen}(\mathsf{hk}_b, (\beta_1, \ldots, \beta_{s_p}), i)$. Correspondingly, by correctness of Π_{H}, we conclude that for all $i \in [s_p]$ and $b \in \{0,1\}$,

$$\mathsf{H.VerOpen}\big(\mathsf{vk}_b, \mathsf{dig}_b, i, \beta_i, \sigma_i^{(b)}\big) = 1. \tag{4.1}$$

We now consider each index $i \in [s_p]$:

- If $i \in [k]$, then $\hat{x}_i = (i, x_i)$ and $\hat{w}_i = \big(\beta_i, \sigma_i^{(0)}, \sigma_i^{(1)}, w_i\big)$. In this case, the opening validity passes by Eq. (4.1). Moreover, by definition, we have that $\beta_i = C(x_i, w_i)$. Hence, the relation is satisfied.
- If $i \in \{k+1, \ldots, s\}$, let $\hat{x}_i = (i, (g_i, j_{i,1}, j_{i,2}))$ and $\hat{w}_i = \big(\beta_i, \sigma_i^{(0)}, \sigma_i^{(1)}, \big(\beta_{j_{i,1}}, \sigma_{j_{i,1}}^{(0)}, \sigma_{j_{i,1}}^{(1)}, \beta_{j_{i,2}}, \sigma_{j_{i,2}}^{(0)}, \sigma_{j_{i,2}}^{(1)}\big)\big)$. Again, the opening validity check passes by Eq. (4.1). Moreover, by definition, $\beta_i = g_i\big(\beta_{j_{i,1}}, \beta_{j_{i,2}}\big)$ so the wire consistency-check passes. Finally, if $i = s$, then $\beta_s = P(C(x_1, w_1), \ldots, (x_t, w_t)) = 1$ by construction.
- Finally, if $i > s$, the relation is always satisfied.

Thus, we conclude that for all $i \in [s_p]$, the relation $\mathcal{R}$ is always satisfied. By completeness of Π'_{BARG}, this means $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, (\hat{x}_1, \ldots, \hat{x}_{s_p}), \pi_{\mathsf{BARG}}) = 1$, where $\pi_{\mathsf{BARG}} \leftarrow \mathsf{Prove}'(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{aug}}, (\hat{x}_1, \ldots, \hat{x}_{s_p}), (\hat{w}_1, \ldots, \hat{w}_{s_p}))$. Letting $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$, we conclude that $\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), \pi) = 1$, and completeness follows. □

Theorem 4.6 (Succinctness). *If Π'_{BARG} and Π_{H} satisfy succinctness, then Construction 4.4 has slightly succinct proofs.*

Proof. Fix $\lambda, n, s_c, s_p \in \mathbb{N}$, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, and a monotone Boolean policy $P\colon \{0,1\}^k \to \{0,1\}$ of size $s \le s_p$. Take any $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ in the support of $\mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$. Consider any collection of statements $x_1, \ldots, x_k \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$ where $P(C(x_1, w_1), \ldots, C(x_t, w_t)) = 1$. We bound the size of $\pi_{\mathsf{BARG}} \leftarrow \mathsf{Prove}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$:

- First, the Prove algorithm computes dig_0 and dig_1 using the hash keys hk_0 and hk_1, respectively. Since $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$, we appeal to succinctness of Π_{H} to conclude that $|\mathsf{dig}_0|, |\mathsf{dig}_1| \le \mathrm{poly}(\lambda + \log s_p)$.
- Next, consider the size of the BARG proof π_{BARG}. We first bound the size of the circuit C_{aug} for computing the relation $\mathcal{R}[C, k, s, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$ from Fig. 1. By construction, C_{aug} performs a constant number of calls to H.VerOpen and also needs to evaluate the underlying circuit C (which has size at most s_c). By succinctness of Π_{H}, each invocation of H.VerOpen can be computed by a circuit of size $\mathrm{poly}(\lambda + \log s_p)$. Hence, the size of the circuit C_{aug} can be bounded by $\mathrm{poly}(\lambda + \log s_p + s_c)$. By succinctness of Π_{BARG}, we conclude that the size of the proof π_{BARG} output by Prove′ is bounded by $\mathrm{poly}(\lambda + |C_{\mathsf{aug}}| + \log s_p) \le \mathrm{poly}(\lambda + s_c + \log s_p)$.

Putting the pieces together, the proof $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ output by Prove is bounded by $\mathrm{poly}(\lambda + s_c + \log s_p)$, and (slight) succinctness follows. □

Soundness. We now state the soundness theorem, but give the proof in the subsequent section (Section 4.2).

Theorem 4.7 (Non-Adaptive Soundness)**.** *If Π_{H} satisfies set hiding, index hiding with extracted guess, zero fixing and extractor validity against non-uniform adversaries, and Π_{BARG} is somewhere extractable and satisfies set hiding against non-uniform adversaries, then Construction 4.4 satisfies non-adaptive soundness against non-uniform adversaries.*

4.2 Proof of Theorem 4.7 (Non-Adaptive Soundness)

In this section, we prove non-adaptive soundness of Construction 4.4. Take any efficient non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ for the non-adaptive soundness game. Then, for any $\lambda \in \mathbb{N}$, let $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$, where

- $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is a Boolean circuit of size at most s_c;
- $P \colon \{0,1\}^k \to \{0,1\}$ is a layered monotone Boolean circuit of size $s \le s_p$; and
- $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$ where $x_i \in \{0,1\}^n$ for all $i \in [k]$.

Let d be the depth of P. For each $i \in [k]$, let $\beta_i \in \{0,1\}$ be equal to 1 if $(C, x_i) \in \mathcal{L}_{\mathsf{CSAT}}$ and 0 otherwise. Extending the definition to all $i \in [s]$, let $\beta_i \in \{0,1\}$ be the value of wire i in the evaluation of P on input $(\beta_1, \ldots, \beta_k)$. For each layer $i \in [d]$, define

$$J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\} \tag{4.2}$$

to be the indices of the zero wires in layer i of $P(\beta_1, \ldots, \beta_k)$. We model each J_i as an *ordered* set (ordered in ascending order). We write $J_i[t]$ to denote the t^{th} element in J_i and $J_i[1, \ldots, t] = \bigcup_{j \in [t]} J_i[j]$ to denote the first t elements of J_i.

4.2.1 Hybrid Experiment Specification

To prove Theorem 4.7, we start by defining a sequence of hybrid experiments. Each of these hybrids is indexed implicitly by the security parameter λ, but we omit this for ease of exposition.

Outer games. We start by defining a sequence of "outer hybrids." Here, we provide a general overview of our methodology. The initial hybrid Hyb_0 corresponds to the real non-adaptive soundness game, while Hyb_i corresponds to the hybrid where one of the zero-fixing hash keys is binding on the set J_i (as defined by Eq. (4.2)). We show that the outputs of each adjacent pair of hybrid distributions can only change by a negligible amount, and moreover, that the zero-fixing hash function binding on J_i in Hyb_i outputs $\mathsf{Matching}$. Finally, in hybrid Hyb_d, the following two conditions hold:

- The hash key is zero-fixing on the single output wire (since we know that $P(\beta_1, \ldots, \beta_k) = 0$, where β_i is the indicator bit for whether $(C, x_i) \in \mathcal{L}_{\mathsf{CSAT}}$).
- The hash function declares the output bit to be $\mathsf{Matching}$.

Consider the probability that the proof verifies in Hyb_d:

- Suppose the BARG is extractable on the instance associated with the output wire of P. In this case, if the proof verifies in Hyb_d, then somewhere extractability of the BARG allows us to extract an opening to 1 with respect to both zero-fixing hash functions. This follows by definition of the instance $\hat{x}_s$ in Prove and Verify (where s is the size of P).
- Since $P(\beta_1, \ldots, \beta_k) = 0$, one of the zero-fixing hash functions will be zero-fixing on the output wire in hybrid Hyb_d. Moreover, this hash function outputs $\mathsf{Matching}$. If we can extract an opening to 1 for the output wire, this breaks zero-fixing security of the hash function.

Thus, when the BARG is extractable on the instance associated with the output wire, the probability that the proof verifies in Hyb_d is negligible. Finally, if the outputs of each adjacent pair of hybrids cannot differ by a non-negligible amount, we conclude the probability that the proof verifies in Hyb_0 is also negligible. This demonstrates non-adaptive soundness. We now define the sequence of games:

- Hyb_0: This is the non-adaptive soundness game. For ease of exposition, we partition the game into two phases:
  - **Phase 1:** On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs $1^n, 1^{s_c}, 1^{s_p}$, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, a monotone Boolean circuit $P\colon \{0,1\}^k \to \{0,1\}$ of size $s \le s_p$, an instance $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$, and the state $\mathsf{st}_{\mathcal{A}}$. If $(C, P, \mathbf{x}) \in \mathcal{L}_{\mathsf{MP\text{-}CSAT}}$, then the experiment outputs 0.
  - **Phase 2:** The challenger computes $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$. Specifically, the challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
    - $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.
    - $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.

    The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
- Hyb_i for $i \in [d]$: Same as Hyb_0, but $\mathsf{hk}_{\mathsf{low}}$ binds on J_i, where $\mathsf{low} = i \bmod 2$ and $\mathsf{high} = 1 - \mathsf{low}$. Specifically, the game proceeds as follows:
  - **Phase 1:** On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs $1^n, 1^{s_c}, 1^{s_p}$, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, a monotone Boolean circuit $P\colon \{0,1\}^k \to \{0,1\}$ of size $s \le s_p$, an instance $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$, and the state $\mathsf{st}_{\mathcal{A}}$. If $(C, P, \mathbf{x}) \in \mathcal{L}_{\mathsf{MP\text{-}CSAT}}$, then the experiment outputs 0. In addition, the challenger computes the following quantities:
    - For $j \in [k]$, let $\beta_j = 1$ if $(C, x_j) \in \mathcal{L}_{\mathsf{CSAT}}$ (Definition 2.5) and $\beta_j = 0$ otherwise.
    - For $j \in [k+1, s]$, let β_i to be the value of the wire j in the evaluation of P on $(\beta_1, \ldots, \beta_k)$.
    - For each layer $\ell \in [d]$, let $J_\ell = \{j \in \mathsf{layer}_\ell(P) : \beta_j = 0\}$.
  - **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
    - $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.

    The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.

Inner games. To argue that each pair of outer hybrids is computationally indistinguishable, we now define a sequence of "inner hybrids." Whereas the outer hybrids advance layer by layer, the inner hybrids advance *across* a layer. In more detail, recall that the difference between Hyb_i and Hyb_{i+1} is that *one* of the two hash keys (i.e., $\mathsf{hk}_{\mathsf{low}}$) goes from binding on J_i to binding on J_{i+1}. The idea in the inner hybrids is to program the *other* hash key (i.e., $\mathsf{hk}_{\mathsf{high}}$) to be binding on J_{i+1}. Initially, $\mathsf{hk}_{\mathsf{high}}$ is binding on the empty set. We then step through $|J_{i+1}|$ intermediate hybrids, where on the t^{th} step, the hash key $\mathsf{hk}_{\mathsf{high}}$ goes from being binding on $J_{i+1}[1, \ldots, t-1]$ to being binding on $J_{i+1}[1, \ldots, t]$. Each transition relies on the security of the BARG and the zero-fixing hash function. We now define the full sequence of hybrids; each one is indexed by $i \in \{0, \ldots, d\}$.

- $\mathsf{Hyb}_{i,t,1}$ for $t \in [|J_{i+1}|]$: Same as Hyb_i, but $\mathsf{hk}_{\mathsf{high}}$ binds on the first $t-1$ wires in J_{i+1}.
  - **Phase 1:** Same as Hyb_i.

- **Phase 2:** The challenger samples the following components:
  - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
  - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
  - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
  - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
  - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
  - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.

- $\mathsf{Hyb}_{i,t,2}$ for $t \in [|J_{i+1}|]$: Same as $\mathsf{Hyb}_{i,t,1}$, but $\mathsf{crs}_{\mathsf{BARG}}$ is set to be extractable on index $J_{i+1}[t]$.
  - **Phase 1:** Same as Hyb_i.
  - **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$.
    - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$.

    The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.

- $\mathsf{Hyb}_{i,t,3}$ for $t \in [|J_{i+1}|]$: Same as $\mathsf{Hyb}_{i,t,2}$, but the challenger additionally checks that it extracts a valid witness for $\hat{x}_{J_{i+1}[t]}$.
  - **Phase 1:** Same as Hyb_i.
  - **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$.
    - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$.

    The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.
    - $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ where $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$.

- $\mathsf{Hyb}_{i,t,4}$ for $t \in [|J_{i+1}|]$: Same as $\mathsf{Hyb}_{i,t,3}$, but the challenger additionally checks that the extracted value for wire $J_{i+1}[t]$ is a 0.
  - **Phase 1:** Same as Hyb_i.

- **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$.
    - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.
    - $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ and $b = 0$ where $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$.

- $\mathsf{Hyb}_{i,t,5}$ for $t \in [|J_{i+1}|]$: Same as $\mathsf{Hyb}_{i,t,4}$ but $\mathsf{hk}_{\mathsf{high}}$ now binds on $J_{i+1}[1, \ldots, t]$.
  - **Phase 1:** Same as Hyb_i.
  - **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$.
    - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t])$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.
    - $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ and $b = 0$ where $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$.

- $\mathsf{Hyb}_{i,t,6}$ for $t \in [|J_{i+1}|]$: Same as $\mathsf{Hyb}_{i,t,5}$ but the challenger does not check the extracted witnesses.
  - **Phase 1:** Same as Hyb_i.
  - **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$.
    - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t])$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.

  In particular, the challenger does *not* check any conditions on $\hat{w} \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$.

- $\mathsf{Hyb}_{i,t,7}$ for $t \in [|J_{i+1}|]$: Same as $\mathsf{Hyb}_{i,t,6}$, except the BARG is restored to normal mode.

- **Phase 1:** Same as Hyb_i.
- **Phase 2:** The challenger samples the following components:
  - $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
  - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
  - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t])$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):

  - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
  - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
  - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.

- $\mathsf{Hyb}_{i,\mathsf{final}}$: Same as $\mathsf{Hyb}_{i,|J_{i+1}|,7}$, but we no longer checks that $\mathsf{dig}_{\mathsf{low}}$ matches the binding set J_i.

  - **Phase 1:** Same as Hyb_i.
  - **Phase 2:** The challenger samples the following components:
    - $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
    - If $i = 0$, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1})$.

    The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):

    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.

    In particular, the challenger no longer checks if $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$ when $i > 1$.

4.2.2 Analysis of Hybrid Experiments

We now show that the probability of a hybrid experiment outputting 1 cannot decrease by a non-negligible amount when transitioning from one hybrid to the next. The goal is to eventually show that $\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1]$ is negligibly close to $\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]$ for all $i \in [d]$. We argue this via a sequence of *non-uniform* reductions to the security properties of the underlying zero-fixing hash function and BARG. Specifically, our reduction algorithms construct a non-uniform adversary where there is an initial (inefficient) preprocessing phase that outputs an advice string of polynomial size, and a polynomial-time online algorithm that takes the advice as input and interacts with the challenger according to the specifications of the target security game. Our reductions share a common preprocessing phase, which we abstract out as a standalone Preprocess algorithm defined as follows:

- $\mathsf{Preprocess}(C, P, \mathbf{x})$: On input a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a monotone Boolean policy $P \colon \{0,1\}^k \to \{0,1\}$ of size s, and an instance $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$, the preprocessing algorithm first checks if $(C, P, \mathbf{x}) \in \mathcal{L}_{\mathsf{MP\text{-}CSAT}}$. If so, it outputs $\bot$. Otherwise, it computes $\beta_i \in \{0,1\}$ for $i \in [s]$ as follows:

  - For $i \in [k]$, set $\beta_i = 1$ if and only if $(C, x_i) \in \mathcal{L}_{\mathsf{CSAT}}$.
  - For $i \in [k+1, s]$, set β_i to be the value of the wire i in the evaluation of P on $(\beta_1, \ldots, \beta_k)$.

  Output $\tau = \big(C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s)\big)$.

We now analyze each pair of adjacent hybrid experiments.

Claim 4.8. *If* Π_{H} *satisfies extractor validity against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$*, it holds that*

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

Proof. Take any $i \in \{0, \ldots, d-1\}$ and suppose $\left|\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i,1,1}(\mathcal{A}) = 1]\right| = \varepsilon(\lambda)$ for some non-negligible ε. By construction, the only difference between Hyb_i and $\mathsf{Hyb}_{i,1,1}$ is the *additional* check in $\mathsf{Hyb}_{i,1,1}$:

$$\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}.$$

Thus, with probability at least ε, the adversary $\mathcal{A}$ in an execution of $\mathsf{Hyb}_{i,1,1}$ and Hyb_i outputs a proof $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ where $\mathsf{H.ValidateDigest}(\mathsf{vk}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = 1$ and $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{NotMatching}$. In all other cases, the output of the two experiments are identical. We use $\mathcal{A}$ to construct a non-uniform adversary $\mathcal{B}$ that breaks extract validity of Π_{H} as follows:

- **Preprocessing phase:** On input the security parameter 1^λ, run $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Compute $\tau \leftarrow \mathsf{Preprocess}(C, P, \mathbf{x})$ and output $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$.
- **Online phase:** On input the state $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$ where $\tau = \big(C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s)\big)$, proceed as follows:
  1. Send 1^{s_p} and the set $\varnothing$ to the challenger. The challenger replies with a hash key hk and a verification key vk.
  2. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'\big(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}\big)$, where n', s' are defined as in Construction 4.4. If $i = 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, if $i \neq 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$, where $J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\}$. Let $\mathsf{hk}_{\mathsf{high}} \leftarrow \mathsf{hk}$ and $\mathsf{vk}_{\mathsf{high}} \leftarrow \mathsf{vk}$.
  3. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and run $\pi \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Parse $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ and output $\mathsf{dig}_{\mathsf{high}}$.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_i and $\mathsf{Hyb}_{i,1,1}$ for $\mathcal{A}$, so with probability at least ε, the digest $\mathsf{dig}_{\mathsf{high}}$ satisfies $\mathsf{H.ValidateDigest}(\mathsf{vk}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = 1$ and $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{NotMatching}$. Correspondingly, algorithm $\mathcal{B}$ breaks extractor validity with advantage ε. □

Claim 4.9. *If* Π_{BARG} *satisfies set hiding against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$ *and* $t \in [|J_{i+1}|]$*, it holds that*

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

Proof. Take any $i \in \{0, \ldots, d-1\}$ and $t \in [|J_{i+1}|]$, and suppose $\left|\Pr[\mathsf{Hyb}_{i,t,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i,t,2}(\mathcal{A}) = 1]\right| = \varepsilon(\lambda)$ for some non-negligible ε. We construct a non-uniform adversary $\mathcal{B}$ that breaks set hiding of Π_{BARG}:

- **Preprocessing phase:** On input the security parameter 1^λ, run $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Compute $\tau \leftarrow \mathsf{Preprocess}(C, P, \mathbf{x})$ and output $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$.
- **Online phase:** On input the state $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$ where $\tau = \big(C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s)\big)$, proceed as follows:
  1. Compute $J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\}$ and $J_{i+1} = \{j \in \mathsf{layer}_{i+1}(P) : \beta_j = 0\}$.
  2. Send $1^{s_p}, 1^{n'}, 1^{s'}$, and the index $J_{i+1}[t]$ to the challenger, where n', s' are computed as in Construction 4.4. The challenger replies with $\mathsf{crs}_{\mathsf{BARG}}$.
  3. If $i = 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, if $i > 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$. Sample $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$.
  4. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and run $\pi \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$.
  5. Let $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ and let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. Output 1 if all of the following conditions hold (and abort with $\perp$ otherwise):

- $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
- $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
- If $i \geq 1$ then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
- $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$

We consider two possibilities:

- In $\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 0)$, the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$. In this case, by the construction of $\mathcal{B}$, it holds that crs is sampled exactly as in $\mathsf{Hyb}_{i,t,1}$. Moreover, $\mathcal{B}$ computes its output exactly as specified by $\mathsf{Hyb}_{i,t,1}$. This means that $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{i,t,1}(\mathcal{A})$ and thus
$$\Pr[\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 0) = 1] = \Pr[\mathsf{Hyb}_{i,t,1}(\mathcal{A}) = 1].$$

- In $\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 1)$, the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$. In this case, by the construction of $\mathcal{B}$, it holds that crs is sampled exactly as in $\mathsf{Hyb}_{i,t,2}$. Moreover, $\mathcal{B}$ computes its output exactly as specified by $\mathsf{Hyb}_{i,t,2}$. This means that $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{i,t,2}(\mathcal{A})$ and thus
$$\Pr[\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 1) = 1] = \Pr[\mathsf{Hyb}_{i,t,2}(\mathcal{A}) = 1].$$

We conclude that algorithm $\mathcal{B}$ breaks the index hiding property of Π_{BARG} with the same advantage ε. □

Claim 4.10. *If* Π_{BARG} *satisfies somewhere extractability in trapdoor mode against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$, $t \in [|J_{i+1}|]$, *it holds that*
$$\left|\Pr[\mathsf{Hyb}_{i,t,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i,t,3}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Take any $i \in \{0, \ldots, d-1\}$ and $t \in [|J_{i+1}|]$. Suppose $\left|\Pr[\mathsf{Hyb}_{i,t,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i,t,3}(\mathcal{A}) = 1]\right| = \varepsilon(\lambda)$ for some non-negligible ε. By construction, the only difference between $\mathsf{Hyb}_{i,t,2}$ and $\mathsf{Hyb}_{i,t,3}$ is the *additional* check in $\mathsf{Hyb}_{i,t,3}$:
$$C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1 \text{ where } \hat{w} \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t]). \tag{4.3}$$

Thus, with probability at least ε, the adversary $\mathcal{A}$ in an execution of $\mathsf{Hyb}_{i,t,2}$ and $\mathsf{Hyb}_{i,t,3}$ outputs a proof $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ where $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$ and Eq. (4.3) does *not* hold (i.e., $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 0$). In all other cases, the outputs of $\mathsf{Hyb}_{i,t,2}$ and $\mathsf{Hyb}_{i,t,3}$ are identical. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the somewhere extractability game of Π_{BARG}:

- **Preprocessing phase:** On input the security parameter 1^λ, run $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Compute $\tau \leftarrow \mathsf{Preprocess}(C, P, \mathbf{x})$ and output $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$.

- **Online phase:** On input the state $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$ where $\tau = \big(C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s)\big)$, proceed as follows:

  1. Compute $J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\}$ and $J_{i+1} = \{j \in \mathsf{layer}_{i+1}(P) : \beta_j = 0\}$.
  2. Send $1^{s_p}, 1^{n'}, 1^{s'}$ and the index $J_{i+1}[t]$ to the challenger, where n', s' are computed as in Construction 4.4. The challenger replies with $\mathsf{crs}_{\mathsf{BARG}}$.
  3. If $i = 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, if $i > 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$. Sample $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$.
  4. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and run $\pi \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. Parse $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ and output the circuit C_{aug}, the statements $\hat{\mathbf{x}}$, and the proof π_{BARG}.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{i,t,2}$ and $\mathsf{Hyb}_{i,t,3}$ for $\mathcal{A}$, so with probability ε, it outputs π_{BARG} such that $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$ and Eq. (4.3) does not hold. In particular, this means the proof π_{BARG} verifies with respect to $\mathsf{crs}_{\mathsf{BARG}}$ and yet $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 0$ where $\hat{w} \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$, and $\mathsf{td}_{\mathsf{BARG}}$ is the trapdoor associated with $\mathsf{crs}_{\mathsf{BARG}}$ that the challenger sampled. This means $\mathcal{B}$ wins the somewhere extractability game with the same advantage ε. □

Claim 4.11. *If Π_{H} satisfies zero-fixing against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}, t \in [|J_{i+1}|]$, *it holds that*

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

Proof. By construction, the only difference between $\mathsf{Hyb}_{i,t,3}$ and $\mathsf{Hyb}_{i,t,4}$ is the *additional* check in $\mathsf{Hyb}_{i,t,4}$ that the extracted bit b satisfies $b = 0$. We consider two cases in our analysis:

- Suppose $i = 0$. In this case, $J_1[t]$ refers to an *input* wire in P, which means $J_1[t] \leq k$. Suppose $\mathsf{Hyb}_{i,t,3}(\mathcal{A})$ outputs 1. This means that $C_{\mathsf{aug}}(\hat{x}_{J_1[k]}, \hat{w}) = 1$ where

$$\hat{w} = \left(b, \sigma^{(0)}, \sigma^{(1)}, w\right) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_1[t]).$$

  Since $J_1[k] \leq k$ and by construction of $\hat{\mathbf{x}}$, we have that $\hat{x}_{J_1[k]} = \left(J_1[k], x_{J_1[k]}\right)$. By definition of C_{aug} (see Fig. 1), we have that $C_{\mathsf{aug}}(\hat{x}_{J_1[k]}, \hat{w}) = 1$ only if $C(x_{J_1[k]}, w) = b$. However, by definition of J_1, it must be the case that $\left(C, x_{J_1[k]}\right) \notin \mathcal{L}_{\mathsf{CSAT}}$. This means $C(x_{J_1[k]}, w) = 0 = b$. In this case, $\mathsf{Hyb}_{i,t,4}(\mathcal{A})$ also outputs 1. Conversely, since the verification conditions in $\mathsf{Hyb}_{i,t,4}$ are a *superset* of the conditions in $\mathsf{Hyb}_{i,t,3}$, if $\mathsf{Hyb}_{i,t,4}(\mathcal{A}) = 1$, then $\mathsf{Hyb}_{i,t,3}(\mathcal{A}) = 1$. We conclude that in this case

$$\Pr[\mathsf{Hyb}_{i,t,3}(\mathcal{A}) = 1]] = \Pr[\mathsf{Hyb}_{i,t,4} = 1].$$

- Suppose $i > 0$. In this case, security reduces to the zero-fixing security of Π_{H}. We give this proof below.

To argue the second case, take any $i \in \{1, \ldots, d-1\}$ and $t \in [|J_{i+1}|]$, and suppose that

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

for some non-negligible ε. By construction, the only difference between $\mathsf{Hyb}_{i,t,3}$ and $\mathsf{Hyb}_{i,t,4}$ is the *additional* check in $\mathsf{Hyb}_{i,t,4}$ that $b = 0$ where $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$. Thus, with probability at least ε, the adversary $\mathcal{A}$ in an execution of $\mathsf{Hyb}_{i,t,3}$ and $\mathsf{Hyb}_{i,t,4}$ will output a proof $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ where

$$\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching} \quad \text{and} \quad C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1 \quad \text{and} \quad b \neq 0. \tag{4.4}$$

In all other cases, the outputs of $\mathsf{Hyb}_{i,t,3}$ and $\mathsf{Hyb}_{i,t,4}$ are identical. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the zero-fixing game for Π_{H}:

- **Preprocessing phase:** On input the security parameter 1^λ, run $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Compute $\tau \leftarrow \mathsf{Preprocess}(C, P, \mathbf{x})$ and output $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$.
- **Online phase:** On input the state $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$ where $\tau = \left(C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s)\right)$, proceed as follows:
  1. Compute $J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\}$ and $J_{i+1} = \{j \in \mathsf{layer}_{i+1}(P) : \beta_j = 0\}$.
  2. Send 1^{s_p} and the set J_i to the challenger. The challenger replies with a hash key hk and a verification key vk.
  3. Set $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$, $\mathsf{hk}_{\mathsf{low}} \leftarrow \mathsf{hk}$, and $\mathsf{vk}_{\mathsf{low}} \leftarrow \mathsf{vk}$. Finally, sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'\left(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t]\right)$, where s' is defined as in Construction 4.4.
  4. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and run $\pi \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$.
  5. Let $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ and suppose $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. Compute

$$\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t]),$$

     and parse the extracted witness w as $w = \left(b_1, \sigma_1^{(0)}, \sigma_1^{(1)}, b_2, \sigma_2^{(0)}, \sigma_2^{(1)}\right)$.

6. Parse $\hat{x}_{J_{i+1}[t]} = (g, j_1, j_2)$. If there exists $\alpha \in \{1,2\}$ such that $j_\alpha \in J_i$ and $b_\alpha = 1$, output the digest $\mathsf{dig}_{\mathsf{low}}$, the index j_α, and the opening $\sigma_\alpha^{(\mathsf{low})}$.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{i,t,3}$ and $\mathsf{Hyb}_{i,t,4}$ for $\mathcal{A}$. Thus, with probability at least ε, algorithm $\mathcal{A}$ will output a proof $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ where Eq. (4.4) holds. Since $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$, the following properties hold:

- $b = g(b_1, b_2)$, where $g = g_{J_{i+1}[t]} \in \{\textsc{and}, \textsc{or}\}$ is the gate in the circuit P that computes wire $J_{i+1}[t]$.
- $\mathsf{H.VerOpen}(\mathsf{vk}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}, j_1, b_1, \sigma_1^{(\mathsf{low})}) = 1$ and $\mathsf{H.VerOpen}(\mathsf{vk}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}, j_2, b_2, \sigma_2^{(\mathsf{low})}) = 1$.

By definition, j_1, j_2 are the indices of the input wires to the gate whose output wire is $J_{i+1}[t]$. We consider two possibilities:

- Suppose $b_1 \le \beta_{j_1}$ and $b_2 \le \beta_{j_2}$. By definition (see the details of the Preprocess algorithm), $\beta_{J_{i+1}[t]} = g(\beta_{j_1}, \beta_{j_2})$. Since $J_{i+1}[t] \in J_{i+1}$, this means $\beta_{J_{i+1}[t]} = 0$. Since g is a monotone gate and $b_1 \le \beta_{j_1}$ and $b_2 \le \beta_{j_2}$, we have that $b = g(b_1, b_2) \le g(\beta_{j_1}, \beta_{j_2}) = 0$. Since $b \in \{0,1\}$, this means that $b = 0$. However, if Eq. (4.4) holds, then $b \ne 0$, so this case does not happen.
- Suppose there exist $\alpha \in \{1,2\}$ such that $b_\alpha > \beta_{j_\alpha}$. This means that $\beta_{j_\alpha} = 0$ and $b_\alpha = 1$. Since P is a layered monotone circuit, this means $j_1, j_2 \in \mathsf{layer}_i(P)$. Since $\beta_{j_\alpha} = 0$, this means that $j_\alpha \in J_i$. In conjunction with Eq. (4.4), this means

  $$\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching} \quad \text{and} \quad \mathsf{H.VerOpen}(\mathsf{vk}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}, j_\alpha, 1, \sigma_\alpha^{(\mathsf{low})}) = 1 \quad \text{and} \quad j_\alpha \in J_i,$$

  where $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$ is the hash function parameters sampled by the zero-fixing challenger. In this case, algorithm $\mathcal{B}$ wins the zero-fixing game.

We conclude that if the proof π output by $\mathcal{A}$ satisfies Eq. (4.4) with probability ε, then algorithm $\mathcal{B}$ wins the zero-fixing game with advantage at least ε. □

Claim 4.12. *If Π_{H} satisfies one-sided index hiding with extracted guess security against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $i \in \{0, \ldots, d-1\}$, $t \in [|J_{i+1}|]$, it holds that*

$$\Pr[\mathsf{Hyb}_{i,t,5}(\mathcal{A}) = 1] \ge \Pr[\mathsf{Hyb}_{i,t,4}(\mathcal{A}) = 1] - \mathsf{negl}(\lambda).$$

Proof. Take any $i \in \{0, \ldots, d-1\}$ and $t \in [|J_{i+1}|]$. Suppose $\Pr[\mathsf{Hyb}_{i,t,5}(\mathcal{A}) = 1] \le \Pr[\mathsf{Hyb}_{i,t,4}(\mathcal{A})] - \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct a non-uniform adversary $\mathcal{B}$ for the index hiding with extracted guess game of Π_{H}:

- **Preprocessing phase:** On input the security parameter 1^λ, run $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Compute $\tau \leftarrow \mathsf{Preprocess}(C, P, \mathbf{x})$ and output $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$.
- **Online phase:** On input the state $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$ where $\tau = (C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s))$, proceed as follows:
  1. Compute $J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\}$ and $J_{i+1} = \{j \in \mathsf{layer}_{i+1}(P) : \beta_j = 0\}$.
  2. Send 1^{s_p}, the set $J_{i+1}[1, \ldots, t]$, and the index $J_{i+1}[t]$ to the challenger. The challenger replies with hk and vk.
  3. If $i = 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. Otherwise, if $i > 0$, sample $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$. Let $\mathsf{hk}_{\mathsf{high}} \leftarrow \mathsf{hk}$ and $\mathsf{vk}_{\mathsf{high}} \leftarrow \mathsf{vk}$.
  4. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_{i+1}[t])$, where s' is defined as in Construction 4.4.
  5. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and run $\pi \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. Parse $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$ and compute $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_{i+1}[t])$.

6. Check each of the following conditions:
   - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
   - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
   - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
   - $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ and $b = 0$.

   If any condition fails to verify, then output $\perp$. Otherwise, output the digest $\mathsf{dig}_{\mathsf{high}}$ and the opening $\sigma^{(\mathsf{high})}$.

We consider the two possibilities:

- Suppose the challenger responds according to the specification of $\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 0)$. In this case, the challenger samples $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t-1])$. By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{i,t,4}$ for $\mathcal{A}$. The output of $\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 0)$ is 1 if and only if *all* of the following events occur:

  - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
  - If $i \geq 1$, then $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.
  - $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ and $b = 0$.
  - $\mathsf{H.VerOpen}\big(\mathsf{vk}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}, J_{i+1}[t], 0, \sigma^{(\mathsf{high})}\big) = 1$ and $\mathsf{H.Extract}\big(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}\big) = \mathsf{Matching}$.

  We now argue that
  $$\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 0) = 1 \iff \mathsf{Hyb}_{i,t,4}(\mathcal{A}) = 1.$$
  The forward direction is immediate since the set of conditions under which $\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 0)$ outputs 1 is a strict superset of the conditions under which $\mathsf{Hyb}_{i,t,4}(\mathcal{A})$ outputs 1. For the backward direction, it suffices to show that if $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ and $b = 0$, then $\mathsf{H.VerOpen}\big(\mathsf{vk}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}, J_{i+1}[t], 0, \sigma^{(\mathsf{high})}\big) = 1$. By construction of C_{aug}, we have that $C_{\mathsf{aug}}(\hat{x}_{J_{i+1}[t]}, \hat{w}) = 1$ implies that for all $\alpha \in \{1, 2\}$, it holds that
  $$\mathsf{H.VerOpen}\big(\mathsf{vk}_\alpha, \mathsf{dig}_\alpha, J_{i+1}[t], 0, \sigma^{(\alpha)}\big) = 1.$$
  In particular, this holds for $\alpha = \mathsf{high}$, so the claim holds. We conclude then that
  $$\Pr[\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 0) = 1] = \Pr[\mathsf{Hyb}_{i,t,4}(\mathcal{A}) = 1].$$

- Suppose the challenger responds according to the specification of $\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 1)$. In this case, the challenger samples $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}[1, \ldots, t])$. Thus, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{i,t,5}$ for $\mathcal{A}$. Since $\mathsf{Hyb}_{i,t,4}$ and $\mathsf{Hyb}_{i,t,5}$ share *identical* verification conditions, we can appeal to the same argument as before to argue that
  $$\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 1) = 1 \iff \mathsf{Hyb}_{i,t,5}(\mathcal{A}) = 1.$$
  Correspondingly,
  $$\Pr[\mathsf{ExptIHE}_{\mathcal{B}}(\lambda, 1) = 1] = \Pr[\mathsf{Hyb}_{i,t,5}(\mathcal{A}) = 1].$$

We conclude that $\mathcal{B}$ breaks the one-sided index hiding with extracted guess with the same advantage ε. □

Claim 4.13. *For every $i \in \{0, \ldots, d-1\}$, $t \in [|J_{i+1}|]$, it holds that*
$$\Pr[\mathsf{Hyb}_{i,t,6}(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_{i,t,5}(\mathcal{A}) = 1].$$

Proof. The only difference between $\mathsf{Hyb}_{i,t,5}$ and $\mathsf{Hyb}_{i,t,6}$ is that $\mathsf{Hyb}_{i,t,5}$ performs an additional check that the extracted witness $\hat{w}$ satisfies certain properties. Thus, whenever $\mathsf{Hyb}_{i,t,6}$ outputs 1, hybrid $\mathsf{Hyb}_{i,t,5}$ also outputs 1 and the claim follows. □

Claim 4.14. *If Π_{BARG} satisfies set hiding against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $i \in \{0, \ldots, d-1\}$ and $t \in [|J_{i+1}|]$, it holds that*

$$\left|\Pr[\mathsf{Hyb}_{i,t,6}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i,t,7}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. This follows by a similar argument as the proof of Claim 4.9. □

Claim 4.15. *For every $i \in \{0, \ldots, d-1\}$, it holds that*

$$\Pr[\mathsf{Hyb}_{i,\mathsf{final}}(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_{i,|J_{i+1}|,7}(\mathcal{A}) = 1].$$

Proof. The only difference between $\mathsf{Hyb}_{i,|J_{i+1}|,7}$ and $\mathsf{Hyb}_{i,\mathsf{final}}$ is that $\mathsf{Hyb}_{i,|J_{i+1}|,7}$ performs an additional check that $\mathsf{dig}_{\mathsf{low}}$ is $\mathsf{Matching}$. Thus, whenever $\mathsf{Hyb}_{i,\mathsf{final}}$ outputs 1, hybrid $\mathsf{Hyb}_{i,|J_{i+1}|,7}$ also outputs 1 and the claim follows. □

Claim 4.16. *If Π_{H} satisfies set hiding property against efficient non-uniform adversaries then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $i \in \{0, \ldots, d-1\}$, it holds that*

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

Proof. We consider two cases in our analysis:

- Suppose $i = 0$. Then, in hybrid $\mathsf{Hyb}_{0,\mathsf{final}}$, the challenger samples $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$ and $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_1)$. This is identical to how the challenger samples hk_0 and hk_1 in Hyb_1. Similarly, the verification conditions in the two experiments are identical, so we conclude that

$$\Pr[\mathsf{Hyb}_{0,\mathsf{final}}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{i+1}(\mathcal{A}) = 1].$$

- Suppose $i > 0$. Let $\alpha = i \bmod 2$. By construction, in hybrid $\mathsf{Hyb}_{i,\mathsf{final}}$, the challenger samples

$$(\mathsf{hk}_\alpha, \mathsf{vk}_\alpha, \mathsf{td}_\alpha) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i) \quad \text{and} \quad (\mathsf{hk}_{1-\alpha}, \mathsf{vk}_{1-\alpha}, \mathsf{td}_{1-\alpha}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}).$$

  In hybrid Hyb_{i+1}, the challenger samples

$$(\mathsf{hk}_\alpha, \mathsf{vk}_\alpha, \mathsf{td}_\alpha) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing) \quad \text{and} \quad (\mathsf{hk}_{1-\alpha}, \mathsf{vk}_{1-\alpha}, \mathsf{td}_{1-\alpha}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1}). \tag{4.5}$$

  Both experiments check $\mathsf{H.Extract}(\mathsf{td}_{1-\alpha}, \mathsf{dig}_{1-\alpha}) = \mathsf{Matching}$. Thus, the only difference between $\mathsf{Hyb}_{i,\mathsf{final}}$ and Hyb_{i+1} is the distribution of hk_α. In this case, security reduces to the set hiding security of Π_{H}. We give this proof below.

To argue the second case, take any $i \in \{1, \ldots, d-1\}$ and suppose that $|\Pr[\mathsf{Hyb}_{i,\mathsf{final}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i+1}(\mathcal{A}) = 1]| = \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to build an efficient non-uniform adversary $\mathcal{B}$ that breaks set hiding of Π_{H} as follows:

- **Preprocessing phase:** On input the security parameter 1^λ, run $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Compute $\tau \leftarrow \mathsf{Preprocess}(C, P, \mathbf{x})$ and output $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$.
- **Online phase:** On input the state $\mathsf{st}_{\mathcal{B}} = (1^n, 1^{s_c}, 1^{s_p}, \tau, \mathsf{st}_{\mathcal{A}})$ where $\tau = \big(C, P, \mathbf{x}, (\beta_1, \ldots, \beta_s)\big)$, proceed as follows:
  1. Compute $J_i = \{j \in \mathsf{layer}_i(P) : \beta_j = 0\}$ and $J_{i+1} = \{j \in \mathsf{layer}_{i+1}(P) : \beta_j = 0\}$.
  2. Send 1^{s_p} and the set J_i to the challenger. The challenger replies with hk and vk.
  3. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'\big(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}\big)$, where s' is defined as in Construction 4.4. Let $\alpha = i \bmod 2$. Sample $(\mathsf{hk}_{1-\alpha}, \mathsf{vk}_{1-\alpha}, \mathsf{td}_{1-\alpha}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_{i+1})$ and set $\mathsf{hk}_\alpha \leftarrow \mathsf{hk}$, $\mathsf{vk}_\alpha \leftarrow \mathsf{vk}$.
  4. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and run $\pi \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$.
  5. Let $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. Output 1 if all of the following conditions hold (and 0 otherwise):

- $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
- $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
- $\mathsf{H.Extract}(\mathsf{td}_{1-\alpha}, \mathsf{dig}_{1-\alpha}) = \mathsf{Matching}$.

We now consider two possibilities:

- In $\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 0)$, the challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$. In this case, algorithm $\mathcal{B}$ samples crs according to the specification of Hyb_{i+1}. Moreover, algorithm $\mathcal{B}$ computes its output exactly as described in Hyb_{i+1}. This means that $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{i+1}(\mathcal{A})$ and thus

$$\Pr[\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 0) = 1] = \Pr[\mathsf{Hyb}_{i+1}(\mathcal{A}) = 1].$$

- In $\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 1)$, the challenger samples $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$. In this case, algorithm $\mathcal{B}$ samples crs according to the specification of $\mathsf{Hyb}_{i,\mathsf{final}}$. Moreover, algorithm $\mathcal{B}$ computes the output exactly as described in $\mathsf{Hyb}_{i,\mathsf{final}}$. This means that $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{i,\mathsf{final}}(\mathcal{A})$ and thus:

$$\Pr[\mathsf{ExptSH}_{\mathcal{B}}(\lambda, 1) = 1] = \Pr[\mathsf{Hyb}_{i,\mathsf{final}}(\mathcal{A}) = 1].$$

We conclude that algorithm $\mathcal{B}$ breaks the set hiding property with advantage ε in this case and the claim follows. □

Completing the proof. Combining Claims 4.8 to 4.16, we conclude that there exists a negligible function $\mu(\cdot)$ such that for all $i \in [d]$,

$$\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1] - O(1) \cdot |J_i| \cdot \mu(\lambda).$$

Moreover, by the same sequence, we conclude that

$$\Pr[\mathsf{Hyb}_{d-1,1,4}(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_{d-1}(\mathcal{A}) = 1] - O(1) \cdot \mu(\lambda).$$

Putting the pieces together,

$$\Pr[\mathsf{Hyb}_{d-1,1,4}(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - O(1) \cdot d \cdot |J_i| \cdot \mu(\lambda) = \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \mathsf{negl}(\lambda),$$

since $d \cdot |J_i| \leq s = \mathsf{poly}(\lambda)$. Note that we take $\mathsf{Hyb}_{d-1,1,4}$ to be our final hybrid since it imposes the *most* constraints (subsequent hybrids remove requirements from the experiment). To complete the proof we show that for all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_{d-1,1,4}(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Claim 4.17. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_{d-1,1,4}(\mathcal{A}) = 1] = 0$.*

Proof. Fix an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and let $(1^n, 1^{s_c}, 1^{s_p}, C, P, \mathbf{x}, \mathsf{st}_{\mathcal{A}})$ be the output of $\mathcal{A}_1(1^\lambda)$. Let $s = |P|$ and d be the depth of P. As usual, for $i \in [k]$, let $\beta_i = 1$ if $(C, x_i) \in \mathcal{L}_{\mathsf{CSAT}}$ and $\beta_i = 0$ otherwise. For $i \in [k+1, s]$, let β_i be the value of wire i in the evaluation of P on $(\beta_1, \ldots, \beta_k)$. By construction, all hybrids require that $\beta_s = P(\beta_1, \ldots, \beta_k) = 0$, and therefore $J_d[1] = \{\beta_s\}$. However, the conditions for $\widetilde{\mathsf{Hyb}}_{d-1,1,4}$ to output 1 cannot hold simultaneously:

- On the one hand, there must exist a witness $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w)$ for instance $\hat{x}_s$ (of the relation in Fig. 1) where $b = 0$.

