Batch Arguments to NIZKs from One-Way Functions

Eli Bradley
UT Austin
elibradley@utexas.edu

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

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

Abstract

Succinctness and zero-knowledge are two fundamental properties in the study of cryptographic proof systems. Several recent works have formalized the connections between these two notions by showing how to realize non-interactive zero-knowledge (NIZK) arguments from succinct non-interactive arguments. Specifically, Champion and Wu (CRYPTO 2023) as well as Bitansky, Kamath, Paneth, Rothblum, and Vasudevan (ePrint 2023) recently showed how to construct a NIZK argument for NP from a (somewhere-sound) non-interactive batch argument (BARG) and a dual-mode commitment scheme (and in the case of the Champion-Wu construction, a local pseudorandom generator). The main open question is whether a BARG suffices for a NIZK (just assuming one-way functions).

In this work, we first show that an *adaptively-sound* BARG for NP together with an one-way function imply a computational NIZK argument for NP. We then show that the weaker notion of somewhere soundness achieved by existing BARGs from standard algebraic assumptions are also *adaptively* sound if we assume *sub-exponential* security. This transformation may also be of independent interest. Taken together, we obtain a NIZK argument for NP from one-way functions and a sub-exponentially-secure somewhere-sound BARG for NP.

If we instead assume *plain* public-key encryption, we show that a standard *polynomially-secure* somewhere-sound batch argument for NP suffices for the same implication. As a corollary, this means a somewhere-sound BARG can be used to generically upgrade any semantically-secure public-key encryption scheme into one secure against chosen-ciphertext attacks. More broadly, our results demonstrate that constructing non-interactive batch arguments for NP is essentially no easier than constructing NIZK arguments for NP.

1 Introduction

A non-interactive argument system for an NP relation $\mathcal{R}$ allows a (computationally-bounded) prover to convince a verifier that a statement $x \in \{0,1\}^*$ is true (i.e., that $x \in \mathcal{L}$) with a single message π (which is referred to as a "proof"). The argument system is succinct if the size of the proof π is sublinear in the size of the circuit computing $\mathcal{R}$ and is zero-knowledge [GMR85] if the proof π reveals nothing more about x other than the fact that x is true. Both succinctness and zero-knowledge are fundamental properties of proof systems, and their combination in the form of zero-knowledge succinct non-interactive arguments (zkSNARGs) have found extensive applications to verifiable computation, authentication schemes, and privacy-preserving digital currencies. A recent line of works [KMY20, CW23, BKP+23a] have studied the formal relationship between succinctness and zero-knowledge. Since a succinct argument is not long enough to encode a traditional NP witness, it intuitively must lose some information about the witness associated with the statement. Thus, it is not surprising that succinct argument systems give rise to zero-knowledge arguments. Such a connection was first formalized by Kitagawa, Matsuda, and Yamakawa [KMY20], who showed that a succinct non-interactive argument for NP can be used in conjunction with one-way functions to construct a non-interactive zero-knowledge (NIZK) argument.

Batch arguments. The construction in [KMY20] uses *adaptively-sound* SNARGs for NP as its starting point. However, constructing adaptively-sound SNARGs for NP in the plain model is challenging, and currently, all existing constructions from falsifiable assumptions rely on indistinguishability obfuscation [WW24a, WZ24, WW24b]. The question then is whether argument systems satisfying *weaker* notions of succinctness could still imply zero-knowledge. This was studied recently in two works [CW23, BKP+23a], which established a similar implication starting from the *weaker* notion of a non-interactive batch argument (BARG). Batch arguments allow a prover to amortize the communication cost of NP verification; namely, the prover can convince the verifier of a collection of t NP statements $(x_1, \ldots, x_t)$ with a proof of size $\text{poly}(\lambda, s) \cdot o(t)$, where s is the size of the circuit computing the associated NP relation and λ is a security parameter. Unlike the case of SNARGs, a recent line of works have shown how to construct batch arguments for NP from a broad range of standard number-theoretic assumptions [KVZ21, CJJ21a, CJJ21b, HJKS22, WW22, DGKV22, GSWW22, CGJ+23, KLVW23, BBK+23, NWW24]. The work of Champion and Wu [CW23] showed how to obtain a computational NIZK argument from a somewhere-sound[1] BARG for NP in conjunction with a dual-mode commitment scheme and a (sub-exponentially-secure) local pseudorandom generator (PRG). The work of Bitansky, Kamath, Paneth, Rothblum, and Vasudevan [BKP+23a] showed how to obtain a *statistical* NIZK argument from a somewhere-sound BARG for NP with a dual-mode commitment scheme.[2] In light of these works, a natural question is whether we can construct NIZK arguments solely from BARGs (and one-way functions).

This work. In this work, we first show how to construct a computational NIZK argument for NP from any one-way function together with a (polynomially-secure) *adaptively-sound* BARG for NP. Adaptive soundness is the most "natural" security notion for a BARG and can often be more convenient to use in constructions. However, most BARG constructions based on standard algebraic assumptions (e.g., [CJJ21b, WW22, DGKV22, CGJ+23]) only satisfy a weaker notion of "somewhere soundness" where the adversary has to pre-commit to the index of the false instance. As an additional contribution (of independent interest), we show in Section 5 that using complexity leveraging, any *sub-exponentially-secure* somewhere-sound BARG can also be used to obtain an adaptively-sound BARG. Thus, existing somewhere-sound BARGs based on standard algebraic assumptions also satisfy adaptive soundness at the expense of making a stronger sub-exponential hardness assumption. We believe this fact could also be useful in other applications of BARGs. In combination, we obtain a NIZK argument for NP from one-way functions and either (1) an adaptively-sound BARG for NP; *or* (2) a sub-exponentially-secure somewhere-sound BARG for NP (Corollary 3.5). Compared with [CW23] and [BKP+23a], our construction only requires BARGs and one-way functions. The previous works [CW23, BKP+23a] additionally relied on a dual-mode commitment scheme (and in the case of [CW23], also a local pseudorandom generator).

If we additionally assume *vanilla* public-key encryption, then we can obtain a computational NIZK for NP from a *polynomially-secure* somewhere-sound BARG for NP (Corollary 4.8). Namely, the use of public-key encryption allows us to relax the adaptive soundness requirement on the BARG to somewhere soundness. Like previous works, our constructions do *not* need extractability (although most existing BARGs support some type of extraction). We summarize our main results in the following informal theorem:

Theorem 1.1 (Informal). *There exists a NIZK for* NP *assuming the existence of* either

- *a one-way function and an* adaptively-sound *BARG for* NP*; or*
- *a public-key encryption scheme and a* somewhere-sound *BARG for* NP.

Moreover, adaptively-sound BARGs for NP *can be constructed from* sub-exponentially-secure *somewhere-sound BARGs for* NP.

Broadly speaking, our results demonstrate that constructing BARGs for NP is no easier than constructing (computational) NIZK arguments. Indeed, existing algebraic constructions of BARGs [CJJ21b, WW22, DGKV22, CGJ+23] are based on ideas and techniques that were previously used to build NIZK arguments.

An implication to CCA-security. Combined with classic results [NY90, SCO+01] on constructing public-key encryption with security against chosen-ciphertext attacks (CCA-security) from semantically-secure public-key encryption and (designated-verifier) NIZK arguments, our results show that BARGs for NP can be used to upgrade any semantically-secure public-key encryption scheme into a CCA-secure one. In this setting of upgrading the security of public-key encryption, we only require polynomial hardness of the BARG for NP (since we are given the semantically-secure public-key encryption to start).

[1]A BARG satisfies somewhere soundness if the common reference string (CRS) can be programmed at a specific index i, and adaptive soundness is guaranteed with respect to the i^{th} statements x_i. Moreover, the CRS (computationally) hides the special index i.

[2]In an independent update that was concurrent to this work, the most recent revision of their work [BKP+23b] also show how to construct computational NIZK arguments from batch arguments and one-way functions. We discuss the concurrent work at the end of this section.

Concurrent work. In a recent and concurrent update[3] to their original work (December 2023) [BKP+23b], Bitansky, Kamath, Paneth, Rothblum, and Vasudevan independently showed how to construct a computational NIZK argument for NP from a somewhere-sound non-interactive batch argument for NP and a one-way function. Notably, their construction only relies on polynomial-hardness of the underlying batch argument. The techniques in the two works are very different. We use a BARG to directly construct a hidden-bits generator (similar to [KMY20, CW23]), which implies a NIZK via existing transformations [QRW19, KMY20]. On the other hand, [BKP+23b] starts with a direct construction of a NIZK argument from a BARG which satisfies *distributional* zero-knowledge and has inverse polynomial zero-knowledge error. To obtain a full-fledged NIZK for NP with negligible zero-knowledge error, [BKP+23b] takes a gate-by-gate approach followed by a privacy-amplification step (using multiparty computation techniques).

In another independent and concurrent work [Mat23], Matsuda shows how to use a BARG to generically upgrade any CPA-secure public-key encryption scheme into a CCA-secure scheme. As noted above, such a transformation follows as an immediate corollary of Theorem 1.1. However, the approach from [Mat23] can be instantiated with any BARG with proof size $t^\varepsilon \cdot \mathsf{poly}(\lambda, s)$, where t is the number of instances, s is the circuit size, and *any* constant $0 < \varepsilon < 1$. Our construction will rely on a BARG with proof size $t^\varepsilon \cdot \mathsf{poly}(\lambda, s)$ for *sufficiently small constant* ε. Many existing BARG constructions [KVZ21, CJJ21b, HJKS22, WW22, DGKV22, GSWW22, CGJ+23, KLVW23] achieve succinctness $\mathsf{poly}(\lambda, s, \log t)$, which satisfy both sets of requirements.

1.1 Technical Overview

Our construction follows the approach from [KMY20, CW23] of using an (adaptively-sound) SNARG [KMY20] or a (somewhere-sound) BARG [CW23] to construct a hidden-bits generator [QRW19]. In conjunction with a NIZK in the ideal hidden bits model [FLS90], this yields a NIZK in the common reference string (CRS) model. We start with a brief description of the hidden-bits model and the [CW23] construction, which is the starting point of this work.

The hidden-bits model. The hidden-bits model [FLS90] is an *idealized* model for constructing *unconditional* NIZK proofs. In this model, a trusted party samples a uniform *random* sequence of bits $r_1, \ldots, r_m \xleftarrow{\mathrm{R}} \{0,1\}$. The trusted party gives $\mathbf{r} = r_1 r_2 \cdots r_m$ to the prover. To construct a proof for the statement x, the prover chooses a subset of indices $I \subseteq [m]$ and a proof string π. The trusted party then gives the bits $\mathbf{r}_I \coloneqq \{r_i\}_{i \in I}$ and the proof string π to the verifier. The work [FLS90] shows how to construct a NIZK for NP with statistical soundness and perfect zero-knowledge in the hidden-bits model.

Hidden-bits generators. Many works have shown how to leverage cryptographic tools to transform a NIZK in the hidden-bits model into a NIZK in the CRS model [FLS90, BY92, CHK03, GR13, CL18, QRW19, LPWW20, KMY20, CW23, Wat24, WWW24]. Similar to previous work [KMY20, CW23], we use the abstraction based on the hidden-bits generators introduced by Quach, Rothblum, and Wichs [QRW19]. Our presentation here is adapted from that in [CW23]. A hidden-bits generator allows a prover to generate a sequence of (pseudorandom) bits $\mathbf{r} = r_1 r_2 \cdots r_m$ and selectively reveal a subset of the bits $\mathbf{r}_I \coloneqq \{r_i\}_{i \in I}$ for some $I \subseteq [m]$ to the verifier. Specifically, a hidden-bits generator consists of four algorithms $(\mathsf{Setup}, \mathsf{GenBits}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:

- The Setup algorithm takes the security parameter λ and the hidden-bits string length m, and outputs a common reference string crs for the hidden-bits generator.
- The $\mathsf{GenBits}$ algorithm takes the common reference string crs and outputs a hidden-bits string $\mathbf{r} \in \{0,1\}^m$ and a generator state st.
- The Prove algorithm takes the generator state st, a subset of indices $I \subseteq [m]$, and outputs a proof π for the subset of bits $\mathbf{r}_I \coloneqq \{x_i\}_{i \in I}$.

[3] The original version of their work (May 2023) [BKP+23c] showed how to construct a statistical NIZK argument with a *non-uniform* prover from somewhere-sound BARGs and lossy public-key encryption. In a subsequent revision (June 2023) [BKP+23a], they improved their result to obtain a statistical NIZK argument with a *uniform* prover from somewhere-sound BARGs and lossy public-key encryption. In the most recent revision (December 2023) [BKP+23b], they additionally showed how to obtain a computational NIZK argument from somewhere-sound BARGs. These works also show additional implications between batch arguments and (statistical) witness indistinguishability.

- The Verify algorithm takes the common reference string crs, a subset of indices $I \subseteq [m]$, a subset of bits $\mathbf{r}_I = \{x_i\}_{i \in I}$, the opening π, and decides whether to accept or reject.

The correctness and security properties for a hidden-bits generator are defined as follows:

- **Correctness:** Correctness says that the verification algorithm accepts the proof output by Prove. Namely, if we sample $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs})$, then for all $I \subseteq [m]$, $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \mathsf{Prove}(\mathsf{st}, I)) = 1$.
- **Binding:** The binding property says that the set of valid hidden-bits strings (of length m) constitutes a *sparse* subset of the set of all m-bit strings. Namely, for every common reference string crs in the support of Setup, there exists a sparse subset $\mathcal{V}^{\mathsf{crs}} \subset \{0,1\}^m$ where $|\mathcal{V}^{\mathsf{crs}}| \le 2^{m^\gamma \cdot \mathsf{poly}(\lambda)}$ for some constant $\gamma < 1$. Moreover, an efficient adversary can only come up with valid proofs π for sequences $\mathbf{r}_I \in \{0,1\}^{|I|}$ where $\mathbf{r}_I = \mathbf{r}'_I$ for some $\mathbf{r}' \in \mathcal{V}^{\mathsf{crs}}$.
- **Hiding:** The hiding property says that the unrevealed bits are pseudorandom. Specifically, for any set $I \subseteq [m]$ and sampling $\mathbf{r} \leftarrow \mathsf{GenBits}(\mathsf{crs})$, the distribution of the unrevealed bits $\mathbf{r}_{\bar{I}}$ (where $\bar{I} = [m] \setminus I$) is computationally indistinguishable from uniform given the common reference string crs, the revealed bits $\mathbf{r}_I$, and the proof π.

The Champion-Wu hidden-bits generator. Building on the prior work of [KMY20], Champion and Wu [CW23] recently showed how to construct a hidden-bits generator from a batch argument for NP, a (leakage-resilient) local PRG (i.e., a PRG where each output bit depends on a small number of bits of the seed), and a dual-mode commitment scheme. In a dual-mode commitment [DN02], the CRS can be sampled in one of two computationally indistinguishable modes. One mode yields equivocable (or statistically hiding) commitments while the other yields extractable (or statistically binding) commitments. A dual-mode commitment can be built from a *lossy* public-key encryption scheme [BHY09]. We provide a basic outline of the [CW23] construction below:

- The common reference string consists of a CRS for a dual-mode commitment scheme and a CRS for a somewhere-sound BARG for NP.
- To sample a hidden-bits string, the prover samples a seed $\mathbf{s} \in \{0,1\}^n$ for the leakage-resilient local PRG.[4] The hidden-bits string $\mathbf{r} \in \{0,1\}^m$ is the output of $\mathbf{r} := \mathsf{PRG}(\mathbf{s})$. To open to a subset $I \subseteq [m]$, the prover does the following:
  - Commit to the PRG seed $\mathbf{s}$ using the dual-mode commitment. Let σ be the resulting commitment.
  - The prover constructs a BARG proof π that for each output index $i \in I$, the output bit r_i is consistent with the i^{th} bit of $\mathsf{PRG}(\mathbf{s})$, where $\mathbf{s}$ is the seed associated with the commitment σ.

  The proof consists of the commitment σ together with the BARG proof π.
- To check the opening, the verifier simply checks the BARG proof π (with respect to the committed seed σ).

To argue binding and hiding security, the analysis in [CW23] proceeds as follows:

- **Binding:** To argue binding, [CW23] programs the CRS for the dual-mode commitment to be extracting. If the local PRG has super-linear stretch, then the image of the PRG is a sparse subset of $\{0,1\}^m$. Moreover, somewhere-soundness of the BARG ensures that the prover can only open to bit-strings that are consistent with *some* seed (specifically, the seed $\mathbf{s}$ associated with the commitment σ). In the security proof, this latter step relies on the reduction being able efficiently extract the PRG seed $\mathbf{s}$ from the commitment σ. Namely, if there is an index i where r_i does not match the i^{th} bit of $\mathsf{PRG}(\mathbf{s})$, then the instance associated with the i^{th} output bit r_i in the BARG must be false. This is sufficient to setup a reduction to somewhere soundness of the BARG.

[4]Technically, the construction in [CW23] composes an arbitrary (sub-exponentially-secure) local PRG with a randomness extractor. Using the Gentry-Wichs leakage-simulation lemma [GW11] and assuming sub-exponential hardness of the local PRG, this yields a leakage-resilient local PRG. For ease of exposition, we describe their blueprint assuming a leakage-resilient local PRG.

- **Hiding:** To argue that the scheme is hiding, [CW23] first switches the dual-mode commitments to be equivocating (i.e., in this case, the commitment σ completely hides the seed $\mathbf{s}$). Then, it treats the BARG proof π as "leakage" on the PRG seed $\mathbf{s}$. As long as π is much shorter than $\mathbf{s}$, they can appeal to leakage-resilience of the local PRG to argue that the unrevealed bits remain pseudorandom. Since the length of π scales with the size of the circuit that computes each bit of the PRG output, [CW23] requires that each output bit of the PRG be computed by a circuit that is significantly shorter than the length of the PRG seed. This is why they require the PRG to have small locality.

Our approach. In this work, we make two key modifications to the previous construction of [CW23] that eliminates the need for both the dual-mode commitment as well as the local PRG. We describe these two techniques below:

- **Removing locality by committing to internal wires.** As noted above, the [CW23] approach assumed a local PRG because they needed to ensure that the size of the circuit computing the PRG is much smaller than the length of the PRG seed. They do this because each instance of the BARG is associated with one of the output bits of the PRG. In this work, we take a different approach. Instead of just committing to the bits of the PRG seed $\mathbf{s}$ (as in [CW23]), we instead commit to *all* of the wires in the circuit computing $\mathsf{PRG}(\mathbf{s})$. The BARG is then used to check not only the validity of $\mathbf{r}_I$ where $\mathbf{r} := \mathsf{PRG}(\mathbf{s})$, but all of the internal gates in the computation of $\mathsf{PRG}(\mathbf{s})$. In this setting, each statement in the BARG only needs to check correct computation of an individual *gate* (with respect to the committed wire values). The size of the circuit depends only on the security parameter for the *commitment* scheme (which can be set independently of the seed length of the PRG). As such, we no longer need to assume locality of the PRG. In particular, we construct the leakage-resilient PRG from a leakage-resilient weak pseudorandom function (PRF), which can in turn be based solely on (polynomial-hard) one-way functions [HLWW13, QWW21].

- **One-time dual-mode commitments.** A closer examination of the [CW23] construction shows that *one-time* equivocation suffices for the security analysis.[5] Specifically, one-time equivocation for a (bit) commitment scheme means that it is possible to jointly sample a common reference string along with a single commitment $\tilde{c}$ and openings $\tilde{\sigma}_0, \tilde{\sigma}_1$ of $\tilde{c}$ to the bits 0 and 1, respectively. One-time equivocable (bit) commitments follow from the classic bit-commitment scheme of Naor [Nao89], which is based on one-way functions.

Using these techniques, we now obtain the following two instantiations (which correspond to the two main implications in Theorem 1.1):

- **A construction based on one-way functions.** While Naor's dual-mode bit-commitment scheme is either statistically binding or one-time equivocable, it does not *support* an efficient extracting mode (i.e., we do not have an efficient extraction algorithm that takes as input an extraction trapdoor and a commitment and outputs the committed value). Note that lack of extraction is not surprising since an extractable commitment would imply a public-key encryption scheme. In this work, we provide an alternative proof of binding that relies on *adaptive* soundness of the BARG (as opposed to somewhere soundness). We then show that using complexity leveraging, any sub-exponentially-secure somewhere-sound BARG is also adaptively sound. While simple, this latter transformation may also be of independent interest; we describe this in Section 5. Taken together, this yields a hidden-bits generator from any sub-exponentially-secure somewhere-sound BARG for NP, and correspondingly, a NIZK for NP from the same assumption. We describe this construction in Section 3.

- **A construction based on public-key encryption.** By composing Naor's bit commitment scheme with a public-key encryption scheme, we also show how to construct a one-time dual-mode commitment with an efficient (trapdoor) extraction algorithm as well as a one-time equivocation mode. Similar ideas were used in a number of works studying CCA-secure encryption [KW19] and designated-verifier NIZKs [LQR+19]. Combined with the blueprint above, we obtain a hidden-bits generator from any polynomially-secure somewhere-sound BARG for NP and a semantically-secure public-key encryption scheme. This yields a NIZK for NP from the same set of assumptions. We describe this construction in Section 4.

[5] As shown in [QRW19, KMY20], this suffices for single-theorem zero-knowledge, which can be boosted to multi-theorem zero-knowledge using the classic transformation of [FLS90].

2 Preliminaries

We write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. We write $\mathsf{poly}(\lambda)$ to denote a fixed function that is $O(\lambda^c)$ for some $c \in \mathbb{N}$ and $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$. We say an event occurs with overwhelming probability if its complement occurs with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. For a finite set S, we write $x \xleftarrow{R} S$ to denote a uniform random draw from S. When $\mathcal{D}$ is a probability distribution, we write $x \leftarrow \mathcal{D}$ to denote a sample from $\mathcal{D}$. We say that two ensembles of distributions $\mathcal{D}_1 := \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_2 := \{\mathcal{D}_{2,\lambda}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if no efficient adversary can distinguish them except with $\mathsf{negl}(\lambda)$ probability. We say they are statistically indistinguishable if their statistical distance is negligible. Throughout this work, we consider security against non-uniform adversaries.

Basic cryptographic primitives. We now recall the definitions of some standard cryptographic primitives.

Definition 2.1 (Public-Key Encryption)**.** A public-key bit-encryption scheme with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ is a triple of efficient algorithms $\Pi_{\mathsf{PKE}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a public key pk and secret key sk.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: On input the public key pk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On input the secret key sk and a ciphertext ct, the decryption algorithm outputs a message $m \in \mathcal{M}$.

We require Π_{PKE} to satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}$,
$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m) \end{array}\right] = 1.$$
- **Semantic security:** For a security parameter λ and a bit $\beta \in \{0, 1\}$, we define the semantic security game between an adversary $\mathcal{A}$ and a challenger as follows:
  1. The challenger samples a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives $(1^\lambda, \mathsf{pk})$ to $\mathcal{A}$.
  2. Algorithm $\mathcal{A}$ outputs two message $m_0, m_1 \in \mathcal{M}_\lambda$. The challenger replies with the ciphertext $\mathsf{ct}_\beta \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_\beta)$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  Then Π_{PKE} satisfies semantic security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$|\Pr[b' = 1 \mid \beta = 0] - \Pr[b' = 1 \mid \beta = 1]| = \mathsf{negl}(\lambda).$$

Leakage-resilient weak PRF. We now introduce the definition of a leakage-resilient weak pseudorandom function (PRF). First, recall that the weak PRF security game asserts that the PRF evaluations on *random* inputs are pseudorandom. We say a weak PRF is leakage-resilient if this pseudorandomness property holds even if the adversary gets *arbitrary* leakage on the PRF key. In this work, we consider a definition where the adversary is first allowed to request (an arbitrary polynomial number of) random evaluations of the weak PRF before specifying its leakage function. We give the formal definition below. Our definition is adapted from that of [HLWW13].

Definition 2.2 (Leakage-Resilient Weak Pseudorandom Function [HLWW13, adapted])**.** Let $\mathcal{Y}$ be a finite set. A leakage-resilient weak pseudorandom function with output space $\mathcal{Y}$ is a pair of efficient algorithms $\Pi_{\mathsf{LRwPRF}} = (\mathsf{Setup}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to k$: On input the security parameter $\lambda \in \mathbb{N}$ and a leakage parameter $\ell \in \mathbb{N}$, the setup algorithm outputs a key k. We assume that k implicitly contains the security parameter 1^λ, the leakage parameter 1^ℓ, and defines the domain $\mathcal{X}$ of the PRF. Let $\kappa = \kappa(\lambda, \ell)$ be the bit-length of the key k output by $\mathsf{Setup}(1^\lambda, 1^\ell)$.

- $\mathsf{Eval}(k, x) \to y$: On input a key k (which specifies the domain $\mathcal{X}$ of the PRF) and an input $x \in \mathcal{X}$, the evaluation algorithm outputs a value $y \in \mathcal{Y}$. The evaluation algorithm is deterministic.

