Batch Arguments for NP and More from Standard Bilinear Group Assumptions

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

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

Abstract

Non-interactive batch arguments for NP provide a way to amortize the cost of NP verification across multiple instances. They enable a prover to convince a verifier of multiple NP statements with communication much smaller than the total witness length and verification time much smaller than individually checking each instance.

In this work, we give the first construction of a non-interactive batch argument for NP from standard assumptions on groups with bilinear maps (specifically, from either the subgroup decision assumption in composite-order groups or from the k-Lin assumption in prime-order groups for any $k \geq 1$). Previously, batch arguments for NP were only known from LWE, or a combination of multiple assumptions, or from non-standard/non-falsifiable assumptions. Moreover, our work introduces a new *direct* approach for batch verification and avoids heavy tools like correlation-intractable hash functions or probabilistically-checkable proofs common to previous approaches.

As corollaries to our main construction, we obtain the first publicly-verifiable non-interactive delegation scheme for RAM programs (i.e., a succinct non-interactive argument (SNARG) for P) with a CRS of sublinear size (in the running time of the RAM program), as well as the first aggregate signature scheme (supporting bounded aggregation) from standard assumptions on bilinear maps.

1 Introduction

Consider the following scenario: a prover has a batch of m NP statements $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and seeks to convince the verifier that all of these statements are true (i.e., convince the verifier that $\mathbf{x}_i \in \mathcal{L}$ for all $i \in [m]$, where $\mathcal{L}$ is the associated NP language). A naïve solution is for the prover to provide the m witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m$ to the verifier and have the verifier check the NP relation on each pair $(\mathbf{x}_i, \mathbf{w}_i)$. A natural question is whether we could do this more efficiently. Namely, can the prover convince the verifier that $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{L}$ with a proof of size $o(m)$—that is, can the size of the proof grow *sublinearly* with the number of instances?

Batch arguments. The focus of this work is on constructing non-interactive *batch arguments* (BARGs) for NP languages in the common reference string (CRS) model. In this model, a (trusted) setup algorithm samples a common reference string crs that is used to construct and verify proofs. The goal of a BARG is to amortize the cost of NP verification across multiple instances. Specifically, a BARG for NP allows a prover to construct a proof π of m NP statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ where the size of the proof π scales sublinearly with m. We focus on the setting where the proof is *non-interactive* and *publicly verifiable*. The soundness requirement is that no *computationally-bounded* prover can convince the verifier of a tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ that contains a false instance $\mathbf{x}_i \notin \mathcal{L}$; namely, we focus on batch *argument* systems.

Constructing non-interactive batch arguments for NP is challenging, and until very recently, constructions have either relied on idealized models [Mic95, Gro16, BBHR18, COS20, CHM+20, Set20] or on non-standard [KPY19], and oftentimes, non-falsifiable cryptographic assumptions [Gro10, BCCT12, DFH12, Lip13, PHGR13, GGPR13, BCI+13, BCPR14, BISW17, BCC+17] (see also Section 1.3 for more detail). This state of affairs changed in two very recent and exciting works by Choudhuri et al. In the first work [CJJ21a], they show how to construct a BARG assuming both subexponential hardness of DDH in pairing-free groups and polynomial hardness of QR. Subsequently, they

construct a BARG from polynomial hardness of LWE [CJJ21b]. Both works leverage correlation-intractable hash functions [CGH98, CCH+19, PS19, JJ21] to *provably* instantiate the Fiat-Shamir heuristic [FS86].

In this work, we take a *direct* approach for constructing BARGs from bilinear maps, and provide a new instantiation from either polynomial hardness of the k-Lin assumption on prime-order bilinear groups, or from polynomial hardness of the subgroup decision assumption on composite-order bilinear groups. This is the first BARG for NP under standard assumptions over bilinear groups. Moreover, our construction is direct and avoids powerful tools like correlation-intractable hash functions or probabilistically-checkable proofs used in many previous constructions.

Delegation for RAM programs. A closely related problem is delegation for RAM programs (also known as a succinct non-interactive argument (SNARG) for the class P of polynomial-time deterministic computations). In a delegation scheme for RAM programs, the prover has a RAM program $\mathcal{P}$, an input x, and output y, and its goal is to convince the verifier that $y = \mathcal{P}(x)$. The efficiency requirement is that the length of the proof and the verification time should be sublinear (ideally, polylogarithmic) in the running time of the RAM program. There is a close connection between batch arguments for NP and delegation schemes for RAM programs [BHK17, KPY19, KVZ21, CJJ21b], and several of these works show how to construct a delegation scheme for RAM programs using a batch argument for NP. As a corollary to our main construction, we use our BARG to obtain a non-interactive delegation scheme for RAM programs under the SXDH assumption in asymmetric bilinear groups. The CRS size of our construction is short (i.e., sublinear in the running time of the RAM computation).

Previously, Kalai et al. [KPY19] constructed a delegation scheme for RAM programs with a short CRS from a non-standard, but falsifiable, q-type assumption on bilinear groups, and more recently, González and Zacharakis [GZ21] showed how to construct a delegation scheme with a *long* CRS for arithmetic circuits from a *bilateral* k-Lin assumption in asymmetric bilinear groups.[1] Choudhuri et al. [CJJ21b] showed how to construct a delegation scheme for RAM programs from LWE, and previously, Jawale et al. [JKKZ21] constructed a delegation scheme for bounded-depth circuits also from LWE; both of these schemes also have a short CRS. Recently, Hulett et al. [HJKS22] showed how to construct a SNARG for P from sub-exponential DDH (in *pairing-free* groups) in conjunction with the QR assumption. In the designated-verifier model where a *secret* key is needed to check proofs, Kalai et al. [BHK17] showed how to construct a delegation scheme from any computational private information retrieval scheme.

1.1 Our Contributions

In this work, we introduce a simpler and more direct approach for constructing BARGs using bilinear maps. Our main result is a BARG for NP assuming either the polynomial hardness of k-Lin in asymmetric prime-order pairing groups (for any $k \geq 1$)[2], or alternatively, the subgroup decision assumption in composite-order pairing groups. We capture this in the informal theorem statement below:

Theorem 1.1 (Informal)**.** *Take any constant $\varepsilon > 0$. Under the k-Lin assumption (for any $k \geq 1$) in a prime-order pairing group (alternatively, the subgroup decision assumption in a composite-order pairing group), there exists a publicly-verifiable non-interactive BARG for Boolean circuit satisfiability with proof size $\mathsf{poly}(\lambda, |C|)$, verification complexity $\mathsf{poly}(\lambda, m, n) + \mathsf{poly}(\lambda, |C|)$, and CRS size $m^{\varepsilon} \cdot \mathsf{poly}(\lambda)$, where λ is a security parameter, $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is the Boolean circuit, n is the statement size, and m is the number of instances. The BARG satisfies semi-adaptive soundness (Definition 2.5).*

A new approach for batch verification. In contrast to many recent works (see also Section 1.3) on constructing succinct arguments that rely on probabilistically-checkable proofs (PCPs) [KRR13, KRR14, BHK17, CJJ21b, KVZ21] or correlation-intractable hash functions [JKKZ21, CJJ21a, CJJ21b, HJKS22], we take a direct "low-tech" approach in our construction. Our construction follows a "commit-and-prove" strategy and is reminiscent of the classic pairing-based non-interactive proof systems by Groth et al. [GOS06] and Groth and Sahai [GS08]. Essentially, the prover starts by providing a (succinct) commitment to the values associated with each wire in the circuit. The prover commits to m bits for each wire, one for each instance, and we require that the size of the commitment be sublinear in m. Then, for

[1] In the bilateral version of the k-Lin assumption, the challenge is encoded in *both* groups rather than one of the groups.

[2] Recall that the case $k = 1$ corresponds to the DDH assumption holding in each base group (i.e., SXDH). The case $k = 2$ corresponds to the DLIN assumption [BBS04, HK07, Sha07]

each gate in the circuit, the prover provides a short proof that the committed wire values are consistent with the gate operation. The succinct commitment scheme to the wire labels can be viewed as a non-hiding version of the vector commitment scheme of Catalano and Fiore [CF13]. The key challenge in the construction is proving consistency of the gate computations given only the *succinct* commitments to the input and output wires of each gate. We give a technical overview of our approach in Section 1.2 and the formal description in Sections 3 and 4.

Application to delegating RAM programs. The proof size in Theorem 1.1 is *independent* of the number of instances m, but the verification time contains a component $\mathsf{poly}(\lambda, m, n)$ that scales with m. For general NP languages, some type of linear dependence on the number of instances is inherent since the verification algorithm must at least read the input (of size $m \cdot n$). However, when the statements have a "succinct description," (e.g., they are simply the indices $1, \ldots, m$), and it is unnecessary for the verifier to read the full input, we can reduce the the verification cost down to $\mathsf{poly}(\lambda, \log m, |C|)$. This setting is useful for applications to delegation [CJJ21b, KVZ21]. Our main constructions (Theorem 1.1 and Construction 4.5) directly support this setting. Indeed, combining our new pairing-based BARGs with the compiler from Choudhuri et al. [CJJ21b], we also obtain a delegation scheme for RAM programs from the SXDH assumption over pairing groups.

We note here that invoking the compiler from [CJJ21a] additionally requires a "somewhere extractable commitment" scheme (that supports succinct local openings). The pairing-based techniques underlying our BARG construction naturally give rise to a somewhere extractable commitment (in conjunction with a somewhere extractable hash function [HW15, OPWW15]). This is the first construction of a somewhere extractable commitment that supports succinct local openings from standard assumptions over bilinear groups and may be of independent interest. We describe the construction in Section 6. We summarize our result on delegation in the following informal theorem:

Theorem 1.2 (Informal)**.** *Take any constant $\varepsilon > 0$. Under the* SXDH *assumption in a prime-order pairing group, for every polynomial $T = T(\lambda)$, there exists a publicly-verifiable non-interactive delegation scheme for RAM programs with proof size* $\mathsf{poly}(\lambda, \log T)$*, verification complexity* $\mathsf{poly}(\lambda, \log T)$*, a verification key of size* $\mathsf{poly}(\lambda, \log T)$*, and a proving key of size* $T^{\varepsilon} \cdot \mathsf{poly}(\lambda)$*. Here, λ is the security parameter and T is the running time of the RAM program. The delegation scheme is adaptively sound.*

Theorem 1.2 gives the first RAM delegation scheme from standard assumptions over bilinear maps with a CRS whose size is *sublinear* in the running time of the computation. Previously constructions of RAM delegation based on pairings either relied on non-standard q-type assumptions [KPY19] or a CRS of size *super-linear* in the running time of the RAM computation [GZ21].

Application to aggregate signatures. As a final application, we use our BARG for NP to obtain the first aggregate signature scheme that supports bounded aggregation from standard assumptions over bilinear maps. In an aggregate signature scheme, there is a public algorithm that takes a collection of message-signature pairs $(\mu_1, \sigma_1), \ldots, (\mu_m, \sigma_m)$ under (possibly distinct) verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_m$, respectively, and outputs a new signature σ_{agg} on $(\mu_1, \ldots, \mu_m)$ under the joint verification key $(\mathsf{vk}_1, \ldots, \mathsf{vk}_m)$. The requirement is that the size of σ_{agg} scales *sublinearly* with m. A BARG for circuit satisfiability directly yields an aggregate signature scheme via the following straightforward construction. Define the circuit $C(\mathsf{vk}, m, \sigma)$ that takes as input the verification key vk, message μ, and signature σ, and outputs 1 if σ is a valid signature on μ under vk. An aggregate signature on $(\mu_1, \sigma_1, \mathsf{vk}_1), \ldots, (\mu_m, \sigma_m, \mathsf{vk}_m)$ is a BARG proof that $C(\mathsf{vk}_i, \mu_i, \sigma_i) = 1$ for all $i \in [m]$. Succinctness of the BARG ensures that the size of the aggregate signature is sublinear in the number of signatures m. Realizing the above blueprint requires that the underlying BARG satisfy a (weak) form of extractability; the BARGs we construct in this work satisfy this property, and we refer to Section 7 for the details. We obtain the first aggregate signature scheme supporting (bounded) aggregation from standard pairing assumptions. We summarize the instantiation here and compare with previous approaches in Section 1.3:

Corollary 1.3 (Informal)**.** *Under the k-*Lin *assumption (for any $k \geq 1$) in a prime-order pairing group (alternatively, the subgroup decision assumption in a composite-order pairing group), there exists an aggregate signature scheme that supports bounded aggregation. In particular, for any* a priori *bounded polynomial $m = m(\lambda)$, aggregating up to $T \leq m$ message-signature pairs $(\mu_1, \sigma_1), \ldots, (\mu_T, \sigma_T)$ under verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_T$ yields an aggregate signature σ_{agg} of size* $\mathsf{poly}(\lambda)$*.*

1.2 Technical Overview

In this work, we focus on constructing BARGs for the language of Boolean circuit satisfiability. Let $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be a Boolean circuit of size s. A tuple $(C, \mathbf{x}_1, \ldots, \mathbf{x}_m)$ is true if for all $i \in [m]$, there exists a witness $\mathbf{w}_i$ such that $C(\mathbf{x}_i, \mathbf{w}_i) = 1$.

General blueprint. Our BARG for circuit satisfiability follows a "commit-and-prove" paradigm. To construct a proof π of a statement $(C, \mathbf{x}_1, \ldots, \mathbf{x}_m)$ with associated witnesses $(\mathbf{w}_1, \ldots, \mathbf{w}_m)$, the prover proceeds as follows:

- **Wire commitments:** The prover starts by evaluating $C(\mathbf{x}_i, \mathbf{w}_i)$ for each $i \in [m]$. Let t be the number of wires in circuit C. For each instance $i \in [m]$ and wire $k \in [t]$, we write $w_{i,k} \in \{0,1\}$ to denote the value of wire k in instance i. Then $(w_{1,k}, \ldots, w_{m,k}) \in \{0,1\}^m$ is the vector of assignments to wire k across all m instances. The prover starts by constructing a *vector* commitment U_k to each vector $(w_{1,k}, \ldots, w_{m,k})$. Here, we require the commitment to be succinct: namely, $|U_k| = \mathsf{poly}(\lambda, \log m)$, where λ is a security parameter. The prover additionally constructs a proof V_k that U_k is a commitment to a 0/1 vector (i.e., $w_{i,k} \in \{0,1\}$ for all $i \in [m]$).[3] We similarly require that $|V_k| = \mathsf{poly}(\lambda, \log m)$. Both the commitments to the wire assignments $U_1, \ldots, U_k$ and the proofs of valid assignment $V_1, \ldots, V_k$ are included in the BARG proof.
- **Gate satisfiability:** We consider Boolean circuits with fan-in two. Namely, each gate G_ℓ in C can be described by a tuple of $(k_1, k_2, k_3) \in [t]^3$, where k_1, k_2 are the indices for the input wires and k_3 is the index for the output wire. Since NAND gates are universal, we will assume that all of the gates in C are NAND gates.[4] Let s be the number of gates (i.e., the size) of the circuit. For each gate $\ell \in [s]$, the prover constructs a proof W_ℓ that the committed assignments U_{k_3} to the output wire are consistent with the committed assignments U_{k_1}, U_{k_2} to the input wires. For example, if G_ℓ is a NAND gate, U_{k_1} is a commitment to $(w_{1,k_1}, \ldots, w_{m,k_1})$, U_{k_2} is a commitment to $(w_{1,k_2}, \ldots, w_{m,k_2})$, then the prover needs to demonstrate that U_{k_3} is a commitment to $(\mathsf{NAND}(w_{1,k_1}, w_{1,k_2}), \ldots, \mathsf{NAND}(w_{m,k_1}, w_{m,k_2}))$. The size of each proof W_ℓ must also be succinct: $|W_\ell| = \mathsf{poly}(\lambda, \log m)$. The prover includes a proof of gate satisfiability W_ℓ for each gate $\ell \in [s]$.

The overall proof is $\pi = \big(\{(U_k, V_k)\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]}\big)$, and the proof size is $|C| \cdot \mathsf{poly}(\lambda, \log m)$, which satisfies the efficiency requirements on the BARG. To verify the proof, the verifier checks the following:

- **Input validity:** Without loss of generality, we associate wires $1, \ldots, n$ with the bits of the statement. The verifier checks that $U_1, \ldots, U_n$ are commitments to the bits of $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$. In our construction, each commitment is a *deterministic* function of the input vector, so the verifier can compute $U_1, \ldots, U_n$ directly from $\mathbf{x}_1, \ldots, \mathbf{x}_m$.
- **Wire validity:** For each $k \in [t]$, the verifier checks that U_k is a commitment to a 0/1 vector using V_k.
- **Gate consistency:** For each gate $G_\ell = (k_1, k_2, k_3)$, the verifier uses W_ℓ to check that U_{k_1}, U_{k_2}, and U_{k_3} are commitments to a set of valid wire assignments consistent with the gate operation G_ℓ.
- **Output satisfiability:** Let t be the index of the output wire in C. The verifier checks that the commitment to the output wire U_t is a commitment to the all-ones vector (indicating that all m instances accept).

Since the verifier needs to read the statement, the statement validity check runs in time $\mathsf{poly}(\lambda, n, m)$. The remaining checks run in time $|C| \cdot \mathsf{poly}(\lambda)$, which yields the desired verification complexity.

1.2.1 Construction from Composite-Order Pairing Groups

To illustrate the main ideas underlying our construction, we first describe it using symmetric composite-order groups and argue soundness under the subgroup decision assumption [BGN05]. We believe this construction is conceptually simple and best illustrates the core ideas behind the construction. The approach described here translates to the setting of asymmetric prime-order pairing groups to yield a construction from the k-Lin assumption.

[3] Technically, this is only required for the input wires corresponding to the witness.

[4] Our techniques extend naturally to support binary-valued gates that can compute *arbitrary* quadratic functions of their inputs; see Remark 4.16.

Composite-order pairing groups. A symmetric composite-order pairing group consists of two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ of order $N = pq$, where p, q are prime. Let g be a generator of $\mathbb{G}$. By the Chinese Remainder Theorem, we can write $\mathbb{G} \cong \mathbb{G}_p \times \mathbb{G}_q$, where $\mathbb{G}_p$ is a subgroup of order p (generated by $g_p = g^q$) and $\mathbb{G}_q$ is a subgroup of order q (generated by $g_q = g^p$). Additionally, there exists an efficiently-computable, non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ called the "pairing:" namely, for all $a, b \in \mathbb{Z}_N$, it holds that $e(g^a, g^b) = e(g, g)^{ab}$. Finally, the subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ are orthogonal: $e(g_p, g_q) = 1$, where 1 denotes the identity element in $\mathbb{G}_T$. In our construction, the real scheme operates entirely in the order-p subgroup $\mathbb{G}_p$ of $\mathbb{G}$; the full group $\mathbb{G}$ only plays a role in the soundness analysis.

Vector commitments. The first ingredient we need to implement the above blueprint is a vector commitment scheme for vectors of dimension m (m being the number of instances). We start by constructing a common reference string with m group elements $(A_1, \ldots, A_m)$ where each $A_i = g_p^{\alpha_i}$ for some $\alpha_i \xleftarrow{R} \mathbb{Z}_N$. A commitment to a vector $(w_{1,k}, \ldots, w_{m,k})$ is a subset product of the associated group elements $U_k = \prod_{i\in[m]} A_i^{w_{i,k}} = g_p^{\sum_{i\in[m]} \alpha_i w_{i,k}} \in \mathbb{G}_p$. We note that this is essentially the vector commitment scheme of Catalano and Fiore [CF13] instantiated in $\mathbb{G}_p$, but without randomization (in our setting, we do *not* require a hiding property on the commitments). With this instantiation, the commitment to each wire has size $\mathrm{poly}(\lambda)$, and is independent of m.

Wire validity checks. The second ingredient we require is a way for the prover to demonstrate that the committed values satisfy the wire validity and gate consistency relations. We start by describing the wire validity checks. Consider a vector of candidate wire assignments $(w_1, \ldots, w_m)$. The prover needs to convince the verifier that $w_i \in \{0, 1\}$ for all $i \in [m]$, or equivalently, that $w_i^2 = w_i$. Now, a correctly-generated commitment to $(w_1, \ldots, w_m)$ is an encoding of $\sum_{i\in[m]} \alpha_i w_i$ (in the exponent). We can now write

$$\left(\sum_{i\in[m]} \alpha_i\right)\left(\sum_{i\in[m]} \alpha_i w_i\right) = \sum_{i\in[m]} \alpha_i^2 w_i + \sum_{i\neq j} \alpha_i \alpha_j w_j$$

$$\left(\sum_{i\in[m]} \alpha_i w_i\right)^2 = \sum_{i\in[m]} \alpha_i^2 w_i^2 + \sum_{i\neq j} \alpha_i \alpha_j w_i w_j.$$

When $w_i^2 = w_i$, the difference between these two expressions is $\sum_{i\neq j} \alpha_i \alpha_j (1 - w_i) w_j$. Notably, this difference is a linear combination of the products $\alpha_i \alpha_j$ where $i \neq j$; we refer to these terms as the *cross terms*. Conversely, if $w_i^2 \neq w_i$ for some i, then the difference between the two relations *always* depends on the *non-cross-term* α_i^2. This suggests the following strategy for proof generation and verification: we publish encodings $B_{i,j} := g_p^{\alpha_i \alpha_j}$ for $i \neq j$ in the CRS to allow the prover to "cancel out" cross terms but *not* the non-cross terms. We also include an encoding $A := \prod_{i\in[m]} A_i = g_p^{\sum_{i\in[m]} \alpha_i}$ that will be used for verification. Specifically, we define the CRS to be

$$\mathsf{crs} = \left(\{A_i := g_p^{\alpha_i}\}_{i\in[m]}\,,\; A := \textstyle\prod_{i\in[m]} A_i = g_p^{\sum_{i\in[m]} \alpha_i}\,,\; \{B_{i,j} := g_p^{\alpha_i \alpha_j}\}_{i\neq j}\right). \tag{1.1}$$

Then, the prover can compute the quantity $V = \prod_{i\neq j} B_{i,j}^{(1-w_i)w_j} = g_p^{\sum_{i\neq j} \alpha_i \alpha_j (1-w_i) w_j}$. By the above relations, we see that if $U = g_p^{\sum_{i\in[m]} \alpha_i w_i}$, then

$$e(A, U) = e(U, U) e(g_p, V). \tag{1.2}$$

The analysis above shows that if U is a valid commitment to a binary vector, then the prover can always compute V that satisfies the verification relation. When U is *not* a commitment to a binary vector, we need to argue that the prover cannot craft a proof V that satisfies Eq. (1.2). The intuition is that there will be "non-cross-terms" that cannot be cancelled using the components available to the prover. Formalizing this intuition requires some care and we provide additional details below. We also note here that the size of the CRS (Eq. (1.1)) in our construction scales *quadratically* with the number of instances m. In the following, we will describe a bootstrapping technique to reduce the CRS size to scale with m^ε for any constant $\varepsilon > 0$.

Gate consistency checks. The approach we take for wire validity checks readily extends to enable gate consistency checks. We describe our approach for verifying a single NAND gate. To simplify the description, suppose U_1 and U_2 are vector commitments to the input wires $(w_{1,1}, \ldots, w_{m,1})$ and $(w_{1,2}, \ldots, w_{m,2})$, and U_3 is a vector commitment to the output wire $(w_{1,3}, \ldots, w_{m,3})$. The prover wants to show that $w_{i,3} = \mathsf{NAND}(w_{i,1}, w_{i,2})$ for all $i \in [m]$. This is equivalent to checking satisfiability of the *quadratic* relation $w_{i,3} + w_{i,1}w_{i,2} = 1$. In this case, the prover computes the element $W \in \mathbb{G}_p$ such that

$$\frac{e(A, U_3)e(U_1, U_2)}{e(A, A)} = e(g_p, W). \tag{1.3}$$

Suppose U_1, U_2, U_3 are properly-generated commitments. Then, if we consider the exponents for the left-hand side of the verification relation, we have

$$\underbrace{\sum_{i\in[m]} \alpha_i^2 w_{i,3} + \sum_{i\neq j} \alpha_i \alpha_j w_{j,3}}_{e(A,U_3)} + \underbrace{\sum_{i\in[m]} \alpha_i^2 w_{i,1} w_{i,2} + \sum_{i\neq j} \alpha_i \alpha_j w_{i,1} w_{j,2}}_{e(U_1,U_2)} - \underbrace{\sum_{i\in[m]} \alpha_i^2 - \sum_{i\neq j} \alpha_i \alpha_j}_{e(A,A)}.$$

If $w_{i,3} + w_{i,1}w_{i,2} = 1$, then all of the non-cross terms vanish, and we are left with $\sum_{i\neq j} \alpha_i \alpha_j (w_{j,3} + w_{i,1}w_{j,2} - 1)$. The prover can thus set $W = \prod_{i\neq j} B_{i,j}^{w_{j,3}+w_{i,1}w_{j,2}-1}$ to satisfy the above verification relation. Similar to the case with wire consistency checks, we now have to show that if there exists an $i \in [m]$ where $w_{i,3} + w_{i,1}w_{i,2} \neq 1$, then the prover is *unable* to compute a W that satisfies Eq. (1.3).

Proving soundness. To argue soundness of our argument system, we take the dual-mode approach from [CJJ21a, CJJ21b].[5] Specifically in this setting, there are two computationally indistinguishable ways to sample the CRS: (1) the normal mode described above; and (2) a trapdoor mode that takes as input an instance index $i^* \in [m]$ and outputs a trapdoor CRS crs^*. The requirement is that in trapdoor mode, the scheme is *statistically* sound for instance i^*. Namely, with overwhelming probability over the choice of crs^*, there does *not* exist any proof π for $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ that convinces the verifier when $\mathbf{x}_{i^*}$ is false. However, it is still possible that there exists valid proofs of tuples where $\mathbf{x}_{i^*}$ is true but $\mathbf{x}_i$ is false for some $i \neq i^*$. By a standard hybrid argument, it is easy to see that a BARG with this dual-mode "somewhere statistical soundness" property also satisfies *non-adaptive soundness* (i.e., soundness for statements that are *independent* of the CRS).[6] Achieving the stronger notion of *adaptive* soundness where security holds for statements that depend on the CRS seems challenging and in certain settings, will either require non-black-box techniques or basing security on non-falsifiable assumptions [GW11, BHK17].