- On the other hand, by definition of instance $\hat{x}_s$, since s is the output wire, it must be that $b = 1$.

Therefore $\widetilde{\mathsf{Hyb}}_{d-1,1,4}$ is unwinnable. □

Combining Eq. (4.5) and Claim 4.17, we conclude that there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$0 \geq \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \mathsf{negl}(\lambda).$$

This means that $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda)$, which proves Theorem 4.7. □

5 Generic Construction of Zero-Fixing Hash Functions

In this section, we show how to construct a zero-fixing hash function by combining an index BARG (Definition 2.9), an additively homomorphic encryption scheme with bounded support (Definition 2.1), and a vector encryption scheme with succinct local openings (Definition 2.3).

Binary tree indexing. In the following construction, we will work with complete binary trees. We will use the following procedure to associate a unique index with each node in the binary tree:

Definition 5.1 (Binary Tree Indexing)**.** Let $\mathcal{T}$ be a complete binary tree with $n = 2^k$ leaves. Then $\mathcal{T}$ contains exactly $2n - 1$ nodes. We associate a unique index $i \in [2n - 1]$ via the following procedure:

- First, associate the value $v = 1$ to the root node.
- If v is the value associated with a node, then associate values $2v$ and $2v+1$ with its left and right child. Recursively apply this process to assign a value to every node in the tree.
- The index i associated with a node is defined to be $2n - v$, where v is the value associated with the node.

By design, Definition 5.1 has the following properties:

- The leaf nodes are indexed 1 through n and the root node is indexed $2n - 1$.
- The index of every non-leaf node is greater than the index of its children.
- Given the index of any non-leaf node, we can efficiently compute the indices of its left and right child.

Construction 5.2 (Zero-Fixing Hash Function)**.** Our construction will rely on the following building blocks:

- Let $\Pi'_{\mathsf{BARG}} = (\mathsf{Gen}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{TrapGen}', \mathsf{Extract}')$ be a somewhere extractable *index* BARG (Definition 2.9).
- Let $\Pi_{\mathsf{HE}} = (\mathsf{HE.Gen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Add})$ be an additively homomorphic encryption scheme with bounded support (Definition 2.1). For a security parameter λ and a range parameter n, let $\ell_{\mathsf{ct}}(\lambda, n)$ be a bound on the length of the ciphertexts output by either $\mathsf{HE.Enc}(\mathsf{pk}, \cdot)$ or $\mathsf{HE.Add}(\mathsf{pk}, \cdot, \cdot)$ for any $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{HE.Gen}(1^\lambda, 1^n)$.
- Let $\Pi_{\mathsf{Com}} = (\mathsf{Com.Setup}, \mathsf{Com.Commit}, \mathsf{Com.Verify})$ be a vector commitment scheme with succinct local openings (Definition 2.3).

We construct a zero-fixing hash $\Pi_{\mathsf{H}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{ProveOpen}, \mathsf{VerOpen}, \mathsf{Extract}, \mathsf{ValidateDigest})$. In the following description, we assume without loss of generality that the bound on the input length $n \in \mathbb{N}$ is a power of two (i.e., $n = 2^k$ for some integer $k \in \mathbb{N}$). Next, we define the following NP relation which we will be using in our construction:

Statement: index $i \in [n]$
Witness: ciphertexts $\hat{v}^{(0)}, \hat{v}^{(1)}$, openings $\sigma^{(0)}, \sigma^{(1)}$, and an auxiliary witness $\tilde{w}$
Hard-coded: the common reference string $\mathsf{crs}_{\mathsf{Com}}$ for Π_{Com}, an index $i^* \in [n] \cup \{\bot\}$, a value $y \in \{0, 1, \bot\}$, and for each $b \in \{0, 1\}$, a public key pk_b for Π_{HE}, commitments $\mathsf{com}_{\mathsf{hk}}^{(b)}$ and two ciphertexts $\mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}$

On input a statement $i \in [n]$ and a witness $\left(\hat{v}^{(0)}, \hat{v}^{(1)}, \sigma^{(0)}, \sigma^{(1)}, \tilde{w}\right)$:

- **Leaf nodes:** If $i \in [n]$, then parse $\tilde{w} = \left(\hat{\mathsf{ct}}^{(0)}, \hat{\mathsf{ct}}^{(1)}, \sigma_{\mathsf{hk}}^{(0)}, \sigma_{\mathsf{hk}}^{(1)}\right)$. Output 1 if the following conditions hold:

  1. **Opening to ciphertext:** for $b \in \{0, 1\}$, $\mathsf{Com.Verify}\left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, i, \hat{v}^{(b)}, \sigma^{(b)}\right) = 1$.
  2. **Opening to ciphertext in** hk**:** for $b \in \{0, 1\}$, $\mathsf{Com.Verify}\left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(b)}, i, \hat{\mathsf{ct}}^{(b)}, \sigma_{\mathsf{hk}}^{(b)}\right) = 1$.
  3. **Consistent choice of ciphertexts:** $\left(\hat{v}^{(0)} = \mathsf{ct}_{\mathsf{zero}}^{(0)} \wedge \hat{v}^{(1)} = \mathsf{ct}_{\mathsf{zero}}^{(1)}\right)$ or $\left(\hat{v}^{(0)} = \hat{\mathsf{ct}}^{(0)} \wedge \hat{v}^{(1)} = \hat{\mathsf{ct}}^{(1)}\right)$.
  4. **Validity of ciphertext at target index:** If $i = i^*$, then *additionally* check that:

  $$\hat{v}^{(b)} = \begin{cases} \mathsf{ct}_{\mathsf{zero}}^{(b)} & y = 0 \\ \hat{\mathsf{ct}}^{(b)} & y = 1. \end{cases}$$

  If any of these conditions are not satisfied, output 0.

- **Non-leaf nodes:** If $i \in [n+1, 2n-1]$, then parse $\tilde{w} = (\tilde{w}_{\mathrm{L}}, \tilde{w}_{\mathrm{R}})$, where $\tilde{w}_d = \left(\hat{v}_d^{(0)}, \hat{v}_d^{(1)}, \sigma_d^{(0)}, \sigma_d^{(1)}\right)$ for $d \in \{\mathrm{L}, \mathrm{R}\}$. Output 1 if all of the following conditions hold for all $b \in \{0, 1\}$:

  1. **Opening to ciphertext:** $\mathsf{Com.Verify}\left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, i, \hat{v}^{(b)}, \sigma^{(b)}\right) = 1$.
  2. **Opening to child ciphertexts:** $\mathsf{Com.Verify}(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, i_{\mathrm{L}}, \hat{v}_{\mathrm{L}}^{(b)}, \sigma_{\mathrm{L}}^{(b)}) = 1$ and $\mathsf{Com.Verify}(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, i_{\mathrm{R}}, \hat{v}_{\mathrm{R}}^{(b)}, \sigma_{\mathrm{R}}^{(b)}) = 1$, where i_{L} and i_{R} are the indices of the left and right child of i (according to the indexing scheme from Definition 5.1).
  3. **Correctness of evaluation:** $\hat{v}^{(b)} = \mathsf{HE.Add}\left(\mathsf{pk}_b, \hat{v}_{\mathrm{L}}^{(b)}, \hat{v}_{\mathrm{R}}^{(b)}\right)$.
  4. **Validity of root:** If $i = 2n - 1$ then $\hat{v}^{(b)} = \mathsf{ct}_{\mathsf{root}}^{(b)}$.

  If any of these conditions are not satisfied, output 0.

Figure 2: The index relation $\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b \in \{0,1\}}, i^*, y\right]$.

We describe our construction below:

- $\mathsf{Setup}(1^\lambda, 1^n, S)$: On input a security parameter λ, the input length $n = 2^k$, and a set $S \subseteq [n]$, the setup algorithm starts by sampling the following:
  - Sample a pair of public keys: $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
  - Sample the CRS for the commitment scheme with block length $\ell_{\mathsf{ct}}(\lambda, n)$ and up to $2n-1$ blocks: $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.
  - Sample the CRS for an index BARG (that supports extractability on up to 3 positions): $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3)$, where s is a bound on the size of the circuit computing the index relation from Fig. 2. Note that since Π_{BARG} is an *index* BARG, Gen' does *not* separately take the statement length as input (Definition 2.9).

  Next, for each $b \in \{0, 1\}$, construct an encryption of 0: $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Next, for each $i \in S$ and $b \in \{0, 1\}$, construct the hash key ciphertexts as follows:

  - If $i \in S$, compute $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$.
  - If $i \notin S$, compute $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.

  Next, the setup algorithm constructs a commitment to the ciphertexts associated with the hash key. Specifically, for each $b \in \{0,1\}$, it computes

$$\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right).$$

  Finally, the setup algorithm constructs the hash key hk, the verification key vk, and the trapdoor td as follows:

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right) \tag{5.1}$$

$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right) \tag{5.2}$$

$$\mathsf{td} = (\mathsf{sk}_0, \mathsf{sk}_1). \tag{5.3}$$

- $\mathsf{Hash}(\mathsf{hk}, x)$: On input a hash key hk (parsed as in Eq. (5.1)) and a string $x \in \{0,1\}^n$, the hashing algorithm proceeds as follows:
  - Construct two complete binary trees $\mathcal{T}_0, \mathcal{T}_1$, each with n leaves. For each tree $\mathcal{T}_b$, we assign a ciphertext $v_i^{(b)}$ to each node $i \in [2s-1]$ in the tree as follows (where the nodes are indexed using Definition 5.1):
    * If $i \in [n]$, let $v_i^{(b)} \leftarrow \mathsf{ct}_{\mathsf{zero}}^{(b)}$ if $x_i = 0$ and $v_i^{(b)} \leftarrow \mathsf{ct}_i^{(b)}$ if $x_i = 1$.
    * For each internal node $i \in [n+1, 2n-1]$, let $v_i^{(b)} \leftarrow \mathsf{HE.Add}\left(\mathsf{pk}_b, v_{i_{\mathrm{L}}}^{(b)}, v_{i_{\mathrm{R}}}^{(b)}\right)$, where i_{L} and i_{R} are the indices associated with the left and right child of node i under the canonical tree indexing scheme (Definition 5.1).
  - For $b \in \{0,1\}$, construct commitments $(\mathsf{com}_b, \sigma_1^{(b)}, \ldots, \sigma_{2n-1}^{(b)}) \leftarrow \mathsf{Com.Commit}(\mathsf{crs}_{\mathsf{Com}}, (v_1^{(b)}, \ldots, v_{2n-1}^{(b)}))$ to the ciphertexts associated with $\mathcal{T}_b$.
  - For $b \in \{0,1\}$, let $\mathsf{ct}_{\mathsf{root}}^{(b)} = v_{2n-1}^{(b)}$ (i.e., the ciphertext associated with the root of $\mathcal{T}_b$). Let $C_\perp$ be the circuit that computes the following instantiation of the relation from Fig. 2:

$$\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b \in \{0,1\}}, \perp, \perp\right].$$

  - For each $i \in [2n-1]$, let $\tau_i = \left(v_i^{(0)}, v_i^{(1)}, \sigma_i^{(0)}, \sigma_i^{(1)}\right)$ be the opening for the ciphertexts associated with node i in $\mathcal{T}_0$ and $\mathcal{T}_1$. Then, for each $i \in [2s-1]$, define the auxiliary witness $\tilde{w}_i$ to be
    * If $i \in [n]$ then $\tilde{w}_i = \left(\mathsf{ct}_i^{(0)}, \mathsf{ct}_i^{(1)}, \sigma_{\mathsf{hk},i}^{(0)}, \sigma_{\mathsf{hk},i}^{(1)}\right)$.
    * If $i \in [n+1, 2n-1]$ then $\tilde{w}_i = (\tau_{i_{\mathrm{L}}}, \tau_{i_{\mathrm{R}}})$ where $i_{\mathrm{L}}, i_{\mathrm{R}}$ are the indices of the left and right child of node i, respectively.

    Finally, let $w_i = (\tau_i, \tilde{w}_i)$ for each $i \in [2n-1]$. Compute $\pi_{\mathsf{dig}} \leftarrow \mathsf{Prove}'(\mathsf{crs}_{\mathsf{BARG}}, C_\perp, 2n-1, (w_1, \ldots, w_{2n-1}))$.
  - Output the digest

$$\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right).$$

- $\mathsf{ProveOpen}(\mathsf{hk}, x, i^*)$: On input a hash key hk (parsed as in Eq. (5.1)), a string $x \in \{0,1\}^n$ and an index $i^* \in [n]$, the opening algorithm proceeds as follows:
  - Let $C_{i^*, x_{i^*}}$ be the circuit that computes the following instantiation of the relation from Fig. 2:

$$\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b \in \{0,1\}}, i^*, x_{i^*}\right].$$

  - Compute the witnesses w_i for each $i \in [2n-1]$ using the same procedure as in the Hash algorithm.

  - Output the opening $\sigma \leftarrow \mathsf{Prove}'(\mathsf{crs}_{\mathsf{BARG}}, C_{i^*,x_{i^*}}, 2n-1, (w_1, \ldots, w_{2n-1}))$

- $\mathsf{VerOpen}(\mathsf{vk}, \mathsf{dig}, i, b, \sigma)$: On input the verification key vk (parsed according to Eq. (5.2)), a digest $\mathsf{dig} = \big(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$, an index $i^* \in [n]$, a bit $b \in \{0,1\}$ and an opening σ, the verification algorithm outputs $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,b}, 2n-1, \sigma)$ where $C_{i^*,b}$ is the circuit computing the following relation from Fig. 2:
$$\mathcal{R}\big[\mathsf{crs}_{\mathsf{Com}}, \big\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\big\}_{b\in\{0,1\}}, i^*, b\big].$$

- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$: On input a trapdoor $\mathsf{td} = (\mathsf{sk}_0, \mathsf{sk}_1)$ and a digest $\mathsf{dig} = \big(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$, the extraction algorithm outputs Matching if $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{root}}^{(0)}) = 0$. Otherwise, it outputs NotMatching.

- $\mathsf{ValidateDigest}(\mathsf{vk}, \mathsf{dig})$: On input the verification key vk (parsed according to Eq. (5.2)) and a digest $\mathsf{dig} = \big(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$, the digest-validation algorithm outputs $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_\perp, 2n-1, \pi_{\mathsf{dig}})$ where $C_\perp$ is the circuit computing the following relation from Fig. 2:
$$\mathcal{R}\big[\mathsf{crs}_{\mathsf{Com}}, \big\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\big\}_{b\in\{0,1\}}, \perp, \perp\big].$$

Theorem 5.3 (Correctness)**.** *Suppose Π_{Com} is correct and Π'_{BARG} is complete. Then, Construction 5.2 is correct.*

Proof. Take any $\lambda, n \in \mathbb{N}$ and $x \in \{0,1\}^n$. Let $i^* \in [n]$ be an index. Suppose $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \varnothing)$. Parse
$$\begin{aligned}
\mathsf{hk} &= \Big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \big\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\big\}_{b\in\{0,1\}}\Big)\\
\mathsf{vk} &= \big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\big)\\
\mathsf{td} &= (\mathsf{sk}_0, \mathsf{sk}_1).
\end{aligned}$$

We now show each property individually.

Opening correctness. Take any index $i^* \in [n]$ and let $\sigma \leftarrow \mathsf{ProveOpen}(\mathsf{hk}, x, i^*)$. By definition, this means $\sigma' \leftarrow \mathsf{Prove}'(\mathsf{crs}_{\mathsf{BARG}}, C_{i^*,x_{i^*}}, 2n-1, (w_1, \ldots, w_{2n-1}))$, where $C_{i^*,x_{i^*}}$ is the circuit that computes the index relation
$$\mathcal{R}\Big[\mathsf{crs}_{\mathsf{Com}}, \big\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\big\}_{b\in\{0,1\}}, i^*, x_{i^*}\Big]$$
from Fig. 2. By construction of ProveOpen (and by correspondence, Hash), $w_i = (\tau_i, \tilde{w}_i)$ and $\tau_i = \big(v_i^{(0)}, v_i^{(1)}, \sigma_i^{(0)}, \sigma_i^{(1)}\big)$. We now show that $C_{i^*,x_{i^*}}(i, w_i) = 1$ for all $i \in [2n-1]$:

- **Leaf nodes:** Suppose $i \in [n]$. Then, $\tilde{w}_i = \big(\mathsf{ct}_i^{(0)}, \mathsf{ct}_i^{(1)}, \sigma_{\mathsf{hk},i}^{(0)}, \sigma_{\mathsf{hk},i}^{(1)}\big)$. Consider each of the conditions:
  1. **Opening to ciphertext:** By construction of Hash, for $b \in \{0,1\}$, the commitment com_b is a vector commitment to $\big(v_1^{(b)}, \ldots, v_{2n-1}^{(b)}\big)$, and the opening for position i is $\sigma_i^{(b)}$. This check follows by correctness of the vector commitment scheme.
  2. **Opening to ciphertext in hk:** By construction of Setup, for $b \in \{0,1\}$, the commitment com_b is a vector commitment to $\big(\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}\big)$ with opening $\sigma_{\mathsf{hk},i}^{(b)}$. This check follows by correctness of the vector commitment scheme.
  3. **Consistent choice of ciphertexts:** By construction of Hash, for $b \in \{0,1\}$, we have that depending on the value of x_i, either $v_i^{(b)} = \mathsf{ct}_{\mathsf{zero}}^{(b)}$ or $v_i^{(b)} = \mathsf{ct}_i^{(b)}$, and the check passes.
  4. **Validity of ciphertext at target index:** By construction of Hash, $v_{i^*}^{(b)} = \mathsf{ct}_{\mathsf{zero}}^{(b)}$ if $x_{i^*} = 0$ and $v_{i^*}^{(b)} = \mathsf{ct}^{(b)}$ if $x_{i^*} = 1$. As such, this check passes.

- **Non-leaf nodes:** Suppose $i \in [n+1, 2n-1]$. Then, $\tilde{w}_i = \big(\tau_{i_{\mathsf{L}}}, \tau_{i_{\mathsf{R}}}\big)$. Consider each of the conditions:

1. **Opening to ciphertext:** This follows by the same reason as above.
2. **Opening to child ciphertexts:** This also follows by construction of com_b (namely, com_b is a vector commitment to $(\hat{v}_1^{(b)}, \ldots, \hat{v}_{2n-1}^{(b)})$ with openings $\sigma_1^{(b)}, \ldots, \sigma_{2n-1}^{(b)}$).
3. **Correctness of evaluation:** By construction of Hash, for all non-leaf nodes $i \in [n+1, 2n-1]$, it holds that $v_i^{(b)} \leftarrow \mathsf{HE.Add}(\mathsf{pk}_b, v_{i_{\mathrm{L}}}^{(b)}, v_{i_{\mathrm{R}}}^{(b)})$, and the checks passes (since HE.Add is deterministic).
4. **Validity of root:** The Hash algorithm defines $\mathsf{ct}_{\mathsf{root}}^{(b)} = v_{2n-1}^{(b)}$, so this condition is trivially satisfied.

Since $C_{i^*, x_{i^*}}(i, w_i) = 1$ for all $i \in [2n-1]$, correctness follows by completeness of Π'_{BARG}.

Digest correctness. This follows by an analogous argument as that used to argue opening correctness, with the one difference being the circuit $C_\perp$ computes the the index relation

$$\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b \in \{0,1\}}, \perp, \perp\right].$$

In other words, $C_\perp$ (as defined in Hash and in ValidateDigest) does not define a target index i^* or value $b \in \{0,1\}$, and thus checks a *strict* subset of the conditions as $C_{i^*, x_{i^*}}$ defined in VerOpen. Finally, the witness $(w_1, \ldots, w_{2n-1})$ is defined in an identical manner as before, so all of the required conditions checked by $C_\perp$ are satisfied. □

Theorem 5.4 (Succinctness). *If Π_{HE} is compact and Π_{Com}, Π_{BARG} are succinct, then Construction 5.2 is succinct.*

Proof. Take any $\lambda, n \in \mathbb{N}$ and any $(\mathsf{hk}, \mathsf{vk}, \mathsf{td})$ in the support of $\mathsf{Setup}(1^\lambda, 1^n, \varnothing)$. Take any input $x \in \{0,1\}^n$ and index $i \in [n]$, and let $\mathsf{dig} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$, $\pi_{\mathsf{ProveOpen}} \leftarrow \mathsf{ProveOpen}(\mathsf{hk}, x, i)$. Parse

$$\begin{aligned} \mathsf{hk} &= \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right) \\ \mathsf{vk} &= \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right) \\ \mathsf{td} &= (\mathsf{sk}_0, \mathsf{sk}_1) \\ \mathsf{dig} &= \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{Hash}}\right) \end{aligned}$$

By compactness of Π_{HE}, the lengths of the public keys $\mathsf{pk}_0, \mathsf{pk}_1$ as well as the ciphertexts $\mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_i^{(b)}$ for all $i \in [n]$ and $b \in \{0,1\}$ are bounded by $\mathrm{poly}(\lambda + \log n)$. By succinctness of Π_{Com}, it holds that $\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}, \mathsf{com}_0$ and com_1 all have length $\mathrm{poly}(\lambda + \log n)$. Next, let s be a bound on the size of the circuits computing the relation in Fig. 2 The relation in Fig. 2 requires a constant number of opening of ciphertext checks, each of which can be implemented by a circuit of size $\mathrm{poly}(\lambda + \log n)$. Similarly, the correctness of homomorphic evaluation check and the constant number of ciphertext comparisons also require a circuit of size $\mathrm{poly}(\lambda + \log n)$. Thus, the size s of the circuit in Fig. 2 is bounded by $\mathrm{poly}(\lambda + \log n)$. By succinctness of Π_{BARG}, it holds that the length of the verification key $\mathsf{vk}_{\mathsf{BARG}}$ and the proofs π_{Hash} and $\pi_{\mathsf{ProveOpen}}$ have size $\mathrm{poly}(\lambda + \log n)$. In total, everything is polynomial in $\mathrm{poly}(\lambda + \log n)$ and therefore all of the succinctness requirements (Definition 3.1) are satisfied by Construction 5.2. □

Security. In the subsequent sections, we prove each of the required security properties on Construction 5.2. Instantiating the underlying batch argument (Fact 2.7), the additively homomorphic encryption (Fact 2.2), and the vector commitment scheme with existing constructions (Fact 2.4), we obtain the following corollary:

Corollary 5.5 (Zero-Fixing Hash Functions). *Assuming any of (1) the plain LWE assumption, (2) the k-Lin assumption over pairing groups for any constant k, or (3) the (sub-exponential) DDH assumption in pairing-free groups, there exists a zero-fixing hash function.*

Theorem 1.1 now follows in conjunction with our generic construction (Construction 4.4).

5.1 Security Analysis of Construction 5.2

In this section, we prove that Construction 5.2 satisfies the security requirements on a zero-fixing hash function.

5.1.1 Additive Invariants on Ciphertexts and Predicate Propagation

At a high level, the different security properties of the zero-fixing hash function (zero fixing, extractor validity, and index hiding with extracted guess) will rely on reasoning about various properties on the ciphertext associated with the root node in our tree of ciphertexts (i.e., the hash digest). The analysis of each of these properties follow a similar strategy where we first establish that a certain predicate holds for the ciphertexts in the leaves (i.e., the *honestly-generated* ciphertexts in the hash key). Then, we appeal to the security of the BARG to "propagate" the invariants to the root ciphertext. In this section, we describe a general abstraction for this predicate-propagation strategy that will help unify the analysis of the different security requirements. This construction exploits the fact that the ciphertext tree is perfectly balanced and has depth $\log n$ (where n is the input length); as such, we can rely on a similar type of inductive analysis as that in [BBK+23] for arguing soundness of a monotone policy BARG for *log-depth* predicates. We start by formally defining the type of invariants we consider in our security analysis.

Definition 5.6 (Tree-Based Additive Invariant on Ciphertexts)**.** Let n be a power of two and let $\Pi_{\mathsf{HE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ be a homomorphic encryption scheme. We say that an efficiently-computable predicate $P \colon \{0,1\}^* \to \{0,1\}$ is a tree-based additive invariant for Π_{HE} if for all $\lambda, n \in \mathbb{N}$, all indices $i^* \in [n] \cup \{\bot\}$, all key-pairs $(\mathsf{sk}_0, \mathsf{pk}_0)$, $(\mathsf{sk}_1, \mathsf{pk}_1)$ in the support of $\mathsf{Gen}(1^\lambda, 1^n)$, all indices $j, j_{\mathrm{L}}, j_{\mathrm{R}} \in [2n-1]$ where j_{L} and j_{R} are the children of j according to the indexing scheme in Definition 5.1, and all ciphertexts $\big(\mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}\big)$, $\big(\mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}\big)$ where

$$P\big(i^*, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}, \mathsf{sk}, \mathsf{sk}', j_{\mathrm{L}}\big) = 1 \quad \text{and} \quad P\big(i^*, \mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}, \mathsf{sk}, \mathsf{sk}', j_{\mathrm{R}}\big) = 1,$$

it holds that

$$P\big(i^*, \mathsf{ct}_{\mathsf{sum}}, \mathsf{ct}'_{\mathsf{sum}}, \mathsf{sk}, \mathsf{sk}', j\big) = 1,$$

where $\mathsf{ct}_{\mathsf{sum}}^{(0)} = \mathsf{Add}\big(\mathsf{pk}, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(0)}\big)$ and $\mathsf{ct}_{\mathsf{sum}}^{(1)} = \mathsf{Add}\big(\mathsf{pk}_1, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}\big)$. This implies that if P holds for the two children of a node, then it also holds for the parent node.

Predicate propagation experiment. We now define the general predicate propagation experiment we use in the analysis of Construction 5.2. This is a general experiment specification that captures the structure of the security definitions for a zero-fixing hash function.

Definition 5.7 (Predicate Propagation Experiment)**.** The predicate propagation experiment for Construction 5.2 is parameterized by the following two components:

- A tree-based additive invariant P (Definition 5.6) for the homomorphic encryption scheme Π_{HE}.
- An efficiently-computable "challenge-derivation" function $\mathsf{DeriveChal}(S, i)$ that takes as input a set $S \subseteq [n]$ and an index $i \in [n]$ and outputs two sets $S_0, S_1 \subseteq [n]$ and an index idx that is either a pair (i^*, y^*) or $\bot$. In the predicate propagation experiment, the sets S_0 and S_1 will determine the distribution of the ciphertexts in the common reference string. The index idx will determine the verification check. Each of the security properties (i.e., zero fixing, extractor validity, and index hiding with extracted guess) will induce a different choice of $\mathsf{DeriveChal}$ (to be specified in their respective proofs).

We now define the predicate propagation experiment $\mathsf{Expt}[P, \mathsf{DeriveChal}]$ between a challenger and an adversary $\mathcal{A}$:

1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$ (or a special symbol $\bot$).
2. The challenger computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$.
3. The challenger now samples the following quantities as in Setup:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.

- Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3)$, where s is a bound on the size of the circuit computing the index relation from Fig. 2.
- For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
- For each $b \in \{0,1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$

4. The challenger constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

The challenger gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.

6. The output of the experiment is 1 if

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1 \quad \text{and} \quad P\left(i^*, \mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\right) = 0.$$

Otherwise, the output is 0. Here, the circuit C_{idx} computes the relation from Fig. 2:

- If $\mathsf{idx} = (i, y)$, then C_{idx} computes the relation $\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b \in \{0,1\}}, i, y\right]$ as in VerOpen.
- If $\mathsf{idx} = \perp$, then C_{idx} computes the relation $\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b \in \{0,1\}}, \perp, \perp\right]$ as in ValidateDigest.

In words, the adversary "wins" the game if it produces a proof π that verifies, but the digest does *not* satisfy the tree-based additive invariant P.

Proof strategy. As we show in the subsequent sections (Sections 5.1.2 to 5.1.5), most of the security properties for the zero-fixing hash function (zero fixing, extractor validity, and index hiding with extracted guess) are a special case of the general predicate propagation experiment (with a suitable choice of the tree-based additive invariant P and the challenge-derivation function DeriveChal). Our goal below is to show that if specific "pre-conditions" hold, then for all efficient adversaries $\mathcal{A}$, the probability that $\mathsf{Expt}[P, \mathsf{DeriveChal}]$ outputs 1 is negligible. In turn, this will imply the desired security properties on the zero-fixing hash function.

Predicate propagation hybrid experiment. The proof π the adversary outputs is a BARG on $2n-1$ statements. We can associate these $2n-1$ statements with the nodes of a complete binary tree with n leaves. For each $j \in [2n-1]$, we now define an intermediate predicate propagation experiment $\mathsf{Expt}_j[P, \mathsf{DeriveChal}]$ where instead of checking the tree-based additive invariant holds for the values $\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}$ from the digest dig, the challenger instead checks the invariant for the value associated with node j in the tree obtained by *extracting* a witness from the BARG. In the subsequent analysis (Theorem 5.9), we show (inductively) that if the invariant holds for the values extracted from the children of a node j, then it also holds for the values extracted from node j itself. In this way, if the invariant P holds for all the values associated with the leaves of the tree, then the invariant also holds for the values associated with the root of the tree. Finally, the relation in Fig. 2 from Construction 5.2 enforces that the adversarially-chosen values $\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}$ are consistent with the values that would be extracted from the root of the tree. This allows us to reason about properties of the adversarially-chosen values $\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}$. We define this experiment below and then state the main predicate propagation theorem (Theorem 5.9).

Definition 5.8 (Predicate Propagation Hybrid Experiment)**.** Let $j \in \mathbb{N}$ be an index. For a tree-based additive invariant P and a challenge-derivation function DeriveChal, we define the predicate propagation hybrid experiment $\mathsf{Expt}_j[P, \mathsf{DeriveChal}]$ between a challenger and an adversary $\mathcal{A}$ as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$.
2. The challenger computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$.
3. The challenger now samples the following quantities as in Setup:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j\})$, where s is a bound on the size of the circuit computing the index relation from Fig. 2.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0,1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$
4. The challenger constructs hk and vk according to Eqs. (5.1) and (5.2):
$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b\in\{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$
   The challenger gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.
6. The challenger computes $\left(\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j\right) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j)$.
7. The output of the experiment is 1 if the following conditions hold:
$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1 \text{ and } C_{\mathsf{idx}}\left(j, (\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j)\right) = 1 \text{ and } P\left(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\right) = 0.$$
   Otherwise, the output is 0. As in Definition 5.7, the circuit C_{idx} computes the relation from Fig. 2:
   - If $\mathsf{idx} = (i, y)$, then C_{idx} computes the relation $\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b\in\{0,1\}}, i, y\right]$.
   - If $\mathsf{idx} = \perp$, then C_{idx} computes the relation $\mathcal{R}\left[\mathsf{crs}_{\mathsf{Com}}, \left\{\mathsf{pk}_b, \mathsf{com}_{\mathsf{hk}}^{(b)}, \mathsf{com}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_{\mathsf{root}}^{(b)}\right\}_{b\in\{0,1\}}, \perp, \perp\right]$.

   Otherwise, the output is 0.

Theorem 5.9 (Predicate Propagation). *Let P be a tree-based additive invariant and let* DeriveChal *be a challenge-derivation function. Suppose Π_{Com} satisfies computational binding and Π_{BARG} satisfies set hiding with extraction, set hiding, and somewhere extractability. Let $\mathcal{A}$ be any efficient adversary for the predicate propagation experiment. Suppose that for every index $j \in [n]$ (where $n = n(\lambda)$ is the input length chosen by $\mathcal{A}$), there exists a negligible function $\varepsilon_j(\cdot)$ such that*
$$\Pr[\mathsf{Expt}_j[P, \mathsf{DeriveChal}](\mathcal{A}) = 1] = \varepsilon_j(\lambda).$$
Then there exists a negligible function $\mathsf{negl}(\cdot)$ *such that*
$$\Pr[\mathsf{Expt}[P, \mathsf{DeriveChal}](\mathcal{A}) = 1] = \mathsf{negl}(\lambda).$$

Proof. To simplify notation, we write $\mathsf{Expt} := \mathsf{Expt}[P, \mathsf{DeriveChal}]$ and $\mathsf{Expt}_j := \mathsf{Expt}_j[P, \mathsf{DeriveChal}]$ in the following proof. Fix an adversary $\mathcal{A}$ and let n be the input length chosen by $\mathcal{A}$. We proceed by induction on the index $j \in [2n-1]$. In the following, we will view the index j as an index of a node in a (complete) binary tree with n leaves (indexed according to Definition 5.1). As such, we can refer to the "height" of an index j. Then, we show the following lemma:

Lemma 5.10. *Suppose the conditions of Theorem 5.9 hold. Take any index $j \in [2n-1]$ and let h be the height of node j (where the leaf nodes have height 0). Then, there exists a negligible function $\varepsilon_j(\lambda)$ such that*
$$\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] = 2^h \cdot \varepsilon_j(\lambda).$$

Proof. Suppose the conditions of Theorem 5.9 hold. We prove the lemma by induction on the height h of the index $j \in [2n-1]$.

Base case. For the indices $j \in [n]$ of height 0 (i.e., the leaves of the tree), the lemma follows by assumption.

Inductive step. Suppose the inductive hypothesis holds for every index $j' \in [2n-1]$ of height h. Let $j \in [2n-1]$ be an index with height $h+1$. Let $j_{\mathrm{L}}, j_{\mathrm{R}} \in [2n-1]$ be the indices of the left and right child of node j (as defined in Definition 5.1). By construction, j_{L} and j_{R} have height h. The inductive hypothesis now asserts that for $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$,

$$\Pr\left[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1\right] = 2^h \cdot \varepsilon_{j^*}(\lambda), \tag{5.4}$$

for some negligible function $\varepsilon_{j^*}(\lambda)$. We now define an intermediate experiment Expt'_j for each node j of height $h > 0$:

1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$.
2. The challenger computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$.
3. The challenger now samples the following quantities as in Setup:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\})$, where s is a bound on the size of the circuit computing the index relation from Fig. 2.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0,1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$
4. The challenger constructs hk and vk according to Eqs. (5.1) and (5.2):

   $$\begin{aligned}\mathsf{hk} &= \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b\in\{0,1\}}\right) \\ \mathsf{vk} &= \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)\end{aligned}$$

   The challenger gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.
6. The challenger computes $\left(\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j\right) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j)$.
7. The output of the experiment is 1 if the following conditions hold:

   $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$ and $C_{\mathsf{idx}}\left(j, (\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j)\right) = 1$ and $P\left(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\right) = 0$.

   Otherwise, the output is 0.

In our analysis below, we define an additional set of events in an execution of Expt'_j with $\mathcal{A}$. First, define the following two quantities:

- $\left(\hat{v}_{j_{\mathrm{L}}}^{(0)}, \hat{v}_{j_{\mathrm{L}}}^{(1)}, \sigma_{j_{\mathrm{L}}}^{(0)}, \sigma_{j_{\mathrm{L}}}^{(1)}, \tilde{w}_{j_{\mathrm{L}}}\right) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j_{\mathrm{L}})$.
- $\left(\hat{v}_{j_{\mathrm{R}}}^{(0)}, \hat{v}_{j_{\mathrm{R}}}^{(1)}, \sigma_{j_{\mathrm{R}}}^{(0)}, \sigma_{j_{\mathrm{R}}}^{(1)}, \tilde{w}_{j_{\mathrm{R}}}\right) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j_{\mathrm{R}})$.

Now, define the following events:

- $\mathsf{E}_{\mathsf{Verify}}^{(j)}$: This is the event that $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$.
- $\mathsf{E}_{P,j^*}^{(j)}$ for each $j^* \in \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$: This is the event where $P\left(i^*, \hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j^*\right) = 1$.

- $\mathsf{E}^{(j)}_{\mathsf{ValidCom},j^*}$ for each $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$: This is the event

$$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_0, j^*, \hat{v}^{(0)}_{j^*}, \sigma^{(0)}_{j^*}\big) = 1 = \mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_1, j^*, \hat{v}^{(1)}_{j^*}, \sigma^{(1)}_{j^*}\big).$$

- $\mathsf{E}^{(j)}_{\mathsf{SAT},j^*}$ for each $j^* \in \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$: This is the event $C_{\mathsf{idx}}\big(j^*, \big(\hat{v}^{(0)}_{j^*}, \hat{v}^{(1)}_{j^*}, \sigma^{(0)}_{j^*}, \sigma^{(1)}_{j^*}, \tilde{w}_{j^*}\big)\big) = 1$.

We now relate the probability that $\mathsf{Expt}_j(\mathcal{A})$ outputs 1 to the probability that $\mathsf{Expt}_{j_{\mathrm{L}}}(\mathcal{A})$ and $\mathsf{Expt}_{j_{\mathrm{R}}}(\mathcal{A})$ outputs 1. To do so, we first program the BARG to be extracting on the set $\{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$. We then argue via somewhere extractability of the BARG and computational binding of the commitment scheme that if the values associated with the nodes j_{L} and j_{R} satisfy the predicate P and the proof verifies, then the value associated with j must also satisfy the predicate P. In this case, the output of $\mathsf{Expt}_j(\mathcal{A})$ is guaranteed to be 0.

Claim 5.11. *If* Π_{BARG} *satisfies set hiding with extraction, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $j^* \in \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$*, it holds that*

$$\left|\Pr[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1] - \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \wedge \mathsf{E}^{(j)}_{\mathsf{SAT},j^*} \wedge \neg\mathsf{E}^{(j)}_{P,j^*}\right]\right| = \mathsf{negl}(\lambda).$$

Proof. Take any $j^* \in \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$ and suppose

$$\left|\Pr[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1] - \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \wedge \mathsf{E}^{(j)}_{\mathsf{SAT},j^*} \wedge \neg\mathsf{E}^{(j)}_{P,j^*}\right]\right| \geq \varepsilon(\lambda),$$

for some non-negligible ε. Importantly, note that the events $\mathsf{E}^{(j)}_{\mathsf{Verify}}$, $\mathsf{E}^{(j)}_{\mathsf{SAT},j^*}$, and $\mathsf{E}^{(j)}_{P,j^*}$ are defined for Expt'_j and *not* Expt_{j^*}. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the set hiding with extraction game of Π_{BARG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.
2. Algorithm $\mathcal{B}$ outputs $1^{2n-1}, 1^s, 1^3$, the challenge set $J = \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$, and the challenge index $j^* \in J$ to the challenger, where s is the bound on the size of the circuit in Fig. 2. The challenger responds with $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}})$.
3. Algorithm $\mathcal{B}$ computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}^{(b)}_{\mathsf{zero}} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}^{(b)}_i \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}^{(b)}_i \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0,1\}$, let $\big(\mathsf{com}^{(b)}_{\mathsf{hk}}, \sigma^{(b)}_{\mathsf{hk},1}, \ldots, \sigma^{(b)}_{\mathsf{hk},n}\big) \leftarrow \mathsf{Com.Commit}\big(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}^{(b)}_1, \ldots, \mathsf{ct}^{(b)}_n)\big)$
4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \Big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \big\{\mathsf{pk}_b, \mathsf{ct}^{(b)}_{\mathsf{zero}}, \mathsf{ct}^{(b)}_1, \ldots, \mathsf{ct}^{(b)}_n, \sigma^{(b)}_{\mathsf{hk},1}, \ldots, \sigma^{(b)}_{\mathsf{hk},n}\big\}_{b\in\{0,1\}}\Big)$$
$$\mathsf{vk} = \big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}^{(0)}_{\mathsf{zero}}, \mathsf{ct}^{(1)}_{\mathsf{zero}}, \mathsf{com}^{(0)}_{\mathsf{hk}}, \mathsf{com}^{(1)}_{\mathsf{hk}}\big)$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \big(\mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$ and a proof π.
6. Let C_{idx} be the circuit as defined in Definition 5.7. Algorithm $\mathcal{B}$ first checks

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1.$$

   If the check fails, algorithm $\mathcal{B}$ aborts with output $\perp$. Otherwise, algorithm $\mathcal{B}$ sends the circuit C_{idx}, the instance number $2n-1$, and the proof π to the challenger. The challenger replies with a string which $\mathcal{B}$ parses as $\big(\hat{v}^{(0)}_{j^*}, \hat{v}^{(1)}_{j^*}, \sigma^{(0)}_{j^*}, \sigma^{(1)}_{j^*}, \tilde{w}_{j^*}\big)$.

7. Algorithm $\mathcal{B}$ outputs 1 if

$$C_{\mathsf{idx}}\big(j^*, \big(\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*}\big)\big) = 1 \quad \text{and} \quad P\big(i^*, \hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0.$$

   Otherwise, algorithm $\mathcal{B}$ outputs 0.

Let $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}})$ be the parameters sampled by the challenger in the set hiding with extraction game. In the game, after $\mathcal{B}$ outputs $(C_{\mathsf{idx}}, 2n-1, \pi)$, the challenger checks $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$. If the check passes, it replies with $\big(\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*}\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j^*)$. We now consider the two possibilities:

- Suppose the challenger responds according to the specification of $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 0)$. In this case, the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\})$. Thus, algorithm $\mathcal{B}$ perfectly simulates for $\mathcal{A}$ an execution of Expt'_j. We claim that algorithm $\mathcal{B}$ outputs 1 if and only if the event $\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \mathsf{E}_{\mathsf{SAT},j^*}^{(j)} \wedge \neg \mathsf{E}_{P,j^*}^{(j)}$ occurs. This event corresponds to the following set of conditions:

  $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$ and $C_{\mathsf{idx}}\big(j^*, (\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*})\big) = 1$ and $P\big(i^*, \hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j^*\big) = 0.$

  where $\big(\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*}\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j^*)$. This is the same set of conditions that algorithm $\mathcal{B}$ checks, so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr\big[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \mathsf{E}_{\mathsf{SAT},j^*}^{(j)} \wedge \neg \mathsf{E}_{P,j^*}^{(j)}\big]$ in this case.