For a security parameter λ and bit $\beta \in \{0, 1\}$, we define the leakage-resilient weak pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:

1. **Pre-challenge evaluation queries:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s it would like to make. The challenger samples a key $k \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. Let $\mathcal{X}$ be the domain of the PRF associated with k. The challenger samples inputs $x_1, \ldots, x_s \xleftarrow{\text{R}} \mathcal{X}$ and replies to $\mathcal{A}$ with $\{(x_i, \mathsf{Eval}(k, x_i))\}_{i \in [s]}$.

2. **Leakage query:** Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak}\colon \{0,1\}^{\kappa(\lambda,\ell)} \to \{0,1\}^\ell$. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.

3. **Challenge queries:** Algorithm $\mathcal{A}$ then outputs the number of challenge queries 1^t it would like to make.

   - If $\beta = 0$, the challenger samples $x_i' \xleftarrow{\text{R}} \mathcal{X}$, $y_i \leftarrow \mathsf{Eval}(k, x_i')$ for each $i \in [t]$.
   - If $\beta = 1$, the challenger samples $x_i' \xleftarrow{\text{R}} \mathcal{X}$, $y_i \xleftarrow{\text{R}} \mathcal{Y}$ for each $i \in [t]$.

   The challenger gives $\{(x_i', y_i)\}_{i \in [t]}$ to $\mathcal{A}$.

4. **Output:** Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

Finally, we say Π_{LRwPRF} satisfies leakage-resilient weak pseudorandomness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

The work of [QWW21] show how to construct a leakage-resilient weak PRF from one-way functions where the size of the key scales with $\ell \cdot \mathsf{poly}(\lambda)$ and ℓ is the leakage parameter. Technically, [QWW21] show how to construct a leakage-resilient symmetric encryption scheme, but their construction implicitly uses a leakage-resilient weak PRF. For completeness, we provide the details and analysis of their construction (adapted to the setting of leakage-resilient weak PRFs) in Appendix A. Below, we state the main conclusion (derived from Construction A.6):

Theorem 2.3 (Leakage-Resilient Weak PRF from One-Way Functions [QWW21, adapted])**.** *Let λ be a security parameter and ℓ be a leakage parameter. Suppose $\rho \geq O(\lambda + \log \ell)$. Assuming the existence of one-way functions, there exists a leakage-resilient weak PRF with domain $\{0,1\}^\rho$, range $\{0,1\}$, and key length $(\ell + \lambda)\lambda$.*

One-time dual-mode bit commitment. Next, we recall the notion of a one-time dual-mode bit commitment scheme. This is a bit commitment scheme where the common reference string can be sampled in one of two computationally indistinguishable modes: binding mode and equivocable mode. When the CRS is in binding mode, the commitment scheme is statistically binding. When the CRS is sampled in equivocable mode, the CRS sampling algorithm outputs an equivocable commitment $\tilde{c}$ together with two openings $\tilde{\sigma}_0, \tilde{\sigma}_1$ of $\tilde{c}$ to the bits 0 and 1, respectively. In other words, the special commitment $\tilde{c}$ is an equivocable commitment that can be efficiently opened to a 0 and a 1. We now give the formal definition. Naor's classic bit commitment scheme [Nao89] based on one-way functions is a one-time dual-mode bit commitment scheme.

Definition 2.4 (One-Time Dual-Mode Bit Commitment)**.** A one-time dual-mode bit commitment is a tuple of algorithms $\Pi_{\mathsf{BC}} = (\mathsf{SetupBind}, \mathsf{SetupEquivocate}, \mathsf{Commit}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{SetupBind}(1^\lambda) \to \mathsf{crs}$: On input the security parameter λ, the setup algorithm for the binding mode outputs a common reference string crs.

- $\mathsf{SetupEquivocate}(1^\lambda) \to (\mathsf{crs}, \tilde{c}, \tilde{\sigma}_0, \tilde{\sigma}_1)$: On input the security parameter λ, the setup algorithm for the equivocating mode outputs a common reference string crs along with a commitment $\tilde{c}$ and openings $\tilde{\sigma}_0, \tilde{\sigma}_1$.
- $\mathsf{Commit}(\mathsf{crs}, b) \to (c, \sigma)$: On input the common reference string crs and a bit $b \in \{0,1\}$, the commit algorithm outputs a commitment c and an opening σ.
- $\mathsf{Verify}(\mathsf{crs}, c, b, \sigma) \to \{0,1\}$: On input the common reference string crs, a commitment c, a bit $b \in \{0,1\}$, and an opening σ, the verification algorithm outputs a bit $b' \in \{0,1\}$.

We require Π_{BC} to satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all common reference strings crs in the support of either $\mathsf{SetupBind}(1^\lambda)$ or $\mathsf{SetupEquivocate}(1^\lambda)$, and all bits $b \in \{0,1\}$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, c, b, \sigma) = 1 : (c, \sigma) \leftarrow \mathsf{Commit}(\mathsf{crs}, b)\right] = 1.$$
- **Mode indistinguishability:** For a security parameter $\lambda \in \mathbb{N}$ and a bit $\beta \in \{0,1\}$, we define the mode indistinguishability game between an adversary $\mathcal{A}$ and a challenger as follows:
  1. If $\beta = 0$, the challenger samples $\mathsf{crs} \leftarrow \mathsf{SetupBind}(1^\lambda)$. If $\beta = 1$, the challenger samples $(\mathsf{crs}, \tilde{c}, \tilde{\sigma}_0, \tilde{\sigma}_1) \leftarrow \mathsf{SetupEquivocate}(1^\lambda)$. The challenger gives $(1^\lambda, \mathsf{crs})$ to $\mathcal{A}$.
  2. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. The challenger gives c, σ to $\mathcal{A}$, where (c, σ) are computed as follows:
     - If $\beta = 0$, $(c, \sigma) \leftarrow \mathsf{Commit}(\mathsf{crs}, b)$.
     - If $\beta = 1$, $(c, \sigma) \leftarrow (\tilde{c}, \tilde{\sigma}_b)$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

  The bit commitment scheme satisfies mode indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\left|\Pr[b' = 1 \mid \beta = 0] - \Pr[b' = 1 \mid \beta = 1]\right| = \mathsf{negl}(\lambda).$$
- **Statistical binding in binding mode:** For all security parameters $\lambda \in \mathbb{N}$ and all (not necessarily efficient) adversaries $\mathcal{A}$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, c, 0, \sigma_0) = 1 = \mathsf{Verify}(\mathsf{crs}, c, 1, \sigma_1) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{SetupBind}(1^\lambda) \\ (c, \sigma_0, \sigma_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Theorem 2.5 (Bit Commitment from One-Way Functions [Nao89, adapted])**.** *Assuming the existence of one-way functions, there exists a one-time dual-mode bit commitment scheme.*

2.1 Cryptographic Proof Systems

In this section, we recall the definition of a non-interactive zero-knowledge (NIZK) argument for NP as well as that of a non-interactive batch argument (BARG) for NP. We will often consider the language of Boolean circuit satisfiability, which we recall below:

Definition 2.6 (Boolean Circuit Satisfiability)**.** The language $\mathcal{L}_{\mathsf{SAT}}$ of Boolean circuit satisfiability consists of pairs $(C, \mathbf{x})$ of circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and inputs $\mathbf{x} \in \{0,1\}^n$ such that there exists $\mathbf{w} \in \{0,1\}^h$ where $C(\mathbf{x}, \mathbf{w}) = 1$:
$$\mathcal{L}_{\mathsf{SAT}} = \left\{(C, \mathbf{x}) : \begin{array}{c} C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}, \mathbf{x} \in \{0,1\}^n \\ \exists \mathbf{w} \in \{0,1\}^h : C(\mathbf{x}, \mathbf{w}) = 1 \end{array}\right\}.$$

Non-interactive zero-knowledge. We now recall the notion of a non-interactive zero-knowledge argument [GMR85, BFM88] for an arbitrary NP language.

Definition 2.7 (NIZK Argument for NP)**.** A non-interactive zero-knowledge argument for an NP relation $\mathcal{R}$ (with associated language $\mathcal{L}$) is a tuple of efficient algorithms $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w}) \to \pi$: On input the common reference string crs, a statement $\mathbf{x}$, and a witness $\mathbf{w}$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, \mathbf{x}, \pi) \to b$: On input the common reference string crs, a statement $\mathbf{x}$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

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

- **Completeness:** For all $\lambda \in \mathbb{N}$ and all $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathbf{x}, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w}) \end{array}\right] = 1.$$
- **Adaptive computational soundness:** For all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\Pr\left[\mathbf{x} \notin \mathcal{L} \wedge \mathsf{Verify}(\mathsf{crs}, \mathbf{x}, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbf{x}, \pi) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda).$$
- **Adaptive multi-theorem computational zero-knowledge:** For every efficient adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ and a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and sampling $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\widetilde{\mathsf{crs}}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$, we have that
$$\left|\Pr\left[\mathcal{A}^{O_0(\mathsf{crs},\cdot,\cdot)}(1^\lambda, \mathsf{crs}) = 1\right] - \Pr\left[\mathcal{A}^{O_1(\mathsf{st}_{\mathcal{S}},\cdot,\cdot)}(1^\lambda, \widetilde{\mathsf{crs}}) = 1\right]\right| = \mathsf{negl}(\lambda),$$
and where the oracles O_0 and O_1 are defined as follows:
  - $O_0(\mathsf{crs}, \mathbf{x}, \mathbf{w})$: On input the common reference string crs, a statement $\mathbf{x}$, and a witness $\mathbf{w}$, the oracle outputs $\perp$ if $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$. If $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, it outputs $\mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w})$.
  - $O_1(\mathsf{st}_{\mathcal{S}}, \mathbf{x}, \mathbf{w})$: On input the simulator state $\mathsf{st}_{\mathcal{S}}$, a statement $\mathbf{x}$ and a witness $\mathbf{w}$, the oracle outputs $\perp$ if $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$. If $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, it outputs $\mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, \mathbf{x})$.

Non-interactive batch arguments. Next, we recall the definition of a non-interactive batch argument for the language of Boolean circuit satisfiability. We start by defining the standard notion of *adaptive* (computational) soundness, and then follow it with a relaxation called somewhere soundness [CJJ21b, KVZ21].

Definition 2.8 (Batch Argument for NP [CJJ21b, adapted])**.** A non-interactive batch argument (BARG) for Boolean circuit satisfiability is a tuple of three efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^T, 1^s) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, a bound on the number of instances $T \in \mathbb{N}$, and a bound on the circuit size $s \in \mathbb{N}$, the setup algorithm outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), (\mathbf{w}_1, \ldots, \mathbf{w}_t)) \to \pi$: On input the common reference string crs, a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \{0,1\}^h$, the prove algorithm outputs a proof π.

- $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) \to b$: On input the common reference string crs, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$ and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

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

- **Completeness:** For all $\lambda, T, s \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, all $t \leq T$, all statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and all witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \{0,1\}^h$ where $C(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all $i \in [t]$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^T, 1^s); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), (\mathbf{w}_1, \ldots, \mathbf{w}_t)) \end{array}\right] = 1.$$

- **Succinct proof size:** There exists a polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, T, s \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^T, 1^s)$, and all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, all $t \leq T$, all statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and all witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \{0,1\}^h$, the size of the proof π output by $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), (\mathbf{w}_1, \ldots, \mathbf{w}_t))$ satisfies $|\pi| \leq \mathsf{poly}(\lambda + \log t + s)$.

- **Adaptive soundness:** For a security parameter λ, we define the adaptive security game between an adversary $\mathcal{A}$ and a challenger as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the bound on the number of instances 1^T and the bound on the circuit size 1^s.
  2. The challenger samples a common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^T, 1^s)$ and gives 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, statements $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ where $\mathbf{x}_i \in \{0,1\}^n$ for all $i \in [t]$ and where $t \leq T$, and a proof π.
  4. The output of the experiment is 1 if $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$ and for some $i \in [t]$, $(C, \mathbf{x}_i) \notin \mathcal{L}_{\mathsf{SAT}}$. Otherwise, the output is 0.

  Then Π_{BARG} satisfies adaptive security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the adaptive security game.

Definition 2.9 (Somewhere-Sound Batch Argument for NP [CJJ21b, adapted])**.** A somewhere-sound non-interactive batch argument (BARG) for Boolean circuit satisfiability is a tuple of three efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. The Prove and Verify algorithms are defined exactly as in Definition 2.8, while the Setup algorithm now takes an additional index parameter:

- $\mathsf{Setup}(1^\lambda, 1^T, 1^s, i) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, a bound on the number of instances $T \in \mathbb{N}$, a bound on the circuit size $s \in \mathbb{N}$, and an index $i \in [T]$, the setup algorithm outputs a common reference string crs.

Moreover, the adaptive soundness property from Definition 2.8 is replaced by the following index hiding and somewhere soundness properties:

- **Index hiding:** For a security parameter λ and a bit $\beta \in \{0,1\}$, we define the index-hiding game between an adversary $\mathcal{A}$ and a challenger as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the bound on the number of instances 1^T, the bound on the circuit size 1^s, and a pair of indices $i_0, i_1 \in [T]$.
  2. The challenger samples a common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^T, 1^s, i_\beta)$ and gives crs to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

  Then Π_{BARG} satisfies index hiding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$|\Pr[b' = 1 \mid \beta = 0] - \Pr[b' = 1 \mid \beta = 1]| = \mathsf{negl}(\lambda).$$

We say Π_{BARG} satisfies sub-exponential index hiding security if there exists some constant $c > 1$ such that for all adversaries $\mathcal{A}$ running in time $2^{\lambda^{1/c}} \cdot \mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

- **Somewhere soundness:** For a security parameter λ, we define the somewhere-soundness game between an adversary $\mathcal{A}$ and a challenger as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the bound on the number of instances 1^T, the bound on the circuit size 1^s, and the index $i^* \in [T]$.
  2. The challenger samples a common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^T, 1^s, i^*)$ and gives 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, statements $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ where $\mathbf{x}_i \in \{0,1\}^n$ for all $i \in [t]$ and $t \le T$, and a proof π.
  4. The output of the experiment is $b = 1$ if $i^* \le t$, $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$, and $(C, \mathbf{x}_i^*) \notin \mathcal{L}_{\mathsf{SAT}}$. Otherwise, the output is $b = 0$.

  Then Π_{BARG} satisfies somewhere-soundness if for all efficient adversaries $\mathcal{A}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the somewhere-soundness game.

Constructions of BARGs. A number of recent works have established the existence of BARGs for NP from standard number-theoretic assumptions including the learning with errors (LWE) assumption [CJJ21b, DGKV22], the k-Lin assumption over groups with bilinear maps [WW22], and (sub-exponential) hardness of decisional Diffie-Hellman (DDH) over pairing-free groups [CGJ+23].

Soundness definitions. Somewhere soundness is a relaxation of adaptive soundness where the adversary has the ability to choose the statements based on the CRS, but it is required to pre-commit to the index of a false statement. In Section 5 (Theorem 5.4), we show that using complexity leveraging, any somewhere-sound BARG which satisfies sub-exponential index hiding implies an adaptively-sound BARG. Namely, if index hiding security of the somewhere-sound BARG holds against an adversary that is able to decide the underlying NP language, then somewhere soundness implies adaptive soundness by a simple guessing argument. As a security notion, adaptive soundness is sometimes more convenient to work with compared to somewhere soundness. As such, we use adaptive soundness in our one-way function based construction in Section 3, which yields a construction based on sub-exponentially secure somewhere-sound BARGs. For our construction based on public-key encryption (Section 4), somewhere soundness suffices for our security analysis and we avoid the need for complexity leveraging. This yields a construction based only on polynomial hardness, but additionally relies on public-key encryption.

2.2 Hidden-Bits Generator

In this section, we recall the notion of a hidden-bits generator with subset-dependent proofs from [QRW19, KMY20]. For a bitstring $\mathbf{r} \in \{0,1\}^n$ and a set of indices $I \subseteq [n]$, we write $\mathbf{r}_I \in \{0,1\}^{|I|}$ to denote the substring corresponding to the bits of $\mathbf{r}$ indexed by I. Our presentation here is adapted from the work of [CW23].

Definition 2.10 (Hidden-Bits Generator [KMY20, Definition 11])**.** A hidden-bits generator with subset-dependent proofs is a tuple of efficient algorithms $\Pi_{\mathsf{HBG}} = (\mathsf{Setup}, \mathsf{GenBits}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^m) \to \mathsf{crs}$: On input the security parameter λ, and the output length m, the setup algorithm outputs a common reference string crs.

- $\mathsf{GenBits}(\mathsf{crs}) \to (\mathbf{r}, \mathsf{st})$: On input the the common reference string crs, the generator algorithm outputs a string $\mathbf{r} \in \{0,1\}^m$ and a state st.

- $\mathsf{Prove}(\mathsf{st}, I) \to \pi$: On input the state st and a subset $I \subseteq [m]$, the prove algorithm outputs a proof π.

- $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) \to b$: On input a common reference string crs, a subset $I \subseteq [m]$, a string $\mathbf{r}_I \in \{0,1\}^{|I|}$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require Π_{HBG} to satisfy the following properties:

- **Correctness:** For all $\lambda, m \in \mathbb{N}$ and all subsets $I \subseteq [m]$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m); \\ (\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs}); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{st}, I) \end{array}\right] = 1.$$

- **Somewhat computational binding:** For every crs in the support of the algorithm $\mathsf{Setup}(1^\lambda, 1^m)$, there exists a set $\mathcal{V}^{\mathsf{crs}}$ with the following properties:
  (i) **Output sparsity.** There exists a universal constant $\gamma < 1$ and a fixed polynomial $p(\cdot)$ such that for every polynomial $m = m(\lambda)$, there exists $\lambda_m \in \mathbb{N}$ such that for all $\lambda \geq \lambda_m$ and every crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, $|\mathcal{V}^{\mathsf{crs}}| \leq 2^{m^\gamma \cdot p(\lambda)}$
  (ii) **Computational binding.** For a security parameter λ, we define the computational binding game between an adversary $\mathcal{A}$ and a challenger as follows:
    (a) On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.
    (b) The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and gives crs to $\mathcal{A}$.
    (c) Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$.
    (d) The output of the experiment is $b = 1$ if $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{\mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}}\}$. Otherwise, the output is $b = 0$.

    We say the Π_{HBG} is computationally binding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the computational binding security game.

- **Computational hiding:** For a security parameter λ and bit $\beta \in \{0,1\}$, we define the computational hiding game between an adversary $\mathcal{A}$ and a challenger as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m and a subset $I \subseteq [m]$.
  2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$, $(\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs})$, $\pi \leftarrow \mathsf{Prove}(\mathsf{st}, I)$ and $\mathbf{r}' \xleftarrow{\mathrm{R}} \{0,1\}^m$.
     - If $\beta = 0$, the challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$, where $\bar{I} = [m] \setminus I$.
     - If $\beta = 1$, the challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}'_{\bar{I}})$ to $\mathcal{A}$ where $\bar{I} = [m] \setminus I$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

  We say the Π_{HBG} is computationally hiding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

Theorem 2.11 (NIZK from Hidden-Bits Generator [KMY20])**.** *If there exists a hidden-bits generator with subset-dependent proofs, then there exists a computational NIZK argument for* NP*. The NIZK argument satisfies adaptive computational soundness and adaptive multi-theorem zero knowledge.*

3 Hidden-Bits Generator from Adaptively-Sound BARGs and OWFs

In this section, we show how to construct a hidden-bits generator with subset-dependent proofs using an adaptively-sound batch argument for NP, a one-time dual-mode bit commitment scheme, and a leakage-resilient weak PRF. By Theorems 2.3 and 2.5, the dual-mode bit commitment scheme and the leakage-resilient weak PRF can be constructed from one-way functions. From Theorem 5.4, we can construct an adaptively-sound BARG for NP from any sub-exponentially-secure somewhere-sound BARG for NP. Invoking Theorem 2.11 now yields a NIZK for NP from any sub-exponentially-secure somewhere-sound BARG for NP (Corollary 3.5).

Boolean circuits. We start by describing the conventions we use for describing Boolean circuits. Let $C\colon \{0,1\}^n \to \{0,1\}$ be a Boolean circuit where each gate is a fan-in-2 NAND gate. Let s be the size of C, as measured by the number of wires in C. We associate an index $i \in [s]$ with each wire:

- **Input wires:** We index the n input wires with the values $1, \ldots, n$ and will refer to the wire at index $i \in [n]$ as the "i^{th} input wire" to C.
- **Internal wires:** Each non-input wire is associated with an index $i \in \{n+1, \ldots, s\}$ with the property that the value of wire i is completely determined by the value of the wires indexed $j_{i,\textsc{l}}, j_{i,\textsc{r}} \in \{1, \ldots, i-1\}$. Specifically, the value of wire i is the NAND of the value of its left input wire (i.e., the wire indexed $j_{i,\textsc{l}}$) and the value of its right input wire (i.e., the wire indexed $j_{i,\textsc{r}}$).
- **Output wire:** The output wire is associated with the index s.

Construction 3.1 (Hidden-Bits Generator from Batch Arguments)**.** Let $\lambda \in \mathbb{N}$ be a security parameter and $m \in \mathbb{N}$ be an output length parameter. Our construction depends on the following primitives:

- Let $\Pi_{\mathsf{LRwPRF}} = (\mathsf{LRwPRF.Setup}, \mathsf{LRwPRF.Eval})$ be a leakage-resilient weak PRF (Definition 2.2) with range $\{0,1\}$. Let $\ell = \ell(\lambda, m)$ be a leakage parameter which will be set according to the requirements of the security analysis (Theorems 3.3 and 3.4). Let $\kappa = \kappa(\lambda, m)$ be the bit-length of the keys output by $\mathsf{LRwPRF.Setup}(1^\lambda, 1^{\ell(\lambda,m)})$. Let $n = n(\lambda, m)$ be the bit-length of the domain of LRwPRF (when instantiated with security parameter λ and leakage parameter $\ell = \ell(\lambda, m)$).
- Let $C\colon \{0,1\}^{\kappa+n} \to \{0,1\}$ be the Boolean circuit that evaluates Π_{LRwPRF}. Namely, the circuit evaluates $C(k, \mathbf{z}) \coloneqq \mathsf{LRwPRF.Eval}(k, \mathbf{z})$. Let s be the size of C (i.e., the number of wires in C). In the following, we will define the following sets to refer to the wires in C:
  - Let $S_{\mathsf{key}} = \{1, \ldots, \kappa\}$ be the indices of the wires corresponding to the PRF key.
  - Let $S_{\mathsf{eval}} = \{\kappa+1, \ldots, \kappa+n\}$ be the indices of the wires corresponding to the evaluation point $\mathbf{z}$.
  - Let $S_{\mathsf{int}} = \{\kappa+n+1, \ldots, s\}$ be the indices of the non-input wires.
- Let $\Pi_{\mathsf{BC}} = (\mathsf{BC.SetupBind}, \mathsf{BC.SetupEquivocate}, \mathsf{BC.Commit}, \mathsf{BC.Verify})$ be a one-time dual-mode bit commitment scheme (Definition 2.4).
- Let $\Pi_{\mathsf{BARG}} = (\mathsf{BARG.Setup}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$ be an (adaptively-sound) batch argument for NP. Let $\ell_{\mathsf{BARG}} = \ell_{\mathsf{BARG}}(\lambda, T, s)$ denote a bound on the length of the proof output by Π_{BARG} as a function of the bound on the number of instances T and the size s of the associated NP relation.
- Define the NP relation $\mathcal{R}$ as follows:

  Statement: common reference string $\mathsf{crs}_{\mathsf{BC}}^{(\textsc{l})}, \mathsf{crs}_{\mathsf{BC}}^{(\textsc{r})}, \mathsf{crs}_{\mathsf{BC}}^{(\textsc{out})}$ and commitments $c_{\textsc{l}}, c_{\textsc{r}}, c_{\textsc{out}}$
  Witness: bits $b_{\textsc{l}}, b_{\textsc{r}}, b_{\textsc{out}} \in \{0,1\}$ and openings $\sigma_{\textsc{l}}, \sigma_{\textsc{r}}, \sigma_{\textsc{out}}$

  Output 1 if the following conditions hold:

  - For each $i \in \{\textsc{l}, \textsc{r}, \textsc{out}\}$, $\mathsf{BC.Verify}(\mathsf{crs}_{\mathsf{BC}}^{(i)}, c_i, b_i, \sigma_i) = 1$.
  - $b_{\textsc{out}} = \textsc{nand}(b_{\textsc{l}}, b_{\textsc{r}})$.

  Otherwise, output 0.

  Let $C_{\mathcal{R}}$ be the circuit computing the NP relation $\mathcal{R}$ and $\mathcal{L}$ be the associated NP language.

We construct our hidden-bits generator $\Pi_{\mathsf{HBG}} = (\mathsf{Setup}, \mathsf{GenBits}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^m)$: On input the security parameter λ and output length m, the setup algorithm start by sampling the following collection of common reference strings for the bit commitment scheme:

- **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
- **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
- **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

Next, the setup algorithm samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$. The setup algorithm commits to $\mathbf{z}_1, \ldots, \mathbf{z}_m$ as follows:

- For each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, compute $\left(c^{(i,j)}, \sigma^{(i,j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$.