Somewhere statistical soundness. To argue that our construction above satisfies somewhere statistical soundness, we start by describing the trapdoor CRS. To ensure statistical soundness for index $i^* \in [m]$, we replace the encoding $A_{i^*} = g_p^{\alpha_{i^*}}$ associated with instance i^* with $A_{i^*} \leftarrow g^{\alpha_{i^*}} \in \mathbb{G}$. Critically, A_{i^*} is now in the *full group* rather than the order-p subgroup $\mathbb{G}_p$. The encodings A_i associated with instances $i \neq i^*$ are still sampled from $\mathbb{G}_p$. We can construct the cross terms $B_{i,j}$ in a similar manner as before: the components for $i, j \neq i^*$ are unaffected and we set $B_{i^*,j} = B_{j,i^*} = A_{i^*}^{\alpha_j} \in \mathbb{G}$. The trapdoor CRS is computationally indistinguishable from the normal CRS by the subgroup decision assumption [BGN05]. Consider the wire consistency checks and gate consistency checks:

- **Wire consistency checks.** Let $U \in \mathbb{G}$ be a commitment to a tuple of wire values and $V \in \mathbb{G}$ be the wire consistency proof. We can decompose U as $U = g_p^{\beta_p} g_q^{\beta_q}$ for some $\beta_p \in \mathbb{Z}_p, \beta_q \in \mathbb{Z}_q$. Moreover, by construction, the verification component A is defined to be $A = \prod_{i\in[m]} A_i = g_p^{\sum_{i\in[m]} \alpha_i} g_q^{\alpha_{i^*}}$. Consider now the verification relation from Eq. (1.2). If this relation holds in $\mathbb{G}_T$, it must in particular hold in the order-q subgroup of $\mathbb{G}_T$. The key observation is that projecting the relation into the order-q subgroup of $\mathbb{G}_T$ *isolates* instance i^* (since

[5] This is different from the notion of "dual-mode" proof system often encountered in the setting of non-interactive zero-knowledge (NIZK) [GOS06, PS19, LPWW20]. There, the CRS can be sampled in two computationally indistinguishable modes: one mode ensures statistical soundness and the other ensures statistical zero knowledge.

[6] Our construction satisfies the stronger notion of semi-adaptive somewhere soundness [CJJ21b], where the adversary first commits to an index i^*, but is allowed to choose the statements $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ after seeing the CRS. The adversary wins if the proof is valid but $\mathbf{x}_{i^*}$ is false. This notion is needed for the implications to delegation.

only the encoding A_{i^*} contains components in the order-q subgroup). Moreover, the pairing $e(g_p, V)$ *vanishes* in the order-q subgroup, so the prover has *no control* over the validity check in the order-q subgroup. Now, for Eq. (1.2) to be satisfied, it must be the case that $\alpha_{i^*}\beta_q = \beta_q^2 \bmod q$. Thus, either $\beta_q = 0$ or $\beta_q = \alpha_{i^*}$ and so the wire checks ensure that $U_k = g_p^{\beta_p} g_q^{\xi_k \alpha_{i^*}}$ where $\xi_k \in \{0,1\}$ for all $k \in [m]$.

- **Gate consistency checks.** Now, consider the gate consistency checks. We again consider the projection of the pairing check into the order-q subgroup. If we project Eq. (1.3) in the order-q subgroup and using the above relations for U_k and A, we obtain the relation

$$\xi_{k_3}\alpha_{i^*}^2 + \xi_{k_1}\xi_{k_2}\alpha_{i^*}^2 - \alpha_{i^*}^2 = 0 \bmod q.$$

  If $\alpha_{i^*} \neq 0 \bmod q$, then $\xi_{k_3} + \xi_{k_1}\xi_{k_2} - 1 = 0 \bmod q$. Since $\xi_{k_1}, \xi_{k_2}, \xi_{k_3} \in \{0,1\}$, this means that $\xi_{k_3} = \mathsf{NAND}(\xi_{k_1}, \xi_{k_2})$.

The above relations show that $(\xi_1, \ldots, \xi_t) \in \{0,1\}^t$ constitutes a valid assignment to the wires of $C((\xi_1, \ldots, \xi_n), \mathbf{w}^*)$ where $\mathbf{w}^* = (\xi_{n+1}, \ldots, \xi_{n+h})$. Again considering the verification relations in the order-q subgroup, the input validity checks ensure that $\mathbf{x}_{i^*} = (\xi_1, \ldots, \xi_n)$ and the output satisfiability check ensures that $C(\mathbf{x}_{i^*}, \mathbf{w}^*) = \xi_t = 1$. The above argument shows that if all of the validity checks pass, then we can *extract* a witness for instance i^*. Thus, statistical soundness for instance $\mathbf{x}_{i^*}$ holds. In fact, this extraction procedure can be made efficient given a trapdoor (i.e., the factorization of N). We provide the full construction and security analysis in Section 3.

1.2.2 The Prime-Order Instantiation, Bootstrapping, and Applications

The BARG construction from symmetric composite-order groups is conceptually simple to describe and illustrates the main ideas behind our construction. We now describe several extensions and generalizations of these ideas.

Instantiation from k-Lin. The ideas underlying the composite-order construction (Sections 1.2.1 and 3) naturally extend to the setting of asymmetric prime-order groups. Recall that an asymmetric prime-order group consists of two base groups $\mathbb{G}_1$ and $\mathbb{G}_2$, a target group $\mathbb{G}_T$, all of prime order p, and an efficiently-computable, non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In this setting, we can base security on the standard k-Lin assumption for any $k \geq 1$. Recall that the case $k = 1$ corresponds to the SXDH assumption (i.e., DDH in $\mathbb{G}_1$ and $\mathbb{G}_2$) and the case $k = 2$ corresponds to the DLIN assumption [BBS04, HK07, Sha07]. The key property we relied on in the soundness analysis of the composite-order construction is the ability to isolate a single instance by *projecting* the verification relations into a suitable subgroup. In the prime-order setting, we can simulate this projection property by considering subspaces of vector spaces [GS08, Fre10]. We refer to Section 4 for the full description and security analysis.

Bootstrapping to reduce CRS size. The size of the CRS in the above construction scales *quadratically* with the number of instances m (due to the cross terms). However, we can adapt the bootstrapping approach from Kalai et al. [KPY19] reduce the size of the CRS to grow with m^ε (for any constant $\varepsilon > 0$). Soundness of the bootstrapping construction critically relies on the ability to extract the witness for *one* of the instances in the BARG.

The construction is simple. To verify statements $\mathbf{x}_1, \ldots, \mathbf{x}_m$, we consider a two-tiered construction where we group the statements into m/B batches of statements, each containing exactly B statements. We use a BARG (on B instances) to prove that all of the statements in each batch $(\mathbf{x}_{B(i-1)+1}, \ldots, \mathbf{x}_{iB})$ are true. Let π_i be the BARG proof for the i^{th} batch. The prover then shows that it knows accepting proofs $\pi_1, \ldots, \pi_{m/B}$ of each of the m/B batches of statements. Here, it will be critical that the size of the BARG verification circuit for checking π_i be *sublinear* in the batch size B. This is not possible in general since the verification circuit has to read the statement which already has length B. However, when the underlying BARG satisfies a "split verification" property (Definition 2.9), where the verification algorithm decomposes into (1) a circuit-independent preprocessing step that reads the statement and outputs a *succinct* verification key vk; and (2) a fast "online" verification step whose running time is *polylogarithmic* in the number of instances, it suffices to use the BARG to *only* check the online verification step.

Now, if we set $B = \sqrt{m}$ in this framework, both the BARG for checking each batch of B statements as well as the BARG for verifying the $m/B = \sqrt{m}$ batches are BARGs on $\sqrt{m}$ instances. Thus, we can use a BARG on $\sqrt{m}$ instances to construct a BARG on m instances. If we start with a BARG with CRS size m^d, then the two-tiered construction reduces the CRS size to roughly $m^{d/2}$. We can apply this approach recursively (with a constant number of iterations) to reduce the CRS size from $\mathsf{poly}(\lambda, m)$ to $m^\varepsilon \cdot \mathsf{poly}(\lambda)$ for any constant $\varepsilon > 0$. We refer to Section 5 for the full details.

Application to delegation. Choudhuri et al. [CJJ21b] showed how to combine a "BARG for index languages" with a somewhere extractable commitment scheme to obtain a delegation scheme for RAM programs. In a BARG for index languages, the statements to the m instances are always fixed to be the binary representation of the integers $1, \ldots, m$. In this setting, the prover and the verifier do *not* need to read the statement anymore, and correspondingly, the verification algorithm is required to run in time $\mathsf{poly}(\lambda, \log m, |C|)$ when checking a circuit C.

Our BARG construction extends naturally to this setting. In the construction described in Section 1.2.1 (see also Section 3), the verifier starts by computing the commitments $U_1, \ldots, U_n$ to the bits of the statement. This takes time $\mathsf{poly}(\lambda, n, m)$ since the verifier has to minimally read the statement (of length mn). However in the case of an index BARG, the statements are known in *advance*, so the encodings U_i can be computed in advance and included as part of a verification key $\mathsf{vk} = (U_1, \ldots U_n)$ that the verifier uses for verification. Given vk, the statement validity checks can be implemented by simply comparing the precomputed commitments with those provided by the adversary; notably this check is now *independent* of the number of instances. Using the precomputed commitments, we can bring the overall verification cost down to $|C| \cdot \mathsf{poly}(\lambda, \log m)$, which meets the efficiency requirements for an index BARG.

The second ingredient we require to instantiate the Choudhuri et al. [CJJ21b] compiler is a somewhere extractable commitment scheme. Our techniques for constructing BARGs can also be used to directly construct a somewhere extractable commitment scheme (when combined with a somewhere statistically binding hash function [HW15, OPWW15]). We can thus appeal to the compiler of Choudhuri et al. to obtain a delegation scheme for RAM programs from the SXDH assumption in bilinear groups.[7] Similar to the case with BARGs, we first describe a construction with a long CRS where the length of the CRS grows quadratically with the length of the committed message (Section 6.2). We then describe a similar kind of bootstrapping technique to obtain a somewhere extractable commitment scheme with a CRS of size sublinear in the message size (Section 6.3). We refer to Section 6 for the full details.

Application to aggregate signatures. As described in Section 1.1, our BARG construction directly implies an aggregate signature scheme supporting bounded aggregation. We describe this construction in Section 7.

Generalized BARGs. As previously noted for the case of BARGs for index languages, when the statements are fixed in advance, we can *precompute* commitments to them during setup and include the honestly-generated commitments to their values as part of a verification key. In this case, the verifier can use the precomputed encodings during verification and no longer needs to perform the statement validity checks. In Appendix A, we describe a more generalized view where some of the statement wires are fixed while others can be chosen by the prover. This generalization captures both the standard setting (where all of the statement wires can be chosen by the prover) and the BARG for index languages setting (where all of the statement wires are fixed ahead of time) as special cases.

1.3 Related Work

SNARGs. Batch arguments for NP can be constructed from any succinct non-interactive argument (SNARG) for NP. Existing constructions of SNARGs have either relied on random oracles [Mic95, BBHR18, COS20, CHM+20, Set20], the generic group model [Gro16], or strong non-falsifiable assumptions [Gro10, BCCT12, DFH12, Lip13, PHGR13, GGPR13, BCI+13, BCPR14, BISW17, BCC+17]. Indeed, Gentry and Wichs [GW11] showed that no construction of an (adaptively-sound) SNARG for NP can be proven secure via a black-box reduction to a falsifiable assumption [Nao03]. This separation also extends to adaptively-sound BARGs *of knowledge* (i.e., "BARKs") for NP [BHK17]. The only construction of non-adaptively sound SNARGs from falsifiable assumptions is the construction based on indistinguishability obfuscation [SW14]. We note that Lipmaa and Pavlyk [LP21] recently proposed a candidate SNARG from a non-standard, but falsifiable, q-type assumption on bilinear groups. However, we were recently informed [Wic22] that the proof of security was fundamentally flawed and later confirmed this with the authors of [LP21].

[7] While our BARG scheme can be based on the k-Lin assumption over bilinear groups for any $k \geq 1$, existing constructions of somewhere statistically binding hash functions [OPWW15] rely on the DDH assumption. As such, our current instantiation is based on SXDH. It seems plausible that the DDH-based construction of somewhere statistically binding hash functions can be extended to achieve hardness under the k-Lin assumption, but this is orthogonal to the primary focus of our work.

Batch arguments for NP. If we focus specifically on constructions of BARGs for NP, Kalai et al. [KPY19] showed how to construct a BARG for NP from a non-standard, but falsifiable, q-type assumption on bilinear groups. More recently, Choudhuri et al. gave constructions from subexponentially-hard DDH in pairing-free groups in conjunction with polynomial hardness of the QR assumption [CJJ21a], as well as from polynomial hardness of the LWE assumption [CJJ21b]. Both of these constructions leverage correlation-intractable hash functions. The size of the proof in the DDH + QR construction grows with $\sqrt{m}$, where m is the number of instances, while that in the LWE construction scales *polylogarithmically* with the number of instances. Our work provides the first BARG for NP from standard assumptions on bilinear groups (with proof size that is *independent* of the number of instances).

Interactive schemes. Batch arguments for NP have also been considered in the interactive setting. First, the classic IP = PSPACE theorem [LFKN90, Sha90] implies a interactive *proof* for batch NP verification, albeit with an *inefficient* prover. For interactive proofs with an *efficient* prover, batch verification is known for the class UP of NP languages with *unique* witnesses [RRR16, RRR18, RR20]. If we relax to interactive *arguments*, Brakerski et al. [BHK17] constructed 2-message BARGs for NP from any computational private information retrieval (PIR) scheme.

Delegation schemes. Many works have focused on constructing delegation schemes for deterministic computations. In the interactive setting, we have succinct *proofs* for both bounded-depth computations [GKR08] and bounded-space computations [RRR16]. In the non-interactive setting, Kalai et al. [KPY19] gave the first construction from a falsifiable (but non-standard) assumption on bilinear groups. Using correlation-intractable hash functions based on LWE, Jawale et al. [JKKZ21] and Choudhuri et al. [CJJ21b] constructed delegation schemes for bounded-depth computations and general polynomial-time computations, respectively. Recently, González and Zacharakis [GZ21] constructed a delegation scheme for arithmetic circuits with a *long* CRS from a *bilateral* (or "split") k-Lin assumption in asymmetric groups. The size of the CRS in their construction is *quadratic* in the circuit size. Our scheme is based on the vanilla SXDH assumption in asymmetric groups and has a CRS whose size is *sublinear* in the running time of the RAM computation (specifically, T^{ε} for any constant $\varepsilon > 0$, where T is the running time of the RAM computation).

Aggregate signatures. Aggregate signatures were introduced by Boneh et al. [BGLS03] who also gave an efficient construction using bilinear maps in the random oracle model. In the standard model, constructions of aggregate signatures have typically considered restricted settings such as sequential aggregation [LMRS04, LOS+06] where the aggregate signature is constructed by having each signer *sequentially* "add" its signature to an aggregated signature, or synchronized aggregation [GR06, AGH10, HW18], which assumes that signers have a synchronized clock and aggregation is only allowed on signatures from the same time period (with exactly 1 signature from each signer per time period). Other (standard model) constructions have relied on heavy tools such as multilinear maps [RS09, FHPS13] or indistinguishability obfuscation [HKW15]. Aggregate signatures can also be constructed generically from *adaptively-sound* succinct arguments *of knowledge* (SNARKs), which are only known from non-falsifiable assumptions or idealized models. In the case of bounded aggregation (where there is an *a priori* bound on the number of signatures that can be aggregated), the somewhere extractable BARG by Choudhuri et al. [CJJ21b] can be used to obtain a construction from LWE. Our work provides the first instantiation of an aggregate signature supporting bounded aggregation from standard assumptions over bilinear groups in the plain model.

2 Preliminaries

For a positive integer n, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For a positive integer $p \in \mathbb{N}$, we write $\mathbb{Z}_p$ to denote the ring of integers modulo p. We use bold-face uppercase letters (e.g., $\mathbf{A}$, $\mathbf{B}$ to denote matrices) and bold-face lowercase letters (e.g., $\mathbf{x}$, $\mathbf{w}$) to denote vectors. For a finite set S, we write $x \xleftarrow{R} S$ to indicate that x is sampled uniformly at random from S. We use non-bold-face letters to denote their components (e.g., $\mathbf{x} = (x_1, \ldots, x_n)$). We write $\mathsf{poly}(\lambda)$ to denote a 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 E occurs with overwhelming probability if its complement occurs with negligible probability. An algorithm is efficient if it runs in probabilistic polynomial time in its input length. We say that two families 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 algorithm

can distinguish them with non-negligible probability. We say they are statistically indistinguishable if the statistical distance between them is bounded by a negligible function.

2.1 Non-Interactive Batch Arguments for NP

In this work, we consider the NP-complete language of Boolean circuit satisfiability. For ease of exposition, we focus on Boolean circuits comprised exclusively of NAND gates in our main construction. In Remark 4.16, we describe how to generalize the construction to support gates that compute arbitrary quadratic relations over their inputs. This allows us to support both general gates (e.g., AND, OR, XOR) as well as gates with more than two inputs.

For a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ with t wires, we associate wires $1,\ldots,n$ with the bits of the statement $x_1,\ldots,x_n$, and wires $n+1,\ldots,n+h$ with the bits of the witness $w_1,\ldots,w_h$, respectively. We associate wire t with the output wire. We measure the size s of C by the number of NAND gates it has. By construction, $t \le n+h+s$. We now define the (batch) circuit satisfiability language we consider in this work:

Definition 2.1 (Circuit Satisfiability)**.** We define $\mathcal{L}_{\mathsf{CSAT}} = \{(C,\mathbf{x}) \mid \exists \mathbf{w} \in \{0,1\}^h : C(\mathbf{x},\mathbf{w}) = 1\}$ to be the language of Boolean circuit satisfiability, where $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is a Boolean circuit and $\mathbf{x} \in \{0,1\}^n$ is a statement. For a positive integer $m \in \mathbb{N}$, we define the *batch circuit satisfiability* language $\mathcal{L}_{\mathsf{BatchCSAT},m}$ as follows:

$$\mathcal{L}_{\mathsf{BatchCSAT},m} = \{(C,\mathbf{x}_1,\ldots,\mathbf{x}_m) \mid \forall i \in [m] : \exists \mathbf{w}_i \in \{0,1\}^h : C(\mathbf{x}_i,\mathbf{w}_i) = 1\},$$

where $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is a Boolean circuit and $\mathbf{x}_1,\ldots,\mathbf{x}_m \in \{0,1\}^n$ are the instances.

Definition 2.2 (Batch Argument for Circuit Satisfiability)**.** A non-interactive batch argument (BARG) for circuit satisfiability is a tuple of three efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ with the following properties:

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

Definition 2.3 (Completeness)**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ is complete if for all $\lambda, m, s \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, all statements $\mathbf{x}_1,\ldots,\mathbf{x}_m \in \{0,1\}^n$, and all witnesses $\mathbf{w}_1,\ldots,\mathbf{w}_m \in \{0,1\}^h$ where $C(\mathbf{x}_i,\mathbf{w}_i) = 1$ for all $i \in [m]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1,\ldots,\mathbf{x}_m), \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1,\ldots,\mathbf{x}_m), (\mathbf{w}_1,\ldots,\mathbf{w}_m)) \end{array}\right] = 1.$$

Definition 2.4 (Soundness)**.** Let $\Pi_{\mathsf{BARG}} = (\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ be a BARG. We consider two notions of soundness:

- **Non-adaptive soundness:** We say that Π_{BARG} satisfies non-adaptive soundness if for all polynomials $m = m(\lambda)$, $s = s(\lambda)$, and efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, and every statement $(C,\mathbf{x}_1,\ldots,\mathbf{x}_m) \notin \mathcal{L}_{\mathsf{BatchCSAT},m}$, where $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is a Boolean circuit of size at most $s(\lambda)$ and $\mathbf{x}_1,\ldots,\mathbf{x}_n \in \{0,1\}^n$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1,\ldots,\mathbf{x}_m), \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s); \\ \pi \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}, C, (\mathbf{x}_1,\ldots,\mathbf{x}_m)) \end{array}\right] = \mathsf{negl}(\lambda).$$

- **Adaptive soundness:** We say that Π_{BARG} is adaptively sound if for every efficient adversary $\mathcal{A}$ and every polynomial $m = m(\lambda)$, $s = s(\lambda)$, there exists a negligible function of $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1,\ldots,\mathbf{x}_m), \pi) = 1 \\ \text{and} \\ (C,\mathbf{x}_1,\ldots,\mathbf{x}_m) \notin \mathcal{L}_{\mathsf{BatchCSAT},m} \end{array} : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s); \\ (C,\mathbf{x}_1,\ldots,\mathbf{x}_m,\pi) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Definition 2.5 (Semi-Adaptive Somewhere Soundness [CJJ21b])**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ satisfies semi-adaptive somewhere soundness if there exists an efficient algorithm TrapSetup with the following properties:

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

We require TrapSetup satisfy the following two properties:

- **CRS indistinguishability:** For integers $m \in \mathbb{N}$, $s \in \mathbb{N}$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, define the CRS indistinguishability experiment $\mathsf{ExptCRS}_{\mathcal{A}}(\lambda, m, s, b)$ as follows:

  1. Algorithm $\mathcal{A}(1^\lambda, 1^m, 1^s)$ outputs an index $i^* \in [m]$.
  2. If $b = 0$, the challenger gives $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s)$ to $\mathcal{A}$. If $b = 1$, the challenger gives $\mathsf{crs}^* \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  Then, Π_{BARG} satisfies CRS indistinguishability if for every efficient adversary $\mathcal{A}$, every polynomial $m = m(\lambda)$, $s = s(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

- **Somewhere soundness in trapdoor mode:** Define the somewhere soundness security game between an adversary $\mathcal{A}$ and a challenger as follows:

  - Algorithm $\mathcal{A}(1^\lambda, 1^m, 1^s)$ outputs an index $i^* \in [m]$.
  - The challenger samples $\mathsf{crs}^* \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$ 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, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and a proof π. The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 1$ and $(C, \mathbf{x}_{i^*}) \notin \mathcal{L}_{\mathsf{CSAT}}$. Otherwise, the output is $b = 0$.

  Then, Π_{BARG} satisfies somewhere soundness in trapdoor mode if for every adversary $\mathcal{A}$, and every polynomial $m = m(\lambda)$, $s = s(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the somewhere soundness security game.

Definition 2.6 (Somewhere Argument of Knowledge [CJJ21b])**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a somewhere argument of knowledge if there exists a pair of efficient algorithms $(\mathsf{TrapSetup}, \mathsf{Extract})$ with the following properties:

- $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*) \to (\mathsf{crs}^*, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $m \in \mathbb{N}$, the size of the circuit $s \in \mathbb{N}$, and an index $i^* \in [m]$, the trapdoor setup algorithm outputs a common reference string crs^* and an extraction trapdoor td.
- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) \to \mathbf{w}^*$ On input the trapdoor td, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m$, and a proof π, the extraction algorithm outputs a witness $\mathbf{w}^* \in \{0,1\}^h$. The extraction algorithm is deterministic.

We require $(\mathsf{TrapSetup}, \mathsf{Extract})$ to satisfy the following two properties:

- **CRS indistinguishability:** Same as in Definition 2.5.
- **Somewhere extractable in trapdoor mode:** Define the somewhere extractable security game between an adversary $\mathcal{A}$ and a challenger as follows:

  - Algorithm $\mathcal{A}(1^\lambda, 1^m, 1^s)$ outputs an index $i^* \in [m]$.
  - The challenger samples $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$ 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, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and a proof π. Let $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{w}_m), \pi)$.

– The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 1$ and $C(\mathbf{x}_{i^*}, \mathbf{w}^*) \neq 1$. Otherwise, the output is $b = 0$.

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

Remark 2.7 (Soundness Notions)**.** The notion of semi-adaptive somewhere soundness from Definition 2.5 is stronger than and implies non-adaptive soundness. Somewhere extractability (Definition 2.6) is a further strengthening of semi-adaptive somewhere soundness.

Definition 2.8 (Succinctness)**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is succinct if there exists a fixed polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $\lambda, m, s \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$, and all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, the following properties hold:

- **Succinct proofs:** The proof π output by $\mathsf{Prove}(\mathsf{crs}, C, \cdot, \cdot)$ satisfies $|\pi| \leq \mathsf{poly}(\lambda, \log m, s)$.
- **Succinct CRS:** $|\mathsf{crs}| \leq \mathsf{poly}(\lambda, m, n) + \mathsf{poly}(\lambda, \log m, s)$.
- **Succinct verification:** The verification algorithm runs in time $\mathsf{poly}(\lambda, m, n) + \mathsf{poly}(\lambda, \log m, s)$.

BARGs with split verification. Our bootstrapping construction in Section 5 (for reducing the size of the CRS) will rely on a BARG with a split verification property where the verification algorithm can be decomposed into a input-dependent algorithm that pre-processes the statements into a short verification key together with a fast online verification algorithm that takes the precomputed verification key and checks the proof. A similar property was also considered by Choudhuri et al. [CJJ21b] to realize their RAM delegation construction.

Definition 2.9 (BARG with Split Verification)**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ supports split verification if there exists a pair of efficient and *deterministic* algorithms $(\mathsf{GenVK}, \mathsf{OnlineVerify})$ with the following properties:

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

Then, we say Π_{BARG} supports split verification if $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$ outputs

$$\mathsf{OnlineVerify}(\mathsf{GenVK}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_m)), C, \pi).$$

We additionally require that there exists a fixed polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $\lambda, m, s \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$, and all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, the following efficiency properties hold (in addition to the properties in Definition 2.8):

- **Succinct verification key:** The verification key generation algorithm GenVK runs in time $\mathsf{poly}(\lambda, m, n)$, and the size of the vk output by GenVK satisfies $|\mathsf{vk}| \leq \mathsf{poly}(\lambda, \log m, n)$.
- **Succinct online verification:** The algorithm $\mathsf{OnlineVerify}(\mathsf{vk}, C, \pi)$ runs in time $\mathsf{poly}(\lambda, \log m, s)$.

Remark 2.10 (BARGs for Index Languages [CJJ21b])**.** BARGs for index languages [CJJ21b] ("index BARGs") are a useful building block for constructing delegation schemes for RAM programs. In an index BARG with m instances, the statement to the i^{th} instance is the binary representation of the index i. Since the statements are fixed in an index BARG, they are *not* included in the input to the Prove and Verify algorithms. Moreover, the running time

of the verification algorithm Verify on input a verification key vk,[8] a circuit C, and a proof π is required to be $\mathsf{poly}(\lambda, \log m, |C|)$. It is easy to see that any BARG with a split verification procedure can also be used to build an index BARG. Specifically, after the Setup algorithm samples the common reference string crs, it precomputes the (short) verification key $\mathsf{vk} \leftarrow \mathsf{GenVK}(\mathsf{crs}, (1, 2, \ldots, m))$. The verification algorithm Verify then takes as input the precomputed verification key vk, the circuit C, and the proof π, and outputs $\mathsf{OnlineVerify}(\mathsf{vk}, C, \pi)$. The succinctness requirements on the split verification procedure implies the succinctness requirement on the index BARG.