- Suppose the challenger responds according to the specification of $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 1)$. In this case, the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j^*\})$. Thus, algorithm $\mathcal{B}$ simulates for $\mathcal{A}$ an execution of Expt_{j^*}. We claim that algorithm $\mathcal{B}$ outputs 1 if and only if $\mathsf{Expt}_{j^*}(\mathcal{A})$ outputs 1. The latter corresponds to the following conditions being satisfied:

  $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$ and $C_{\mathsf{idx}}\big(j^*, (\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*})\big) = 1$ and $P\big(i^*, \hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j^*\big) = 0.$

  where $\big(\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*}\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j^*)$. Once again, this is the same set of conditions that $\mathcal{B}$ checks. Thus, in this case algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1]$.

We conclude that the distinguishing advantage of $\mathcal{B}$ is precisely

$$\Big|\Pr[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1] - \Pr\big[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \mathsf{E}_{\mathsf{SAT},j^*}^{(j)} \wedge \neg \mathsf{E}_{P,j^*}^{(j)}\big]\Big| = \varepsilon,$$

which completes the proof. □

Claim 5.12. *If* Π_{BARG} *is somewhere extractable then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $j^* \in \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$, *it holds that* $\Pr\big[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \neg \mathsf{E}_{\mathsf{SAT},j^*}^{(j)}\big] = \mathsf{negl}(\lambda)$.

Proof. Take any $j^* \in \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$ and suppose $\Pr\big[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \neg \mathsf{E}_{\mathsf{SAT},j^*}^{(j)}\big] \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks somewhere extractability of Π_{BARG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.

2. Algorithm $\mathcal{B}$ outputs $1^{2n-1}, 1^s, 1^3$, the challenge set $J = \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$, and the challenge index $j^* \in J$ to the challenger, where s is the bound on the size of the circuit in Fig. 2. The challenger responds with $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}})$.

3. Algorithm $\mathcal{B}$ computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:

   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.

- For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
- For each $b \in \{0,1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$

4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.
6. Let C_{idx} be the circuit as defined in Definition 5.7. Algorithm $\mathcal{B}$ outputs the circuit C_{idx}, the instance number $2n-1$, and the proof π.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of Expt_j. Thus, with probability at least ε, the digest dig and proof π output by $\mathcal{A}$ satisfies $\mathsf{E}_{\mathsf{Verify}}^{(j)}$ but not $\mathsf{E}_{\mathsf{SAT},j^*}^{(j)}$. This means

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1 \quad \text{and} \quad C_{\mathsf{idx}}\left(j^*, \left(\hat{v}_{j^*}^{(0)}, \hat{v}_{j^*}^{(1)}, \sigma_{j^*}^{(0)}, \sigma_{j^*}^{(1)}, \tilde{w}_{j^*}\right)\right) = 0.$$

This means algorithm $\mathcal{B}$ successfully breaks somewhere extractability of Π_{BARG} and the claim holds. □

Claim 5.13. *Suppose the conditions in Claims 5.11 and 5.12 hold. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \left(\neg\mathsf{E}_{\mathsf{ValidCom},j_{\mathrm{L}}}^{(j)} \vee \neg\mathsf{E}_{P,j_{\mathrm{L}}}^{(j)} \vee \neg\mathsf{E}_{\mathsf{ValidCom},j_{\mathrm{R}}}^{(j)} \vee \neg E_{P,j_{\mathrm{R}}}^{(j)}\right)\right] \le 2^{h+1} \cdot \varepsilon_j(\lambda) + \mathsf{negl}(\lambda),$$

where $\varepsilon_j(\lambda) = \max(\varepsilon_{j_{\mathrm{L}}}(\lambda), \varepsilon_{j_{\mathrm{R}}}(\lambda))$.

Proof. By Claim 5.11 there exists a negligible function $\mathsf{negl}_1(\cdot)$ such that for all $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$, it holds that:

$$\left|\Pr[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1] - \Pr\left[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \mathsf{E}_{\mathsf{SAT},j^*}^{(j)} \wedge \neg\mathsf{E}_{P,j^*}^{(j)}\right]\right| \le \mathsf{negl}_1(\lambda). \tag{5.5}$$

By Claim 5.12 there exists a negligible function $\mathsf{negl}_2(\cdot)$ such that for all $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$ it holds that

$$\Pr\left[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \neg\mathsf{E}_{\mathsf{SAT},j^*}^{(j)}\right] \le \mathsf{negl}_2(\lambda). \tag{5.6}$$

By definition, if $\mathsf{Expt}'_j(\mathcal{A}) = 1$, then event $\mathsf{E}_{\mathsf{Verify}}^{(j)}$ also occurs. Thus, for all events E, it holds that

$$\Pr[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \mathsf{E}] \le \Pr\left[\mathsf{E}_{\mathsf{Verify}}^{(j)} \wedge \mathsf{E}\right]. \tag{5.7}$$

Similarly, by construction of the circuit C_{idx}, the event $\neg\mathsf{E}_{\mathsf{ValidCom},j^*}^{(j)}$ implies event $\neg\mathsf{E}_{\mathsf{SAT},j^*}^{(j)}$. Thus, for any event E, it holds that

$$\Pr\left[\neg\mathsf{E}_{\mathsf{ValidCom},j^*}^{(j)} \wedge \mathsf{E}\right] \le \Pr\left[\neg\mathsf{E}_{\mathsf{SAT},j^*}^{(j)} \wedge \mathsf{E}\right]. \tag{5.8}$$

Take any $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$. Since the height of j^* is h, the inductive hypothesis applies and Eq. (5.4) holds. We first show that

$$\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \neg\mathsf{E}_{P,j^*}^{(j)}\right] \le 2^h \cdot \varepsilon_{j^*}(\lambda) + \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda). \tag{5.9}$$

This follows by the following sequence of calculations:

$$\begin{aligned}
\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \land \neg \mathsf{E}^{(j)}_{P,j^*}\right] &\le \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \land \neg \mathsf{E}^{(j)}_{P,j^*}\right] && \text{by Eq. (5.7)} \\
&= \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \land \mathsf{E}^{(j)}_{\mathsf{SAT},j^*} \land \neg \mathsf{E}^{(j)}_{P,j^*}\right] + \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \land \neg \mathsf{E}^{(j)}_{\mathsf{SAT},j^*} \land \neg \mathsf{E}^{(j)}_{P,j^*}\right] && \\
&\le \Pr[\mathsf{E}^{(j)}_{\mathsf{Verify}} \land \mathsf{E}^{(j)}_{\mathsf{SAT},j^*} \land \neg \mathsf{E}^{(j)}_{P,j^*}] + \mathsf{negl}_2(\lambda) && \text{by Eq. (5.6)} \\
&\le \Pr[\mathsf{Expt}_{j^*}(\mathcal{A}) = 1] + \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) && \text{by Eq. (5.5)} \\
&\le 2^h \cdot \varepsilon_{j^*}(\lambda) + \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) && \text{by Eq. (5.4).}
\end{aligned}$$

Next, we have

$$\begin{aligned}
\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \land \neg \mathsf{E}_{\mathsf{ValidCom},j^*}\right] &\le \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \land \neg \mathsf{E}_{\mathsf{ValidCom},j^*}\right] && \text{by Eq. (5.7)} \\
&\le \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \land \neg \mathsf{E}_{\mathsf{SAT},j^*}\right] && \text{by Eq. (5.8)} \\
&\le \mathsf{negl}_2(\lambda) && \text{by Eq. (5.6).}
\end{aligned}$$

Combined with Eq. (5.9) and applying a union bound, we have

$$\begin{aligned}
\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \land \left(\neg \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \lor \neg \mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \lor \neg \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \lor \neg E^{(j)}_{P,j_{\mathrm{R}}}\right)\right] &\le 2^h \cdot \left(\varepsilon_{j_{\mathrm{L}}}(\lambda) + \varepsilon_{j_{\mathrm{R}}}(\lambda)\right) + \delta(\lambda) \\
&\le 2^{h+1} \cdot \varepsilon_j(\lambda) + \delta(\lambda),
\end{aligned}$$

where $\delta(\lambda) = 2\mathsf{negl}_1(\lambda) + 4\mathsf{negl}_2(\lambda) = \mathsf{negl}(\lambda)$ and $\varepsilon_j(\lambda) = \max(\varepsilon_{j_{\mathrm{L}}}(\lambda), \varepsilon_{j_{\mathrm{R}}}(\lambda))$. □

Claim 5.14. *If P is a tree-based additive invariant and Π_{Com} is computationally binding, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \land \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \land \mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \land \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \land \mathsf{E}^{(j)}_{P,j_{\mathrm{R}}}\right] \le \mathsf{negl}(\lambda).$$

Proof. Suppose

$$\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \land \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \land \mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \land \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \land \mathsf{E}^{(j)}_{P,j_{\mathrm{R}}}\right] \ge \varepsilon(\lambda),$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the binding game for Π_{Com} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.
2. Algorithm $\mathcal{B}$ outputs the block length $1^{\ell_{\mathsf{ct}}(\lambda,n)}$ and the vector length $2n-1$ to the challenger. The challenger responds with $\mathsf{crs}_{\mathsf{Com}}$.
3. Algorithm $\mathcal{B}$ computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\})$.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}^{(b)}_{\mathsf{zero}} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}^{(b)}_i \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}^{(b)}_i \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0,1\}$, let $\left(\mathsf{com}^{(b)}_{\mathsf{hk}}, \sigma^{(b)}_{\mathsf{hk},1}, \ldots, \sigma^{(b)}_{\mathsf{hk},n}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}^{(b)}_1, \ldots, \mathsf{ct}^{(b)}_n)\right)$
4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

   $$\begin{aligned}
   \mathsf{hk} &= \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \{\mathsf{pk}_b, \mathsf{ct}^{(b)}_{\mathsf{zero}}, \mathsf{ct}^{(b)}_1, \ldots, \mathsf{ct}^{(b)}_n, \sigma^{(b)}_{\mathsf{hk},1}, \ldots, \sigma^{(b)}_{\mathsf{hk},n}\}_{b\in\{0,1\}}\right) \\
   \mathsf{vk} &= \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}^{(0)}_{\mathsf{zero}}, \mathsf{ct}^{(1)}_{\mathsf{zero}}, \mathsf{com}^{(0)}_{\mathsf{hk}}, \mathsf{com}^{(1)}_{\mathsf{hk}}\right)
   \end{aligned}$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \big(\mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$ and a proof π.
6. Algorithm $\mathcal{B}$ computes the following:
   - $\big(\hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \sigma^{(0)}_j, \sigma^{(1)}_j, \tilde{w}_j\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j)$.
   - $\big(\hat{v}^{(0)}_{j_{\mathrm{L}}}, \hat{v}^{(1)}_{j_{\mathrm{L}}}, \sigma^{(0)}_{j_{\mathrm{L}}}, \sigma^{(1)}_{j_{\mathrm{L}}}, \tilde{w}_{j_{\mathrm{L}}}\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j_{\mathrm{L}})$.
   - $\big(\hat{v}^{(0)}_{j_{\mathrm{R}}}, \hat{v}^{(1)}_{j_{\mathrm{R}}}, \sigma^{(0)}_{j_{\mathrm{R}}}, \sigma^{(1)}_{j_{\mathrm{R}}}, \tilde{w}_{j_{\mathrm{R}}}\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j_{\mathrm{R}})$.

   In addition, it parses $\tilde{w}_j = (\tilde{w}_{j,\mathrm{L}}, \tilde{w}_{j,\mathrm{R}})$ and $\tilde{w}_{\mathrm{L}} = \big(\hat{v}^{(0)}_{j,\mathrm{L}}, \hat{v}^{(1)}_{j,\mathrm{L}}, \sigma^{(0)}_{j,\mathrm{L}}, \sigma^{(1)}_{j,\mathrm{L}}\big)$ and $\tilde{w}_{\mathrm{R}} = \big(\hat{v}^{(0)}_{j,\mathrm{R}}, \hat{v}^{(1)}_{j,\mathrm{R}}, \sigma^{(0)}_{j,\mathrm{R}}, \sigma^{(1)}_{j,\mathrm{R}}\big)$.
7. Algorithm $\mathcal{B}$ checks if there exists $b \in \{0,1\}$ and $d \in \{\mathrm{L}, \mathrm{R}\}$ such that $\hat{v}^{(b)}_{j_d} \neq \hat{v}^{(b)}_{j,d}$ and

   $$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, j_d, \hat{v}^{(b)}_{j,d}, \sigma^{(b)}_{j,d}\big) = 1 \quad \text{and} \quad \mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, j_d, \hat{v}^{(b)}_{j_d}, \sigma^{(b)}_{j_d}\big) = 1.$$

   If so, it outputs the commitment com_b, the index $j_d \in [2n-1]$, and the value-opening pairs $\big(\hat{v}^{(b)}_{j,d}, \sigma^{(b)}_{j,d}\big)$ and $\big(\hat{v}^{(b)}_{j_d}, \sigma^{(b)}_{j_d}\big)$. Otherwise, algorithm $\mathcal{B}$ aborts with output $\bot$.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of Expt'_j for adversary $\mathcal{A}$. By assumption, with probability at least ε, algorithm $\mathcal{A}$ will output a digest dig and a proof π such that the following conditions hold:

- $\mathsf{Expt}'_j(\mathcal{A}) = 1$: This means $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$, $C_{\mathsf{idx}}\big(j, (\hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \sigma^{(0)}_j, \sigma^{(1)}_j, \tilde{w}_j)\big) = 1$, and $P\big(\hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0$.
- $\mathsf{E}^{(j)}_{\mathsf{ValidCom}, j^*}$ for $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$: This means

  $$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_0, j^*, \hat{v}^{(0)}_{j^*}, \sigma^{(0)}_{j^*}\big) = 1 = \mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_1, j^*, \hat{v}^{(1)}_{j^*}, \sigma^{(1)}_{j^*}\big).$$

- $\mathsf{E}^{(j)}_{P, j^*}$ for $j^* \in \{j_{\mathrm{L}}, j_{\mathrm{R}}\}$: This means $P\big(\hat{v}^{(0)}_{j^*}, \hat{v}^{(1)}_{j^*}, \mathsf{sk}_0, \mathsf{sk}_1, j^*\big) = 1$.

We consider two possibilities:

- Suppose for all $b \in \{0,1\}$, we have $\hat{v}^{(b)}_{j_{\mathrm{L}}} = \hat{v}^{(b)}_{j,\mathrm{L}}$ and $\hat{v}^{(b)}_{j_{\mathrm{R}}} = \hat{v}^{(b)}_{j,\mathrm{R}}$. Since $C_{i^*,y}\big(j, (\hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \sigma^{(0)}_j, \sigma^{(1)}_j, \tilde{w}_j)\big) = 1$, this means that $\hat{v}^{(b)}_j = \mathsf{HE.Add}\big(\mathsf{pk}_b, \hat{v}^{(b)}_{j,\mathrm{L}}, \hat{v}^{(b)}_{j,\mathrm{R}}\big)$ for all $b \in \{0,1\}$. Combined with the third condition, this means

  $$P\big(i^*, \hat{v}^{(0)}_{j,\mathrm{L}}, \hat{v}^{(1)}_{j,\mathrm{L}}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{L}}\big) = P\big(i^*, \hat{v}^{(0)}_{j_{\mathrm{L}}}, \hat{v}^{(1)}_{j_{\mathrm{L}}}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{L}}\big) = 1$$
  $$P\big(i^*, \hat{v}^{(0)}_{j,\mathrm{R}}, \hat{v}^{(1)}_{j,\mathrm{R}}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{R}}\big) = P\big(i^*, \hat{v}^{(0)}_{j_{\mathrm{R}}}, \hat{v}^{(1)}_{j_{\mathrm{R}}}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{R}}\big) = 1.$$

  Since P is a tree-based additive invariant, this means that

  $$P\big(i^*, \hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 1.$$

  However, this contradicts the condition that $P\big(i^*, \hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0$, so this case does not occur.
- Suppose there exists $b \in \{0,1\}$ and $d \in \{\mathrm{L}, \mathrm{R}\}$ where $\hat{v}^{(b)}_{j_d} \neq \hat{v}^{(b)}_{j,d}$. Since $C_{i^*,y}\big(j, (\hat{v}^{(0)}_j, \hat{v}^{(1)}_j, \sigma^{(0)}_j, \sigma^{(1)}_j, \tilde{w}_j)\big) = 1$, this means that $\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, j_d, \hat{v}^{(b)}_{j,d}, \sigma^{(b)}_{j,d}\big) = 1$. By the second condition, we also have

  $$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_b, j_d, \hat{v}^{(b)}_{j_d}, \sigma^{(b)}_{j_d}\big) = 1.$$

  In this case, algorithm $\mathcal{B}$ outputs the commitment com_b, the index j_d, and the value-opening pairs $\big(\hat{v}^{(b)}_{j,d}, \sigma^{(b)}_{j,d}\big)$ and $\big(\hat{v}^{(b)}_{j_d}, \sigma^{(b)}_{j_d}\big)$. This is a pair of valid openings for com_b so algorithm $\mathcal{B}$ wins the binding game.

We conclude that algorithm $\mathcal{B}$ succeeds with the same advantage ε and the claim follows. □

Claim 5.15. *Suppose the conditions of Claims 5.13 and 5.14 hold. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathsf{Expt}'_j(\mathcal{A}) = 1] \leq 2^{h+1} \cdot \varepsilon_j(\lambda) + \mathsf{negl}(\lambda),$$

where $\varepsilon_j(\lambda) = \max(\varepsilon_{j_{\mathrm{L}}}(\lambda), \varepsilon_{j_{\mathrm{R}}}(\lambda))$.

Proof. By the law of total probability, we have

$$\begin{aligned}\Pr[\mathsf{Expt}'_j(\mathcal{A}) = 1] \leq \Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \wedge \mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \wedge \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \wedge \mathsf{E}^{(j)}_{P,j_{\mathrm{R}}}\right] + \\ \Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \left(\neg\mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \vee \neg\mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \vee \neg\mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \vee \neg\mathsf{E}^{(j)}_{P,j_{\mathrm{R}}}\right)\right].\end{aligned}$$

By Claims 5.13 and 5.14, there exist negligible functions $\mathsf{negl}_1(\cdot)$ and $\mathsf{negl}_2(\cdot)$ such that:

$$\begin{aligned}\Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \left(\neg\mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \vee \neg\mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \vee \neg\mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \vee \neg\mathsf{E}^{(j)}_{P,j_{\mathrm{R}}}\right)\right] \leq 2^{h+1} \cdot \varepsilon_j(\lambda) + \mathsf{negl}_1(\lambda) \\ \Pr\left[\mathsf{Expt}'_j(\mathcal{A}) = 1 \wedge \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{L}}} \wedge \mathsf{E}^{(j)}_{P,j_{\mathrm{L}}} \wedge \mathsf{E}^{(j)}_{\mathsf{ValidCom},j_{\mathrm{R}}} \wedge \mathsf{E}^{(j)}_{P,j_{\mathrm{R}}}\right] \leq \mathsf{negl}_2(\lambda).\end{aligned}$$

where $\varepsilon_j(\lambda) = \max(\varepsilon_{j_{\mathrm{L}}}(\lambda), \varepsilon_{j_{\mathrm{R}}}(\lambda))$. The claim follows. □

Completing the proof of Lemma 5.10. To complete the proof of the inductive step (for Lemma 5.10), we first appeal to Claim 5.15 to conclude that there exists negligible function $\mathsf{negl}_1(\cdot)$ such that

$$\Pr[\mathsf{Expt}'_j(\mathcal{A}) = 1] \leq 2^{h+1} \cdot \varepsilon_j(\lambda) + \mathsf{negl}_1(\lambda),$$

where $\varepsilon_j(\lambda) = \max(\varepsilon_{j_{\mathrm{L}}}(\lambda), \varepsilon_{j_{\mathrm{R}}}(\lambda))$. From the inductive hypothesis, $\varepsilon_{j_{\mathrm{L}}}(\lambda)$ and $\varepsilon_{j_{\mathrm{R}}}(\lambda)$ are both negligible functions. By definition of Expt'_j, we have that

$$\Pr[\mathsf{Expt}'_j(\mathcal{A}) = 1] = \Pr\left[\mathsf{E}^{(j)}_{\mathsf{Verify}} \wedge \mathsf{E}^{(j)}_{\mathsf{SAT},j} \wedge \neg\mathsf{E}^{(j)}_{P,j}\right].$$

By Claim 5.11, there exists a negligible function $\mathsf{negl}_2(\cdot)$ such that

$$\left|\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] - \Pr[\mathsf{Expt}'_j(\mathcal{A}) = 1]\right| \leq \mathsf{negl}_2(\lambda).$$

We conclude that

$$\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] \leq 2^{h+1} \cdot \varepsilon_j(\lambda) + \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda).$$

Setting $\varepsilon'_j(\lambda) = \max\left(\varepsilon_j(\lambda), (\mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda))/2^{h+1}\right)$, we have that $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] \leq 2^{h+1} \cdot \varepsilon'_j(\lambda)$, where $\varepsilon'_j(\lambda)$ is a negligible function. Lemma 5.10 now follows by induction on the height h. □

Completing the proof of Theorem 5.9. We now use Lemma 5.10 to complete the proof of Theorem 5.9. Suppose the conditions of Theorem 5.9 hold. Noting that the index $2n-1$ has height $h = \log n$ in a complete binary tree with n leaves, we appeal to Lemma 5.10 and conclude that there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{Expt}_{2n-1}(\mathcal{A}) = 1] \leq n \cdot \mathsf{negl}(\lambda). \tag{5.10}$$

To complete the proof, we define a sequence of hybrid experiments:

- Hyb_0: This is the experiment $\mathsf{Expt}_{2n-1}[P, \mathsf{DeriveChal}]$ with adversary $\mathcal{A}$.
- Hyb_1: Same as Hyb_0, except the output of the experiment is 1 if the following properties hold:
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$;

- $C_{\mathsf{idx}}\big(2n-1, (\hat{v}^{(0)}_{2n-1}, \hat{v}^{(1)}_{2n-1}, \sigma^{(0)}_{2n-1}, \sigma^{(1)}_{2n-1}, \tilde{w}_{2n-1})\big) = 1$; and
- $P\big(i^*, \mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\big) = 1$.

- Hyb_2: Same as Hyb_1, except the output of the experiment is 1 if the following properties hold:
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$; and
  - $P\big(i^*, \mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\big) = 1$.

  In particular, the challenger no longer checks the value of C_{idx}. Note that in this experiment, the challenger's behavior no longer depends on the BARG trapdoor $\mathsf{td}_{\mathsf{BARG}}$.

- Hyb_3: Same as Hyb_2, except when sampling the BARG parameters at the beginning of the experiment, the challenger now samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3)$. This corresponds to the experiment $\mathsf{Expt}[P, \mathsf{DeriveChal}]$ with adversary $\mathcal{A}$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A}) = 1$ to denote the output of Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of adjacent experiments.

Claim 5.16. *It holds that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.

Proof. These experiments are identical. Specifically, by definition of C_{idx} (and specifically, the relation in Fig. 2), if $C_{\mathsf{idx}}\big(2n-1, (\hat{v}^{(0)}_{2n-1}, \hat{v}^{(1)}_{2n-1}, \sigma^{(0)}_{2n-1}, \sigma^{(1)}_{2n-1}, \tilde{w}_{2n-1})\big) = 1$, then $\hat{v}^{(b)}_{2n-1} = \mathsf{ct}^{(b)}_{\mathsf{root}}$ for $b \in \{0,1\}$. This means that

$$P\big(i^*, \hat{v}^{(0)}_{2n-1}, \hat{v}^{(1)}_{2n-1}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\big) = P\big(i^*, \mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\big) = 1.$$

Thus, the output of $\mathsf{Hyb}_0(\mathcal{A})$ is identical to that of $\mathsf{Hyb}_1(\mathcal{A})$. □

Claim 5.17. *If* Π_{BARG} *is somewhere extractable, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\big|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. Suppose $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \geq \varepsilon(\lambda)$, for some non-negligible ε. Since the only difference between Hyb_1 and Hyb_2 is the conditions the challenger checks at the very end of the experiment, this means that with probability at least ε, the adversary in Hyb_1 will output a digest dig and a proof π such that the following conditions hold:

- $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$.
- $P\big(i^*, \mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\big) = 1$.
- $C_{\mathsf{idx}}\big(2n-1, (\hat{v}^{(0)}_{2n-1}, \hat{v}^{(1)}_{2n-1}, \sigma^{(0)}_{2n-1}, \sigma^{(1)}_{2n-1}, \tilde{w}_{2n-1})\big) = 0$.

In all other settings, the output of the two experiments are identical. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks somewhere extractability of Π_{BARG} (similar to the proof of Claim 5.12):

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.
2. Let $j = 2n-1$ and $j_{\mathsf{L}}, j_{\mathsf{R}}$ be the indices of the input wires that determine the value of the output wire j. Algorithm $\mathcal{B}$ outputs $1^{2n-1}, 1^s, 1^3$, the challenge set $J = \{j, j_{\mathsf{R}}, j_{\mathsf{L}}\}$, and the challenge index $j = 2n-1$ to the challenger. Here, s is the bound on the size of the circuit in Fig. 2. The challenger responds with $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}})$.
3. Algorithm $\mathcal{B}$ computes $(S_0, S_1, \mathsf{idx}) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.

- For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
- For each $b \in \{0,1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$

4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b\in\{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.

6. Let C_{idx} be the circuit as defined in Definition 5.7. Algorithm $\mathcal{B}$ outputs the circuit C_{idx}, the instance number $2n-1$, and the proof π.

By definition, the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\})$. This means algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1. Thus, with probability at least ε, the digest dig and proof π output by $\mathcal{A}$ satisfies

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1 \quad \text{and} \quad C_{\mathsf{idx}}\left(2n-1, \left(\hat{v}_{2n-1}^{(0)}, \hat{v}_{2n-1}^{(1)}, \sigma_{2n-1}^{(0)}, \sigma_{2n-1}^{(1)}, \tilde{w}_{2n-1}\right)\right) = 0,$$

where $\left(\hat{v}_{2n-1}^{(0)}, \hat{v}_{2n-1}^{(1)}, \sigma_{2n-1}^{(0)}, \sigma_{2n-1}^{(1)}, \tilde{w}_{2n-1}\right) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, 2n-1)$. This means algorithm $\mathcal{B}$ successfully breaks somewhere extractability of Π_{BARG} and the claim holds. □

Claim 5.18. *If Π_{BARG} satisfies set hiding then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks set hiding of Π_{BARG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.

2. Let $j = 2n-1$ and $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the indices of the input wires that determine the value of the output wire j. Algorithm $\mathcal{B}$ outputs 1^{2n-1}, 1^s, 1^3 and the challenge set $J = \{j, j_{\mathrm{L}}, j_{\mathrm{R}}\}$ to the challenger. Here, s is the bound on the size of the circuit in Fig. 2. The challenger responds with $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}})$.

3. Algorithm $\mathcal{B}$ computes $(S_0, S_1) \leftarrow \mathsf{DeriveChal}(S, \hat{\imath})$. It then samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S_b$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0,1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$

4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b\in\{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.

6. Let $C_{i^*,x_{i^*}}$ be the circuit as defined in Definition 5.7. Algorithm $\mathcal{B}$ outputs 1 if

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1 \quad \text{and} \quad P\left(i^*, \mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\right) = 1$$

   Otherwise, algorithm $\mathcal{B}$ outputs 0.

We now consider the two possibilities:

- Suppose the challenger responds according to the specification of $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0)$. In this case, the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3)$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_3 for $\mathcal{A}$. Moreover, algorithm $\mathcal{B}$ computes the outputs according to the same specification of Hyb_3, so we conclude that algorithm $\mathcal{B}$ outputs 1 with $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.
- Suppose the challenger responds according to the specification of $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 1)$. In this case, the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j, j_{\mathsf{L}}, j_{\mathsf{R}}\})$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_2 for $\mathcal{A}$, and correspondingly, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that the distinguishing advantage of $\mathcal{B}$ is exactly ε, which concludes the proof. □

Combining Claims 5.16 to 5.18, we conclude that $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$. By construction, $\mathsf{Hyb}_0(\mathcal{A}) \equiv \mathsf{Expt}_{2n-1}(\mathcal{A})$ and $\mathsf{Hyb}_3(\mathcal{A}) \equiv \mathsf{Expt}(\mathcal{A})$. From Eq. (5.10), we have that $\Pr[\mathsf{Expt}_{2n-1}(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$ and Theorem 5.9 follows. □

5.1.2 Set Hiding

In this section, we show that Construction 5.2 satisfies set hiding. This follows immediately from CPA-security of the underlying encryption scheme. Recall that in Construction 5.2, the only difference between a hash key that binds to the empty set $\varnothing$ versus the set S is that some of the ciphertexts in the hash key switch from encryptions of 0 (when binding to the empty set) to an encryption of 1 (when binding to the set S). We formalize this below:

Theorem 5.19 (Set Hiding). *If Π_{HE} is CPA-secure, then Construction 5.2 satisfies set hiding.*

Proof. Let $\mathcal{A}$ be an adversary for the set hiding game. We define a hybrid experiment Hyb_β for each $\beta \in \{0, 1, 2\}$ as follows:

1. On input 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^n and set $S \subseteq [n]$.
2. The challenger now samples the following quantities:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$. and $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3)$.
   - For each $b \in \{0, 1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $i \in [n]$ and $b \in \{0, 1\}$, if $i \in S$ and $b < \beta$, the challenger samples $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$. Otherwise, if $i \notin S$ or $b \geq \beta$, it samples $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0, 1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$
3. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Let $\mathsf{Hyb}_\beta(\mathcal{A})$ be the output of Hyb_β with adversary $\mathcal{A}$. By construction $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 1) \equiv \mathsf{Hyb}_2(\mathcal{A})$. We now argue that each adjacent pair of hybrid distributions are computationally indistinguishable.

Claim 5.20. *If* Π_{HE} *is CPA-secure, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose that $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| = 1] \geq \varepsilon(\lambda)$ for some non-negligible ε. We construct a CPA-security adversary $\mathcal{B}$ against Π_{HE} as follows:

1. On input 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ to obtain the input length 1^n and a set $S \subseteq [n]$. Denote $S = \{i_1, \ldots, i_t\} \subseteq [n]$, where $t = |S|$. Algorithm $\mathcal{B}$ sends 1^n to the challenger as the input range. The challenger replies with a public key pk_0.
2. Algorithm $\mathcal{B}$ samples $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$. It also samples $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$ and $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n}, 1^s, 1^3)$. Finally, for $b \in \{0,1\}$, it computes $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$ for $b \in \{0,1\}$.
3. Then, for each $i \in [n]$, algorithm $\mathcal{B}$ does the following:
   - If $i \in S$, then it makes an encryption query on the pair $(0,1)$ and receives the ciphertext ct_i^*. Algorithm $\mathcal{B}$ sets $\mathsf{ct}_i^{(0)} = \mathsf{ct}_i^*$. If $i \notin S$, it sets $\mathsf{ct}_i^{(0)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_0, 0)$.
   - It computes $\mathsf{ct}_i^{(1)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_1, 0)$.
4. Finally, for $b \in \{0,1\}$, compute $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$.
5. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2) and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
6. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

Observe that if the ct_i^* are encryptions of 0 then $\mathcal{B}$ perfectly simulates Hyb_0. If ct_i^* are encryptions of 1, then $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$. We conclude that the advantage of $\mathcal{B}$ is ε. □

Claim 5.21. *If* Π_{HE} *is CPA-secure, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Follows by an analogous argument as the proof of Claim 5.20. The only difference is the reduction algorithm $\mathcal{B}$ sets pk_1 and the ciphertexts $\mathsf{ct}_i^{(1)}$ for $i \in S$ to be the public key and challenge ciphertexts it receives for the CPA challenger, respectively. □

Theorem 5.19 now follows by combining Claims 5.20 and 5.21. □

5.1.3 Zero Fixing

In this section, we show that Construction 5.2 satisfies zero-fixing security. In the zero-fixing game, the hash key in Construction 5.2 is chosen to bind to a set S. This means that the ciphertext in the hash key associated with the set S are replaced by encryptions of 1. Suppose the adversary constructs an opening to 1 for an index $i \in S$. This means the adversary must have "used" the ciphertext associated with index i (which encrypts 1) when constructing the digest; formally, this will be enforced by the BARG. Since one of ciphertexts in the ciphertext tree is an encryption of 1, we can appeal to the predicate propagation property (Theorem 5.9) to argue that the root of the tree must encrypt a value that is non-zero. In this case, the extraction algorithm would declare the digest $\mathsf{NotMatching}$ and zero-fixing holds. We now give the formal argument:

Theorem 5.22 (Zero Fixing)**.** *Suppose Π_{HE} satisfies perfect correctness and evaluation correctness, Π_{Com} is computationally binding, and Π_{BARG} satisfies set hiding with extraction, set hiding, and is somewhere extractable. Then Construction 5.2 satisfies* selective *zero-fixing.*

Proof. We will leverage Theorem 5.9. We start by defining a tree-based additive invariant P as follows. Let n be a power-of-two and take any index $i^* \in [n]$. We now inductively associate an interval $I_j^{(i^*)}$ with each node $j \in [2n-1]$ of a complete binary tree with n leaves (indexed according to Definition 5.1) as follows:

- For $j \in [n]$, if $j = i^*$ then let $I_j^{(i^*)} = [1,1]$. Otherwise, let $I_j^{(i^*)} = [0,1]$.
- For an index $j \in [n+1, 2n-1]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the indices of the children of j according to Definition 5.1. If $I_{j_{\mathrm{L}}}^{(i^*)} = [a_{\mathrm{L}}, b_{\mathrm{L}}]$ and $I_{j_{\mathrm{R}}}^{(i^*)} = [a_{\mathrm{R}}, b_{\mathrm{R}}]$, then define $I_j^{(i^*)} = [a_{\mathrm{L}} + a_{\mathrm{R}}, b_{\mathrm{L}} + b_{\mathrm{R}}] = I_{j_{\mathrm{L}}}^{(i^*)} + I_{j_{\mathrm{R}}}^{(i^*)}$, where we define interval addition to be component-wide addition: $[a_{\mathrm{L}}, b_{\mathrm{L}}] + [a_{\mathrm{R}}, b_{\mathrm{R}}] = [a_{\mathrm{L}} + a_{\mathrm{R}}, b_{\mathrm{L}} + b_{\mathrm{R}}]$.

We now define the interval-validity predicate P_{ValidInt} as follows: $P_{\mathsf{ValidInt}} \colon \{0,1\}^* \to \{0,1\}$ as follows:

$$P_{\mathsf{ValidInt}}\big(i^*, \mathsf{ct}^{(0)}, \mathsf{ct}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) := \begin{cases} 1 & \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}^{(0)}\big) \in I_j^{(i^*)} \\ 0 & \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}^{(0)}\big) \notin I_j^{(i^*)}. \end{cases} \tag{5.11}$$

We now show that P_{ValidInt} is a tree-based additive invariant. We start by characterizing the intervals $I_j^{(i^*)}$ for all $j \in [2n-1]$.

Claim 5.23. *For all $i^* \in [n]$ and any node $j \in [2n-1]$ of height h in the binary tree, if i^* is in the subtree rooted at j then $I_j^{(i^*)} = [1, 2^h]$. Otherwise, $I_j^{(i^*)} = [0, 2^h]$.*

Proof. This follows by induction starting from the leaves. For every leaf node $j \in [n]$, the associated interval $I_j^{(i^*)}$ is $[0,1]$ if $j \neq i^*$ and $[1,1]$ if $j = i^*$. Thus, the claim holds for all of the leaf nodes. For the induction step, suppose j is a node of height $h+1$. Let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the indices of the children of j. If i^* is in the subtree of j, then it is either in the subtree of j_{L} or j_{R} but not both. By the induction hypothesis, $I_{j_{\mathrm{L}}}^{(i^*)} + I_{j_{\mathrm{R}}}^{(i^*)} = [0, 2^h] + [1, 2^h] = [1, 2^{h+1}]$. If i^* is not in the subtree of j, then by the induction hypothesis, $I_{j_{\mathrm{L}}}^{(i^*)} + I_{j_{\mathrm{R}}}^{(i^*)} = [0, 2^h] + [0, 2^h] = [0, 2^{h+1}]$. □

Claim 5.24. *If Π_{HE} satisfies evaluation correctness, then P_{ValidInt} is a tree-based additive invariant.*

Proof. Take any $i^* \in [n]$, $(\mathsf{sk}_0, \mathsf{pk}_0)$, $(\mathsf{sk}_1, \mathsf{pk}_1)$ in the support of $\mathsf{HE.Gen}(1^\lambda, 1^n)$, any triple of indices $j, j_{\mathrm{L}}, j_{\mathrm{R}} \in [2n-1]$ where $j_{\mathrm{L}}, j_{\mathrm{R}}$ are the indices of the children of j, and any set of ciphertexts $\big(\mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}\big)$, $\big(\mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}\big)$. Suppose j has height $h \le \log n$. Let $\mathsf{ct}_{\mathsf{sum}}^{(0)} = \mathsf{HE.Add}\big(\mathsf{pk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(0)}\big)$ and $\mathsf{ct}_{\mathsf{sum}}^{(1)} = \mathsf{HE.Add}\big(\mathsf{pk}_1, \mathsf{ct}_{\mathrm{L}}^{(1)}, \mathsf{ct}_{\mathrm{R}}^{(1)}\big)$. Suppose

$$P_{\mathsf{ValidInt}}\big(i^*, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{L}}\big) = 1 \quad \text{and} \quad P_{\mathsf{ValidInt}}\big(i^*, \mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{R}}\big) = 1.$$

This means $\mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}\big) \in I_{j_{\mathrm{L}}}^{(i^*)}$ and $\mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{R}}^{(0)}\big) \in I_{j_{\mathrm{R}}}^{(i^*)}$. By Claim 5.23, this means

$$\mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}\big), \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{R}}^{(0)}\big) \in [0, 2^{h-1}].$$

Since $2^{h-1} \le n/2$, we can appeal to evaluation correctness of Π_{H} and conclude that

$$\mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{HE.Add}\big(\mathsf{pk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(0)}\big)\big) = \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}\big) + \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{R}}^{(0)}\big) \in I_{j_{\mathrm{L}}}^{(i^*)} + I_{j_{\mathrm{R}}}^{(i^*)} = I_j^{(i^*)}.$$

We conclude that $P_{\mathsf{ValidInt}}\big(i^*, \mathsf{ct}_{\mathsf{sum}}^{(0)}, \mathsf{ct}_{\mathsf{sum}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 1$. □

Proof of Theorem 5.22. Returning now to the proof of Theorem 5.22, let $\mathcal{A}$ be any efficient adversary for the zero-fixing security game. We start by defining a mapping DeriveChal as

$$\mathsf{DeriveChal}(S, i) \coloneqq (S, i) \mapsto (S, S, (i, 1)).$$

Let $\mathsf{Expt} \coloneqq \mathsf{Expt}[P_{\mathsf{ValidInt}}, \mathsf{DeriveChal}]$ be the predicate propagation experiment from Definition 5.7. First, we claim that

$$\Pr[\mathsf{ExptZF}_{\mathcal{A}}(\lambda) = 1] \le \Pr[\mathsf{Expt}(\mathcal{A}) = 1]. \tag{5.12}$$

By construction, the adversary's view in ExptZF and Expt is identical. It suffices to consider the outputs of the two experiments. Suppose $\mathsf{ExptZF}_{\mathcal{A}}(\lambda) = 1$. This means the adversary $\mathcal{A}$ outputs $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and an opening π such that $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = \mathsf{Matching}$ and $\mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, i, 1, \pi) = 1$. This means the following:

- By construction, $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ outputs Matching if $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{root}}^{(0)}) = 0$. From Claim 5.24, we have that $I_{2n-1}^{(i^*)} = [1, n]$. Hence, in this case, $\mathsf{HE.Dec}\left(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{root}}^{(0)}\right) \notin I_{2n-1}^{(i^*)}$, so $P_{\mathsf{ValidInt}}\left(i^*, \mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\right) = 0$ in the predicate propagation experiment $\mathsf{Expt}(\mathcal{A})$.

- By definition, VerOpen outputs 1 if $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,1}, 2n-1, \pi) = 1$, where $i^* \in S$ is the index chosen by the adversary at the beginning of the (selective) zero-fixing game. By construction of DeriveChal, we have that $\mathsf{idx} = (i^*, 1)$ in the execution of $\mathsf{Expt}(\mathcal{A})$. This means that $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$.

Since $P_{\mathsf{ValidInt}}\left(i^*, \mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\right) = 0$ and $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$, the predicate propagation experiment $\mathsf{Expt}(\mathcal{A})$ also outputs 1. We now show using Theorem 5.9 that $\Pr[\mathsf{Expt}(\mathcal{A}) = 1] \le \mathsf{negl}(\lambda)$. To leverage Theorem 5.9, we analyze the predicate propagation hybrid experiment $\mathsf{Expt}_j \coloneqq \mathsf{Expt}_j[P_{\mathsf{ValidInt}}, \mathsf{DeriveChal}]$ from Definition 5.8.

Claim 5.25. *If Π_{HE} is perfectly correct and Π_{Com} satisfies computational binding, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $j \in [n]$, it holds that $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists some $j \in [n]$ where $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] \ge \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks computational binding of Π_{Com}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.

2. Algorithm $\mathcal{B}$ outputs the block length $1^{\ell_{\mathsf{ct}}(\lambda, n)}$ and the vector length $2n-1$ to the challenger. The challenger responds with $\mathsf{crs}_{\mathsf{Com}}$.

3. Algorithm $\mathcal{B}$ computes $(S, S, (i^*, 1)) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:

   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j\})$.
   - For each $b \in \{0, 1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0, 1\}$, if $i \in S$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0, 1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \dots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \dots, \mathsf{ct}_n^{(b)})\right)$

4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \dots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \dots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.

6. Algorithm $\mathcal{B}$ extracts $\big(\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j)$ and parses $\tilde{w}_j = \big(\hat{\mathsf{ct}}^{(0)}, \hat{\mathsf{ct}}^{(1)}, \sigma_{\mathsf{hk}}^{(0)}, \sigma_{\mathsf{hk}}^{(1)}\big)$.
7. Output the commitment $\mathsf{com}_{\mathsf{hk}}^{(0)}$, the index j, and the value-opening pairs $\big(\mathsf{ct}_j^{(0)}, \sigma_{\mathsf{hk},j}^{(0)}\big)$ and $\big(\hat{\mathsf{ct}}^{(0)}, \sigma_{\mathsf{hk}}^{(0)}\big)$.