Let $s_{\mathsf{int}} = |S_{\mathsf{int}}| = s - (\kappa + n)$ be the number of non-input wires in C. Sample a CRS for the BARG: $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{m s_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|})$. Output the common reference string

$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\mathsf{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right). \tag{3.1}$$

- $\mathsf{GenBits}(\mathsf{crs})$: On input the common reference string crs (parsed according to Eq. (3.1)), the generator algorithm samples a weak PRF key $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$ and computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$ for each $i \in [m]$. It outputs the hidden bits string $\mathbf{r} = r_1 \| \cdots \| r_m$ and the state $\mathsf{st} = (\mathsf{crs}, k)$.
- $\mathsf{Prove}(\mathsf{st}, I)$: On input the state $\mathsf{st} = (\mathsf{crs}, k)$ (where crs is parsed according to Eq. (3.1) and $k \in \{0,1\}^\kappa$) and a set of indices $I \subseteq [m]$, the prove algorithm proceeds as follows:
  - **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$, compute
  $$\left(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(\mathsf{key},j)}, k_j\right).$$
  - **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$, let $w_1^{(i)}, \ldots, w_s^{(i)}$ be the wire values of $C(k, \mathbf{z}_i)$. For each $i \in [m]$ and $j \in S_{\mathsf{int}}$, compute
  $$\left(c^{(i,j)}, \sigma^{(i,j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(i,j)}, w_j^{(i)}\right).$$
  - **Construct a BARG proof of validity:** Recall that $s_{\mathsf{int}} = |S_{\mathsf{int}}| = s - (\kappa + n)$ is the number of non-input wires in the circuit C. These indices associated with these wires are $\kappa + n + 1, \ldots, \kappa + n + s_{\mathsf{int}}$. Now, for each $i \in [m]$, define the following:
    * As before, let $w_1^{(i)}, \ldots, w_s^{(i)} \in \{0,1\}$ be the wire values of $C(k, \mathbf{z}_i)$.
    * For each $j \in [s_{\mathsf{int}}]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the wire indices that determine the value of the jth non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$. Define the statement $x^{(i,j)}$ and witness $w^{(i,j)}$ as follows:
    $$x^{(i,j)} = \left(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{OUT}})}, c^{(i,j_{\mathrm{L}})}, c^{(i,j_{\mathrm{R}})}, c^{(i,j_{\mathrm{OUT}})}\right) \tag{3.2}$$
    $$w^{(i,j)} = \left(w_{j_{\mathrm{L}}}^{(i)}, w_{j_{\mathrm{R}}}^{(i)}, w_{j_{\mathrm{OUT}}}^{(i)}, \sigma^{(i,j_{\mathrm{L}})}, \sigma^{(i,j_{\mathrm{R}})}, \sigma^{(i,j_{\mathrm{OUT}})}\right). \tag{3.3}$$

    Here, for all $i \in [m]$ and $j \in S_{\mathsf{key}}$, we adopt the convention that $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} := \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}$, $c^{(i,j)} := c^{(\mathsf{key},j)}$, and $\sigma^{(i,j)} := \sigma^{(\mathsf{key},j)}$.

  Construct the proof
  $$\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}\left(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, (w^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}\right). \tag{3.4}$$

  Output
  $$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right). \tag{3.5}$$
- $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi)$: On input a common reference string crs (parsed according to Eq. (3.1)), a subset $I \subseteq [m]$, a string $\mathbf{r}_I \in \{0,1\}^{|I|}$, and a proof $\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right)$, the verification algorithm checks the following conditions:

- **Validity of output commitments:** For all $i \in I$, check $\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,s)}, c^{(i,s)}, r_i, \sigma^{(i,s)}\big) = 1$.
- **Validity of BARG proof:** For each $i \in I$ and $j \in [s_{\mathsf{int}}]$, compute $x^{(i,j)}$ from crs and π according to Eq. (3.2). Then, check that $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, \pi_{\mathsf{BARG}}) = 1$.

If both checks pass, output 1. Otherwise, output 0.

Theorem 3.2 (Correctness). *If Π_{BARG} is complete and Π_{BC} is correct, then Construction 3.1 is correct.*

Proof. Let λ be a security parameter, m be an output length, and $I \subseteq [m]$ a set of indices. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Then,

$$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\big\}_{j \in S_{\mathsf{key}}}, \big\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\big\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \big\{\big(c^{(i,j)}, \sigma^{(i,j)}\big)\big\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}} \Big).$$

Let $(\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs})$ where $\mathsf{st} = (\mathsf{crs}, k)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{st}, I)$. Then,

$$\pi = \big(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\big).$$

For each $i \in I$, let $w_1^{(i)}, \ldots, w_s^{(i)} \in \{0,1\}$ be the wire values of $C(k, \mathbf{z}_i)$. Consider the output of $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi)$. First, we show that for every $i \in I$ and $j \in [s]$,

$$\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, w_j^{(i)}, \sigma^{(i,j)}\big) = 1. \tag{3.6}$$

We consider three cases:

- Suppose $j \in S_{\mathsf{key}}$. By definition, this means $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} = \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}$, $c^{(i,j)} = c^{(\mathsf{key},j)}$, $\sigma^{(i,j)} = \sigma^{(\mathsf{key},j)}$ and $w_j^{(i)} = k_j$. By construction, the Prove algorithm computes the bit-commitment and opening $(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, k_j)$, where the Setup algorithm sampled $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$. By correctness of Π_{BC},
$$\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, w_j^{(i)}, \sigma^{(i,j)}\big) = \mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}, k_j, \sigma^{(\mathsf{key},j)}\big) = 1.$$
- Suppose $j \in S_{\mathsf{eval}}$. Then, $w_j^{(i)} = z_{i,j-\kappa}$. By construction, the Setup algorithm samples $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa})$. Again, since $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$, it follows by correctness of Π_{BC} that
$$\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, w_j^{(i)}, \sigma^{(i,j)}\big) = \mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, z_{i,j-\kappa}, \sigma^{(i,j)}\big) = 1.$$
- Suppose $j \in S_{\mathsf{int}}$. By construction, the Prove algorithm computes the commitments and openings $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}^{(i,j)}, w_j^{(i)})$. Since $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$, by correctness of the bit-commitment Π_{BC},
$$\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, w_j^{(i)}, \sigma^{(i,j)}\big) = 1.$$

We now consider the two checks performed by Verify:

- **Validity of output commitments.** Let $i \in I$. The value of the output wire is $w_s^{(i)} = C(k, \mathbf{z}_i) = r_i$. From Eq. (3.6),
$$\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,s)}, c^{(i,s)}, r_i, \sigma^{(i,s)}\big) = \mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,s)}, c^{(i,s)}, w_s^{(i)}, \sigma^{(i,s)}\big) = 1.$$
- **Validity of BARG proof.** Take any index $i \in I$ and $j \in [s_{\mathsf{int}}]$. Let j_{L} and j_{R} be the wire indices that determine the value of the j^{th} non-input wire $j_{\mathsf{OUT}} = (\kappa + n) + j$. Let $x^{(i,j)}$ and $w^{(i,j)}$ be the statement and witness as defined in Eqs. (3.2) and (3.3):
$$\begin{aligned} x^{(i,j)} &= \Big(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathsf{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathsf{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathsf{OUT}})}, c^{(i,j_{\mathsf{L}})}, c^{(i,j_{\mathsf{R}})}, c^{(i,j_{\mathsf{OUT}})}\Big) \\ w^{(i,j)} &= \Big(w_{j_{\mathsf{L}}}^{(i)}, w_{j_{\mathsf{R}}}^{(i)}, w_{j_{\mathsf{OUT}}}^{(i)}, \sigma^{(i,j_{\mathsf{L}})}, \sigma^{(i,j_{\mathsf{R}})}, \sigma^{(i,j_{\mathsf{OUT}})}\Big). \end{aligned}$$

Now, the following conditions hold:

- By Eq. (3.6), $\mathsf{BC.Verify}(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{pos}})}, c^{(i,j_{\mathrm{pos}})}, w_{j_{\mathrm{pos}}}^{(i)}, \sigma^{(i,j_{\mathrm{pos}})}) = 1$ for each $\mathrm{pos} \in \{\textsc{l}, \textsc{r}, \textsc{out}\}$.
- Since the wire values $w_1^{(i)}, \ldots, w_s^{(i)}$ are associated with the wire values of $C(k, \mathbf{z}_i)$, it holds that $w_{j_{\textsc{out}}}^{(i)} = \textsc{nand}(w_{j_{\textsc{l}}}^{(i)}, w_{j_{\textsc{r}}}^{(i)})$.

By construction of $\mathcal{R}$, this means $(x^{(i,j)}, w^{(i,j)}) \in \mathcal{R}$ for each $i \in I$ and $j \in [s_{\mathrm{int}}]$. By completeness of Π_{BARG},

$$\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathrm{int}}]}, \pi_{\mathsf{BARG}}) = 1.$$

Since both checks pass, we conclude that $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi)$ outputs 1, as required. □

Security. We now state the main security theorems for Construction 3.1 and defer their formal proofs to Sections 3.1 and 3.2.

Theorem 3.3 (Somewhat Computational Binding)**.** *Suppose $\kappa(\lambda, m) \le m^{\delta} \cdot p(\lambda)$ for some constant $\delta < 1$ and a fixed polynomial $p(\cdot)$. Then, if Π_{BC} is statistically binding in binding mode and Π_{BARG} is adaptively sound, it follows that Construction 3.1 satisfies somewhat computational binding.*

Theorem 3.4 (Computational Hiding)**.** *Suppose Π_{BC} satisfies mode indistinguishability and Π_{LRwPRF} is a leakage-resilient weak PRF. If $\ell(\lambda, m) \ge \ell_{\mathsf{BARG}}(\lambda, m s_{\mathrm{int}}, |C_{\mathcal{R}}|)$, Construction 3.1 satisfies computational hiding.*

Parameter instantiations. We now describe a possible instantiation of the underlying building blocks in Construction 3.1 to obtain a NIZK argument from a sub-exponentially-secure somewhere-sound BARG. We summarize our main result in Corollary 3.5.

- We instantiate the one-time dual-mode bit commitment scheme Π_{BC} using Naor's bit commitment scheme [Nao89] (Theorem 2.5) based on one-way functions. With this instantiation, the size of the circuit $C_{\mathcal{R}}$ in Construction 3.1 can be bounded by $O(\lambda^{c_1})$ for some constant $c_1 \in \mathbb{N}$.

- We instantiate the leakage-resilient weak PRF Π_{LRwPRF} with the scheme based on one-way functions [HLWW13, QWW21] (Theorem 2.3). Let ℓ be the leakage parameter for the leakage-resilient weak PRF. With this instantiation, the keys have length at most $\kappa = \ell \cdot O(\lambda^2)$ and the inputs have length $n = O(\lambda + \log \ell)$. Let s be the size of the Boolean circuit that takes as input a key k and an input $\mathbf{z}$ and outputs $\mathsf{LRwPRF.Eval}(k, \mathbf{z})$. Since the construction is efficient, the size of this circuit can be upper-bounded by $s = O((\ell\lambda)^{c_2})$ for some constant $c_2 \in \mathbb{N}$.

- We instantiate the batch argument Π_{BARG} with a scheme where the proof size satisfies

$$\ell_{\mathsf{BARG}}(\lambda, T, s) < T^{\varepsilon} \cdot O((\lambda + s)^{c_3}),$$

  where $0 < \varepsilon < 1/(1+c_2)$ and $c_3 \in \mathbb{N}$ are fixed constants. Existing BARG constructions based on standard number-theoretic assumptions [CJJ21b, WW22, DGKV22, KLVW23, CGJ+23] satisfy an even stronger succinctness guarantee where $\ell_{\mathsf{BARG}}(\lambda, T, s) \le \mathrm{poly}(\lambda + \log T + s)$ where the proof size is *polylogarithmic* in the number of instances T; this is the default definition from Definition 2.8. However, as we show here, our construction applies even in settings where the BARG proof size scales with T^{ε} for sufficiently small constants $\varepsilon < 1$. Finally, all of the aforementioned BARG constructions satisfy somewhere soundness, which can be bootstrapped to an adaptively-sound construction using Theorem 5.4 (via complexity leveraging).

- With this choice of parameters, we choose constants $\delta_1 = \varepsilon/(1 - \varepsilon c_2)$ and $\delta_2 = (\varepsilon c_2 + c_1 c_3)/(1 - \varepsilon c_2)$. Finally, we set $\ell(\lambda, m) = m^{\delta_1} \cdot \Theta(\lambda^{\delta_2})$.

It is easy to see that this setting of parameters satisfies the requirements in Theorems 3.3 and 3.4:

- **Binding (Theorem 3.3):** For this choice of parameters,

  $$\kappa(\lambda, m) = \ell(\lambda, m) \cdot O(\lambda^2) = m^{\delta_1} \cdot \mathsf{poly}(\lambda) = m^{\varepsilon/(1-\varepsilon c_2)} \cdot \mathsf{poly}(\lambda).$$

  Moreover, since $\varepsilon < 1/(1 + c_2)$, this means $\varepsilon + \varepsilon c_2 < 1$ so $\varepsilon < 1 - \varepsilon c_2$. Correspondingly, this means that $0 < \varepsilon/(1 - \varepsilon c_2) < 1$. Thus, the condition of Theorem 3.3 is satisfied.

- **Hiding (Theorem 3.4):** For this choice of parameters, we have

  $$\begin{aligned}\ell_{\mathsf{BARG}}(\lambda, ms_{\mathsf{int}}, |C_{\mathcal{R}}|) &< (ms)^{\varepsilon} \cdot O((\lambda + \lambda^{c_1})^{c_3}) \\ &= m^{\varepsilon}(\ell\lambda)^{\varepsilon c_2} \cdot O(\lambda^{c_1 c_3}) \\ &= m^{\varepsilon(1+\delta_1 c_2)} \cdot O\big(\lambda^{(\delta_2+1)(\varepsilon c_2)+c_1 c_3}\big).\end{aligned}$$

  Next, we can write

  $$\varepsilon(1 + \delta_1 c_2) = \varepsilon\left(1 + \frac{\varepsilon c_2}{1 - \varepsilon c_2}\right) = \frac{\varepsilon}{1 - \varepsilon c_2} = \delta_1$$
  $$(\delta_2 + 1)\varepsilon c_2 + c_1 c_3 = \delta_2 \varepsilon c_2 + \varepsilon c_2 + c_1 c_3 = (\varepsilon c_2 + c_1 c_3)\left(\frac{\varepsilon c_2}{1 - \varepsilon c_2} + 1\right) = \frac{\varepsilon c_2 + c_1 c_3}{1 - \varepsilon c_2} = \delta_2.$$

  Correspondingly, we see that for this choice of parameters,

  $$\ell_{\mathsf{BARG}}(\lambda, ms_{\mathsf{int}}, |C_{\mathcal{R}}|) < m^{\varepsilon(1+\delta_1 c_2)} \cdot O\big(\lambda^{(\delta_2+1)(\varepsilon c_2)+c_1 c_3}\big) = m^{\delta_1} \cdot O(\lambda^{\delta_2}) = \ell(\lambda, m),$$

  which satisfies the requirement in Theorem 3.4.

We summarize this instantiation with the following corollary.

Corollary 3.5 (NIZKs from Sub-Exponentially Secure Somewhere-Sound BARGs)**.** *Assuming the existence of one-way functions and either (1) an adaptively-sound BARG for* NP *(Definition 2.8); or (2) a sub-exponentially-secure somewhere-sound BARG for* NP *(Definition 2.9), there exists a computational NIZK argument for* NP.

3.1 Proof of Theorem 3.3 (Somewhat Computational Binding)

Let crs be a common reference string in the support of $\mathsf{Setup}(1^\lambda, 1^m)$ (parsed according to Eq. (3.1)). We define the set $\mathcal{V}^{\mathsf{crs}} \subseteq \{0,1\}^m$ as follows:

$$\mathcal{V}^{\mathsf{crs}} := \{(C(k, \mathbf{z}_1), \ldots, C(k, \mathbf{z}_m)) \mid k \in \{0,1\}^\kappa\} \tag{3.7}$$

We now show that each of the requirements of Definition 2.10 is satisfied:

Output sparsity. By assumption $\kappa(\lambda, m) \le m^\delta \cdot p(\lambda)$ for a constant $\delta < 1$ and a fixed polynomial $p(\cdot)$. This means

$$|\mathcal{V}^{\mathsf{crs}}| \le 2^{\kappa(\lambda,m)} \le 2^{m^\delta \cdot p(\lambda)},$$

which satisfies the output sparsity requirement.

Computational binding. We define a function $\mathsf{BC.Extract}(\mathsf{crs}, c) \to \{0, 1, \bot\}$. On input a common reference string crs for Π_{BC} and a purported commitment c, the function is defined as follows:

- If there exists σ_0, σ_1 such that $\mathsf{BC.Verify}(\mathsf{crs}, c, b, \sigma_b) = 1$ for all $b \in \{0,1\}$, then output $\bot$.
- Otherwise, if there exists (b, σ) for some $b \in \{0,1\}$ where $\mathsf{BC.Verify}(\mathsf{crs}, c, b, \sigma) = 1$, then output b.
- Otherwise, output 0.

Note that BC.Extract may *not* be efficiently computable. Let $\mathcal{A}$ be an efficient adversary for the computational binding security game for Construction 3.1. We now define a sequence of hybrid experiments between the challenger and $\mathcal{A}$.

- Hyb_0: This is the real computational binding game. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically,

$$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{ \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \big\}_{j \in S_{\mathsf{key}}}, \big\{ \mathsf{crs}_{\mathsf{BC}}^{(i,j)} \big\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \big\{ \big(c^{(i,j)}, \sigma^{(i,j)} \big) \big\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}} \Big),$$

    where the individual components are sampled according to the specification of Setup in Construction 3.1. The challenger gives crs to algorithm $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \big(\pi_{\mathsf{BARG}}, \{ c^{(\mathsf{key},j)} \}_{j \in S_{\mathsf{key}}}, \{ c^{(i,j)} \}_{i \in I, j \in S_{\mathsf{int}}}, \{ \sigma^{(i,s)} \}_{i \in I} \big).$$

  - The output of the experiment is $b = 1$ if $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{ \mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}} \}$. Otherwise, the output is $b = 0$.
- Hyb_1: This is Hyb_0, except the challenger additionally checks that each of the bit commitments in the proof π has at most one possible opening. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically,

$$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{ \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \big\}_{j \in S_{\mathsf{key}}}, \big\{ \mathsf{crs}_{\mathsf{BC}}^{(i,j)} \big\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \big\{ \big(c^{(i,j)}, \sigma^{(i,j)} \big) \big\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}} \Big),$$

    where the individual components are sampled according to the specification of Setup in Construction 3.1. The challenger gives crs to algorithm $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \big(\pi_{\mathsf{BARG}}, \{ c^{(\mathsf{key},j)} \}_{j \in S_{\mathsf{key}}}, \{ c^{(i,j)} \}_{i \in I, j \in S_{\mathsf{int}}}, \{ \sigma^{(i,s)} \}_{i \in I} \big).$$

  - For each $j \in S_{\mathsf{key}}$, the challenger computes $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\big(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\big)$. Then, for each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, the challenger computes $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}\big)$.
  - The output of the experiment is $b = 1$ if the following conditions hold:
    - $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{ \mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}} \}$.
    - For all $j \in S_{\mathsf{key}}$, $y^{(\mathsf{key},j)} \neq \bot$.
    - For all $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, $y^{(i,j)} \neq \bot$.

    Otherwise, the output is $b = 0$.
- Hyb_2: This is Hyb_1, except the challenger additionally checks that the extracted bits $y^{(\mathsf{key},j)}$ and $y^{(i,j)}$ correspond to consistent evaluations of the circuit C. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically,

$$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{ \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \big\}_{j \in S_{\mathsf{key}}}, \big\{ \mathsf{crs}_{\mathsf{BC}}^{(i,j)} \big\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \big\{ \big(c^{(i,j)}, \sigma^{(i,j)} \big) \big\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}} \Big),$$

    where the individual components are sampled according to the specification of Setup in Construction 3.1. The challenger gives crs to algorithm $\mathcal{A}$.

– Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

– For each $j \in S_{\mathsf{key}}$, the challenger computes $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right)$. Then, for each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, the challenger computes $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}\right)$.

– The output of the experiment is $b = 1$ if all of the following hold:

  * $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{\mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}}\}$.
  * For all $j \in S_{\mathsf{key}}$, $y^{(\mathsf{key},j)} \neq \bot$.
  * For all $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, $y^{(i,j)} \neq \bot$.
  * For each $i \in I$ and $j \in [s_{\mathsf{int}}]$, let j_{L} and j_{R} be the wire indices that determine the value of the j^{th} non-input wire $j_{\mathsf{OUT}} = (\kappa + n) + j$ in the circuit C. The challenger checks that $y^{(i,j_{\mathsf{OUT}})} = \mathsf{NAND}(y^{(i,j_{\mathsf{L}})}, y^{(i,j_{\mathsf{R}})})$. Here, we use the convention that for all $j \in S_{\mathsf{key}}$, $y^{(i,j)} := y^{(\mathsf{key},j)}$.

  Otherwise, the output is $b = 0$.

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

Claim 3.6. *If Π_{BC} is statistically binding in binding mode, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. Since the only differences between Hyb_0 and Hyb_1 are the additional conditions that the challenger checks, it must be the case that in an execution of Hyb_0 with $\mathcal{A}$, one of the two events occurs with probability at least ε:

- Algorithm $\mathcal{A}$ outputs a commitment $c^{(\mathsf{key},j)}$ such that $\mathsf{BC.Extract}\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right) = \bot$.
- Algorithm $\mathcal{A}$ outputs a commitment $c^{(i,j)}$ such that $\mathsf{BC.Extract}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}\right) = \bot$.

Equivalently, with probability at least ε, the output of $\mathcal{A}$ contains a commitment $c^{(i,j)}$ where the extracted value $\mathsf{BC.Extract}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}) = \bot$ for some $i \in [m] \cup \{\mathsf{key}\}$ and $j \in [s]$. We show that this implies an (inefficient) algorithm $\mathcal{B}$ that breaks statistical binding of Π_{BC}. We highlight key steps in green.

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a common reference string $\mathsf{crs}_{\mathsf{BC}}^*$.
2. Algorithm $\mathcal{B}$ samples $i^* \xleftarrow{\mathrm{R}} [m] \cup \{\mathsf{key}\}$ and $j^* \xleftarrow{\mathrm{R}} [s]$.
3. Algorithm $\mathcal{B}$ now samples a common reference string for the hidden-bits generator. It starts by sampling the following collection of common reference strings for the bit commitment scheme:

   - **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

   Algorithm $\mathcal{B}$ redefines $\mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*)} := \mathsf{crs}_{\mathsf{BC}}^*$.

4. Algorithm $\mathcal{B}$ implements the remainder of the Setup algorithm exactly as described in Hyb_0 and Hyb_1. Namely, it samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$ and commits to $\mathbf{z}_1, \ldots, \mathbf{z}_m$ as follows:

   - For each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, compute $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$.

Let $s_{\mathsf{int}} = |S_{\mathsf{int}}| = s - (\kappa + n)$ be the number of non-input wires in C. Algorithm $\mathcal{B}$ samples a CRS for the BARG $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{m s_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|})$ and defines

$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{ \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \right\}_{j \in S_{\mathsf{key}}}, \left\{ \mathsf{crs}_{\mathsf{BC}}^{(i,j)} \right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{ \left(c^{(i,j)}, \sigma^{(i,j)} \right) \right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}} \right).$$

It gives crs to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ then outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \left(\pi_{\mathsf{BARG}}, \{ c^{(\mathsf{key},j)} \}_{j \in S_{\mathsf{key}}}, \{ c^{(i,j)} \}_{i \in I, j \in S_{\mathsf{int}}}, \{ \sigma^{(i,s)} \}_{i \in I} \right).$$

6. If $i^* = \mathsf{key}$ and $j^* \in S_{\mathsf{key}}$ or $i^* \in I$ and $j^* \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, then algorithm $\mathcal{B}$ checks (by exhaustive search) whether there exists σ_b such that

$$\mathsf{BC.Verify}(\mathsf{crs}_{\mathsf{BC}}^*, c^{(i^*,j^*)}, b, \sigma_b) = 1,$$

for all $b \in \{0,1\}$. If so, algorithm $\mathcal{B}$ outputs $c^{(i^*,j^*)}, \sigma_0, \sigma_1$. In all other cases, algorithm $\mathcal{B}$ outputs $\perp$.