3 BARG for NP from Subgroup Decision in Bilinear Groups

In this section, we show how to construct a BARGs from the subgroup decision assumption over symmetric composite-order groups. We refer to Section 1.2.1 for a general overview of this construction. We start by recalling the definition of a composite-order pairing group [BGN05] and the subgroup decision assumption.

Definition 3.1 (Composite-Order Bilinear Groups [BGN05])**.** A (symmetric) composite-order bilinear group generator is an efficient algorithm CompGroupGen that takes as input the security parameter λ and outputs a description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, q, g, e)$ of a bilinear group where p, q are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order $N = pq$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map (called the "pairing"). We require that the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing operation to be efficiently computable.

Definition 3.2 (Subgroup Decision [BGN05])**.** The subgroup decision assumption holds with respect to a composite-order bilinear group generator CompGroupGen if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathcal{A}((\mathbb{G}, \mathbb{G}_T, N, g_p, e), g^r) = 1] - \Pr[\mathcal{A}((\mathbb{G}, \mathbb{G}_T, N, g_p, e), g_p^r) = 1]\right| = \mathsf{negl}(\lambda),$$

where $(\mathbb{G}, \mathbb{G}_T, p, q, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$, $N \leftarrow pq$, $g_p \leftarrow g^q$, and $r \xleftarrow{R} \mathbb{Z}_N$.

Construction 3.3 (BARG for NP from Subgroup Decision)**.** Take any integer $m \in \mathbb{N}$. We construct a BARG with split verification for the language of circuit satisfiability as follows:

- $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$: On input the security parameter λ, the number of instances m, and the bound on the circuit size s, the setup algorithm does the following:
  - Run $(\mathbb{G}, \mathbb{G}_T, p, q, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and let $N = pq$, $g_p \leftarrow g^q$. In particular, g_p generates a subgroup of order p in $\mathbb{G}$. Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_p, e)$.
  - For each $i \in [m]$, sample $\alpha_i \xleftarrow{R} \mathbb{Z}_N$. For each $i \in [m]$, let $A_i \leftarrow g_p^{\alpha_i}$. Let $A \leftarrow \prod_{i \in [m]} A_i$.
  - For each $i, j \in [m]$ where $i \neq j$, compute $B_{i,j} \leftarrow g_p^{\alpha_i \alpha_j}$.
  - Output the common reference string $\mathsf{crs} = \left(\mathcal{G}, A, \{A_i\}_{i \in [m]}, \{B_{i,j}\}_{i \neq j}\right)$.
- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$: On input the common reference string $\mathsf{crs} = (\mathcal{G}, A, \{A_i\}_{i \in [m]}, \{B_{i,j}\}_{i \neq j})$, the circuit $C : \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$, define t to be the number of wires in C and s to be the number of gates in C. Then, for $i \in [m]$ and $j \in [t]$, let $w_{i,j} \in \{0,1\}$ be the value of wire j in $C(\mathbf{x}_i, \mathbf{w}_i)$. The prover proceeds as follows:
  - **Encoding wire values:** For each $k \in [t]$, let $U_k = \prod_{i \in [m]} A_i^{w_{i,k}}$.
  - **Validity of wire assignments:** For each $k \in [t]$, let $V_k = \prod_{i \neq j} B_{i,j}^{(1 - w_{i,k}) w_{j,k}}$.
  - **Validity of gate computation:** For each NAND gate $G_\ell = (k_1, k_2, k_3) \in [t]^3$ (where $\ell \in [s]$), compute $W_\ell = \prod_{i \neq j} B_{i,j}^{1 - w_{i,k_1} w_{j,k_2} - w_{j,k_3}}$

[8] Here, we allow the verification algorithm to take in a separate verification key vk, which may be *shorter* than the full common reference string crs. Note that the vk is assumed to be public (i.e., the CRS contains vk and possibly additional components used to construct proofs).

Finally, output the proof $\pi = \big(\{U_k, V_k\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]}\big)$.

- $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$: We decompose the verification algorithm into $(\mathsf{GenVK}, \mathsf{OnlineVerify})$:
  - $\mathsf{GenVK}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_m))$: On input the common reference string $\mathsf{crs} = (\mathcal{G}, A, \{A_i\}_{i\in[m]}, \{B_{i,j}\}_{i\neq j})$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, the verification key generation algorithm computes $U_k^* = \prod_{i\in[m]} A_i^{x_{i,k}}$ for each $k \in [n]$, and outputs the verification key $\mathsf{vk} = (U_1^*, \ldots, U_n^*)$.
  - $\mathsf{OnlineVerify}(\mathsf{vk}, C, \pi)$: On input the verification key $\mathsf{vk} = (U_1^*, \ldots, U_n^*)$, a circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and the proof $\pi = (\{U_k, V_k\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]})$, the verification algorithm checks the following:
    - **Validity of statement:** For each input wire $k \in [n]$, $U_k = U_k^*$.
    - **Validity of wire assignments:** For each $k \in [t]$,
    $$e(A, U_k) = e(g_p, V_k)e(U_k, U_k). \tag{3.1}$$
    - **Validity of gate computation:** For each gate $G_\ell = (k_1, k_2, k_3) \in [t]^3$,
    $$e(A, A) = e(U_{k_1}, U_{k_2})e(A, U_{k_3})e(g_p, W_\ell). \tag{3.2}$$
    - **Output satisfiability:** The output encoding U_t satisfies $U_t = A$.

    The algorithm outputs 1 if all checks pass, and outputs 0 otherwise.

  The verification algorithm outputs $\mathsf{OnlineVerify}(\mathsf{GenVK}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_m)), C, \pi)$.

Theorem 3.4 (Completeness). *Construction 3.3 is complete.*

Proof. Take any circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$ such that $C(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all $i \in [m]$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$. We show that $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$ outputs 1. Consider each of the verification relations:

- **Validity of statement:** By construction of GenVK, $U_k^* = \prod_{i\in[m]} A_i^{x_{i,k}}$ for each $k \in [n]$. By construction of Prove, $U_k = \prod_{i\in[m]} A_i^{w_{i,k}}$. By definition, the first n wires in C coincide with the wires to the statement, so $w_{i,k} = x_{i,k}$ for $k \in [n]$, and $U_k = U_k^*$ for all $k \in [n]$.

- **Validity of wire assignments:** Take any $k \in [t]$. Then $U_k = \prod_{i\in[m]} A_i^{w_{i,k}} = g_p^{\sum_{i\in[m]} \alpha_i w_{i,k}}$. Now,
$$\left(\sum_{i\in[m]} \alpha_i\right)\left(\sum_{j\in[m]} \alpha_j w_{j,k}\right) = \sum_{i\in[m]} \alpha_i^2 w_{i,k} + \sum_{i\neq j} \alpha_i\alpha_j w_{j,k},$$
and
$$\left(\sum_{i\in[m]} \alpha_i w_{i,k}\right)\left(\sum_{j\in[m]} \alpha_j w_{j,k}\right) = \sum_{i\in[m]} \alpha_i^2 w_{i,k} + \sum_{i\neq j} \alpha_i\alpha_j w_{i,k}w_{j,k},$$
using the fact that $w_{i,k} \in \{0,1\}$ so $w_{i,k}^2 = w_{i,k}$. Finally $V_k = \prod_{i\neq j} B_{i,j}^{(1-w_{i,k})w_{j,k}} = g_p^{\sum_{i\neq j} \alpha_i\alpha_j(1-w_{i,k})w_{j,k}}$. Thus, we can write
$$\begin{aligned} e(g_p, V_k)e(U_k, U_k) &= e(g_p, g_p)^{\sum_{i\neq j} \alpha_i\alpha_j(1-w_{i,k})w_{j,k} + \sum_{i\in[m]} \alpha_i^2 w_{i,k} + \sum_{i\neq j} \alpha_i\alpha_j w_{i,k}w_{j,k}} \\ &= e(g_p, g_p)^{\sum_{i\in[m]} \alpha_i^2 w_{i,k} + \sum_{i\neq j} \alpha_i\alpha_j w_{j,k}} \\ &= e(A, U_k). \end{aligned}$$

- **Validity of gate computation:** Take any gate $G_\ell = (k_1, k_2, k_3) \in [t]^3$. Consider first the exponents for the terms $e(U_{k_1}, U_{k_2})$, $e(A, U_{k_3})$, and $e(A, A)$:

$$\left(\sum_{i\in[m]} \alpha_i w_{i,k_1}\right)\left(\sum_{j\in[m]} \alpha_j w_{j,k_2}\right) = \sum_{i\in[m]} \alpha_i^2 w_{i,k_1} w_{i,k_2} + \sum_{i\neq j} \alpha_i \alpha_j w_{i,k_1} w_{j,k_2}$$
$$\left(\sum_{i\in[m]} \alpha_i\right)\left(\sum_{j\in[m]} \alpha_j w_{j,k_3}\right) = \sum_{i\in[m]} \alpha_i^2 w_{i,k_3} + \sum_{i\neq j} \alpha_i \alpha_j w_{j,k_3}$$
$$\left(\sum_{i\in[m]} \alpha_i\right)\left(\sum_{j\in[m]} \alpha_j\right) = \sum_{i\in[m]} \alpha_i^2 + \sum_{i\neq j} \alpha_i \alpha_j.$$

  By definition $w_{i,k_3} = \mathsf{NAND}(w_{i,k_1}, w_{i,k_2})$. This means that for each $i \in [m]$, either ($w_{i,k_1} w_{i,k_2} = 1$ and $w_{i,k_3} = 0$) or ($w_{i,k_1} w_{i,k_2} = 0$ and $w_{i,k_3} = 1$). This means that

$$\sum_{i\in[m]} \alpha_i^2 (w_{i,k_1} w_{i,k_2} + w_{i,k_3}) = \sum_{i\in[m]} \alpha_i^2.$$

  Combining the above relations in the exponent, we have that

$$\begin{aligned}\frac{e(A,A)}{e(U_{k_1}, U_{k_2}) e(A, U_{k_3})} &= \frac{e(g_p, g_p)^{\sum_{i\in[m]} \alpha_i^2 + \sum_{i\neq j} \alpha_i \alpha_j}}{e(g_p, g_p)^{\sum_{i\in[m]} \alpha_i^2 + \sum_{i\neq j} \alpha_i \alpha_j (w_{i,k_1} w_{j,k_2} + w_{j,k_3})}} \\ &= \prod_{i\neq j} e(g_p, B_{i,j})^{1 - w_{i,k_1} w_{j,k_2} - w_{j,k_3}} \\ &= e(g_p, W_\ell).\end{aligned}$$

- **Output satisfiability:** Since $C(\mathbf{x}_i, \mathbf{w}_i) = 1$, it follows that $w_{i,t} = 1$ for all $i \in [m]$. By definition, $U_t = \prod_{i\in[m]} A_i^{w_{i,t}} = \prod_{i\in[m]} A_i = A$. □

Theorem 3.5 (Somewhere Argument of Knowledge)**.** *Suppose the subgroup decision assumption holds with respect to* CompGroupGen. *Then, Construction 3.3 is a somewhere argument of knowledge.*

Proof. We start by defining the trapdoor setup and extraction algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$: The trapdoor algorithm uses the following procedure (we highlight in green the differences in the common reference string components between TrapSetup and Setup):
  1. Run $(\mathbb{G}, \mathbb{G}_T, p, q, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and let $N = pq$, $g_p \leftarrow g^q$. Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_p, e)$.
  2. For each $i \in [m]$, sample $\alpha_i \xleftarrow{R} \mathbb{Z}_N$. For each $i \neq i^*$, let $A_i \leftarrow g_p^{\alpha_i}$. Let $A_{i^*} \leftarrow g^{\alpha_{i^*}}$. Let $A \leftarrow A_{i^*} \prod_{i\neq i^*} A_i$.
  3. For each $i, j \in [m]$ where $i \neq j$ and $i, j \neq i^*$, compute $B_{i,j} \leftarrow g_p^{\alpha_i \alpha_j}$. Compute $B_{i^*,j} \leftarrow A_{i^*}^{\alpha_j}$ and $B_{i,i^*} \leftarrow A_{i^*}^{\alpha_i}$ for all $i, j \neq i^*$.
  4. Output the common reference string $\mathsf{crs}^* = \left(\mathcal{G}, A, \{A_i\}_{i\in[m]}, \{B_{i,j}\}_{i\neq j}\right)$ and the trapdoor $\mathsf{td} = g_q \leftarrow g^p$.
- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$: On input the trapdoor $\mathsf{td} = g_q$, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and the proof $\pi = \left(\{U_k, V_k\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]}\right)$, the extraction algorithm sets $w_k^* = 0$ if $e(g_q, U_k) = 1$ and $w_k^* = 1$ otherwise for each $k = n+1, \ldots, n+h$. It outputs $\mathbf{w}^* = (w_{n+1}^*, \ldots, w_{n+h}^*)$.

We now show the CRS indistinguishability and somewhere extractable in trapdoor mode properties.

Lemma 3.6 (CRS Indistinguishability)**.** *If the subgroup decision assumption holds with respect to* CompGroupGen*, then Construction 3.3 satisfies CRS indistinguishability.*

Proof. Take any polynomial $m = m(\lambda), s = s(\lambda)$. We proceed via a hybrid argument:

- Hyb_0: This is the real distribution. At the beginning of the security game, the adversary chooses an index $i^* \in [m]$. The challenger then constructs the common reference string by running $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$:
  - Run $(\mathbb{G}, \mathbb{G}_T, p, q, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and let $N = pq$, $g_p \leftarrow g^q$. Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_p, e)$.
  - For each $i \in [m]$, sample $\alpha_i \xleftarrow{R} \mathbb{Z}_N$. For each $i \in [m]$, let $A_i \leftarrow g_p^{\alpha_i}$. Let $A \leftarrow \prod_{i\in[m]} A_i$.
  - For each $i, j \in [m]$ where $i \neq j$, compute $B_{i,j} \leftarrow g_p^{\alpha_i\alpha_j}$.
  - Output the common reference string $\mathsf{crs} = \left(\mathcal{G}, A, \{A_i\}_{i\in[m]}, \{B_{i,j}\}_{i\neq j}\right)$.

  The challenger gives crs to $\mathcal{A}$ and $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_1: Same as Hyb_0 except the challenger constructs A and $B_{i,j}$ using the procedure from TrapSetup:
  - For each $i \in [m]$, sample $\alpha_i \xleftarrow{R} \mathbb{Z}_N$. For each $i \in [m]$, let $A_i \leftarrow g_p^{\alpha_i}$. Let $A \leftarrow A_{i^*} \prod_{i\neq i^*} A_i$.
  - For each $i, j \in [m]$ where $i \neq j$ and $i, j \neq i^*$, compute $B_{i,j} \leftarrow g_p^{\alpha_i\alpha_j}$. Compute $B_{i^*,j} \leftarrow A_{i^*}^{\alpha_j}$ and $B_{i,i^*} \leftarrow A_{i^*}^{\alpha_i}$ for all $i, j \neq i^*$.
- Hyb_2: Same as Hyb_1 except the challenger samples $A_{i^*} \leftarrow g^{\alpha_{i^*}}$:
  - For each $i \in [m]$, sample $\alpha_i \xleftarrow{R} \mathbb{Z}_N$. For each $i \neq i^*$, let $A_i \leftarrow g_p^{\alpha_i}$. Let $A_{i^*} \leftarrow g^{\alpha_{i^*}}$. Let $A \leftarrow A_{i^*} \prod_{i\neq i^*} A_i$.
  - For each $i, j \in [m]$ where $i \neq j$ and $i, j \neq i^*$, compute $B_{i,j} \leftarrow g_p^{\alpha_i\alpha_j}$. Compute $B_{i^*,j} \leftarrow A_{i^*}^{\alpha_j}$ and $B_{i,i^*} \leftarrow A_{i^*}^{\alpha_i}$ for all $i, j \neq i^*$.

  In this experiment, crs is distributed according to $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$.

For an index i, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of experiment Hyb_i with algorithm $\mathcal{A}$. We show that the output distributions each adjacent pair of experiments are computationally indistinguishable (or identical).

Claim 3.7. *For all adversaries* $\mathcal{A}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

Proof. The difference between Hyb_0 and Hyb_1 is purely syntactic. In Hyb_1, $A_i = A_{i^*} \prod_{i\neq i} A_i = \prod_{i\in[m]} A_i$, which matches the distribution in Hyb_0. Similarly, in Hyb_1,

$$B_{i^*,j} = A_{i^*}^{\alpha_j} = g^{\alpha_{i^*}\alpha_j} \quad \text{and} \quad B_{i,i^*} = A_{i^*}^{\alpha_i} = g^{\alpha_{i^*}\alpha_i},$$

which is precisely the distribution of $B_{i^*,j}$ and B_{i,i^*} in Hyb_0 for all $i, j \neq i^*$. Finally $B_{i,j}$ for $i \neq j$ and $i, j \neq i^*$ are identically distributed in the two experiments. □

Claim 3.8. *Suppose the subgroup decision assumption holds with respect to* GroupGen. *Then, for all efficient adversaries* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

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

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the group description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_p, e)$ and the challenge $Z \in \mathbb{G}$ from the subgroup decision challenger.
2. For $i \neq i^*$, algorithm $\mathcal{B}$ samples $\alpha_i \xleftarrow{R} \mathbb{Z}_N$ and sets $A_i \leftarrow g_p^{\alpha_i}$. It sets $A_{i^*} \leftarrow Z$ to be the challenge value. Next, it computes $A \leftarrow Z \prod_{i\neq i^*} A_i$. For $i \neq j$ and $i, j \neq i^*$, algorithm $\mathcal{B}$ computes $B_{i,j} \leftarrow g_p^{\alpha_i\alpha_j}$. For $i, j \neq i^*$, it computes $B_{i^*,j} \leftarrow Z^{\alpha_j}$ and $B_{i,i^*} \leftarrow Z^{\alpha_i}$.
3. Algorithm $\mathcal{B}$ gives $\mathsf{crs} = \left(\mathcal{G}, A, \{A_i\}_{i\in[m]}, \{B_{i,j}\}_{i\neq j}\right)$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

Consider now the two possibilities:

- Suppose $Z = g_p^r$ in the subgroup decision game. Then, $A_{i^*} = g_p^r$ and algorithm $\mathcal{B}$ perfectly simulates the distribution in Hyb_1. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.
- Suppose $Z = g^r$ in the subgroup decision game. Then, $A_{i^*} = g^r$ and algorithm $\mathcal{B}$ perfectly simulates the distribution in Hyb_2. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

The advantage of $\mathcal{B}$ in the subgroup decision game is thus $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \varepsilon$. □

Combining Claims 3.7 and 3.8, CRS indistinguishability holds. □

Lemma 3.9 (Somewhere Extractable in Trapdoor Mode)**.** *Construction 3.3 is somewhere extractable in trapdoor mode.*

Proof. Fix polynomials $m = m(\lambda)$ and $s = s(\lambda)$. Let $i^* \leftarrow \mathcal{A}(1^\lambda, 1^m, 1^s)$ and $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$. By construction,

$$\mathsf{crs}^* = (\mathcal{G}, A, \{A_i\}_{i\in[m]}, \{B_{i,j}\}_{i\neq j}) \quad \text{and} \quad \mathsf{td} = g_q,$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_p, e)$. Let $N = pq$ and g be the generator of $\mathbb{G}$ (i.e., $g_p := g^q$ and $g_q := g^p$). Let $\mathbb{G}_p = \langle g_p \rangle$ be the order-p subgroup of $\mathbb{G}$ generated by g_p. Correspondingly, let $\mathbb{G}_q = \langle g_q \rangle$ be the order-q subgroup of $\mathbb{G}$ generated by g_q. By the Chinese Remainder Theorem, $\mathbb{G} \cong \mathbb{G}_p \times \mathbb{G}_q$.

Let $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be the Boolean circuit, $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ be the statements, and $\pi = \left(\{U_k, V_k\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]}\right)$ be the proof the adversary outputs. Suppose $\mathsf{Verify}(\mathsf{crs}^*, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 1$. By construction of $\mathsf{TrapSetup}$, we can write $A_{i^*} = g^{\alpha_{i^*}} = g_p^{\alpha_{i^*,p}} g_q^{\alpha_{i^*,q}}$ for some $\alpha_{i^*,p} \in \mathbb{Z}_p$ and $\alpha_{i^*,q} \in \mathbb{Z}_q$. Suppose that $\alpha_{i^*,q} \neq 0$. This holds with overwhelming probability since $\alpha_{i^*} \xleftarrow{R} \mathbb{Z}_N$. Now the following properties hold:

- For all $k \in [t]$, either $U_k \in \mathbb{G}_p$ or $U_k / g_q^{\alpha_{i^*,q}} \in \mathbb{G}_p$. This follows from the wire validity checks. Specifically, suppose $U_k = g_p^{\beta_p} g_q^{\beta_q}$. We can also write $A = g_p^{\sum_{i\in[m]} \alpha_i} g_q^{\alpha_{i^*,q}}$. Since verification succeeds, it must be the case that

  $$e(A, U_k) = e(g_p, V_k) e(U_k, U_k).$$

  Consider the projection in the order-q subgroup of $\mathbb{G}_T$. This relation requires that $\alpha_{i^*,q} \cdot \beta_q = \beta_q^2$. This means that either $\beta_q = 0$ (in which case $U_k \in \mathbb{G}_p$) or $\beta_q = \alpha_{i^*,q}$ (in which case $U_k / g_q^{\alpha_{i^*,q}} \in \mathbb{G}_p$).

- For each $k \in [t]$, if $U_k \in \mathbb{G}_p$, then set $\xi_k = 0$. If $U_k / g_q^{\alpha_{i^*,q}} \in \mathbb{G}_p$, then set $\xi_k = 1$. Then, for all gates $G_\ell = (k_1, k_2, k_3) \in [t]^3$ in the circuit, $\xi_{k_3} = \mathsf{NAND}(\xi_{k_1}, \xi_{k_2})$. This follows from the gate validity checks. In particular, if verification succeeds, then Eq. (3.2) holds. From the above analysis, we can write $U_k = g_p^{\beta_{k,p}} g_q^{\xi_k \alpha_{i^*,q}}$ for all $k \in [t]$ and some $\beta_{k,p} \in \mathbb{Z}_p$. Consider the projection of Eq. (3.2) into the order-q subgroup of $\mathbb{G}_T$. This yields the relation

  $$\alpha_{i^*,q}^2 = (\xi_{k_1} \alpha_{i^*,q})(\xi_{k_2} \alpha_{i^*,q}) + \alpha_{i^*,q}(\xi_{k_3} \alpha_{i^*,q}) = \alpha_{i^*,q}^2 (\xi_{k_1} \xi_{k_2} + \xi_{k_3}).$$

  Since $\alpha_{i^*,q} \neq 0$, this means that $1 = \xi_{k_1} \xi_{k_2} + \xi_{k_3}$, or equivalently, $\xi_{k_3} = 1 - \xi_{k_1} \xi_{k_2} = \mathsf{NAND}(\xi_{k_1}, \xi_{k_2})$.

- Let $\mathbf{x}_{i^*} = (x_{i^*,1}, \ldots, x_{i^*,n})$. For $k \in [n]$, $\xi_k = x_{i^*,k}$.

  This follows from the statement validity check. Namely, for $k \in [n]$, the verifier checks that $U_k = A_{i^*}^{x_{i^*,k}} \prod_{i\neq i^*} A_i^{x_{i,k}}$. Since $A_i \in \mathbb{G}_p$ for $i \neq i^*$, it follows that if $x_{i^*,k} = 0$, then $U_k \in \mathbb{G}_p$ (and $\xi_k = 0 = x_{i^*,k}$). Otherwise, if $x_{i^*,k} = 1$, then the component of U_k in $\mathbb{G}_q$ is exactly $g_q^{\alpha_{i^*,q}}$, in which case $\xi_k = 1 = x_{i^*,k}$.

- Finally $\xi_t = 1$. This follows from the output satisfiability check. Namely, the verifier checks that $U_t = A = g_p^{\sum_{i\in[m]} \alpha_i} g_q^{\alpha_{i^*,q}}$. If the verifier accepts, then this relation holds and $\xi_t = 1$.

The above properties show that $\xi_1, \ldots, \xi_t$ is a valid assignment to the wires of C on input $\mathbf{x}_{i^*}$ and witness $\boldsymbol{\xi} = (\xi_{n+1}, \ldots, \xi_{n+h})$. Moreover, $C(\mathbf{x}_{i^*}, \boldsymbol{\xi}) = \xi_t = 1$.

To complete the proof, let $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$. We claim that $\mathbf{w}^* = \boldsymbol{\xi}$. In particular, for $k \in [h]$, if $U_{n+k} \in \mathbb{G}_p$, then $e(g_q, U_k) = 1$ and $w_k^* = 0 = \xi_{n+k}$. Alternatively, if $U_{n+k} / g_p^{\alpha_{i^*,q}} \in \mathbb{G}_p$, then $e(g_q, U_k) = e(g_q, g_q)^{\alpha_{i^*,q}} \neq 1$, so $w_k^* = 1 = \xi_{n+k}$. Thus, with probability $1 - \mathsf{negl}(\lambda)$, either $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 0$ or $C(\mathbf{x}, \mathbf{w}^*) = 1$. □

By Lemmas 3.6 and 3.9, Construction 3.3 is a somewhere argument of knowledge. □

Theorem 3.10 (Succinctness)**.** *Construction 3.3 is succinct and satisfies split verification (Definition 2.9).*

Proof. Take any $\lambda, m, s \in \mathbb{N}$ and consider a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s. Let $t = \mathsf{poly}(s)$ be the number of wires in C. We check each property:

- **Proof size:** A proof π consists of $2t + s$ elements in $\mathbb{G}$, each of which can be represented in $\mathsf{poly}(\lambda)$ bits. Thus, the proof size satisfies $|\pi| = (2t + s) \cdot \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda, s)$
- **CRS size:** The common reference string crs consists of the group description $\mathcal{G}$, and $m + 1 + m(m-1)/2$ elements in $\mathbb{G}$. Thus, $|\mathsf{crs}| = m^2 \cdot \mathsf{poly}(\lambda)$.
- **Verification key size:** The size of the verification key vk output by GenVK consists of n group elements. Thus, $|\mathsf{vk}| = n \cdot \mathsf{poly}(\lambda)$.
- **Verification key generation time:** The algorithm GenVK performs nm group operations. This takes time $\mathsf{poly}(\lambda, m, n)$.
- **Online verification time:** The running time of the online verification algorithm OnlineVerify is

$$\underbrace{n \cdot \mathsf{poly}(\lambda)}_{\text{statement validity}} + \underbrace{t \cdot \mathsf{poly}(\lambda)}_{\text{wire validity}} + \underbrace{s \cdot \mathsf{poly}(\lambda)}_{\text{gate validity}} + \underbrace{\mathsf{poly}(\lambda)}_{\text{output validity}} = \mathsf{poly}(\lambda, s),$$

since $n, t = \mathsf{poly}(s)$. □

Remark 3.11 (Variable Number of Instances)**.** As currently described, the prover and verifier algorithms in Construction 3.3 takes exactly m instances as input. However, the same scheme can also be used to prove any $T \le m$ instances (by ignoring components in the CRS). In this case, the proof size is unchanged, and the verification running time (assuming random read access to the CRS) is $\mathsf{poly}(\lambda, n, T) + \mathsf{poly}(\lambda, s)$.