By construction, the challenger samples $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com}.\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$, which matches the specification in Expt_j. This, algorithm $\mathcal{B}$ perfectly simulates an execution of Expt_j for $\mathcal{A}$. By assumption, with probability ε, algorithm $\mathcal{A}$ outputs dig and π such that the experiment outputs 1. This means the following conditions hold:

$$C_{i^*,1}\big(j, \big(\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j\big)\big) = 1 \quad \text{and} \quad P_{\mathsf{ValidInt}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0.$$

By definition of $C_{i^*,1}$ and using the fact that $j \in [n]$, this means

$$\mathsf{Com}.\mathsf{Verify}\big(\mathsf{crs}_{\mathsf{com}}, \mathsf{com}_{\mathsf{hk}}^{(0)}, j, \hat{\mathsf{ct}}^{(0)}, \sigma_{\mathsf{hk}}^{(0)}\big) = 1 \quad \text{and} \quad \hat{v}_j^{(0)} \in \big\{\mathsf{ct}_{\mathsf{zero}}^{(0)}\,,\ \hat{\mathsf{ct}}^{(0)}\big\}.$$

If $j = i^*$, we additionally have that $\hat{v}_j^{(0)} = \hat{\mathsf{ct}}^{(0)}$. Next, by correctness of Π_{Com},

$$\mathsf{Com}.\mathsf{Verify}\big(\mathsf{crs}_{\mathsf{com}}, \mathsf{com}_{\mathsf{hk}}^{(0)}, j, \mathsf{ct}_j^{(0)}, \sigma_{\mathsf{hk},j}^{(0)}\big) = 1.$$

It suffices to argue that $\mathsf{ct}_j^{(0)} \neq \hat{\mathsf{ct}}^{(0)}$. We consider two cases:

- Suppose $j = i^*$. Recall that in this case, $\hat{v}_j^{(0)} = \hat{\mathsf{ct}}^{(0)}$. By Claim 5.24, we have that $I_j^{(i^*)} = [1,1]$. Since $P_{\mathsf{ValidInt}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0$, this means that $\mathsf{HE}.\mathsf{Dec}(\mathsf{sk}_0, \hat{v}_j^{(0)}) \notin I_j^{(i^*)}$. Thus, $\mathsf{HE}.\mathsf{Dec}(\mathsf{sk}_0, \hat{v}_j^{(0)}) \neq 1$. Next, algorithm $\mathcal{B}$ constructs $\mathsf{ct}_j^{(0)}$ to be an encryption of 1 (since $j = i^* \in S$). By perfect correctness of Π_{HE}, this means that $\hat{\mathsf{ct}}^{(0)} = \hat{v}_j^{(0)} \neq \mathsf{ct}_j^{(0)}$.
- Suppose $j \neq i^*$. By Claim 5.24, we have that $I_j^{(i^*)} = [0,1]$. By the same reasoning as in the previous case, this means that $\mathsf{HE}.\mathsf{Dec}(\mathsf{sk}_0, \hat{v}_j^{(0)}) \notin I_j^{(i^*)}$. In other words, $\mathsf{HE}.\mathsf{Dec}(\mathsf{sk}_0, \hat{v}_j^{(0)}) \notin \{0,1\}$. By construction, $\mathsf{ct}_{\mathsf{zero}}^{(0)}$ is an encryption of 0, so by perfect correctness of Π_{HE}, we have that $\hat{v}_j^{(0)} \neq \mathsf{ct}_{\mathsf{zero}}^{(0)}$. Hence, it must be the case that $\hat{v}_j^{(0)} = \hat{\mathsf{ct}}^{(0)}$. Next, the ciphertext $\mathsf{ct}_j^{(0)}$ is an encryption of either 0 or 1, so we conclude that $\hat{\mathsf{ct}}^{(0)} = \hat{v}_j^{(0)} \neq \mathsf{ct}_j^{(0)}$.

In both cases, we conclude that $\hat{\mathsf{ct}}^{(0)} \neq \mathsf{ct}_j^{(0)}$. In this case, algorithm $\mathcal{B}$ successfully opens $\mathsf{com}_{\mathsf{hk}}^{(0)}$ to two distinct values $\mathsf{ct}_j^{(0)} \neq \hat{\mathsf{ct}}^{(0)}$. Thus algorithm $\mathcal{B}$ breaks binding with the same advantage ε. □

Since for all $j \in [n]$, it holds that $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda)$, we can invoke Theorem 5.9 to conclude that $\Pr[\mathsf{Expt}(\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda)$. Zero-fixing security now follows via Eq. (5.12). □

5.1.4 Extractor Validity

In this section, we show that Construction 5.2 satisfies extractor validity. In the extractor validity game, the hash key is sampled to be zero-fixing on the empty set $\varnothing$, and the goal of the adversary is to produce a valid, but *non-matching* digest. In this setting, the ciphertexts in the hash key are all encryptions of 0. In order to break the extractor validity property, the adversary needs to produce a root ciphertext that encrypts a non-zero value, and yet, still argue that the root ciphertext was derived by summing a collection of ciphertexts that each encryption 0. The latter is ensured by security of the BARG, and specifically the predicate propagation theorem (Theorem 5.9). We give the formal theorem statement and proof below:

Theorem 5.26. *If Π_{HE} satisfies perfect correctness and evaluation correctness, Π_{Com} is computationally binding, and Π_{BARG} satisfies set hiding, set hiding with extraction, and somewhere extractability, then Construction 5.2 satisfies extractor validity.*

Proof. Similar to the proof of Theorem 5.22, we leverage Theorem 5.9. We start by defining a tree-based additive invariant P as follows. Define the "matching" predicate $P_{\mathsf{Matching}} \colon \{0,1\}^* \to \{0,1\}$ as follows:

$$P_{\mathsf{Matching}}(i^*, \mathsf{ct}^{(0)}, \mathsf{ct}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j) \coloneqq \begin{cases} 1 & \mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}^{(0)}) = 0 \\ 0 & \mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}^{(0)}) \neq 0. \end{cases}$$

We first show that P_{Matching} is a tree-based additive invariant.

Claim 5.27. *If Π_{HE} satisfies evaluation correctness, then P_{Matching} is a tree-based additive invariant.*

Proof. Take any $i^* \in [n]$, any $(\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1)$ in the support of $\mathsf{HE.Gen}(1^\lambda, 1^n)$, any triple of indices $j, j_{\mathrm{L}}, j_{\mathrm{R}} \in [2n-1]$ where $j_{\mathrm{L}}, j_{\mathrm{R}}$ are the indices of the children of j, and any set of ciphertexts $\left(\mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}\right)$, $\left(\mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}\right)$. Suppose

$$P_{\mathsf{Matching}}(i^*, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{L}}) = P_{\mathsf{Matching}}(i^*, \mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{R}}) = 1.$$

This means $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}) = 0$ and $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{R}}^{(0)}) = 0$. For $b \in \{0,1\}$, let $\mathsf{ct}_{\mathsf{sum}}^{(b)} = \mathsf{HE.Add}\left(\mathsf{pk}_b, \mathsf{ct}_{\mathrm{L}}^{(b)}, \mathsf{ct}_{\mathrm{R}}^{(b)}\right)$. By evaluation correctness of Π_{HE}, we have $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{sum}}^{(0)}) = 0$ and so $P_{\mathsf{Matching}}\left(i^*, \mathsf{ct}_{\mathsf{sum}}^{(0)}, \mathsf{ct}_{\mathsf{sum}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\right) = 1$. □

Let $\mathcal{A}$ be an efficient adversary for the extractor-validity game. Define the mapping DeriveChal as

$$\mathsf{DeriveChal}(S, i) \coloneqq (S, i) \mapsto (\varnothing, \varnothing, \bot).$$

Let $\mathsf{Expt} \coloneqq \mathsf{Expt}[P_{\mathsf{Matching}}, \mathsf{DeriveChal}]$ be the predicate propagation experiment from Definition 5.7. First, we claim that we can use $\mathcal{A}$ to construct an adversary $\mathcal{A}'$ such that

$$\Pr[\mathsf{ExptEV}_{\mathcal{A}}(\lambda) = 1] \le \Pr[\mathsf{Expt}(\mathcal{A}') = 1]. \tag{5.13}$$

Algorithm $\mathcal{A}'$ works as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{A}'$ runs $\mathcal{A}$ on the same security parameter. Algorithm $\mathcal{A}$ outputs an input length 1^n. Algorithm $\mathcal{A}'$ outputs the input length 1^n, the set $S = \varnothing$, and the index $i^* = \bot$.
2. The challenger replies with $(\mathsf{hk}, \mathsf{vk})$ which $\mathcal{A}'$ forwards to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$. Algorithm $\mathcal{A}'$ outputs the same digest dig and $\pi = \pi_{\mathsf{dig}}$.

We now show that Eq. (5.13) holds. By construction, the pair $(\mathsf{hk}, \mathsf{vk})$ sampled by the challenger are distributed according to the real setup algorithm. Thus, algorithm $\mathcal{A}$ perfectly simulates an execution of $\mathsf{ExptEV}_{\mathcal{A}}$ for adversary $\mathcal{A}$. Thus, with probability $\Pr[\mathsf{ExptEV}_{\mathcal{A}}(\lambda) = 1]$, algorithm $\mathcal{A}$ outputs a digest dig where $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = \mathsf{NotMatching}$ and $\mathsf{ValidateDigest}(\mathsf{hk}, \mathsf{dig}) = 1$. This means the following:

- By construction, $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ outputs NotMatching if $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{root}}^{(0)}) \neq 0$. By construction of P_{Matching}, this means $P_{\mathsf{Matching}}\left(i^*, \mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\right) = 0$.
- Next, ValidateDigest outputs 1 if $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_\bot, 2n-1, \pi_{\mathsf{dig}}) = 1$. By construction of DeriveChal, we have that $\mathsf{idx} = \bot$ in the execution of $\mathsf{Expt}(\mathcal{A})$, so this means that $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi_{\mathsf{dig}}) = 1$.

Since $P_{\mathsf{Matching}}\left(i^*, \mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\right) = 0$ and $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi_{\mathsf{dig}}) = 1$, the predicate propagation experiment $\mathsf{Expt}(\mathcal{A}')$ also outputs 1. Hence, we conclude that $\Pr[\mathsf{Expt}(\mathcal{A}') = 1] \ge \Pr[\mathsf{ExptEV}_{\mathcal{A}}(\lambda) = 1]$. To complete the proof, we now show using Theorem 5.9 that $\Pr[\mathsf{Expt}(\mathcal{A}') = 1] \le \mathsf{negl}(\lambda)$. To leverage Theorem 5.9, we analyze the predicate propagation hybrid experiment $\mathsf{Expt}_j \coloneqq \mathsf{Expt}_j[P_{\mathsf{Matching}}, \mathsf{DeriveChal}]$ from Definition 5.8.

Claim 5.28. *If Π_{HE} is perfectly correct and Π_{Com} satisfies computational binding, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $j \in [n]$, it holds that $\Pr[\mathsf{Expt}_j(\mathcal{A}') = 1] = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists some $j \in [n]$ where $\Pr[\mathsf{Expt}_j(\mathcal{A}') = 1] \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}'$ to construct an adversary $\mathcal{B}$ that breaks computational binding of Π_{Com}.

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}'$ to obtain the input length 1^n, the set $S = \varnothing$, and the index $i^* = \perp$.
2. Algorithm $\mathcal{B}$ outputs the block length $1^{\ell_{\mathsf{ct}}(\lambda,n)}$ and the vector length $2n-1$ to the challenger. The challenger responds with $\mathsf{crs}_{\mathsf{Com}}$.
3. Algorithm $\mathcal{B}$ computes $(\varnothing, \varnothing, \perp) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j\})$.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $b \in \{0,1\}$, let $\big(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\big) \leftarrow \mathsf{Com.Commit}\big(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\big)$.
4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \Big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \big\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\big\}_{b \in \{0,1\}}\Big)$$
$$\mathsf{vk} = \big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\big)$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}'$.
5. Algorithm $\mathcal{A}'$ outputs a digest $\mathsf{dig} = \big(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$ and a proof $\pi = \pi_{\mathsf{dig}}$.
6. Algorithm $\mathcal{B}$ extracts $\big(\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j)$ and parses $\tilde{w}_j = \big(\hat{\mathsf{ct}}^{(0)}, \hat{\mathsf{ct}}^{(1)}, \sigma_{\mathsf{hk}}^{(0)}, \sigma_{\mathsf{hk}}^{(1)}\big)$.
7. Output the commitment $\mathsf{com}_{\mathsf{hk}}^{(0)}$, the index j, and the value-opening pairs $\big(\mathsf{ct}_j^{(0)}, \sigma_{\mathsf{hk},j}^{(0)}\big)$ and $\big(\hat{\mathsf{ct}}^{(0)}, \sigma_{\mathsf{hk}}^{(0)}\big)$.

By construction, the challenger samples $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$, which matches the specification in Expt_j. This, algorithm $\mathcal{B}$ perfectly simulates an execution of Expt_j for $\mathcal{A}'$. By assumption, with probability ε, algorithm $\mathcal{A}'$ outputs dig and π such that the experiment outputs 1. This means the following conditions hold:

$$C_\perp\big(j, (\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j)\big) = 1 \quad \text{and} \quad P_{\mathsf{Matching}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0.$$

By definition of $C_\perp$ and using the fact that $j \in [n]$, this means

$$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{com}}, \mathsf{com}_{\mathsf{hk}}^{(0)}, j, \hat{\mathsf{ct}}^{(0)}, \sigma_{\mathsf{hk}}^{(0)}\big) = 1 \quad \text{and} \quad \hat{v}_j^{(0)} \in \big\{\mathsf{ct}_{\mathsf{zero}}^{(0)}\,,\ \hat{\mathsf{ct}}^{(0)}\big\}.$$

Next, by correctness of Π_{Com},

$$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{com}}, \mathsf{com}_{\mathsf{hk}}^{(0)}, j, \mathsf{ct}_j^{(0)}, \sigma_{\mathsf{hk},j}^{(0)}\big) = 1.$$

It suffices to argue that $\mathsf{ct}_j^{(0)} \neq \hat{\mathsf{ct}}^{(0)}$. Since $P_{\mathsf{Matching}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0$, this means that $\mathsf{HE.Dec}(\mathsf{sk}_0, \hat{v}_j^{(0)}) \neq 0$. Since $\mathsf{ct}_{\mathsf{zero}}^{(0)}$ is an encryption of 0, we can appeal to perfect correctness of Π_{HE} to conclude that $\hat{v}_j^{(0)} \neq \mathsf{ct}_{\mathsf{zero}}^{(0)}$. This means that $\hat{v}_j^{(0)} = \hat{\mathsf{ct}}^{(0)}$. Moreover, $\mathsf{ct}_j^{(0)}$ is also an encryption of 0, so again by perfect correctness of the encryption scheme, we can conclude that $\mathsf{ct}_j^{(0)} \neq \hat{v}_j^{(0)} = \hat{\mathsf{ct}}^{(0)}$. In this case, algorithm $\mathcal{B}$ successfully opens $\mathsf{com}_{\mathsf{hk}}^{(0)}$ to two distinct values $\mathsf{ct}_j^{(0)} \neq \hat{\mathsf{ct}}^{(0)}$. Thus algorithm $\mathcal{B}$ breaks binding with the same advantage ε. □

Since for all $j \in [n]$, it holds that $\Pr[\mathsf{Expt}_j(\mathcal{A}') = 1] \leq \mathsf{negl}(\lambda)$, we can invoke Theorem 5.9 to conclude that $\Pr[\mathsf{Expt}(\mathcal{A}') = 1] \leq \mathsf{negl}(\lambda)$. Extractor-validity security now follows via Eq. (5.13). □

5.1.5 Index Hiding with Extracted Guess

In this section, we show that Construction 5.2 satisfies the index hiding with extracted guess property. The challenge in this reduction is we need to switch from an encryption of 0 to an encryption of 1 (in the hash key) while retaining the ability to decide whether the digest is "Matching" or not (which in the real scheme, requires knowledge of the secret key for the underlying encryption scheme). As described in Section 1.2, we solve this by adopting a Naor-Yung proof strategy.

Theorem 5.29. *If Π_{HE} satisfies perfect correctness, evaluation correctness, and CPA-security, Π_{Com} is computationally binding and Π_{BARG} satisfies set hiding with extraction, set hiding, and is somewhere extractable, then Construction 5.2 satisfies index hiding with extracted guess.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the index hiding with extracted guess security game. We define a sequence of hybrid experiments:

- Hyb_0: This is $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 0)$. Specifically, the game proceeds as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$.
  2. The challenger now samples the following quantities as in Setup:
     - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
     - Sample $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$.
     - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3)$, where s is a bound on the size of the circuit computing the index relation from Fig. 2.
     - For each $b \in \{0,1\}$, sample $\mathsf{ct}^{(b)}_{\mathsf{zero}} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$. Then, for each $i \in [n]$ and $b \in \{0,1\}$, if $i \in S \setminus \{i^*\}$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
     - For each $b \in \{0,1\}$, let $\left(\mathsf{com}^{(b)}_{\mathsf{hk}}, \sigma^{(b)}_{\mathsf{hk},1}, \ldots, \sigma^{(b)}_{\mathsf{hk},n}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$
  3. The challenger constructs hk and vk according to Eqs. (5.1) and (5.2):
     $$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}^{(b)}_{\mathsf{zero}}, \mathsf{ct}^{(b)}_1, \ldots, \mathsf{ct}^{(b)}_n, \sigma^{(b)}_{\mathsf{hk},1}, \ldots, \sigma^{(b)}_{\mathsf{hk},n}\right\}_{b \in \{0,1\}}\right)$$
     $$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}^{(0)}_{\mathsf{zero}}, \mathsf{ct}^{(1)}_{\mathsf{zero}}, \mathsf{com}^{(0)}_{\mathsf{hk}}, \mathsf{com}^{(1)}_{\mathsf{hk}}\right)$$
     The challenger gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
  4. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}^{(0)}_{\mathsf{root}}, \mathsf{ct}^{(1)}_{\mathsf{root}}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and an opening π.
  5. The output of the experiment is 1 if
     $$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,0}, 2n-1, \pi) = 1 \quad \text{and} \quad \mathsf{HE.Dec}\left(\mathsf{sk}_0, \mathsf{ct}^{(0)}_{\mathsf{root}}\right) = 0.$$
     Otherwise, the output is 0.
- Hyb_1: Same as Hyb_0, except the challenger samples $\mathsf{ct}^{(1)}_{i^*} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_1, 1)$.
- Hyb_2: Same as Hyb_1, except the output of the experiment is 1 if
  $$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,0}, 2n-1, \pi) = 1 \quad \text{and} \quad \mathsf{HE.Dec}\left(\mathsf{sk}_1, \mathsf{ct}^{(1)}_{\mathsf{root}}\right) = 0.$$
  Notably, the challenger's behavior in this experiment does *not* depend on sk_0.
- Hyb_3: Same as Hyb_2, except the output of the challenger samples $\mathsf{ct}^{(0)}_{i^*} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_0, 1)$.
- Hyb_4: Same as Hyb_3, except the output of the experiment is 1 if
  $$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,0}, 2n-1, \pi) = 1 \quad \text{and} \quad \mathsf{HE.Dec}\left(\mathsf{sk}_0, \mathsf{ct}^{(0)}_{\mathsf{root}}\right) = 0.$$
  This is experiment $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 1)$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of experiment of Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of hybrid experiments.

Claim 5.30. *If Π_{HE} is CPA-secure, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks CPA security of Π_{HE}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on the same input to obtain the input length 1^n, the set $S \subseteq [n]$, and an index $i^* \in S$.
2. Algorithm $\mathcal{B}$ sends 1^n as the input range. The challenger replies with a public key pk_1.
3. Algorithm $\mathcal{B}$ now samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j\})$.
   - For each $b \in \{0, 1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $i \in [n] \setminus \{i^*\}$ and $b \in \{0, 1\}$, if $i \in S$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$; otherwise, if $i \notin S$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
4. Algorithm $\mathcal{B}$ makes an encryption query on the pair $(0, 1)$. The challenger replies with a ciphertext $\mathsf{ct}_{i^*}^{(1)}$. Algorithm $\mathcal{B}$ also computes $\mathsf{ct}_{i^*}^{(0)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_0, 0)$.
5. Finally, for each $b \in \{0, 1\}$, let $\left(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right) \leftarrow \mathsf{Com.Commit}\left(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\right)$. Next, algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\mathsf{hk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \left\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\right\}_{b \in \{0,1\}}\right)$$
$$\mathsf{vk} = \left(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\right)$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
6. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \left(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\right)$ and a proof π.
7. Algorithm $\mathcal{B}$ outputs 1 if

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,0}, 2n-1, \pi) = 1 \quad \text{and} \quad \mathsf{HE.Dec}\left(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{root}}^{(0)}\right) = 0,$$

   where $C_{i^*,0}$ is the circuit computing the relation from Fig. 2 (which is a function of the components from hk and dig).

Observe that if $\mathsf{ct}_{i^*}^{(1)}$ is an encryption of 0 (under pk_1), then algorithm $\mathcal{B}$ perfectly simulates Hyb_0 for $\mathcal{A}$. Alternatively, if $\mathsf{ct}_{i^*}^{(1)}$ is an encryption of 1 (under pk_1), then algorithm $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$. We conclude that the advantage of $\mathcal{B}$ is ε. □

Claim 5.31. *If Π_{HE} is perfectly correct and satisfies evaluation correctness, Π_{Com} is computationally binding, Π_{BARG} satisfies set hiding with extraction, set hiding, and is somewhere extractable, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. By construction, the only difference between the execution of Hyb_1 and Hyb_2 is the output condition. Let E be the following event in an execution of Hyb_1 and Hyb_2:

$$\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,0}, 2n-1, \pi) = 1 \quad \text{and} \quad \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}^{(0)}_{\mathsf{root}}\big) \neq \mathsf{HE.Dec}\big(\mathsf{sk}_1, \mathsf{ct}^{(1)}_{\mathsf{root}}\big). \tag{5.14}$$

Observe that if E does not occur, then the output of Hyb_1 and Hyb_2 is identical. This means that

$$|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \leq \Pr[\mathsf{E}].$$

We now leverage Theorem 5.9 to argue that $\Pr[\mathsf{E}] = \mathsf{negl}(\lambda)$. To do so, we start by defining a suitable tree-based additive invariant. Similar to the proof of Theorem 5.22, we first associate a "validity interval" with each index $j \in [2n-1]$. For an index $i^* \in [n]$, we define the interval $I_j^{(i^*)}$ with each node j as follows:

- For $j \in [n]$, if $j = i^*$, let $I_j^{(i^*)} = [0,0]$. Otherwise, let $I_j^{(i^*)} = [0,1]$.
- For an index $j \in [n+1, 2n-1]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the indices of the children of j according to Definition 5.1. If $I_{j_{\mathrm{L}}}^{(i^*)} = [a_{\mathrm{L}}, b_{\mathrm{L}}]$ and $I_{j_{\mathrm{R}}}^{(i^*)} = [a_{\mathrm{R}}, b_{\mathrm{R}}]$, then define $I_j^{(i^*)} = [a_{\mathrm{L}} + a_{\mathrm{R}}, b_{\mathrm{L}} + b_{\mathrm{R}}] = I_{j_{\mathrm{L}}}^{(i^*)} + I_{j_{\mathrm{R}}}^{(i^*)}$, where we define interval addition to be component-wide addition: $[a_{\mathrm{L}}, b_{\mathrm{L}}] + [a_{\mathrm{R}}, b_{\mathrm{R}}] = [a_{\mathrm{L}} + a_{\mathrm{R}}, b_{\mathrm{L}} + b_{\mathrm{R}}]$.

By the same argument as in the proof of Claim 5.23, for all $j \in [2n-1]$ and all $i^* \in [n]$, we have that $I_j^{(i^*)} \subseteq [0, 2^h]$, where h is the height of node j. Now, we define the validity predicate $P_{\mathsf{Valid}} \colon \{0,1\}^* \to \{0,1\}$ as follows:

- On input $(i^*, \mathsf{ct}^{(0)}, \mathsf{ct}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j)$, compute $x_b \leftarrow \mathsf{HE.Dec}(\mathsf{sk}_b, \mathsf{ct}^{(b)})$ for each $b \in \{0,1\}$.
- Output 1 if $x_0 = x_1 \in I_j^{(i^*)}$ and 0 otherwise.

In other words, the tuple $(i^*, \mathsf{ct}^{(0)}, \mathsf{ct}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j)$ is valid if the ciphertexts encrypt the *same* value, and moreover, they are within the valid range.[8] We now show that P_{Valid} is a tree-based additive invariant.

Lemma 5.32. *If Π_{HE} satisfies evaluation correctness, then P_{Valid} is a tree-based additive invariant.*

Proof. Take any $i^* \in [n]$, any $(\mathsf{sk}_0, \mathsf{pk}_0)$, $(\mathsf{sk}_1, \mathsf{pk}_1)$ in the support of $\mathsf{HE.Gen}(1^\lambda, 1^n)$, any triple of indices $j, j_{\mathrm{L}}, j_{\mathrm{R}} \in [2n-1]$ where $j_{\mathrm{L}}, j_{\mathrm{R}}$ are the indices of the children of j, and any set of ciphertexts $\big(\mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}\big)$, $\big(\mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}\big)$. Suppose j has height $h \leq \log n$. For $b \in \{0,1\}$, let $\mathsf{ct}_{\mathsf{sum}}^{(b)} = \mathsf{HE.Add}\big(\mathsf{pk}_b, \mathsf{ct}_{\mathrm{L}}^{(b)}, \mathsf{ct}_{\mathrm{R}}^{(b)}\big)$. Suppose

$$P_{\mathsf{Valid}}(i^*, \mathsf{ct}_{\mathrm{L}}^{(0)}, \mathsf{ct}_{\mathrm{L}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{L}}) = P_{\mathsf{Valid}}(i^*, \mathsf{ct}_{\mathrm{R}}^{(0)}, \mathsf{ct}_{\mathrm{R}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j_{\mathrm{R}}) = 1.$$

This means $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}) = \mathsf{HE.Dec}(\mathsf{sk}_1, \mathsf{ct}_{\mathrm{L}}^{(1)}) \in I_{j_{\mathrm{L}}}^{(i^*)}$ and $\mathsf{HE.Dec}(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{R}}^{(0)}) = \mathsf{HE.Dec}(\mathsf{sk}_1, \mathsf{ct}_{\mathrm{R}}^{(1)}) \in I_{j_{\mathrm{R}}}^{(i^*)}$. As argued above, since $j_{\mathrm{L}}, j_{\mathrm{R}}$ have height $h-1$, we conclude that for $b \in \{0,1\}$,

$$\mathsf{HE.Dec}\big(\mathsf{sk}_b, \mathsf{ct}_{\mathrm{L}}^{(b)}\big), \mathsf{HE.Dec}\big(\mathsf{sk}_b, \mathsf{ct}_{\mathrm{R}}^{(b)}\big) \in [0, 2^{h-1}].$$

Since $2^{h-1} \leq n/2$, we can appeal to evaluation correctness of Π_{H} and conclude that

$$\begin{aligned} \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{sum}}^{(0)}\big) &= \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{L}}^{(0)}\big) + \mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathrm{R}}^{(0)}\big) \\ &= \mathsf{HE.Dec}\big(\mathsf{sk}_1, \mathsf{ct}_{\mathrm{L}}^{(1)}\big) + \mathsf{HE.Dec}\big(\mathsf{sk}_1, \mathsf{ct}_{\mathrm{R}}^{(1)}\big) = \mathsf{HE.Dec}\big(\mathsf{sk}_1, \mathsf{ct}_{\mathsf{sum}}^{(1)}\big) \in I_{j_{\mathrm{L}}} + I_{j_{\mathrm{R}}} = I_j, \end{aligned}$$

We conclude that $P_{\mathsf{Valid}}\big(i^*, \mathsf{ct}_{\mathsf{sum}}^{(0)}, \mathsf{ct}_{\mathsf{sum}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 1$. □

[8] The range check is needed to ensure that all of the ciphertexts decrypt to values within the (bounded) support of the homomorphic encryption scheme. This is necessary to invoke evaluation correctness of Π_{HE} (see the proof of Lemma 5.32).

To leverage the predicate-propagation theorem (Theorem 5.9) to prove Claim 5.31, we now define a mapping $\mathsf{DeriveChal}$ as

$$\mathsf{DeriveChal}(S, i) \coloneqq (S, i) \mapsto (S \setminus \{i\}, S, (i, 0)).$$

Let $\mathsf{Expt} \coloneqq \mathsf{Expt}[P_{\mathsf{Valid}}, \mathsf{DeriveChal}]$ be the predicate propagation experiment from Definition 5.7. First, we argue that

$$\Pr[\mathsf{E}] \leq \Pr[\mathsf{Expt}(\mathcal{A}) = 1], \tag{5.15}$$

where E is the event from Eq. (5.14). By construction, the adversary's view in Hyb_1 and Expt is identical. Suppose E occurs in an execution of Hyb_1. Then the following hold:

- First $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{i^*,0}, 2n-1, \pi) = 1$. By construction of $\mathsf{DeriveChal}$, we have that $\mathsf{idx} = (i^*, 0)$ in the execution of $\mathsf{Expt}(\mathcal{A})$. Hence, this means that $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{idx}}, 2n-1, \pi) = 1$.

- Next, $\mathsf{HE.Dec}\big(\mathsf{sk}_0, \mathsf{ct}_{\mathsf{root}}^{(0)}\big) \neq \mathsf{HE.Dec}\big(\mathsf{sk}_1, \mathsf{ct}_{\mathsf{root}}^{(1)}\big)$. This means $P_{\mathsf{Valid}}\big(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, 2n-1\big) = 0$.

Correspondingly, the output in Expt is also 1 in this case. Hence, we conclude that $\Pr[\mathsf{Expt}(\mathcal{A}) = 1] \geq \Pr[\mathsf{E}]$. To complete the proof, we analyze the predicate propagation hybrid experiment $\mathsf{Expt}_j \coloneqq \mathsf{Expt}_j[P_{\mathsf{Valid}}, \mathsf{DeriveChal}]$.

Lemma 5.33. *If Π_{HE} is perfectly correct and Π_{Com} satisfies computational binding, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $j \in [n]$, it holds that $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists some $j \in [n]$ where $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks computational binding of Π_{Com}.

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to obtain the input length 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$.

2. Algorithm $\mathcal{B}$ outputs the block length $1^{\ell_{\mathsf{ct}}(\lambda, n)}$ and the vector length $2n-1$ to the challenger. The challenger responds with $\mathsf{crs}_{\mathsf{Com}}$.

3. Algorithm $\mathcal{B}$ computes $(S \setminus \{i^*\}, S, (i^*, 0)) \leftarrow \mathsf{DeriveChal}(S, i^*)$. It then samples the following components:
   - Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HE.Gen}(1^\lambda, 1^n)$.
   - Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{2n-1}, 1^s, 1^3, \{j\})$.
   - For each $b \in \{0,1\}$, sample $\mathsf{ct}_{\mathsf{zero}}^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - For each $i \in [n] \setminus \{i^*\}$ and $b \in \{0,1\}$, if $i \in S$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 1)$. If $i \notin S$, sample $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_b, 0)$.
   - Sample $\mathsf{ct}_{i^*}^{(0)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_0, 0)$ and $\mathsf{ct}_{i^*}^{(1)} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}_1, 1)$.
   - For each $b \in \{0,1\}$, let $\big(\mathsf{com}_{\mathsf{hk}}^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\big) \leftarrow \mathsf{Com.Commit}\big(\mathsf{crs}_{\mathsf{Com}}, (\mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)})\big)$.

4. Algorithm $\mathcal{B}$ constructs hk and vk according to Eqs. (5.1) and (5.2):

$$\begin{aligned} \mathsf{hk} &= \Big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{BARG}}, \big\{\mathsf{pk}_b, \mathsf{ct}_{\mathsf{zero}}^{(b)}, \mathsf{ct}_1^{(b)}, \ldots, \mathsf{ct}_n^{(b)}, \sigma_{\mathsf{hk},1}^{(b)}, \ldots, \sigma_{\mathsf{hk},n}^{(b)}\big\}_{b \in \{0,1\}}\Big) \\ \mathsf{vk} &= \big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_{\mathsf{zero}}^{(0)}, \mathsf{ct}_{\mathsf{zero}}^{(1)}, \mathsf{com}_{\mathsf{hk}}^{(0)}, \mathsf{com}_{\mathsf{hk}}^{(1)}\big) \end{aligned}$$

   Algorithm $\mathcal{B}$ gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = \big(\mathsf{ct}_{\mathsf{root}}^{(0)}, \mathsf{ct}_{\mathsf{root}}^{(1)}, \mathsf{com}_0, \mathsf{com}_1, \pi_{\mathsf{dig}}\big)$ and a proof $\pi = \pi_{\mathsf{dig}}$.

6. Algorithm $\mathcal{B}$ extracts $\big(\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j\big) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, \pi, j)$ and parses $\tilde{w}_j = \big(\hat{\mathsf{ct}}^{(0)}, \hat{\mathsf{ct}}^{(1)}, \sigma_{\mathsf{hk}}^{(0)}, \sigma_{\mathsf{hk}}^{(1)}\big)$.

7. Algorithm $\mathcal{B}$ checks if there exists $b \in \{0,1\}$ where $\hat{\mathsf{ct}}^{(b)} \neq \mathsf{ct}_j^{(b)}$ and $\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(b)}, j, \hat{\mathsf{ct}}^{(b)}, \sigma_{\mathsf{hk}}^{(b)}\big) = 1$. If so, it outputs the commitment $\mathsf{com}_{\mathsf{hk}}^{(b)}$, the index j, and the value-opening pairs $\big(\mathsf{ct}_j^{(b)}, \sigma_{\mathsf{hk},j}^{(b)}\big)$ and $\big(\hat{\mathsf{ct}}^{(b)}, \sigma_{\mathsf{hk}}^{(b)}\big)$.

By construction, the challenger samples $\mathsf{crs}_{\mathsf{Com}} \leftarrow \mathsf{Com.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}(\lambda,n)}, 2n-1)$, which matches the specification in Expt_j. This, algorithm $\mathcal{B}$ perfectly simulates an execution of Expt_j for $\mathcal{A}$. By assumption, with probability ε, algorithm $\mathcal{A}$ outputs dig and π such that the experiment outputs 1. This means the following conditions hold:

$$C_{i^*,0}\big(j, (\hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \sigma_j^{(0)}, \sigma_j^{(1)}, \tilde{w}_j)\big) = 1 \quad \text{and} \quad P_{\mathsf{Valid}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 0.$$

We consider two possibilities:

- Suppose $j = i^*$. By construction of $C_{i^*,0}$ (see Fig. 2), this means $\hat{v}_j^{(b)} = \mathsf{ct}_{\mathsf{zero}}^{(b)}$ for $b \in \{0,1\}$. By construction, $\mathsf{ct}_{\mathsf{zero}}^{(b)}$ is an encryption of 0 under pk_b. In this case, $P_{\mathsf{Valid}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 1$, which contradicts the premise.
- Suppose $j \neq i^*$. By construction of $C_{i^*,0}$, there are now two more possibilities:
  - Suppose for $b \in \{0,1\}$, $\hat{v}_j^{(b)} = \mathsf{ct}_{\mathsf{zero}}^{(b)}$. As in the first case, this means $\hat{v}_j^{(0)}$ and $\hat{v}_j^{(1)}$ decrypt to 0 under sk_0 and sk_1, respectively. In this case $P_{\mathsf{Valid}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 1$, which again contradicts the premise.
  - Suppose for $b \in \{0,1\}$, $\hat{v}_j^{(b)} = \hat{\mathsf{ct}}^{(b)}$. In this case, we also have

    $$\begin{aligned} &\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(0)}, j, \hat{\mathsf{ct}}^{(0)}, \sigma_{\mathsf{hk}}^{(0)}\big) = 1 \\ &\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(1)}, j, \hat{\mathsf{ct}}^{(1)}, \sigma_{\mathsf{hk}}^{(1)}\big) = 1. \end{aligned}$$

    Suppose $\hat{\mathsf{ct}}^{(b)} = \mathsf{ct}_j^{(b)}$ for all $b \in \{0,1\}$. In this case, since $j \neq i^*$, the ciphertexts $\mathsf{ct}_j^{(0)}, \mathsf{ct}_j^{(1)}$ are either both encryptions of 0 (if $j \notin S$) or both encryptions of 1 (if $j \in S$). In this case,

    $$P_{\mathsf{Valid}}\big(i^*, \hat{v}_j^{(0)}, \hat{v}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = P_{\mathsf{Valid}}\big(i^*, \mathsf{ct}_j^{(0)}, \mathsf{ct}_j^{(1)}, \mathsf{sk}_0, \mathsf{sk}_1, j\big) = 1,$$

    which contradicts the premise. Thus, if P_{Valid} is not satisfied, we conclude that there exists some $b \in \{0,1\}$ such that $\hat{\mathsf{ct}}^{(b)} \neq \mathsf{ct}_j^{(b)}$.

Thus, there exists some $b \in \{0,1\}$ such that the following holds:

$$\hat{\mathsf{ct}}^{(b)} \neq \mathsf{ct}_j^{(b)} \quad \text{and} \quad \mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(b)}, j, \hat{\mathsf{ct}}^{(b)}, \sigma_{\mathsf{hk}}^{(b)}\big) = 1.$$

Moreover, by correctness of Π_{Com}, we have that

$$\mathsf{Com.Verify}\big(\mathsf{crs}_{\mathsf{Com}}, \mathsf{com}_{\mathsf{hk}}^{(b)}, j, \mathsf{ct}_j^{(b)}, \sigma_{\mathsf{hk},j}^{(b)}\big) = 1.$$

In this case, algorithm $\mathcal{B}$ successfully breaks the binding property of the commitment scheme. □

Since for all $j \in [n]$, it holds that $\Pr[\mathsf{Expt}_j(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$, we can invoke Theorem 5.9 to conclude that $\Pr[\mathsf{Expt}(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. Claim 5.31 now follows via Eqs. (5.14) and (5.15). □

Claim 5.34. *If Π_{HE} is CPA-secure, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\big|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. Follows by an analogous argument as the proof of Claim 5.30. In particular, the reduction obtains pk_0 and $\mathsf{ct}_{i^*}^{(0)}$ from the challenger. It samples $(\mathsf{pk}_1, \mathsf{sk}_1)$ itself which it can use to compute the output (according to the specification in Hyb_2 and Hyb_3). □

Claim 5.35. *If Π_{HE} is perfectly correct and satisfies evaluation correctness, Π_{Com} is computationally binding, and Π_{BARG} satisfies set hiding with extraction, set hiding, and is somewhere extractable, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Follows by an analogous argument as the proof of Claim 5.31. The only difference is that we take the mapping $\mathsf{DeriveChal}$ to be

$$\mathsf{DeriveChal}(S, i) \coloneqq (S, i) \mapsto (S, S, (i, 0)).$$

The rest of the analysis proceeds exactly as before. □

Theorem 5.29 now follows by combining Claims 5.30, 5.31, 5.34 and 5.35. □

6 Zero-Fixing Hash Function from Bilinear Maps

In this section, we give a direct construction of a zero-fixing hash function from composite-order pairing groups. This construction does *not* require making non-black-box use of cryptography (in contrast to the construction from Section 5) and highlights an algebraic approach for building zero-fixing hash functions. We begin by recalling the concept of composite-order pairing groups [BGN05] and the generalized family of subgroup decision assumptions [BWY11].

Definition 6.1 (Composite-Order Bilinear Group)**.** Let $k \in \mathbb{N}$ be a constant. A symmetric k-prime composite-order bilinear group generator is an efficient algorithm CompGroupGen that takes as input the security parameter λ and outputs a description $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i \in [k]}, g, e)$ of a bilinear group where each p_i is a distinct prime where $p_i = 2^{\Omega(\lambda)}$, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order $N = \prod_{i \in [k]} p_i$, g is a generator of $\mathbb{G}$, and $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map (called the "pairing"). We require that the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing operation be efficiently computable.

Notation. Let $\mathbb{G}$ be a cyclic group with order $N = \prod_{i \in [k]} p_i$ and generator g. We write $\mathbb{Z}_N$ to denote the ring of integers modulo N. In the following, for $i \in [k]$, we write $\mathbb{G}_i = \langle g^{N/p_i} \rangle$ to denote the subgroup of $\mathbb{G}$ of order p_i. Throughout this section, we will write g_i to denote a *random* generator of $\mathbb{G}_i$. For a set $S \subseteq [k]$, we write $\mathbb{G}(S)$ to denote subgroup of $\mathbb{G}$ of order $\prod_{i \in S} p_i$. By the Chinese Remainder Theorem, we can write $\mathbb{G}$ as a direct product $\mathbb{G} \cong \mathbb{G}_{p_1} \times \cdots \times \mathbb{G}_{p_k}$. For a group element $h \in \mathbb{G}$, we can write $h = \prod_{i \in [k]} h_i$ where each $h_i \in \mathbb{G}_i$; we refer to h_i as the component of h in the subgroup $\mathbb{G}_i$. If two elements $h_1, h_2 \in \mathbb{G}$ are equal (i.e., $h_1 = h_2$), then for all $i \in [k]$, the component of h_1 and h_2 in $\mathbb{G}_i$ are also equal. We extend this terminology to $\mathbb{G}_T$.

General subgroup decision assumption. We now recall the general subgroup decision assumption formalized in [BWY11]. The general subgroup decision assumption essentially states that for sets $S_0, S_1 \subseteq [k]$, no efficient adversary can distinguish between a random element of $\mathbb{G}(S_0)$ from $\mathbb{G}(S_1)$ even given random elements from $\mathbb{G}(S)$ for any $S \subseteq [k]$ where $S \cap S_0$ and $S \cap S_1$ are both empty or both non-empty. We give the formal definition below:

Definition 6.2 (General Subgroup Decision [BWY11, adapted])**.** Let $k \in \mathbb{N}$ be a constant and let CompGroupGen be a symmetric k-prime composite-order bilinear group generator. For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, we define the general subgroup decision game $\mathsf{ExptSubgroup}_{\mathcal{A}}(\lambda, b)$ for CompGroupGen as follows:

1. At the beginning of the game, algorithm $\mathcal{A}$ outputs two non-empty sets $S_0, S_1 \subseteq [k]$ and any number of sets $T_1, \ldots, T_n \subseteq [k]$. We require that for all $i \in [n]$ either $S_0 \cap T_i = \varnothing = S_1 \cap T_i$ or $S_0 \cap T_i \neq \varnothing \neq S_1 \cap T_i$.

2. The challenger samples $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i \in [k]}, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It compute $N = \prod_{i \in [k]} p_i$ and sets $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$. For each $i \in [n]$, the challenger samples $X_i \xleftarrow{\text{R}} \mathbb{G}(T_i)$. It also samples $Z \leftarrow \mathbb{G}(S_b)$, and gives the challenge $(\mathcal{G}, X_1, \ldots, X_n, Z)$ to the adversary.

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

We say that the general subgroup decision assumption holds with respect to CompGroupGen if for all efficient adversaries $\mathcal{A}$ $\left|\Pr[\mathsf{ExptSubgroup}_{\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{ExptSubgroup}_{\mathcal{A}}(\lambda, 1)]\right| \leq \mathsf{negl}(\lambda)$.

Constructing zero-fixing hash functions. We now describe our construction of a zero-fixing hash function from composite-order bilinear groups. To simplify the main construction, we will describe our construction with a *long* verification key. The verification algorithm of our construction only requires *local* access to the long verification key, so it is straightforward to compile our construction into one with a short verification key using a collision-resistant hash function (see Remark 6.5).

Construction 6.3 (Zero-Fixing Hash Function from Composite-Order Bilinear Maps)**.** Let CompGroupGen be a 6-prime composite-order pairing group. We construct a zero-fixing hash function $\Pi_{\mathsf{H}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{ProveOpen}, \mathsf{VerOpen}, \mathsf{Extract}, \mathsf{ValidateDigest})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^n, S)$: On input a security parameter λ, an input length n, and a set $S \subseteq [n]$, the setup algorithm samples $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i\in[6]}, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $N = \prod_{i\in[6]} p_i$. For each $i \in [6]$, let $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ be a random generator of $\mathbb{G}_i$. Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be the group description. The setup algorithm now constructs the hash key components as follows:
  - **Main components:** For each $i \in [n]$, sample $\alpha_i, \beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$. Set
  $$A_i = \begin{cases} (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_2 g_3 g_4)^{\alpha_i} & i \in S. \end{cases}$$
  For each $i \in [n]$, let $B_i = (g_1 g_5)^{\beta_i}$.
  - **Cross-terms:** For each $i, j \in [n]$ where $i \neq j$, sample $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $C_{i,j} \leftarrow g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}}$.
  - **Digest validation components:** Sample $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $B^* = (g_1 g_5)^{\beta^*}$. For each $i \in [n]$, sample $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $D_i = g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$.