By construction, the challenger samples $\mathsf{crs}_{\mathsf{BC}}^* \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$. Thus, algorithm $\mathcal{B}$ constructs the common reference string crs exactly as required by Setup. By assumption, this means with probability ε, this means the output of $\mathcal{A}$ contains a commitment $c^{(i,j)}$ where $\mathsf{BC.Extract}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}) = \perp$ for some $i \in [m] \cup \{\mathsf{key}\}$ and $j \in [s]$. Now, algorithm $\mathcal{B}$ samples the indices i^* and j^* uniformly at random, and moreover, these indices are information-theoretically hidden from the view of $\mathcal{A}$. This means that with probability $1/((m+1)s)$, it will be the case that $(i^*, j^*) = (i, j)$. In this case, $\mathsf{BC.Extract}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}) = \perp$, which means there exists σ_b such that $\mathsf{BC.Verify}(\mathsf{crs}_{\mathsf{BC}}^*, c^{(i^*,j^*)}, b, \sigma_b) = 1$ for $b \in \{0,1\}$. Correspondingly, algorithm $\mathcal{B}$ successfully wins the statistical binding game. The advantage of $\mathcal{B}$ is thus $\varepsilon/((m+1)s)$, which is non-negligible as m and s are both polynomially-bounded. □

Claim 3.7. *If Π_{BARG} is adaptively sound, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. Since the only difference between Hyb_1 and Hyb_2 is the additional condition that the challenger checks, it must be the case that in an execution of Hyb_1 with $\mathcal{A}$, the following properties hold with probability at least ε:

- $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{ \mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}} \}$.
- For all $j \in S_{\mathsf{key}}$, $y^{(\mathsf{key},j)} \neq \perp$.
- For all $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, $y^{(i,j)} \neq \perp$.
- There exists indices $i^* \in I$ and $j^* \in [s_{\mathsf{int}}]$ such that $y^{(i^*, j^*_{\mathrm{OUT}})} \neq \mathrm{NAND}(y^{(i^*, j^*_{\mathrm{L}})}, y^{(i^*, j^*_{\mathrm{R}})})$, where j^*_{L} and j^*_{R} are the wire indices that determine the value of the $(j^*)^{\text{th}}$ non-input wire $j^*_{\mathrm{OUT}} = (\kappa + n) + j^*$.

Otherwise, the output in Hyb_1 and Hyb_2 is identical. We show that this means the following two properties also hold:

- $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, \pi_{\mathsf{BARG}}) = 1$, where $x^{(i,j)}$ are the statements derived from crs and π according to Eq. (3.2). This follows by construction of the Verify algorithm. Namely, if $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, then $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, \pi_{\mathsf{BARG}}) = 1$.
- The statement $x^{(i^*,j^*)}$ satisfies $x^{(i^*,j^*)} \notin \mathcal{L}$. To see this, let

$$x^{(i^*,j^*)} = \left(\mathsf{crs}_{\mathsf{BC}}^{(i^*, j^*_{\mathrm{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i^*, j^*_{\mathrm{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i^*, j^*_{\mathrm{OUT}})}, c^{(i^*, j^*_{\mathrm{L}})}, c^{(i^*, j^*_{\mathrm{R}})}, c^{(i^*, j^*_{\mathrm{OUT}})} \right)$$

be the statement as defined in Eq. (3.2). Suppose there exists a witness

$$w = (w'_{\mathrm{L}}, w'_{\mathrm{R}}, w'_{\mathrm{OUT}}, \sigma_{\mathrm{L}}, \sigma_{\mathrm{R}}, \sigma_{\mathrm{OUT}})$$

such that $\mathcal{R}(x^{(i^*,j^*)}, w) = 1$. Then, by construction of w, the following properties hold:

- For each $\text{pos} \in \{\textsc{l}, \textsc{r}, \textsc{out}\}$, $\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i^*, j^*_{\mathsf{pos}})}, c^{(i^*, j^*_{\mathsf{pos}})}, w'_{\mathsf{pos}}, \sigma_{\mathsf{pos}}\big) = 1$
- $w'_{\textsc{out}} = \textsc{nand}(w'_{\textsc{l}}, w'_{\textsc{r}})$.

By assumption, $y^{(i^*, j^*_{\mathsf{pos}})} \neq \perp$, so by construction of $\mathsf{BC.Extract}$, there is at most one possible value to which $c^{(i^*, j^*_{\mathsf{pos}})}$ can open. In particular, the first property now implies that $w'_{\mathsf{pos}} = y^{(i^*, j^*_{\mathsf{pos}})} \in \{0, 1\}$ for all $\text{pos} \in \{\textsc{l}, \textsc{r}, \textsc{out}\}$. Combining the above relations, we conclude that

$$y^{(i^*, j^*_{\textsc{out}})} = w'_{\textsc{out}} = \textsc{nand}(w'_{\textsc{l}}, w'_{\textsc{r}}) = \textsc{nand}\big(y^{(i^*, j^*_{\textsc{l}})}, y^{(i^*, j^*_{\textsc{r}})}\big),$$

which contradicts the premise that $y^{(i^*, j^*_{\textsc{out}})} \neq \textsc{nand}(y^{(i^*, j^*_{\textsc{l}})}, y^{(i^*, j^*_{\textsc{r}})})$.

We now use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks adaptive soundness of Π_{BARG} as follows, highlighting key steps in green:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on the same security parameter 1^λ. Algorithm $\mathcal{A}$ outputs an output length 1^m.

2. Algorithm $\mathcal{B}$ computes s_{int} and $|C_{\mathcal{R}}|$ as in Construction 3.1. It outputs $1^{m s_{\mathsf{int}}}$ as the bound on the number of instances and $1^{|C_{\mathcal{R}}|}$ as the size of the circuit, and receives $\mathsf{crs}_{\mathsf{BARG}}$ from the challenger.

3. Algorithm $\mathcal{B}$ implements the remainder of the Setup algorithm exactly as described in Hyb_1 and Hyb_2, except it uses the common reference string $\mathsf{crs}_{\mathsf{BARG}}$ it received from the challenger instead. Specifically, it sets

$$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key}, j)}\big\}_{j \in S_{\mathsf{key}}}, \big\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\big\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \big\{\big(c^{(i,j)}, \sigma^{(i,j)}\big)\big\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\Big),$$

where the individual components $\mathbf{z}_1, \ldots, \mathbf{z}_m$, $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key}, j)}$, $\mathsf{crs}_{\mathsf{BC}}^{(i,j)}$, $c^{(i,j)}$, and $\sigma^{(i,j)}$ are sampled according to the specification of Hyb_1 (and $\mathsf{crs}_{\mathsf{BARG}}$ is the CRS from the challenger). It gives crs to algorithm $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \big(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key}, j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\big).$$

5. Algorithm $\mathcal{B}$ forms the statements $x^{(i,j)}$ for $i \in I$ and $j \in [s_{\mathsf{int}}]$ according to Eq. (3.2) and outputs the circuit $C_{\mathcal{R}}$, the statements $(x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}$, and the proof π_{BARG}.

We now compute the advantage of $\mathcal{B}$. In the adaptive soundness game, the challenger constructs $\mathsf{crs}_{\mathsf{BARG}}$ as $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{m s_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|})$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1 and Hyb_2 for algorithm $\mathcal{A}$. By the analysis above, with probability at least ε, algorithm $\mathcal{A}$ outputs $(I, \mathbf{r}_I, \pi)$ satisfying the following:

- $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, \pi_{\mathsf{BARG}}) = 1$.
- There exists some $i^* \in I$ and $j^* \in [s_{\mathsf{int}}]$ such that $x^{(i^*, j^*)} \notin \mathcal{L}$.

Correspondingly, algorithm $\mathcal{B}$ wins the adaptive security game with the same advantage ε which completes the proof. □

Claim 3.8. *It holds that* $\Pr\big[\mathsf{Hyb}_2(\mathcal{A}) = 1\big] = 0$.

Proof. Consider an execution of Hyb_2 with adversary $\mathcal{A}$. Suppose $\mathsf{Hyb}_2(\mathcal{A})$ outputs 1. This means the following properties hold:

- $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{\mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}}\}$.
- For all $j \in S_{\mathsf{key}}$, $y^{(\mathsf{key}, j)} \neq \perp$. Recall that $S_{\mathsf{key}} = [\kappa]$.
- For all $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, $y^{(i,j)} \neq \perp$.

- For each $i \in I$ and $j \in [s_{\mathsf{int}}]$, let j_{L} and j_{R} be the wire indices that determine the value of the j^{th} non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$ in the circuit C. Then, $y^{(i,j_{\mathrm{OUT}})} = \textsc{nand}(y^{(i,j_{\mathrm{L}})}, y^{(i,j_{\mathrm{R}})})$.

Let $k \in \{0,1\}^\kappa$ be the bitstring where $k_j = y^{(\mathsf{key},j)}$ for all $j \in [\kappa]$. We now argue that for all $i \in I$, it holds that $r_i = C(k, \mathbf{z}_i) = \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$:

- **Commitments to input wires.** First, by construction, for all $i \in I$ and $j \in S_{\mathsf{eval}}$, $c^{(i,j)}$ is a commitment to $z_{i,j-\kappa}$. This follows by construction of Setup since the challenger samples $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa})$. By correctness of Π_{BC}, then $\mathsf{BC.Verify}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, z_{i,j-\kappa}, \sigma^{(i,j)}) = 1$. Since $y^{(i,j)} \neq \bot$ and by construction of $\mathsf{BC.Extract}$, it must be the case that $y^{(i,j)} = z_{i,j-\kappa}$.

- **Commitments to output wires.** Since $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, this means that for all indices $i \in I$, it holds that $\mathsf{BC.Verify}(\mathsf{crs}_{\mathsf{BC}}^{(i,s)}, c^{(i,s)}, r_i, \sigma^{(i,s)}) = 1$. Again since $y^{(i,s)} \neq \bot$ and by construction of $\mathsf{BC.Extract}$, this means $y^{(i,s)} = r_i$.

- **Commitments to non-input wires.** By assumption, for each $i \in I$ and $j \in [s_{\mathsf{int}}]$, if j_{L} and j_{R} are the wire indices that determine the value of wire $j_{\mathrm{OUT}} = (\kappa + n) + j$, then $y^{(i,j_{\mathrm{OUT}})} = \textsc{nand}(y^{(i,j_{\mathrm{L}})}, y^{(i,j_{\mathrm{R}})})$. That is, $y^{(i,j)}$ is computed for all internal wires $j \in S_{\mathsf{int}}$ according to circuit C. Since this property holds for all internal wires $j \in S_{\mathsf{int}}$, it follows that $(y^{(\mathsf{key},1)}, \ldots, y^{(\mathsf{key},\kappa)}, y^{(i,\kappa+1)}, \ldots, y^{(i,s)})$ corresponds to the wire values for $C(k, \mathbf{z}_i)$. In particular, this means that $y^{(i,s)} = C(k, \mathbf{z}_i)$.

We conclude that for all $i \in I$, it holds that $r_i = C(k, \mathbf{z}_i) = \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. By definition of $\mathcal{V}_I^{\mathsf{crs}}$, this means that $\mathbf{r}_I \in \mathcal{V}_I^{\mathsf{crs}}$, which contradicts the premise that $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$. Correspondingly, we conclude that $\mathsf{Hyb}_2(\mathcal{A})$ could not have output 1. ☐

Combining Claims 3.6 to 3.8, we conclude that $\Pr\left[\mathsf{Hyb}_0(\mathcal{A})\right] = \mathsf{negl}(\lambda)$, which proves Theorem 3.3. ☐

3.2 Proof of Theorem 3.4 (Computational Hiding)

Let $\mathcal{A}$ be an efficient adversary for the computational hiding security game for Construction 3.1. We define a sequence of hybrid experiments between the challenger and $\mathcal{A}$.

- Hyb_0: This is the computational hiding game with $\beta = 0$. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically, it starts by sampling the common reference strings for the bit commitment schemes:
    * **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

    Next, the challenger samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$. For each each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\big)$. Finally, it samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{m s_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|})$ and defines

    $$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\big\}_{j \in S_{\mathsf{key}}}, \big\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\big\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \big\{\big(c^{(i,j)}, \sigma^{(i,j)}\big)\big\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}} \Big).$$

  - Next, the challenger computes $(\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs})$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{st}, I)$. First, the challenger samples a weak PRF key $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$ and computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. It sets $\mathbf{r} = r_1 \| \cdots \| r_m$. To construct the proof π, the challenger first sets $w_1^{(i)}, \ldots, w_s^{(i)}$ to be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$. Then, it does the following:

- **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$, compute the commitment and opening $\left(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(\mathsf{key},j)}, k_j\right)$.
- **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\mathsf{int}}$, compute $\left(c^{(i,j)}, \sigma^{(i,j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(i,j)}, w_j^{(i)}\right)$.
- **Construct a BARG proof of validity:** For each $j \in [s_{\mathsf{int}}]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the wire indices that determine the value of the j^{th} non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$. Then, for each $i \in [m]$, define the statement $x^{(i,j)}$ and witness $w^{(i,j)}$ as follows:

$$x^{(i,j)} = \left(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{OUT}})}, c^{(i,j_{\mathrm{L}})}, c^{(i,j_{\mathrm{R}})}, c^{(i,j_{\mathrm{OUT}})}\right)$$
$$w^{(i,j)} = \left(w_{j_{\mathrm{L}}}^{(i)}, w_{j_{\mathrm{R}}}^{(i)}, w_{j_{\mathrm{OUT}}}^{(i)}, \sigma^{(i,j_{\mathrm{L}})}, \sigma^{(i,j_{\mathrm{R}})}, \sigma^{(i,j_{\mathrm{OUT}})}\right).$$

As in Construction 3.1, we adopt the convention that $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} := \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}$, $c^{(i,j)} := c^{(\mathsf{key},j)}$, and $\sigma^{(i,j)} := \sigma^{(\mathsf{key},j)}$. The challenger computes

$$\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}\left(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, (w^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}\right).$$

Finally the challenger defines the proof π to be

$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

- The challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. 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 switches the bit commitments for the key and the non-input wires to be equivocating.
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.
  - The challenger starts by sampling the common reference strings for the bit commitment schemes:
    - **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample

$$\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \tilde{c}^{(\mathsf{key},j)}, \tilde{\sigma}_0^{(\mathsf{key},j)}, \tilde{\sigma}_1^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

    - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample

$$\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \tilde{c}^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

    The remaining components of the crs are computed exactly as described in Hyb_0 (same for all hybrids).
  - The challenger samples $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$ and computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. It sets $\mathbf{r} = r_1 \| \cdots \| r_m$. To construct the proof π, the challenger first sets $w_1^{(i)}, \ldots, w_s^{(i)}$ to be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$. Then, it does the following:
    - **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$, let $c^{(\mathsf{key},j)} := \tilde{c}^{(\mathsf{key},j)}$ and $\sigma^{(\mathsf{key},j)} := \tilde{\sigma}_b^{(\mathsf{key},j)}$ where $b = k_j$.
    - **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\mathsf{int}}$, let $c^{(i,j)} := \tilde{c}^{(i,j)}$ and $\sigma^{(i,j)} := \tilde{\sigma}_b^{(i,j)}$ where $b = w_j^{(i)}$.
    - **Construct a BARG proof of validity:** The BARG proof π_{BARG} is computed exactly as described in Hyb_0 (same for all hybrids).

    The challenger defines the proof π as in Hyb_0 (same for all hybrids).

- The challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- Hyb_2: Same as Hyb_1, except for all $i \notin I$, the challenger samples $r_i \xleftarrow{\mathrm{R}} \{0,1\}$. Specifically, the experiment proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.
  - The challenger starts by sampling the common reference strings for the bit commitment schemes:
    * **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample
    $$\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \tilde{c}^{(\mathsf{key},j)}, \sigma_0^{(\mathsf{key},j)}, \sigma_1^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$
    * **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample
    $$\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \tilde{c}^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

    The remaining components of the crs are computed exactly as described in Hyb_0 (same for all hybrids).
  - The challenger samples $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$. For each $i \in I$, it computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. For each $i \notin I$, it samples $r_i \xleftarrow{\mathrm{R}} \{0,1\}$. It sets $\mathbf{r} = r_1 \| \cdots \| r_m$. To construct the proof π, the challenger first sets $w_1^{(i)}, \ldots, w_s^{(i)}$ to be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$. Then, it does the following:
    * **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$, let $c^{(\mathsf{key},j)} := \tilde{c}^{(\mathsf{key},j)}$ and $\sigma^{(\mathsf{key},j)} := \tilde{\sigma}_b^{(\mathsf{key},j)}$ where $b = k_j$.
    * **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\mathsf{int}}$, let $c^{(i,j)} := \tilde{c}^{(i,j)}$ and $\sigma^{(i,j)} := \tilde{\sigma}_b^{(i,j)}$ where $b = w_j^{(i)}$.
    * **Construct a BARG proof of validity:** The BARG proof π_{BARG} is computed exactly as described in Hyb_0 (same for all hybrids).

    The challenger defines the proof π as in Hyb_0 (same for all hybrids).
  - The challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- Hyb_3: Same as Hyb_2 except the challenger switches the commitments back to binding mode. This is the computational hiding game with $\beta = 1$. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.
  - The challenger starts by sampling the common reference strings for the bit commitment schemes:
    * **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

    The remaining components of the crs are computed exactly as described in Hyb_0 (same for all hybrids).
  - The challenger samples $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$. For each $i \in I$, it computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. For each $i \notin I$, it samples $r_i \xleftarrow{\mathrm{R}} \{0,1\}$. It sets $\mathbf{r} = r_1 \| \cdots \| r_m$. To construct the proof π, the challenger first sets $w_1^{(i)}, \ldots, w_s^{(i)}$ to be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$. Then, it does the following:
    * **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$, compute the commitment and opening $\left(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(\mathsf{key},j)}, k_j\right)$.

    * **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\mathsf{int}}$, compute $\left(c^{(i,j)}, \sigma^{(i,j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(i,j)}, w_j^{(i)}\right)$.
    * **Construct a BARG proof of validity:** The BARG proof π_{BARG} is computed exactly as described in Hyb_0 (same for all hybrids).

  The challenger defines the proof π as in Hyb_0 (same for all hybrids).
- The challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

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

Claim 3.9. *Suppose Π_{BC} satisfies mode indistinguishability. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Let $T = |S_{\mathsf{key}}| + m \cdot |S_{\mathsf{int}}|$. For each $t \in [T+1]$, we define an intermediate hybrid $\mathsf{Hyb}_{0,t}$ as follows:

- On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.
- The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically, it starts by sampling the common reference strings for the bit commitment schemes:
  - **CRS for the key:** For $j \in S_{\mathsf{key}}$:
    * If $j \geq t$, sample $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * If $j < t$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \tilde{c}^{(\mathsf{key},j)}, \tilde{\sigma}_0^{(\mathsf{key},j)}, \tilde{\sigma}_1^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda)$.
  - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
  - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$:
    * If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) \geq t$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
    * If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) < t$, sample
$$\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \tilde{c}^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

  Next, the challenger samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$. For each each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$. Finally, it samples the BARG common reference string $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|})$ and defines
$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\mathsf{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right).$$

- Next, the challenger computes $(\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs})$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{st}, I)$. First, the challenger samples a weak PRF key $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$ and computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. It sets $\mathbf{r} = r_1 \| \cdots \| r_m$. To construct the proof π, the challenger first sets $w_1^{(i)}, \ldots, w_s^{(i)}$ to be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$. Then, it does the following:
  - **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$:
    * If $j \geq t$, compute $\left(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(\mathsf{key},j)}, k_j\right)$.
    * If $j < t$, let $c^{(\mathsf{key},j)} := \tilde{c}^{(\mathsf{key},j)}$ and $\sigma^{(\mathsf{key},j)} := \tilde{\sigma}_b^{(\mathsf{key},j)}$ where $b = k_j$.
  - **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\mathsf{int}}$:
    * If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) \geq t$, compute $\left(c^{(i,j)}, \sigma^{(i,j)}\right) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}^{(i,j)}, w_j^{(i)}\right)$.

    * If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) < t$ and $\sigma^{(i,j)} := \tilde{\sigma}_b^{(i,j)}$ where $b = w_j^{(i)}$.
  - **Construct a BARG proof of validity:** For each $j \in [s_{\mathsf{int}}]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the wire indices that determine the value of the j^{th} non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$. Then, for each $i \in [m]$, define the statement $x^{(i,j)}$ and witness $w^{(i,j)}$ as follows:

$$x^{(i,j)} = \left(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{OUT}})}, c^{(i,j_{\mathrm{L}})}, c^{(i,j_{\mathrm{R}})}, c^{(i,j_{\mathrm{OUT}})}\right)$$
$$w^{(i,j)} = \left(w_{j_{\mathrm{L}}}^{(i)}, w_{j_{\mathrm{R}}}^{(i)}, w_{j_{\mathrm{OUT}}}^{(i)}, \sigma^{(i,j_{\mathrm{L}})}, \sigma^{(i,j_{\mathrm{R}})}, \sigma^{(i,j_{\mathrm{OUT}})}\right).$$

    As in Construction 3.1, we adopt the convention that $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} := \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}$, $c^{(i,j)} := c^{(\mathsf{key},j)}$, and $\sigma^{(i,j)} := \sigma^{(\mathsf{key},j)}$. The BARG proof π_{BARG} is computed exactly as described in Hyb_0 (same for all hybrids).

  The challenger defines the proof π as in Hyb_0 (same for all hybrids).

- The challenger gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We now show that for all $t \in [T]$,

$$|\Pr[\mathsf{Hyb}_{0,t+1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,t}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Suppose instead that $|\Pr[\mathsf{Hyb}_{0,t+1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,t}(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks mode indistinguishability of Π_{BC}. We highlight the key steps in green.

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a common reference string $\mathsf{crs}_{\mathsf{BC}}^*$.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input the security parameter 1^λ. Algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.

3. Algorithm $\mathcal{B}$ samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically, it starts by sampling the common reference strings for the bit commitment schemes:

   - **CRS for the key:** For $j \in S_{\mathsf{key}}$,
     - If $j > t$, sample $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
     - If $j = t$, set $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{crs}_{\mathsf{BC}}^*$.
     - If $j < t$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \tilde{c}^{(\mathsf{key},j)}, \tilde{\sigma}_0^{(\mathsf{key},j)}, \tilde{\sigma}_1^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda)$.
   - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$:
     - If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) > t$, sample $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
     - If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) = t$, set $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{crs}_{\mathsf{BC}}^*$.
     - If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) < t$, sample

$$\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \tilde{c}^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

   Next, algorithm $\mathcal{B}$ samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$. For each each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa})$. Finally, it samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|})$ and defines

$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\mathsf{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right).$$

4. Next, algorithm $\mathcal{B}$ computes $(\mathbf{r}, \mathsf{st}) \leftarrow \mathsf{GenBits}(\mathsf{crs})$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{st}, I)$. First, algorithm $\mathcal{B}$ samples a weak PRF key $k \leftarrow \mathsf{LRwPRF.Setup}(1^\lambda, 1^\ell)$ and computes $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$. It sets $\mathbf{r} = r_1 \| \cdots \| r_m$. To construct the proof π, algorithm $\mathcal{B}$ first sets $w_1^{(i)}, \ldots, w_s^{(i)}$ to be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$. Then, it does the following:

   - **Commit to the bits of k:** For each $j \in S_{\mathsf{key}}$:
     - If $j > t$, compute $\big(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}\big) \leftarrow \mathsf{BC.Commit}\big(\mathsf{crs}^{(\mathsf{key},j)}, k_j\big)$.
     - If $j = t$, query the challenger on the message bit $k_j \in \{0,1\}$ to receive the commitment $c^{(\mathsf{key},j)}$ and the opening $\sigma^{(\mathsf{key},j)}$.
     - If $j < t$, let $c^{(\mathsf{key},j)} := \tilde{c}^{(\mathsf{key},j)}$ and $\sigma^{(\mathsf{key},j)} := \tilde{\sigma}_b^{(\mathsf{key},j)}$ where $b = k_j$.
   - **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\mathsf{int}}$:
     - If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) > t$, compute $\big(c^{(i,j)}, \sigma^{(i,j)}\big) \leftarrow \mathsf{BC.Commit}\big(\mathsf{crs}^{(i,j)}, w_j^{(i)}\big)$.
     - If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) = t$, query the challenger on the wire value $w_j^{(i)}$ to receive the commitment $c^{(i,j)}$ and the opening $\sigma^{(i,j)}$.
     - If $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) < t$, let $c^{(i,j)} := \tilde{c}^{(i,j)}$ and $\sigma^{(i,j)} := \tilde{\sigma}_b^{(i,j)}$ where $b = w_j^{(i)}$.
   - **Construct a BARG proof of validity:** For each $j \in [s_{\mathsf{int}}]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the wire indices that determine the value of the j^{th} non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$. Then, for each $i \in [m]$, define the statement $x^{(i,j)}$ and witness $w^{(i,j)}$ as follows:

$$x^{(i,j)} = \Big(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{OUT}})}, c^{(i,j_{\mathrm{L}})}, c^{(i,j_{\mathrm{R}})}, c^{(i,j_{\mathrm{OUT}})}\Big)$$
$$w^{(i,j)} = \Big(w_{j_{\mathrm{L}}}^{(i)}, w_{j_{\mathrm{R}}}^{(i)}, w_{j_{\mathrm{OUT}}}^{(i)}, \sigma^{(i,j_{\mathrm{L}})}, \sigma^{(i,j_{\mathrm{R}})}, \sigma^{(i,j_{\mathrm{OUT}})}\Big).$$

     As in Construction 3.1, we adopt the convention that $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} := \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}$, $c^{(i,j)} := c^{(\mathsf{key},j)}$, and $\sigma^{(i,j)} := \sigma^{(\mathsf{key},j)}$. Algorithm $\mathcal{B}$ computes

$$\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}\big(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, (w^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}\big).$$

   Finally algorithm $\mathcal{B}$ defines the proof π to be

$$\pi = \big(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\big).$$

5. Algorithm $\mathcal{B}$ gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

Let β be the bit in the mode indistinguishability game. We consider the two cases:

- Suppose $\beta = 0$. If $j = t$ for some $j \in S_{\mathsf{key}}$, then algorithm $\mathcal{B}$ sets $\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$ and $(c^{(\mathsf{key},j)}, \sigma^{(\mathsf{key},j)}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}_{\mathsf{BC}}, k_j)$. Alternatively, if $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) = t$, then algorithm $\mathcal{B}$ sets $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$ and $(c_{i,j}, \sigma_{i,j}) \leftarrow \mathsf{BC.Commit}(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, w_i^{(j)})$. In this case, algorithm $\mathcal{B}$ perfectly simulates hybrid $\mathsf{Hyb}_{0,t}$.