4 BARG for NP from k-Lin in Bilinear Groups

In this section, we show how to translate the ideas underlying Construction 3.3 to work with asymmetric prime-order groups under the k-Lin assumption. We start by recalling the definition of a prime-order pairing group and the matrix Diffie-Hellman (MDDH) assumption [EHK+13].

Definition 4.1 (Prime-Order Bilinear Group)**.** A prime-order asymmetric group generator GroupGen is an efficient algorithm that takes as input the security parameter 1^λ and outputs a description $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ of two base groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with generators g_1, g_2, respectively, a target group $\mathbb{G}_T$, all of prime order $p = 2^{\Theta(\lambda)}$, and a non-degenerate bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We require that the group operation in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and the pairing operations to be efficiently computable.

Notation. When working with an asymmetric prime-order pairing group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$, we use the implicit representation of group elements [EHK+13]. Specifically, for a matrix $\mathbf{M}$ over $\mathbb{Z}_p$, we write $[\mathbf{M}]_1 := g_1^{\mathbf{M}}$, $[\mathbf{M}]_2 := g_2^{\mathbf{M}}$, and $[\mathbf{M}]_T := g_T^{\mathbf{M}}$, where exponentiation is defined component-wise and $g_T = e(g_1, g_2)$. Given matrices $\mathbf{A}$ and $\mathbf{B}$ over $\mathbb{Z}_p$, we define the pairing operation $e([\mathbf{A}]_1, [\mathbf{B}]_2) := [\mathbf{AB}]_T$. We also denote this by writing $[\mathbf{A}]_1 \cdot [\mathbf{B}]_2 := e([\mathbf{A}]_1, [\mathbf{B}]_2)$. For matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ over $\mathbb{Z}_p$, we write $\mathbf{A}[\mathbf{B}]_1 + [\mathbf{C}]_1\mathbf{D} := [\mathbf{AB} + \mathbf{CD}]_1$ to represent linear operations within $\mathbb{G}_1$ (and analogously in $\mathbb{G}_2$ and $\mathbb{G}_T$). We now recall the k-Lin and matrix Diffie-Hellman assumptions. In the case of k-Lin, recall that the case of $k = 1$ corresponds to the decisional Diffie-Hellman (DDH) assumption and the case $k = 2$ corresponds to the decisional linear (DLIN) assumption [BBS04, HK07, Sha07]. Finally, the symmetric external Diffie-Hellman (SXDH) assumption corresponds to DDH (i.e., 1-Lin) holding in *both* $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 4.2 (k-Lin Assumption [BBS04, HK07, Sha07])**.** Let $k \in \mathbb{N}$. The k-Lin assumption holds in $\mathbb{G}_1$ with respect to GroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$|\Pr[\mathcal{A}(\mathcal{G}, [\mathbf{M}]_1, [\mathbf{Mv}]_1) = 1] - \Pr[\mathcal{A}(\mathcal{G}, [\mathbf{M}]_1, [\mathbf{u}]_1) = 1]| = \mathsf{negl}(\lambda),$$

where $\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$,

$$\mathbf{M} = \begin{bmatrix} \mathsf{diag}(\mathbf{s}) \\ \hline \mathbf{1}^\mathsf{T} \end{bmatrix} \in \mathbb{Z}_p^{(k+1)\times k},$$

$\mathbf{s} = (s_1, \ldots, s_k) \xleftarrow{R} \mathbb{Z}_p^k$, $\mathsf{diag}(\mathbf{s}) \in \mathbb{Z}_p^{k\times k}$ is the diagonal matrix whose entries are $s_1, \ldots, s_k$, $\mathbf{v} \xleftarrow{R} \mathbb{Z}_p^k$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_p^{k+1}$. We define the k-Lin assumption in $\mathbb{G}_2$ with respect to GroupGen in an analogous manner.

Definition 4.3 (Matrix Diffie-Hellman Assumption [EHK+13])**.** Let $k \in \mathbb{N}$. The MDDH_k assumption holds in $\mathbb{G}_1$ with respect to GroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$|\Pr[\mathcal{A}(\mathcal{G}, [\mathbf{M}]_1, [\mathbf{Mv}]_1) = 1] - \Pr[\mathcal{A}(\mathcal{G}, [\mathbf{M}]_1, [\mathbf{u}]_1) = 1]| = \mathsf{negl}(\lambda),$$

where $\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$, $\mathbf{M} \xleftarrow{R} \mathbb{Z}_p^{(k+1)\times k}$, $\mathbf{v} \xleftarrow{R} \mathbb{Z}_p^k$ and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_p^{k+1}$. We define the MDDH_k assumption in $\mathbb{G}_2$ with respect to GroupGen in an analogous manner.

Theorem 4.4 (Matrix Diffie-Hellman [EHK+13])**.** *Let $k \in \mathbb{N}$. Suppose the k-Lin assumption holds in $\mathbb{G}_1$ (resp., $\mathbb{G}_2$) with respect to* GroupGen*. Then MDDH_k holds in $\mathbb{G}_1$ (resp., $\mathbb{G}_2$) with respect to* GroupGen*.*

Construction overview. Our BARG from asymmetric prime-order groups relies on a similar underlying principle as the construction from symmetric composite-order groups (Construction 3.3). Here, we summarize the key differences:

- **Randomizing cross-terms in the CRS.** In the symmetric setting, we associated a single encoding A_i with each instance. In the asymmetric setting, we need to encode the instance in *both* $\mathbb{G}_1$ and $\mathbb{G}_2$ in order to apply the pairing consistency checks. Thus, the prover now generates two commitments to the wire labels for each wire, one in $\mathbb{G}_1$ and the other in $\mathbb{G}_2$. This introduces a new challenge when it comes to constructing the *cross-terms* $B_{i,j}$, as it depends on the exponents associated with the encodings in *both* $\mathbb{G}_1$ and $\mathbb{G}_2$. Proving security would seemingly need to rely on a "bilateral" assumption over pairing groups where the assumption gives out elements with correlated exponents in *both* $\mathbb{G}_1$ and $\mathbb{G}_2$. To avoid this and base security on the vanilla k-Lin assumption, we split the cross-terms into two shares, with one share in $\mathbb{G}_1$ and the other in $\mathbb{G}_2$. The extra randomness in the cross terms allows for a simple simulation strategy in the security analysis (see Lemma 4.8).
- **Simulating projective pairing using outer products.** The key property we relied on in the soundness analysis of the composite-order construction is that the pairing is projecting. Namely, there exists a projection map on $\mathbb{G}$ and $\mathbb{G}_T$ that map into the subgroup of order-q in each respective group; moreover, this projection map *commutes* with the pairing. Then, if a relation like Eq. (3.1) or Eq. (3.2) holds in the target group, the projected relation formed by projecting the left-hand and right-hand sides into the order-q subgroup also holds. As argued in Lemma 3.9, projecting into the order-q subgroup allows us to isolate a single instance i^*, in which case the verification checks ensure *statistically* soundness for instance i^*. To obtain an analog of projective pairings in the prime order setting, we can replace the subgroups with subspaces of a vector space and define the pairing operation to be an outer (tensor) product of vectors [GS08, Fre10]. As we show in Lemma 4.12, this enables a similar strategy to prove soundness.

Construction 4.5 (BARG for NP from k-Lin)**.** Let $k \in \mathbb{N}$ be an integer. We construct a BARG with split verification for the language of circuit satisfiability as follows:

- $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$: On input the security parameter λ, the number of instances m, and the bound on the circuit size s, the setup algorithm does the following:
  - Run $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample matrices $\mathbf{M}, \hat{\mathbf{M}} \xleftarrow{R} \mathbb{Z}_p^{(k+1)\times k}$.

– For each $i \in [m]$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$ and compute $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{a} \leftarrow \sum_{i\in[m]} \mathbf{a}_i$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]} \hat{\mathbf{a}}_i$.

– For each $i, j \in [m]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$ and let $\mathbf{B}_{i,j} \leftarrow \mathbf{M}(\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_j^\top + \mathbf{R}_{i,j}) \in \mathbb{Z}_p^{(k+1)\times k}$ and $\hat{\mathbf{B}}_{i,j} \leftarrow -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top \in \mathbb{Z}_p^{(k+1)\times k}$.

– Output the common reference string $\mathsf{crs} = \big(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\big)$.

- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$: On input the common reference string

$$\mathsf{crs} = \Big(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\Big),$$

the circuit $C\colon \{0,1\}^n \to \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$, define t to be the number of wires in C and s to be the number of gates in C. Then, for $i \in [m]$ and $j \in [t]$, let $w_{i,j} \in \{0,1\}$ be the value of wire j in $C(\mathbf{x}_i, \mathbf{w}_i)$. The prover then proceeds as follows:

– **Encoding the wire values:** For each wire $d \in [t]$, let

$$[\mathbf{u}_d]_1 \leftarrow \sum_{i\in[m]} w_{i,d}[\mathbf{a}_i]_1 \quad \text{and} \quad [\hat{\mathbf{u}}_d]_2 \leftarrow \sum_{i\in[m]} w_{i,d}[\hat{\mathbf{a}}_i]_2.$$

– **Validity of witness wires:** For each $d \in \{n+1, \ldots, n+h\}$, compute

$$[\mathbf{V}_{d,1}]_1 = \sum_{i\neq j}(1 - w_{i,d})w_{j,d}[\mathbf{B}_{i,j}]_1 \quad \text{and} \quad [\hat{\mathbf{V}}_{d,1}]_2 = \sum_{i\neq j}(1 - w_{i,d})w_{j,d}[\hat{\mathbf{B}}_{i,j}]_2,$$

as well as

$$[\mathbf{V}_{d,2}]_1 = \sum_{i\neq j}(1 - w_{j,d})w_{i,d}[\mathbf{B}_{i,j}]_1 \quad \text{and} \quad [\hat{\mathbf{V}}_{d,2}]_2 = \sum_{i\neq j}(1 - w_{j,d})w_{i,d}[\hat{\mathbf{B}}_{i,j}]_2,$$

– **Validity of gate computation:** For each NAND gate $G_\ell = (d_1, d_2, d_3) \in [t]^3$ (where $\ell \in [s]$), compute

$$[\mathbf{W}_{\ell,1}]_1 = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{j,d_3})[\mathbf{B}_{i,j}]_1 \quad \text{and} \quad [\hat{\mathbf{W}}_{\ell,1}]_2 = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{j,d_3})[\hat{\mathbf{B}}_{i,j}]_2$$

as well as

$$[\mathbf{W}_{\ell,2}]_1 = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{i,d_3})[\mathbf{B}_{i,j}]_1 \quad \text{and} \quad [\hat{\mathbf{W}}_{\ell,2}]_2 = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{i,d_3})[\hat{\mathbf{B}}_{i,j}]_2$$

Finally, output the proof

$$\pi = \big(\{[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2\}_{d\in[t]}, \{[\mathbf{V}_{n+d,i}]_1, [\hat{\mathbf{V}}_{n+d,i}]_2\}_{d\in[h],i\in\{1,2\}}, \{[\mathbf{W}_{\ell,i}]_1, [\hat{\mathbf{W}}_{\ell,i}]_2\}_{\ell\in[s],i\in\{1,2\}}\big).$$

- $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$: We decompose the verification algorithm into $(\mathsf{GenVK}, \mathsf{OnlineVerify})$:

– $\mathsf{GenVK}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_m))$: On input the common reference string

$$\mathsf{crs} = \Big(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\Big)$$

and instances $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, the verification key generation algorithm computes

$$[\mathbf{u}_d^*]_1 = \sum_{i\in[m]} x_{i,d}[\mathbf{a}_i]_1 \quad \text{and} \quad [\hat{\mathbf{u}}_d^*]_2 = \sum_{i\in[m]} x_{i,d}[\hat{\mathbf{a}}_i]_2.$$

for each $d \in [n]$ and outputs the verification key $\mathsf{vk} = \big\{[\mathbf{u}_d^*]_1, [\hat{\mathbf{u}}_d^*]_2\big\}_{d\in[n]}$.

– $\mathsf{OnlineVerify}(\mathsf{vk}, C, \pi)$: On input the verification key $\mathsf{vk} = \left\{[\mathbf{u}_d^*]_1, [\hat{\mathbf{u}}_d^*]_2\right\}_{d\in[n]}$, the circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and the proof

$$\pi = \left(\{[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2\}_{d\in[t]}, \{[\mathbf{V}_{n+d,i}]_1, [\hat{\mathbf{V}}_{n+d,i}]_2\}_{d\in[h],i\in\{1,2\}}, \{[\mathbf{W}_{\ell,i}]_1, [\hat{\mathbf{W}}_{\ell,i}]_2\}_{\ell\in[s],i\in\{1,2\}}\right),$$

the verification algorithm checks the following:

* **Validity of statement:** For each statement wire $d \in [n]$, check that $[\mathbf{u}_d]_1 = [\mathbf{u}_d^*]_1$ and $[\hat{\mathbf{u}}_d]_2 = [\hat{\mathbf{u}}_d^*]_2$.
* **Validity of witness wires:** For each witness wire $d \in \{n+1, \ldots, n+h\}$, check that

$$[\mathbf{a}]_1 \cdot [\hat{\mathbf{u}}_d^\mathsf{T}]_2 = \left([\mathbf{u}_d]_1 \cdot [\hat{\mathbf{u}}_d^\mathsf{T}]_2\right) + \left([\mathbf{M}]_1 \cdot [\hat{\mathbf{V}}_{d,1}^\mathsf{T}]_2\right) + \left([\mathbf{V}_{d,1}]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2\right)$$

and that

$$[\mathbf{u}_d]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 = \left([\mathbf{u}_d]_1 \cdot [\hat{\mathbf{u}}_d^\mathsf{T}]_2\right) + \left([\mathbf{M}]_1 \cdot [\hat{\mathbf{V}}_{d,2}^\mathsf{T}]_2\right) + \left([\mathbf{V}_{d,2}]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2.\right)$$

* **Validity of gate computation:** For each gate $G_\ell = (d_1, d_2, d_3) \in [t]^3$, check that

$$[\mathbf{a}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 = \left([\mathbf{u}_{d_1}]_1 \cdot [\hat{\mathbf{u}}_{d_2}^\mathsf{T}]_2\right) + \left([\mathbf{a}]_1 \cdot [\hat{\mathbf{u}}_{d_3}^\mathsf{T}]_2\right) + \left([\mathbf{M}]_1 \cdot [\hat{\mathbf{W}}_{\ell,1}^\mathsf{T}]_2\right) + \left([\mathbf{W}_{\ell,1}]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2\right),$$

and that

$$[\mathbf{a}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 = \left([\mathbf{u}_{d_1}]_1 \cdot [\hat{\mathbf{u}}_{d_2}^\mathsf{T}]_2\right) + \left([\mathbf{u}_{d_3}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2\right) + \left([\mathbf{M}]_1 \cdot [\hat{\mathbf{W}}_{\ell,2}^\mathsf{T}]_2\right) + \left([\mathbf{W}_{\ell,2}]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2\right).$$

* **Output satisfiability:** Finally, the verifier checks that $[\mathbf{u}_t]_1 = [\mathbf{a}]_1$ and $[\hat{\mathbf{u}}_t]_2 = [\hat{\mathbf{a}}]_2$.

Theorem 4.6 (Completeness). *Construction 4.5 is complete.*

Proof. Take any $\lambda, m, s \in \mathbb{N}$, and let $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be a Boolean circuit of size at most s. Take statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$ where $C(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all $i \in [m]$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$, where

$$\mathsf{crs} = \left(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right)$$
$$\pi = \left(\{[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2\}_{d\in[t]}, \{[\mathbf{V}_{n+d,i}]_1, [\hat{\mathbf{V}}_{n+d,i}]_2\}_{d\in[h],i\in\{1,2\}}, \{[\mathbf{W}_{\ell,i}]_1, [\hat{\mathbf{W}}_{\ell,i}]_2\}_{\ell\in[s],i\in\{1,2\}}\right)$$

For $i \in [m]$ and $j \in [t]$, let $w_{i,j} \in \{0,1\}$ denote the value of wire j in $C(\mathbf{x}_i, \mathbf{w}_i)$. First, observe that for all $i \neq j$,

$$\mathbf{M}\hat{\mathbf{B}}_{i,j}^\mathsf{T} + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\mathsf{T} = -\mathbf{M}\mathbf{R}_{i,j}\hat{\mathbf{M}}^\mathsf{T} + \mathbf{M}(\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{R}_{i,j})\hat{\mathbf{M}}^\mathsf{T} = \mathbf{M}\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T}\hat{\mathbf{M}}^\mathsf{T} = \mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T}. \tag{4.1}$$

We show that each of the verification checks pass:

- **Validity of statement:** The honest prover computes $\mathbf{u}_d = \sum_{i\in[m]} w_{i,d}\mathbf{a}_i$ for all $d \in [t]$. Since the first n wires of the circuit corresponds to the statement, we have $w_{i,d} = x_{i,d}$ for all $d \in [n]$ and the check passes. Similarly, $\hat{\mathbf{u}}_d = \sum_{i\in[m]} w_{i,d}\hat{\mathbf{a}}_i = \sum_{i\in[m]} x_{i,d}\hat{\mathbf{a}}_i$.
- **Validity of witness wires:** By construction of $\mathbf{V}_{d,1}$, $\hat{\mathbf{V}}_{d,1}$ and appealing to Eq. (4.1),

$$\mathbf{M}\hat{\mathbf{V}}_{d,1}^\mathsf{T} + \mathbf{V}_{d,1}\hat{\mathbf{M}}^\mathsf{T} = \sum_{i\neq j}(1 - w_{i,d})w_{j,d}(\mathbf{M}\hat{\mathbf{B}}_{i,j}^\mathsf{T} + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\mathsf{T}) = \sum_{i\neq j}(w_{j,d} - w_{i,d}w_{j,d})\mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T}.$$

Similarly, by construction of $\mathbf{u}_d$, $\hat{\mathbf{u}}_d$, and $\mathbf{a}$, we can write

$$\mathbf{u}_d\hat{\mathbf{u}}_d^\mathsf{T} = \sum_{i,j\in[m]} w_{i,d}w_{j,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T} = \sum_{i\in[m]} w_{i,d}^2\mathbf{a}_i\hat{\mathbf{a}}_i^\mathsf{T} + \sum_{i\neq j} w_{i,d}w_{j,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T}$$
$$\mathbf{a}\hat{\mathbf{u}}_d^\mathsf{T} = \sum_{i,j\in[m]} w_{j,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T} = \sum_{i\in[m]} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_i^\mathsf{T} + \sum_{i\neq j} w_{j,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T}$$

Since $w_{i,d} \in \{0,1\}$, we have that $w_{i,d}^2 = w_{i,d}$. Combining the above relations,

$$\mathbf{u}_d \hat{\mathbf{u}}_d^\top + \mathbf{M}\hat{\mathbf{V}}_{d,1}^\top + \mathbf{V}_{d,1}\hat{\mathbf{M}}^\top = \sum_{i\in[m]} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{j,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \mathbf{a}\hat{\mathbf{u}}_d^\top,$$

and the first verification check passes. Validity of the second verification check follows by an analogous calculation. Namely,

$$\mathbf{M}\hat{\mathbf{V}}_{d,2}^\top + \mathbf{V}_{d,2}\hat{\mathbf{M}}^\top = \sum_{i\neq j}(1-w_{j,d})w_{i,d}(\mathbf{M}\hat{\mathbf{B}}_{i,j}^\top + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\top) = \sum_{i\neq j}(w_{i,d} - w_{i,d}w_{j,d})\mathbf{a}_i\hat{\mathbf{a}}_j^\top$$
$$\mathbf{u}_d\hat{\mathbf{a}}^\top = \sum_{i,j\in[m]} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\top,$$

from which we can conclude that

$$\mathbf{u}_d\hat{\mathbf{u}}_d^\top + \mathbf{M}\hat{\mathbf{V}}_{d,2}^\top + \mathbf{V}_{d,2}\hat{\mathbf{M}}^\top = \sum_{i\in[m]} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,d}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \mathbf{u}_d\hat{\mathbf{a}}^\top.$$

- **Validity of gate computation:** Similar to the previous case, we expand each term in the verification relation and apply Eq. (4.1) to obtain

$$\mathbf{M}\hat{\mathbf{W}}_{\ell,1}^\top + \mathbf{W}_{\ell,1}\hat{\mathbf{M}}^\top = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{j,d_3})(\mathbf{M}\hat{\mathbf{B}}_{i,j}^\top + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\top) = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{j,d_3})\mathbf{a}_i\hat{\mathbf{a}}_j^\top$$
$$\mathbf{u}_{d_1}\hat{\mathbf{u}}_{d_2}^\top = \sum_{i,j\in[m]} w_{i,d_1}w_{j,d_2}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,d_1}w_{i,d_2}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,d_1}w_{j,d_2}\mathbf{a}_i\hat{\mathbf{a}}_j^\top$$
$$\mathbf{a}\hat{\mathbf{u}}_{d_3}^\top = \sum_{i,j\in[m]} w_{j,d_3}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,d_3}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{j,d_3}\mathbf{a}_i\hat{\mathbf{a}}_j^\top$$
$$\mathbf{a}\hat{\mathbf{a}}^\top = \sum_{i,j\in[m]} \mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} \mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} \mathbf{a}_i\hat{\mathbf{a}}_j^\top.$$

By definition, $w_{i,d_3} = \mathsf{NAND}(w_{i,d_1}, w_{i,d_2})$ for all $i \in [m]$. In particular, this means that $w_{i,d_3} = 1 - w_{i,d_1}w_{i,d_2}$, or equivalently, $w_{i,d_1}w_{i,d_2} + w_{i,d_3} = 1$. Substituting into the above relations,

$$\mathbf{u}_{d_1}\hat{\mathbf{u}}_{d_2}^\top + \mathbf{a}\hat{\mathbf{u}}_{d_3}^\top + \mathbf{M}\hat{\mathbf{W}}_{\ell,1}^\top + \mathbf{W}_{\ell,1}\hat{\mathbf{M}}^\top = \sum_{i\in[m]} \mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} \mathbf{a}_i\hat{\mathbf{a}}_j^\top = \mathbf{a}\hat{\mathbf{a}}^\top.$$

For the second validation check, we expand as above to obtain

$$\mathbf{M}\hat{\mathbf{W}}_{\ell,2}^\top + \mathbf{W}_{\ell,2}\hat{\mathbf{M}}^\top = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{i,d_3})(\mathbf{M}\hat{\mathbf{B}}_{i,j}^\top + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\top) = \sum_{i\neq j}(1 - w_{i,d_1}w_{j,d_2} - w_{i,d_3})\mathbf{a}_i\hat{\mathbf{a}}_j^\top$$
$$\mathbf{u}_{d_3}\hat{\mathbf{a}}^\top = \sum_{i,j\in[m]} w_{i,d_3}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,d_3}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,d_3}\mathbf{a}_i\hat{\mathbf{a}}_j^\top.$$

Combining the relations, we see that

$$\mathbf{u}_{d_1}\hat{\mathbf{u}}_{d_2}^\top + \mathbf{u}_{d_3}\hat{\mathbf{a}}^\top + \mathbf{M}\hat{\mathbf{W}}_{\ell,2}^\top + \mathbf{W}_{\ell,2}\hat{\mathbf{M}}^\top = \sum_{i\in[m]} \mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} \mathbf{a}_i\hat{\mathbf{a}}_j^\top = \mathbf{a}\hat{\mathbf{a}}^\top.$$

- **Validity of output:** Since $C(\mathbf{x}_i, \mathbf{w}_i) = 1$, it follows that $w_{i,t} = 1$ for all $i \in [m]$. This means that $\mathbf{u}_t = \sum_{i\in[m]} \mathbf{a}_i = \mathbf{a}$ and $\hat{\mathbf{u}}_t = \sum_{i\in[m]} \hat{\mathbf{a}}_i = \hat{\mathbf{a}}$. □

Theorem 4.7 (Somewhere Argument of Knowledge)**.** *Take any positive integer $k \in \mathbb{N}$. If the k-*Lin *assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen*, then Construction 4.5 is a somewhere argument of knowledge.*

Proof. We start by defining the trapdoor setup and extraction algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$: The trapdoor algorithm uses the following procedure (we highlight in green the differences in the common reference string between TrapSetup and Setup):
  - Run $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample matrices $\mathbf{M}, \hat{\mathbf{M}} \xleftarrow{R} \mathbb{Z}_p^{(k+1)\times k}$.
  - For $i \neq i^*$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$ and let $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \xleftarrow{R} \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{0} \neq \boldsymbol{\tau} \in \mathbb{Z}_p^{k+1}$ be any non-zero vector such that $\boldsymbol{\tau}^\top \mathbf{M} = \mathbf{0}$. Since $\mathbf{M}$ has rank at most k, such a $\boldsymbol{\tau}$ always exists and can be efficiently computed.
  - Sample $\mathbf{a}_{i^*}, \hat{\mathbf{a}}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$. Let $\mathbf{a} \leftarrow \sum_{i\in[m]} \mathbf{a}_i$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]} \hat{\mathbf{a}}_i$.
  - For each $i, j \in [m]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$. Construct $\mathbf{B}_{i,j}$ and $\hat{\mathbf{B}}_{i,j}$ for $i \neq j$ as follows:

$$\mathbf{B}_{i,j} = \begin{cases} \mathbf{a}_i \hat{\boldsymbol{\alpha}}_j^\top + \mathbf{M}\mathbf{R}_{i,j} & j \neq i^* \\ \mathbf{M}\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad \hat{\mathbf{B}}_{i,j} = \begin{cases} -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top & j \neq i^* \\ -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top + \hat{\mathbf{a}}_j \boldsymbol{\alpha}_i^\top & j = i^*. \end{cases}$$

  - Output the common reference string $\mathsf{crs}^* = \left(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right)$ and the trapdoor $\mathsf{td} = \boldsymbol{\tau} \in \mathbb{Z}_p^{k+1}$.
- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$: On input the trapdoor $\mathsf{td} = \boldsymbol{\tau} \in \mathbb{Z}_p^{k+1}$, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and the proof

$$\pi = \left(\{[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2\}_{d\in[t]}, \{[\mathbf{V}_{n+d,i}]_1, [\hat{\mathbf{V}}_{n+d,i}]_2\}_{d\in[h],i\in\{1,2\}}, \{[\mathbf{W}_{\ell,i}]_1, [\hat{\mathbf{W}}_{\ell,i}]_2\}_{\ell\in[s],i\in\{1,2\}}\right),$$

  the extraction algorithm computes $\boldsymbol{\tau}^\top[\mathbf{u}_d]_1$. It sets $w_d^* = 0$ if $\boldsymbol{\tau}^\top[\mathbf{u}]_1 = [0]_1$, and $w_d^* = 1$ otherwise for each $d = n+1, \ldots, n+h$. It outputs $\mathbf{w}^* = (w_{n+1}^*, \ldots, w_{n+h}^*)$.