  Output the hash key hk and verification key vk where
  $$\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big)$$
  along with the extraction trapdoor $\mathsf{td} = g_2 g_3$.

- $\mathsf{Hash}(\mathsf{hk}, x)$: On input a hash key $\mathsf{hk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big)$ and an input $x \in \{0,1\}^n$, the hash algorithm computes $h = \prod_{i\in[n]} A_i^{x_i}$ and $u = \prod_{i\in[n]} D_i^{x_i}$. It outputs the digest $\mathsf{dig} = (h, u)$.

- $\mathsf{ValidateDigest}(\mathsf{vk}, \mathsf{dig})$: On input the verification key $\mathsf{vk} = \big(\mathcal{G}, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big)$ and a digest $\mathsf{dig} = (h, u)$, the digest-validation algorithm outputs 1 if $e(h, B^*) = e(g_1, u)$ and 0 otherwise.

- $\mathsf{ProveOpen}(\mathsf{hk}, x, i)$: On input a hash key $\mathsf{hk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big)$, a string $x \in \{0,1\}^n$, and an index $i \in [n]$, the prove algorithm outputs $\sigma = \prod_{j\neq i} C_{j,i}^{x_j}$.

- $\mathsf{VerOpen}(\mathsf{vk}, \mathsf{dig}, i, b, \sigma)$: On input a hash key $\mathsf{vk} = \big(\mathcal{G}, g_1, \{(A_i, B_i)\}_{i\in[n]}, B^*\big)$, a digest $\mathsf{dig} = (h, \pi)$, an index $i \in [n]$, a bit $b \in \{0,1\}$, and an opening σ, the verification algorithm outputs 1 if $e(h, B_i) = e(A_i, B_i)^b \cdot e(g_1, \sigma)$ and 0 otherwise.

- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$: On input a trapdoor td and a digest $\mathsf{dig} = (h, u)$, the extraction algorithm outputs $\mathsf{Matching}$ if $e(h, \mathsf{td}) = 1$ and $\mathsf{NotMatching}$ otherwise.

Theorem 6.4 (Correctness). *Construction 6.3 is correct.*

Proof. Take any $\lambda, n \in \mathbb{N}$ and $x \in \{0,1\}^n$. Let $i \in [n]$ be an index. Suppose $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \varnothing)$, $\mathsf{dig} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$ and $\sigma \leftarrow \mathsf{ProveOpen}(\mathsf{hk}, x, i)$. By construction,
$$\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big).$$
Next, $\mathsf{dig} = (h, u)$ where $h = \prod_{i\in[n]} A_i^{x_i}$ and $u = \prod_{i\in[n]} D_i^{x_i}$. We consider the two properties:

- **Opening correctness:** By construction, $\sigma = \prod_{j\neq i} C_{j,i}^{x_j}$. By orthogonality, we have $e(A_j, B_i) = e(g_1, C_{j,i})$. Then,
$$e(h, B_i) = \prod_{j\in[n]} e(A_j, B_i)^{x_j} = e(A_i, B_i)^{x_i} \prod_{j\neq i} e(A_j, B_i)^{x_j} = e(A_i, B_i)^{x_i} \prod_{j\neq i} e(g_1, C_{j,i})^{x_j} = e(A_i, B_i)^{x_i} e(g_1, \sigma),$$
so $\mathsf{VerOpen}(\mathsf{vk}, \mathsf{dig}, i, x_i, \sigma) = 1$, as required.

- **Digest correctness:** Again by orthogonality, we have $e(A_i, B^*) = e(g_1, D_i)$, so

$$e(h, B^*) = \prod_{j \in [n]} e(A_i, B^*)^{x_i} = \prod_{j \in [n]} e(g_1, D_i)^{x_i} = e(g_1, u),$$

and $\mathsf{ValidateDigest}(\mathsf{vk}, \mathsf{dig}) = 1$. □

Remark 6.5 (Supporting Fast Verification)**.** As described, the size of the verification key in Construction 6.3 scales linearly with the input length n. This is incompatible with the succinctness requirements needed by our monotone BARG construction (Construction 4.4). However, it is straightforward to compress the verification key using a collision-resistant hash function. Observe that the verification algorithm VerOpen in Construction 6.3 only requires *local* access to the verification key (i.e., it only needs to read elements A_i and B_i). The approach then is to only include a succinct *commitment* com to $(A_1, B_1), \ldots, (A_n, B_n)$ in the verification key; the associated openings are included as part of the (long) hash key. Then, the opening for an index i would additionally contain the elements A_i, B_i as well as their openings with respect to com. The verifier would check that the correct elements A_i and B_i were provided and that they satisfy the verification relation. Security still holds as long as the scheme is computationally binding (since a computationally-bounded adversary would not be able to open com at i to any value other than (A_i, B_i)).

Security properties. We now show that each of the security requirements from Definition 3.1 holds under the (general) subgroup decision assumption.

Theorem 6.6 (Set Hiding)**.** *If the general subgroup decision holds with respect to* $\mathsf{CompGroupGen}$*, then Construction 6.3 satisfies set hiding.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the set hiding game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is experiment $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0)$. At the beginning of the game, the adversary outputs an input length n and a set $S \subseteq [n]$. Then the challenger samples $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i \in [6]}, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It samples generators $g_i \xleftarrow{R} \mathbb{G}_i$ and sets $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ where $N = \prod_{i \in [6]} p_i$. It constructs the hash key components as follows:
  - **Main components:** For each $i \in [n]$, sample $\alpha_i, \beta_i \xleftarrow{R} \mathbb{Z}_N$ Set $A_i = (g_1 g_4)^{\alpha_i}$ and $B_i = (g_1 g_5)^{\beta_i}$.
  - **Cross-terms:** For each $i, j \in [n]$ where $i \neq j$, sample $r_{i,j} \xleftarrow{R} \mathbb{Z}_N$ and let $C_{i,j} = g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}}$.
  - **Digest validation components:** Sample $\beta^* \xleftarrow{R} \mathbb{Z}_N$ and let $B^* = (g_1 g_5)^{\beta^*}$. For each $i \in [n]$, sample $r_i^* \xleftarrow{R} \mathbb{Z}_N$ and let $D_i = g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$.

  The challenger gives the hash key hk and verification key vk to $\mathcal{A}$ where

  $$\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\big).$$

  Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

- Hyb_1: Same as Hyb_0, except the challenger now sets $C_{i,j} = A_i^{\beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}}$ and $D_i = A_i^{\beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$. In particular, in this experiment, the exponents α_i only show up in the definition of A_i.

- Hyb_2: Same as Hyb_1, except for $i \in S$, the challenger now sets $A_i = (g_1 g_2 g_3 g_4)^{\alpha_i}$.

- Hyb_3: Same as Hyb_2, except the challenger now sets $C_{i,j} = g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}}$ and $D_i = g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$. This is experiment $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 1)$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of adjacent hybrid experiments.

Lemma 6.7. $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$

Proof. The outputs of $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ are identically distributed. The only difference between these two distributions is the distribution of the cross-terms $C_{i,j}$ and D_i. According to the specification of Hyb_1,

$$C_{i,j} = A_i^{\beta_j}(g_2g_3g_4g_5)^{r_{i,j}} = (g_1g_4)^{\alpha_i\beta_j}(g_2g_3g_4g_5)^{r_{i,j}} = g_1^{\alpha_i\beta_j}(g_2g_3g_5)^{r_{i,j}} g_4^{r_{i,j}+\alpha_i\beta_j}.$$

Since $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ (and independent of all other quantities in hk, vk), the distribution of $r_{i,j} + \alpha_i\beta_j \bmod p_4$ is uniform over $\mathbb{Z}_{p_4}$. We conclude that the distribution of $C_{i,j}$ in Hyb_1 is distributed exactly as in Hyb_0. A similar analysis applies to D_i, and the claim holds. □

Lemma 6.8. *If the general subgroup decision assumption holds with respect to* $\mathsf{CompGroupGen}$*, then it holds that* $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 4\}$ and $S_1 = \{1, 2, 3, 4\}$ and the sets $\{1\}$, $\{4\}$, $\{2, 3, 4\}$, and $\{5\}$ to the subgroup decision challenger.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_4, X_{234}, g_5, Z)$, where g_i is used to denote the random generator of $\mathbb{G}_i$ and $X_{234} \xleftarrow{\text{R}} \mathbb{G}(\{2, 3, 4\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n and a set $S \subseteq [n]$. The challenger samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It constructs the components of the hash key as follows:

   $$A_i = \begin{cases} (g_1g_4)^{\alpha_i} & i \notin S \\ Z^{\alpha_i} & i \in S \end{cases} \quad \text{and} \quad B_i = (g_1g_5)^{\beta_i} \quad \text{and} \quad C_{i,j} = A_i^{\beta_j}(X_{234}g_5)^{r_{i,j}}.$$

   Next, it samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$. It sets

   $$B^* = (g_1g_5)^{\beta^*} \quad \text{and} \quad D_i = A_i^{\beta^*}(X_{234}g_5)^{r_i^*}.$$

   It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which algorithm $\mathcal{B}$ also outputs.

The subgroup decision challenger samples the generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ exactly as in Hyb_1 and Hyb_2. Moreover, $X_{234} = (g_2g_3g_4)^{\gamma}$ for $\gamma \xleftarrow{\text{R}} \mathbb{Z}_N$. Since the only element that depends on $r_{i,j}$ is $C_{i,j}$ and $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$, the distribution of $X_{234}^{r_{i,j}}$ is identical to the distribution of $(g_2g_3g_4)^{r_{i,j}}$. Similarly, the distribution of $X_{234}^{r_i^*}$ is identically distributed to $(g_2g_3g_4)^{r_i^*}$. We now consider the two possibilities:

- Suppose $Z = (g_1g_4)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_1 with α_i replaced by $\alpha_i t$ when $i \in S$. As long as t is non-zero modulo p_1 and p_4 (which happens with negligible probability), the distribution of $\alpha_i t$ is uniform over $\mathbb{Z}_{p_1p_4}$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

- Suppose $Z = (g_1g_2g_3g_4)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_2 with α_i replaced by $\alpha_i t$ whenever $i \in S$. As long as t is non-zero modulo p_1, p_2, p_3, and p_4 (which happens with negligible probability), the distribution of $\alpha_i t$ is uniform over $\mathbb{Z}_{p_1p_2p_3p_4}$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ succeeds with probability that is negligibly close to ε and the claim holds. □

Lemma 6.9. $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

Proof. The outputs of $\mathsf{Hyb}_2(\mathcal{A})$ and $\mathsf{Hyb}_3(\mathcal{A})$ are identically distributed by an analogous argument as the proof of Lemma 6.7. In particular, in Hyb_2, for $i \in S$,

$$C_{i,j} = A_i^{\beta_j}(g_2g_3g_4g_5)^{r_{i,j}} = (g_1g_2g_3g_4)^{\alpha_i\beta_j}(g_2g_3g_4g_5)^{r_{i,j}} = g_1^{\alpha_i\beta_j}(g_2g_3g_4)^{r_{i,j}+\alpha_i\beta_j}g_5^{r_{i,j}}.$$

Again since $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ and independent of all other quantities in hk, vk, the distribution of $r_{i,j} + \alpha_i\beta_j \bmod p_2p_3p_4$ is identical to the distribution of $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_{p_2p_3p_4}$. A similar argument applies to $C_{i,j}$ for $i \notin S$ and the D_i terms. □

Set hiding now follows by combining Lemmas 6.7 to 6.9. □

Theorem 6.10 (Index Hiding with Extracted Guess)**.** *If the general subgroup decision holds with respect to* $\mathsf{CompGroupGen}$*, then Construction 6.3 satisfies one-sided index hiding with extracted guess (Definition 3.3).*

Proof. Let $\mathcal{A}$ be an efficient adversary for the index hiding with extracted guess game. We define a sequence of hybrid experiments:

- Hyb_0: This is the index hiding with extracted guess experiment $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 0)$. Namely, the adversary starts by outputting the input length 1^n, a set $S \subseteq [n]$ and an index $i^* \in S$. The challenger samples $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i\in[6]}, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It samples generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ and sets $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ where $N = \prod_{i\in[6]} p_i$. It constructs the hash key components as follows:
    - **Main components:** For each $i \in [n]$, sample $\alpha_i, \beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$ Set $A_i = (g_1g_2g_3g_4)^{\alpha_i}$ if $i \in S \setminus \{i^*\}$ and $A_i = (g_1g_4)^{\alpha_i}$ if $i \notin S$. Set $A_{i^*} = (g_1g_4)^{\alpha_{i^*}}$. Then, set $B_i = (g_1g_5)^{\beta_i}$.
    - **Cross-terms:** For each $i, j \in [n]$ where $i \neq j$, sample $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $C_{i,j} = g_1^{\alpha_i\beta_j}(g_2g_3g_4g_5)^{r_{i,j}}$.
    - **Digest validation components:** Sample $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $B^* = (g_1g_5)^{\beta^*}$. For each $i \in [n]$, sample $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $D_i = g_1^{\alpha_i\beta^*}(g_2g_3g_4g_5)^{r_i^*}$.

  The challenger gives the hash key hk and verification key vk to $\mathcal{A}$ where

  $$\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big).$$

  Algorithm $\mathcal{A}$ then outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. The output of the experiment is 1 if

  $$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_2g_3) = 1.$$

- Hyb_1: Same as Hyb_0 except the challenger sets $C_{i^*,j} = A_{i^*}^{\beta_j}(g_2g_3g_4g_5)^{r_{i^*,j}}$ for all $j \neq i^*$ and $C_{i,i^*} = B_{i^*}^{\alpha_i}(g_2g_3g_4g_5)^{r_{i,i^*}}$ for all $i \neq i^*$. Similarly, the challenger sets $D_{i^*} = A_{i^*}^{\beta^*}(g_2g_3g_4g_5)^{r_i^*}$. In particular, the exponents α_{i^*} and β_{i^*} only shows up in A_{i^*} and B_{i^*}, respectively.
- Hyb_2: Same as Hyb_1 except the challenger sets $B_{i^*} = (g_1g_5g_6)^{\beta_{i^*}}$.
- Hyb_3: Same as Hyb_2 except the challenger sets $A_{i^*} = (g_1g_4g_6)^{\alpha_{i^*}}$.
- Hyb_4: Same as Hyb_3 except the challenger sets $A_{i^*} = (g_1g_2g_4g_6)^{\alpha_{i^*}}$.
- Hyb_5: Same as Hyb_4 except the challenger sets $A_{i^*} = (g_1g_2g_3g_4g_6)^{\alpha_{i^*}}$.
- Hyb_6: Same as Hyb_5 except the challenger sets $A_{i^*} = (g_1g_2g_3g_4)^{\alpha_{i^*}}$. Namely, there is no longer a $\mathbb{G}_6$ component in A_{i^*}.
- Hyb_7: Same as Hyb_6 except the challenger sets $B_{i^*} = (g_1g_5)^{\beta_{i^*}}$. Namely, there is no longer a $\mathbb{G}_6$ component in B_{i^*}.
- Hyb_8: Same as Hyb_7 except the challenger sets $C_{i^*,j} = g_1^{\alpha_{i^*}\beta_j}(g_2g_3g_4g_4)^{r_{i^*,j}}$ and $C_{i,i^*} = g_1^{\alpha_i\beta_{i^*}}(g_2g_3g_4g_5)^{r_{i,i^*}}$. Similarly, the challenger sets $D_{i^*} = g_1^{\alpha_{i^*}\beta^*}(g_2g_3g_4g_5)^{r_i^*}$. This is experiment $\mathsf{ExptIHE}_{\mathcal{A}}(\lambda, 1)$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of adjacent hybrid experiments. Our goal is to show that $\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] \ge \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \mathsf{negl}(\lambda)$.

Lemma 6.11. $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$

Proof. The outputs of $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ are identically distributed. The only difference between these two distributions is the distribution of $C_{i^*,j}$, C_{i,i^*}, and D_{i^*}. According to the specification of Hyb_1,

$$\begin{aligned} C_{i^*,j} &= A_{i^*}^{\beta_j}(g_2g_3g_4g_5)^{r_{i^*,j}} = (g_1g_4)^{\alpha_{i^*}\beta_j}(g_2g_3g_4g_5)^{r_{i^*,j}} = g_1^{\alpha_{i^*}\beta_j} g_4^{r_{i^*,j}+\alpha_{i^*}\beta_j}(g_2g_3g_5)^{r_{i^*,j}} \\ C_{i,i^*} &= B_{i^*}^{\alpha_i}(g_2g_3g_4g_5)^{r_{i,i^*}} = (g_1g_5)^{\alpha_i\beta_{i^*}}(g_2g_3g_4g_5)^{r_{i,i^*}} = g_1^{\alpha_i\beta_{i^*}}(g_2g_3g_4)^{r_{i,i^*}} g_5^{r_{i,i^*}+\alpha_i\beta_{i^*}} \\ D_{i^*} &= A_{i^*}^{\beta^*}(g_2g_3g_4g_5)^{r_{i^*}^*} = (g_1g_4)^{\alpha_{i^*}\beta^*}(g_2g_3g_4g_5)^{r_{i^*}^*} = g_1^{\alpha_{i^*}\beta^*} g_4^{r_{i^*}^*+\alpha_{i^*}\beta^*}(g_2g_3g_5)^{r_{i^*}^*} \end{aligned}$$

Since $r_{i^*,j}, r_{i,i^*}, r_{i^*}^* \xleftarrow{\text{R}} \mathbb{Z}_N$ (and independent of all other quantities in hk, vk), the elements $C_{i^*,j}, C_{i,i^*}, D_{i^*}$ are distributed exactly as in Hyb_0. □

Lemma 6.12. *If the general subgroup decision assumption holds with respect to* $\mathsf{CompGroupGen}$*, then it holds that* $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \ge \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 5\}$ and $S_1 = \{1, 5, 6\}$ and the sets $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$ to the subgroup decision challenger.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, g_5, Z)$, where g_i is used to denote the random generator of $\mathbb{G}_i$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n, a set $S \subseteq [n]$ and an index $i^* \in S$. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} (g_1g_4)^{\alpha_i} & i \notin S \setminus \{i^*\} \\ (g_1g_2g_3g_4)^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} Z & i = i^* \\ (g_1g_5)^{\beta_i} & i \ne i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i\beta_j}(g_2g_3g_4g_5)^{r_{i,j}} & i, j \ne i^* \\ A_{i^*}^{\beta_j}(g_2g_3g_4g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i}(g_2g_3g_4g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

   Next, it samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$. It sets

$$B^* = (g_1g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i\beta^*}(g_2g_3g_4g_5)^{r_i^*} & i \ne i^* \\ A_{i^*}^{\beta^*}(g_2g_3g_4g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

   It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \ne j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. Algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2g_3) = 1.$$

The subgroup decision challenger samples the generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ exactly as in Hyb_1 and Hyb_2. Moreover, all of the components other than B_{i^*} is constructed exactly as described in Hyb_1 and Hyb_2. Thus, it suffices to consider the distribution of B^*. We consider the two possibilities:

- Suppose $Z = (g_1g_5)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_1 with $\beta_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.
- Suppose $Z = (g_1g_5g_6)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_2 with $\beta_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ succeeds with probability ε and the claim holds. □

Lemma 6.13. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 4\}$ and $S_1 = \{1, 4, 6\}$ and the sets $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{1, 6\}$ to the subgroup decision challenger.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, g_5, X_{16}, Z)$, where g_i is used to denote the random generator of $\mathbb{G}_i$ and $X_{16} \xleftarrow{\text{R}} \mathbb{G}(\{1, 6\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} Z & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_2 g_3 g_4)^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} X_{16} g_5^{\beta_{i^*}} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (g_2 g_3 g_4 g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

   Next, it samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$. It sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (g_2 g_3 g_4 g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

   It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. Algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1.$$

The subgroup decision challenger samples the generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ exactly as in Hyb_2 and Hyb_3. We can write $X_{16} = (g_1 g_6)^{\gamma_{16}}$ where $\gamma_{16} \xleftarrow{\text{R}} \mathbb{Z}_N$. The value $\gamma_{16} \bmod p_1 p_6$ corresponds to the value of $\beta_{i^*} \bmod p_1 p_6$ in Hyb_2 and Hyb_3. The remaining components other than A_{i^*} are sampled exactly as required in Hyb_2 and Hyb_3, so it suffices to consider A_{i^*}. We consider the two possibilities:

- Suppose $Z = (g_1 g_4)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_1 with $\alpha_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.
- Suppose $Z = (g_1 g_4 g_6)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_3 with $\alpha_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ succeeds with probability ε and the claim holds. □

Lemma 6.14. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \mathsf{negl}(\lambda)$.

Proof. Let $\mathsf{dig} = (h, u)$ be the digest output by $\mathcal{A}$ in an execution of Hyb_3 and Hyb_4. For an index $i \in \{3, 4\}$, we define events $\mathsf{E}_{i,1}$ and $\mathsf{E}_{i,2}$:

- $\mathsf{E}_{i,1}$: This is the event $e(h, B_{i^*}) = e(g_1, \sigma)$ and $e(h, g_3) = 1$ occurring in Hyb_i.
- $\mathsf{E}_{i,2}$: This is the event that $e(h, g_2) = 1$ occurring in Hyb_i.

By definition, the output in Hyb_i is 1 if and only if both events $\mathsf{E}_{i,1}$ and $\mathsf{E}_{i,2}$ occur. To complete the proof, we start by showing the following two properties: (1) $\left|\Pr[\mathsf{E}_{3,1}] - \Pr[\mathsf{E}_{4,1}]\right| = \mathsf{negl}(\lambda)$; and (2) $\Pr[\mathsf{E}_{4,1} \wedge \neg\mathsf{E}_{4,2}] = \mathsf{negl}(\lambda)$.

Claim 6.15. *If the subgroup decision assumption holds with respect to* CompGroupGen*, then* $\left|\Pr[\mathsf{E}_{3,1}] - \Pr[\mathsf{E}_{4,1}]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose $\left|\Pr[\mathsf{E}_{3,1}] - \Pr[\mathsf{E}_{4,1}]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 4, 6\}$ and $S_1 = \{1, 2, 4, 6\}$ and the sets $\{1\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{2, 3, 4\}$.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_3, g_4, g_5, g_6, X_{234}, Z)$ where g_i is a random generator of $\mathbb{G}_i$ and $X_{234} \xleftarrow{\text{R}} \mathbb{G}(\{2, 3, 4\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} Z & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 X_{234})^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} (g_1 g_5 g_6)^{\beta_{i^*}} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (X_{234} g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (X_{234} g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (X_{234} g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

   It samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (X_{234} g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (X_{234} g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

   It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_3) = 1.$$

We first argue that $\mathcal{B}$ correctly simulates the hash key according to the specification of Hyb_3 and Hyb_4. First, we can write $X_{234} = (g_2 g_3 g_4)^{\gamma_{234}}$ where $\gamma_{234} \xleftarrow{\text{R}} \mathbb{Z}_N$. Since the challenger samples $r_{i,j}$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are distributed exactly as they are in Hyb_3 and Hyb_4 unless γ_{234} is zero in the p_2, p_3, or p_4 components. This happens with negligible probability over the choice of γ_{234}. Similarly, A_i for $i \in S \setminus \{i^*\}$ is distributed identically; the distributions of $\alpha_i \bmod p_2 p_3 p_4$ and that of $\alpha_i \gamma_{234} \bmod p_2 p_3 p_4$ when $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$ are identical as long as γ_{234} is non-zero in the p_2, p_3, and p_4 subgroups. It suffices to consider the distribution of A_{i^*}:

- Suppose $Z = (g_1 g_4 g_6)^t$ for $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_3 with $\alpha_{i^*} = t \bmod p_1 p_4 p_6$. Thus, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{3,1}]$.

- Suppose $Z = (g_1 g_2 g_4 g_6)^t$ for $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_2 with $\alpha_{i^*} = t \bmod p_1 p_4 p_6$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{4,1}]$.

We conclude that $\mathcal{B}$ succeeds with probability ε and the claim holds. □

Claim 6.16. *If the subgroup decision assumption holds with respect to* CompGroupGen*, then* $\Pr[\mathsf{E}_{4,1} \wedge \neg \mathsf{E}_{4,2}] = \mathsf{negl}(\lambda)$.

Proof. Suppose $\Pr[\mathsf{E}_{4,1} \wedge \neg \mathsf{E}_{4,2}] \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{3, 6\}$ and $S_1 = \{2, 3, 6\}$ and the sets $\{1\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{2, 3\}$, $\{2, 6\}$.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_3, g_4, g_5, g_6, X_{23}, X_{26}, Z)$ where g_i is a random generator of $\mathbb{G}_i$, $X_{23} \xleftarrow{\text{R}} \mathbb{G}(\{2, 3\})$, and $X_{26} \xleftarrow{\text{R}} \mathbb{G}(\{2, 6\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} (g_1 g_4)^{\alpha_{i^*}} X_{26} & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_4 X_{23})^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} (g_1 g_5 g_6)^{\beta_{i^*}} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (X_{23} g_4 g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (X_{23} g_4 g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (X_{23} g_4 g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

It samples $\beta^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (X_{23} g_4 g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (X_{23} g_4 g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

It sets $\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\big)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_3) = 1 \quad \text{and} \quad e(h, Z) = 1.$$

We first argue that algorithm $\mathcal{B}$ correctly simulates an execution of Hyb_4 for $\mathcal{A}$. First, we can write $X_{23} = (g_2 g_3)^{\gamma_{23}}$ and $X_{26} = (g_2 g_6)^{\gamma_{26}}$, where $\gamma_{23}, \gamma_{26} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. With overwhelming probability, γ_{23} is non-zero in the p_2 and p_3 subgroups. In the following, we will assume this is the case. Since the challenger samples $r_{i,j}, r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are distributed exactly as required in Hyb_4. Similarly, the distribution of A_{i^*} coincides with setting α_{i^*} mod $p_2 p_6$ as γ_{26} mod $p_2 p_6$ and α_i mod $p_2 p_3$ as $\alpha_i \gamma_{23}$ mod $p_2 p_3$. Since each $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $\gamma_{26} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, this matches the distribution in Hyb_4. Thus, with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{B}$ outputs $\mathsf{dig} = (h, u)$ such that

$$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_3) = 1 \quad \text{and} \quad e(h, g_2) \neq 1. \tag{6.1}$$

Suppose Eq. (6.1) holds. We first claim that with overwhelming probability, $e(h, g_6) = 1$. Suppose otherwise. If $\beta_{i^*} \neq 0 \bmod p_6$ (which happens with overwhelming probability), then B_{i^*} is non-zero in the order p_6 subgroup. If $e(h, g_6) \neq 1$, then $e(h, B_{i^*})$ is non-zero in the p_6 subgroup. By construction $e(g_1, \sigma)$ is necessarily 0 in the p_6 subgroup, so it can no longer be the case that $e(h, B_{i^*}) = e(g_1, \sigma)$. Now consider the probability that $\mathcal{B}$ outputs 1:

- Suppose $Z = (g_3 g_6)^t$ for some $t \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. As argued previously, with overwhelming probability, if Eq. (6.1) holds, then $e(h, g_6) = 1$. Since $e(h, g_3) = 1$, this means $e(h, Z) = 1$ and algorithm $\mathcal{B}$ outputs 1 with overwhelming probability.
- Suppose $Z = (g_2 g_3 g_6)^t$ for some $t \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. Since $e(h, g_2) \neq 1$, then $e(h, Z) \neq 1$ so long as $t \neq 0 \bmod p_2$, which holds with overwhelming probability. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with negligible probability.

We now compute the advantage of $\mathcal{B}$. We consider three possibilities:

- Suppose $\mathcal{A}$ outputs (h, u) such that $e(h, B_{i^*}) \neq e(g_1, \sigma)$ or $e(h, g_3) \neq 1$. Then, the output of $\mathcal{B}$ is always 0.
- Suppose $\mathcal{A}$ outputs (h, u) such that $e(h, B_{i^*}) = e(g_1, \sigma)$ and $e(h, g_3) = 1$ and $e(h, g_2) = 1$. By the previous analysis, if $e(h, B_{i^*}) = e(g_1, \sigma)$, then $e(h, g_6) = 1$ with overwhelming probability. Since $Z = (g_3 g_6)^t$ or $Z = (g_2 g_3 g_6)^t$, this means that $\mathcal{B}$ outputs 1 with overwhelming probability regardless for both possible values of Z.
- Suppose $\mathcal{A}$ outputs (h, u) such that $e(h, B_{i^*}) = e(g_1, \sigma)$ and $e(h, g_3) = 1$ and $(h, g_2) \neq 1$. By the earlier analysis, this case occurs with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, and in this case, algorithm $\mathcal{B}$ outputs 1 with probability $1 - \mathsf{negl}(\lambda)$ if $Z = (g_3 g_6)^t$ and with probability $\mathsf{negl}(\lambda)$ if $Z = (g_2 g_3 g_6)^t$.

Let ρ_1, ρ_2, ρ_3 be the probabilities of each of these cases. Then,

$$\begin{aligned} \Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 0) = 1] &= \rho_2(1 - \mathsf{negl}(\lambda)) + \rho_3 \cdot (1 - \mathsf{negl}(\lambda)) \\ \Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 1) = 1] &= \rho_2(1 - \mathsf{negl}(\lambda)) + \rho_3 \cdot \mathsf{negl}(\lambda) \end{aligned}$$

The advantage of $\mathcal{B}$ is thus $\rho_3 - \mathsf{negl}(\lambda) \geq \varepsilon - \mathsf{negl}(\lambda)$, and the claim holds. □

To complete the proof we have that

$$\begin{aligned}\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] &= \Pr[\mathsf{E}_{4,1} \wedge \mathsf{E}_{4,2}] \\ &= \Pr[\mathsf{E}_{4,1}] - \Pr[\mathsf{E}_{4,1} \wedge \neg\mathsf{E}_{4,2}] \\ &\geq \Pr[\mathsf{E}_{4,1}] - \mathsf{negl}(\lambda) && \text{by Claim 6.16} \\ &\geq \Pr[\mathsf{E}_{3,1}] - \mathsf{negl}(\lambda) && \text{by Claim 6.15,}\end{aligned}$$

and Lemma 6.14 follows. □

Lemma 6.17. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \mathsf{negl}(\lambda)$.

Proof. The proof follows by a similar argument as that of Lemma 6.14. Let $\mathsf{dig} = (h, u)$ be the digest output by $\mathcal{A}$ in an execution of Hyb_4 and Hyb_5. For an index $i \in \{4, 5\}$, we define events $\mathsf{E}_{i,1}$ and $\mathsf{E}_{i,2}$ as in the proof of Lemma 6.14 (changes marked in green):

- $\mathsf{E}_{i,1}$: This is the event $e(h, B_{i^*}) = e(g_1, \sigma)$ and $e(h, g_2) = 1$ occurring in Hyb_i.
- $\mathsf{E}_{i,2}$: This is the event that $e(h, g_3) = 1$ occurring in Hyb_i.

By definition, the output in Hyb_i is 1 if and only if both events $\mathsf{E}_{i,1}$ and $\mathsf{E}_{i,2}$ occur. To complete the proof, we start by showing the following two properties: (1) $\left|\Pr[\mathsf{E}_{4,1}] - \Pr[\mathsf{E}_{5,1}]\right| = \mathsf{negl}(\lambda)$; and (2) $\Pr[\mathsf{E}_{5,1} \wedge \neg\mathsf{E}_{5,2}] = \mathsf{negl}(\lambda)$.

Claim 6.18. *If the subgroup decision assumption holds with respect to* CompGroupGen*, then* $\left|\Pr[\mathsf{E}_{4,1}] - \Pr[\mathsf{E}_{5,1}]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose $\left|\Pr[\mathsf{E}_{4,1}] - \Pr[\mathsf{E}_{5,1}]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 2, 4, 6\}$ and $S_1 = \{1, 2, 3, 4, 6\}$ and the sets $\{1\}$, $\{2\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{2, 3, 4\}$.
2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_4, g_5, g_6, X_{234}, Z)$ where g_i is a random generator of $\mathbb{G}_i$ and $X_{234} \xleftarrow{\mathrm{R}} \mathbb{G}(\{2, 3, 4\})$.
3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} Z & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 X_{234})^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} (g_1 g_5 g_6)^{\beta_{i^*}} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (X_{234} g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (X_{234} g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (X_{234} g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

   It samples $\beta^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (X_{234} g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (X_{234} g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

   It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_2) = 1.$$

We first argue that $\mathcal{B}$ correctly simulates the hash key according to the specification of Hyb_4 and Hyb_5. First, we can write $X_{234} = (g_2 g_3 g_4)^{\gamma_{234}}$ where $\gamma_{234} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. Since the challenger samples $r_{i,j}$ and $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are distributed exactly as they are in Hyb_4 and Hyb_5 unless γ_{234} is zero in the p_2, p_3, or p_4 components. This happens with negligible probability over the choice of γ_{234}. Similarly, A_i for $i \in S \setminus \{i^*\}$ is distributed identically; the reduction algorithm effectively samples $\alpha_i \bmod p_2 p_3 p_4$, $\alpha_i \gamma_{234} \bmod p_2 p_3 p_4$ which are identically distributed when $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and γ_{234} is non-zero in the p_2, p_3, and p_4 subgroups. It suffices to consider the distribution of $\mathbf{A}_{i^*}$:

- Suppose $Z = (g_1 g_2 g_4 g_6)^t$ for $t \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_4 with $\alpha_{i^*} = t \bmod p_1 p_2 p_4 p_6$. Thus, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{4,1}]$.
- Suppose $Z = (g_1 g_2 g_3 g_4 g_6)^t$ for $t \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_5 with $\alpha_{i^*} = t \bmod p_1 p_2 p_3 p_4 p_6$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{5,1}]$.

We conclude that $\mathcal{B}$ succeeds with probability ε and the claim holds. □

Claim 6.19. *If the subgroup decision assumption holds with respect to* $\mathsf{CompGroupGen}$*, then* $\Pr[\mathsf{E}_{5,1} \wedge \neg \mathsf{E}_{5,2}] = \mathsf{negl}(\lambda)$.

Proof. Suppose $\Pr[\mathsf{E}_{5,1} \wedge \neg \mathsf{E}_{5,2}] \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{2, 6\}$ and $S_1 = \{2, 3, 6\}$ and the sets $\{1\}$, $\{2\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{2, 3\}$, $\{3, 6\}$.
2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_4, g_5, g_6, X_{23}, X_{36}, Z)$ where g_i is a random generator of $\mathbb{G}_i$, $X_{23} \xleftarrow{\mathrm{R}} \mathbb{G}(\{2, 3\})$, and $X_{36} \xleftarrow{\mathrm{R}} \mathbb{G}(\{3, 6\})$.
3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets
$$A_i = \begin{cases} (g_1 g_2 g_4)^{\alpha_{i^*}} X_{36} & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_4 X_{23})^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} (g_1 g_5 g_6)^{\beta_{i^*}} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (X_{23} g_4 g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (X_{23} g_4 g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (X_{23} g_4 g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$
It samples $\beta^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets
$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (X_{23} g_4 g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (X_{23} g_4 g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$
It sets $\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\big)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ outputs 1 if
$$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_2) = 1 \quad \text{and} \quad e(h, Z) = 1.$$

We first argue that algorithm $\mathcal{B}$ correctly simulates an execution of Hyb_4 for $\mathcal{A}$. First, we can write $X_{23} = (g_2 g_3)^{\gamma_{23}}$ and $X_{36} = (g_3 g_6)^{\gamma_{36}}$, where $\gamma_{23}, \gamma_{36} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. With overwhelming probability γ_{23} is non-zero in the p_2 and p_3 subgroups. In the following, we will assume this is the case. Since the challenger samples $r_{i,j}, r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are distributed exactly as required in Hyb_5. Similarly, the distribution of A_{i^*} coincides with setting $\alpha_{i^*} \bmod p_2 p_3 p_6$ as $\gamma_{236} \bmod p_2 p_3 p_6$ and $\alpha_i \bmod p_2 p_3$ as $\alpha_i \gamma_{23} \bmod p_2 p_3$. Since each $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $\gamma_{23}, \gamma_{236} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, this matches the distribution in Hyb_5. Thus, with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{B}$ outputs $\mathsf{dig} = (h, u)$ such that
$$e(h, B_{i^*}) = e(g_1, \sigma) \quad \text{and} \quad e(h, g_2) = 1 \quad \text{and} \quad e(h, g_3) \neq 1. \tag{6.2}$$

Suppose Eq. (6.2) holds. We first claim that with overwhelming probability, $e(h, g_6) = 1$. Suppose otherwise. If $\beta_{i^*} \neq 0 \bmod p_6$ (which happens with overwhelming probability), then B_{i^*} is non-zero in the order p_6 subgroup. If $e(h, g_6) \neq 1$, then $e(h, B_{i^*})$ is non-zero in the p_6 subgroup. By construction $e(g_1, \sigma)$ is necessarily 0 in the p_6 subgroup, so it can no longer be the case that $e(h, B_{i^*}) = e(g_1, \sigma)$. Now consider the probability that $\mathcal{B}$ outputs 1:

- Suppose $Z = (g_2 g_6)^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$. As argued previously, with overwhelming probability, if Eq. (6.2) holds, then $e(h, g_6)$. Since $e(h, g_2) = 1$, this means $e(h, Z) = 1$ and algorithm $\mathcal{B}$ outputs 1 with overwhelming probability.

- Suppose $Z = (g_2 g_3 g_6)^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$. Since $e(h, g_3) \neq 1$, then $e(h, Z) \neq 1$ so long as $t \neq 0 \bmod p_3$, which holds with overwhelming probability. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with negligible probability.

We now compute the advantage of $\mathcal{B}$. We consider three possibilities:

- Suppose $\mathcal{A}$ outputs (h, u) such that $e(h, B_{i^*}) \neq e(g_1, \sigma)$ or $e(h, g_2) \neq 1$. Then, the output of $\mathcal{B}$ is always 0.

- Suppose $\mathcal{A}$ outputs (h, u) such that $e(h, B_{i^*}) = e(g_1, \sigma)$ and $e(h, g_2) = 1$ and $e(h, g_3) = 1$. By the previous analysis, if $e(h, B_{i^*}) = e(g_1, \sigma)$, then $e(h, g_6) = 1$ with overwhelming probability. Since $Z = (g_2 g_6)^t$ or $Z = (g_2 g_3 g_6)^t$, this means that $\mathcal{B}$ outputs 1 with overwhelming probability regardless for both possible values of Z.

- Suppose $\mathcal{A}$ outputs (h, u) such that $e(h, B_{i^*}) = e(g_1, \sigma)$ and $e(h, g_2) = 1$ and $(h, g_3) \neq 1$. By the earlier analysis, this case occurs with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, and in this case, algorithm $\mathcal{B}$ outputs 1 with probability $1 - \mathsf{negl}(\lambda)$ if $Z = (g_2 g_6)^t$ and with probability $\mathsf{negl}(\lambda)$ if $Z = (g_2 g_3 g_6)^t$.

Let ρ_1, ρ_2, ρ_3 be the probabilities of each of these cases. Then,

$$\begin{aligned}\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 0) = 1] &= \rho_2(1 - \mathsf{negl}(\lambda)) + \rho_3 \cdot (1 - \mathsf{negl}(\lambda)) \\ \Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 1) = 1] &= \rho_2(1 - \mathsf{negl}(\lambda)) + \rho_3 \cdot \mathsf{negl}(\lambda)\end{aligned}$$

The advantage of $\mathcal{B}$ is thus $\rho_3 - \mathsf{negl}(\lambda) \geq \varepsilon - \mathsf{negl}(\lambda)$, and the claim holds. □

To complete the proof we have that

$$\begin{aligned}\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] &= \Pr[\mathsf{E}_{5,1} \wedge \mathsf{E}_{5,2}] \\ &= \Pr[\mathsf{E}_{5,1}] - \Pr[\mathsf{E}_{5,1} \wedge \neg \mathsf{E}_{5,2}] \\ &\geq \Pr[\mathsf{E}_{5,1}] - \mathsf{negl}(\lambda) && \text{by Claim 6.19} \\ &\geq \Pr[\mathsf{E}_{4,1}] - \mathsf{negl}(\lambda) && \text{by Claim 6.18,}\end{aligned}$$

and Lemma 6.17 follows. □

Lemma 6.20. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\left|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. The proof is analogous to the proof of Lemma 6.13, except that the challenge subgroups are $S_0 = \{1, 2, 3, 4\}$ and $S_1 = \{1, 2, 3, 4, 6\}$. □

Lemma 6.21. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\left|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. The proof is analogous to the proof of Lemma 6.12, except that A_{i^*} is now in $\mathbb{G}(\{1, 2, 3, 4\})$. □

Lemma 6.22. $\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]$.

Proof. This follows by an analogous argument as the proof of Lemma 6.11. □

Combining Lemmas 6.11 to 6.14, 6.17 and 6.20 to 6.22, the index hiding with extracted guess property holds. □

Theorem 6.23 (Zero Fixing)**.** *If the general subgroup decision holds with respect to* CompGroupGen*, then Construction 6.3 satisfies selective zero-fixing.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the zero fixing game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is the selective version of the experiment $\mathsf{ExptZF}_{\mathcal{A}}(\lambda)$. Namely, the adversary starts by outputting an input length 1^n, a set $S \subseteq [n]$, and an index $i^* \in S$. The challenger samples $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i \in [6]}, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It samples generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ and sets $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ where $N = \prod_{i \in [6]} p_i$. It constructs the hash key components as follows:
  - **Main components:** For each $i \in [n]$, sample $\alpha_i, \beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$ Set $A_i = (g_1 g_4)^{\alpha_i}$ if $i \notin S$ and $A_i = (g_1 g_2 g_3 g_4)^{\alpha_i}$ if $i \in S$. Then, set $B_i = (g_1 g_5)^{\beta_i}$.
  - **Cross-terms:** For each $i, j \in [n]$ where $i \neq j$, sample $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $C_{i,j} = g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}}$.
  - **Digest validation components:** Sample $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $B^* = (g_1 g_5)^{\beta^*}$. For each $i \in [n]$, sample $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $D_i = g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$.

  The challenger gives the hash key hk and verification key vk to $\mathcal{A}$ where

  $$\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\big).$$

  Algorithm $\mathcal{A}$ then outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. The output of the experiment is 1 if

  $$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1.$$

- Hyb_1: Same as Hyb_0 except the challenger now sets $C_{i^*,j} = A_{i^*}^{\beta_j} (g_2 g_3 g_4 g_5)^{r_{i^*,j}}$ for all $j \neq i^*$ and $C_{i,i^*} = B_{i^*}^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_{i,i^*}}$ for all $i \neq i^*$. Similarly, the challenger sets $D_{i^*} = A_{i^*}^{\beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$. In particular, the exponents α_{i^*} and β_{i^*} only shows up in A_{i^*} and B_{i^*}, respectively.
- Hyb_2: Same as Hyb_1 except the challenger now sets $B_{i^*} = (g_1 g_5 g_6)^{\beta_{i^*}}$.
- Hyb_3: Same as Hyb_2 except the challenger now sets $A_{i^*} = (g_1 g_2 g_3 g_4 g_6)^{\alpha_{i^*}}$
- Hyb_4: Same as Hyb_3 except the experiment outputs 0 if $e(h, g_6) \neq 1$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of adjacent hybrid experiments.