- Suppose $\beta = 1$. If $j = t$ for some $j \in S_{\mathsf{key}}$, then algorithm $\mathcal{B}$ sets $(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \tilde{c}^{(\mathsf{key},j)}, \tilde{\sigma}_0^{(\mathsf{key},j)}, \tilde{\sigma}_1^{(\mathsf{key},j)}) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda)$ and $c^{(\mathsf{key},j)} \leftarrow \tilde{c}^{(\mathsf{key},j)}$, $\sigma^{(\mathsf{key},j)} \leftarrow \tilde{\sigma}_b^{(\mathsf{key},j)}$ where $b = k_j$. Alternatively, if $|S_{\mathsf{key}}| + i \cdot |S_{\mathsf{int}}| + (j - \kappa - n) = t$, then algorithm $\mathcal{B}$ sets the bit-commitment and openings $(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda)$ and $\sigma^{(i,j)} \leftarrow \tilde{\sigma}_b^{(i,j)}$ for $b = w_j^{(i)}$. In this case, algorithm $\mathcal{B}$ perfectly simulates hybrid $\mathsf{Hyb}_{0,t+1}$.

We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability with the same advantage ε. Thus, for all $t \in [T]$, we have that the output distribution of $\mathsf{Hyb}_{0,t}(\mathcal{A})$ is computationally indistinguishable from the output distribution of $\mathsf{Hyb}_{0,t+1}$. By construction $\mathsf{Hyb}_0(\mathcal{A}) \equiv \mathsf{Hyb}_{0,1}(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A}) \equiv \mathsf{Hyb}_{0,T+1}(\mathcal{A})$. Since $T = |S_{\mathsf{key}}| + m \cdot |S_{\mathsf{int}}|$ is polynomially-bounded, the claim now follows by a hybrid argument. □

Claim 3.10. *Suppose* Π_{LRwPRF} *is a leakage-resilient weak PRF. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. We start by defining the Boolean circuit $\mathsf{leak}\colon \{0,1\}^\kappa \to \{0,1\}^{\ell_{\mathsf{BARG}}}$ that takes a PRF key $k \in \{0,1\}^\kappa$ as input and generates a BARG proof according to the specification of the Prove algorithm:

Hard-wired: PRF inputs $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \{0,1\}^n$, a common reference string $\mathsf{crs}_{\mathsf{BARG}}$ for the BARG, and the following sets of commitment-opening tuples:

- **Key:** for each $j \in S_{\mathsf{key}}$, $\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}, \tilde{\sigma}_0^{(\mathsf{key},j)}, \tilde{\sigma}_1^{(\mathsf{key},j)}\right)$;
- **Evaluation point:** for each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, \sigma^{(i,j)}\right)$;
- **Non-input wires:** for each $i \in [m]$ and $j \in S_{\mathsf{int}}$, $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, c^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}\right)$.

On input the PRF key $k \in \{0,1\}^\kappa$:

- For each $i \in [m]$, let $w_1^{(i)}, \ldots, w_s^{(i)}$ be the wire values of $C(k, \mathbf{z}_i)$ for each $i \in [m]$.
- For each $j \in S_{\mathsf{key}}$, let $\sigma^{(\mathsf{key},j)} := \tilde{\sigma}_b^{(\mathsf{key},j)}$ where $b = k_j$.
- For each $i \in [m]$ and $j \in S_{\mathsf{int}}$, let $\sigma^{(i,j)} := \tilde{\sigma}_b^{(i,j)}$ where $b = w_j^{(i)}$.
- For each $j \in [s_{\mathsf{int}}]$, let $j_{\mathrm{L}}, j_{\mathrm{R}}$ be the wire indices that determine the value of the jth non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$. Then, for each $i \in [m]$, define the statement $x^{(i,j)}$ and witness $w^{(i,j)}$ as follows:

$$x^{(i,j)} = \left(\mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i,j_{\mathrm{OUT}})}, c^{(i,j_{\mathrm{L}})}, c^{(i,j_{\mathrm{R}})}, c^{(i,j_{\mathrm{OUT}})}\right)$$
$$w^{(i,j)} = \left(w_{j_{\mathrm{L}}}^{(i)}, w_{j_{\mathrm{R}}}^{(i)}, w_{j_{\mathrm{OUT}}}^{(i)}, \sigma^{(i,j_{\mathrm{L}})}, \sigma^{(i,j_{\mathrm{R}})}, \sigma^{(i,j_{\mathrm{OUT}})}\right).$$

  As in Construction 3.1, we adopt the convention that $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} := \mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}$, $c^{(i,j)} := c^{(\mathsf{key},j)}$, and $\sigma^{(i,j)} := \sigma^{(\mathsf{key},j)}$. Compute and output

$$\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}\left(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, (w^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}\right).$$

Figure 1: The leakage circuit $\mathsf{leak}\colon \{0,1\}^\kappa \to \{0,1\}^{\ell_{\mathsf{BARG}}}$.

Returning to the proof of Claim 3.10, suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the leakage-resilient weak PRF security game for Π_{LRwPRF}. As usual, we highlight key steps in green.

1. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input input the security parameter 1^λ. Algorithm $\mathcal{A}$ outputs the output length 1^m and a subset $I \subseteq [m]$.
2. Algorithm $\mathcal{B}$ starts by sampling the common reference strings for the bit commitment schemes:
   - **CRS for the key:** For $j \in S_{\mathsf{key}}$, it samples

$$\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \tilde{c}^{(\mathsf{key},j)}, \sigma_0^{(\mathsf{key},j)}, \sigma_1^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

   - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it samples the bit-commitment CRS $\mathsf{crs}_{\mathsf{BC}}^{(i,j)} \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

- **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\text{int}}$, it samples
$$\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \tilde{c}^{(i,j)}, \tilde{\sigma}_0^{(i,j)}, \tilde{\sigma}_1^{(i,j)}\right) \leftarrow \mathsf{BC.SetupEquivocate}(1^\lambda).$$

3. Algorithm $\mathcal{B}$ outputs the leakage parameter 1^ℓ and the number of pre-challenge evaluation queries $1^{|I|}$. The challenger replies with a set of $|I|$ input-output pairs $(\mathbf{z}_i, r_i)$. Algorithm $\mathcal{B}$ associates each one with an index $i \in I$. Namely, let $\{(\mathbf{z}_i, r_i)\}_{i \in I}$ be the challenger's response.
4. For each each $i \in [m]$ and $j \in S_{\text{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$. Finally, it samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\text{int}}}, 1^{|C_{\mathcal{R}}|})$ and defines
$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\text{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\text{eval}} \cup S_{\text{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\text{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right).$$
   Next, algorithm $\mathcal{B}$ sets the commitments to the key and the non-input wires as follows:
   - **Commit to the bits of k:** For each $j \in S_{\text{key}}$, let $c^{(\mathsf{key},j)} := \tilde{c}^{(\mathsf{key},j)}$.
   - **Commit to the non-input wires for $C(k, \mathbf{z}_i)$:** For each $i \in [m]$ and $j \in S_{\text{int}}$, let $c^{(i,j)} := \tilde{c}^{(i,j)}$.
5. Algorithm $\mathcal{B}$ makes a leakage query on the circuit leak defined in Fig. 1 (with $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$ as the inputs, $\mathsf{crs}_{\mathsf{BARG}}$ as the BARG common reference string, and the above-generated commitment/opening tuples for the key, evaluation point, and non-input wires). The challenger replies with a proof π_{BARG}.
6. Algorithm $\mathcal{B}$ outputs $1^{m-|I|}$ as the number of challenge queries. The challenger replies with a set of $(m - |I|)$ input-output pairs $\mathbf{z}_i, r_i)$. Algorithm $\mathcal{B}$ associates each one with an index $i \in \bar{I}$. Namely, let $\{(\mathbf{z}_i, r_i)\}_{i \in \bar{I}}$ be the challenger's response.
7. Algorithm $\mathcal{B}$ sets $\mathbf{r} = r_1 \| \cdots \| r_m$. Then, for each $i \in [m]$, it sets $\sigma^{(i,s)} := \tilde{\sigma}_b^{(i,s)}$ where $b = r_i$.
   Finally algorithm $\mathcal{B}$ defines the proof π to be
$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\text{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\text{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$
8. Algorithm $\mathcal{B}$ gives $(\mathsf{crs}, I, \mathbf{r}_I, \pi, \mathbf{r}_{\bar{I}})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

Since $\ell \geq \ell_{\mathsf{BARG}} \geq |\pi_{\mathsf{BARG}}|$, algorithm $\mathcal{B}$ is an admissible adversary. Next, in the leakage-resilient weak PRF security game, in the pre-challenge phase, the challenger samples $\mathbf{z}_i \xleftarrow{\text{R}} \{0,1\}^n$ and $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$ for all $i \in I$, which matches the distribution in Hyb_1 and Hyb_2. Next, the challenger computes $\pi_{\mathsf{BARG}} \leftarrow \mathsf{leak}(k)$, which precisely coincides with the algorithm used in Hyb_1 and Hyb_2 to construct π_{BARG}. We conclude that crs, $\mathbf{r}_I$, and π are distributed exactly as required in Hyb_1 and Hyb_2. We consider the challenge queries. Let $\beta \in \{0,1\}$ the bit in the leakage-resilient weak PRF security game. We consider the two possibilities:

- Suppose $\beta = 0$. Then, the challenger samples $\mathbf{z}_i \xleftarrow{\text{R}} \{0,1\}^n$ and $r_i \leftarrow \mathsf{LRwPRF.Eval}(k, \mathbf{z}_i)$ for each $i \in \bar{I}$. In this case, algorithm $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$.
- Suppose $\beta = 1$. Then, the challenger samples $\mathbf{z}_i \xleftarrow{\text{R}} \{0,1\}^n$ and $r_i \xleftarrow{\text{R}} \{0,1\}$ for each $i \in \bar{I}$. In this case, algorithm $\mathcal{B}$ perfectly simulates Hyb_2 for $\mathcal{A}$.

We conclude that algorithm $\mathcal{B}$ breaks leakage-resilient weak PRF security with the same advantage ε. □

Claim 3.11. *Suppose Π_{BC} satisfies mode indistinguishability. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. This follows by an analogous argument as the proof of Claim 3.9, except the reduction algorithm samples $r_i \xleftarrow{\text{R}} \{0,1\}$ for all $i \in \bar{I}$ (as in the specification of Hyb_2 and Hyb_3). □

Combining Claims 3.9 to 3.11, computational hiding follows by a hybrid argument. □

4 Hidden-Bits Generator from BARGs and Public-Key Encryption

In this section, we show how to construct a hidden-bits generator with subset-dependent proofs by combining a *polynomial-hard* somewhere-sound BARG with a public-key encryption scheme. Compared to Corollary 3.5, this construction only relies on polynomial hardness on the somewhere-sound BARG (as opposed to sub-exponential hardness), but in exchange, it requires an additional assumption of public-key encryption. As described in Section 1.1, this construction follows the same template as the previous construction (Section 3), but uses public-key encryption to construct a one-time dual-mode bit commitment with *efficient* extraction.

4.1 One-Time Dual-Mode Bit Commitment with Extraction

The main building block we use in this section is a one-time dual-mode bit commitment scheme that supports *efficient* extraction. Recall that in a standard one-time dual-mode bit commitment scheme (Definition 2.4), we only require the bit commitment scheme to be statistically binding in binding mode. Here, we upgrade the statistical binding to a strong extractability guarantee. This will allow us to base security of our hidden-bits generator somewhere soundness rather than adaptive soundness.

Definition 4.1 (One-Time Dual-Mode Bit Commitment with Extraction)**.** A one-time dual-mode bit commitment with extraction is a tuple of algorithms $\Pi_{\mathsf{BC}} = (\mathsf{SetupBind}, \mathsf{SetupEquivocate}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{SetupBind}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter λ, the setup algorithm for the binding mode outputs a common reference string crs and trapdoor td.
- $\mathsf{SetupEquivocate}(1^\lambda) \to (\mathsf{crs}, \tilde{c}, \tilde{\sigma}_0, \tilde{\sigma}_1)$: On input the security parameter λ, the setup algorithm for the equivocating mode outputs a common reference string crs along with a commitment $\tilde{c}$ and openings $\tilde{\sigma}_0, \tilde{\sigma}_1$.
- $\mathsf{Commit}(\mathsf{crs}, b) \to (c, \sigma)$: On input the common reference string crs and a bit $b \in \{0,1\}$, the commit algorithm outputs a commitment c and an opening σ.
- $\mathsf{Verify}(\mathsf{crs}, c, b, \sigma) \to \{0,1\}$: On input the common reference string crs, a commitment c, a bit $b \in \{0,1\}$, and an opening σ, the verification algorithm outputs a bit $b' \in \{0,1\}$.
- $\mathsf{Extract}(\mathsf{td}, c) \to \{0,1\}$: On input the trapdoor td and a commitment c, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require Π_{BC} satisfy the following properties:

- **Correctness:** Same as in Definition 2.4.
- **Mode indistinguishability:** Same as in Definition 2.4.
- **Extractable in binding mode:** For all adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, c, b, \sigma) = 1 \land b \neq b' : \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{SetupBind}(1^\lambda) \\ (c, \sigma, b) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ b' \leftarrow \mathsf{Extract}(\mathsf{td}, c) \end{array}\right] = \mathsf{negl}(\lambda).$$

Constructing a one-time dual-mode bit commitment with extraction scheme. We can construct a one-time dual-mode bit commitment with extraction scheme by composing a vanilla one-time dual-mode bit commitment scheme (Definition 2.4) with a public-key encryption scheme. A similar approach was used implicitly in previous works [KW19, LQR+19].

Construction 4.2 (One-Time Dual-Mode Bit Commitment with Extraction)**.** Our construction relies on the following primitives:

- Let $\Pi'_{\mathsf{BC}} = (\mathsf{BC.SetupBind}', \mathsf{BC.SetupEquivocate}', \mathsf{BC.Commit}', \mathsf{BC.Verify}')$ be a one-time dual mode bit commitment scheme. Let $\ell_{\mathsf{BC}} = \ell_{\mathsf{BC}}(\lambda)$ be a bound on the length of the openings output by $\mathsf{BC.Commit}'$.
- Let $\Pi_{\mathsf{PKE}} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ be a public-key encryption scheme with message space $\mathcal{M}_\lambda = \{0,1\}^{\ell_{\mathsf{BC}}(\lambda)+1} \cup \{\perp\}$. Let $\rho = \rho(\lambda)$ be the randomness complexity of $\mathsf{PKE.Encrypt}$ (i.e., the number of bits of randomness that $\mathsf{PKE.Encrypt}$ takes as input).

We construct a one-time dual mode bit commitment scheme scheme with extraction $\Pi_{\mathsf{BC}} = (\mathsf{SetupBind}, \mathsf{SetupEquivocate}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Extract})$ as follows:

- $\mathsf{SetupBind}(1^\lambda)$: On input the security parameter λ, the binding mode setup algorithm samples $\mathsf{crs}' \leftarrow \mathsf{BC.SetupBind}'(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. It then outputs the common reference string $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$ and the extraction trapdoor $\mathsf{td} = (\mathsf{crs}', \mathsf{sk})$.
- $\mathsf{SetupEquivocate}(1^\lambda)$: On input the security parameter λ, the equivocating mode setup algorithm samples $(\mathsf{crs}', \tilde{c}', \tilde{\sigma}'_0, \tilde{\sigma}'_1) \leftarrow \mathsf{BC.SetupEquivocate}'(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. Next, for $b \in \{0,1\}$, it samples $r_b \xleftarrow{\text{R}} \{0,1\}^\rho$ and sets $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \tilde{\sigma}'_b); r_b)$. It outputs the common reference string $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$, the commitment $\tilde{c} = (\tilde{c}', \mathsf{ct}_0, \mathsf{ct}_1)$, and the openings $\tilde{\sigma}_0 = (\tilde{\sigma}'_0, r_0)$, $\tilde{\sigma}_1 = (\tilde{\sigma}'_1, r_1)$.
- $\mathsf{Commit}(\mathsf{crs}, b)$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$ and a bit $b \in \{0,1\}$, the commit algorithm constructs a commitment $(c', \sigma') \leftarrow \mathsf{BC.Commit}'(\mathsf{crs}', b)$. Then, it samples $r_b \xleftarrow{\text{R}} \{0,1\}^\rho$ and computes $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$. It also computes $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, \perp)$. Finally, it outputs the commitment $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and the opening $\sigma = (\sigma', r_b)$.
- $\mathsf{Verify}(\mathsf{crs}, c, b, \sigma)$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$, a commitment $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$, a bit $b \in \{0,1\}$, and an opening $\sigma = (\sigma', r_b)$, the verification algorithm outputs 1 if $\mathsf{BC.Verify}'(\mathsf{crs}', c', b, \sigma') = 1$ and $\mathsf{ct}_b = \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$.
- $\mathsf{Extract}(\mathsf{td}, c)$: On input an extraction trapdoor $\mathsf{td} = (\mathsf{crs}', \mathsf{sk})$ and a commitment $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$, the extraction algorithm computes the message $m_0 \leftarrow \mathsf{PKE.Decrypt}(\mathsf{sk}, \mathsf{ct}_0)$. If $m_0 = (0, \sigma')$ and $\mathsf{BC.Verify}'(\mathsf{crs}', c', 0, \sigma') = 1$, then it outputs 0. Otherwise, it outputs 1.

Theorem 4.3 (Correctness)**.** *If Π'_{BC} and Π_{PKE} are correct, then Construction 4.2 is correct.*

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter and take any common reference string $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$ in the support of $\mathsf{SetupBind}(1^\lambda)$ or $\mathsf{SetupEquivocate}(1^\lambda)$. By construction of $\mathsf{SetupBind}$ and $\mathsf{SetupEquivocate}$, this means crs' is in the support of $\mathsf{BC.SetupBind}'(1^\lambda)$ or $\mathsf{BC.SetupEquivocate}'(1^\lambda)$. Let $b \in \{0,1\}$ be a bit and take any commitment-opening pair (c, σ) in the support of $\mathsf{Commit}(\mathsf{crs}, b)$. Then, we can write $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma = (\sigma', r_b)$, where (c', σ') is in the support of $\mathsf{BC.Commit}'(\mathsf{crs}', b)$ and $\mathsf{ct}_b = \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$. By correctness of Π'_{BC}, $\mathsf{BC.Verify}'(\mathsf{crs}', c', b, \sigma') = 1$. Then, $\mathsf{Verify}(\mathsf{crs}, c, b, \sigma) = 1$, completing the proof of correctness. □

Theorem 4.4 (Mode Indistinguishability)**.** *Suppose Π'_{BC} satisfies mode indistinguishability. Then Construction 4.2 satisfies mode indistinguishability.*

Proof. Let $\mathcal{A}$ be an efficient adversary in the mode indistinguishability game for Construction 4.2. We define a sequence of hybrid experiments between the challenger and $\mathcal{A}$.

- Hyb_0: This is the real mode indistinguishability game with bit $\beta = 0$. Specifically, the game proceeds as follows:
  1. The challenger starts by sampling $\mathsf{crs}' \leftarrow \mathsf{BC.SetupBind}'(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and sets $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$. The challenger gives the security parameter 1^λ and crs to $\mathcal{A}$.
  2. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. The challenger computes $(c', \sigma') \leftarrow \mathsf{BC.Commit}'(\mathsf{crs}', b)$ and samples $r_b \xleftarrow{\text{R}} \{0,1\}^\rho$. It sets the ciphertext $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$ and $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, \perp$). Finally, the challenger gives (c, σ) to the adversary where $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma = (\sigma', r_b)$.
  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 switches the commitments to be equivocating.
  1. The challenger starts by sampling $(\mathsf{crs}', \tilde{c}', \tilde{\sigma}_0', \tilde{\sigma}_1') \leftarrow \mathsf{BC.SetupEquivocate}'(1^\lambda)$, a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and sets $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$. The challenger gives the security parameter 1^λ and crs to $\mathcal{A}$.
  2. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. The challenger sets $(c', \sigma') \leftarrow (\tilde{c}, \tilde{\sigma}_b')$ and samples $r_b \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. It sets $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$ and $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, \perp)$. Finally, the challenger gives (c, σ) to the adversary where $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma = (\sigma', r_b)$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_2: Same as Hyb_1, except the challenger replaces ct_{1-b} with an encryption of $(1-b, \tilde{\sigma}_{1-b}')$. This is the mode indistinguishability game with bit $\beta = 1$.
  1. The challenger starts by sampling $(\mathsf{crs}', \tilde{c}', \tilde{\sigma}_0', \tilde{\sigma}_1') \leftarrow \mathsf{BC.SetupEquivocate}'(1^\lambda)$, a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and sets $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$. The challenger gives the security parameter 1^λ and crs to $\mathcal{A}$.
  2. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. The challenger sets $(c', \sigma') \leftarrow (\tilde{c}, \tilde{\sigma}_b')$ and samples $r_b \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. It sets $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$ and $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (1-b, \tilde{\sigma}_{1-b}'))$. Finally, the challenger gives (c, σ) to the adversary where $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma = (\sigma', r_b)$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

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

Claim 4.5. *If* Π_{BC}' *satisfies mode indistinguishability, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $\left|\Pr\left[\mathsf{Hyb}_1(\mathcal{A})\right] - \Pr\left[\mathsf{Hyb}_0(\mathcal{A})\right]\right| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks mode indistinguishability for Π_{BC}':

1. Algorithm $\mathcal{B}$ receives security parameter 1^λ and common reference string crs' from the challenger. Algorithm $\mathcal{B}$ samples a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and sets $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on input 1^λ and crs.
2. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which algorithm $\mathcal{B}$ forwards to its challenger. The challenger replies with a challenge (c', σ'). Algorithm $\mathcal{B}$ samples $r_b \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and computes $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$. It constructs $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, \perp)$ and sets $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma = (\sigma', r_b)$. Algorithm $\mathcal{B}$ gives (c, σ) to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

Let β be the bit in the mode indistinguishability game for Π_{BC}'. We consider the two possibilities:

- Suppose $\beta = 0$. In this case, the challenger samples $\mathsf{crs}' \leftarrow \mathsf{BC.SetupBind}'(1^\lambda)$ and $(c', \sigma') \leftarrow \mathsf{BC.Commit}'(\mathsf{crs}', b)$, so algorithm $\mathcal{B}$ perfectly simulates Hyb_0 for $\mathcal{A}$.
- Suppose $\beta = 1$. In this case, the challenger samples $(\mathsf{crs}', \tilde{c}', \tilde{\sigma}_0', \tilde{\sigma}_1') \leftarrow \mathsf{BC.SetupEquivocate}'(1^\lambda)$ and $(c', \sigma') \leftarrow \mathsf{BC.Commit}'(\tilde{c}', \tilde{\sigma}_b')$, so algorithm $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$.

We conclude that algorithm $\mathcal{B}$ wins the mode indistinguishability game with the same advantage ε. □

Claim 4.6. *If* Π_{PKE} *is semantically secure, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks semantic security of Π_{PKE} as follows:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the security parameter 1^λ and public key pk from the challenger.
2. Algorithm $\mathcal{B}$ samples $(\mathsf{crs}', \tilde{c}', \tilde{\sigma}_0', \tilde{\sigma}_1') \leftarrow \mathsf{BC.SetupEquivocate}'(1^\lambda)$ and sets $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on input 1^λ and crs.
3. Algorithm $\mathcal{B}$ outputs a bit $b \in \{0,1\}$. Algorithm $\mathcal{B}$ sets $(c', \sigma') \leftarrow (\tilde{c}, \tilde{\sigma}_b')$. It then samples $r_b \xleftarrow{\text{R}} \{0,1\}^\rho$ and computes $\mathsf{ct}_b \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$
4. Next, algorithm $\mathcal{B}$ makes a challenge query on the pair of messages $\perp$ and $(1-b, \tilde{\sigma}'_{1-b})$. The challenger replies with a ciphertext ct_{1-b}. Algorithm $\mathcal{B}$ sets $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma \leftarrow (\sigma', r_b)$. Algorithm $\mathcal{B}$ gives (c, σ) to $\mathcal{A}$.
5. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

First, the challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ so the distribution of the CRS is distributed exactly according to the specification of Hyb_1 and Hyb_2. It suffices to consider the distribution of the challenge ciphertext. Let β be the bit in the semantic security game for Π_{PKE}. We consider the two possibilities:

- Suppose $\beta = 0$. In this case, the challenger computes $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, \perp)$, so algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1.
- Suppose $\beta = 1$. In this case, the challenger computes $\mathsf{ct}_{1-b} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, (1-b, \tilde{\sigma}'_{1-b}))$, so algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_2.