We now show the CRS indistinguishability and somewhere extractable in trapdoor mode properties.

Lemma 4.8 (CRS Indistinguishability). *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen*, then Construction 4.5 satisfies CRS indistinguishability.*

Proof. Take any polynomial $m = m(\lambda)$, $s = s(\lambda)$. We now proceed via a simple hybrid argument:

- Hyb_0: This is the real distribution. At the beginning of the security game, the adversary chooses an index $i^* \in [m]$. The challenger then constructs the common reference string by running $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$:
  - Run $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample matrices $\mathbf{M}, \hat{\mathbf{M}} \xleftarrow{R} \mathbb{Z}_p^{(k+1)\times k}$.
  - For each $i \in [m]$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$ and compute $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{a} \leftarrow \sum_{i\in[m]} \mathbf{a}_i$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]} \hat{\mathbf{a}}_i$.
  - For each $i, j \in [m]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$ and let $\mathbf{B}_{i,j} \leftarrow \mathbf{M}(\boldsymbol{\alpha}_i \hat{\boldsymbol{\alpha}}_j^\top + \mathbf{R}_{i,j}) \in \mathbb{Z}_p^{(k+1)\times k}$ and $\hat{\mathbf{B}}_{i,j} \leftarrow -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top \in \mathbb{Z}_p^{(k+1)\times k}$.
  - Set $\mathsf{crs} = \left(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right)$.

  The challenger gives crs to $\mathcal{A}$ and $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_1: Same as Hyb_0 except the challenger constructs $\mathbf{B}_{i,j}$ and $\hat{\mathbf{B}}_{i,j}$ as in TrapSetup:
  - For each $i \in [m]$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$ and compute $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{a} \leftarrow \sum_{i\in[m]} \mathbf{a}_i$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]} \hat{\mathbf{a}}_i$.
  - For each $i, j \in [m]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$ and compute

$$\mathbf{B}_{i,j} = \begin{cases} \mathbf{a}_i \hat{\boldsymbol{\alpha}}_j^\top + \mathbf{M}\mathbf{R}_{i,j} & j \neq i^* \\ \mathbf{M}\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad \hat{\mathbf{B}}_{i,j} = \begin{cases} -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top & j \neq i^* \\ -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top + \hat{\mathbf{a}}_j \boldsymbol{\alpha}_i^\top & j = i^*. \end{cases}$$

- Hyb_2: Same as Hyb_1 except the challenger samples $\mathbf{a}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$:
  - For each $i \in [m]$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$. For $i \neq i^*$, let $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$ and sample $\mathbf{a}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$. For all $i \in [m]$, let $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{a} \leftarrow \sum_{i\in[m]} \mathbf{a}_i$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]} \hat{\mathbf{a}}_i$.
  - For each $i, j \in [m]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$ and compute

$$\mathbf{B}_{i,j} = \begin{cases} \mathbf{a}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{M}\mathbf{R}_{i,j} & j \neq i^* \\ \mathbf{M}\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad \hat{\mathbf{B}}_{i,j} = \begin{cases} -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} & j \neq i^* \\ -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} + \hat{\mathbf{a}}_j\boldsymbol{\alpha}_i^\mathsf{T} & j = i^*. \end{cases}$$

- Hyb_3: Same as Hyb_2 except the challenger sample $\hat{\mathbf{a}}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$:
  - For $i \neq i^*$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$ and let $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \xleftarrow{R} \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Sample $\mathbf{a}_{i^*}, \hat{\mathbf{a}}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$. Let $\mathbf{a} \leftarrow \sum_{i\in[m]} \mathbf{a}_i$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]} \hat{\mathbf{a}}_i$.
  - For each $i, j \in [m]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$ and compute

$$\mathbf{B}_{i,j} = \begin{cases} \mathbf{a}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{M}\mathbf{R}_{i,j} & j \neq i^* \\ \mathbf{M}\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad \hat{\mathbf{B}}_{i,j} = \begin{cases} -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} & j \neq i^* \\ -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} + \hat{\mathbf{a}}_j\boldsymbol{\alpha}_i^\mathsf{T} & j = i^*. \end{cases}$$

  In this experiment, crs is distributed according to $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of experiment Hyb_i with algorithm $\mathcal{A}$. We now show that each adjacent pair of hybrid experiments are computationally indistinguishable (or identical). In the following analysis, we use the fact that the k-Lin assumption implies the MDDH_k assumption (see Theorem 4.4). We will use the MDDH_k assumption in our analysis below.

Claim 4.9. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.*

Proof. This is just a syntactic relabeling. We consider the two cases $j = i^*$ and $j \neq i^*$ separately:

- Suppose $j \neq i^*$. In Hyb_0,

$$\mathbf{B}_{i,j} = \mathbf{M}(\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{R}_{i,j}) = (\mathbf{M}\boldsymbol{\alpha}_i)\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{M}\mathbf{R}_{i,j} = \mathbf{a}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{M}\mathbf{R}_{i,j}.$$

  Thus $\mathbf{B}_{i,j}$ is identically distributed in Hyb_0 and Hyb_1. In both experiments, $\hat{\mathbf{B}}_{i,j} = -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T}$.

- Suppose $j = i^*$. Consider the distribution of $\mathbf{B}_{i,i^*}$ and $\hat{\mathbf{B}}_{i,i^*}$ in Hyb_0 and Hyb_1 for $i \neq i^*$. In Hyb_0,

$$\mathbf{B}_{i,i^*} = \mathbf{M}(\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_{i^*}^\mathsf{T} + \mathbf{R}_{i,i^*}) \quad \text{and} \quad \hat{\mathbf{B}}_{i,i^*} = -\hat{\mathbf{M}}\mathbf{R}_{i,i^*}^\mathsf{T},$$

  where $\mathbf{R}_{i,i^*} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$. Suppose we instead sampled $\mathbf{R}_{i,i^*}$ as $\mathbf{R}_{i,i^*}^* - \boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_{i^*}^\mathsf{T}$ where $\mathbf{R}_{i,i^*}^* \xleftarrow{R} \mathbb{Z}_p^{k\times k}$. Certainly, $\mathbf{R}_{i,i^*}$ is still uniform over $\mathbb{Z}_p^{k\times k}$. Substituting into the above expressions, we have

$$\begin{aligned} \mathbf{B}_{i,i^*} &= \mathbf{M}(\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_{i^*}^\mathsf{T} + \mathbf{R}_{i,i^*}) = \mathbf{M}\mathbf{R}_{i,i^*}^* \\ \hat{\mathbf{B}}_{i,i^*} &= -\hat{\mathbf{M}}\mathbf{R}_{i,i^*}^\mathsf{T} = -\hat{\mathbf{M}}(\mathbf{R}_{i,i^*}^*)^\mathsf{T} + \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_{i^*}\boldsymbol{\alpha}_i^\mathsf{T} = -\hat{\mathbf{M}}(\mathbf{R}_{i,i^*}^*)^\mathsf{T} + \hat{\mathbf{a}}_{i^*}\boldsymbol{\alpha}_i^\mathsf{T}, \end{aligned}$$

  which is precisely the distribution of $\mathbf{B}_{i,i^*}$ and $\hat{\mathbf{B}}_{i,i^*}$ in Hyb_1. Thus, the adversary's view in Hyb_0 and Hyb_1 is identically distributed and the claim follows. □

Claim 4.10. *Suppose the MDDH_k assumption holds in the group $\mathbb{G}_1$. Then, for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ such that $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ for the MDDH_k assumption in $\mathbb{G}_1$:

1. Algorithm $\mathcal{B}$ receives the group description $\mathcal{G}$, the matrix $[\mathbf{M}]_1 \in \mathbb{G}_1^{(k+1)\times k}$ and a challenge $[\mathbf{z}]_1 \in \mathbb{G}_1^{k+1}$ from the MDDH_k challenger.
2. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ to obtain the challenge index $i^* \in [m]$.
3. For all $i \in [m]$, algorithm $\mathcal{B}$ samples $\boldsymbol{\alpha}_i \xleftarrow{\text{R}} \mathbb{Z}_p^k$. For $i \neq i^*$, it sets $[\mathbf{a}_i]_1 \leftarrow [\mathbf{M}]_1\boldsymbol{\alpha}_i$ and it sets $[\mathbf{a}_{i^*}]_1 \leftarrow [\mathbf{z}]_1$. Next, it samples $\hat{\mathbf{M}} \xleftarrow{\text{R}} \mathbb{Z}_p^{(k+1)\times k}$, $\hat{\boldsymbol{\alpha}}_i \xleftarrow{\text{R}} \mathbb{Z}_p^k$ and $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$ for all $i \in [m]$.
4. Algorithm $\mathcal{B}$ sets $[\mathbf{a}]_1 \leftarrow \sum_{i\in[m]}[\mathbf{a}_i]_1$ and $\hat{\mathbf{a}} \leftarrow \sum_{i\in[m]}\hat{\mathbf{a}}_i$. Then, for $i \neq j$, it samples $\mathbf{R}_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_p^{k\times k}$ and computes

$$[\mathbf{B}_{i,j}]_1 = \begin{cases} [\mathbf{a}_i]_1\hat{\boldsymbol{\alpha}}_j^\top + [\mathbf{M}]_1\mathbf{R}_{i,j} & j \neq i^* \\ [\mathbf{M}]_1\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad \hat{\mathbf{B}}_{i,j} = \begin{cases} -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top & j \neq i^* \\ -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\top + \hat{\mathbf{a}}_j\boldsymbol{\alpha}_i^\top & j = i^*. \end{cases}$$

   Importantly, algorithm $\mathcal{B}$ only computes $[\mathbf{B}_{i,j}]_1$ and $\hat{\mathbf{B}}_{i,j}$ where $i \neq j$. It does *not* need to compute $\hat{\mathbf{B}}_{i^*,i^*}$ which would depend on the (non-existent) value $\boldsymbol{\alpha}_{i^*}$.
5. It sets $\mathsf{crs} = \left(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right)$ and gives crs to $\mathcal{A}$. Finally, it outputs whatever $\mathcal{A}$ outputs.

Using the above procedure, algorithm $\mathcal{B}$ is able to construct all of the components of crs from the encodings $[\mathbf{M}]_1$ and $[\mathbf{z}]_1$. If $\mathbf{z} = \mathbf{M}\mathbf{v}$ for some $\mathbf{v} \xleftarrow{\text{R}} \mathbb{Z}_p^k$, then crs is distributed as in Hyb_1. Conversely, if $\mathbf{z} \xleftarrow{\text{R}} \mathbb{Z}_p^k$, then crs is distributed as in Hyb_2. Hence, $\mathcal{B}$ breaks MDDH_k with the same advantage ε. □

Claim 4.11. *Suppose the* MDDH_k *assumption holds in group* $\mathbb{G}_2$*. Then, for all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. This follows by a similar argument as in the proof of Claim 4.10. Suppose there exists an efficient adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the MDDH_k assumption in $\mathbb{G}_2$:

1. Algorithm $\mathcal{B}$ receives the group description $\mathcal{G}$, the matrix $[\hat{\mathbf{M}}]_2 \in \mathbb{G}_2^{(k+1)\times k}$ and a challenge $[\hat{\mathbf{z}}]_2 \in \mathbb{G}_2^{k+1}$ from the MDDH_k challenger.
2. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ to obtain the challenge index $i^* \in [m]$.
3. It samples $\mathbf{M} \xleftarrow{\text{R}} \mathbb{Z}_p^{(k+1)\times k}$. For $i \neq i^*$, it samples $\boldsymbol{\alpha}_i \xleftarrow{\text{R}} \mathbb{Z}_p^k$ and sets $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$. It samples $\mathbf{a}_{i^*} \xleftarrow{\text{R}} \mathbb{Z}_p^k$.
4. For $i \neq i^*$, it samples $\hat{\boldsymbol{\alpha}}_i \xleftarrow{\text{R}} \mathbb{Z}_p^k$ and sets $[\hat{\mathbf{a}}_i]_2 \leftarrow [\hat{\mathbf{M}}]_2\hat{\boldsymbol{\alpha}}_i$. It sets $[\hat{\mathbf{a}}_{i^*}]_2 \leftarrow [\hat{\mathbf{z}}]_2$.
5. Algorithm $\mathcal{B}$ sets $\mathbf{a} \leftarrow \sum_{i\in[m]}\mathbf{a}_i$ and $[\hat{\mathbf{a}}]_2 \leftarrow \sum_{i\in[m]}[\hat{\mathbf{a}}_i]_2$.
6. For $i \neq j$, it samples $\mathbf{R}_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_p^{k\times k}$ and computes

$$\mathbf{B}_{i,j} = \begin{cases} \mathbf{a}_i\hat{\boldsymbol{\alpha}}_j^\top + \mathbf{M}\mathbf{R}_{i,j} & j \neq i^* \\ \mathbf{M}\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad [\hat{\mathbf{B}}_{i,j}]_2 = \begin{cases} -[\hat{\mathbf{M}}]_2\mathbf{R}_{i,j}^\top & j \neq i^* \\ -[\hat{\mathbf{M}}]_2\mathbf{R}_{i,j}^\top + [\hat{\mathbf{a}}_j]_2\boldsymbol{\alpha}_i^\top & j = i^*. \end{cases}$$

   Importantly, algorithm $\mathcal{B}$ only needs to computes $[\hat{\mathbf{B}}_{i,j}]_2$ where $i \neq j$. It does *not* need to compute $[\hat{\mathbf{B}}_{i^*,i^*}]_2$ which would depend on the (non-existent) value $\boldsymbol{\alpha}_{i^*}$.
7. It sets $\mathsf{crs} = \left(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right)$ and gives crs to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Using the above procedure, algorithm $\mathcal{B}$ is able to construct all of the components of crs from the encodings $[\mathbf{M}]_2$ and $[\mathbf{z}]_2$. If $\mathbf{z} = \mathbf{M}\mathbf{v}$ for some $\mathbf{v} \xleftarrow{R} \mathbb{Z}_p^k$, then crs is distributed as in Hyb_2. Conversely, if $\mathbf{z} \xleftarrow{R} \mathbb{Z}_p^k$, then crs is distributed as in Hyb_3. Hence, $\mathcal{B}$ breaks MDDH_k with the same advantage ε. □

Combining Claims 4.9 to 4.11, we conclude that under the MDDH_k assumption, CRS indistinguishability holds. Since k-Lin implies MDDH_k (Theorem 4.4), the same result holds under k-Lin. □

Lemma 4.12 (Somewhere Extractable in Trapdoor Mode)**.** *For all constants $k \in \mathbb{N}$, Construction 4.5 is somewhere sound in trapdoor mode.*

Proof. Take any polynomial $m = m(\lambda)$ and $s = s(\lambda)$. Let $i^* \leftarrow \mathcal{A}(1^\lambda, 1^m, 1^s)$ and $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$. By construction,

$$\mathsf{crs}^* = \left(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right) \quad \text{and} \quad \mathsf{td} = \boldsymbol{\tau},$$

where $\mathbf{M}, \hat{\mathbf{M}} \xleftarrow{R} \mathbb{Z}_p^{(k+1)\times k}$, $\mathbf{a}_{i^*}, \hat{\mathbf{a}}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$, and for $i \neq i^*$, $\mathbf{a}_i = \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i = \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$ where $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$. We start by proving the following claim that will be useful in our analysis:

Claim 4.13. *With probability $1 - \mathsf{negl}(\lambda)$, the following properties hold:*

(i) For every vector $\mathbf{v} \in \mathbb{Z}_p^{k+1}$, there exists $s, \hat{s} \in \mathbb{Z}_p$ and $\mathbf{t}, \hat{\mathbf{t}} \in \mathbb{Z}_p^k$ such that $\mathbf{v} = s\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}$ and $\mathbf{v} = \hat{s}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}$. In particular, $\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} \neq \mathbf{0}$.

(ii) Every matrix $\mathbf{A} \in \mathbb{Z}_p^{(k+1)\times(k+1)}$ can be uniquely written as

$$\mathbf{A} = s\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*} + \sum_{i\in[k]} t_i \mathbf{a}_{i^*}\hat{\mathbf{m}}_i^\mathsf{T} + \sum_{i\in[k]} u_i \mathbf{m}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \sum_{i,j\in[k]} v_{i,j}\mathbf{m}_i\hat{\mathbf{m}}_j^\mathsf{T}.$$

where $s, t_i, u_i, v_{i,j} \in \mathbb{Z}_p$, $\mathbf{m}_1, \ldots, \mathbf{m}_k$ are the columns of $\mathbf{M}$, and $\hat{\mathbf{m}}_1, \ldots, \hat{\mathbf{m}}_k$ are the columns of $\hat{\mathbf{M}}$. Moreover, we define the projection operator $\mathrm{proj}(\mathbf{A}) \mapsto s\mathbf{a}_{i^*}\mathbf{a}_{i^*}^\mathsf{T}$.

(iii) Let $\mathbf{A} \in \mathbb{Z}_p^{(k+1)\times(k+1)}$ and suppose that there exists $s \in \mathbb{Z}_p$ and $\mathbf{t}_1, \mathbf{t}_2, \mathbf{z}_1, \mathbf{z}_2 \in \mathbb{Z}_p^k$ such that

$$\mathbf{A} = s\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \mathbf{a}_{i^*}\mathbf{t}_1^\mathsf{T}\hat{\mathbf{M}}^\mathsf{T} + \mathbf{M}\mathbf{t}_2\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \mathbf{M}\mathbf{z}_1\mathbf{z}_2^\mathsf{T}\hat{\mathbf{M}}^\mathsf{T}.$$

Then, $\mathrm{proj}(\mathbf{A}) = s\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}$.

(iv) For all $\mathbf{V} \in \mathbb{Z}_p^{(k+1)\times k}$, $\mathrm{proj}(\mathbf{M}\mathbf{V}^\mathsf{T}) = 0 = \mathrm{proj}(\mathbf{V}\hat{\mathbf{M}}^\mathsf{T})$.

Proof. We show each statement separately:

(i) This statement is equivalent to saying that the matrices $\mathbf{M}' = [\mathbf{a}_{i^*} \mid \mathbf{M}]$ and $\hat{\mathbf{M}}' = [\hat{\mathbf{a}}_{i^*} \mid \hat{\mathbf{M}}] \in \mathbb{Z}_p^{(k+1)\times(k+1)}$ are full rank. By construction, the distribution of $\mathbf{M}'$ and $\hat{\mathbf{M}}'$ is uniform over $\mathbb{Z}_p^{(k+1)\times(k+1)}$. By the Schwartz-Zippel lemma, the determinant of $\mathbf{M}'$ and $\hat{\mathbf{M}}'$ is non-zero with probability at least $1 - (k+1)/p = \mathsf{negl}(\lambda)$.

(ii) Define $\mathbf{M}' = [\mathbf{a}_{i^*} \mid \mathbf{M}]$ and $\hat{\mathbf{M}}' = [\hat{\mathbf{a}}_{i^*} \mid \hat{\mathbf{M}}]$ as before, and consider $\mathbf{M}' \otimes \hat{\mathbf{M}}' \in \mathbb{Z}_p^{(k+1)^2\times(k+1)^2}$. Since $\mathbf{M}'$ and $\hat{\mathbf{M}}'$ are invertible with overwhelming probability, the matrix $\mathbf{M}' \otimes \hat{\mathbf{M}}'$ is also invertible (with inverse $(\mathbf{M}')^{-1} \otimes (\hat{\mathbf{M}}')^{-1}$). Thus, the columns of $\mathbf{M}' \otimes \hat{\mathbf{M}}'$ form a basis for $\mathbb{Z}_p^{(k+1)^2}$. Suppose we rearrange each column of $\mathbf{M}' \otimes \hat{\mathbf{M}}'$ into a $(k+1)$-by-$(k+1)$ matrix in row-major order. This yields the following collection of matrices:

$$\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}, \quad \{\mathbf{a}_{i^*}\hat{\mathbf{m}}_i^\mathsf{T}\}_{i\in[k]}, \quad \{\mathbf{m}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T}\}_{i\in[k]} \quad \{\mathbf{m}_i\hat{\mathbf{m}}_j^\mathsf{T}\}_{i,j\in[k]}. \tag{4.2}$$

Since the columns of $\mathbf{M}' \otimes \hat{\mathbf{M}}'$ form a basis for $\mathbb{Z}_p^{(k+1)^2}$, the matrices in Eq. (4.2) form a basis for $\mathbb{Z}_p^{(k+1)\times(k+1)}$, and the claim follows.

(iii) We express $\mathbf{A}$ as a linear combination of the basis vectors in Eq. (4.2):

$$\begin{aligned}\mathbf{A} &= s\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top + \mathbf{a}_{i^*}\sum_{i\in[k]} t_{1,i}\hat{\mathbf{m}}_i^\top + \sum_{i\in[k]} t_{2,i}\mathbf{m}_i\hat{\mathbf{a}}_{i^*}^\top + \left(\sum_{i\in[k]} z_{1,i}\mathbf{m}_i\right)\left(\sum_{j\in[k]} z_{2,j}\hat{\mathbf{m}}_j^\top\right)\\ &= s\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top + \sum_{i\in[k]} t_{1,i}\mathbf{a}_{i^*}\hat{\mathbf{m}}_i^\top + \sum_{i\in[k]} t_{2,i}\mathbf{m}_i\hat{\mathbf{a}}_{i^*}^\top + \sum_{i,j\in[k]} z_{1,i}z_{2,j}\mathbf{m}_i\hat{\mathbf{m}}_j^\top,\end{aligned}$$

By definition, $\mathrm{proj}(\mathbf{A}) = s\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}$.

(iv) By Property (i), we can write $\mathbf{V} = \hat{\mathbf{a}}_{i^*}\hat{\mathbf{s}}^\top + \hat{\mathbf{M}}\hat{\mathbf{T}}$ where $\hat{\mathbf{s}} \in \mathbb{Z}_p^k$ and $\hat{\mathbf{T}} \in \mathbb{Z}_p^{k\times k}$. We can further decompose $\hat{\mathbf{T}} = \sum_{i,j\in[k]} \hat{t}_{i,j}\mathbf{e}_i\mathbf{e}_j^\top$ where $\hat{t}_{i,j}$ is the $(i,j)^{\text{th}}$ component of $\hat{\mathbf{T}}$ and $\mathbf{e}_i \in \mathbb{Z}_p^k$ denotes the i^{th} canonical basis vector. Then,

$$\mathbf{M}\mathbf{V}^\top = \mathbf{M}\hat{\mathbf{s}}\hat{\mathbf{a}}_{i^*}^\top + \mathbf{M}\hat{\mathbf{T}}^\top\hat{\mathbf{M}}^\top = \mathbf{M}\hat{\mathbf{s}}\hat{\mathbf{a}}_{i^*}^\top + \sum_{i,j\in[k]} \hat{t}_{i,j}\mathbf{M}\mathbf{e}_j\mathbf{e}_i^\top\hat{\mathbf{M}}^\top.$$

By Property (iii), $\mathrm{proj}(\mathbf{M}\mathbf{V}^\top) = 0$. For $\mathrm{proj}(\mathbf{V}\hat{\mathbf{M}}^\top)$, we again appeal to Property (i) and write $\mathbf{V} = \mathbf{a}_{i^*}\mathbf{s}^\top + \mathbf{M}\mathbf{T}$ for some $\mathbf{s} \in \mathbb{Z}_p^k$ and $\mathbf{T} \in \mathbb{Z}_p^{k\times k}$. By an analogous computation, we have

$$\mathbf{V}\hat{\mathbf{M}}^\top = \mathbf{a}_{i^*}\mathbf{s}^\top\hat{\mathbf{M}}^\top + \sum_{i,j\in[k]} t_{i,j}\mathbf{M}\mathbf{e}_i\mathbf{e}_j^\top\hat{\mathbf{M}}^\top.$$

Again by Property (iii), $\mathrm{proj}(\mathbf{V}\hat{\mathbf{M}}^\top) = 0$. □

Returning to the proof of Lemma 4.12, let $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be the Boolean circuit, $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ be the set of statements, and

$$\pi = \left(\{[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2\}_{d\in[t]}, \{[\mathbf{V}_{n+d,i}]_1, [\hat{\mathbf{V}}_{n+d,i}]_2\}_{d\in[h],i\in\{1,2\}}, \{[\mathbf{W}_{\ell,i}]_1, [\hat{\mathbf{W}}_{\ell,i}]_2\}_{\ell\in[s],i\in\{1,2\}}\right)$$

be the proof the adversary outputs. Suppose $\mathsf{Verify}(\mathsf{crs}^*, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 1$. We now show the following claim:

Claim 4.14. *Suppose* $\mathsf{Verify}(\mathsf{crs}^*, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 1$. *Then, for all* $d \in [t]$, *there exists* $\mathbf{t}_d, \hat{\mathbf{t}}_d \in \mathbb{Z}_p^k$ *and* $\xi_d \in \{0,1\}$ *such that* $\mathbf{u}_d = \xi_d\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_d$ *and* $\hat{\mathbf{u}}_d = \xi_d\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_d$. *Moreover,* $\mathbf{x}_{i^*} = (\xi_1, \ldots, \xi_n)$, $\xi_t = 1$, *and for each gate* $G_\ell = (d_1, d_2, d_3) \in [t]^3$, $\xi_d = \mathit{NAND}(\xi_{d_1}, \xi_{d_2})$.

Proof. Let $\boldsymbol{\beta} = \sum_{i\neq i^*} \boldsymbol{\alpha}_i$ and $\hat{\boldsymbol{\beta}} = \sum_{i\neq i^*} \hat{\boldsymbol{\alpha}}_i$. By construction, $\mathbf{a} = \sum_{i\in[m]} \mathbf{a}_i = \mathbf{a}_{i^*} + \sum_{i\neq i^*} \mathbf{M}\boldsymbol{\alpha}_i = \mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta}$. Similarly, $\hat{\mathbf{a}} = \sum_{i\in[m]} \hat{\mathbf{a}}_i = \hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}}$. We now show the claim for each wire $d \in [t]$:

- The claim holds for all statement wires $d \in [n]$. Since Verify outputs 1,

$$\mathbf{u}_d = \sum_{i\in[m]} x_{i,d}\mathbf{a}_i = x_{i^*,d}\mathbf{a}_{i^*} + \sum_{i\neq i^*} x_{i,d}\mathbf{M}\boldsymbol{\alpha}_i = x_{i^*,d}\mathbf{a}_{i^*} + \mathbf{M}\left(\sum_{i\neq i^*} x_{i,d}\boldsymbol{\alpha}_i\right).$$

  Thus $\mathbf{u}_d$ has the desired form. Correspondingly, we can write $\hat{\mathbf{u}}_d = x_{i^*,d}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\sum_{i\neq i^*} x_{i,d}\hat{\boldsymbol{\alpha}}_i$.