Lemma 6.24. $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$

Proof. This follows by the same argument as in the proof of Lemma 6.11. □

Lemma 6.25. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\big|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ where $\big|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\big| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 5\}$ and $S_1 = \{1, 5, 6\}$ and the sets $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$ to the subgroup decision challenger.
2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, g_5, Z)$, where g_i is used to denote the random generator of $\mathbb{G}_i$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n, a set $S \subseteq [n]$ and an index $i^* \in S$. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_2 g_3 g_4)^{\alpha_i} & i \in S \end{cases} \quad \text{and} \quad B_i = \begin{cases} Z & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (g_2 g_3 g_4 g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

Next, it samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$. It sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (g_2 g_3 g_4 g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. Algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1.$$

The subgroup decision challenger samples the generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ exactly as in Hyb_1 and Hyb_2. Moreover, all of the components other than B_{i^*} is constructed exactly as described in Hyb_1 and Hyb_2. Thus, it suffices to consider the distribution of B^*. We consider the two possibilities:

- Suppose $Z = (g_1 g_5)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_1 with $\beta_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.
- Suppose $Z = (g_1 g_5 g_6)^t$ where $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_2 with $\beta_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ succeeds with probability ε and the claim holds. □

Lemma 6.26. *If the general subgroup decision assumption holds with respect to* CompGroupGen, *then it holds that* $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{1, 2, 3, 4\}$ and $S_1 = \{1, 2, 3, 4, 6\}$ and the sets $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{1, 6\}$ to the subgroup decision challenger.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, g_5, X_{16}, Z)$, where g_i is used to denote the random generator of $\mathbb{G}_i$ and $X_{16} \xleftarrow{\text{R}} \mathbb{G}(\{1, 6\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} Z & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_2 g_3 g_4)^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} X_{16} g_5^{\beta_{i^*}} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (g_2 g_3 g_4 g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

Next, it samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$. It sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (g_2 g_3 g_4 g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. Algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1.$$

The subgroup decision challenger samples the generators $g_i \overset{\text{R}}{\leftarrow} \mathbb{G}_i$ exactly as in Hyb_2 and Hyb_3. We can write $X_{16} = (g_1 g_6)^{\gamma_{16}}$ where $\gamma_{16} \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$. The value $\gamma_{16} \bmod p_1 p_6$ corresponds to the value of $\beta_{i^*} \bmod p_1 p_6$ in Hyb_2 and Hyb_3. The remaining components other than A_{i^*} are sampled exactly as required in Hyb_2 and Hyb_3, so it suffices to consider A_{i^*}. We consider the two possibilities:

- Suppose $Z = (g_1 g_2 g_3 g_4)^t$ where $t \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$. This corresponds to an execution of Hyb_1 with $\alpha_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.
- Suppose $Z = (g_1 g_2 g_3 g_4 g_6)^t$ where $t \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$. This corresponds to an execution of Hyb_3 with $\alpha_{i^*} = t$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ succeeds with probability ε and the claim holds. □

Lemma 6.27. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. Since the only difference between Hyb_3 and Hyb_4 is the extra condition, it must be the case that with probability ε, algorithm $\mathcal{A}$ outputs (h, u, σ) such that

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1 \quad \text{and} \quad e(h, g_6) \neq 1. \tag{6.3}$$

In all other cases, the output in Hyb_3 and Hyb_4 are identical. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{2, 3, 5\}$ and $S_1 = \{2, 3, 5, 6\}$ and the sets $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{2, 6\}, \{5, 6\}$ to the subgroup decision challenger.
2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, g_5, X_{26}, X_{56}, Z)$, where g_i is used to denote the random generator of $\mathbb{G}_i$, $X_{26} \overset{\text{R}}{\leftarrow} \mathbb{G}(\{2, 6\})$, and $X_{56} \overset{\text{R}}{\leftarrow} \mathbb{G}(\{5, 6\})$.
3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$, $\beta_i \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$, $r_{i,j} \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = \begin{cases} (g_1 g_3 g_4)^{\alpha_{i^*}} X_{26} & i = i^* \\ (g_1 g_4)^{\alpha_i} & i \notin S \\ (g_1 g_2 g_3 g_4)^{\alpha_i} & i \in S \setminus \{i^*\} \end{cases} \quad \text{and} \quad B_i = \begin{cases} g_1^{\beta_{i^*}} X_{56} & i = i^* \\ (g_1 g_5)^{\beta_i} & i \neq i^* \end{cases} \quad \text{and} \quad C_{i,j} = \begin{cases} g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}} & i, j \neq i^* \\ A_{i^*}^{\beta_j} (g_2 g_3 g_4 g_5)^{r_{i^*,j}} & i = i^* \\ B_{i^*}^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_{i,i^*}} & j = i^*. \end{cases}$$

   Next, it samples $\beta^* \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$ and $r_i^* \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$ for each $i \in [n]$. It sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = \begin{cases} g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*} & i \neq i^* \\ A_{i^*}^{\beta^*} (g_2 g_3 g_4 g_5)^{r_{i^*}^*} & i = i^*. \end{cases}$$

   It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
4. Algorithm $\mathcal{A}$ outputs a digest $\mathsf{dig} = (h, u)$ and an opening σ. Algorithm $\mathcal{B}$ outputs 1 if

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1 \quad \text{and} \quad e(h, Z) = 1.$$

The subgroup decision challenger samples the generators $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ exactly as in Hyb_3 and Hyb_4. We can write $X_{26} = (g_2 g_6)^{\gamma_{26}}$ and $X_{56} = (g_5 g_6)^{\gamma_{56}}$ where $\gamma_{26}, \gamma_{56} \xleftarrow{\text{R}} \mathbb{Z}_N$. The value $\gamma_{26} \bmod p_1 p_6$ corresponds to the value of $\alpha_{i^*} \bmod p_2 p_6$ while the value γ_{56} corresponds to the value of $\beta_{i^*} \bmod p_5 p_6$. Thus, algorithm $\mathcal{B}$ perfectly simulates the hash key for algorithm $\mathcal{A}$. Thus, with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{A}$ outputs (h, u, σ) that satisfies Eq. (6.3). Then, we have the following:

- It must be the case that $e(h, g_5) = 1$. Suppose otherwise. This means that h is non-zero in the G_5 subgroup. Consider the first verification condition $e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma)$. If h is non-zero in the $\mathbb{G}_5$ component, then the left-hand side $e(h, B_{i^*})$ is non-zero in the order-p_5 subgroup unless $\gamma_{56} = 0 \bmod p_5$, which happens with negligible probability. However, the right-hand side is guaranteed to be zero in the order p_5 subgroup (since neither A_{i^*} nor g_1 have non-zero components in $\mathbb{G}_5$).
- Suppose $Z = (g_2 g_3 g_5)^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$. Since $e(h, g_2 g_3) = 1$ and $e(h, g_5) = 1$, this means that $e(g, Z) = 1$ and algorithm $\mathcal{B}$ always outputs 1.
- Suppose $Z = (g_2 g_3 g_5 g_6)^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$. From Eq. (6.3), we have that $e(h, g_6) \neq 1$, so h has a non-zero component in the G_6 subgroup. As long as $t \bmod p_6$ is non-zero (which happens with overwhelming probability), then $e(h, Z) \neq 1$. In this case, algorithm $\mathcal{B}$ outputs 1 with negligible probability.

We have established that when Eq. (6.3) holds, algorithm $\mathcal{B}$ is able to successfully distinguish the subgroup decision challenge. To complete the proof, we show that when Eq. (6.3) does not hold,[9] then algorithm $\mathcal{B}$'s behavior is *independent* of the challenge Z.

1. Suppose $\mathcal{A}$ outputs (h, u, σ) where either $e(h, B_{i^*}) \neq e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma)$ or $e(h, g_2 g_3) \neq 1$. In this case, algorithm $\mathcal{B}$ always outputs 0.
2. Suppose $\mathcal{A}$ outputs (h, u, σ) where $e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma)$, $e(h, g_2 g_3) = 1$, and $e(h, g_6) = 1$. Since $e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma)$, our earlier analysis implies that with overwhelming probability over the choice of γ_{56}, $e(h, g_5) = 1$. Thus, in this case, $e(h, g_2 g_3 g_5 g_6) = 1$, so $e(h, Z) = 1$ for both possible choice of Z. As such, algorithm $\mathcal{B}$ always outputs 1 in this case.
3. Finally, suppose $\mathcal{A}$ outputs (h, u, σ) that satisfies Eq. (6.3). By our analysis above, algorithm $\mathcal{B}$ outputs 1 with probability 1 when $Z = (g_2 g_3 g_5)^t$ and probability $\mathsf{negl}(\lambda)$ when $Z = (g_2 g_3 g_5 g_6)^t$.

If we let ρ_1, ρ_2, ρ_3 be the probabilities of each of these possible cases, then we have

$$\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 0) = 1] = \rho_2 + \rho_3$$
$$\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 1) = 1] = \rho_2 + \rho_3 \cdot \mathsf{negl}(\lambda)$$

The advantage of $\mathcal{B}$ is thus $\rho_3(1 - \mathsf{negl}(\lambda))$. By our above analysis, we have that $\rho_3 \geq \varepsilon - \mathsf{negl}(\lambda)$ and so algorithm $\mathcal{B}$ breaks the general subgroup decision assumption with advantage at least $\varepsilon - \mathsf{negl}(\lambda)$. □

Lemma 6.28. $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Proof. In order for the output of Hyb_4 to be 1, the adversary $\mathcal{A}$ must output (h, u, σ) such that

$$e(h, B_{i^*}) = e(A_{i^*}, B_{i^*}) \cdot e(g_1, \sigma) \quad \text{and} \quad e(h, g_2 g_3) = 1 \quad \text{and} \quad e(h, g_6) = 1.$$

We claim that this can only happen with negligible probability over the choice of α_{i^*} and β_{i^*}. By construction in Hyb_4, as long as $\alpha_{i^*}, \beta_{i^*} \neq 0 \bmod p_6$ (which holds with overwhelming probability), $e(A_{i^*}, B_{i^*})$ will have a non-zero component in the order p_6-subgroup. However, if $e(h, g_6) = 1$, then h is zero in the order p_6-subgroup. Likewise, $e(g_1, \sigma)$ does not have an order p_6 subgroup. This means the left-hand side of the verification relation is zero in the order-p_6 subgroup while the right-hand side is non-zero. As such, the verification relation is unsatisfiable as long as $\alpha_{i^*}, \beta_{i^*} \neq 0 \bmod p_6$. □

[9] Note that algorithm $\mathcal{B}$ cannot check for itself whether Eq. (6.3) occurs or not since it does not know g_6 (and indeed, knowledge of g_6 would trivially break the subgroup decision assumption). Thus, our proof strategy is simply to argue that when Eq. (6.3) does not happen, then the behavior of algorithm $\mathcal{B}$ is independent of the challenge Z.

Combining Lemmas 6.24 to 6.28, we have that $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$, and zero fixing security holds. □

Theorem 6.29 (Extractor Validity)**.** *If the general subgroup decision holds with respect to* CompGroupGen*, then Construction 6.3 satisfies extractor validity.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the extractor validity game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is experiment $\mathsf{ExptEV}_{\mathcal{A}}(\lambda)$. Namely, the adversary starts by outputting an input length 1^n. The challenger samples $(\mathbb{G}, \mathbb{G}_T, \{p_i\}_{i\in[6]}, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It samples generators $g_i \xleftarrow{\mathrm{R}} \mathbb{G}_i$ and sets $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, where $N = \prod_{i\in[6]} p_i$. It constructs the hash key components as follows:
  - **Main components:** For each $i \in [n]$, sample $\alpha_i, \beta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ Set $A_i = (g_1 g_4)^{\alpha_i}$ and $B_i = (g_1 g_5)^{\beta_i}$.
  - **Cross-terms:** For each $i, j \in [n]$ where $i \neq j$, sample $r_{i,j} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $C_{i,j} = g_1^{\alpha_i \beta_j} (g_2 g_3 g_4 g_5)^{r_{i,j}}$.
  - **Digest validation components:** Sample $\beta^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $B^* = (g_1 g_5)^{\beta^*}$. For each $i \in [n]$, sample $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $D_i = g_1^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*}$.

  The challenger gives the hash key hk and verification key vk to $\mathcal{A}$ where

  $$\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i\in[n]}, \{C_{i,j}\}_{i\neq j}, B^*\big).$$

  Algorithm $\mathcal{A}$ then outputs a digest $\mathsf{dig} = (h, u)$ and the output of the experiment is 1 if

  $$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_2 g_3) \neq 1.$$

- Hyb_1: Same as Hyb_0, except the challenger now sets $D_i = (B^*)^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_i^*}$ for all $i \in [n]$. In particular, the exponent β^* only shows up in the definition of B^*.
- Hyb_2: Same as Hyb_1, except the challenger now sets $B^* = (g_1 g_2 g_5)^{\beta^*}$.
- Hyb_3: Same as Hyb_2, except the challenger now sets $B^* = (g_1 g_2 g_3 g_5)^{\beta^*}$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of adjacent hybrid experiments.

Lemma 6.30. $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$

Proof. The outputs of $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ are identically distributed. The only difference between these two distributions is the distribution of D_i. According to the specification of Hyb_1,

$$D_i = (B^*)^{\alpha_i} (g_2 g_3 g_4 g_5)^{r_i^*} = (g_1 g_5)^{\alpha_i \beta^*} (g_2 g_3 g_4 g_5)^{r_i^*} = g_1^{\alpha_i \beta^*} (g_2 g_3 g_4)^{r_i^*} g_5^{r_i^* + \alpha_i \beta^*}.$$

Since $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ (and independent of all other quantities in hk, vk), the distribution of $r_i^* + \alpha_i \beta^* \bmod p_5$ is uniform over $\mathbb{Z}_{p_5}$. We conclude that the distribution of D_i in Hyb_1 is distributed exactly as in Hyb_0. □

Lemma 6.31. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \leq \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] + \mathsf{negl}(\lambda)$.

Proof. Let $\mathsf{dig} = (h, u)$ be the digest output by $\mathcal{A}$ in an execution of Hyb_1 and Hyb_2. For an index $i \in \{1, 2\}$, we define events $\mathsf{E}_{i,1}$ and $\mathsf{E}_{i,2}$:

- $\mathsf{E}_{i,1}$: This is the event $e(h, B^*) = e(g_1, u)$ and $e(h, g_3) \neq 1$ occurring in Hyb_i.
- $\mathsf{E}_{i,2}$: This is the event $e(h, B^*) = e(g_1, u)$ and $e(h, g_3) = 1$ and $e(h, g_2) \neq 1$ occurring in Hyb_i.

If the output of Hyb_i is 1, exactly one of $\mathsf{E}_{i,1}$ or $\mathsf{E}_{i,2}$ *must* happen (note that these events are mutually exclusive). Thus, for $i \in \{1, 2\}$, we can write

$$\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] = \Pr[\mathsf{E}_{i,1}] + \Pr[\mathsf{E}_{i,2}]. \tag{6.4}$$

We now analyze the probabilities of these events:

Claim 6.32. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then it holds that* $|\Pr[\mathsf{E}_{1,1}] - \Pr[\mathsf{E}_{2,1}]| = \mathsf{negl}(\lambda)$*.*

Proof. Suppose $|\Pr[\mathsf{E}_{1,1}] - \Pr[\mathsf{E}_{2,1}]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{5\}$ and $S_1 = \{2, 5\}$ and the sets $\{1\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{2, 5\}$.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_3, g_4, g_5, X_{25}, Z)$ where g_i is a random generator of $\mathbb{G}_i$, $X_{25} \xleftarrow{\text{R}} \mathbb{G}(\{2, 5\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

   $$A_i = (g_1 g_4)^{\alpha_i} \quad \text{and} \quad B_i = (g_1 g_5)^{\beta_i} \quad \text{and} \quad C_{i,j} = g_1^{\alpha_i \beta_j} (X_{25} g_3 g_4)^{r_{i,j}}.$$

   It samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets

   $$B^* = g_1^{\beta^*} Z \quad \text{and} \quad D_i = (B^*)^{\alpha_i} (X_{25} g_3 g_4)^{r_i^*}.$$

   It sets $\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\big)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ outputs 1 if

   $$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_3) \neq 1.$$

We first argue that $\mathcal{B}$ correctly simulates the hash key according to the specification of Hyb_1 and Hyb_2. First, we can write $X_{25} = (g_2 g_5)^{\gamma_{25}}$ where $\gamma_{25} \xleftarrow{\text{R}} \mathbb{Z}_N$. Since the challenger samples $r_{i,j}$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are distributed exactly as they are in Hyb_1 and Hyb_2 unless γ_{25} is zero in the p_2 or p_5 components. This happens with negligible probability over the choice of γ_{25}. It suffices to consider the distribution of B^*:

- Suppose $Z = g_5^t$ for $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_1 with $\beta^* = t \bmod p_5$. Thus, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{1,1}]$.
- Suppose $Z = (g_2 g_5)^t$ for $t \xleftarrow{\text{R}} \mathbb{Z}_N$. This corresponds to an execution of Hyb_2 with $\beta^* = t \bmod p_2 p_5$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{2,1}]$.

We conclude that algorithm $\mathcal{B}$ succeeds with advantage $\varepsilon - \mathsf{negl}(\lambda)$ and the claim follows. □

Claim 6.33. *If the general subgroup decision assumption holds with respect to* CompGroupGen*, then* $\Pr[\mathsf{E}_{1,2}] = \mathsf{negl}(\lambda)$*.*

Proof. Suppose $\Pr[\mathsf{E}_{1,2}] \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{2, 3\}$ and $S_1 = \{3\}$ and the sets $\{1\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{2, 3\}$ to the subgroup decision challenger.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_3, g_4, g_5, X_{23}, Z)$ where g_i is a random generator of $\mathbb{G}_i$ and $X_{23} \xleftarrow{\text{R}} \mathbb{G}(\{2, 3\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = (g_1 g_4)^{\alpha_i} \quad \text{and} \quad B_i = (g_1 g_5)^{\beta_i} \quad \text{and} \quad C_{i,j} = g_1^{\alpha_i \beta_j} (X_{23} g_4 g_5)^{r_{i,j}}.$$

It samples $\beta^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets

$$B^* = (g_1 g_5)^{\beta^*} \quad \text{and} \quad D_i = (B^*)^{\alpha_i} (X_{23} g_4 g_5)^{r_i^*}.$$

It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ output 1 if the following hold:

$$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_3) = 1 \quad \text{and} \quad e(h, Z) = 1.$$

We first argue that algorithm $\mathcal{B}$ correctly simulates an execution of Hyb_1 for $\mathcal{A}$. First, we can write $X_{23} = (g_2 g_3)^{\gamma_{23}}$ where $\gamma_{23} \xleftarrow{\text{R}} \mathbb{Z}_N$. Since the challenger samples $r_{i,j}, r_i^* \xleftarrow{\text{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are statistically close to that in Hyb_1 unless γ_{23} is zero in the p_2 or p_3 components. This happens with negligible probability over the choice of γ_{23}. In the following analysis, we assume that γ_{23} is non-zero in both its p_2 and p_3 components. Thus, with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{B}$ outputs $\mathsf{dig} = (h, u)$ such that event $\mathsf{E}_{1,2}$ occurs. This means

$$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_3) = 1 \quad \text{and} \quad e(h, g_2) \neq 1. \tag{6.5}$$

Suppose Eq. (6.5) holds. We consider the probability that $\mathcal{B}$ outputs 1:

- If $Z = (g_2 g_3)^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$ and $e(h, g_2) \neq 1$, then as long as $t \neq 0 \bmod p_2$, it will be the case that $e(h, Z) \neq 1$, so algorithm $\mathcal{B}$ outputs 0.
- If $Z = g_3^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$ and $e(h, g_3) = 1$, then algorithm $\mathcal{B}$ always outputs 1.

We now compute the advantage of $\mathcal{B}$. We consider three possibilities:

1. Suppose $\mathcal{A}$ outputs (h, u) where either $e(h, B^*) \neq e(g_1, u)$ or $e(h, g_3) \neq 1$. Then, the output of $\mathcal{B}$ is always 0.

2. Suppose $\mathcal{A}$ outputs (h, u) where $e(h, B^*) = e(g_1, u)$, $e(h, g_3) = 1$, and $e(h, g_2) = 1$. Since $Z = (g_2 g_3)^t$ or $Z = g_3^t$, in both cases, $e(h, Z) = 1$ and algorithm $\mathcal{B}$ always outputs 1.

3. Suppose $\mathcal{A}$ outputs (h, u) such that Eq. (6.5) holds. By the above analysis, this case happens with probability at least $\varepsilon - \mathsf{negl}(\lambda)$. Then algorithm $\mathcal{B}$ outputs 1 with negligible probability if $Z = (g_2 g_3)^t$ and with probability 1 if $Z = g_3^t$.

Let ρ_1, ρ_2, ρ_3 be the probabilities of each of these cases. Then,

$$\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 0) = 1] = \rho_2 + \rho_3 \cdot \mathsf{negl}(\lambda)$$
$$\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 1) = 1] = \rho_2 + \rho_3$$

The advantage of $\mathcal{B}$ is thus $\rho_3(1 - \mathsf{negl}(\lambda)) \geq \varepsilon - \mathsf{negl}(\lambda)$, and the claim holds. □

Returning to the proof of Lemma 6.31, we appeal to Eq. (6.4) to write

$$\begin{aligned}
\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] &= \Pr[\mathsf{E}_{1,1}] + \Pr[\mathsf{E}_{1,2}] && \text{by Eq. (6.4)} \\
&\leq \Pr[\mathsf{E}_{1,1}] + \mathsf{negl}(\lambda) && \text{by Claim 6.33} \\
&\leq \Pr[\mathsf{E}_{2,1}] + \mathsf{negl}(\lambda) && \text{by Claim 6.32} \\
&\leq \Pr[\mathsf{E}_{2,1}] + \Pr[\mathsf{E}_{2,2}] + \mathsf{negl}(\lambda) && \\
&= \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] + \mathsf{negl}(\lambda) && \text{by Eq. (6.4),}
\end{aligned}$$

which proves the lemma. □

Lemma 6.34. *If the general subgroup decision assumption holds with respect to* $\mathsf{CompGroupGen}$*, then it holds that* $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \le \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] + \mathsf{negl}(\lambda)$.

Proof. This follows by a similar argument as the proof of Lemma 6.31. Let $\mathsf{dig} = (h, u)$ be the digest output by $\mathcal{A}$ in an execution of Hyb_2 and Hyb_3. For an index $i \in \{2, 3\}$, we define an analogous set of events $\mathsf{E}_{i,1}$ and $\mathsf{E}_{i,2}$ as in the proof of Lemma 6.31 (changes marked in green):

- $\mathsf{E}_{i,1}$: This is the event $e(h, B^*) = e(g_1, u)$ and $e(h, g_2) \neq 1$ occurring in Hyb_i.
- $\mathsf{E}_{i,2}$: This is the event $e(h, B^*) = e(g_1, u)$ and $e(h, g_2) = 1$ and $e(h, g_3) \neq 1$ occurring in Hyb_i.

Once again, we can write $\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] \Pr[\mathsf{E}_{i,1}] + \Pr[\mathsf{E}_{i,2}]$. We analyze the probabilities of each of these events:

Claim 6.35. *If the general subgroup decision assumption holds with respect to* $\mathsf{CompGroupGen}$*, then it holds that* $|\Pr[\mathsf{E}_{2,1}] - \Pr[\mathsf{E}_{3,1}]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{E}_{2,1}] - \Pr[\mathsf{E}_{3,1}]| \ge \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{5\}$ and $S_1 = \{3, 5\}$ and the sets $\{1\}$, $\{2\}$, $\{4\}$, $\{5\}$, $\{3, 5\}$.
2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_4, g_5, X_{35}, Z)$ where g_i is a random generator of $\mathbb{G}_i$, $X_{35} \xleftarrow{R} \mathbb{G}(\{3, 5\})$.
3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{R} \mathbb{Z}_N$, $\beta_i \xleftarrow{R} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{R} \mathbb{Z}_N$ for $i, j \in [n]$. It sets
$$A_i = (g_1 g_4)^{\alpha_i} \quad \text{and} \quad B_i = (g_1 g_5)^{\beta_i} \quad \text{and} \quad C_{i,j} = g_1^{\alpha_i \beta_j} (X_{35} g_2 g_4)^{r_{i,j}}.$$
It samples $\beta^* \xleftarrow{R} \mathbb{Z}_N$ and $r_i^* \xleftarrow{R} \mathbb{Z}_N$ for each $i \in [n]$, and sets
$$B^* = (g_1 g_2)^{\beta^*} Z \quad \text{and} \quad D_i = (B^*)^{\alpha_i} (X_{35} g_2 g_4)^{r_i^*}.$$
It sets $\mathsf{hk} = \mathsf{vk} = \left(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\right)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.
4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ outputs 1 if
$$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_2) \neq 1.$$

We first argue that $\mathcal{B}$ correctly simulates the hash key according to the specification of Hyb_2 and Hyb_3. First, we can write $X_{35} = (g_3 g_5)^{\gamma_{35}}$ where $\gamma_{35} \xleftarrow{R} \mathbb{Z}_N$. Since the challenger samples $r_{i,j}$ and $r_i^* \xleftarrow{R} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are distributed exactly as they are in Hyb_2 and Hyb_3 unless γ_{35} is zero in the p_3 or p_5 components. This happens with negligible probability over the choice of γ_{35}. It suffices to consider the distribution of B^*:

- Suppose $Z = g_5^t$ for $t \xleftarrow{R} \mathbb{Z}_N$. This corresponds to an execution of Hyb_2 with $\beta^* = t \bmod p_5$. Thus, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{2,1}]$.
- Suppose $Z = (g_2 g_5)^t$ for $t \xleftarrow{R} \mathbb{Z}_N$. This corresponds to an execution of Hyb_3 with $\beta^* = t \bmod p_2 p_5$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{E}_{3,1}]$.

We conclude that algorithm $\mathcal{B}$ succeeds with advantage $\varepsilon - \mathsf{negl}(\lambda)$ and the claim follows. □

Claim 6.36. *If the general subgroup decision assumption holds with respect to* $\mathsf{CompGroupGen}$*, then* $\Pr[\mathsf{E}_{2,2}] = \mathsf{negl}(\lambda)$.

Proof. Suppose $\Pr[\mathsf{E}_{2,2}] \ge \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the general subgroup decision assumption:

1. At the beginning of the game, algorithm $\mathcal{B}$ submits $S_0 = \{2, 3\}$ and $S_1 = \{2\}$ and the sets $\{1\}$, $\{2\}$, $\{4\}$, $\{5\}$, $\{2, 3\}$ to the subgroup decision challenger.

2. The challenger replies with the challenge $(\mathcal{G}, g_1, g_2, g_4, g_5, X_{23}, Z)$ where g_i is a random generator of $\mathbb{G}_i$ and $X_{23} \xleftarrow{\mathrm{R}} \mathbb{G}(\{2, 3\})$.

3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ who outputs the input length 1^n. Algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $\beta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, $r_{i,j} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for $i, j \in [n]$. It sets

$$A_i = (g_1 g_4)^{\alpha_i} \quad \text{and} \quad B_i = (g_1 g_5)^{\beta_i} \quad \text{and} \quad C_{i,j} = g_1^{\alpha_i \beta_j} (X_{23} g_4 g_5)^{r_{i,j}}.$$

It samples $\beta^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for each $i \in [n]$, and sets

$$B^* = (g_1 g_2 g_5)^{\beta^*} \quad \text{and} \quad D_i = (B^*)^{\alpha_i} (X_{23} g_4 g_5)^{r_i^*}.$$

It sets $\mathsf{hk} = \mathsf{vk} = \big(\mathcal{G}, g_1, \{A_i, B_i, D_i\}_{i \in [n]}, \{C_{i,j}\}_{i \neq j}, B^*\big)$ and gives $(\mathsf{hk}, \mathsf{vk})$ to $\mathcal{A}$.

4. After $\mathcal{A}$ outputs the digest $\mathsf{dig} = (h, u)$, algorithm $\mathcal{B}$ output 1 if the following hold:

$$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_2) = 1 \quad \text{and} \quad e(h, Z) = 1.$$

We first argue that algorithm $\mathcal{B}$ correctly simulates an execution of Hyb_2 for $\mathcal{A}$. First, we can write $X_{23} = (g_2 g_3)^{\gamma_{23}}$ where $\gamma_{23} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. Since the challenger samples $r_{i,j}, r_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and each of these values is used exactly once in the construction of hk, the distributions of $C_{i,j}$ and D_i are statistically close to that in Hyb_2 unless γ_{23} is zero in the p_2 or p_3 component. This happens with negligible probability over the choice of γ_{23}, so in the following analysis, we assume that this is not the case. Thus, with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{B}$ outputs $\mathsf{dig} = (h, u)$ such that

$$e(h, B^*) = e(g_1, u) \quad \text{and} \quad e(h, g_2) = 1 \quad \text{and} \quad e(h, g_3) \neq 1. \tag{6.6}$$

Suppose Eq. (6.6) holds. We consider the probability that $\mathcal{B}$ outputs 1:

- If $Z = (g_2 g_3)^t$ for some $t \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $e(h, g_3) \neq 1$, then with overwhelming probability over the choice of t, $e(h, Z) \neq 1$, and algorithm $\mathcal{B}$ outputs 0.
- If $Z = g_2^t$ for some $t \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $e(h, g_2) = 1$, so algorithm $\mathcal{B}$ outputs 1.

We now compute the advantage of $\mathcal{B}$. We consider three possibilities:

1. Suppose $\mathcal{A}$ outputs (h, u) where either $e(h, B^*) \neq e(g_1, u)$ or $e(h, g_2) \neq 1$. Then, the output of $\mathcal{B}$ is always 0.

2. Suppose $\mathcal{A}$ outputs (h, u) where $e(h, B^*) = e(g_1, u)$, $e(h, g_2) = 1$, and $e(h, g_3) = 1$. Since either $Z = (g_2 g_3)^t$ or $Z = g_2^t$, in both cases, $e(h, Z) = 1$ and algorithm $\mathcal{B}$ outputs 1.

3. Suppose $\mathcal{A}$ outputs (h, u) such that Eq. (6.6) holds. By the above analysis, this case happens with probability at least $\varepsilon - \mathsf{negl}(\lambda)$. Then algorithm $\mathcal{B}$ outputs 1 with negligible probability if $Z = (g_2 g_3)^t$ and with probability 1 if $Z = g_2^t$.

Let ρ_1, ρ_2, ρ_3 be the probabilities of each of these cases. Then,

$$\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 0) = 1] = \rho_2 + \rho_3 \cdot \mathsf{negl}(\lambda)$$
$$\Pr[\mathsf{ExptSubgroup}_{\mathcal{B}}(\lambda, 1) = 1] = \rho_2 + \rho_3$$

The advantage of $\mathcal{B}$ is thus $\rho_3(1 - \mathsf{negl}(\lambda)) \geq \varepsilon - \mathsf{negl}(\lambda)$, and the claim holds. □

Returning to the proof of Lemma 6.34, we can now write

$$\begin{aligned}\Pr[\mathsf{Hyb}_2(\mathcal{A})=1] &= \Pr[\mathsf{E}_{2,1}]+\Pr[\mathsf{E}_{2,2}] \\ &\le \Pr[\mathsf{E}_{2,1}]+\mathsf{negl}(\lambda) && \text{by Claim 6.36} \\ &\le \Pr[\mathsf{E}_{3,1}]+\mathsf{negl}(\lambda) && \text{by Claim 6.35} \\ &\le \Pr[\mathsf{E}_{3,1}]+\Pr[\mathsf{E}_{3,2}]+\mathsf{negl}(\lambda) \\ &= \Pr[\mathsf{Hyb}_3(\mathcal{A})=1]+\mathsf{negl}(\lambda),\end{aligned}$$

and the lemma holds. □

Lemma 6.37. $\Pr[\mathsf{Hyb}_3(\mathcal{A})=1]=\mathsf{negl}(\lambda)$.

Proof. In order for the output in Hyb_3 to be 1, the adversary $\mathcal{A}$ must output (h,u,σ) such that

$$e(h,B^*)=e(g_1,u) \quad \text{and} \quad e(h,g_2g_3)\neq 1.$$

We claim that this can only happen with negligible probability over the choice of β^*. Specifically, if β^* is non-zero in the p_2 and p_3 subgroups, and $e(h,g_2g_3)\neq 1$, then $e(h,B^*)$ is non-zero in the order p_2p_3 subgroup. However $e(g_1,u)$ is always zero in the order p_2p_3 subgroup, so the verification relation is unsatisfiable. □

By Lemmas 6.30, 6.31 and 6.34, we have that $\Pr[\mathsf{Hyb}_0(\mathcal{A})=1]\le\Pr[\mathsf{Hyb}_3(\mathcal{A})=1]+\mathsf{negl}(\lambda)$ By Lemma 6.37, we have that $\Pr[\mathsf{Hyb}_3(\mathcal{A})=1]\le\mathsf{negl}(\lambda)$. We conclude that $\Pr[\mathsf{Hyb}_0(\mathcal{A})=1]\le\mathsf{negl}(\lambda)$ and extractor validity holds. □

7 Monotone-Policy Aggregate Signatures

In this section, we formalize our construction of monotone policy aggregate signatures from a *non-adaptively* sound monotone policy BARG for NP together with a "puncturable" signature scheme (called an all-but-one signature scheme in [GVW19]).

Definition 7.1 (Puncturable Signature [GVW19, adapted])**.** An puncturable (or all-but-one) signature scheme with message space $\{0,1\}^\lambda$ is a tuple of efficient algorithms $\Pi_{\mathsf{PunctSig}}=(\mathsf{Gen},\mathsf{GenPunc},\mathsf{Sign},\mathsf{Verify})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda)\rightarrow(\mathsf{vk},\mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a key pair $(\mathsf{vk},\mathsf{sk})$.
- $\mathsf{GenPunc}(1^\lambda,m^*)\rightarrow(\mathsf{vk},\mathsf{sk})$: On input a security parameter λ and a message $m^*\in\{0,1\}^\lambda$, the punctured key generation algorithm outputs a key pair $(\mathsf{vk},\mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{sk},m)\rightarrow\sigma$: On input a signing key sk and a message $m\in\{0,1\}^\lambda$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk},m,\sigma)\rightarrow b$: On input a verification key vk, a message $m\in\{0,1\}^\lambda$, and a signature σ, the verification algorithm outputs a bit $b\in\{0,1\}$.

Moreover, the puncturable signature scheme should satisfy the following properties:

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

$$\Pr\left[\mathsf{Verify}(\mathsf{vk},m,\sigma)=1 \;:\; \begin{array}{c}(\mathsf{vk},\mathsf{sk})\leftarrow\mathsf{Gen}(1^\lambda)\\ \sigma\leftarrow\mathsf{Sign}(\mathsf{sk},m)\end{array}\right]=1.$$

- **Punctured correctness:** For all $\lambda\in\mathbb{N}$, all $m^*\in\{0,1\}^\lambda$, and all $\sigma^*\in\{0,1\}^*$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{vk},m^*,\sigma^*)=1 \;:\; (\mathsf{vk},\mathsf{sk})\leftarrow\mathsf{GenPunc}(1^\lambda,m^*)\right]=0.$$

- **Verification key indistinguishability:** For any adversary $\mathcal{A}$ and any $b \in \{0,1\}$, we define the verification key indistinguishability experiment $\mathsf{ExptVKI}_{\mathcal{A}}(\lambda, b)$ as follows:

  1. On input a security parameter λ, the adversary $\mathcal{A}$ outputs a message $m^* \in \{0,1\}^\lambda$ and sends it to the challenger.
  2. The challenger samples $(\mathsf{vk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(\mathsf{vk}_1, \mathsf{sk}_1) \leftarrow \mathsf{GenPunc}(1^\lambda, m^*)$ and gives vk_b to the adversary.
  3. Next, the challenger can make signing queries on messages $m \in \{0,1\}^\lambda \setminus \{m^*\}$. On each signing query, the challenger replies with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_b, m)$.
  4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{PunctSig} satisfies verification key indistinguishability if for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that

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

Remark 7.2 (Multiple Verification Keys)**.** By a standard hybrid argument, we can show that any puncturable signature scheme that satisfies verification key indistinguishability also satisfies a stronger multi-key version of the definition where the adversary can ask for multiple verification keys (punctured at the same message m^*) and signatures on messages $m \neq m^*$ with respect to those keys. We define this formally below:

- **Multiple verification keys indistinguishability:** For any adversary $\mathcal{A}$ and any $b \in \{0,1\}$, we define the multiple verification key indistinguishability experiment $\mathsf{ExptMVKI}_{\mathcal{A}}(\lambda, b)$ as follows:

  1. On input a security parameter λ, the adversary $\mathcal{A}$ outputs the number of challenge keys 1^n together with a message a message $m^* \in \{0,1\}^\lambda$.
  2. For each $i \in [n]$, the challenger samples $(\mathsf{vk}_0^{(i)}, \mathsf{sk}_0^{(i)}) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(\mathsf{vk}_1^{(i)}, \mathsf{sk}_1^{(i)}) \leftarrow \mathsf{GenPunc}(1^\lambda, m^*)$. It gives the verification keys $\mathsf{vk}_b^{(1)}, \ldots, \mathsf{vk}_b^{(n)}$ to the adversary.
  3. The adversary can now make signature queries. Each signing query consists of an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$. The challenger responds with $\sigma \leftarrow \mathsf{Sign}\big(\mathsf{sk}_b^{(i)}, m\big)$.
  4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{PunctSig} satisfies multiple verification key indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

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

Fact 7.3 (Puncturable Signatures [GVW19])**.** Assuming either (1) the plain LWE assumption, or (2) the decision linear assumption in a pairing group, there exists a puncturable signature scheme.

Monotone policy aggregate signature. We now define the notion of a monotone policy aggregate (multi)-signature.

Definition 7.4 (Monotone Policy Aggregate Signatures)**.** Let $\Pi_{\mathsf{Sig}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme with message space $\{0,1\}^\lambda$. A monotone policy aggregation scheme for Π_{Sig} is a tuple of polynomial time algorithms $\Pi_{\mathsf{Agg}} = (\mathsf{Setup}, \mathsf{Aggregate}, \mathsf{AggVerify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^k, 1^{s_p}) \to \mathsf{crs}$: On input a security parameter λ, a bound on the number of signers k, and a bound s_p on the policy size, the setup algorithm outputs a common reference string crs.

- $\mathsf{Aggregate}\big(\mathsf{crs}, m, P, (\mathsf{vk}_1, \sigma_1), \ldots, (\mathsf{vk}_k, \sigma_k)\big) \to \sigma_{\mathsf{agg}}$: On input the common reference string crs, a message $m \in \{0,1\}^\lambda$, a policy circuit $P \colon \{0,1\}^k \to \{0,1\}$, a collection of verification key/signature pairs $(\mathsf{vk}_i, \sigma_i)$, the aggregation algorithm produces an aggregate signature σ_{agg}.

- $\mathsf{AggVerify}\big(\mathsf{crs}, m, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \sigma_{\mathsf{agg}}\big) \to b$: On input the common reference string crs, a message m, a policy circuit $P \colon \{0,1\}^k \to \{0,1\}$, a tuple of k verification keys and an aggregate signature σ_{agg}, the aggregate verification algorithm outputs a bit $b \in \{0,1\}$.

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

- **Correctness:** For all $\lambda, k, s_p \in \mathbb{N}$, all messages $m \in \{0,1\}^\lambda$, all monotone circuits $P \colon \{0,1\}^k \to \{0,1\}$ and all key and signature tuples $\{(i, \mathsf{vk}_i, \sigma_i)\}_{i \in [k]}$ where $P(\mathsf{Verify}(\mathsf{vk}_1, m, \sigma_1), \ldots, \mathsf{Verify}(\mathsf{vk}_k, m, \sigma_k)) = 1$, it holds that

$$\Pr\left[\mathsf{AggVerify}\big(\mathsf{crs}, m, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \sigma_{\mathsf{agg}}\big) = 1 \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^{s_p}) \\ \sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}\big(\mathsf{crs}, m, P, (\mathsf{vk}_1, \sigma_1), \ldots, (\mathsf{vk}_k, \sigma_k)\big) \end{array} \right] = 1.$$

- **Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, k, s_p \in \mathbb{N}$, all messages $m \in \{0,1\}^\lambda$, all monotone circuits $P \colon \{0,1\}^k \to \{0,1\}$ and all pairs $\{(\mathsf{vk}_i, \sigma_i)\}_{i \in [k]}$, the size of the aggregate signature σ_{agg} in the correctness experiment satisfies $|\sigma_{\mathsf{agg}}| = \mathsf{poly}(\lambda + \log |P|)$.
- **Static security:** For any adversary $\mathcal{A}$ define the static unforgeability experiment $\mathsf{ExptSU}_{\mathcal{A}}(\lambda)$ as follows:
  1. On input the security parameter λ, the adversary $\mathcal{A}$ outputs the number of parties 1^k, a number of verification keys 1^n, the bound on the policy size 1^{s_p}, a challenge message $m^* \in \{0,1\}^\lambda$, and a monotone policy $P \colon \{0,1\}^k \to \{0,1\}$.
  2. The challenger samples key-pairs $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ for all $i \in [n]$ and sends $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$ to the adversary.
  3. The adversary $\mathcal{A}$ can now issue signing queries. Each signing query consists of an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$. The challenger responds with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$.
  4. After the adversary is finished making signing queries, it outputs a tuple of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$.
  5. The challenger replies with the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^{s_p})$.
  6. The adversary $\mathcal{A}$ can continue to make signing queries. The challenger responds to these exactly as before.
  7. The adversary outputs the aggregate signature σ_{agg}^*.
  8. The output of the experiment is 1 if all of the following holds:
     - For each $i \in [k]$, let $b_i = 0$ if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$. Otherwise, let $b_i = 1$. Then, it holds that $P(b_1, \ldots, b_k) = 0$.
     - $\mathsf{AggVerify}\big(\mathsf{crs}, m^*, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*\big) = 1$.

     Otherwise, the output is 0.

  We say that Π_{Agg} satisfies static security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathsf{ExptSU}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda)$.

Aggregating puncturable signatures. We now show that we can combine any monotone policy BARG (satisfying non-adaptive soundness) with a puncturable signature scheme to obtain a statically-secure monotone policy aggregate signature scheme.