We conclude that algorithm $\mathcal{B}$ breaks semantic security of Π_{PKE} with the same advantage ε. □

Combining Claims 4.5 and 4.6, mode indistinguishability follows. □

Theorem 4.7 (Extractable in Binding Mode). *If Π_{PKE} is perfectly correct and Π'_{BC} is statistically binding, then Construction 4.2 is extractable in binding mode.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that breaks extractability in binding mode with non-negligible probability $\varepsilon = \varepsilon(\lambda)$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks statistical binding of Π'_{BC}:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the common reference string crs' from the challenger. Algorithm $\mathcal{B}$ then samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. It sets $\mathsf{crs} = (\mathsf{crs}', \mathsf{pk})$ and gives crs to $\mathcal{A}$.
2. Algorithm $\mathcal{A}$ outputs (c, σ, b) where $c = (c', \mathsf{ct}_0, \mathsf{ct}_1)$ and $\sigma = (\sigma', r_b)$.
3. If $b = 1$, then algorithm $\mathcal{B}$ computes $m_0 \leftarrow \mathsf{PKE.Decrypt}(\mathsf{sk}, \mathsf{ct}_0)$ and parses m_0 as $(0, \sigma_0')$. If m_0 does not have this form or $b \neq 1$, then algorithm $\mathcal{B}$ aborts. Otherwise, algorithm $\mathcal{B}$ outputs (c', σ_0', σ').

By construction, algorithm $\mathcal{B}$ perfectly simulates the statistical binding security game for $\mathcal{A}$. Thus, with probability at least ε, the following properties hold:

- $\mathsf{BC.Verify}'(\mathsf{crs}', c', b, \sigma') = 1$ and $\mathsf{ct}_b = \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$.
- $\mathsf{Extract}(\mathsf{td}, c') \neq b$.

We consider two possibilities:

- Suppose $b = 0$. Since $\mathsf{ct}_b = \mathsf{PKE.Encrypt}(\mathsf{pk}, (b, \sigma'); r_b)$, perfect correctness of Π_{PKE} implies that $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_0) = (0, \sigma')$. Moreover, since $\mathsf{BC.Verify}(\mathsf{crs}', c', 0, \sigma') = 1$, it follows that $\mathsf{Extract}(\mathsf{td}, c')$ also outputs 0, which is a contradiction. Thus, this case does not happen.
- Suppose $b = 1$. If $\mathsf{Extract}(\mathsf{td}, c')$ outputs 0, then it must be the case that $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_0)$ outputs $(0, \sigma_0')$ where $\mathsf{BC.Verify}'(\mathsf{crs}', c', 0, \sigma_0') = 1$. Moreover, we have that $\mathsf{BC.Verify}'(\mathsf{crs}', c', 1, \sigma') = 1$. In this case, algorithm $\mathcal{B}$ successfully breaks binding of the commitment scheme.

We conclude that algorithm $\mathcal{B}$ breaks statistical binding of Π'_{BC} with the same advantage ε. □

Corollary 4.8 (One-Time Dual-Mode Bit Commitment with Extraction). *Assuming the existence of public-key encryption, there exists a one-time dual-mode bit commitment with extraction scheme.*

4.2 Hidden-Bits Generator Construction

We now describe our hidden-bits generator based on public-key encryption. The construction replaces the one-time dual-mode bit commitment scheme in Construction 3.1 with a scheme that supports extraction. This allows basing security on *polynomial* somewhere soundness of the underlying BARG rather than adaptive soundness (which necessitated sub-exponential hardness). The construction is identical to Construction 3.1, except we modify the binding analysis to rely on somewhere soundness of the BARG (and extraction) rather than adaptive soundness.

Construction 4.9 (Hidden-Bits Generator from Polynomial-Hard Batch Arguments)**.** The construction is identical to Construction 3.1, except for the following two differences:

- We replace the one-time dual-mode bit commitment scheme with a one-time dual-mode bit commitment scheme with extraction $\Pi_{\mathsf{BC}} = (\mathsf{SetupBind}, \mathsf{SetupEquivocate}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Extract})$. Note that the Extract algorithm is only needed in the security analysis, so the scheme semantics are identical to Construction 3.1.
- We replace the adaptively-sound BARG with a somewhere sound BARG. Functionally-speaking, the only difference in Setup is when sampling the CRS for the BARG, the scheme additionally provides a dummy index 1. Namely, the Setup algorithm samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, 1)$.

Correctness and security analysis. We now state the correctness and security theorems for Construction 4.9. The correctness and hiding proofs are identical to the respective proofs for Construction 3.1 (Theorem 3.2 and Theorem 3.4). We defer the binding proof to Section 4.3.

Theorem 4.10 (Correctness)**.** *If Π_{BARG} is complete and Π_{BC} is correct, then Construction 4.9 is correct.*

Proof. Follows by the same argument as in the proof of Theorem 3.2. □

Theorem 4.11 (Somewhat Computational Binding)**.** *Suppose $\kappa(\lambda, m) \le m^\delta \cdot p(\lambda)$ for some constant $\delta < 1$ and a fixed polynomial $p(\cdot)$. If Π_{BC} is extractable in binding mode and Π_{BARG} is somewhere sound, then Construction 4.9 satisfies somewhat computational binding.*

Theorem 4.12 (Computational Hiding)**.** *Suppose Π_{BC} satisfies mode indistinguishability and Π_{LRwPRF} is a secure leakage-resilient weak PRF. If $\ell(\lambda, m) \ge \ell_{\mathsf{BARG}}(\lambda, ms_{\mathsf{int}}, |C_{\mathcal{R}}|)$, Construction 4.9 satisfies computational hiding.*

Proof. Follows by the same argument as in the proof of Theorem 3.4 (see Section 3.2). □

Parameter instantiations. We can instantiate the underlying primitives following the same methodology as in Section 3. This yields the following corollary:

Corollary 4.13 (NIZKs from Somewhere-Sound BARGs and PKE)**.** *Assuming the existence of public-key encryption and somewhere-sound BARGs for* NP *(Definitions 2.8 and 2.9), there exists a computational NIZK argument for* NP.

4.3 Proof of Theorem 4.11 (Somewhat Computational Binding)

Let crs be a common reference string in the support of $\mathsf{Setup}(1^\lambda, 1^m)$ (parsed according to Eq. (3.1)). We define the set $\mathcal{V}^{\mathsf{crs}} \subseteq \{0,1\}^m$ exactly as in Eq. (3.7) in the proof of Theorem 3.3:

$$\mathcal{V}^{\mathsf{crs}} := \{(C(k, \mathbf{z}_1), \dots, C(k, \mathbf{z}_m)) \mid k \in \{0,1\}^\kappa\}$$

We now show that each of the requirements of Definition 2.10 is satisfied:

Output sparsity. Output sparsity of Construction 4.9 follows by the same argument as in the proof of Theorem 3.3 (see Section 3.1).

Computational binding. Let $\mathcal{A}$ be an efficient adversary for the computational binding security game for Construction 4.9. Consider an execution of the computational binding game between a challenger and $\mathcal{A}$:

- On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.
- The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically, the challenger starts by sampling the following collection of common reference strings for the bit commitment scheme:
  - **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \mathsf{td}^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
  - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample the bit-commitment CRS $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
  - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample the bit-commitment CRS $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

  Next, the challenger samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\text{R}} \{0,1\}^n$ and for each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$. Let $s_{\mathsf{int}} = |S_{\mathsf{int}}| = s - (\kappa + n)$ be the number of non-input wires in C. Finally, the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{m s_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, 1)$ and constructs the common reference string

  $$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\mathsf{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right)$$

  and gives crs to algorithm $\mathcal{A}$.
- Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

  $$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

- The output of the experiment is $b = 1$ if $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$ and $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, where $\mathcal{V}_I^{\mathsf{crs}} := \{\mathbf{r}_I : \mathbf{r} \in \mathcal{V}^{\mathsf{crs}}\}$. Otherwise, the output is $b = 0$.

In an execution of the computational binding game, we define the following quantities:

- For each $j \in S_{\mathsf{key}}$, let $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right)$. We will use the convention that for all $j \in S_{\mathsf{key}}$, $y^{(i,j)} := y^{(\mathsf{key},j)}$.
- For each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, let $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(i,j)}, c^{(i,j)}\right)$.
- For each $i \in I$ and $j \in [s_{\mathsf{int}}]$, let j_{L} and j_{R} be the wire indices that determine the value of the jth non-input wire $j_{\mathrm{OUT}} = (\kappa + n) + j$ in the circuit C.

Next, we define the event Bad as follows:

$$\mathsf{Bad} \text{ occurs } \iff (\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1) \wedge \left(\exists i \in I, j \in [s_{\mathsf{int}}] : y^{(i,j_{\mathrm{OUT}})} \neq \mathrm{NAND}(y^{(i,j_{\mathrm{L}})}, y^{(i,j_{\mathrm{R}})})\right). \tag{4.1}$$

Observe that the event Bad is efficiently-checkable (with knowledge of the trapdoors $\mathsf{td}^{(\mathsf{key},j)}$ and $\mathsf{td}^{(i,j)}$ for all i, j). In the following, we start by showing that in the computational binding experiment, $\Pr[\mathsf{Bad}] = \mathsf{negl}(\lambda)$. To do this, we proceed via a hybrid argument:

- Hyb_0: This is the real computational binding experiment, except the output is 1 only if event Bad occurs. Specifically, the experiment proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Specifically, let $s_{\mathsf{int}} = |S_{\mathsf{int}}| = s-(\kappa+n)$ be the number of non-input wires in C. The challenger starts by sampling the BARG CRS $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{m s_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, 1)$. Then, the challenger samples the following collection of common reference strings for the bit commitment scheme:

* **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \mathsf{td}^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
* **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
* **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

Next, the challenger samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$ and for each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$. The challenger constructs the common reference string

$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\mathsf{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right)$$

and gives crs to algorithm $\mathcal{A}$.

– Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

– For each $j \in S_{\mathsf{key}}$, the challenger computes $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right)$. For each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, the challenger computes $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(i,j)}, c^{(i,j)}\right)$.

– The output of the experiment is 1 if event Bad (Eq. (4.1)) occurs: namely, if $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$ and there exists $i \in I$ and $j \in [s_{\mathsf{int}}]$ such that $y^{(i,j_{\mathrm{OUT}})} \neq \mathrm{NAND}(y^{(i,j_{\mathrm{L}})}, y^{(i,j_{\mathrm{R}})})$.

• Hyb_1: Same as Hyb_0 except the challenger now samples a random index $t^* \xleftarrow{\mathrm{R}} [ms_{\mathsf{int}}]$, and only outputs 1 if the $(t^*)^{\text{th}}$ wire is incorrectly assigned. Specifically, the experiment proceeds as follows:

– On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.

– The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. The challenger samples $t^* \xleftarrow{\mathrm{R}} [ms_{\mathsf{int}}]$ and the BARG CRS $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, 1)$. The remaining components of crs are sampled exactly as in Hyb_0 (same for all hybrids). The challenger gives crs to algorithm $\mathcal{A}$.

– Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

– For each $j \in S_{\mathsf{key}}$, the challenger computes $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right)$. For each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, the challenger computes $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(i,j)}, c^{(i,j)}\right)$.

– The challenger parses $t^* = (i^* - 1) \cdot s_{\mathsf{int}} + j^*$ for some $i^* \in [m]$ and $j^* \in [s_{\mathsf{int}}]$. If $i^* \notin I$, the challenger outputs 0. Let $j_{\mathrm{L}}^*, j_{\mathrm{R}}^*$ be the wire indices that determine the value of the $(j^*)^{\text{th}}$ non-input wire $j_{\mathrm{OUT}}^* = (\kappa + n) + j^*$ of C. The output is 1 if $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$ and $y^{(i^*, j_{\mathrm{OUT}}^*)} \neq \mathrm{NAND}\left(y^{(i^*, j_{\mathrm{L}}^*)}, y^{(i^*, j_{\mathrm{R}}^*)}\right)$.

• Hyb_2: Same as Hyb_1 except the challenger programs the BARG to be somewhere sound on index t^*. Specifically, the experiment proceeds as follows:

– On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the output length 1^m.

– The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. The challenger samples $t^* \xleftarrow{\mathrm{R}} [ms_{\mathsf{int}}]$ and $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, t^*)$. The remaining components of crs are sampled exactly as in Hyb_0 (same for all hybrids). The challenger gives crs to algorithm $\mathcal{A}$.

– Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

– For each $j \in S_{\mathsf{key}}$, the challenger computes $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right)$. For each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, the challenger computes $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(i,j)}, c^{(i,j)}\right)$.

– The challenger parses $t^* = (i^* - 1) \cdot s_{\text{int}} + j^*$ for some $i^* \in [m]$ and $j^* \in [s_{\text{int}}]$. If $i^* \notin I$, the challenger outputs 0. Let $j^*_{\text{L}}, j^*_{\text{R}}$ be the wire indices that determine the value of the $(j^*)^{\text{th}}$ non-input wire $j^*_{\text{OUT}} = (\kappa + n) + j^*$ of C. The output is 1 if $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$ and $y^{(i^*, j^*_{\text{OUT}})} \neq \text{NAND}\big(y^{(i^*, j^*_{\text{L}})}, y^{(i^*, j^*_{\text{R}})}\big)$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of an execution of Hyb_i with adversary $\mathcal{A}$. By construction, observe that $\Pr[\mathsf{Bad}] = \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$. We now analyze each of the hybrid experiments.

Claim 4.14. *It holds that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \geq 1/(ms_{\text{int}}) \cdot \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.

Proof. By construction, the view of adversary $\mathcal{A}$ is identically distributed in Hyb_0 and Hyb_1 (since none of the challenger's messages depend on t^*). Suppose $\mathsf{Hyb}_0(\mathcal{A}) = 1$. Then, at the end of the experiment, $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$ and there exists some $i \in I \subseteq [m]$ and $j \in [s_{\text{int}}]$ such that $y^{(i, j_{\text{OUT}})} \neq \text{NAND}\big(y^{(i, j_{\text{L}})}, y^{(i, j_{\text{R}})}\big)$. Since the challenger in Hyb_1 samples t^* uniformly at random from $[ms_{\text{int}}]$, so with probability $1/(ms_{\text{int}})$, it will be the case that $t^* = i(s_{\text{int}} - 1) + j$. In this case, the output in $\mathsf{Hyb}_1(\mathcal{A})$ is also 1, and the claim holds. □

Claim 4.15. *Suppose* Π_{BARG} *satisfies index hiding. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks index hiding of the BARG:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ chooses the output length 1^m.

2. Define s_{int} as in Hyb_1 and Hyb_2. Algorithm $\mathcal{B}$ now samples a random index $t^* \xleftarrow{\text{R}} [ms_{\text{int}}]$. It outputs $1^{ms_{\text{int}}}$ as the number of instances, $1^{|C_{\mathcal{R}}|}$ as the bound on the circuit size, and the indices 1 and t^* as its challenge to the index hiding challenger.

3. The challenger replies with $\mathsf{crs}_{\text{BARG}}$. Algorithm $\mathcal{B}$ now simulates the rest of the common reference string according to the specification of Hyb_0 and Hyb_1. Specifically, it starts by sampling the following collection of common reference strings for the bit commitment scheme:

   - **CRS for the key:** For $j \in S_{\text{key}}$, sample $\big(\mathsf{crs}_{\text{BC}}^{(\text{key},j)}, \mathsf{td}^{(\text{key},j)}\big) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\text{eval}}$, sample the bit-commitment CRS $\big(\mathsf{crs}_{\text{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\big) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\text{int}}$, sample the bit-commitment CRS $\big(\mathsf{crs}_{\text{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\big) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

   Next, algorithm $\mathcal{B}$ samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\text{R}} \{0,1\}^n$ and for each $i \in [m]$ and $j \in S_{\text{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\big(\mathsf{crs}_{\text{BC}}^{(i,j)}, z_{i,j-\kappa}\big)$. Finally, algorithm $\mathcal{B}$ constructs the common reference string

   $$\mathsf{crs} = \Big((\mathbf{z}_1, \ldots, \mathbf{z}_m), \big\{\mathsf{crs}_{\text{BC}}^{(\text{key},j)}\big\}_{j \in S_{\text{key}}}, \big\{\mathsf{crs}_{\text{BC}}^{(i,j)}\big\}_{i \in [m], j \in S_{\text{eval}} \cup S_{\text{int}}}, \big\{\big(c^{(i,j)}, \sigma^{(i,j)}\big)\big\}_{i \in [m], j \in S_{\text{eval}}}, \mathsf{crs}_{\text{BARG}}\Big)$$

   and gives crs to algorithm $\mathcal{A}$.

4. Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

   $$\pi = \big(\pi_{\text{BARG}}, \{c^{(\text{key},j)}\}_{j \in S_{\text{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\text{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\big).$$

5. For each $j \in S_{\text{key}}$, algorithm $\mathcal{B}$ computes $y^{(\text{key},j)} \leftarrow \mathsf{BC.Extract}\big(\mathsf{td}^{(\text{key},j)}, c^{(\text{key},j)}\big)$. For each $i \in I$ and $j \in S_{\text{eval}} \cup S_{\text{int}}$, algorithm $\mathcal{B}$ computes $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\big(\mathsf{td}^{(i,j)}, c^{(i,j)}\big)$.

6. Algorithm $\mathcal{B}$ parses $t^* = (i^* - 1) \cdot s_{\text{int}} + j^*$ for some $i^* \in [m]$ and $j^* \in [s_{\text{int}}]$. If $i^* \notin I$, algorithm $\mathcal{B}$ outputs 0. Let $j^*_{\text{L}}, j^*_{\text{R}}$ be the wire indices that determine the value of the $(j^*)^{\text{th}}$ non-input wire $j^*_{\text{OUT}} = (\kappa + n) + j^*$ of C. Algorithm $\mathcal{B}$ outputs 1 if $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$ and $y^{(i^*, j^*_{\text{OUT}})} \neq \text{NAND}\big(y^{(i^*, j^*_{\text{L}})}, y^{(i^*, j^*_{\text{R}})}\big)$ and 0 otherwise.

Let $\beta \in \{0,1\}$ be the bit in the index hiding game. We consider two possibilities:

- When $\beta = 0$, the challenger constructs the BARG CRS as $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, 1)$. In this case, the output of algorithm $\mathcal{B}$ is distributed according to $\mathsf{Hyb}_1(\mathcal{A})$.
- When $\beta = 1$, the challenger constructs the BARG CRS as $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{|C_{\mathcal{R}}|}, t^*)$. In this case, the output of algorithm $\mathcal{B}$ is distributed according to $\mathsf{Hyb}_2(\mathcal{A})$.

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

Claim 4.16. *Suppose* Π_{BC} *is extractable in binding mode and* Π_{BARG} *is somewhere sound. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, it holds that* $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Proof. Suppose $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary that breaks somewhere binding of the BARG (or extractability of the commitment scheme):

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on the same security parameter 1^λ. Algorithm $\mathcal{A}$ starts by outputting the string length 1^m.
2. Algorithm $\mathcal{B}$ computes s_{int} and $|C_{\mathcal{R}}|$ as in Construction 4.9. It also samples an index $t^* \xleftarrow{\mathrm{R}} [ms_{\mathsf{int}}]$. It outputs $1^{ms_{\mathsf{int}}}$ as the bound on the number of instances and $1^{|C_{\mathcal{R}}|}$ as the size of the circuit.
3. The challenger replies with $\mathsf{crs}_{\mathsf{BARG}}$. Algorithm $\mathcal{B}$ now simulates the rest of the common reference string according to the specification of Hyb_0 and Hyb_1. Specifically, it starts by sampling the following collection of common reference strings for the bit commitment scheme:
   - **CRS for the key:** For $j \in S_{\mathsf{key}}$, sample $\left(\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}, \mathsf{td}^{(\mathsf{key},j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the evaluation point:** For $i \in [m]$ and $j \in S_{\mathsf{eval}}$, sample the bit-commitment CRS $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.
   - **CRS for the non-input wires:** For $i \in [m]$ and $j \in S_{\mathsf{int}}$, sample bit-commitment CRS $\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\right) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$.

   Next, algorithm $\mathcal{B}$ samples $\mathbf{z}_1, \ldots, \mathbf{z}_m \xleftarrow{\mathrm{R}} \{0,1\}^n$ and for each $i \in [m]$ and $j \in S_{\mathsf{eval}}$, it computes $(c^{(i,j)}, \sigma^{(i,j)}) \leftarrow \mathsf{BC.Commit}\left(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\right)$. Finally, algorithm $\mathcal{B}$ constructs the common reference string

$$\mathsf{crs} = \left((\mathbf{z}_1, \ldots, \mathbf{z}_m), \left\{\mathsf{crs}_{\mathsf{BC}}^{(\mathsf{key},j)}\right\}_{j \in S_{\mathsf{key}}}, \left\{\mathsf{crs}_{\mathsf{BC}}^{(i,j)}\right\}_{i \in [m], j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}}, \left\{\left(c^{(i,j)}, \sigma^{(i,j)}\right)\right\}_{i \in [m], j \in S_{\mathsf{eval}}}, \mathsf{crs}_{\mathsf{BARG}}\right)$$

   and gives crs to algorithm $\mathcal{A}$.
4. Algorithm $\mathcal{A}$ outputs a tuple $(I, \mathbf{r}_I, \pi)$, where

$$\pi = \left(\pi_{\mathsf{BARG}}, \{c^{(\mathsf{key},j)}\}_{j \in S_{\mathsf{key}}}, \{c^{(i,j)}\}_{i \in I, j \in S_{\mathsf{int}}}, \{\sigma^{(i,s)}\}_{i \in I}\right).$$

5. Algorithm $\mathcal{B}$ forms the statements $x^{(i,j)}$ for $i \in I$ and $j \in [s_{\mathsf{int}}]$ according to Eq. (3.2) and outputs the circuit $C_{\mathcal{R}}$, the statements $(x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}$, and the proof π_{BARG}.

Before proceeding with the analysis, we also define the following quantities from Hyb_2:

- For each $j \in S_{\mathsf{key}}$, let $y^{(\mathsf{key},j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(\mathsf{key},j)}, c^{(\mathsf{key},j)}\right)$. For each $i \in I$ and $j \in S_{\mathsf{eval}} \cup S_{\mathsf{int}}$, let $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\left(\mathsf{td}^{(i,j)}, c^{(i,j)}\right)$.
- Let $j_{\mathrm{L}}^*, j_{\mathrm{R}}^*$ be the wire indices that determine the value of the $(j^*)^{\text{th}}$ non-input wire $j_{\mathsf{OUT}}^* = (\kappa + n) + j^*$ of C.
- Let $t^* = (i^* - 1) \cdot s_{\mathsf{int}} + j^*$ for some $i^* \in [m]$ and $j^* \in [s_{\mathsf{int}}]$.

In the somewhere-soundness game, the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^{ms_{\mathsf{int}}}, 1^{C_{\mathcal{R}}}, t^*)$. Thus, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_2 for $\mathcal{A}$. By assumption, with probability ε, algorithm $\mathcal{B}$ outputs a proof π such that the following properties hold:

(i) $i^* \in I$ and $y^{(i^*, j^*_{\text{OUT}})} \neq \text{NAND}\big(y^{(i^*, j^*_{\text{L}})}, y^{(i^*, j^*_{\text{R}})}\big)$

(ii) $\mathsf{BARG.Verify}\big(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathcal{R}}, (x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}, \pi_{\mathsf{BARG}}\big) = 1$.

Property (ii) follows by construction of the Verify algorithm. Consider now the $(t^*)^{\text{th}}$ statement $x^{(i^*,j^*)}$. From Eq. (3.2), we have that

$$x^{(i^*,j^*)} = \left(\mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*_{\text{L}})}, \mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*_{\text{R}})}, \mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*_{\text{OUT}})}, c^{(i^*,j^*_{\text{L}})}, c^{(i^*,j^*_{\text{R}})}, c^{(i^*,j^*_{\text{OUT}})}\right).$$

Take any candidate witness $w^{(i^*,j^*)} = (b_{\text{L}}, b_{\text{R}}, b_{\text{OUT}}, \sigma_{\text{L}}, \sigma_{\text{R}}, \sigma_{\text{OUT}})$ and consider $\mathcal{R}\big(x^{(i^*,j^*)}, y^{(i^*,j^*)}\big)$:

- Suppose for all $\mathsf{pos} \in \{\text{L}, \text{R}, \text{OUT}\}$, it holds that $\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*_{\mathsf{pos}})}, c^{(i^*,j^*_{\mathsf{pos}})}, b_{\mathsf{pos}}, \sigma_{\mathsf{pos}}\big) = 1$. Moreover, recall that $y^{(i^*,j^*_{\mathsf{pos}})} = \mathsf{BC.Extract}\big(\mathsf{td}^{(i^*,j^*_{\mathsf{pos}})}, c^{(i^*,j^*_{\mathsf{pos}})}\big)$. By extractability of the bit commitment scheme, with overwhelming probability over the choice of $\mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*)}$, it holds that $b_{\mathsf{pos}} = y^{(i^*,j^*_{\mathsf{pos}})}$.
- By Property (i), if $b_{\mathsf{pos}} = y^{(i^*,j^*_{\mathsf{pos}})}$ for all $\mathsf{pos} \in \{\text{L}, \text{R}, \text{OUT}\}$, then $b_{\text{OUT}} \neq \text{NAND}(b_{\text{L}}, b_{\text{R}})$.