- Consider a witness wire $d \in \{n+1, \ldots, n+h\}$. By Claim 4.13 (i), we can write $\mathbf{u}_d = \xi_d\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_d$, and $\hat{\mathbf{u}}_d = \hat{\xi}_d\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_d$, for some $\xi_d, \hat{\xi}_d \in \mathbb{Z}_p$ and $\mathbf{t}_d, \hat{\mathbf{t}}_d \in \mathbb{Z}_p^k$. Our goal is to show $\xi_d = \hat{\xi}_d \in \{0,1\}$. Consider the following terms from the verification relations:

$$\begin{aligned}\mathbf{a}\hat{\mathbf{u}}_d^\top &= (\mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta})(\hat{\xi}_d\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_d)^\top\\ \mathbf{u}_d\hat{\mathbf{a}}^\top &= (\xi_d\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_d)(\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}})^\top\\ \mathbf{u}_d\hat{\mathbf{u}}_d^\top &= (\xi_d\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_d)(\hat{\xi}_d\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_d)^\top\end{aligned}$$

Since Verify outputs 1, both verification relations are satisfied. The same must hold for their projections. By Claim 4.13 (iii), (iv), the following relations must hold:

$$\underbrace{\mathsf{proj}(\mathbf{a}\hat{\mathbf{u}}_d^\top)}_{\hat{\xi}_d\cdot\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} = \underbrace{\mathsf{proj}(\mathbf{u}_d\hat{\mathbf{u}}_d^\top)}_{\xi_d\hat{\xi}_d\cdot\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} + \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{V}}_{d,1}^\top)}_{0} + \underbrace{\mathsf{proj}(\mathbf{V}_{d,1}\hat{\mathbf{M}}^\top)}_{0}$$
$$\underbrace{\mathsf{proj}(\mathbf{u}_d\hat{\mathbf{a}}^\top)}_{\xi_d\cdot\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} = \underbrace{\mathsf{proj}(\mathbf{u}_d\hat{\mathbf{u}}_d^\top)}_{\xi_d\hat{\xi}_d\cdot\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} + \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{V}}_{d,2}^\top)}_{0} + \underbrace{\mathsf{proj}(\mathbf{V}_{d,2}\hat{\mathbf{M}}^\top)}_{0},.$$

By Claim 4.13 (i), $\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top \neq \mathbf{0}$, and we conclude that $\hat{\xi}_d = \xi_d\hat{\xi}_d = \xi_d$. This implies $\hat{\xi}_d = \xi_d = \xi_d^2$, and so $\xi_d = \hat{\xi}_d \in \{0,1\}$.

- Consider a wire that is the output of some gate $G_\ell = (d_1, d_2, d_3) \in [t]^3$, and suppose moreover that the claim holds for d_1, d_2: namely, $\mathbf{u}_{d_1} = \xi_{d_1}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{d_1}$, $\hat{\mathbf{u}}_{d_1} = \xi_{d_1}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_{d_1}$, $\mathbf{u}_{d_2} = \xi_{d_2}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{d_2}$, and $\hat{\mathbf{u}}_{d_2} = \xi_{d_2}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_{d_2}$, for $\xi_{d_1}, \xi_{d_2} \in \{0,1\}$ and $\mathbf{t}_{d_1}, \mathbf{t}_{d_2}, \hat{\mathbf{t}}_{d_1}, \hat{\mathbf{t}}_{d_2} \in \mathbb{Z}_p^k$. By Claim 4.13 (iii), (iv), we can write $\mathbf{u}_{d_3} = \xi_{d_3}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{d_3}$ and $\hat{\mathbf{u}}_{d_3} = \hat{\xi}_{d_3}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_{d_3}$ for some $\xi_{d_3}, \hat{\xi}_{d_3} \in \mathbb{Z}_p$ and $\mathbf{t}_{d_3}, \hat{\mathbf{t}}_{d_3} \in \mathbb{Z}_p^k$. Our goal is to show that $\xi_{d_3} = \hat{\xi}_{d_3} \in \{0,1\}$ and moreover, $\xi_{d_3} = \mathsf{NAND}(\xi_{d_1}, \xi_{d_2})$. Similar to the previous case, we consider the terms in the two verification relations:

$$\begin{aligned}
\mathbf{a}\hat{\mathbf{a}}^\top &= (\mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta})(\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}})^\top \\
\mathbf{u}_{d_1}\hat{\mathbf{u}}_{d_2}^\top &= (\xi_{d_1}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{d_1})(\xi_{d_2}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_{d_2})^\top \\
\mathbf{a}\hat{\mathbf{u}}_{d_3}^\top &= (\mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta})(\hat{\xi}_{d_3}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_{d_3})^\top \\
\mathbf{u}_{d_3}\hat{\mathbf{a}}^\top &= (\xi_{d_3}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{d_3})(\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}})^\top.
\end{aligned}$$

We apply the projection operator to the two verification relations and by Claim 4.13 (iii), (iv),

$$\underbrace{\mathsf{proj}(\mathbf{a}\hat{\mathbf{a}}^\top)}_{\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} = \underbrace{\mathsf{proj}(\mathbf{u}_{d_1}\hat{\mathbf{u}}_{d_2}^\top)}_{\xi_{d_1}\xi_{d_2}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} + \underbrace{\mathsf{proj}(\mathbf{a}\hat{\mathbf{u}}_{d_3}^\top)}_{\hat{\xi}_{d_3}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} + \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{W}}_{\ell,1}^\top)}_{0} + \underbrace{\mathsf{proj}(\mathbf{W}_{\ell,1}\hat{\mathbf{M}}^\top)}_{0}$$
$$\underbrace{\mathsf{proj}(\mathbf{a}\hat{\mathbf{a}}^\top)}_{\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} = \underbrace{\mathsf{proj}(\mathbf{u}_{d_1}\hat{\mathbf{u}}_{d_2}^\top)}_{\xi_{d_1}\xi_{d_2}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} + \underbrace{\mathsf{proj}(\mathbf{u}_{d_3}\hat{\mathbf{a}}^\top)}_{\xi_{d_3}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\top} + \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{W}}_{\ell,2}^\top)}_{0} + \underbrace{\mathsf{proj}(\mathbf{W}_{\ell,2}\hat{\mathbf{M}}^\top)}_{0}.$$

If both relations hold, we conclude

$$1 = \xi_{d_1}\xi_{d_2} + \hat{\xi}_{d_3} = \xi_{d_1}\xi_{d_2} + \xi_{d_3}.$$

This means $\xi_{d_3} = \hat{\xi}_{d_3} = 1 - \xi_{d_1}\xi_{d_2} = \mathsf{NAND}(\xi_{d_1}, \xi_{d_2})$.

- For the output wire, the output satisfiability check requires that $\mathbf{u}_t = \mathbf{a} = \mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta}$ and $\hat{\mathbf{u}}_t = \hat{\mathbf{a}} = \hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}}$. This means that $\xi_t = \hat{\xi}_t = 1$.

The first two cases show that the claim holds for all input wires $d \in [n+h]$. The final case shows that if the claim holds for the input wires to a gate, then it holds for the output wire. Inductively applying the argument to the gates of the circuit in topological order, we conclude that the claim holds for all $d \in [t]$. □

Let $\xi_1, \ldots, \xi_t \in \{0,1\}$ be the bits from Claim 4.14. By Claim 4.14, $\mathbf{x}_{i^*} = (\xi_1, \ldots, \xi_n)$, and for all gates $G = (d_1, d_2, d_3) \in [t]^3$, $\xi_{d_3} = \mathsf{NAND}(\xi_{d_1}, \xi_{d_2})$. Thus, $\xi_1, \ldots, \xi_t$ is a set of valid wire assignments for the computation $C(\mathbf{x}_{i^*}, \boldsymbol{\xi})$ where $\boldsymbol{\xi} = (\xi_{n+1}, \ldots, \xi_{n+h})$. Since the output wire $\xi_t = 1$, this means that $C(\mathbf{x}_{i^*}, \boldsymbol{\xi}) = 1$.

To complete the proof, let $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$. We claim that $\mathbf{w}^* = \boldsymbol{\xi}$. By Claim 4.14, $\mathbf{u}_d = \xi_d\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_d$. Then, $\boldsymbol{\tau}^\top\mathbf{u}_d = \xi_d\boldsymbol{\tau}^\top\mathbf{a}_{i^*} + \boldsymbol{\tau}^\top\mathbf{M}\mathbf{t}_d = \xi_d\boldsymbol{\tau}^\top\mathbf{a}_{i^*}$ since $\boldsymbol{\tau}^\top\mathbf{M} = \mathbf{0}$. Moreover, since $\mathbf{a}_{i^*}$ is uniform over $\mathbb{Z}_p^{k+1}$ and independent of $\boldsymbol{\tau}$, it follows that $\boldsymbol{\tau}^\top\mathbf{a}_{i^*} \neq 0$ with probability $1 - 1/p = 1 - \mathsf{negl}(\lambda)$. Thus, if $\xi_{n+d} = 0$, then $w_d^* = 0 = \xi_{n+d}$, and if $\xi_{n+d} = 1$, then $w_d^* = 1 = \xi_{n+d}$. Thus, $\mathbf{w}^* = (\xi_{n+1}, \ldots, \xi_{n+h}) = \boldsymbol{\xi}$. Thus, with probability $1 - \mathsf{negl}(\lambda)$, either $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi) = 0$ or $C(\mathbf{x}, \mathbf{w}^*) = 1$. The claim follows. □

By Lemmas 4.8 and 4.12, Construction 4.5 is a somewhere argument of knowledge. □

Theorem 4.15 (Succinctness). *For all constants $k \in \mathbb{N}$, Construction 4.5 is succinct and satisfies split verification (Definition 2.9).*

Proof. Take any $\lambda, m, s \in \mathbb{N}$ and consider a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s. Let $t = \mathsf{poly}(s)$ be the number of wires in C. We check each property:

- **Proof size:** A proof π consists of $t(k+1) + 2hk(k+1) + 2sk(k+1)$ elements in each of $\mathbb{G}_1$ and $\mathbb{G}_2$. Each group element can be represented in $\mathsf{poly}(\lambda)$ bits. Since k is constant and $h \le t = \mathsf{poly}(s)$, the overall proof size is $|\pi| = \mathsf{poly}(\lambda, s)$.
- **CRS size:** The common reference string crs consists of the group description $\mathcal{G}$ and $O(k^2m^2)$ elements in each of $\mathbb{G}_1$ and $\mathbb{G}_2$. When $k \in \mathbb{N}$ is a constant, the size of the verification key is $|\mathsf{vk}| = m^2 \cdot \mathsf{poly}(\lambda)$.
- **Verification key size:** The size of the verification key vk outputs by GenVK consists of $n(k+1)$ elements in each of $\mathbb{G}_1$ and $\mathbb{G}_2$. For constant k, $|\mathsf{vk}| = n \cdot \mathsf{poly}(\lambda)$.
- **Verification key generation time:** The algorithm GenVK performs $2mn(k+1)$ group operations, which requires time $\mathsf{poly}(\lambda, m, n)$.
- **Online verification time:** The running time of the online verification algorithm OnlineVerify is bounded by

$$\underbrace{nk \cdot \mathsf{poly}(\lambda)}_{\text{statement validity}} + \underbrace{hk^3 \cdot \mathsf{poly}(\lambda)}_{\text{wire validity}} + \underbrace{sk^3 \cdot \mathsf{poly}(\lambda)}_{\text{gate validity}} + \underbrace{k \cdot \mathsf{poly}(\lambda)}_{\text{output validity}} = \mathsf{poly}(\lambda, s),$$

  since $n \le s$, $h \le t = \mathsf{poly}(s)$, and $k \in \mathbb{N}$ is a constant. □

Remark 4.16 (Verifying General Quadratic Relations). The technique underlying the wire validity and gate consistency checks in Construction 4.5 readily extends to gates that compute arbitrary quadratic predicates on their inputs. For instance, this includes standard Boolean gates such as AND, OR, and XOR gates as well as gates with more than two input wires. Consider a binary-valued gate predicate of the form

$$w_\ell = \gamma + \sum_{\rho \in [T_1]} \delta_\rho w_{i_\rho} + \sum_{\rho \in [T_2]} \hat{\delta}_\rho w_{j_{\rho,1}} w_{j_{\rho,2}} \in \{0,1\}, \tag{4.3}$$

where $\ell \in [t]$ is the index of the output wire, $i_\rho, j_{\rho,1}, j_{\rho,2} \in [t]$ are indices of the input wires, and $\gamma, \delta_\rho, \hat{\delta}_\rho \in \mathbb{Z}$ are fixed coefficients associated with the gate. To support gates of this type, we adapt Construction 4.5 as follows. As in Construction 4.5, let $[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2$ be vector commitments to the values $(w_{1,d}, \ldots, w_{m,d})$ of wire d across the m instances. To check the above relation is satisfied, the prover computes

$$\zeta_{i,j} = \gamma + \delta_\rho w_{i,i_\rho} + \hat{\delta}_\rho w_{i,j_{\rho,1}} w_{j,j_{\rho,2}} - w_{i,\ell} \quad \text{and} \quad [\mathbf{W}_1]_1 = \sum_{i \ne j} \zeta_{i,j} [\mathbf{B}_{i,j}]_1 \quad \text{and} \quad [\hat{\mathbf{W}}_1]_2 = \sum_{i \ne j} \zeta_{i,j} [\hat{\mathbf{B}}_{i,j}]_2,$$

and

$$\zeta'_{i,j} = \gamma + \delta_\rho w_{i,i_\rho} + \hat{\delta}_\rho w_{i,j_{\rho,1}} w_{j,j_{\rho,2}} - w_{j,\ell} \quad \text{and} \quad [\mathbf{W}_2]_1 = \sum_{i \ne j} \zeta'_{i,j} [\mathbf{B}_{i,j}]_1 \quad \text{and} \quad [\hat{\mathbf{W}}_2]_2 = \sum_{i \ne j} \zeta'_{i,j} [\hat{\mathbf{B}}_{i,j}]_2.$$

To check that the gate is satisfied, the verifier checks

$$\gamma [\mathbf{a}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 + \sum_{\rho \in [T_1]} \delta_\rho [\mathbf{u}_{i_\rho}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 + \sum_{\rho \in [T_2]} \hat{\delta}_\rho [\mathbf{u}_{j_{\rho,1}}]_1 \cdot [\hat{\mathbf{u}}^\mathsf{T}_{j_{\rho,2}}] - [\mathbf{u}_\ell]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 = [\mathbf{M}]_1 \cdot [\hat{\mathbf{W}}_1^\mathsf{T}]_2 + [\mathbf{W}_1]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2,$$

and

$$\gamma [\mathbf{a}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 + \sum_{\rho \in [T_1]} \delta_\rho [\mathbf{u}_{i_\rho}]_1 \cdot [\hat{\mathbf{a}}^\mathsf{T}]_2 + \sum_{\rho \in [T_2]} \hat{\delta}_\rho [\mathbf{u}_{j_{\rho,1}}]_1 \cdot [\hat{\mathbf{u}}^\mathsf{T}_{j_{\rho,2}}] - [\mathbf{a}]_1 \cdot [\hat{\mathbf{u}}^\mathsf{T}_\ell]_2 = [\mathbf{M}]_1 \cdot [\hat{\mathbf{W}}_2^\mathsf{T}]_2 + [\mathbf{W}_2]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2.$$

Completeness. To argue completeness, consider each term in the first verification relation:

$$\begin{aligned}
\mathbf{a}\hat{\mathbf{a}}^\top &= \sum_{i,j\in[m]} \mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} \mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} \mathbf{a}_i\hat{\mathbf{a}}_j^\top \\
\mathbf{u}_{i_\rho}\hat{\mathbf{a}}^\top &= \sum_{i,j\in[m]} w_{i,i_\rho}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,i_\rho}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,i_\rho}\mathbf{a}_i\hat{\mathbf{a}}_j^\top \\
\mathbf{u}_{j_{\rho,1}}\hat{\mathbf{u}}_{j_{\rho,2}}^\top &= \sum_{i,j\in[m]} w_{i,j_{\rho,1}}w_{j,j_{\rho,2}}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,j_{\rho,1}}w_{i,j_{\rho,2}}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,j_{\rho,1}}w_{j,j_{\rho,2}}\mathbf{a}_i\hat{\mathbf{a}}_j^\top \\
\mathbf{u}_\ell\hat{\mathbf{a}}^\top &= \sum_{i,j\in[m]} w_{i,\ell}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\in[m]} w_{i,\ell}\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} w_{i,\ell}\mathbf{a}_i\hat{\mathbf{a}}_j^\top
\end{aligned}$$

Then, the first verification relation becomes

$$\gamma\mathbf{a}\hat{\mathbf{a}}^\top + \sum_{\rho\in[T_1]} \delta_\rho\mathbf{u}_{i_\rho}\hat{\mathbf{a}}^\top + \sum_{\rho\in[T_2]} \hat{\delta}_\rho\mathbf{u}_{j_{\rho,1}}\hat{\mathbf{u}}_{j_{\rho,2}} - \mathbf{u}_\ell\hat{\mathbf{a}}^\top = \sum_{i\in[m]} Z_i\mathbf{a}_i\hat{\mathbf{a}}_i^\top + \sum_{i\neq j} Z_{i,j}\mathbf{a}_i\hat{\mathbf{a}}_j^\top,$$

where

$$\begin{aligned}
Z_i &= \gamma + \sum_{\rho\in[T_1]} \delta_\rho w_{i,i_\rho} + \sum_{\rho\in[T_2]} \hat{\delta}_\rho w_{i,j_{\rho,1}}w_{i,j_{\rho,2}} - w_{i,\ell} \\
Z_{i,j} &= \gamma + \sum_{\rho\in[T_1]} \delta_\rho w_{i,i_\rho} + \sum_{\rho\in[T_2]} \hat{\delta}_\rho w_{i,j_{\rho,1}}w_{j,j_{\rho,2}} - w_{i,\ell} = \zeta_{i,j}.
\end{aligned}$$

If Eq. (4.3) holds for all m instances, then $Z_i = 0$ for all $i \in [\ell]$ and we are only left with $\sum_{i\neq j} Z_{i,j}\mathbf{a}_i\hat{\mathbf{a}}_j^\top$. By construction, the right-hand side of the first verification relation is

$$\mathbf{M}\hat{\mathbf{W}}_1^\top + \mathbf{W}_1\hat{\mathbf{M}}^\top = \sum_{i\neq j} \zeta_{i,j}(\mathbf{M}\hat{\mathbf{B}}_{i,j}^\top + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\top) = \sum_{i\neq j} \zeta_{i,j}\mathbf{a}_i\hat{\mathbf{a}}_j^\top = \sum_{i\neq j} Z_{i,j}\mathbf{a}_i\hat{\mathbf{a}}_j,$$

using the relation from Eq. (4.1). Thus, the first verification relation holds. A similar calculation applies to the second verification relation and completeness follows.

Somewhere argument of knowledge. The somewhere argument of knowledge property follows analogously as the proof of Theorem 4.7. Since we did not need to modify the CRS to support general gates, CRS indistinguishability holds. It suffices to show that the scheme is somewhere extractable in trapdoor mode. The proof of Lemma 4.12 uses an inductive strategy where we show that as long as the commitments to the input wires of a gate is well-formed, then the commitment to the output wire respects the gate constraint. Specifically, for each input wire d to the gate, suppose that $\mathbf{u}_d = \xi_d\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_d$ and $\hat{\mathbf{u}}_d = \xi_d\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_d$ for some $\xi_d \in \{0,1\}$ and $\mathbf{t}_d, \hat{\mathbf{t}}_d \in \mathbb{Z}_p^k$. By Claim 4.13 (iii), (iv), the commitments $\mathbf{u}_\ell$ and $\hat{\mathbf{u}}_\ell$ to the output wires can be written as $\mathbf{u}_\ell = \xi_\ell\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_\ell$ and $\hat{\mathbf{u}}_\ell = \hat{\xi}_\ell\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_\ell$ for some $\xi_\ell, \hat{\xi}_\ell \in \mathbb{Z}_p$ and $\mathbf{t}_\ell, \hat{\mathbf{t}}_\ell \in \mathbb{Z}_p^k$. Our goal is to show that $\xi_\ell = \hat{\xi}_\ell$ and moreover, $\xi_\ell = \gamma + \sum_{\rho\in[T_1]} \delta_\rho\xi_{i_\rho} + \sum_{\rho\in[T_2]} \hat{\delta}_\rho\xi_{j_{\rho,1}}\xi_{j_{\rho,2}} \in \{0,1\}$. Following the identical strategy as in the proof of Lemma 4.12, we consider the terms in the verification relations:

$$\begin{aligned}
\mathbf{a}\hat{\mathbf{a}}^\top &= (\mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta})(\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}})^\top \\
\mathbf{u}_{i_\rho}\hat{\mathbf{a}}^\top &= (\xi_{i_\rho}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{i_\rho})(\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}})^\top \\
\mathbf{u}_{j_{\rho,1}}\hat{\mathbf{u}}_{j_{\rho,2}}^\top &= (\xi_{j_{\rho,1}}\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_{j_{\rho,1}})(\xi_{j_{\rho,2}}\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_{j_{\rho,2}})^\top \\
\mathbf{u}_\ell\hat{\mathbf{a}}^\top &= (\xi_\ell\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}_\ell)(\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\boldsymbol{\beta}})^\top \\
\mathbf{a}\hat{\mathbf{u}}_\ell^\top &= (\mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta})(\hat{\xi}_\ell\hat{\mathbf{a}}_{i^*} + \hat{\mathbf{M}}\hat{\mathbf{t}}_\ell)^\top.
\end{aligned}$$

We apply the projection operator to the two verification relations and by Claim 4.13 (iii), (iv),

$$\underbrace{\mathsf{proj}(\gamma \mathbf{a}\hat{\mathbf{a}}^\mathsf{T})}_{\gamma \mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} + \underbrace{\sum_{\rho\in[T_1]} \mathsf{proj}(\delta_\rho \mathbf{u}_{i_\rho}\hat{\mathbf{a}}^\mathsf{T})}_{\sum_{\rho\in[T_1]}\delta_\rho\xi_{i_\rho}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} + \underbrace{\sum_{\rho\in[T_2]} \mathsf{proj}(\hat{\delta}_\rho \mathbf{u}_{j_{\rho,1}}\hat{\mathbf{u}}_{j_{\rho,2}}^\mathsf{T})}_{\sum_{\rho\in[T_2]}\hat{\delta}_\rho\xi_{j_{\rho,1}}\xi_{j_{\rho,2}}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} - \underbrace{\mathsf{proj}(\mathbf{u}_\ell\hat{\mathbf{a}}^\mathsf{T})}_{\xi_\ell \mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} = \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{W}}_1^\mathsf{T})}_{0} + \underbrace{\mathsf{proj}(\mathbf{W}_1\hat{\mathbf{M}}^\mathsf{T})}_{0}$$

$$\underbrace{\mathsf{proj}(\gamma \mathbf{a}\hat{\mathbf{a}}^\mathsf{T})}_{\gamma \mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} + \underbrace{\sum_{\rho\in[T_1]} \mathsf{proj}(\delta_\rho \mathbf{u}_{i_\rho}\hat{\mathbf{a}}^\mathsf{T})}_{\sum_{\rho\in[T_1]}\delta_\rho\xi_{i_\rho}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} + \underbrace{\sum_{\rho\in[T_2]} \mathsf{proj}(\hat{\delta}_\rho \mathbf{u}_{j_{\rho,1}}\hat{\mathbf{u}}_{j_{\rho,2}}^\mathsf{T})}_{\sum_{\rho\in[T_2]}\hat{\delta}_\rho\xi_{j_{\rho,1}}\xi_{j_{\rho,2}}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} - \underbrace{\mathsf{proj}(\mathbf{a}\hat{\mathbf{u}}_\ell^\mathsf{T})}_{\hat{\xi}_\ell \mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T}} = \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{W}}_2^\mathsf{T})}_{0} + \underbrace{\mathsf{proj}(\mathbf{W}_2\hat{\mathbf{M}}^\mathsf{T})}_{0}.$$

In combination, this means that

$$\xi_\ell = \gamma + \sum_{\rho\in[T_1]} \delta_\rho \xi_{i_\rho} + \sum_{\rho\in[T_2]} \hat{\delta}_\rho \xi_{j_{\rho,1}}\xi_{j_{\rho,2}} = \hat{\xi}_\ell.$$

Since Eq. (4.3) is a binary-valued predicate and the input assignments $\xi_{i_\rho}, \xi_{j_{\rho,1}}, \xi_{j_{\rho,2}} \in \{0,1\}$ by the inductive hypothesis, this means that $\xi_\ell \in \{0,1\}$. By the same argument as in the proof of Lemma 4.12, we conclude that the extracted wire assignment $(\xi_1, \ldots, \xi_t)$ satisfies the gate constraint Eq. (4.3).

5 BARG Bootstrapping to Reduce CRS Size

In this section, we describe how to recursively compose succinct batch arguments for NP with a long CRS to obtain a BARG with a short CRS (i.e., with size that is sublinear in the number of instances). The bootstrapping construction applies to any BARG with a split verification procedure (Definition 2.9). We refer to Section 1.2.2 for an overview of the construction.