Construction 7.5 (Monotone Policy Aggregate Signature)**.** Let $\Pi_{\mathsf{BARG}} = (\mathsf{BARG.Gen}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$ be a monotone policy BARG for NP and let $\Pi_{\mathsf{PunctSig}} = (\mathsf{PS.Gen}, \mathsf{PS.GenPunc}, \mathsf{PS.Sign}, \mathsf{PS.Verify})$ be a puncturable signature scheme with message space $\{0,1\}^\lambda$. Let $\ell_{\mathsf{vk}} = \ell_{\mathsf{vk}}(\lambda)$ be a bound on the length of the verification keys of Π_{PunctSig}. For any message $m \in \{0,1\}^\lambda$, define the binary relation $\mathcal{R}[m]$ where

$$\mathcal{R}[m](\mathsf{vk}, \sigma) = \begin{cases} 1 & \mathsf{PS.Verify}(\mathsf{vk}, m, \sigma) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Let C_m be the Boolean circuit that computes the relation $\mathcal{R}[m]$, and let $s_c = s_c(\lambda)$ be a bound on the size of C_m. We construct a monotone aggregate scheme $\Pi_{\mathsf{Agg}} = (\mathsf{Setup}, \mathsf{Aggregate}, \mathsf{AggVerify})$ for Π_{PunctSig} as follows:

- $\mathsf{Setup}(1^\lambda, 1^k, 1^{s_p})$: Sample $\mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p})$ and output crs.
- $\mathsf{Aggregate}\big(\mathsf{crs}, m, P, (\mathsf{vk}_1, \sigma_1), \ldots, (\mathsf{vk}_k, \sigma_k)\big)$: Output $\mathsf{BARG.Prove}(\mathsf{crs}, C_m, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), (\sigma_1, \ldots, \sigma_k))$.
- $\mathsf{AggVerify}\big(\mathsf{crs}, m, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \sigma_{\mathsf{agg}}\big)$: Output $\mathsf{BARG.Verify}(\mathsf{crs}, C_m, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \sigma_{\mathsf{agg}})$.

Theorem 7.6 (Correctness)**.** *If Π_{BARG} is complete, then Construction 7.5 is correct.*

Proof. Fix $\lambda, k, s_p \in \mathbb{N}$, message $m \in \{0,1\}^\lambda$, a monotone policy $P\colon \{0,1\}^k \to \{0,1\}$ and k tuples $\{(i, \mathsf{vk}_i, \sigma_i)\}_{i \in [k]}$ such that $P(\mathsf{Verify}(\mathsf{vk}_1, m, \sigma_1), \ldots, \mathsf{Verify}(\mathsf{vk}_k, m, \sigma_k)) = 1$. By construction of C_m, it holds that $C_m(\mathsf{vk}_i, \sigma_i) = \mathsf{Verify}(\mathsf{vk}_i, m, \sigma_i)$ for all $i \in [k]$. Thus, $P(C_m(\mathsf{vk}_1, \sigma_1), \ldots, C_m(\mathsf{vk}_k, \sigma_k)) = 1$. The theorem now follows by completeness of Π_{BARG}. □

Theorem 7.7 (Succinctness)**.** *If Π_{BARG} is succinct then Construction 7.5 has succinct aggregate signatures.*

Proof. This follows directly from the succinctness of Π_{BARG} and the fact that the aggregate signature is simply a BARG proof. Fix $\lambda, k, s_p \in \mathbb{N}$, message $m \in \{0,1\}^\lambda$, a monotone policy $P\colon \{0,1\}^k \to \{0,1\}$ and k tuples $\{(i, \mathsf{vk}_i, \sigma_i)\}_{i \in [k]}$. The aggregate signature σ_{agg} is a BARG proof for circuit C_m, policy P, the statements $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$ and the signatures $(\sigma_1, \ldots, \sigma_k)$. By succinctness of the BARG, the length of σ_{agg} is $\mathsf{poly}(\lambda + s_c + \log |P|)$. For every message $m \in \{0,1\}^\lambda$, the circuit C_m simply checks whether the input verification key and signature verify the message m, so $s_c(\lambda) = \mathsf{poly}(\lambda)$. Hence, the overall proof size is $\mathsf{poly}(\lambda + \log |P|)$ and the claim follows. □

Remark 7.8 (Fast Verification via RAM Delegation)**.** Similar to Remark 2.13 it is possible to use a RAM delegation scheme [CJJ21b, WW22, KLVW23, CGJ+23] to delegate the aggregate signature verification to the aggregator. Currently, the aggregate verification algorithm AggVerify in Construction 7.5 runs in time $\mathsf{poly}(\lambda + |P|)$. This is because the aggregation algorithm needs to read the policy as well as the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$. If the policy P and the verification keys are known in advance, the aggregator can include a proof π that the function $F_{\mathsf{crs}, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)}(m, \sigma_{\mathsf{agg}}) \coloneqq \mathsf{AggVerify}(\mathsf{crs}, m, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \sigma_{\mathsf{agg}})$ satisfies $F_{\mathsf{crs}, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)}(m, \sigma_{\mathsf{agg}}) = 1$. In this case, the common reference string would also contain a CRS for the RAM delegation scheme. The new aggregate verification algorithm would only check the RAM delegation proof (with respect to the function $F_{\mathsf{crs}, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)}$); formally, the RAM delegation scheme would take as input a *hash* h of the parameters $(\mathsf{crs}, P, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$, and the verification algorithm for the RAM program only needs to take the (honestly-precomputed) hash h, the message m, and the signature σ_{agg}. With this modification, the aggregate verification algortihm (given the precomputed hash h) runs in time $\mathsf{poly}(\lambda + \log |P|)$.

Theorem 7.9 (Static Security)**.** *If Π_{PunctSig} satisfies (multiple) verification key indistinguishability and Π_{BARG} satisfies non-adaptive soundness, then Construction 7.5 is statically unforgeable.*

Proof. Let $\mathcal{A}$ be any efficient adversary for the static security game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is the static unforgeability experiment:
  1. On input the security parameter λ, the adversary $\mathcal{A}$ outputs the number of parties 1^k, a number of verification keys 1^n, a bound on the policy size 1^{s_p}, a challenge message $m^* \in \{0,1\}^\lambda$, and a monotone policy $P\colon \{0,1\}^k \to \{0,1\}$.
  2. The challenger samples key-pairs $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PS.Gen}(1^\lambda)$ for all $i \in [n]$ and sends $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$ to the adversary.
  3. The adversary $\mathcal{A}$ can now issue signing queries. Each signing query consists of an index $i \in \mathcal{I}$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$. The challenger responds with $\sigma \leftarrow \mathsf{PS.Sign}(\mathsf{sk}_i, m)$.
  4. After the adversary is finished making signing queries, it outputs a tuple of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$.
  5. The challenger replies with the common reference string $\mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p})$.
  6. The adversary $\mathcal{A}$ can continue to make signing queries. The challenger responds to these exactly as before. When $\mathcal{A}$ finishes making signing queries, the adversary outputs the aggregate signature σ_{agg}^*.

7. The output of the experiment is 1 if all of the following holds:
   - For each $i \in [k]$, let $b_i = 0$ if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$. Otherwise, let $b_i = 1$. Then, it holds that $P(b_1, \ldots, b_k) = 0$.
   - $\mathsf{BARG.Verify}\big(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*\big) = 1$.

   Otherwise, the output is 0.

- Hyb_1: Same as Hyb_0, except the challenger uses the following modified procedure to sample key-pairs (Step 2):
  - For all $i \in [n]$, sample $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PS.GenPunc}(1^\lambda, m^*)$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of Hyb_i with adversary $\mathcal{A}$. We now show that the output distributions of Hyb_0 and Hyb_1 are computationally indistinguishable, and moreover, that for all efficient adversaries $\mathcal{A}$, the output of $\mathsf{Hyb}_1(\mathcal{A})$ is 1 with negligible probability.

Lemma 7.10. *If* Π_{PunctSig} *satisfies verification key indistinguishability, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| = \varepsilon$ for some non-negligible ε. We construct an adversary $\mathcal{B}$ for the multiple verification key indistinguishability game of Π_{PunctSig} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ computes $(1^k, 1^n, 1^{s_P}, m^*, P) \leftarrow \mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ forwards m^* and 1^n, to the challenger.

2. The challenger replies with a tuple of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. Algorithm $\mathcal{B}$ forwards $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ to $\mathcal{A}$.

3. Whenever $\mathcal{A}$ makes a signing query on an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$, algorithm $\mathcal{B}$ forwards (i, m) to the challenger to obtain a signature σ. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with σ.

4. When the adversary $\mathcal{A}$ outputs a tuple $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$, algorithm $\mathcal{B}$ computes $\mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_C}, 1^{s_P})$ and gives crs to $\mathcal{A}$.

5. Whenever algorithm $\mathcal{A}$ makes additional signing queries, algorithm $\mathcal{B}$ responds in the same manner as before.

6. When $\mathcal{A}$ outputs a signature σ_{agg}^*, algorithm $\mathcal{B}$ checks the following:
   - For each $i \in [k]$, let $b_i = 0$ if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$. Otherwise, let $b_i = 1$. Then, check that $P(b_1, \ldots, b_k) = 0$.
   - $\mathsf{AggVerify}\big(\mathsf{crs}, m^*, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*\big) = 1$.

   Algorithm $\mathcal{B}$ outputs 1 if both checks pass and 0 otherwise.

By construction, algorithm $\mathcal{B}$ constructs the key-pairs $(\mathsf{vk}_i, \mathsf{sk}_i)$ for $i \in [k] \setminus \mathcal{I}$ exactly as required in Hyb_0 and Hyb_1. It suffices to consider the distribution of the verification keys vk_i for $i \in \mathcal{I}$ and the responses to the signing queries. We consider the two possibilities:

- If the challenger responds according to the specification of $\mathsf{ExptMVKI}_{\mathcal{B}}(\lambda, 0)$, then it samples $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$. Moreover, the challenger responds to a signing query on (j, m) where $j \in [n]$ and $m \in \{0,1\}^\lambda \setminus \{m^*\}$ with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$. This is precisely the distribution in $\mathsf{Hyb}_0(\mathcal{A})$. Finally, algorithm $\mathcal{B}$ computes the output using the same procedure as in Hyb_0 and Hyb_1. Therefore, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{ExptMVKI}_{\mathcal{B}}(\lambda, 0) = 1]$.

- If the challenger responds according to the specification of $\mathsf{ExptMVKI}_{\mathcal{B}}(\lambda, 1)$, then it samples $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{GenPunc}(1^\lambda, m^*)$. Moreover, the challenger responds to a signing query on (j, m) where $j \in [n]$ and $m \in \{0,1\}^\lambda \setminus \{m^*\}$ with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$. This is precisely the distribution in $\mathsf{Hyb}_1(\mathcal{A})$. We conclude that $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \Pr[\mathsf{ExptMVKI}_{\mathcal{B}}(\lambda, 1) = 1]$.

We conclude that algorithm $\mathcal{B}$ wins the multiple verification key indistinguishability game with the same non-negligible advantage ε, and the claim follows. □

Lemma 7.11. *If* Π_{BARG} *satisfies non-adaptive soundness, and* Π_{PunctSig} *satisfies punctured correctness, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Proof. Suppose $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \varepsilon$ for some non-negligible ε. We construct an adversary $\mathcal{B}$ for the non-adaptive soundness game as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $(1^k, 1^n, 1^{s_P}, m^*, P) \leftarrow \mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ then samples $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PS.GenPunc}(1^\lambda, m^*)$ for all $i \in [n]$. It forwards the verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$.

2. Whenever algorithm $\mathcal{A}$ makes a signing query on an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$, algorithm $\mathcal{B}$ replies with a signature $\sigma \leftarrow \mathsf{Sign}(sk_i, m)$.

3. When the adversary $\mathcal{A}$ outputs a tuple $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$, algorithm $\mathcal{B}$ forwards the instance size $1^{\ell_{\mathsf{vk}}}$, the circuit size 1^{s_C}, the monotone policy size bound 1^{s_P}, the circuit C_{m^*}, the monotone policy P, and the instance $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$ to the BARG challenger. The challenger replies with a common reference string crs which $\mathcal{B}$ forwards to $\mathcal{A}$.

4. Whenever algorithm $\mathcal{A}$ makes additional signing queries, algorithm $\mathcal{B}$ responds in the same manner as before.

5. At the end of the game, algorithm $\mathcal{A}$ outputs an aggregate signature σ_{agg}. Algorithm $\mathcal{B}$ forwards $\pi = \sigma_{\mathsf{agg}}$ to the challenger.

The challenger constructs the common reference string as $\mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_C}, 1^{s_P})$. Thus, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1 for $\mathcal{A}$. Thus, with probability at least ε, algorithm $\mathcal{A}$ outputs an aggregate signature σ_{agg} where

$$\mathsf{BARG.Verify}(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}) = 1,$$

and $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$ and $b_i = 1$ otherwise. We argue that algorithm $\mathcal{B}$ wins the non-adaptive soundness game when this happens:

- By punctured correctness of Π_{PunctSig}, for all $\sigma \in \{0,1\}^*$, it holds that $\mathsf{Verify}(\mathsf{vk}_i, m^*, \sigma) = 0$ for all $i \in [n]$. Correspondingly, this means that for all $i \in [n]$, it holds that $C_{m^*}(\mathsf{vk}_i, \sigma) = 0$ for all inputs $\sigma \in \{0,1\}^*$.

- Thus for all $i \in [k]$, if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$, then $C_{m^*}(\mathsf{vk}_j^*, \sigma) = 0$ for all $\sigma \in \{0,1\}^*$. Next $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ whenever $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$, and $b_i = 1$ otherwise. This means $(C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)) \notin \mathcal{L}_{\mathsf{MP\text{-}CSAT}}$.

- If $\mathsf{BARG.Verify}(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}) = 1$, and $(C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)) \notin \mathcal{L}_{\mathsf{MP\text{-}CSAT}}$, then algorithm $\mathcal{B}$ wins the non-adaptive soundness game.

Thus, algorithm $\mathcal{B}$ breaks the non-adaptive soundness of Π_{BARG} with the same advantage ε. □

Theorem 7.9 now follows by Lemmas 7.10 and 7.11 and a hybrid argument. □

8 Semi-Somewhere Extractability of Monotone Policy BARGs

In this section, we show that our proof of non-adaptive soundness for our monotone policy BARG in Section 4 easily extends to achieve a notion of *extractability.*[10] Our notion of extractability is a relaxed version of the somewhere extractability notion from [BBK+23]. In the notion from [BBK+23], there is a trapdoor setup algorithm that takes as input a set of indices S and outputs an extraction trapdoor. The guarantee is that whenever the prover produces a proof for a tuple of statements $(x_1, \ldots, x_k)$ with respect to a circuit C and policy P for which S is "critical," then

[10] As we discussed in Section 1.1, it is not clear what the right or most useful notion of extraction is in the context of monotone policy BARGs. The desired notion of extractability may in fact be application-dependent. For this reason, we focus on non-adaptive soundness for the main construction and include this section primarily as an illustration that our approach can support some non-trivial form of extractability.

the extraction algorithm will output a witness w_i where $C(x_i, w_i) = 1$ for some index $i \in S$. In this setting, a set S is critical for a policy P if every input $(b_1, \ldots, b_k) \in \{0,1\}^k$ where $P(b_1, \ldots, b_k) = 1$ has an index $i \in S$ where $b_i = 1$ (i.e., every input that satisfies the policy P must set some index in the critical set S to 1). In addition, the trapdoor CRS should hide the set S.

Semi-somewhere extractability. To extract a witness from the critical set S, the [BBK+23] construction program S into the CRS and then rely on an FHE-based hash function to homomorphically "propagate" one of the witnesses in S into the hash digest. This enables an efficient extraction procedure. In our setting, we do not use FHE. Instead, we observe that our existing proof in Section 4 already achieves a notion of extractability by relying only on somewhere extractability of the underlying (vanilla) BARG. The caveat of our notion is that there is a $1/k$ loss in the success probability of our extractor. Namely, if an adversary produces a proof on $(x_1, \ldots, x_k)$ with probability ε, then the extractor will output a witness w_i for some $i \in S$ with probability ε/k. We refer to our notion as semi-somewhere extractability. We give the formal definition below:

Definition 8.1 (Semi-Somewhere Extractable Monotone BARG)**.** A semi-somewhere extractable monotone policy BARG for Boolean circuit satisfiability is a tuple of polynomial time algorithms $\Pi_{\mathsf{MP\text{-}BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{TrapGen}, \mathsf{Extract})$ such that $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is monotone policy BARG for Boolean circuit satisfiability and the two additional algorithms $(\mathsf{TrapGen}, \mathsf{Extract})$ have the following syntax:

- $\mathsf{TrapGen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the instance size $n \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$, a bound on the size of the Boolean circuit $s_c \in \mathbb{N}$, a bound on the size of the policy $s_p \in \mathbb{N}$, and a subset $S \subseteq [k]$, the indexed generator algorithm outputs a common reference string crs and a trapdoor td.
- $\mathsf{Extract}(\mathsf{td}, C, P, (x_1, \ldots, x_k), \pi) \to (i, w_i)$: On input a trapdoor td, a Boolean circuit C, a monotone policy P, instances $x_1, \ldots, x_k$, a proof π, and an index i, the extraction algorithm outputs an index i and an NP witness w_i.

Moreover, $\Pi_{\mathsf{MP\text{-}BARG}}$ should satisfy the following properties:

- **Set hiding:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, define the set hiding experiment $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, b)$ as follows:
  1. On input a security parameter λ, algorithm $\mathcal{A}$ starts by outputting the instance size 1^n, the bound on the size of the NP relation 1^{s_c}, the bound on the size of the policy 1^{s_p}, the number of instances 1^k, and a set $S \subseteq [k]$.
  2. The challenger samples $\mathsf{crs}_0 \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$ and $(\mathsf{crs}_1, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S)$ and sends crs_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 $\Pi_{\mathsf{MP\text{-}BARG}}$ satisfies set hiding if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that
  $$\left|\Pr[\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 1) = 1]\right| = \mathsf{negl}(\lambda).$$
- **Semi-somewhere extractability:** For an integer $k \in \mathbb{N}$ and an adversary $\mathcal{A}$, define the semi-somewhere extractability experiment $\mathsf{ExptSE}_{\mathcal{A}}(\lambda, k)$ as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the instance size 1^n, the bound on the size of the NP relation 1^{s_c}, the bound on the size of the policy 1^{s_p}, a monotone Boolean circuit $P \colon \{0,1\}^k \to \{0,1\}$ of size at most s_p, and a set $S \subseteq [k]$.
  2. The challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S)$ and sends crs to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s_c, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof π.
  4. The challenger extracts a witness $(i, w_i) \leftarrow \mathsf{Extract}(\mathsf{td}, C, P, (x_1, \ldots, x_k), \pi)$.

5. The output of the experiment is 1 if $C(x_i, w_i) = 1$ and $i \in S$. Otherwise, the output is 0.

An adversary $\mathcal{A}$ is admissible if it outputs a set $S \subseteq [k]$ and a policy P such that $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise. Let

$$\varepsilon_{\mathcal{A}}(\lambda, k) \coloneqq \Pr[\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), \pi) = 1] \tag{8.1}$$

in an execution of $\mathsf{ExptSE}_{\mathcal{A}}(\lambda, k)$. We say that Π_{BARG} is semi-somewhere extractable if for every polynomial $k = k(\lambda)$ and every efficient and admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{ExptSE}_{\mathcal{A}}(\lambda, k) = 1] \geq \frac{1}{k} \cdot \varepsilon_{\mathcal{A}}(\lambda, k) - \mathsf{negl}(\lambda).$$

Remark 8.2 (On Semi-Somewhere Extractability)**.** An important caveat of the semi-somewhere extractability notion in Definition 8.1 is that we allow the extractor to succeed with *smaller* probability (by an inverse polynomial factor) than the honest prover. While this is still sufficient for applications to monotone policy aggregate signatures (see Section 8.1), this may not be the case in all settings where an extraction guarantee might be employed. As an example, suppose we have an adversary $\mathcal{A}$ that samples statements from one of two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ (with equal probability) and produces a valid proof on the statement with probability ε. Normally, we would hope that the extractor algorithm would be able to extract witnesses for statements sampled from *both* $\mathcal{D}_1$ and $\mathcal{D}_2$. However, since we allow for an inverse polynomial loss in the extractor's success probability, it could be the case that the extractor *only* works for instances sampled from $\mathcal{D}_1$ and never outputs witnesses for instances sampled from $\mathcal{D}_2$. If this were to happen in a security proof which relies on the ability to extract witnesses from instances drawn from $\mathcal{D}_2$, then the proof would no longer go through. Thus, using the semi-somewhere extractability notion in the context of a security proof could require some extra care.

Adapting Construction 4.4. We now show how to extend Construction 4.4 to support semi-somewhere extractability. The construction relies on the fact that the proof of Construction 4.4 (Section 4.2) implicitly achieves a notion of extractability.

Construction 8.3 (Semi-Somewhere Extractable Monotone BARG)**.** Let $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ be the monotone policy BARG of Construction 4.4. Let $\Pi'_{\mathsf{BARG}} = (\mathsf{Gen}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{TrapGen}', \mathsf{Extract}')$ be the underlying somewhere extractable BARG for Boolean circuit satisfiability, and $\Pi_{\mathsf{H}} = (\mathsf{H.Setup}, \mathsf{H.Hash}, \mathsf{H.ProveOpen}, \mathsf{H.VerOpen}, \mathsf{H.Extract}, \mathsf{H.ValidateDigest})$ be the underlying zero-fixing hash function. We extend Construction 4.4 with the following algorithms:

- $\mathsf{TrapGenIndex}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S, j) \to (\mathsf{crs}, \mathsf{td}_j)$: On input the security parameter $\lambda \in \mathbb{N}$, the instance size $n \in \mathbb{N}$, a bound on the size of the Boolean circuit $s_c \in \mathbb{N}$, a bound on the size of the policy $s_p \in \mathbb{N}$, the number of instances k, a set $S \subseteq [k]$, and an index $j \in S$, the indexed trapdoor generator algorithm proceeds as follows:
  - Let $j_1, \ldots, j_{|S|} \in S$ be the elements of S in ascending order. Let $t \in [|S|]$ be the index where $j = j_t$. Define the set $S_t = \{j_1, \ldots, j_{t-1}\}$ if $t > 1$ and $S_t = \varnothing$ otherwise.
  - Sample two hash keys $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$ and $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, S_t)$.
  - Let s' be a bound on the size of the circuit that computes the relation $\mathcal{R}[C, k, s_p, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{dig}_0, \mathsf{dig}_1]$ from Fig. 1 when instantiated with an arbitrary Boolean circuit C of size at most s_c, an input length $k \leq s_p$ and digests $\mathsf{dig}_0, \mathsf{dig}_1$ associated with the hash and verification keys $(\mathsf{hk}_0, \mathsf{vk}_0)$ and $(\mathsf{hk}_1, \mathsf{vk}_1)$. Let $n' = 3 \cdot \lceil \log s_p \rceil + 1$ be the bound on the statement length. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, j)$.
  - Outputs the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and the trapdoor $\mathsf{td}_j = \mathsf{td}_{\mathsf{BARG}}$.

  Looking ahead, the helper algorithm $\mathsf{TrapGenIndex}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S, j_t)$ implements the setup algorithm according to the specification of the hybrid experiments $\widetilde{\mathsf{Hyb}}_{0,t,3}$ and $\widetilde{\mathsf{Hyb}}_{0,t,4}$ in the proof of Theorem 8.7. These are the analogs of the hybrid experiments $\mathsf{Hyb}_{0,t,3}$ and $\mathsf{Hyb}_{0,t,4}$ from the proof of Theorem 4.7 in Section 4.2.

- $\mathsf{TrapGen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the instance size $n \in \mathbb{N}$, a bound on the size of the Boolean circuit $s_c \in \mathbb{N}$, a bound on the size of the policy $s_p \in \mathbb{N}$, the number of instances k, and a set $S \subseteq [k]$, the generator algorithm proceeds as follows:
  - Sample a random $j \xleftarrow{R} S$.
  - Compute $(\mathsf{crs}, \mathsf{td}_j) \leftarrow \mathsf{TrapGenIndex}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, j)$.
  - Output the common reference string crs, and the trapdoor $\mathsf{td} = (j, \mathsf{td}_j)$.
- $\mathsf{Extract}(\mathsf{td}, C, P, (x_1, \ldots, x_k), \pi) \to (j, w_j)$: On input a trapdoor $\mathsf{td} = (j, \mathsf{td}_j)$, a Boolean circuit C, a monotone policy P, instances $x_1, \ldots, x_k$, and a proof π, the algorithm computes $\hat{w}_j \leftarrow \mathsf{Extract}(\mathsf{td}_j, C_{\mathsf{aug}}, (\hat{x}_1, \ldots, \hat{x}_{s_p}), \pi, j)$, where the circuit C_{aug} and the instances $\hat{x}_1, \ldots, \hat{x}_{s_p}$ are computed from $C, P, x_1, \ldots, x_k$ as in Construction 4.4. Parse $\hat{w}_j = (b, \sigma^{(0)}, \sigma^{(1)}, w)$ and if $b = 1$, output (j, w). If $b \neq 1$, output $\perp$.

Theorem 8.4 (Set Hiding). *If Π_{BARG} satisfies index hiding and Π_{H} satisfies set hiding, then Construction 8.3 satisfies set hiding.*

Proof. Let $\mathcal{A}$ be an efficient non-uniform adversary for the set hiding game of $\Pi_{\mathsf{MP\text{-}BARG}}$. We proceed via a hybrid argument:

- Hyb_0: This is experiment $\mathsf{ExptSH}_{\mathcal{A}}[\lambda, 0]$:
  1. On input a security parameter λ, algorithm $\mathcal{A}$ starts by outputting the instance size 1^n, the bound on the size of the NP relation 1^{s_c}, the bound on the size of the policy 1^{s_p}, the number of instances 1^k, and a set $S \subseteq [k]$.
  2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$. Namely, it samples
     - $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.
     - $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.
     - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$, where n', s' are defined as in Construction 8.3.

     The challenger sends $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ which is the output of the experiment.
- Hyb_1: Same as Hyb_0, except the challenger samples $\mathsf{crs}_{\mathsf{BARG}}$ to bind to a random index $j_t \in S$. Concretely, the challenger samples a random index $j \xleftarrow{R} S$ and samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, j)$.
- Hyb_2: Same as Hyb_1, except the challenger samples $\mathsf{hk}_1, \mathsf{vk}_1$ to be zero-fixing on the set S_t. This is $\mathsf{ExptSH}_{\mathcal{A}}[\lambda, 1]$. Specifically, in this experiment, the challenger samples $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, S_t)$, where $S_t = \{j_1, \ldots, j_{t-1}\}$, the indices $j_1, \ldots, j_{|S|}$ are the elements of S in ascending order, and $t \in [|S|]$ is the index where $j = j_t$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of Hyb_i with adversary $\mathcal{A}$. We now show that each pair of adjacent output distributions are computationally indistinguishable.

Claim 8.5. *If Π_{BARG} satisfies index hiding, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the index hiding game of Π_{BARG} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain $(1^n, 1^{s_c}, 1^{s_p}, 1^k, S)$.
2. Algorithm $\mathcal{B}$ samples $j \xleftarrow{R} S$ and send $(1^{s_p}, 1^{n'}, 1^{s'}, j)$ to the BARG challenger. The challenger replies with $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}})$.

3. Algorithm $\mathcal{B}$ computes $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$ and $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.

4. Algorithm $\mathcal{B}$ sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and gives crs to $\mathcal{A}$. It outputs whatever $\mathcal{A}$ outputs.

If the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_0 for $\mathcal{A}$. Conversely, if it samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, j)$, algorithm $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$. We conclude that the advantage of algorithm $\mathcal{B}$ is ε, and the claim holds. □

Claim 8.6. *If Π_{H} satisfies set hiding, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \varepsilon$ for some non-negligible ε. We construct an attacker $\mathcal{B}$ for the set hiding game of Π_{H} as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain $(1^n, 1^{s_c}, 1^{s_p}, 1^k, S)$.

2. Let $j_1, \ldots, j_{|S|} \in S$ be the elements of S in ascending order. Algorithm $\mathcal{B}$ samples $j \xleftarrow{\mathrm{R}} S$ and sets $t \in [|S|]$ to be the index where $j = j_t$. Algorithm $\mathcal{B}$ send $(1^{s_p}, S_t)$ to the challenger, where $S_t = \{j_1, \ldots, j_{t-1}\}$ if $t > 1$ and $S_t = \varnothing$ otherwise. The challenger replies with $(\mathsf{hk}_1, \mathsf{vk}_1)$.

3. Algorithm $\mathcal{B}$ samples a hash key $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$ along with the BARG parameters $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, j)$.

4. Algorithm $\mathcal{B}$ sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and gives crs to $\mathcal{A}$. It outputs whatever $\mathcal{A}$ outputs.

If the challenger samples $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$. Conversely, if the challenger samples $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, S_t)$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_2 for $\mathcal{A}$. Correspondingly, the advantage of algorithm $\mathcal{B}$ is ε, and the claim follows. □

Theorem 8.4 now follows from Claims 8.5 and 8.6. □

Theorem 8.7. *If Π_{H} satisfies set hiding, index hiding with extracted guess, zero fixing and extractor validity against non-uniform adversaries, and Π_{BARG} is somewhere extractable and satisfies set hiding against non-uniform adversaries, then Construction 8.3 is semi-somewhere extractable against non-uniform adversaries.*

Proof. To prove Theorem 8.7, we use a similar strategy as in the proof of Theorem 4.7 (Section 4.2). Here, we give a high-level overview. Specifically, we start by defining sequence of hybrids $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_d$, where d is the depth of the monotone circuit P. These are essentially the same experiments from the proof of Theorem 4.7 in Section 4.2. The initial hybrid corresponds to the semi-somewhere extractability experiment where the output is 1 if the adversary outputs an accepting proof (i.e., the output in the initial hybrid is 1 with probability $\varepsilon_{\mathcal{A}}(\lambda, k)$ as defined in Eq. (8.1)). In the final hybrid, we show that the output is 1 probability 0. Finally, we argue that any difference in advantage between adjacent hybrids can *only* occur in settings where the extractor is successful. There are a maximum of k such experiments (one associated with each of the inputs to P). Since the probability of an experiments drops from ε to 0, in at least one of these intermediary experiments, the probability must decrease by ε/k (up to negligible differences); this directly translates into the extractor succeeding with probability at least ε/k (up to negligible differences). We give the formal argument below.

Outer hybrids. Take any polynomial $k = k(\lambda)$ and any efficient (non-uniform) and admissible adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Let $P \colon \{0,1\}^k \to \{0,1\}$ and $S \subseteq [k]$ be the monotone policy and the challenge set that algorithm $\mathcal{A}_1$ outputs (on input the security parameter λ). Let d be the depth of P and s be its size. We now define the sequence of outer hybrids:

- $\widetilde{\mathsf{Hyb}}_0$: This is the analog of Hyb_0 from the proof of Theorem 4.7 (Section 4.2). We define it here:

- **Phase 1:** On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs $1^n, 1^{s_c}, 1^{s_p}$, a monotone Boolean circuit $P\colon \{0,1\}^k \to \{0,1\}$ of size $s \le s_p$, a set $S \subseteq [k]$ and the state $\mathsf{st}_{\mathcal{A}}$. The experiment outputs 0 if $P(b_1, \ldots, b_k) = 1$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise.
- **Phase 2:** The challenger computes $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p})$. Specifically, the challenger samples the following components:
  - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
  - $(\mathsf{hk}_0, \mathsf{vk}_0, \mathsf{td}_0) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.
  - $(\mathsf{hk}_1, \mathsf{vk}_1, \mathsf{td}_1) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a Boolean circuit C of size at most s_c, an instance $\mathbf{x} = (x_1, \ldots, x_k)$, and a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
  - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.

- $\widetilde{\mathsf{Hyb}}_i$ for $i \in [d]$: Same as Hyb_0, but $\mathsf{hk}_{\mathsf{low}}$ binds on J_i, where $\mathsf{low} = i \bmod 2$ and $\mathsf{high} = 1 - \mathsf{low}$. This is the analog of Hyb_1 from the proof of Theorem 4.7 (Section 4.2). Specifically, the game proceeds as follows:
  - **Phase 1:** On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs $1^n, 1^{s_c}, 1^{s_p}$, a monotone Boolean circuit $P\colon \{0,1\}^k \to \{0,1\}$ of size $s \le s_p$, a set $S \subseteq [k]$ and the state $\mathsf{st}_{\mathcal{A}}$. The experiment outputs 0 if $P(b_1, \ldots, b_k) = 1$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise. The challenger then computes the following quantities:
    - For each $j \in [k]$, let $\beta_j =$ if $j \in S$ and $\beta_j = 1$ otherwise.
    - For $j \in [k+1, s]$ let β_j be the value of the wire j in the evaluation of P on input $(\beta_1, \ldots, \beta_k)$.
    - For each layer $\ell \in [d]$, let $J_\ell = \left\{ j \in \mathsf{layer}_\ell(P) : \beta_j = 0 \right\}$.
  - **Phase 2:** The challenger samples the following components:
    - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{vk}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'})$.
    - $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_i)$.
    - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.

    The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The output of the experiment is 1 if all of the following conditions hold (and 0 otherwise):
    - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
    - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
    - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{low}}, \mathsf{dig}_{\mathsf{low}}) = \mathsf{Matching}$.

Inner games. To argue that each pair of outer hybrids is computationally indistinguishable, we define a sequence of "inner hybrids" exactly as in the proof of Theorem 4.7 in Section 4.2. Specifically, for each $i \in \{0, \ldots, d\}$, each $t \in [|J_i|]$ and each $\ell \in \{1, \ldots, 7\}$, we define $\widetilde{\mathsf{Hyb}}_{i,t,\ell}$ as follows:

- **Phase 1:** Same as $\widetilde{\mathsf{Hyb}}_i$. Note that algorithm $\mathcal{A}_1$ does not output the Boolean circuit C or the statements $\mathbf{x}$ in this phase.
- **Phase 2:** Same as in $\mathsf{Hyb}_{i,t,\ell}$ from Section 4.2, *except* the adversary additionally outputs the Boolean circuit C and the instances $\mathbf{x}$ in this phase (as in $\widetilde{\mathsf{Hyb}}_i$).

We now analyze each pair of hybrid experiments. With the exception of one of the transitions (from $\widetilde{\mathsf{Hyb}}_{0,t,3}$ to $\widetilde{\mathsf{Hyb}}_{0,t,4}$), each transition follows by a similar argument as the corresponding transition in the proof of Theorem 4.7.

Claim 8.8. *If* Π_{H} *satisfies extractor validity against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$*, it holds that*

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

Proof. Follows by a similar argument as the proof of Claim 4.8. □

Claim 8.9. *If* Π_{BARG} *satisfies set hiding against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$ *and* $t \in [|J_{i+1}|]$*, it holds that*

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

Proof. Follows by a similar argument as the proof of Claim 4.9. □

Claim 8.10. *If* Π_{BARG} *satisfies somewhere extractability in trapdoor mode against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$*,* $t \in [|J_{i+1}|]$*, it holds that*

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

Proof. Follows by a similar argument as the proof of Claim 4.10. □

Extracting valid witnesses. As mentioned above the transition from $\widetilde{\mathsf{Hyb}}_{0,t,3}$ to $\widetilde{\mathsf{Hyb}}_{0,t,4}$ diverges from the corresponding analysis (Claim 4.11) in the proof of Theorem 4.7. In Claim 4.11, the relevant statement $x_{J_1[t]}$ was false, and thus, by somewhere extractability of the BARG, we were able to argue that the outputs of $\mathsf{Hyb}_{0,t,3}$ to $\mathsf{Hyb}_{0,t,4}$ could only change by a negligible amount. Upon closer inspection, the proof of Claim 4.11 actually shows a *stronger* property: the difference between these two hybrids is exactly equal to the probability of extracting a valid witness for the instance $x_{J_1[t]}$. In the case of Claim 4.11, the statement $x_{J_1[t]}$ was false, so this probability was identically 0. In the somewhere extractability game, this probability could be noticeable. But that means our extractor succeeds with noticeable probability. To formalize this, we start with a full specification of $\widetilde{\mathsf{Hyb}}_{0,t,3}$:

- **Phase 1:** Same as $\widetilde{\mathsf{Hyb}}_i$.
- **Phase 2:** The challenger samples the following components.
  - $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^{s_p}, 1^{n'}, 1^{s'}, J_1[t])$.
  - $(\mathsf{hk}_{\mathsf{low}}, \mathsf{vk}_{\mathsf{low}}, \mathsf{td}_{\mathsf{low}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, \varnothing)$.
  - $(\mathsf{hk}_{\mathsf{high}}, \mathsf{vk}_{\mathsf{high}}, \mathsf{td}_{\mathsf{high}}) \leftarrow \mathsf{H.Setup}(1^\lambda, 1^{s_p}, J_1[1, \ldots, t-1])$.

  The challenger sets $\mathsf{crs} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_0, \mathsf{hk}_1, \mathsf{vk}_0, \mathsf{vk}_1)$ and runs $\mathcal{A}_2$ on input $(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{A}_2$ outputs a proof string $\pi = (\mathsf{dig}_0, \mathsf{dig}_1, \pi_{\mathsf{BARG}})$. Let $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_{s_p})$ and C_{aug} be as defined in Prove and Verify in Construction 4.4. The challenger then computes $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w) \leftarrow \mathsf{Extract}'(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}, J_1[t])$. The output is 1 if the following conditions hold (and 0 otherwise):

  - $\mathsf{H.ValidateDigest}(\mathsf{vk}_0, \mathsf{dig}_0) = \mathsf{H.ValidateDigest}(\mathsf{vk}_1, \mathsf{dig}_1) = 1$.
  - $\mathsf{Verify}'(\mathsf{vk}_{\mathsf{BARG}}, C_{\mathsf{aug}}, \hat{\mathbf{x}}, \pi_{\mathsf{BARG}}) = 1$.
  - $\mathsf{H.Extract}(\mathsf{td}_{\mathsf{high}}, \mathsf{dig}_{\mathsf{high}}) = \mathsf{Matching}$.
  - $C_{\mathsf{aug}}(\hat{x}_{j_t}, \hat{w}) = 1$.

Since $S = J_1$ by construction, the challenger in $\widetilde{\mathsf{Hyb}}_{0,t,3}$ is sampling the common reference string crs according to the specification of $\mathsf{TrapGenIndex}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S, t)$. Similarly, the witness $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w)$ in $\widetilde{\mathsf{Hyb}}_{0,t,3}$ is computed using the same procedure as $\mathsf{Extract}(\mathsf{td}, C, P, (x_1, \ldots, x_k), \pi)$ where $\mathsf{td} = (J_1[t], \mathsf{td}_{\mathsf{BARG}})$. For $t \in [|J_1|]$, let

$$p_t \coloneqq \Pr\left[\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A}) = 1 \land b = 1 \text{ in the execution of } \widetilde{\mathsf{Hyb}}_{0,t,3}\right].$$

By definition of C_{aug}, if $\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A})$ outputs 1 and the extracted bit b satisfies $b = 1$, this means that $C(x_{J_1[t]}, w) = 1$. In this case, $\mathsf{Extract}(\mathsf{td}, C, P, (x_1, \ldots, x_k), \pi)$ outputs w such that $C(x_{J_1[t]}, w) = 1$. In particular, this means that

$$p_t \geq \Pr\left[C(x_{S[t]}, w) = 1 : \begin{array}{c} (1^n, 1^{s_c}, 1^{s_p}, P, S, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda) \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapGenIndex}(1^\lambda, 1^n, 1^{s_c}, 1^{s_p}, 1^k, S, S[t]) \\ (C, (x_1, \ldots, x_k), \pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}}) \\ (t', w) \leftarrow \mathsf{Extract}(\mathsf{td}, C, P, (x_1, \ldots, x_k), \pi). \end{array} \right], \tag{8.2}$$

where we write $S[t]$ to denote the t^{th} value in S in ascending order. We now show that the difference between the outputs of $\widetilde{\mathsf{Hyb}}_{0,t,3}$ and $\widetilde{\mathsf{Hyb}}_{0,t,3}$ is exactly p_t:

Claim 8.11. *It holds that* $\Pr[\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A}) = 1] = \Pr[\widetilde{\mathsf{Hyb}}_{0,t,4}(\mathcal{A}) = 1] + p_t$.

Proof. By construction, the only difference between $\widetilde{\mathsf{Hyb}}_{0,t,3}$ and $\widetilde{\mathsf{Hyb}}_{0,t,4}$ is the *additional* check in $\mathsf{Hyb}_{0,t,4}$ that the extracted bit b satisfies $b = 0$. Let E_0 be the event that $b = 0$ in the execution of $\widetilde{\mathsf{Hyb}}_{0,t,3}$ and E_1 be the event that $b = 1$. Then,

$$\begin{aligned} \Pr[\widetilde{\mathsf{Hyb}}_{0,t,4}(\mathcal{A}) = 1] &= \Pr[\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A}) = 1 \wedge \mathsf{E}_0] \\ &= \Pr[\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A}) = 1] - \Pr[\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A}) = 1 \wedge \mathsf{E}_1] \\ &= \Pr[\widetilde{\mathsf{Hyb}}_{0,t,3}(\mathcal{A}) = 1] - p_t. \end{aligned}$$

The claim follows. □

Claim 8.12. *If* Π_{H} *satisfies zero-fixing against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{1, \ldots, d-1\}, t \in [|J_{i+1}|]$*, it holds that*

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

Proof. Follows by a similar argument as the proof of Claim 4.11 (for the case where $i > 0$). □

Claim 8.13. *If* Π_{H} *satisfies one-sided index hiding with extracted guess security against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}, t \in [|J_{i+1}|]$*, it holds that*

$$\Pr[\widetilde{\mathsf{Hyb}}_{i,t,4}(\mathcal{A}) = 1] \leq \Pr[\widetilde{\mathsf{Hyb}}_{i,t,5}(\mathcal{A}) = 1] + \mathsf{negl}(\lambda).$$

Proof. Follows by a similar argument as the proof of Claim 4.12. □

Claim 8.14. *For every* $i \in \{0, \ldots, d-1\}, t \in [|J_{i+1}|]$*, it holds that*

$$\Pr[\widetilde{\mathsf{Hyb}}_{i,t,6}(\mathcal{A}) = 1] \geq \Pr[\widetilde{\mathsf{Hyb}}_{i,t,5}(\mathcal{A}) = 1].$$

Proof. Follows by a similar argument as the proof of Claim 4.13. □

Claim 8.15. *If* Π_{BARG} *satisfies set hiding against efficient non-uniform adversaries, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $i \in \{0, \ldots, d-1\}$ *and* $t \in [|J_{i+1}|]$*, it holds that*

$$\left| \Pr[\widetilde{\mathsf{Hyb}}_{i,t,6}(\mathcal{A}) = 1] - \Pr[\widetilde{\mathsf{Hyb}}_{i,t,7}(\mathcal{A}) = 1] \right| \leq \mathsf{negl}(\lambda).$$

Proof. Follows by a similar argument as the proof of Claim 4.14. □

Claim 8.16. *For every* $i \in \{0, \ldots, d-1\}$*, it holds that*

$$\Pr[\widetilde{\mathsf{Hyb}}_{i,|J_{i+1}|,7}(\mathcal{A}) = 1] \leq \Pr[\widetilde{\mathsf{Hyb}}_{i,\mathsf{final}}(\mathcal{A}) = 1].$$

Proof. Follows by a similar argument as the proof of Claim 4.15. □

Claim 8.17. *If Π_{H} satisfies set hiding property against efficient non-uniform adversaries then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $i \in \{0, \ldots, d-1\}$, it holds that*

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

Proof. Follows by a similar argument as the proof of Claim 4.16. □

Claim 8.18. *If $\mathcal{A}$ is admissible, then* $\Pr[\widetilde{\mathsf{Hyb}}_{d-1,1,4}(\mathcal{A}) = 1] = 0$.