We conclude that either there exists $\mathsf{pos} \in \{\text{L}, \text{R}, \text{OUT}\}$ such that $\mathsf{BC.Verify}\big(\mathsf{crs}_{\mathsf{BC}}^{(i^*,j^*_{\mathsf{pos}})}, c^{(i^*,j^*_{\mathsf{pos}})}, b_{\mathsf{pos}}, \sigma_{\mathsf{pos}}\big) \neq 1$ or $b_{\text{OUT}} \neq \text{NAND}(b_{\text{L}}, b_{\text{R}})$. In both cases, $\mathcal{R}\big(x^{(i^*,j^*)}, y^{(i^*,j^*)}\big) = 0$. We conclude that $(C_{\mathcal{R}}, x^{(i^*,j^*)}) \notin \mathcal{L}_{\mathsf{SAT}}$. However, by Property (ii), π_{BARG} is a valid proof for the statements $(x^{(i,j)})_{i \in I, j \in [s_{\mathsf{int}}]}$. Correspondingly, algorithm $\mathcal{B}$ breaks somewhere soundness of the BARG whenever $\mathsf{Hyb}_2(\mathcal{A}) = 1$ and the claim holds. □

Completing the proof of Theorem 4.11. Combining Claims 4.14 to 4.16, we conclude that the value $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. Correspondingly, this means that in the computational binding game, $\Pr[\mathsf{Bad}] = \mathsf{negl}(\lambda)$. Consider now the probability that the output in the computational binding game is 1 *and* the event Bad does *not* occur. This means that the adversary outputs $(I, \mathbf{r}_I, \pi)$ where the following conditions all occur:

(i) $\mathbf{r}_I \notin \mathcal{V}_I^{\mathsf{crs}}$;

(ii) $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$; and

(iii) For all $i \in I$ and $j \in [s_{\mathsf{int}}]$, $y^{(i,j_{\text{OUT}})} = \text{NAND}\big(y^{(i,j_{\text{L}})}, y^{(i,j_{\text{R}})}\big)$, where $y^{(i,j)} \leftarrow \mathsf{BC.Extract}\big(\mathsf{td}^{(i,j)}, c^{(i,j)}\big)$. As usual, we adopt the convention that for $j \in S_{\mathsf{key}}$, $y^{(i,j)} := y^{(\mathsf{key},j)}$, $\mathsf{td}^{(i,j)} := \mathsf{td}^{(\mathsf{key},j)}$, and $c^{(i,j)} := c^{(\mathsf{key},j)}$.

Let $k = y^{(\mathsf{key},1)} y^{(\mathsf{key},2)} \cdots y^{(\mathsf{key},\kappa)}$. We appeal to extractability of the bit commitment scheme to complete the proof:

- In the computational binding game, the challenger samples $\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, \mathsf{td}^{(i,j)}\big) \leftarrow \mathsf{BC.SetupBind}(1^\lambda)$ and $c^{(i,j)} \leftarrow \mathsf{BC.Commit}\big(\mathsf{crs}_{\mathsf{BC}}^{(i,j)}, z_{i,j-\kappa}\big)$ for all $i \in I$ and $j \in S_{\mathsf{eval}}$. Correctness and extractability of Π_{BC} imply that with overwhelming probability, $y^{(i,j)} = \mathsf{BC.Extract}\big(\mathsf{td}^{(i,j)}, c^{(i,j)}\big) = z_{i,j-\kappa}$.
- Property (iii) now implies that for all $i \in I$, it holds that $y^{(i,s)} = C(k, \mathbf{z}_i)$. Since it holds that $y^{(i,s)} = \mathsf{BC.Extract}\big(\mathsf{td}^{(i,s),c^{(i,s)}}\big)$, we appeal again to extractability of the commitment scheme to conclude that with overwhelming probability, the only valid opening for each commitment $c^{(i,s)}$ is to the value $y^{(i,s)} = C(k, \mathbf{z}_i)$.
- From Property (ii), $\mathsf{Verify}(\mathsf{crs}, I, \mathbf{r}_I, \pi) = 1$, which means $\mathcal{A}$ produces valid openings $\sigma^{(i,s)}$ of the commitments $c^{(i,s)}$ to the values r_i. As argued before, the only valid opening for each $c^{(i,s)}$ is to the value $y^{(i,s)} = C(k, \mathbf{z}_i)$. Thus, it must be the case that $r_i = C(k, \mathbf{z}_i)$ for all $i \in I$.

Thus, we have established that $r_i = C(k, \mathbf{z}_i)$ for all $i \in I$. By definition of $\mathcal{V}_I^{\mathsf{crs}}$ (see Eq. (3.7)), this means $\mathbf{r}_I \in \mathcal{V}_I^{\mathsf{crs}}$, which contradicts Property (i). Hence, we conclude that with overwhelming probability over the randomness used to sample the common reference strings $\mathsf{crs}_{\mathsf{BC}}^{(i,j)}$, the probability that the output in the computational binding game is 1 is $\mathsf{negl}(\lambda)$. The claim holds. □

5 BARGs with Adaptive Soundness via Sub-Exponential Hardness

In this section, we show that using complexity leveraging, any sub-exponentially-secure somewhere-sound BARG (Definition 2.9) is also adaptively sound (Definition 2.8). Specifically, the analysis relies on sub-exponential index hiding. We describe our construction and analysis below:

Construction 5.1 (Adaptively-Sound BARG from a Somewhere-Sound BARG). Let λ be a security parameter and s be a circuit-size parameter. Let $\Pi_{\mathsf{SSBARG}} = (\mathsf{SSBARG.Setup}, \mathsf{SSBARG.Prove}, \mathsf{SSBARG.Verify})$ be a somewhere-sound batch argument for Boolean circuit satisfiability. Let $\lambda_{\mathsf{SSBARG}} = \lambda_{\mathsf{SSBARG}}(\lambda, s)$ be a polynomial which will be set in the security proof (Theorem 5.4). We construct an adaptively-secure batch argument for Boolean circuit satisfiability $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^T, 1^s)$: Output $\mathsf{crs} \leftarrow \mathsf{SSBARG.Setup}(1^{\lambda_{\mathsf{SSBARG}}(\lambda,s)}, 1^T, 1^s, 1)$.
- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), (\mathbf{w}_1, \ldots, \mathbf{w}_t))$: Output
$$\pi \leftarrow \mathsf{SSBARG.Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), (\mathbf{w}_1, \ldots, \mathbf{w}_t)).$$
- $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi)$: Output $b \leftarrow \mathsf{SSBARG.Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi)$.

Theorem 5.2 (Completeness). *If Π_{SSBARG} is complete, then Construction 5.1 is complete.*

Proof. Follows immediately from completeness of Π_{SSBARG}. □

Theorem 5.3 (Succinctness). *If Π_{SSBARG} is succinct, then Construction 5.1 is succinct.*

Proof. Take any common reference string crs in the support of the setup algorithm $\mathsf{Setup}(1^\lambda, 1^T, 1^s)$. Then, crs is in the support of $\mathsf{SSBARG.Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, 1)$. Let $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be a Boolean circuit of size at most s. Take any sequence of statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$ and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \{0,1\}^h$, where $t \le T$. Let $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), (\mathbf{w}_1, \ldots, \mathbf{w}_t))$. By succinctness of Π_{SSBARG}, $|\pi| \le p(\lambda_{\mathsf{SSBARG}} + \log t + s)$ for some fixed polynomial $p(\cdot)$. Next, $\lambda_{\mathsf{SSBARG}} = (\lambda + s)^c$ for some constant $c \in \mathbb{N}$. Thus, we conclude that $|\pi| \le p((\lambda + s)^c + \log t + s) \le q(\lambda + \log t + s)$, for a fixed polynomial q that depends only on the polynomial p and the constant c. □

Theorem 5.4 (Adaptive Soundness). *Suppose Π_{SSBARG} is a somewhere-sound BARG which satisfies sub-exponential index hiding with parameter $c > 1$ and somewhere soundness. Suppose moreover that $\lambda_{\mathsf{SSBARG}}(\lambda, s) = (\lambda + s)^c$. Then, Construction 5.1 is adaptively sound.*

Proof. Let $\mathcal{A}$ be an efficient adversary for Π_{BARG} in the adaptive soundness game. Without loss of generality, we assume adversary $\mathcal{A}$ always outputs a fixed value $1^{T(\lambda)}, 1^{s(\lambda)}$ for each value of λ.[6] We now define a sequence of games between a challenger and the adversary $\mathcal{A}$:

- Hyb_0: This is the real adaptive soundness game. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the bound on the number of instances 1^T and the bound on the circuit size 1^s.
  - The challenger sets $\lambda_{\mathsf{SSBARG}} = \lambda_{\mathsf{SSBARG}}(\lambda, s) = (\lambda + s)^c$ and samples a common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, 1)$ and gives crs to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, a collection of $t \le T$ statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and a proof π.

[6]This is without loss of generality since an efficient adversary always outputs T and s that is bounded by *some* polynomial $T_{\max}$ and $s_{\max}$ in the security parameter λ. Any algorithm $\mathcal{A}$ that succeeds in the adaptive soundness game with advantage ε implies an adversary $\mathcal{B}$ that *always* outputs a fixed value of $T(\lambda) \le T_{\max}(\lambda)$ and $s(\lambda) \le s_{\max}(\lambda)$ and succeeds with advantage at least $\varepsilon/(T_{\max} s_{\max})$. Strictly speaking, the values of $T(\lambda)$ and $s(\lambda)$ would be provided as "non-uniform" advice to the reduction algorithms arising in the security proof.

  - The output of the game is 1 if $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$ and for some $i \in [t]$, $(C, \mathbf{x}_i) \notin \mathcal{L}_{\mathsf{SAT}}$. Otherwise, the output is 0.

- Hyb_1: This is Hyb_0, except the challenger guesses a specific index $i^* \xleftarrow{\mathrm{R}} [T]$ where the statement is false. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the bound on the number of instances 1^T and the bound on the circuit size 1^s.
  - The challenger sets $\lambda_{\mathsf{SSBARG}} = \lambda_{\mathsf{SSBARG}}(\lambda, s) = (\lambda + s)^c$ and then samples $i^* \xleftarrow{\mathrm{R}} [T]$. It constructs the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, 1)$ and gives crs to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, a collection of $t \leq T$ statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and a proof π.
  - The output of the game is 1 if $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$, $i^* \leq t$, and $(C, \mathbf{x}_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$. Otherwise, the output is 0.

- Hyb_2: This is Hyb_1, except the challenger samples the CRS to be somewhere sound on index i^*. Specifically, the game proceeds as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the bound on the number of instances 1^T and the bound on the circuit size 1^s.
  - The challenger sets $\lambda_{\mathsf{SSBARG}} = \lambda_{\mathsf{SSBARG}}(\lambda, s) = (\lambda + s)^c$ and then samples $i^* \xleftarrow{\mathrm{R}} [T]$. It constructs the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, i^*)$ and gives crs to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, a collection of $t \leq T$ statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and a proof π.
  - The output of the game is 1 if $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$, $i^* \leq t$, and $(C, \mathbf{x}_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$. Otherwise, the output is 0.

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

Lemma 5.5. *It holds that* $\Pr\left[\mathsf{Hyb}_1(\mathcal{A}) = 1\right] \geq 1/T \cdot \Pr\left[\mathsf{Hyb}_0(\mathcal{A}) = 1\right]$.

Proof. By construction, the view of adversary $\mathcal{A}$ is identically distributed in Hyb_0 and Hyb_1. Suppose $\mathsf{Hyb}_0(\mathcal{A}) = 1$. Then, the adversary $\mathcal{A}$ outputs a circuit C, statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and a proof π such that there exists $i \in [t]$ where $(C, \mathbf{x}_i) \notin \mathcal{L}_{\mathsf{SAT}}$, $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$, and $t \leq T$. Consider the output in Hyb_1. In Hyb_1, the challenger samples the index $i^* \xleftarrow{\mathrm{R}} [T]$, so with probability $1/T$, it holds that $i^* = i$. In this case, the output in Hyb_1 is also 1. Correspondingly, we conclude that

$$\Pr\left[\mathsf{Hyb}_1(\mathcal{A}) = 1\right] \geq 1/T \cdot \Pr\left[\mathsf{Hyb}_0(\mathcal{A}) = 1\right]. \qquad \square$$

Lemma 5.6. *Suppose* Π_{SSBARG} *satisfies sub-exponential index hiding with parameter* $c > 1$ *and* $\lambda_{\mathsf{SSBARG}}(\lambda, s) = (\lambda + s)^c$. *Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists a polynomial $q = q(\lambda)$, and an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

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

Let $\lambda_{\mathcal{A}} \in \mathbb{N}$ be the constant beyond which the polynomial $q(\lambda)$ is monotone (i.e., for all $\lambda, \lambda' \geq \lambda_{\mathcal{A}}$, if holds that $q(\lambda) \geq q(\lambda')$ if and only if $\lambda \geq \lambda'$). Define now the infinite set $\Lambda_{\mathcal{B}} \subseteq \mathbb{N}$

$$\Lambda_{\mathcal{B}} = \{\lambda_{\mathsf{SSBARG}}(\lambda, s(\lambda)) : \lambda \in \Lambda_{\mathcal{A}} \wedge \lambda \geq \lambda_{\mathcal{A}}\}.$$

We use $\mathcal{A}$ to construct a sub-exponential-time (non-uniform) adversary $\mathcal{B}$ for the index hiding game for Π_{SSBARG}. The advice string associated with $\lambda_{\mathsf{SSBARG}} \in \Lambda_{\mathsf{SSBARG}}$ is some parameter $\lambda \in \Lambda_{\mathcal{A}}$ and $\lambda \geq \lambda_{\mathcal{A}}$ where $\lambda_{\mathsf{SSBARG}} = (\lambda + s(\lambda))^c$:

1. On input the security parameter $1^{\lambda_{\mathsf{SSBARG}}}$ (and advice string 1^λ), algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and receives the bound on the number of instances 1^T and the bound on the circuit size 1^s.
2. Algorithm $\mathcal{B}$ samples $i^* \xleftarrow{\text{R}} [T]$ and outputs $1^T, 1^s$ along with the challenge indices 1 and i^* to the challenger. The challenger replies with crs. Algorithm $\mathcal{B}$ gives 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, statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and a proof π.
4. If $i^* > t$ or $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 0$, then algorithm $\mathcal{B}$ halts with output 0.
5. Otherwise, algorithm $\mathcal{B}$ checks that for every $\mathbf{w} \in \{0,1\}^h$, $C(\mathbf{x}_{i^*}, \mathbf{w}) = 0$. It outputs 1 if so and 0 otherwise.

Let $\beta \in \{0,1\}$ be the bit in the index hiding game. We consider the two possibilities:

- When $\beta = 0$, the challenger constructs the CRS as $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, 1)$, which coincides with the specification in Hyb_1. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr\left[\mathsf{Hyb}_1(\mathcal{A}) = 1\right]$.
- When $\beta = 1$, the challenger constructs the CRS as $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, i^*)$, which coincides with the specification in Hyb_2. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr\left[\mathsf{Hyb}_2(\mathcal{A}) = 1\right]$.

Thus, for all $\lambda_{\mathsf{SSBARG}} \in \Lambda_{\mathcal{B}}$, algorithm $\mathcal{B}$ succeeds with advantage

$$\left|\Pr\left[\mathsf{Hyb}_2(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Hyb}_1(\mathcal{A}) = 1\right]\right| \geq 1/q(\lambda) \geq 1/q(\lambda_{\mathsf{SSBARG}}),$$

since $\lambda_{\mathsf{SSBARG}} \geq \lambda \geq \lambda_{\mathcal{A}}$, so $q(\lambda_{\mathsf{SSBARG}}) \geq q(\lambda)$. Finally, the running time of $\mathcal{B}$ is $2^h \cdot \mathsf{poly}(\lambda + s(\lambda))$. By definition, $h \leq s(\lambda) \leq \lambda_{\mathsf{SSBARG}}^{1/c}$, so the overall running time of $\mathcal{B}$ is bounded by $2^h \cdot \mathsf{poly}(s) \leq 2^{\lambda_{\mathsf{SSBARG}}^{1/c}} \cdot \mathsf{poly}(\lambda_{\mathsf{SSBARG}})$. □

Lemma 5.7. *Suppose* Π_{SSBARG} *satisfies somewhere-soundness and* $\lambda_{\mathsf{SSBARG}}(\lambda, s) = (\lambda + s)^c$ *for a constant* $c > 1$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\Pr\left[\mathsf{Hyb}_2(\mathcal{A}) = 1\right] = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists a polynomial $q = q(\lambda)$, and an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$, we have that $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq 1/q(\lambda)$. Let $\lambda_{\mathcal{A}} \in \mathbb{N}$ be the constant beyond which the polynomial $q(\lambda)$ is monotone (i.e., for all $\lambda, \lambda' \geq \lambda_{\mathcal{A}}$, if holds that $q(\lambda) \geq q(\lambda')$ if and only if $\lambda \geq \lambda'$). Define now the infinite set $\Lambda_{\mathcal{B}} \subseteq \mathbb{N}$

$$\Lambda_{\mathcal{B}} = \{\lambda_{\mathsf{SSBARG}}(\lambda, s(\lambda)) : \lambda \in \Lambda_{\mathcal{A}} \wedge \lambda \geq \lambda_{\mathcal{A}}\}.$$

We use $\mathcal{A}$ to construct an efficient (non-uniform) adversary $\mathcal{B}$ for the somewhere-soundness game for Π_{SSBARG}. The advice string associated with $\lambda_{\mathsf{SSBARG}} \in \Lambda_{\mathcal{B}}$ is some parameter $\lambda \in \Lambda_{\mathcal{A}}$ and $\lambda \geq \lambda_{\mathcal{A}}$ where $\lambda_{\mathsf{SSBARG}} = (\lambda + s(\lambda))^c$:

1. On input the security parameter $1^{\lambda_{\mathsf{SSBARG}}}$ (and advice string 1^λ), algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on input 1^λ and receives the bound on the number of instances 1^T and the bound on the circuit size 1^s.
2. Algorithm $\mathcal{B}$ samples $i^* \xleftarrow{\text{R}} [T]$ and gives $1^T, 1^s, i^*$ to the challenger. The challenger replies with crs. Algorithm $\mathcal{B}$ gives 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, statements $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$, and a proof π. Algorithm $\mathcal{B}$ then outputs $C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi$.

The challenger constructs the CRS as $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{SSBARG}}}, 1^T, 1^s, i^*)$, so the view of $\mathcal{A}$ is distributed exactly as in Hyb_2. The somewhere-soundness game outputs 1 if $\mathsf{Verify}(C, (\mathbf{x}_1, \ldots, \mathbf{x}_t), \pi) = 1$, $i^* \leq t$, and $(C, \mathbf{x}_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$, which occurs with probability

$$\Pr\left[\mathsf{Hyb}_2(\mathcal{A}) = 1\right] \geq 1/q(\lambda) \geq 1/q(\lambda_{\mathsf{SSBARG}})$$

since $\lambda_{\mathsf{SSBARG}} \geq \lambda \geq \lambda_{\mathcal{A}}$ so $q(\lambda_{\mathsf{SSBARG}}) \geq q(\lambda)$. Thus, algorithm $\mathcal{B}$ breaks somewhere-soundness with advantage at least $1/q(\lambda_{\mathsf{SSBARG}})$ for all $\lambda_{\mathsf{SSBARG}} \in \Lambda_{\mathcal{B}}$, which completes the proof. □

Returning to the proof of Theorem 5.4, we conclude via Lemmas 5.5 and 5.6 that $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq 1/T \cdot \Pr\left[\mathsf{Hyb}_0(\mathcal{A}) = 1\right] - \mathsf{negl}(\lambda)$. By Lemma 5.7, this means $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq T \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$ since $T = \mathsf{poly}(\lambda)$. Since Hyb_0 corresponds to the real (adaptive) soundness experiment, the claim holds. □

Acknowledgments

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*, 2023.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, 1988.

[BHK11] Mark Braverman, Avinatan Hassidim, and Yael Tauman Kalai. Leaky pseudo-entropy functions. In *ITCS*, 2011.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT*, 2009.

[BKP$^+$23a] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. *IACR Cryptol. ePrint Arch.*, 2023. https://eprint.iacr.org/archive/2023/754/20230626:185215.

[BKP$^+$23b] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. *IACR Cryptol. ePrint Arch.*, 2023. https://eprint.iacr.org/archive/2023/754/20231204:075616.

[BKP$^+$23c] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. *IACR Cryptol. ePrint Arch.*, 2023. https://eprint.iacr.org/archive/2023/754/20230525:044715.

[BY92] Mihir Bellare and Moti Yung. Certifying cryptographic tools: The case of trapdoor permutations. In *CRYPTO*, 1992.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In *CRYPTO*, 2023.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, 2003.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO*, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, 2021.

[CL18] Ran Canetti and Amit Lichtenberg. Certifying trapdoor permutations, revisited. In *TCC*, 2018.

[CW23] Jeffrey Champion and David J. Wu. Non-interactive zero-knowledge from non-interactive batch arguments. In *CRYPTO*, 2023.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *FOCS*, 2022.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO*, 2002.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, 2004.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, 1990.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GR13] Oded Goldreich and Ron D. Rothblum. Enhancements of trapdoor permutations. *J. Cryptol.*, 26(3), 2013.

[GSWW22] Rachit Garg, Kristin Sheridan, Brent Waters, and David J. Wu. Fully succinct batch arguments for np from indistinguishability obfuscation. In *TCC*, 2022.

[GUV07] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *CCC*, 2007.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, 2011.

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In *EUROCRYPT*, 2022.

[HLWW13] Carmit Hazay, Adriana López-Alt, Hoeteck Wee, and Daniel Wichs. Leakage-resilient cryptography from minimal assumptions. In *EUROCRYPT*, 2013.

[KLVW23] Yael Tauman Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, 2023.

[KMY20] Fuyuki Kitagawa, Takahiro Matsuda, and Takashi Yamakawa. NIZK from SNARG. In *TCC*, 2020.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In *TCC*, 2021.

[KW19] Venkata Koppula and Brent Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In *CRYPTO*, 2019.

[LPWW20] Benoît Libert, Alain Passelègue, Hoeteck Wee, and David J. Wu. New constructions of statistical NIZKs: Dual-mode DV-NIZKs and more. In *EUROCRYPT*, 2020.

[LQR$^+$19] Alex Lombardi, Willy Quach, Ron D. Rothblum, Daniel Wichs, and David J. Wu. New constructions of reusable designated-verifier NIZKs. In *CRYPTO*, 2019.

[Mat23] Takahiro Matsuda. Chosen ciphertext security via BARGs. *IACR Cryptol. ePrint Arch.*, 2023.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, 1989.

[NWW24] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from bargs and additively homomorphic encryption. In *TCC*, 2024.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, 1990.

[QRW19] Willy Quach, Ron D. Rothblum, and Daniel Wichs. Reusable designated-verifier NIZKs for all NP from CDH. In *EUROCRYPT*, 2019.

[QWW21] Willy Quach, Brent Waters, and Daniel Wichs. Targeted lossy functions and applications. In *CRYPTO*, 2021.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.

[Wat24] Brent Waters. A new approach for non-interactive zero-knowledge from learning with errors. In *STOC*, pages 399–410, 2024.

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

[WW24a] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In *STOC*, pages 387–398, 2024.

[WW24b] Brent Waters and David J. Wu. A pure indistinguishability obfuscation approach to adaptively-sound snargs for NP. *IACR Cryptol. ePrint Arch.*, page 933, 2024.

[WWW24] Brent Waters, Hoeteck Wee, and David J. Wu. New techniques for preimage sampling: Improved NIZKs and more from LWE. Cryptology ePrint Archive, Paper 2023/1938, 2024. `https://eprint.iacr.org/2023/1938`.

[WZ24] Brent Waters and Mark Zhandry. Adaptive security in SNARGs via iO and lossy functions. In *CRYPTO*, pages 72–104, 2024.

A Leakage-Resilient Weak PRFs from OWFs

In this section, we describe the leakage-resilient weak pseudorandom function (Theorem 2.3) based on one-way functions implicit in the leakage-resilient symmetric encryption scheme from [QWW21, §5.4]. We include the construction and analysis here for completeness. We start by recalling a few standard notions on min-entropy, pseudoentropy functions, and randomness extractors.