Construction 5.1 (BARG Bootstrapping)**.** Let $B \in \mathbb{N}$ be a batch size parameter. Let $\Pi_{\mathsf{BARG}}^{(0)} = (\mathsf{BARG}_0.\mathsf{Setup}, \mathsf{BARG}_0.\mathsf{Prove}, \mathsf{BARG}_0.\mathsf{GenVK}, \mathsf{BARG}_0.\mathsf{OnlineVerify})$ be a batch argument with split verification. We construct a new BARG with split verification as follows:

- $\mathsf{Setup}(1^\lambda, 1^m, 1^s)$: On input the security parameter λ, the number of instances m, and a bound on the circuit size s, the setup algorithm proceeds as follows:
  - Sample $\mathsf{crs}_{\mathsf{base}} \leftarrow \mathsf{BARG}_0.\mathsf{Setup}(1^\lambda, 1^B, 1^s)$.
  - Let $\ell_\pi = \ell_\pi(\lambda, B, s)$ and $\ell_{\mathsf{vk}} = \ell_{\mathsf{vk}}(\lambda, B, s)$ be the length of the proofs π and verification keys vk output by $\mathsf{BARG}_0.\mathsf{Prove}(\mathsf{crs}_{\mathsf{base}}, \cdot, \cdot, \cdot)$ and $\mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}_{\mathsf{base}}, \cdot)$, respectively.
  - Define the Boolean circuit $C_{\mathsf{top}} \colon \{0,1\}^{\ell_{\mathsf{vk}}} \times \{0,1\}^{\ell_\pi} \to \{0,1\}$ as $C_{\mathsf{top}}(\mathsf{vk}, \pi) := \mathsf{BARG}_0.\mathsf{OnlineVerify}(\mathsf{vk}, C, \pi)$. Let s_{top} be a bound on the size of the circuit C_{top}.
  - Sample $\mathsf{crs}_{\mathsf{top}} \leftarrow \mathsf{BARG}_0.\mathsf{Setup}(1^\lambda, 1^{m/B}, 1^{s_{\mathsf{top}}})$ and output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$.

  We will require that $B \le m$.
- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$: On input $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$, the Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$, the prove algorithm proceeds as follows:
  - For each $i \in [m/B]$, compute $\pi_i \leftarrow \mathsf{BARG}_0.\mathsf{Prove}(\mathsf{crs}_{\mathsf{base}}, C, (\mathbf{x}_{(i-1)B+1}, \ldots, x_{iB}), (\mathbf{w}_{(i-1)B+1}, \ldots, \mathbf{w}_{iB}))$.
  - Output the proof $\pi \leftarrow \mathsf{BARG}_0.\mathsf{Prove}\big(\mathsf{crs}_{\mathsf{top}}, C_{\mathsf{top}}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_{m/B}), (\pi_1, \ldots, \pi_{m/B})\big)$.
- $\mathsf{GenVK}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_m))$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$ and statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, the verification key generation algorithm proceeds as follows:
  - For each $i \in [m/B]$, compute $\mathsf{vk}_i \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}_{\mathsf{base}}, (\mathbf{x}_{(i-1)B+1}, \ldots, \mathbf{x}_{iB}))$.

- Compute and output $\mathsf{vk} \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}_{\mathsf{top}}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_{m/B}))$.

- $\mathsf{OnlineVerify}(\mathsf{vk}, C, \pi)$: On input a verification key vk and a proof π, output $\mathsf{BARG}_0.\mathsf{OnlineVerify}(\mathsf{vk}, C_{\mathsf{top}}, \pi)$.

Theorem 5.2 (Completeness). *If $\Pi^{(0)}_{\mathsf{BARG}}$ is complete, then Construction 5.1 is also complete.*

Proof. Follows by construction. □

Theorem 5.3 (Somewhere Argument of Knowledge). *If $\Pi^{(0)}_{\mathsf{BARG}}$ is a somewhere argument of knowledge, then Construction 5.1 is also a somewhere argument of knowledge.*

Proof. We start by defining the trapdoor setup and extraction algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$: Write $i^* = (i^*_{\mathsf{top}} - 1)B + i^*_{\mathsf{base}}$ where $i^*_{\mathsf{top}} \in [m/B]$ and $i^*_{\mathsf{base}} \in [B]$. The trapdoor setup algorithm samples the CRS components using the corresponding trapdoor setup algorithms:
  - Sample $(\mathsf{crs}^*_{\mathsf{base}}, \mathsf{td}_{\mathsf{base}}) \leftarrow \mathsf{BARG}_0.\mathsf{TrapSetup}(1^\lambda, 1^B, 1^s, i^*_{\mathsf{base}})$.
  - Sample $(\mathsf{crs}^*_{\mathsf{top}}, \mathsf{td}_{\mathsf{top}}) \leftarrow \mathsf{BARG}_0.\mathsf{TrapSetup}(1^\lambda, 1^{m/B}, 1^{s_{\mathsf{top}}}, i^*_{\mathsf{top}})$.
  - Output $\mathsf{crs}^* = (\mathsf{crs}^*_{\mathsf{base}}, \mathsf{crs}^*_{\mathsf{top}})$ and the trapdoor $\mathsf{td} = (\mathsf{crs}_{\mathsf{top}}, i^*_{\mathsf{top}}, \mathsf{td}_{\mathsf{base}}, \mathsf{td}_{\mathsf{top}})$.

- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi)$: On input the trapdoor $\mathsf{td} = (\mathsf{crs}_{\mathsf{top}}, i^*_{\mathsf{top}}, \mathsf{td}_{\mathsf{base}}, \mathsf{td}_{\mathsf{top}})$, the circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ and a proof π, proceed as follows:
  - For each $i \in [m/B]$, compute $\mathsf{vk}^*_i \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}^*_{\mathsf{top}}, (\mathbf{x}_{(i-1)B+1}, \ldots, \mathbf{x}_{iB}))$.
  - Compute $\pi_{\mathsf{base}} \leftarrow \mathsf{BARG}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{top}}, C_{\mathsf{top}}, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_{m/B}), \pi)$,
  - Output $\mathsf{BARG}_0.\mathsf{Extract}\big(\mathsf{td}_{\mathsf{base}}, C, (\mathbf{x}_{(i^*_{\mathsf{top}}-1)B+1}, \ldots, \mathbf{x}_{i^*_{\mathsf{top}}B}), \pi_{\mathsf{base}}\big)$.

We now show the CRS indistinguishability and somewhere extractable in trapdoor mode properties.

Lemma 5.4 (CRS Indistinguishability). *If $\Pi^{(0)}_{\mathsf{BARG}}$ is a somewhere argument of knowledge (specifically, it satisfies CRS indistinguishability), then Construction 5.1 satisfies CRS indistinguishability.*

Proof. This is immediate by a standard hybrid argument. Namely, the CRS in Construction 5.1 consists of two independent common reference strings for $\Pi^{(0)}_{\mathsf{BARG}}$. □

Lemma 5.5 (Somewhere Extractable in Trapdoor Mode). *If $\Pi^{(0)}_{\mathsf{BARG}}$ is a somewhere argument of knowledge (specifically, if it is somewhere extractable in trapdoor mode), then Construction 5.1 is somewhere extractable in trapdoor mode.*

Proof. Take any polynomial $m = m(\lambda)$ and $s = s(\lambda)$. Let $i^* \leftarrow \mathcal{A}(1^\lambda, 1^m, 1^s)$ and write $i^* = (i^*_{\mathsf{top}} - 1)B + i^*_{\mathsf{base}}$ where $i^*_{\mathsf{top}} \in [m/B]$ and $i^*_{\mathsf{base}} \in [B]$. Let $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be the Boolean circuit, $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ be the set of statements, and π be the proof output by the adversary. Let $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$ and $\mathsf{vk}^* \leftarrow \mathsf{GenVK}(\mathsf{crs}^*, (\mathbf{x}_1, \ldots, \mathbf{x}_m))$. Then $\mathsf{crs}^* = (\mathsf{crs}^*_{\mathsf{base}}, \mathsf{crs}^*_{\mathsf{top}})$ and $\mathsf{td} = (\mathsf{td}_{\mathsf{base}}, \mathsf{td}_{\mathsf{top}})$ where

- $(\mathsf{crs}^*_{\mathsf{base}}, \mathsf{td}_{\mathsf{base}}) \leftarrow \mathsf{BARG}_0.\mathsf{TrapSetup}(1^\lambda, 1^B, 1^s, i^*_{\mathsf{base}})$;
- $(\mathsf{crs}^*_{\mathsf{top}}, \mathsf{td}_{\mathsf{top}}) \leftarrow \mathsf{BARG}_0.\mathsf{TrapSetup}(1^\lambda, 1^{m/B}, 1^{s_{\mathsf{top}}}, i^*_{\mathsf{top}})$;
- $\mathsf{vk}^*_i \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}\big(\mathsf{crs}^*_{\mathsf{base}}, (\mathbf{x}_{(i-1)B+1}, \ldots, \mathbf{x}_{iB})\big)$ for each $i \in [m/B]$; and
- $\mathsf{vk}^* \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}\big(\mathsf{crs}^*_{\mathsf{top}}, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_{m/B})\big)$.

Suppose $\mathsf{OnlineVerify}(\mathsf{vk}^*, C, \pi) = 1$. Let $\pi_{\mathsf{base}} \leftarrow \mathsf{BARG}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{top}}, C_{\mathsf{top}}, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_{m/B}), \pi)$ be the extracted proof and let $\mathbf{w}^* \leftarrow \mathsf{BARG}_0.\mathsf{Extract}\big(\mathsf{td}_{\mathsf{base}}, C, (\mathbf{x}_{(i^*_{\mathsf{top}}-1)B+1}, \ldots, \mathbf{x}_{i^*_{\mathsf{top}}B}), \pi_{\mathsf{base}}\big)$ be the extracted witness. We proceed via a sequence of claims:

Claim 5.6. *If $\Pi^{(0)}_{\mathsf{BARG}}$ is a somewhere extractable argument of knowledge, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{BARG}_0.\mathsf{OnlineVerify}\left(\mathsf{vk}^*_{i^*_{\mathsf{top}}}, C, \pi_{\mathsf{base}}\right) = 1\right] = 1 - \mathsf{negl}(\lambda).$$

Proof. First $(\mathsf{crs}^*_{\mathsf{top}}, \mathsf{td}_{\mathsf{top}})$ is sampled using $\mathsf{BARG}_0.\mathsf{TrapSetup}$ with index i^*_{top}. If $\mathsf{BARG}_0.\mathsf{OnlineVerify}(\mathsf{vk}^*, C_{\mathsf{top}}, \pi) = 1$ with $\mathsf{vk}^* \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}^*_{\mathsf{top}}, (\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_{m/B}))$, then $C_{\mathsf{top}}\left(\mathsf{vk}^*_{i^*_{\mathsf{top}}}, \pi_{\mathsf{base}}\right) = 1$ with probability $1 - \mathsf{negl}(\lambda)$. Otherwise, we have an adversary that breaks somewhere extractability of $\Pi^{(0)}_{\mathsf{BARG}}$. By definition of C_{top}, this means $\mathsf{BARG}_0.\mathsf{OnlineVerify}\left(\mathsf{vk}^*_{i^*_{\mathsf{top}}}, C, \pi_{\mathsf{base}}\right) = 1$. □

Claim 5.7. *If $\Pi^{(0)}_{\mathsf{BARG}}$ is a somewhere extractable argument of knowledge, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\Pr[C(\mathbf{x}_{i^*}, \mathbf{w}^*) = 1] = 1 - \mathsf{negl}(\lambda)$.

Proof. This follows from the fact that $(\mathsf{crs}_{\mathsf{base}}, \mathsf{td}_{\mathsf{base}})$ is sampled using $\mathsf{BARG}_0.\mathsf{TrapSetup}$ with index i^*_{base}. By Claim 5.6, with probability $1 - \mathsf{negl}(\lambda)$, $\mathsf{BARG}_0.\mathsf{OnlineVerify}\left(\mathsf{vk}^*_{i^*_{\mathsf{top}}}, C, \pi_{\mathsf{base}}\right) = 1$, where

$$\mathsf{vk}^*_{i^*_{\mathsf{top}}} \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}^*_{\mathsf{base}}, (\mathbf{x}_{(i^*_{\mathsf{top}}-1)B+1}, \ldots, \mathbf{x}_{i^*_{\mathsf{top}}B})).$$

Somewhere extractability of $\Pi^{(0)}_{\mathsf{BARG}}$ then implies that with probability $1 - \mathsf{negl}(\lambda)$,

$$C(\mathbf{x}_{(i^*_{\mathsf{top}}-1)B+i^*_{\mathsf{base}}}, \mathbf{w}^*) = C(\mathbf{x}_{i^*}, \mathbf{w}^*) = 1.$$ □

Combining Claims 5.6 and 5.7, we conclude that with probability $1 - \mathsf{negl}(\lambda)$, the extracted witness $\mathbf{w}^*$ satisfies $C(\mathbf{x}_{i^*}, \mathbf{w}^*) = 1$ and the claim follows. □

The somewhere argument of knowledge property now follows from Lemmas 5.4 and 5.5. □

Theorem 5.8 (Succinctness)**.** *Suppose $\Pi^{(0)}_{\mathsf{BARG}}$ is a succinct BARG with split verification and CRS size $\ell_0(\lambda, m, s) = m^d \cdot \mathsf{poly}(\lambda, s)$, for some constant $d \in \mathbb{N}$. Then Construction 5.1 is a succinct BARG with split verification and CRS size*

$$\ell(\lambda, m, s, B) = B^d \cdot \mathsf{poly}(\lambda, s) + (m/B)^d \cdot \mathsf{poly}(\lambda, \log m, s).$$

Moreover, if $\ell_0(\lambda, m, s) = m^d \cdot \mathsf{poly}(\lambda)$, then $\ell(\lambda, m, s, B) = (B^d + (m/B)^d) \cdot \mathsf{poly}(\lambda)$.

Proof. We verify each of the required properties:

- **CRS size:** The CRS in Construction 5.1 consists of two common reference strings $(\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$ for $\Pi^{(0)}_{\mathsf{BARG}}$. The size of $\mathsf{crs}_{\mathsf{base}}$ is $\ell_0(\lambda, B, s)$ and the size of $\mathsf{crs}_{\mathsf{top}}$ is $\ell_0(\lambda, m/B, s')$ where s' is a bound on the size of the circuit C_{top} computing $\mathsf{BARG}_0.\mathsf{OnlineVerify}(\mathsf{vk}, \cdot)$ where $\mathsf{vk} \leftarrow \mathsf{BARG}_0.\mathsf{GenVK}(\mathsf{crs}_{\mathsf{base}}, \cdot)$. By succinctness of $\Pi^{(0)}_{\mathsf{BARG}}$, the size s_{top} of C_{top} is bounded by some polynomial $\mathsf{poly}(\lambda, \log m, s)$. Thus,

  $$\ell(\lambda, m, s, B) = B^d \cdot \mathsf{poly}(\lambda, s) + (m/B)^d \cdot \mathsf{poly}(\lambda, \log m, s),$$

  as required. When ℓ_0 is independent of s, the same is true for ℓ.

- **Proof size:** The proof π in Construction 5.1 consists of a proof for $\Pi^{(0)}_{\mathsf{BARG}}$ instantiated with m/B instances and circuits of size at most $s_{\mathsf{top}} = \mathsf{poly}(\lambda, \log m, s)$. Thus, $|\pi| \le \mathsf{poly}(\lambda, \log(m/B), s_{\mathsf{top}}) = \mathsf{poly}(\lambda, \log m, s)$.

- **Verification key generation time:** The verification key generation algorithm GenVK consists of two main components:

  - First, it runs m/B copies of $\mathsf{BARG}_0.\mathsf{GenVK}$ with B instances (of length n) and circuits of size at most s. By succinctness of $\Pi^{(0)}_{\mathsf{BARG}}$, each copy runs in time $\mathsf{poly}(\lambda, B, n)$, so generating $\mathsf{vk}_1, \ldots, \mathsf{vk}_{m/B}$ requires time $m/B \cdot \mathsf{poly}(\lambda, B, n) = \mathsf{poly}(\lambda, m, n)$.

– Next, it runs $\mathsf{BARG}_0.\mathsf{GenVK}$ with m/B instances (of length ℓ_{vk} where ℓ_{vk} is a bound on the length of the verification keys vk_i) and circuits of size at most s_{top}. Again by succinctness of $\Pi^{(0)}_{\mathsf{BARG}}$, $\ell_{\mathsf{vk}} \leq \mathsf{poly}(\lambda, \log m, n)$. Thus, this step requires time $\mathsf{poly}(\lambda, m/B, \ell_{\mathsf{vk}}) = \mathsf{poly}(\lambda, m, n)$.

Since both steps complete in time $\mathsf{poly}(\lambda, m, n)$, the claim holds.

- **Verification key size:** The verification key vk in Construction 5.1 consists of a single verification key for $\Pi^{(0)}_{\mathsf{BARG}}$ with m/B instances and circuits of size at most s_{top}. By succinctness of $\Pi^{(0)}_{\mathsf{BARG}}$, $|\mathsf{vk}| \leq \mathsf{poly}(\lambda, \log(m/B), s_{\mathsf{top}}) = \mathsf{poly}(\lambda, \log m, s)$.
- **Online verification time:** The verification algorithm in Construction 5.1 simply runs $\mathsf{BARG}_0.\mathsf{OnlineVerify}$ with m/B instances and a circuit of size s_{top}. By succinctness of $\Pi^{(0)}_{\mathsf{BARG}}$, the running time is at most

$$\mathsf{poly}(\lambda, \log(m/B), s_{\mathsf{top}}) = \mathsf{poly}(\lambda, \log m, s). \qquad \square$$

Corollary 5.9 (BARG for NP with Short CRS)**.** *Suppose there exists a batch argument for* NP *with split verification and a CRS of size* $\mathsf{poly}(\lambda, m, s)$*, where* m *is the number of instances and* s *is the circuit size. Then, for every constant* $\varepsilon > 0$*, there exists a batch argument for* NP *with split verification and a CRS of size* $m^{\varepsilon} \cdot \mathsf{poly}(\lambda, s)$*.*

Proof. Let $\Pi^{(0)}_{\mathsf{BARG}}$ be the BARG with CRS size at most $m^d \cdot \mathsf{poly}(\lambda, s)$ for some constant $d \in \mathbb{N}$. Let $k = \lceil \log(2d/\varepsilon) \rceil \in \mathbb{N}$. For $i \in [k]$, let $\Pi^{(i)}_{\mathsf{BARG}}$ be the BARG formed by applying Construction 5.1 to $\Pi^{(i-1)}_{\mathsf{BARG}}$ with $B = \sqrt{m}$. Let ℓ_i denote the length of the CRS in $\Pi^{(i)}_{\mathsf{BARG}}$. Since $\ell_0(\lambda, m, s) = m^d \cdot \mathsf{poly}(\lambda, s)$, we can inductively apply Theorem 5.8 to show that

$$\ell_i(\lambda, m, s) = m^{d/2^i} \cdot \mathsf{poly}(\lambda, \log m, s).$$

Substituting $k = \lceil \log(2d/\varepsilon) \rceil$ into the above, we have that

$$\ell_k(\lambda, m, s) \leq m^{\varepsilon/2} \cdot \mathsf{poly}(\lambda, \log m, s) < m^{\varepsilon} \cdot \mathsf{poly}(\lambda, s),$$

since $2d/\varepsilon$ is a constant. The other succinctness requirements are preserved since we compose a *constant* number of times. $\square$

Corollary 5.10 (BARG for NP with Short CRS from Pairings)**.** *For any constant* $k \geq 1$*, if the* k-Lin *assumption holds in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *with respect to a prime-order group generator* $\mathsf{GroupGen}$ *(or, alternatively, if the subgroup decision assumption holds with respect to a composite-order group generator* $\mathsf{CompGroupGen}$*), then for every constant* $\varepsilon > 0$*, there exists a BARG for* NP *with split verification and a CRS of size* $m^{\varepsilon} \cdot \mathsf{poly}(\lambda)$*.*

Proof. Follows by combining Construction 4.5 (alternatively, Construction 3.3) with Corollary 5.9. Note that the CRS in Construction 4.5 (alternatively, Construction 3.3) is independent of the circuit size s (Theorems 3.10 and 4.15). $\square$

Remark 5.11 (Bootstrapping Tradeoffs)**.** The bootstrapping construction from Construction 5.1 and Corollary 5.10 is best viewed as a way to reduce the CRS size dependence on the number of instances m (e.g., from m^2 to m^{ε}) in exchange for a *higher* dependence on the security parameter λ. In general, the dependence on the security parameter scales *exponentially* with the depth of the composition. This is also the reason we are limited to constant-depth composition. Recursive composition yields a similar blowup (with respect to λ) in the proof size, verification key size, and verification time.

6 Delegation for RAM Programs

In this section, we show how our techniques for constructing BARGs for NP can be leveraged to obtain delegation schemes for RAM programs. We obtain the delegation scheme by invoking the generic compiler by Choudhuri et al. [CJJ21b] which combines a BARG for index languages with a somewhere extractable commitment scheme. Choudhuri et al. showed that the Hubácek-Wichs somewhere statistically binding (SSB) hash function [HW15] is

already a somewhere extractable commitment, thus obtaining an instantiation from LWE. However, the SSB hash function from DDH [OPWW15] does not satisfy the stronger extractability requirement. In this section (Section 6.2), we show that our techniques for constructing BARGs can be combined with any SSB hash function to obtain a somewhere extractable commitment with a long CRS. We then describe an analogous bootstrapping procedure to reduce the CRS size (Section 6.3). Finally, we combine our somewhere extractable commitment with the BARG for index languages (Corollary 5.10 and Remark 2.10) to obtain a RAM delegation scheme (Corollaries 6.28 and 6.30).

6.1 Somewhere Extractable Commitments

We begin by recalling the concept of a somewhere statistically binding (SSB) hash function [HW15] and the closely-related notion of a somewhere extractable commitments from Choudhuri et al. [CJJ21b].

Definition 6.1 (Somewhere Statistically Binding Hash Function [HW15, OPWW15])**.** A somewhere statistically binding (SSB) hash function with block length ℓ_{blk}, output length ℓ_{hash}, and opening length ℓ_{open} is a tuple of efficient algorithms $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*) \to \mathsf{hk}$: On input the security parameter λ, the block size ℓ_{blk}, the message length $N \le 2^\lambda$, and an index $i^* \in [N]$, the setup algorithm outputs a hashing key hk. Both N and i^* are encoded in *binary*; in particular, this means that $|\mathsf{hk}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$. We let $\Sigma = \{0,1\}^{\ell_{\mathsf{blk}}}$ denote the block alphabet.
- $\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \to h$: On input the hash key hk and a message $\mathbf{x} \in \Sigma^N$, the hash algorithm *deterministically* outputs a hash $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$.
- $\mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \to \pi_i$: On input the hash key hk, an input $\mathbf{x} \in \Sigma^N$ and an index $i \in [L]$, the open algorithm outputs an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$.
- $\mathsf{Verify}(\mathsf{hk}, h, i, x_i, \pi_i) \to b$: On input the hash key hk, a hash value $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$, an index $i \in [N]$, a value $x_i \in \Sigma$, and an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$, the verification algorithm outputs a bit $b \in \{0,1\}$ indicating whether it accepts or rejects.

We require the following properties:

- **Correctness**: For all security parameters $\lambda \in \mathbb{N}$, all block sizes $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, all integers $N \le 2^\lambda$, all indices $i, i^* \in [N]$, and any $\mathbf{x} \in \Sigma^N$,

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, h, i, x_i, \pi_i) = 1 : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*); \\ h \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathbf{x}); \pi_i \leftarrow \mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \end{array}\right] = 1.$$

- **Index hiding:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the index hiding game $\mathsf{ExptIH}_{\mathcal{A}}(\lambda, b)$ as follows:
  1. Algorithm $\mathcal{A}(1^\lambda)$ chooses an integer N and two indices $i_0, i_1 \in [N]$.
  2. The challenger sets $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i_b)$, and gives hk to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

  We require that for all polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ and all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

- **Somewhere statistically binding:** We say that a hash key hk is statistically binding for an index $i^* \in [N]$ if there does not exist $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$, $x \neq x' \in \Sigma$, and π, π' where $\mathsf{Verify}(\mathsf{hk}, h, i^*, x, \pi) = 1 = \mathsf{Verify}(\mathsf{hk}, h, i^*, x', \pi')$. We require that for all polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ and all $N \le 2^\lambda$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $i \in [N]$,

$$\Pr[\mathsf{hk} \text{ is statistically binding for index } i : \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i)] = 1 - \mathsf{negl}(\lambda).$$

- **Succinctness:** The hash length ℓ_{hash}, and opening length ℓ_{open} are all fixed polynomials in the security parameter λ and the block size ℓ_{blk} (and independent of N).

Definition 6.2 (Somewhere Extractable Commitment [CJJ21b, adapted])**.** A somewhere extractable commitment scheme with block size ℓ_{blk} and locality L is a tuple of efficient algorithms $\Pi_{\mathsf{SECom}} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L) \to (\mathsf{crs}, \mathsf{vk})$: On input the security parameter $\lambda \in \mathbb{N}$, the block size ℓ_{blk}, the number of blocks N, and the locality parameter L, the setup algorithm outputs a common reference string crs and a verification key vk.
- $\mathsf{Commit}(\mathsf{crs}, \mathbf{v}) \to (c, \tau)$: On input the common reference string crs, and a vector $\mathbf{v} \in (\{0,1\}^{\ell_{\mathsf{blk}}})^N$, the commit algorithm outputs a commitment c and a state τ.
- $\mathsf{Open}(\mathsf{crs}, \tau, i) \to \pi_i$: On input the common reference string crs, the commitment state τ, and an index i, the open algorithm outputs a local opening π_i.
- $\mathsf{Verify}(\mathsf{vk}, c, i, v, \pi) \to b$: On input the verification key vk, the commitment c, an index $i \in [N]$, a block $v \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

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

- **Correctness:** For all security parameters λ, block sizes ℓ_{blk}, message lengths N, locality parameters L, messages $\mathbf{v} = (v_1, \ldots, v_N) \in (\{0,1\}^{\ell_{\mathsf{blk}}})^N$, and indices $i \in [N]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, c, i, v_i, \pi_i) = 1 : \begin{array}{c}(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L); \\ (c, \tau) \leftarrow \mathsf{Commit}(\mathsf{crs}, \mathbf{v}); \pi_i \leftarrow \mathsf{Open}(\mathsf{crs}, \tau, i)\end{array}\right] = 1.$$

- **Somewhere extractable:** There exists a pair of efficient algorithms $(\mathsf{TrapSetup}, \mathsf{Extract})$ with the following properties:
  - $\mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L, S) \to (\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{td})$: On input the security parameter λ, the block size ℓ_{blk}, the number of blocks N, the locality parameter L, and a set $S \subseteq [N]$, the trapdoor setup algorithm outputs a common reference string crs^*, verification key vk^*, and an extraction trapdoor td.
  - $\mathsf{Extract}(\mathsf{td}, c, i) \to \mathbf{v}$: On input the extraction trapdoor td, a commitment c, and an index $i \in [N]$, the extraction algorithm either outputs a block $\mathbf{v} \in \{0,1\}^{\ell_{\mathsf{blk}}}$ or a special symbol $\mathbf{v} = \perp$. The extraction algorithm is *deterministic*.