Proof. The proof is almost identical to that of Claim 4.17, but relies on the fact that $\mathcal{A}$ is an admissible adversary. Namely, if $\mathcal{A}$ is admissible for the semi-somewhere extractability game, then it outputs a set S and a policy P such that $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise. By construction, the challenger in $\widetilde{\mathsf{Hyb}}_{d-1,1,4}(\mathcal{A}) = 1$ sets $\beta_i = b_i$ for all $i \in [k]$, and $\beta_{k+1}, \ldots, \beta_s$ to be the wire values for $P(b_1, \ldots, b_k)$. In particular, this means that $\beta_s = 0$, and therefore $J_d[1] = \{\beta_s\}$. In this case, $\widetilde{\mathsf{Hyb}}_{d-1,1,4}$ cannot always output 1 since the following two conditions must simultaneously hold:

- On the one hand, there must exist a witness $\hat{w} = (b, \sigma^{(0)}, \sigma^{(1)}, w)$ for instance $\hat{x}_s$ (of the relation in Fig. 1) where $b = 0$.
- On the other hand, by definition of instance $\hat{x}_s$, since s is the output wire, it must be that $b = 1$.

Therefore the output in $\widetilde{\mathsf{Hyb}}_{d-1,1,4}$ is always 0. □

Combining Claims 8.8 to 8.18, we conclude that

$$\Pr[\widetilde{\mathsf{Hyb}}_0(\mathcal{A}) = 1] \le \sum_{t \in [|S|]} p_t + \mathsf{negl}(\lambda). \tag{8.3}$$

To complete the proof, we relate the probability $\varepsilon_{\mathcal{A}}(\lambda, k)$ from Eq. (8.1) that $\mathcal{A}$ outputs a valid proof in the semi-somewhere extractability game to the probability that $\widetilde{\mathsf{Hyb}}_0(\mathcal{A})$ outputs 1. The only difference between $\widetilde{\mathsf{Hyb}}_0$ and the semi-somewhere extractability game is the fact that in $\widetilde{\mathsf{Hyb}}_0$, the common reference string is norm (output by Gen) whereas in the semi-somewhere extractability game, it is output by $\mathsf{TrapGen}$. We give the formal reduction below:

Claim 8.19. *If Construction 8.3 satisfies set hiding, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$|\Pr[\widetilde{\mathsf{Hyb}}_0(\mathcal{A}) = 1] - \varepsilon_{\mathcal{A}}(\lambda, k)| \le \mathsf{negl}(\lambda)$$

where $\varepsilon_{\mathcal{A}}$ is the probability that $\mathcal{A}$ outputs a verifying proof in the ExptSE *experiment from Definition 8.1.*

Proof. Suppose $|\Pr[\widetilde{\mathsf{Hyb}}_0(\mathcal{A}) = 1] - \varepsilon_{\mathcal{A}}(\lambda, k)| \ge \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the set hiding game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $(1^n, 1^{s_c}, 1^{s_P}, P, S, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(1^\lambda)$. Algorithm $\mathcal{B}$ then forwards 1^n, 1^{s_c}, 1^{s_P}, $1^{k(\lambda)}$, and S to the challenger.
2. The challenger replies with crs. Algorithm $\mathcal{B}$ runs $(C, (x_1, \ldots, x_k), \pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, \mathsf{st}_{\mathcal{A}})$. Algorithm $\mathcal{B}$ then outputs $\mathsf{Verify}(\mathsf{crs}, C, P, (x_1, \ldots, x_k), \pi)$.

If the challenger samples $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_P})$, then $\mathcal{B}$ perfectly simulates the distribution $\widetilde{\mathsf{Hyb}}_0$ and outputs 1 with probability $\Pr[\widetilde{\mathsf{Hyb}}_0(\mathcal{A})] = 1]$. If the challenger samples $\mathsf{crs} \leftarrow \mathsf{TrapGen}(1^\lambda, 1^n, 1^{s_c}, 1^{s_P}, 1^k, S)$, then $\mathcal{B}$ perfectly simulates the distribution $\mathsf{ExptSE}_{\mathcal{A}}(\lambda, k)$ and outputs 1 with probability $\varepsilon_{\mathcal{A}}(\lambda, k)$. We conclude that algorithm $\mathcal{B}$ succeeds with the same advantage ε. □

Combining Eq. (8.3) with Claim 8.19, we have that

$$\varepsilon_{\mathcal{A}}(\lambda, k) \le \sum_{t \in [|S|]} p_t + \mathsf{negl}(\lambda).$$

Next, since TrapGen invokes TrapGenIndex on a random $t \in S$ and appealing to Eq. (8.2), we conclude that

$$\Pr[\mathsf{ExptSE}_{\mathcal{A}}(\lambda) = 1] = \sum_{t \in [|S|]} \frac{p_t}{|S|} \ge \sum_{t \in [|S|]} \frac{p_t}{k} \ge \frac{1}{k} \cdot \varepsilon_{\mathcal{A}}(\lambda, k) - \mathsf{negl}(\lambda). \qquad \square$$

8.1 Monotone Policy Aggregate Signature via Semi-Somewhere Extractability

In Section 7, we showed how to combine a non-adaptively-sound monotone BARG with a puncturable signature scheme to obtain a monotone policy aggregate signature scheme. In this section, we show that the same construction is also secure for *any* signature scheme (not necessarily puncturable) if we rely on semi-somewhere extractability instead. We first recall the notion of a standard (non-puncturable) signature scheme:

Definition 8.20 (Digital Signature)**.** An digital signature scheme with message space $\{0,1\}^\lambda$ is a tuple of efficient algorithms $\Pi_{\mathsf{Sig}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a key pair $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input a signing key sk and a message $m \in \{0,1\}^\lambda$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to b$: On input a verification key vk, a message $m \in \{0,1\}^\lambda$, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$.

Moreover, the signature scheme should satisfy the following properties:

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

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \ : \ \begin{array}{c} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array} \right] = 1.$$

- **Unforgeability:** For all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 : \begin{array}{c} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \end{array} \right] = \mathsf{negl}(\lambda),$$

  where we say $\mathcal{A}$ is admissible if it does not query the signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$ on the message m^* in the above security game.

Theorem 8.21 (Static Unforgeability)**.** *Consider an instantiation of Construction 4.4 where the puncturable signature scheme Π_{PunctSig} is replaced by a standard digital signature scheme $\Pi_{\mathsf{Sig}} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ (Definition 8.20). If $\Pi_{\mathsf{MP\text{-}BARG}}$ is semi-somewhere extractable and satisfies set hiding, and Π_{Sig} is unforgeable, then Construction 7.5 is statically unforgeable.*

Proof. Let $\mathcal{A}$ be any efficient adversary for the static security game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is the static unforgeability experiment:
  1. On input the security parameter λ, the adversary $\mathcal{A}$ outputs the number of parties 1^k, a number of verification keys 1^n, a bound on the policy size 1^{s_p}, a challenge message $m^* \in \{0,1\}^\lambda$, and a monotone policy $P \colon \{0,1\}^k \to \{0,1\}$.

2. The challenger samples key-pairs $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$ for all $i \in [n]$ and sends $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ to the adversary.
3. The adversary $\mathcal{A}$ can now issue signing queries. Each signing query consists of an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$. The challenger responds with $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$.
4. After the adversary is finished making signing queries, it outputs a tuple of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$.
5. The challenger replies with the common reference string $\mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p})$.
6. The adversary $\mathcal{A}$ can continue to make signing queries. The challenger responds to these exactly as before. When $\mathcal{A}$ finishes making signing queries, the adversary outputs the aggregate signature σ_{agg}^*.
7. The output of the experiment is 1 if all of the following holds:
   - For each $i \in [k]$, let $b_i = 0$ if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$. Otherwise, let $b_i = 1$. Then, it holds that $P(b_1, \ldots, b_k) = 0$.
   - $\mathsf{BARG.Verify}\big(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*\big) = 1$.

   Otherwise, the output is 0.

- Hyb_1: Same as Hyb_0, except the challenger uses the following modified procedure to sample the BARG common reference string:
  - Sample $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TrapGen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p}, 1^k, S)$, where $S = \big\{i \in [k] \mid \exists j \in [n] : \mathsf{vk}_i^* = \mathsf{vk}_j\big\}$.
- Hyb_2: Same as Hyb_1, except the experiment outputs 1 if all of the following holds:
  - For each $i \in [k]$, let $b_i = 0$ if $\mathsf{vk}_i^* = \mathsf{vk}_j$ for some $j \in [n]$. Otherwise, let $b_i = 1$. Then, it holds that $P(b_1, \ldots, b_k) = 0$.
  - Compute $(i, \sigma_i) \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*)$ and check that $C_{m^*}(\mathsf{vk}_i^*, \sigma_i) = 1$ and $i \in S$.

  If either check fails, then the challenger outputs 0. Notably, the experiment no longer checks the condition $\mathsf{BARG.Verify}\big(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*\big) = 1$.
- Hyb_3: Same as Hyb_2, except the challenger samples a random index $i^* \xleftarrow{\mathrm{R}} [n]$ at the beginning of the security game. At the end of the game, after computing $(i, \sigma_i) \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*)$, the challenger *additionally* checks that $\mathsf{vk}_i^* = \mathsf{vk}_{i^*}$. If the check fails, the challenger outputs 0.

Lemma 8.22. *If $\Pi_{\mathsf{MP\text{-}BARG}}$ satisfies set hiding, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the set hiding game as follows:

1. On input a security parameter 1^λ, compute $(1^k, 1^n, 1^{s_p}, m^*, P) \leftarrow \mathcal{A}(1^\lambda)$.
2. Algorithm $\mathcal{B}$ samples key-pairs $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$ for all $i \in [n]$ and send $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ to $\mathcal{A}$.
3. Whenever algorithm $\mathcal{A}$ makes a signing query on an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$, algorithm $\mathcal{B}$ responds with $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$.
4. After $\mathcal{A}$ is finished making signing queries, it outputs a tuple of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$.
5. Let $S = \big\{i \in [k] \mid \exists j \in [n] : \mathsf{vk}_i^* = \mathsf{vk}_j\big\}$. Algorithm $\mathcal{B}$ sends the tuple $(1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p}, 1^k, S)$ to the $\Pi_{\mathsf{MP\text{-}BARG}}$ challenger. The challenger replies with a common reference string crs, which algorithm $\mathcal{B}$ forwards to algorithm $\mathcal{A}$.
6. Algorithm $\mathcal{B}$ responds to additional signing queries exactly as before.

7. Upon receiving an aggregate signature σ^*_{agg} from $\mathcal{A}$, algorithm $\mathcal{B}$ outputs 1 if all of the following holds:
   (a) $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise.
   (b) $\mathsf{BARG.Verify}\big(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_k), \sigma^*_{\mathsf{agg}}\big) = 1$.

   Otherwise, algorithm $\mathcal{B}$ outputs 0.

If the set hiding challenger samples $\mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p})$, then $\mathcal{B}$ perfectly simulates Hyb_0 for $\mathcal{A}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$. On the other hand, if the set hiding challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TrapGen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p}, 1^k, S)$, then $\mathcal{B}$ perfectly simulates an execution of Hyb_1 for $\mathcal{A}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$. We conclude that algorithm $\mathcal{B}$ breaks set hiding with the same advantage ε. □

Lemma 8.23. *If $\Pi_{\mathsf{MP\text{-}BARG}}$ is semi-somewhere extractable, then there exists a negligible function such that*

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \frac{1}{k} \cdot \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \mathsf{negl}(\lambda).$$

Proof. Let $k = k(\lambda)$ be the number of parties that algorithm $\mathcal{A}$ outputs.[11] We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the semi-somewhere extractability game (with parameter $k = k(\lambda)$ as follows:

1. On input a security parameter 1^λ, algorithm $\mathcal{B}$ computes $(1^k, 1^n, 1^{s_p}, m^*, P) \leftarrow \mathcal{A}(1^\lambda)$.

2. Algorithm $\mathcal{B}$ samples key-pairs $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$ for all $i \in [n]$ and send $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ to $\mathcal{A}$.

3. Whenever algorithm $\mathcal{A}$ makes a signing query on an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$, algorithm $\mathcal{B}$ responds with $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$.

4. After $\mathcal{A}$ is finished making signing queries, it outputs a tuple of verification keys $(\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_k)$.

5. Let $S = \big\{i \in [k] \mid \exists j \in [n] : \mathsf{vk}^*_i = \mathsf{vk}_j\big\}$. Let $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise. If $P(b_1, \ldots, b_k) = 0$, then algorithm $\mathcal{B}$ aborts with output $\perp$. Otherwise, algorithm $\mathcal{B}$ sends the tuple $(1^{\ell_{\mathsf{vk}}}, 1^{s_c}, 1^{s_p}, P, S)$ to the challenger and receives a common reference string crs. Algorithm $\mathcal{B}$ forwards crs to $\mathcal{A}$.

6. Algorithm $\mathcal{B}$ responds to additional signing queries exactly as before.

7. Upon receiving an aggregate signature σ^*_{agg} from $\mathcal{A}$, algorithm $\mathcal{B}$ outputs the circuit C_{m^*}, the instances $(\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_k)$ and the proof σ^*_{agg}.

Algorithm $\mathcal{B}$ is admissible by construction (since $P(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $i \in S$ and $b_i = 1$ otherwise). Next, algorithm $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in the hybrids Hyb_1 and Hyb_2. By assumption, with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$, algorithm $\mathcal{A}$ outputs σ_{agg} where

$$\mathsf{BARG.Verify}\big(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_k), \sigma^*_{\mathsf{agg}}\big) = 1.$$

This means that

$$\varepsilon_{\mathcal{B}}(\lambda, k) = \Pr[\mathsf{BARG.Verify}\big(\mathsf{crs}, C_{m^*}, P, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_k), \sigma^*_{\mathsf{agg}}\big) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1], \tag{8.4}$$

where $\varepsilon_{\mathcal{B}}(\lambda, k)$ is the quantity from Eq. (8.1). By somewhere extractability of $\Pi_{\mathsf{MP\text{-}BARG}}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{ExptSE}_{\mathcal{B}}(\lambda, k) = 1] \geq \frac{1}{k} \cdot \varepsilon_{\mathcal{B}}(\lambda, k) - \mathsf{negl}(\lambda), \tag{8.5}$$

Next, the output of $\mathsf{ExptSE}_{\mathcal{B}}(\lambda, k)$ is 1 if $C_{m^*}(\mathsf{vk}^*_i, \sigma_i) = 1$ and $i \in S$ where

$$(i, \sigma_i) \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{m^*}, P, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_k), \sigma^*_{\mathsf{agg}}).$$

[11] We can assume that for each value of $\lambda \in \mathbb{N}$, algorithm $\mathcal{A}$ always picks a *fixed* value of k. This can be the value that maximizes its success probability for each value of λ (formally, we can take this "maximizing" value of k to be non-uniform advice provided to $\mathcal{A}$).

This is the same set of conditions checked in Hyb_2 and we conclude that

$$\Pr[\mathsf{ExptSE}_{\mathcal{B}}(\lambda, k) = 1] = \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]. \tag{8.6}$$

Combining Eqs. (8.4) to (8.6), we have that

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \frac{1}{k} \cdot \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \mathsf{negl}(\lambda). \qquad \square$$

Lemma 8.24. $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \frac{1}{n} \cdot \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

Proof. By construction, the adversary's view in Hyb_2 and Hyb_3 is identical. The only difference is how the output of the experiment is computed. Suppose $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] = \varepsilon$. Then, with probability ε, the extracted value (i, σ_i) in Hyb_2 and Hyb_3 satisfies $i \in S$. By definition of S, there exists some $j \in [n]$ such that $\mathsf{vk}_i^* = \mathsf{vk}_j$. Since $i^* \xleftarrow{\text{R}} [n]$ (and is entirely independent of the view of the adversary), $\Pr[j = i^*] = 1/n$. When $j = i^*$, the output in Hyb_3 is also 1 (and otherwise, it is 0). As such, $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \frac{1}{n} \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$ and the claim follows. □

Lemma 8.25. *If Π_{Sig} is unforgeable, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.*

Proof. Suppose $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] > \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks unforgeability of Π_{Sig}:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a verification key vk from the challenger.
2. On input a security parameter 1^λ, algorithm $\mathcal{B}$ computes $(1^k, 1^n, 1^{s_P}, m^*, P) \leftarrow \mathcal{A}(1^\lambda)$.
3. Algorithm $\mathcal{B}$ samples a random index $i^* \xleftarrow{\text{R}} [n]$. For $i \in [n] \setminus \{i^*\}$, algorithm $\mathcal{B}$ samples $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$. It sets $\mathsf{vk}_{i^*} \coloneqq \mathsf{vk}$ and send $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ to $\mathcal{A}$.
4. Whenever $\mathcal{A}$ makes a signing query on an index $i \in [n]$ and a message $m \in \{0,1\}^\lambda \setminus \{m^*\}$, if $i = i^*$, then algorithm $\mathcal{B}$ forwards the query to the challenger and receives a signature σ. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with σ. If $i \neq i^*$, then algorithm $\mathcal{B}$ replies with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$.
5. At some point, algorithm $\mathcal{A}$ outputs a collection of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$.
6. Let $S = \left\{ i \in [k] \mid \exists j \in [n] : \mathsf{vk}_i^* = \mathsf{vk}_j \right\}$. For each $i \in S$, set $b_i = 0$ and set $b_i = 1$ otherwise. If $P(b_1, \ldots, b_k) = 1$, then algorithm $\mathcal{B}$ aborts. Otherwise, algorithm $\mathcal{B}$ samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{BARG.TrapGen}(1^\lambda, 1^{\ell_{\mathsf{vk}}}, 1^{s_C}, 1^{s_P}, 1^k, S)$ and send crs to $\mathcal{A}$.
7. Algorithm $\mathcal{B}$ responds to additional signing queries exactly as before.
8. Upon receiving an aggregate signature σ_{agg}^* from $\mathcal{A}$, algorithm $\mathcal{B}$ computes
$$(i, \sigma_i) \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{m^*}, P, (\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*), \sigma_{\mathsf{agg}}^*).$$
If $\mathsf{vk}_i^* \neq \mathsf{vk}_{i^*} = \mathsf{vk}$ then algorithm $\mathcal{B}$ aborts. Otherwise, algorithm output $\sigma^* = \sigma_i$ and m^*.

Algorithm $\mathcal{B}$ is admissible since it never needs to query its challenger for a signature on m^*. Next, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_3 for $\mathcal{A}$. Thus, with probability ε, algorithm $\mathcal{A}$ outputs a tuple of verification keys $(\mathsf{vk}_1^*, \ldots, \mathsf{vk}_k^*)$ and a signature σ_{agg} such that the extracted index-signature pair (i, σ_i) satisfies $\mathsf{vk}_i^* = \mathsf{vk}_{i^*} = \mathsf{vk}$ and $C_{m^*}(\mathsf{vk}_i^*, \sigma_i) = 1$. By definition of C_{m^*}, this means that

$$1 = \mathsf{Verify}(\mathsf{vk}_i^*, m^*, \sigma_i) = \mathsf{Verify}(\mathsf{vk}, m^*, \sigma_i),$$

in which case $\mathcal{B}$ wins the unforgeability game. □

Theorem 8.21 now follows immediately from Lemmas 8.22 to 8.25. □

Acknowledgments

We thank Yuval Ishai for helpful pointers on batch arguments. Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2151131, CNS-2140975, CNS-2318701, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[BBK$^+$23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, pages 252–283, 2023.

[BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In *EUROCRYPT*, 2024.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BKP22] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable private information retrieval. In *TCC*, pages 3–32, 2022.

[BWY11] Mihir Bellare, Brent Waters, and Scott Yilek. Identity-based encryption secure against selective opening attack. In *TCC*, pages 235–252, 2011.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *PKC*, pages 55–72, 2013.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential DDH. In *CRYPTO*, pages 635–668, 2023.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO*, pages 394–423, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, pages 68–79, 2021.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *FOCS*, pages 1057–1068, 2022.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In *CRYPTO*, pages 498–531, 2023.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *PKC*, pages 3–33, 2019.

[GZ21] Alonso González and Alexandros Zacharakis. Succinct publicly verifiable computation. *IACR Cryptol. ePrint Arch.*, page 353, 2021.

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In *EUROCRYPT*, pages 520–549, 2022.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, pages 163–172, 2015.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, pages 1545–1552, 2023.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, pages 1115–1124, 2019.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In *TCC*, pages 330–368, 2021.

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1987.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *FOCS*, pages 1045–1056, 2022.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *TCC*, pages 283–315, 2017.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[Vad06] Salil P. Vadhan. An unconditional study of computational zero knowledge. *SIAM J. Comput.*, 36(4):1160–1214, 2006.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, pages 433–463, 2022.

A Set Hiding with Extraction for BARGs

We now show to construct a BARG satisfying Definition 2.8 from any somewhere extractable BARG that supports extraction on a single instance (e.g., [CJJ21b, WW22, HJKS22, DGKV22, KLVW23]). Our construction is a direct parallel of the analogous constructions from [GZ21, CJJ21b] in the setting of somewhere extractable commitments.

Construction A.1 (BARGs Satisfying Set Hiding with Extraction)**.** Let $\Pi'_{\mathsf{BARG}} = (\mathsf{Gen}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{TrapGen}', \mathsf{Extract}')$ be a somewhere-extractable BARG for Boolean circuit satisfiability that supports extraction on a single instance. We use Π'_{BARG} to construct a new BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{TrapGen}, \mathsf{Extract})$ that supports extraction on multiple instances and which satisfies Definition 2.8:

- $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell)$: On input the security parameter λ, the number of instances k, the instance length n, the bound on the size of the Boolean circuit s, and the bound on the size of the extraction set ℓ, the generator algorithm samples $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$ for each $i \in [\ell]$. Then, it samples a random permutation $\tau \colon [\ell] \to [\ell]$ and outputs $\mathsf{crs} = \big(\mathsf{crs}'_{\tau(1)}, \ldots, \mathsf{crs}'_{\tau(\ell)}\big)$ and $\mathsf{vk} = \big(\mathsf{vk}'_{\tau(1)}, \ldots, \mathsf{vk}'_{\tau(\ell)}\big)$.
- $\mathsf{Prove}(\mathsf{crs}, C, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}'_1, \ldots, \mathsf{crs}'_\ell)$, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$, the prove algorithm computes $\pi'_i \leftarrow \mathsf{Prove}'(\mathsf{crs}'_i, C, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$ for all $i \in [k]$ and outputs $\pi = (\pi'_1, \ldots, \pi'_\ell)$.
- $\mathsf{Verify}(\mathsf{vk}, C, (x_1, \ldots, x_k), \pi)$: On input the verification key $\mathsf{vk} = (\mathsf{vk}'_1, \ldots, \mathsf{vk}'_\ell)$, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof $\pi = (\pi'_1, \ldots, \pi'_\ell)$, the verification algorithm outputs 1 if for all $i \in [\ell]$, it holds that $\mathsf{Verify}'(\mathsf{vk}'_i, C, (x_1, \ldots, x_k), \pi'_i) = 1$. Otherwise, it outputs 0.

- $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell, S)$: On input the security parameter λ, the number of instances k, the instance size n, the bound on the size of the Boolean circuit s, the bound on the size of the extraction set ℓ, and a set of indices $S \subseteq [k]$ of size at most ℓ, the trapdoor-generator algorithm proceeds as follows:
  - Let $S = \{j_1, \ldots, j_d\}$ where $j_1 < j_2 < \cdots < j_d$ are in sorted order.
  - For each $i \in [d]$, sample $(\mathsf{crs}'_i, \mathsf{vk}'_i, \mathsf{td}'_i) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_i\})$. For each $i \in \{d+1, \ldots, \ell\}$, sample $\mathsf{crs}'_i \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$.
  - Sample a random permutation $\tau \colon [\ell] \to [\ell]$, and define the dictionary D where $\mathsf{D}[j_i] \mapsto \left(\tau^{-1}(i), \mathsf{td}'_i\right)$ for all $i \in S$. Output $\mathsf{crs} = \left(\mathsf{crs}'_{\tau(1)}, \ldots, \mathsf{crs}'_{\tau(\ell)}\right)$, $\mathsf{vk} = \left(\mathsf{vk}'_{\tau(1)}, \ldots, \mathsf{vk}'_{\tau(\ell)}\right)$, and $\mathsf{td} = \mathsf{D}$.

- $\mathsf{Extract}(\mathsf{td}, C, (x_1, \ldots, x_k), \pi, i)$: On input the trapdoor $\mathsf{td} = \mathsf{D}$, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, a proof $\pi = (\pi'_1, \ldots, \pi'_\ell)$, and an index i, the extraction algorithm outputs $\perp$ if $i \notin \mathsf{D}$. Otherwise, let $(t, \mathsf{td}') \leftarrow \mathsf{D}[i]$ and output $\mathsf{Extract}'\left(\mathsf{td}', C, (x_1, \ldots, x_k), \pi'_t, i\right)$.

Theorem A.2 (Completeness). *If Π'_{BARG} satisfies completeness, then Π_{BARG} in Construction A.1 is also complete.*

Proof. This follows by construction. Specifically, take any $\lambda, k, n, s, \ell \in \mathbb{N}$, any Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, any sequence of statements $x_1, \ldots, x_k \in \{0,1\}^n$ along with witnesses $w_1, \ldots, w_k \in \{0,1\}^h$ where $C(x_i, w_i) = 1$ for all $i \in [k]$. Then, the following properties hold:

- Suppose $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell)$. By construction, this means $\mathsf{crs} = (\mathsf{crs}'_1, \ldots, \mathsf{crs}'_\ell)$ and $\mathsf{vk} = (\mathsf{vk}'_1, \ldots, \mathsf{vk}'_\ell)$. Moreover, for all $i \in [\ell]$, we have that $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$.
- Let $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$. By construction, $\pi = (\pi'_1, \ldots, \pi'_\ell)$ where for all $i \in [\ell]$, we have that $\pi'_i \leftarrow \mathsf{Prove}'(\mathsf{crs}'_i, C, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$.
- By completeness of Π'_{BARG}, $\mathsf{Verify}'(\mathsf{vk}'_i, C, (x_1, \ldots, x_k), \pi'_i) = 1$ for all $i \in [\ell]$. Thus, $\mathsf{Verify}(\mathsf{vk}, C, (x_1, \ldots, x_k), \pi)$ outputs 1 and completeness holds. □

Theorem A.3 (Set Hiding). *If Π'_{BARG} satisfies set hiding, then Π_{BARG} in Construction A.1 also satisfies set hiding.*

Proof. We start by defining a sequence of hybrid experiments.

- Hyb_0: This is experiment $\mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0)$. Namely, after the adversary chooses the parameters $k, n, s, \ell \in \mathbb{N}$ and the set $S \subseteq [k]$, the challenger replies with $\mathsf{crs} = \left(\mathsf{crs}'_{\tau(1)}, \ldots, \mathsf{crs}'_{\tau(\ell)}\right)$ and $\mathsf{vk} = \left(\mathsf{vk}'_{\tau(1)}, \ldots, \mathsf{vk}'_{\tau(\ell)}\right)$ where $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$ for all $i \in [\ell]$ and τ is a random permutation. At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$ which is the output of the experiment.
- Hyb_i for $i \in [d]$: Same as Hyb_0, except for indices $t \le i$, the challenger now computes $(\mathsf{crs}'_t, \mathsf{vk}'_t, \mathsf{td}'_t) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_t\})$, where $S = \{j_1, \ldots, j_d\}$ and $j_1 < \cdots < j_d$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$. By construction, $\mathsf{Hyb}_0(\mathcal{A}) \equiv \mathsf{ExptSH}_{\mathcal{A}}(\lambda, 0)$ and $\mathsf{Hyb}_d(\mathcal{A}) \equiv \mathsf{ExptSH}_{\mathcal{A}}(\lambda, 1)$. We now show that if Π'_{BARG} satisfies set hiding, then for all $i \in [d]$, and for all efficient adversaries $\mathcal{A}$, the output distributions $\mathsf{Hyb}_{i-1}(\mathcal{A})$ and $\mathsf{Hyb}_i(\mathcal{A})$ are computationally indistinguishable. To see this, suppose there exists an efficient adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]\right| \ge \varepsilon(\lambda),$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that breaks set hiding of Π'_{BARG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts by running adversary $\mathcal{A}$ on the same security parameter. Algorithm $\mathcal{A}$ outputs 1^k, 1^n, 1^s, 1^ℓ, and a set $S = \{j_1, \ldots, j_d\}$, where $j_1 < \cdots < j_d$.
2. Algorithm $\mathcal{B}$ sends 1^k, 1^n, 1^s, 1^1, and $\{j_i\}$ to the set hiding challenger for Π'_{BARG} and receives a pair $(\mathsf{crs}^*, \mathsf{vk}^*)$. It sets $\mathsf{crs}'_i = \mathsf{crs}^*$ and $\mathsf{vk}'_i = \mathsf{vk}^*$.

3. For $t < i$, algorithm $\mathcal{B}$ samples $(\mathsf{crs}'_t, \mathsf{vk}'_t, \mathsf{td}'_t) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_t\})$. For $t \in \{i+1, \dots, \ell\}$, it samples $(\mathsf{crs}'_j, \mathsf{vk}'_j) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$.

4. Finally, algorithm $\mathcal{B}$ samples a random permutation $\tau\colon [\ell] \to [\ell]$ and gives $\mathsf{crs} = \left(\mathsf{crs}'_{\tau(1)}, \dots, \mathsf{crs}'_{\tau(\ell)}\right)$ and $\mathsf{vk} = \left(\mathsf{vk}'_{\tau(1)}, \dots, \mathsf{vk}'_{\tau(\ell)}\right)$ to $\mathcal{A}$. It outputs whatever $\mathcal{A}$ outputs.

By design, if $(\mathsf{crs}^*, \mathsf{vk}^*) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_{i-1} for $\mathcal{A}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1]$. Alternatively, if $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{td}^*) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_i\})$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_i for $\mathcal{A}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]$. We conclude that algorithm $\mathcal{B}$ breaks set hiding of Π'_{BARG} with the same non-negligible advantage ε. The claim now follows by a hybrid argument. □

Theorem A.4 (Set Hiding with Extraction)**.** *If Π'_{BARG} satisfies set hiding, then Π_{BARG} in Construction A.1 satisfies set hiding with extraction.*

Proof. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is experiment $\mathsf{ExptSHwE}_{\mathcal{A}}(\lambda, 0)$:
  - At the beginning of the game, the adversary chooses the parameters $k, n, s, \ell \in \mathbb{N}$, the set $S \subseteq [k]$, and the index $i^* \in S$. Let $S = \{j_1, \dots, j_d\}$ where $j_1 < \cdots < j_d$. Let $t^* \in [d]$ be the index where $i^* = j_{t^*}$.
  - Then, for each $i \in [d]$, the challenger samples $(\mathsf{crs}'_i, \mathsf{vk}'_i, \mathsf{td}'_i) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_i\})$. For each $i \in \{d+1, \dots, \ell\}$, sample $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$. Then, it samples a random permutation $\tau\colon [\ell] \to [\ell]$. It gives $\mathsf{crs} = \left(\mathsf{crs}'_{\tau(1)}, \dots, \mathsf{crs}'_{\tau(\ell)}\right)$ and $\mathsf{vk} = \left(\mathsf{vk}'_{\tau(1)}, \dots, \mathsf{vk}'_{\tau(\ell)}\right)$ to $\mathcal{A}$. The challenger also sets $\mathsf{td}' = \mathsf{td}'_{t^*}$ and $z = \tau^{-1}(t^*)$.
  - Algorithm $\mathcal{A}$ outputs a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \dots, x_k \in \{0,1\}^n$, and a proof $\pi = (\pi'_1, \dots, \pi'_\ell)$.
  - The challenger checks that for all $i \in [\ell]$, it holds that $\mathsf{Verify}'(\mathsf{vk}_{\tau(i)}, C, (x_1, \dots, x_k), \pi'_i) = 1$. If not, the challenger halts with output 0. Otherwise, the challenger replies with $\mathsf{Extract}'(\mathsf{td}', C, (x_1, \dots, x_k), \pi'_z, i^*)$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_1: Same as Hyb_0, except the challenger swaps $(\mathsf{crs}'_1, \mathsf{vk}'_1)$ with $(\mathsf{crs}'_{t^*}, \mathsf{vk}'_{t^*})$. In more detail, the challenger samples $(\mathsf{crs}'_1, \mathsf{vk}'_1, \mathsf{td}'_1) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_{t^*}\})$ and $(\mathsf{crs}'_{t^*}, \mathsf{vk}'_{t^*}, \mathsf{td}'_{t^*}) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_1\})$. In addition, it sets $\mathsf{td}' = \mathsf{td}'_1$ and $z = \tau^{-1}(1)$. The remainder of the experiment proceeds as in Hyb_0.
- Hyb_i for $i \in \{2, \dots, d\}$: Same as Hyb_1 except for indices $t \in \{2, \dots, i\}$, the challenger now computes $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of an execution of hybrid Hyb_i with adversary $\mathcal{A}$. By construction $\mathsf{Hyb}_0(\mathcal{A}) \equiv \mathsf{ExptSHwE}_{\mathcal{A}}(\lambda, 0)$ while $\mathsf{Hyb}_d(\mathcal{A}) \equiv \mathsf{ExptSHwE}_{\mathcal{A}}(\lambda, 1)$. To complete the proof, we now show that the output of each adjacent pair of hybrid experiments are indistinguishable.

Lemma A.5. *For all adversaries $\mathcal{A}$, we have that $\mathsf{Hyb}_0(\mathcal{A}) \equiv \mathsf{Hyb}_1(\mathcal{A})$.*

Proof. The view of the adversary in the two experiments is identical since τ is a random permutation. More precisely, the distribution in Hyb_1 corresponds to the distribution in Hyb_0 where the permutation τ is replaced by $\tau \circ \sigma$ where $\sigma\colon [\ell] \to [\ell]$ is the elementary permutation that transposes j_1 with j_{t^*} (and fixes all other inputs). Since τ is uniform, the distributions of τ and $\tau \circ \sigma$ are identical. □

Lemma A.6. *If Π'_{BARG} satisfies set hiding, then for all $i \in \{2, \dots, d\}$ and all efficient adversaries $\mathcal{A}$, it holds that $\left|\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks set hiding of Π'_{BARG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running adversary $\mathcal{A}$ on the same security parameter. Algorithm $\mathcal{A}$ outputs 1^k, 1^n, 1^s, 1^ℓ, a set $S = \{j_1, \ldots, j_d\}$ where $j_1 < \cdots < j_d$, and an index $i^* \in S$. Let $t^* \in [d]$ be the index where $i^* = j_{t^*}$. Let $\rho_1 = j_{t^*}$, $\rho_{t^*} = j_1$, and $\rho_i = j_i$ for all $i \notin \{1, t^*\}$.

2. Algorithm $\mathcal{B}$ sends 1^k, 1^n, 1^s, 1^1, and $\{\rho_i\}$ to the set hiding challenger for Π'_{BARG} and receives a common reference string crs^* and verification key vk^*.

3. Algorithm $\mathcal{B}$ samples $(\mathsf{crs}'_1, \mathsf{vk}'_1, \mathsf{td}'_1) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^1, \{\rho_1\})$. For $2 \le t < i$, algorithm $\mathcal{B}$ samples $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$. Finally, for all indices $t \in \{i+1, \ldots, d\}$, it samples $(\mathsf{crs}'_i, \mathsf{vk}'_i, \mathsf{td}'_i) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{\rho_i\})$. For each $t \in \{d+1, \ldots, \ell\}$, it samples $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$. It sets $\mathsf{crs}'_i = \mathsf{crs}^*$ and $\mathsf{vk}'_i = \mathsf{vk}^*$.

4. Algorithm $\mathcal{B}$ samples a random permutation $\tau \colon [\ell] \to [\ell]$ and gives $\mathsf{crs} = \left(\mathsf{crs}'_{\tau(1)}, \ldots, \mathsf{crs}'_{\tau(\ell)}\right)$ and $\mathsf{vk} = \left(\mathsf{vk}'_{\tau(1)}, \ldots, \mathsf{vk}'_{\tau(\ell)}\right)$ to $\mathcal{A}$. It also sets $\mathsf{td}' = \mathsf{td}'_1$ and $z = \tau^{-1}(1)$.

5. Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof $\pi = (\pi'_1, \ldots, \pi'_\ell)$.

6. Algorithm $\mathcal{B}$ first checks that for all $i \in [\ell]$, it holds that $\mathsf{Verify}'(\mathsf{vk}_{\tau(i)}, C, (x_1, \ldots, x_k), \pi'_i) = 1$. If not, it halts with output 0. Otherwise, algorithm $\mathcal{B}$ replies with $\mathsf{Extract}'(\mathsf{td}', C, (x_1, \ldots, x_k), \pi'_z, i^*)$.

7. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

By design, if $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{td}^*) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{\rho_i\})$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_{i-1} for $\mathcal{A}$, so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1]$. Conversely, if $(\mathsf{crs}^*, \mathsf{vk}^*) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_i for $\mathcal{A}$ and algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]$. Thus, the distinguishing advantage of algorithm $\mathcal{B}$ is at least ε, which is non-negligible by assumption. □

Security now follows by combining Lemmas A.5 and A.6 and appealing to the fact that $d \le \ell = \mathsf{poly}(\lambda)$. □

Theorem A.7 (Somewhere Extraction). *If Π'_{BARG} is somewhere extractable in trapdoor mode, then Π_{BARG} in Construction A.1 is also somewhere extractable in trapdoor mode.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that breaks the somewhere extractability of Construction A.1 with non-negligible probability $\varepsilon(\lambda)$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks the somewhere extractability of Π_{BARG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts by running algorithm $\mathcal{A}$ on the same parameter. Algorithm $\mathcal{A}$ outputs 1^k, 1^n, 1^s, 1^ℓ, and a set $S = \{j_1, \ldots, j_d\}$, where $j_1 < \cdots < j_d$. Algorithm $\mathcal{B}$ samples a random index $t^* \xleftarrow{\text{R}} [d]$ and sends 1^k, 1^n, 1^s, 1^1, and $\{j_{t^*}\}$ to its challenger. It receives a common reference string crs^* and a verification key vk^*.

2. For $t \in [d] \setminus \{t^*\}$, algorithm $\mathcal{B}$ samples $(\mathsf{crs}'_t, \mathsf{vk}'_t, \mathsf{td}'_t) \leftarrow \mathsf{TrapGen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1, \{j_t\})$. For each $i \in \{d+1, \ldots, \ell\}$, it samples $(\mathsf{crs}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$. Finally, it sets $\mathsf{crs}'_{t^*} = \mathsf{crs}^*$.

3. Finally, algorithm $\mathcal{B}$ samples a random permutation $\tau \colon [\ell] \to [\ell]$ and gives $\mathsf{crs} = \left(\mathsf{crs}'_{\tau(1)}, \ldots, \mathsf{crs}'_{\tau(\ell)}\right)$ and $\mathsf{vk} = \left(\mathsf{vk}'_{\tau(1)}, \ldots, \mathsf{vk}'_{\tau(\ell)}\right)$ to $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $x_1, \ldots, x_k \in \{0,1\}^n$ and a proof $\pi = (\pi'_1, \ldots, \pi'_\ell)$. Algorithm $\mathcal{B}$ outputs the circuit C, the statement $x_{j^*_t}$, and the proof $\pi'_{\tau^{-1}(t^*)}$.

First, algorithm $\mathcal{B}$ perfectly simulates the common reference string crs for $\mathcal{A}$, so with probability at least ε, algorithm $\mathcal{A}$ outputs $(C, x_1, \ldots, x_k, \pi)$ such that there exists some $t \in [d]$ such that the following two conditions hold:

- $\mathsf{Verify}'\left(\mathsf{vk}'_{\tau(z)}, C, (x_1, \ldots, x_k), \pi'_z\right) = 1$ where $z = \tau^{-1}(t) \in [\ell]$; and

- $C(x_{j_t}, w_{j_t}) \neq 1$ where $w_{j_t} \leftarrow \mathsf{Extract}'\left(\mathsf{td}'_t, C, (x_1, \ldots, x_k), \pi'_z, j_t\right)$.

Moreover, the special index t^* is perfectly hidden from the view of $\mathcal{A}$, so with probability $1/|S| \geq 1/\ell$, it will be the case that $t^* = t$. In this case, $\tau(z) = \tau(\tau^{-1}(t)) = t$, so we have that π'_z verifies with respect to $\mathsf{vk}'_{\tau(z)} = \mathsf{vk}'_{t^*}$, but the extracted witness $w_{j_{t^*}}$ is such that $C(x_{j_{t^*}}, w_{j_{t^*}}) \neq 1$. This breaks somewhere extractability of Π_{BARG}. Thus, if $\mathcal{A}$ succeeds with advantage ε, then algorithm $\mathcal{B}$ succeeds with advantage at least ε/ℓ, which is non-negligible as $\ell = \mathsf{poly}(\lambda)$. □

Theorem A.8 (Succinctness)**.** *If* Π'_{BARG} *is succinct, then* Π_{BARG} *in Construction A.1 is also succinct.*

Proof. Take any $\lambda, k, n, s, \ell \in \mathbb{N}$ and any $(\mathsf{crs}, \mathsf{vk})$ in the support of $\mathsf{Gen}(1^\lambda, 1^k, 1^n, 1^s, 1^\ell)$. Then, $\mathsf{crs} = (\mathsf{crs}'_1, \ldots, \mathsf{crs}'_\ell)$ and $\mathsf{vk} = (\mathsf{vk}'_1, \ldots, \mathsf{vk}'_\ell)$, where $(\mathsf{crs}'_i, \mathsf{vk}'_i)$ is in the support of $\mathsf{Gen}'(1^\lambda, 1^k, 1^n, 1^s, 1^1)$. Take any Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$. We consider each condition separately:

- **Succinct proof:** By succinctness of Π_{BARG}, the proofs π' output by $\mathsf{Prove}'(\mathsf{crs}', C, \cdot, \cdot)$ satisfy $|\pi'_i| \leq \mathsf{poly}(\lambda + \log k + s)$. Then, the proofs output by $\mathsf{Prove}(\mathsf{crs}, C, \cdot, \cdot)$ satisfy $|\pi| \leq \ell \cdot |\pi'_i| \leq \mathsf{poly}(\lambda + \log k + s + \ell)$.
- **Succinct CRS:** By succinctness of Π'_{BARG}, each crs'_i satisfies $|\mathsf{crs}'_i| \leq \mathsf{poly}(\lambda + k + n) + \mathsf{poly}(\lambda + \log m + s)$. The total size of the CRS is a factor of ℓ larger which satisfies the succinctness requirement.
- **Succinct verification key:** By succinctness of Π'_{BARG}, each vk'_i satisfies $|\mathsf{vk}'_i| \leq \mathsf{poly}(\lambda + \log k + s)$. The verification key vk output by Setup is a factor ℓ larger, which satisfies the succinctness requirement. □

Remark A.9 (Index BARGs)**.** While we described Construction A.1 for the case of standard BARGs, the same construction directly extends to the case of index BARGs, and moreover, the construction preserves the efficiency requirements of an index BARG (since it is simply a concatenation of ℓ copies of the underlying BARG).