Min-entropy. We start with some basic definitions on min-entropy adapted from those in [DRS04]. For a discrete random variable X, we write $\mathbf{H}_\infty(X) = -\log(\max_x \Pr[X = x])$ to denote its min-entropy. For two discrete random variables X, Y, we define the conditional min-entropy of X given Y to be

$$\mathbf{H}_\infty(X \mid Y) = -\log(\mathbb{E}_{y \leftarrow Y}[\max_x \Pr[X = x \mid Y = y]]).$$

The following fact about conditional min-entropy will also be useful in our analysis:

Lemma A.1 (Conditional Min-Entropy [DRS04, Lemma 2.2])**.** *Let X, Y be random variables and suppose there are at most 2^λ elements in the support of Y. Then $\mathbf{H}_\infty(X \mid Y) \geq \mathbf{H}_\infty(X, Y) - \lambda \geq \mathbf{H}_\infty(X) - \lambda$.*

Pseudoentropy functions. Next, we recall the notion of a pseudoentropy function (PEF) [BHK11]. Our definition is adapted from the definition in [QWW21]. We consider an adaptation where the lossiness is provided as a parameter to the key-generation algorithm (as opposed to fixed as part of the scheme description).

Definition A.2 (Pseudoentropy Function [BHK11, QWW21, adapted])**.** A pseudoentropy function with input length $n = n(\lambda)$ is a triple of efficient algorithms $\Pi_{\mathsf{PEF}} = (\mathsf{Gen}, \mathsf{LossyGen}, \mathsf{Eval})$ with the following properties:

- $\mathsf{Gen}(1^\lambda, 1^\ell) \to k$: On input the security parameter λ and the lossiness parameter ℓ, the generation algorithm outputs a key k. The key (implicitly) determines the output length m of the pseudoentropy function.
- $\mathsf{LossyGen}(1^\lambda, 1^\ell, x^*) \to k$: On input the security parameter λ, the lossiness parameter ℓ, and the input $x^* \in \{0,1\}^n$, the lossy-generation algorithm outputs a (lossy) key k. The key (implicitly) determines the output length m of the pseudoentropy function.

- $\mathsf{Eval}(k, x) \to y$: On input a key k and an input $x \in \{0,1\}^n$, the evaluation algorithm deterministically outputs a value $y \in \{0,1\}^m$.

We require Π_{PEF} to satisfy the following properties:

- **Lossy at x^*:** For all $x^* \in \{0,1\}^n$,

$$\mathbf{H}_\infty\left(\mathsf{Eval}(k, x^*) \mid \{(x, \mathsf{Eval}(k, x))\}_{x \neq x^*}\right) \geq \ell,$$

  where the min-entropy is taken over the distribution of $k \leftarrow \mathsf{LossyGen}(1^\lambda, 1^\ell, x^*)$.

- **Mode indistinguishability:** For a security parameter $\lambda \in \mathbb{N}$ and a bit $\beta \in \{0,1\}$, we define the mode indistinguishability game between an adversary $\mathcal{A}$ and a challenger as follows:

  1. The challenger gives 1^λ to algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a lossiness parameter 1^ℓ and a value $x^* \in \{0,1\}^n$.
  2. If $\beta = 0$, the challenger samples $k \leftarrow \mathsf{Gen}(1^\lambda, 1^\ell)$. If $\beta = 1$, the challenger samples $k \leftarrow \mathsf{LossyGen}(1^\lambda, 1^\ell, x^*)$. The challenger gives k to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

  The pseudoentropy function satisfies mode indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

Theorem A.3 (Pseudoentropy Functions [QWW21, Theorem 5.2]). *Assuming the existence of one-way functions, for every polynomial input length $n = n(\lambda)$, there exists a secure pseudoentropy function $\Pi_{\mathsf{PEF}} = (\mathsf{Gen}, \mathsf{LossyGen}, \mathsf{Eval})$ where $\mathsf{Gen}(1^\lambda, 1^\ell)$ outputs a key k of length $\ell\lambda$ and $\mathsf{Eval}(k, \cdot)$ outputs a value of length ℓ.*

Randomness extractors. The leakage-resilient weak PRF construction from [QWW21] also requires a strong seeded extractor, which we recall below.

Definition A.4 (Strong Seeded Extractor). We say that a random variable X is an (n, k)-source if X takes on values in $\{0,1\}^n$ and $\mathbf{H}_\infty(X) \geq k$. Then, an efficient algorithm $\mathsf{Ext} \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a strong (k, ε)-extractor if for every (n, k)-source X, the statistical distance between the following distributions is at most ε:

$$\left\{(y, \mathsf{Ext}(x, y)) : x \leftarrow X, y \xleftarrow{\mathrm{R}} \{0,1\}^d\right\} \quad \text{and} \quad \left\{(y, z) : y \xleftarrow{\mathrm{R}} \{0,1\}^d, z \xleftarrow{\mathrm{R}} \{0,1\}^m\right\}.$$

Theorem A.5 (Strong Seeded Extractor [GUV07, Theorem 1.5]). *For every constant $\alpha > 0$, all positive integers $n, k > 0$, and all $\varepsilon > 0$, there is an explicit construction of a (k, ε)-extractor with input length n, output length $m \geq (1 - \alpha)k$, and seed length $d = O(\log n + \log(1/\varepsilon))$.*

Leakage-resilient weak PRF. We now recall the construction and analysis of the leakage-resilient weak PRF scheme obtained by composing a pseudoentropy function with a randomness extractor. This scheme is implicit in the work of [QWW21].

Construction A.6 (Leakage-Resilient Weak PRF). Let λ be a security parameter and ℓ be a leakage parameter. Let $\Pi_{\mathsf{PEF}} = (\mathsf{PEF.Gen}, \mathsf{PEF.LossyGen}, \mathsf{PEF.Eval})$ be a pseudoentropy function with input length $n = n(\lambda)$ and output length $m = m(\lambda, \ell)$. Let $\mathsf{Ext}_{\lambda, \ell}$ be the explicit strong seeded $(\lambda, 2^{-\lambda})$-extractor with input length m, seed length $d = d(\lambda, \ell)$, and output length 1 from Theorem A.5.[7] We construct a leakage-resilient weak pseudorandom function $\Pi_{\mathsf{LRwPRF}} = (\mathsf{Setup}, \mathsf{Eval})$ with domain $\{0,1\}^{n+d}$ and range $\{0,1\}$ as follows:

[7]Without loss of generality, we can always truncate the output to a single bit.

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: On input the security parameter λ and the leakage parameter ℓ, let $\ell_{\mathsf{PEF}} = \ell + \lambda$. Then, sample $k \leftarrow \mathsf{PEF.Gen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}})$ and output the key k. We assume that the parameters λ and ℓ are implicitly associated with k.
- $\mathsf{Eval}(k, x)$: On input a key k and an input $x = x_1 \| x_2$ where $x_1 \in \{0,1\}^n$ and $x_2 \in \{0,1\}^d$, where $d = d(\lambda, \ell)$, output $\mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_1), x_2)$.

Theorem A.7. *If Π_{PEF} is a pseudoentropy function, then Construction A.6 is a secure leakage-resilient weak PRF.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the leakage-resilient weak PRF security game. As we show in Appendix A.1 (Theorem A.11), we can assume without loss of generality that algorithm $\mathcal{A}$ always makes a *single* challenge query. We begin by defining a sequence of hybrid experiments between the challenger and $\mathcal{A}$:

- Hyb_0: This is the real leakage-resilient weak PRF security game with bit $\beta = 0$. As noted previously, we assume the adversary $\mathcal{A}$ makes exactly one challenge query. Specifically, the game proceeds as follows:
  - **Pre-challenge evaluation queries:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s. The challenger responds by sampling $k \leftarrow \mathsf{PEF.Gen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}})$ where $\ell_{\mathsf{PEF}} = \ell + \lambda$. It then samples inputs $x_1, \ldots, x_s \xleftarrow{R} \{0,1\}^{n+d}$. For each $i \in [s]$, the challenger parses $x_i = x_{i,1} \| x_{i,2}$ where $x_{i,1} \in \{0,1\}^n$ and $x_{i,2} \in \{0,1\}^d$. The challenger replies to $\mathcal{A}$ with the set of evaluations $\{(x_i, \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}))\}_{i \in [s]}$.
  - **Leakage query:** Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak}\colon \{0,1\}^\kappa \to \{0,1\}^\ell$, where κ is the length of the key k. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.
  - **Challenge query:** The challenger samples $x^* \xleftarrow{R} \{0,1\}^{n+d}$ and parses $x^* = x_1^* \| x_2^*$ where $x_1^* \in \{0,1\}^n$ and $x_2^* \in \{0,1\}^d$. The challenger computes $y^* \leftarrow \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_1^*), x_2^*)$ and gives (x^*, y^*) to $\mathcal{A}$.
  - **Output:** 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 the PEF key to be lossy at the challenge point $x^\star$. Specifically, the game proceeds as follows:
  - **Pre-challenge evaluation queries:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s. The challenger then samples $x^* \xleftarrow{R} \{0,1\}^{n+d}$ and $k \leftarrow \mathsf{PEF.LossyGen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}}, x^*)$. It then samples inputs $x_1, \ldots, x_s \xleftarrow{R} \{0,1\}^{n+d}$. For each $i \in [s]$, the challenger parses $x_i = x_{i,1} \| x_{i,2}$ where $x_{i,1} \in \{0,1\}^n$ and $x_{i,2} \in \{0,1\}^d$. The challenger replies to $\mathcal{A}$ with the set of evaluations $\{(x_i, \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}))\}_{i \in [s]}$.
  - **Leakage query:** Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak}\colon \{0,1\}^\kappa \to \{0,1\}^\ell$, where κ is the length of the key k. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.
  - **Challenge query:** The challenger parses $x^* = x_1^* \| x_2^*$ where $x_1^* \in \{0,1\}^n$ and $x_2^* \in \{0,1\}^d$. The challenger computes $y^* \leftarrow \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_1^*), x_2^*)$ and gives (x^*, y^*) to $\mathcal{A}$.
  - **Output:** Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_2: Same as Hyb_1, except the challenger samples $y^* \xleftarrow{R} \{0,1\}$ when responding to the challenge query. Specifically, the game proceeds as follows:
  - **Pre-challenge evaluation queries:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s. The challenger then samples $x^* \xleftarrow{R} \{0,1\}^{n+d}$ and $k \leftarrow \mathsf{PEF.LossyGen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}}, x^*)$. It then samples inputs $x_1, \ldots, x_s \xleftarrow{R} \{0,1\}^{n+d}$. For each $i \in [s]$, the challenger parses $x_i = x_{i,1} \| x_{i,2}$ where $x_{i,1} \in \{0,1\}^n$ and $x_{i,2} \in \{0,1\}^d$. The challenger replies to $\mathcal{A}$ with the set of evaluations $\{(x_i, \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}))\}_{i \in [s]}$.
  - **Leakage query:** Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak}\colon \{0,1\}^\kappa \to \{0,1\}^\ell$, where κ is the length of the key k. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.
  - **Challenge query:** The challenger samples $y^* \xleftarrow{R} \{0,1\}$ and gives (x^*, y^*) to $\mathcal{A}$.

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

- Hyb_3: This is Hyb_2, except the challenger samples the PEF key using Gen. This is the leakage-resilient weak PRF security game with bit $\beta = 1$. Specifically, the game proceeds as follows:
    - **Pre-challenge evaluation queries:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s. The challenger responds by sampling $k \leftarrow \mathsf{PEF.Gen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}})$ where $\ell_{\mathsf{PEF}} = \ell + \lambda$. It then samples inputs $x_1, \dots, x_s \xleftarrow{R} \{0,1\}^{n+d}$. For each $i \in [s]$, the challenger parses $x_i = x_{i,1} \| x_{i,2}$ where $x_{i,1} \in \{0,1\}^n$ and $x_{i,2} \in \{0,1\}^d$. The challenger replies to $\mathcal{A}$ with the set of evaluations $\{(x_i, \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}))\}_{i \in [s]}$.
    - **Leakage query:** Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak} \colon \{0,1\}^\kappa \to \{0,1\}^\ell$, where κ is the length of the key k. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.
    - **Challenge query:** The challenger samples $x^* \xleftarrow{R} \{0,1\}^{n+d}$ and $y^* \xleftarrow{R} \{0,1\}$ and gives (x^*, y^*) to $\mathcal{A}$.
    - **Output:** Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

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

Claim A.8. *If Π_{PEF} satisfies mode indistinguishability, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose that $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks mode indistinguishability of Π_{PEF} with the same advantage ε.

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running $\mathcal{A}$ with the same security parameter 1^λ. Algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s.

2. Algorithm $\mathcal{B}$ sets $\ell_{\mathsf{PEF}} = \ell + \lambda$ and samples $x^* \xleftarrow{R} \{0,1\}^{n+d}$. It sends $1^{\ell_{\mathsf{PEF}}}$ and x^* to the challenger. The challenger replies with a PEF key k.

3. Algorithm $\mathcal{B}$ samples inputs $x_1, \dots, x_s \xleftarrow{R} \{0,1\}^{n+d}$. For each $i \in [s]$, algorithm $\mathcal{B}$ parses $x_i = x_{i,1} \| x_{i,2}$ where $x_{i,1} \in \{0,1\}^n$ and $x_{i,2} \in \{0,1\}^d$. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with
$$\{(x_i, \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}))\}_{i \in [s]}.$$

4. Algorithm $\mathcal{A}$ outputs a Boolean circuit $\mathsf{leak} \colon \{0,1\}^\kappa \to \{0,1\}^\ell$. Algorithm $\mathcal{B}$ responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.

5. To simulate the challenge query, algorithm $\mathcal{B}$ parses $x^* = x_1^* \| x_2^*$ where $x_1^* \in \{0,1\}^n$ and $x_2^* \in \{0,1\}^d$. Algorithm $\mathcal{B}$ computes $y^* \leftarrow \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_1^*), x_2^*)$ and gives (x^*, y^*) to $\mathcal{A}$.

6. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ outputs to the challenger.

Let β be the bit in the mode indistinguishability game. We consider the two cases:

- Suppose $\beta = 0$. In this case, the challenger samples $k \leftarrow \mathsf{PEF.Gen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}})$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_0 for $\mathcal{A}$.

- Suppose $\beta = 1$. In this case, the challenger samples $k \leftarrow \mathsf{PEF.LossyGen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}}, x^*)$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1 for $\mathcal{A}$.

We conclude that algorithm $\mathcal{B}$ wins the mode indistinguishability game with the same advantage ε, as required. □

Claim A.9. *Suppose $n \geq \lambda$. If Π_{PEF} is lossy at x^*, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, *then* $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Consider an execution of Hyb_1 and Hyb_2. Let $x_1, \ldots, x_s \in \{0,1\}^{n+d}$ be the values sampled by the challenger in the pre-challenge phase. Let $x^* \in \{0,1\}^{n+d}$ be the challenge point. Parse $x_i = x_{i,1} \| x_{i,2}$ and $x^* = x_1^* \| x_2^*$ as described in Hyb_1 and Hyb_2. Since $x^* \xleftarrow{\mathrm{R}} \{0,1\}^{n+d}$ (and is sampled independently of $x_1, \ldots, x_s$), we have that

$$\Pr[\exists i \in [s] : x_1^* = x_{i,1}] \le \frac{s}{2^n} \le \frac{s}{2^\lambda} = \mathsf{negl}(\lambda),$$

since $s = \mathsf{poly}(\lambda)$. Note that $s = \mathsf{poly}(\lambda)$ since $\mathcal{A}$ is an efficient algorithm. Now, in the following, we will assume that $x_1^* \neq x_{i,1}$ for all $i \in [s]$. Consider now the distribution of y^* in Hyb_1 and Hyb_2. We argue now that the distribution of y^* in the two experiments is statistically indistinguishable (even given the other quantities):

- Since the PEF is lossy at x^* and the PEF key is sampled as $k \leftarrow \mathsf{PEF.LossyGen}(1^\lambda, 1^{\ell_{\mathsf{PEF}}}, x^*)$ and $x_1^* \neq x_{i,1}$ for all $i \in [s]$, it follows that

$$\mathbf{H}_\infty\big(\mathsf{PEF.Eval}(k, x_1^*) \mid \big\{(x_{i,1}, \mathsf{PEF.Eval}(k, x_{i,1}))\big\}_{i \in [s]}\big) \ge \ell_{\mathsf{PEF}}.$$

- Next, the values of $x_{i,2}$ are sampled independently of k and x^* for all $i \in [s]$. This means we can write

$$\mathbf{H}_\infty\Big(\mathsf{PEF.Eval}(k, x_1^*) \mid \big\{\big(x_i, \mathsf{Ext}_{\lambda,\ell}\big(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}\big)\big)\big\}_{i \in [s]}\Big) \ge \ell_{\mathsf{PEF}}.$$

- Since $\mathsf{leak}(k) \in \{0,1\}^\ell$, we can appeal to Lemma A.1 to conclude that

$$\mathbf{H}_\infty\Big(\mathsf{PEF.Eval}(k, x_1^*) \mid \big\{\big(x_i, \mathsf{Ext}_{\lambda,\ell}\big(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}\big)\big)\big\}_{i \in [s]}, \mathsf{leak}(k)\Big) \ge \ell_{\mathsf{PEF}} - \ell = \lambda,$$

  since $\ell_{\mathsf{PEF}} = \ell + \lambda$.

- Since $\mathsf{Ext}_{\lambda,\ell}$ is a strong $(\lambda, 2^{-\lambda})$-extractor and $x_2^* \xleftarrow{\mathrm{R}} \{0,1\}^d$, it follows that the distribution of the challenge bit $y^* := \mathsf{Ext}_{\lambda,\ell}(\mathsf{PEF.Eval}(k, x_1^*), x_2^*)$ in Hyb_1 is $2^{-\lambda}$-close to uniform over $\{0,1\}$ even conditioned on x_2^*, $\big\{\big(x_i, \mathsf{Ext}_{\lambda,\ell}\big(\mathsf{PEF.Eval}(k, x_{i,1}), x_{i,2}\big)\big)\big\}_{i \in [s]}$ and $\mathsf{leak}(k)$. This precisely coincides with the distribution of y^* in Hyb_2, and the claim follows. □

Claim A.10. *If Π_{PEF} satisfies mode indistinguishability, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. This follows by an analogous argument as the proof of Claim A.8, except the reduction algorithm samples $y^* \xleftarrow{\mathrm{R}} \{0,1\}$. □

Combining Claims A.8 to A.10, the claim follows. □

Proof of Theorem 2.3. Combining Construction A.6 with Theorem A.3, we obtain Theorem 2.3. Specifically, we instantiate the underlying building blocks as follows:

- Let $\Pi_{\mathsf{PEF}} = (\mathsf{PEF.Gen}, \mathsf{PEF.LossyGen}, \mathsf{PEF.Eval})$ be the PEF from Theorem A.3 with input length λ.

- Consider an instantiation of Construction A.6 with Π_{PEF}. In this case, the setup algorithm $\mathsf{Setup}(1^\lambda, 1^\ell)$ samples the PEF key k by running $k \leftarrow \mathsf{PEF.Gen}(1^\lambda, 1^{\lambda+\ell})$.

- From Theorem A.3, the length of the weak PRF key k is then $(\lambda + \ell) \cdot \lambda$ and the output length of the PEF is $\ell_{\mathsf{PEF}} = \lambda + \ell$. Then, the seed length of the extractor $\mathsf{Ext}_{\lambda,\ell}$ from Construction A.6 (see also Theorem A.5) is $d(\lambda, \ell) = O(\lambda + \log \ell_{\mathsf{PEF}}) = O(\lambda + \log \ell)$.

Thus, Construction A.6 can be instantiated with a domain of bit-length $\rho \ge \lambda + O(\lambda + \log \ell) = O(\lambda + \log \ell)$, and Theorem 2.3 follows. □

A.1 Single-Challenge Security to Multi-Challenge Security

In this section, we show via a simple hybrid argument that for a leakage-resilient weak PRF, security against a single challenge query implies security against an arbitrary polynomial number of challenge queries (i.e., as defined in Definition 2.2).

Theorem A.11. *Suppose* $\Pi_{\mathsf{LRwPRF}} = (\mathsf{Setup}, \mathsf{Eval})$ *is a leakage-resilient weak PRF. If* Π_{LRwPRF} *is secure against an adversary that makes at most one evaluation query, then it is also secure under Definition 2.2 (where the adversary can make any polynomial number of evaluation queries).*

Proof. Let $\mathcal{A}$ be an efficient adversary for the single-challenge leakage-resilient weak PRF security game. Since $\mathcal{A}$ is efficient, we can bound the number of challenge queries it makes by some polynomial $Q = Q(\lambda)$. For each $j \in [Q+1]$, we now define a hybrid experiment Hyb_j between a challenger and $\mathcal{A}$ as follows:

1. **Pre-challenge evaluation queries:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s it would like to make. The challenger samples a key $k \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. and inputs $x_1, \ldots, x_s \xleftarrow{\mathrm{R}} \mathcal{X}$. The challenger replies to $\mathcal{A}$ with $\{(x_i, \mathsf{Eval}(k, x_i))\}_{i \in [s]}$.
2. **Leakage query:** Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak}\colon \mathcal{K} \to \{0,1\}^\ell$, where $\mathcal{K}$ is the key-space of Π_{LRwPRF}. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{A}$.
3. **Challenge queries:** Algorithm $\mathcal{A}$ then outputs the number of challenge queries 1^t it would like to make. Then, for each $i \in [t]$, the challenger does the following:
   - If $i \geq j$, the challenger samples $x'_i \xleftarrow{\mathrm{R}} \mathcal{X}$, $y_i \leftarrow \mathsf{Eval}(k, x'_i)$.
   - If $i < j$, the challenger samples $x'_i \xleftarrow{\mathrm{R}} \mathcal{X}$, $y_i \xleftarrow{\mathrm{R}} \mathcal{Y}$.

   The challenger gives $\{(x'_i, y_i)\}_{i \in [t]}$ to $\mathcal{A}$.
4. **Output:** Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

Claim A.12. *Suppose* Π_{LRwPRF} *is secure against efficient adversaries that makes at most one challenge query. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and all* $j \in [Q]$,

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

Proof. Suppose $|\Pr[\mathsf{Hyb}^{(j+1)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(j)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that makes at most 1 challenge query in the leakage-resilient weak PRF security game.

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running $\mathcal{A}$ with the same security parameter 1^λ. Algorithm $\mathcal{A}$ starts by outputting the leakage parameter 1^ℓ and the number of pre-challenge queries 1^s it would like to make. Algorithm $\mathcal{B}$ outputs 1^ℓ and 1^{s+Q} to the challenger.
2. Algorithm $\mathcal{B}$ receives $\{(x_i, \mathsf{Eval}(k, x_i))\}_{i \in [s+Q]}$ and forwards $\{(x_i, \mathsf{Eval}(k, x_i))\}_{i \in [s]}$ to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ now outputs a Boolean circuit $\mathsf{leak}\colon \mathcal{K} \to \{0,1\}^\ell$, which algorithm $\mathcal{B}$ forwards to its challenger. The challenger responds with $\mathsf{leak}(k)$ to $\mathcal{B}$, which algorithm $\mathcal{B}$ gives $\mathcal{A}$.
4. In the challenge phase, algorithm $\mathcal{A}$ outputs 1^t where $t \leq Q$ is the number of challenge queries algorithm $\mathcal{A}$ seeks to make. Algorithm $\mathcal{B}$ also receives a singleton challenge $\{(x^*, y^*)\}$ from its challenger. For each $i \in [t]$, algorithm $\mathcal{B}$ does the following:
   - If $i > j$, algorithm $\mathcal{B}$ sets $x'_i \leftarrow x_{s+i}$, $y_i \leftarrow \mathsf{Eval}(k, x_{s+i})$.
   - If $i = j$, algorithm $\mathcal{B}$ sets $x'_i \xleftarrow{\mathrm{R}} x^*$, $y_i \leftarrow y^*$.
   - If $i < j$, algorithm $\mathcal{B}$ samples $x'_i \xleftarrow{\mathrm{R}} \mathcal{X}$, $y_i \xleftarrow{\mathrm{R}} \mathcal{Y}$.

   The challenger gives $\{(x'_i, y_i)\}_{i \in [t]}$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

By construction, the pre-challenge evaluation queries seen by $\mathcal{A}$ are answered exactly according to the specification of the real leakage-resilient weak PRF security game. Similarly, the leakage query $\mathsf{leak}(k)$ is distributed identically as in real security game. Finally, the values (x_i', y_i) for $i \neq j$ are distributed exactly as in Hyb_j and Hyb_{j+1}. Let β be the bit in the leakage-resilient weak PRF security game. We consider the two cases:

- Suppose $\beta = 0$. In this case, the challenger samples $x^* \xleftarrow{\text{R}} \mathcal{X}$ and $y^* \leftarrow \mathsf{Eval}(k, x^*)$. The distribution of (x_t', y_t) perfectly matches the distribution in Hyb_j.
- Suppose $\beta = 1$. For In this case, the challenger samples $y^* \xleftarrow{\text{R}} \mathcal{Y}$. This perfectly matches the distribution in Hyb_{j+1}.

Thus, algorithm $\mathcal{B}$ wins the leakage-resilient weak PRF security game with the same advantage ε. □

To complete the proof, observe that $\mathsf{Hyb}_1(\mathcal{A})$ is equivalent to the leakage-resilient weak PRF security game with bit $\beta = 0$, while Hyb_{Q+1} is equivalent to the leakage-resilient weak PRF security game with bit $\beta = 1$. The claim now follows by a hybrid argument. □