  We moreover require the following two properties:

  - **CRS indistinguishability:** For integers $\ell_{\mathsf{blk}}, N, L \in \mathbb{N}$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, define the CRS indistinguishability experiment $\mathsf{ExptCRS}_{\mathcal{A}}(\lambda, \ell_{\mathsf{blk}}, N, L, b)$ as follows:
    1. Algorithm $\mathcal{A}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L)$ chooses a set $S \subseteq [N]$ of size at most L.
    2. If $b = 0$, the challenger samples $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L)$. If $b = 1$, it samples $(\mathsf{crs}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L, S)$. It gives $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$.
    3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

    We require that for all efficient adversaries $\mathcal{A}$, all polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $N = N(\lambda)$, and $L = L(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

  - **Somewhere extractable in trapdoor mode:** For integers $\ell_{\mathsf{blk}}, N, L \in \mathbb{N}$ and an adversary $\mathcal{A}$, define the somewhere extractability game as follows:
    1. Algorithm $\mathcal{A}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L)$ chooses a set $S \subseteq [N]$ of size at most L.

2. The challenger samples $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L, S)$ and gives $(\mathsf{crs}^*, \mathsf{vk}^*)$ to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ outputs a commitment c, a set of blocks $\{v_i\}_{i \in S}$, and a set of openings $\{\pi_i\}_{i \in S}$.
4. The output of the experiment is $b = 1$ if there exists $i \in S$ such that $\mathsf{Verify}(\mathsf{vk}^*, c, i, v_i, \pi_i) = 1$ and $\mathsf{Extract}(\mathsf{td}, c, i) \neq v_i$. Otherwise, the output is $b = 0$.

We require that for all adversaries $\mathcal{A}$, all polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $N = N(\lambda)$, and $L = L(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the above experiment.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot, \cdot, \cdot, \cdot)$ such that for all $\lambda \in \mathbb{N}$, $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $N = N(\lambda)$, $L = L(\lambda)$, vectors $\mathbf{v} = (v_1, \ldots, v_N) \in (\{0,1\}^{\ell_{\mathsf{blk}}})^N$, indices $i \in [N]$, all pairs $(\mathsf{crs}, \mathsf{vk})$ in the support of $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L)$, all pairs (c, τ) in the support $\mathsf{Commit}(\mathsf{crs}, \mathbf{v})$, and all openings π_i in the support of $\mathsf{Open}(\mathsf{crs}, \tau, i)$, the following properties hold:
  - **Succinct verification key:** $|\mathsf{vk}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, L, \log N)$.
  - **Succinct commitment:** $|c| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, L, \log N)$.
  - **Succinct local opening:** $|\pi_i| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, L, \log N)$.
  - **Succinct verification:** The running time of $\mathsf{Verify}(\mathsf{vk}, c, i, v_i, \pi_i)$ is $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, L, \log N)$. This is implied by the previous properties. Namely, the length of the input to Verify is $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, L, \log N)$, succinct verification holds as long as the running time of Verify is polynomial in its input length (i.e., it is an efficient algorithm).

Remark 6.3 (Fixed Parameter Variants [OPWW15])**.** Definition 6.1 allows for a flexible input length N and block size ℓ_{blk}, and these parameters are provided as input to the Setup algorithm. As described in Okamoto et al. [OPWW15, §2], we can also consider variants of Definition 6.1 with a *fixed* input length N and/or a fixed block size ℓ_{blk}. Analogously, we can consider variants of Definition 6.2 with a fixed locality parameter L and/or a fixed block size ℓ_{blk}.

Remark 6.4 (Separating the Verification Key from CRS)**.** In the definition of somewhere extractable commitments of Choudhuri et al. [CJJ21b], Setup is required to output a single *succinct* CRS that is used by the Commit, Open, and Verify algorithms. In this work, we consider a relaxed notion where Setup outputs a common reference string crs for generating and opening commitments and a separate (but still public) verification key is used to check openings. Importantly, for the primary application to delegation for RAM programs [CJJ21b], it is necessary that the size of the verification key and the running time of the verification algorithm be *succinct*. Less critical is the size of the CRS: namely, if we combine a somewhere extractable commitment scheme with a long CRS (e.g., $|\mathsf{crs}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, L, N)$) with a BARG for index languages, then we obtain a delegation scheme for RAM programs where the CRS size is long (scales polynomially with the running time of the RAM program). However, both the *proof size* and the *verification cost* still scale *polylogarithmically* with the running time of the RAM program. This is conceptually similar to the notion of a preprocessing succinct argument for NP [Gro10, Lip13, BCCT13, GGPR13, BCI+13], where the CRS is long, but the online verification costs (as measured in the proof size and the verification complexity) is succinct.

Remark 6.5 (Extending the Block Size and Locality)**.** Let $\Pi^{(0)}_{\mathsf{SECom}}$ be a somewhere extractable commitment scheme with block size 1. We can extend this to obtain a somewhere extractable commitment scheme Π_{SECom} with arbitrary (polynomial) block size ℓ_{blk} by concatenating ℓ_{blk} copies of the base scheme $\Pi^{(0)}_{\mathsf{SECom}}$. Specifically, a commitment c to a vector $\mathbf{v} \in (\{0,1\}^{\ell_{\mathsf{blk}}})^N$ consists of ℓ_{blk} commitments $(c_1, \ldots, c_{\ell_{\mathsf{blk}}})$ under the base scheme, where the j^{th} commitment c_j is a commitment to the j^{th} bit of each block $(v_{1,j}, \ldots, v_{N,j})$. An opening to block $i \in [N]$ consists of openings $(\pi_1, \ldots, \pi_{\ell_{\mathsf{blk}}})$ where π_j is an opening of c_j to bit $v_{i,j}$. The size of the verification key, commitment, and opening increase by a factor of ℓ_{blk} over that of the base scheme, which satisfies the required succinctness requirements.

A similar approach suffices for extending a somewhere extractable commitment scheme with locality parameter 1 (and arbitrary block size) to one with arbitrary (polynomial) locality parameter L. Very briefly, the somewhere extractable commitment with locality parameter L consists of L *independent* copies of the base scheme. Let $(\mathsf{crs}_1, \mathsf{vk}_1), \ldots, (\mathsf{crs}_L, \mathsf{vk}_L)$ denote the common reference strings and verification keys associated with the L independent copies of the base scheme. A commitment to a vector $\mathbf{v} \in (\{0,1\}^{\ell_{\mathsf{blk}}})^n$ consists of L commitments $c_1, \ldots, c_L$ where c_i is a commitment to $\mathbf{v}$ with respect to $(\mathsf{crs}_i, \mathsf{vk}_i)$. To open the commitment $(c_1, \ldots, c_L)$, the committer provides L openings $\pi_1, \ldots, \pi_L$, and the verifier accepts only if all of the L copies accept. To sample a trapdoor CRS for

indices $j_1, \ldots, j_L \in [N]$, we sample $(\mathsf{crs}_i^*, \mathsf{vk}_i^*, \mathsf{td}_i) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, 1^L, \{j_i\})$ for each $i \in [L]$. Namely, the i^{th} commitment enables extraction of block j_i of the message. CRS indistinguishability and somewhere extractability follow by a standard hybrid argument. Extending from 1-locality to L-locality increases the length of the verification key, commitment, and local opening by a factor of L.

6.2 Somewhere Extractable Commitments from Pairings

We show how to construct a somewhere extractable commitment scheme with block size $\ell_{\mathsf{blk}} = 1$ and locality parameter $L = 1$ by adapting the techniques we used to construct a BARG (see Construction 4.5). We can extend to larger block sizes and locality parameters by concatenation (see Remark 6.5). In particular, the commitment scheme the prover uses to commit to the wire values naturally supports succinct local openings. Somewhere extractability in turn follows from a similar proof strategy as the proof of Theorem 4.7. We can view our construction as a non-hiding version of the Catalano-Fiore vector commitment scheme [CF13] (which also publishes cross-terms in the CRS to support succinct local openings) that satisfies a somewhere extractable property. The original Catalano-Fiore scheme does not support extraction on any index.

The one remaining issue is that the resulting verification key scales linearly with the length of the vector. However, we observe that verifying an opening to an index $i^* \in [N]$ only requires knowledge of a constant number of group elements from the verification key. We can then use the optimization suggested by Catalano and Fiore of moving the verification key into the proving key, and having the prover provide the verification component as part of the commitment opening. Of course, we now need to ensure robustness against a dishonest prover. The approach in Catalano and Fiore is to include signatures to authenticate the verification components, and the verifier would first check the signature before validating the commitment opening. In our setting, we require that the commitment be statistically binding (indeed, extractable) at a particular index; to realize this, we replace the signature with an SSB hash over the verification components. By sampling the SSB hash key to bind at index i^*, the prover is forced to provide the correct verification component for index i^*. We give the full construction and analysis below.

Construction 6.6 (Somewhere Extractable Commitment from Pairings)**.** Let $k \in \mathbb{N}$ and let $\Pi_{\mathsf{SSB}} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ be a somewhere statistically binding hash function. We construct a somewhere extractable commitment with a *fixed* block size $\ell_{\mathsf{blk}} = 1$ and a *fixed* locality parameter $L = 1$ (see Remark 6.3).

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the message length N, the setup algorithm does the following:
  - Run $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample matrices $\mathbf{M}, \hat{\mathbf{M}} \xleftarrow{\text{R}} \mathbb{Z}_p^{(k+1)\times k}$.
  - For each $i \in [N]$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{\text{R}} \mathbb{Z}_p^k$ and compute $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{a} \leftarrow \sum_{i\in[N]} \mathbf{a}_i$.
  - For each $i, j \in [N]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_p^{k\times k}$ and let $\mathbf{B}_{i,j} \leftarrow \mathbf{M}(\boldsymbol{\alpha}_i \hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{R}_{i,j}) \in \mathbb{Z}_p^{(k+1)\times k}$ and $\hat{\mathbf{B}}_{i,j} \leftarrow -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} \in \mathbb{Z}_p^{(k+1)\times k}$.
  - Let $\ell_{\mathsf{blk}}(\lambda)$ be a bound on the number of bits needed to represent an element of $\mathbb{G}_2^{k+1}$. Sample a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, 1)$ and compute $h \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, ([\hat{\mathbf{a}}_1]_2, \ldots, [\hat{\mathbf{a}}_N]_2))$.
  - Output the verification key $\mathsf{vk} = \big(\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1\big)$ and the common reference string $\mathsf{crs} = \big(\mathsf{vk}, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[N]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\big)$.
- $\mathsf{Commit}(\mathsf{crs}, \mathbf{v})$: On input $\mathsf{crs} = \big(\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[N]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\big)$ and a vector $\mathbf{v} = (v_1, \ldots, v_N) \in \{0,1\}^N$, the commit algorithm computes $[\mathbf{u}]_1 \leftarrow \sum_{i\in[N]} v_i [\mathbf{a}_i]_1$. It outputs the commitment $c = [\mathbf{u}]_1$ and the state $\tau = \mathbf{v}$).
- $\mathsf{Open}(\mathsf{crs}, \tau, i)$: On input $\mathsf{crs} = \big(\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[N]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\big)$, the state $\tau = \mathbf{v} \in \{0,1\}^N$, and the index $i \in [N]$, the open algorithm first computes $\pi_{\mathsf{SSB}} \leftarrow \mathsf{Open}(\mathsf{hk}, ([\hat{\mathbf{a}}_1]_2, \ldots, [\hat{\mathbf{a}}_N]_2), i)$. Next, it computes
$$[\mathbf{W}]_1 = \sum_{j\neq i} (v_j - v_i)[\mathbf{B}_{j,i}]_1 \quad \text{and} \quad [\hat{\mathbf{W}}]_2 = \sum_{j\neq i} (v_j - v_i)[\hat{\mathbf{B}}_{j,i}]_2.$$

It outputs the opening $\pi = ([\hat{\mathbf{a}}_i]_2, \pi_{\mathsf{SSB}}, [\mathbf{W}]_1, [\hat{\mathbf{W}}]_2)$.

- $\mathsf{Verify}(\mathsf{vk}, c, i, v, \pi)$: On input the verification key $\mathsf{vk} = (\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1)$, commitment $c = [\mathbf{u}]_1$, index $i \in [N]$, bit $v \in \{0,1\}$, and an opening $\pi = ([\tilde{\mathbf{a}}]_2, \pi_{\mathsf{SSB}}, [\mathbf{W}]_1, [\hat{\mathbf{W}}]_2)$, the verification algorithm accepts if the following two properties hold:
  - $\mathsf{SSB.Verify}(\mathsf{hk}, h, i, [\tilde{\mathbf{a}}]_2, \pi_{\mathsf{SSB}}) = 1$.
  - $[\mathbf{u}]_1 \cdot [\tilde{\mathbf{a}}^\mathsf{T}]_2 = \big(v[\mathbf{a}]_1 \cdot [\tilde{\mathbf{a}}^\mathsf{T}]_2\big) + \big([\mathbf{M}]_1 \cdot [\hat{\mathbf{W}}^\mathsf{T}]_2\big) + \big([\mathbf{W}]_1 \cdot [\hat{\mathbf{M}}^\mathsf{T}]_2\big)$.

Theorem 6.7 (Correctness)**.** *If Π_{SSB} is correct, then Construction 6.6 is correct.*

Proof. Fix a security parameter λ and message length N. Take any vector $\mathbf{v} = (v_1, \ldots, v_N) \in \{0,1\}^N$ and index $i^* \in [N]$. Let $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $(c, \tau) \leftarrow \mathsf{Commit}(\mathsf{crs}, \mathbf{v})$ and $\pi_{i^*} \leftarrow \mathsf{Open}(\mathsf{crs}, \tau, i^*)$. By construction, we can write

$$\mathsf{vk} = \big(\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1\big) \quad \text{and} \quad \mathsf{crs} = \big(\mathsf{vk}, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[N]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\big),$$

$c = [\mathbf{u}]_1$ and $\pi_{i^*} = ([\hat{\mathbf{a}}_i]_2, \pi_{\mathsf{SSB}}, [\mathbf{W}]_1, [\hat{\mathbf{W}}]_2)$. Consider each of the verification relations in $\mathsf{Verify}(\mathsf{vk}, c, i^*, v_{i^*}, \pi_{i^*})$:

- By construction, the hash key hk is generated using $\mathsf{SSB.Setup}$, h is a hash of $([\hat{\mathbf{a}}_1]_2, \ldots, [\hat{\mathbf{a}}_N]_2)$, and π_{SSB} is an opening of h to $[\hat{\mathbf{a}}_{i^*}]_2$ at index i^*. Correctness of Π_{SSB} implies that $\mathsf{SSB.Verify}(\mathsf{hk}, h, i^*, [\hat{\mathbf{a}}_{i^*}]_2, \pi_{\mathsf{SSB}}) = 1$.
- By construction, $\mathbf{u} = \sum_{i\in[N]} v_i[\mathbf{a}_i]_1$. Thus, we can write

$$\mathbf{u}\hat{\mathbf{a}}_{i^*}^\mathsf{T} = \sum_{i\in[N]} v_i\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T} = v_{i^*}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \sum_{i\neq i^*} v_i\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T}$$
$$v_{i^*}\mathbf{a}\hat{\mathbf{a}}_{i^*}^\mathsf{T} = \sum_{i\in[N]} v_{i^*}\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T} = v_{i^*}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \sum_{i\neq i^*} v_{i^*}\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T}$$

  Next, by the same calculation as Eq. (4.1) from the proof of Theorem 4.6, for all $i \neq j$,

$$\mathbf{M}\hat{\mathbf{B}}_{i,j}^\mathsf{T} + \mathbf{B}_{i,j}\hat{\mathbf{M}}^\mathsf{T} = -\mathbf{M}\mathbf{R}_{i,j}\hat{\mathbf{M}}^\mathsf{T} + \mathbf{M}(\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{R}_{i,j})\hat{\mathbf{M}}^\mathsf{T} = \mathbf{M}\boldsymbol{\alpha}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T}\hat{\mathbf{M}}^\mathsf{T} = \mathbf{a}_i\hat{\mathbf{a}}_j^\mathsf{T}.$$

  In particular, this means that

$$\mathbf{M}\hat{\mathbf{W}}^\mathsf{T} + \mathbf{W}\hat{\mathbf{M}}^\mathsf{T} = \sum_{i\neq i^*}(v_i - v_{i^*})(\mathbf{M}\hat{\mathbf{B}}_{i,i^*}^\mathsf{T} + \mathbf{B}_{i,i^*}\hat{\mathbf{M}}^\mathsf{T}) = \sum_{i\neq i^*}(v_i - v_{i^*})\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T}.$$

  Combining the above relations, we have

$$v_{i^*}\mathbf{a}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \mathbf{M}\hat{\mathbf{W}}^\mathsf{T} + \hat{\mathbf{W}}\hat{\mathbf{M}}^\mathsf{T} = v_{i^*}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \sum_{i\neq i^*}(v_{i^*} + v_i - v_{i^*})\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T} = v_{i^*}\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \sum_{i\neq i^*} v_i\mathbf{a}_i\hat{\mathbf{a}}_{i^*}^\mathsf{T} = \mathbf{u}\hat{\mathbf{a}}_{i^*}^\mathsf{T}.$$

  Thus, the verifier accepts. □

Theorem 6.8 (Somewhere Extractable)**.** *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen *and Π_{SSB} is a somewhere statistically binding hash function, then Construction 6.6 is somewhere extractable.*

Proof. We start by defining the trapdoor setup and extraction algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^N, i^*)$: On input the security parameter λ, message length N, and index $i^* \in [N]$ (recall that we are considering the special case of locality $L = 1$ so the set S contains just a single index i^*), the trapdoor setup algorithm samples the common reference string and verification key using the following procedure (we highlight the differences from Setup in green):
  - Run $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample matrices $\mathbf{M}, \hat{\mathbf{M}} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^{(k+1)\times k}$.

- For $i \neq i^*$, sample $\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i \xleftarrow{R} \mathbb{Z}_p^k$ and let $\mathbf{a}_i \leftarrow \mathbf{M}\boldsymbol{\alpha}_i$, $\hat{\mathbf{a}}_i \leftarrow \hat{\mathbf{M}}\hat{\boldsymbol{\alpha}}_i$. Let $\mathbf{0} \neq \mathbf{z} \in \mathbb{Z}_p^{k+1}$ be any non-zero vector such that $\mathbf{z}^\mathsf{T}\mathbf{M} = \mathbf{0}$. Since $\mathbf{M}$ has rank at most k, such a $\mathbf{z}$ always exists and can be efficiently computed.
- Sample $\mathbf{a}_{i^*}, \hat{\mathbf{a}}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$. Let $\mathbf{a} \leftarrow \sum_{i\in[N]} \mathbf{a}_i$.
- For each $i, j \in [N]$ where $i \neq j$, sample $\mathbf{R}_{i,j} \xleftarrow{R} \mathbb{Z}_p^{k\times k}$. Construct $\mathbf{B}_{i,j}$ and $\hat{\mathbf{B}}_{i,j}$ for $i \neq j$ as follows:

$$\mathbf{B}_{i,j} = \begin{cases} \mathbf{a}_i\hat{\boldsymbol{\alpha}}_j^\mathsf{T} + \mathbf{M}\mathbf{R}_{i,j} & j \neq i^* \\ \mathbf{M}\mathbf{R}_{i,j} & j = i^* \end{cases} \qquad \hat{\mathbf{B}}_{i,j} = \begin{cases} -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} & j \neq i^* \\ -\hat{\mathbf{M}}\mathbf{R}_{i,j}^\mathsf{T} + \hat{\mathbf{a}}_j\boldsymbol{\alpha}_i^\mathsf{T} & j = i^*. \end{cases}$$

- Let $\ell_{\mathsf{blk}}(\lambda)$ be a bound on the number of bits needed to represent an element of $\mathbb{G}_2^{k+1}$. Sample a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*)$ and compute $h \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, ([\hat{\mathbf{a}}_1]_2, \ldots, [\hat{\mathbf{a}}_N]_2))$.
- Output the verification key $\mathsf{vk}^* = (\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1)$, the common reference string $\mathsf{crs}^* = (\mathsf{vk}^*, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[N]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j})$, and the trapdoor $\mathsf{td} = (i^*, \mathbf{z})$.

- $\mathsf{Extract}(\mathsf{td}, c, i) \to v$: On input the extraction trapdoor $\mathsf{td} = (i^*, \mathbf{z})$, a commitment $c = [\mathbf{u}]_1$, and an index i, the extraction algorithm outputs $\perp$ if $i \neq i^*$. If $i = i^*$, then extraction algorithm outputs 0 if $\mathbf{z}^\mathsf{T}[\mathbf{u}]_1 = 0$ and 1 otherwise.

We now show the CRS indistinguishability and somewhere extractability properties.

Lemma 6.9 (CRS Indistinguishability)**.** *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen *and Π_{SSB} satisfies index hiding, then Construction 4.5 satisfies CRS indistinguishability.*

Proof. Take any message length $N = N(\lambda)$. We proceed via a hybrid argument:

- Hyb_0: This is the real distribution $\mathsf{ExptCRS}_{\mathcal{A}}(\lambda, N, 0)$. Specifically, at the beginning of the security game, the adversary $\mathcal{A}$ chooses an index $i^* \in [N]$. The challenger then samples $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and gives $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then 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 hash key hk using the procedure in TrapSetup: $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*)$. All of the other components of crs and vk are sampled as in Hyb_0.
- Hyb_2: This is the trapdoor distribution $\mathsf{ExptCRS}_{\mathcal{A}}(\lambda, N, 1)$. Namely, the challenger samples $\mathbf{a}_{i^*}, \hat{\mathbf{a}}_{i^*} \xleftarrow{R} \mathbb{Z}_p^{k+1}$ and defines matrices $\mathbf{B}_{i,j}, \hat{\mathbf{B}}_{i,j}$ according to the specification of TrapSetup.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of experiment $\mathsf{Hyb}_i(\mathcal{A})$ with algorithm $\mathcal{A}$. We now show that each adjacent pair of hybrid experiments are computationally indistinguishable.

Claim 6.10. *If Π_{SSB} satisfies index hiding, then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. This is immediate by index hiding since the only difference between Hyb_0 and Hyb_1 is that hk binds to index 0 in Hyb_0 and to index i^* in Hyb_1. More formally, suppose there exists an efficient algorithm $\mathcal{A}$ such that $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks index hiding of Π_{SSB} (for block size ℓ_{blk}):

1. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ to obtain an index $i^* \in [N]$. It sends indices 0 and i^* as its challenge pair to the index hiding challenger.
2. The index hiding challenger replies to $\mathcal{B}$ with a hash key hk. Algorithm $\mathcal{B}$ samples the other components of crs and vk exactly as described in Hyb_0 and Hyb_1. It gives crs and vk to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

By construction, if hk binds to index 0, then $\mathcal{B}$ perfectly simulates Hyb_0, and if hk binds to index i^*, then $\mathcal{B}$ perfectly simulates Hyb_1. Thus, algorithm $\mathcal{B}$ breaks index hiding with the same advantage ε. □

Claim 6.11. *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen*, then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. First, the hash key hk is identically distributed in the two experiments and independent of the group elements in the CRS and verification key. The hash value h is a deterministic function of hk and the group elements appearing in the CRS. Thus, it suffices to argue that the distribution of group elements in the CRS and verification key is computationally indistinguishable between Hyb_1 and Hyb_2. This now follows by the same argument as the proof of Lemma 4.8. In particular, the group elements in the CRS and verification key of Construction 6.6 are exactly the same as those in Construction 4.5; this is also true for the distribution of the trapdoor CRS and verification key of the two schemes. □

CRS indistinguishability now follows by a standard hybrid argument. □

Lemma 6.12 (Somewhere Extractable in Trapdoor Mode)**.** *If Π_{SSB} is correct and somewhere statistically binding, then Construction 6.6 satisfies extraction correctness.*

Proof. Take any polynomial $N = N(\lambda)$. Take any adversary $\mathcal{A}$ for the somewhere extractability game and let $i^* \in [N]$ be the index chosen by $\mathcal{A}$. Let $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^N, i^*)$. We write

$$\mathsf{vk}^* = \left(\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1\right) \quad \text{and} \quad \mathsf{crs}^* = \left(\mathsf{vk}^*, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[N]}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\right).$$

Let $c = [\mathbf{u}]_1$, $v \in \{0,1\}$, and $\pi = ([\tilde{\mathbf{a}}]_2, \pi_{\mathsf{SSB}}, [\mathbf{W}]_1, [\hat{\mathbf{W}}]_2)$ be the commitment, value, and opening, respectively, chosen by $\mathcal{A}$. Suppose $\mathsf{Verify}(\mathsf{vk}^*, c, i^*, v, \pi) = 1$. Let $v' \leftarrow \mathsf{Extract}(\mathsf{td}, c, i^*)$. We claim that $v = v'$. We first show that under the somewhere statistically binding property of Π_{SSB}, $\tilde{\mathbf{a}} = \hat{\mathbf{a}}_{i^*}$ with overwhelming property.

Claim 6.13. *Suppose Π_{SSB} is correct and somewhere statistically binding. Then, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, $\Pr[\tilde{\mathbf{a}} \neq \hat{\mathbf{a}}_{i^*}] = \mathsf{negl}(\lambda)$, where the probability is taken over the random coins of* TrapSetup.

Proof. Since $\mathsf{Verify}(\mathsf{vk}^*, c, i^*, v, \pi) = 1$, this means that $\mathsf{SSB.Verify}(\mathsf{hk}, h, i^*, [\tilde{\mathbf{a}}]_2, \pi_{\mathsf{SSB}}) = 1$. By construction of TrapSetup, hk is generated using SSB.Setup and moreover, h is the hash of $([\hat{\mathbf{a}}_1]_2, \ldots, [\hat{\mathbf{a}}_N]_2)$ under hk. Let $\pi_{i^*} \leftarrow \mathsf{SSB.Open}(\mathsf{hk}, ([\hat{\mathbf{a}}_1]_2, \ldots, [\hat{\mathbf{a}}_N]_2), i^*)$. Since Π_{SSB} is correct, this means that $\mathsf{SSB.Verify}(\mathsf{hk}, h, i^*, [\hat{\mathbf{a}}_{i^*}]_2, \pi_{i^*}) = 1$. Then, if $\tilde{\mathbf{a}} \neq \hat{\mathbf{a}}_{i^*}$, we conclude that hk is *not* statistically binding at index i^*. Since Π_{SSB} is somewhere statistically binding, this event can only happen with negligible probability. □

The rest of the proof now follows a similar structure as the proof of Lemma 4.12. In particular, the group elements in crs^* and vk^* are distributed exactly as in the trapdoor setup algorithm from the proof of Theorem 4.7. As demonstrated in Claim 6.13, $\tilde{\mathbf{a}} = \hat{\mathbf{a}}_{i^*}$ with overwhelming probability. Moreover, by Claim 4.13 (i), we can write $\mathbf{u} = \xi \mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}$ for some $\xi \in \mathbb{Z}_p$ and $\mathbf{t} \in \mathbb{Z}_p^k$. In addition, let $\boldsymbol{\beta} = \sum_{i\neq i^*} \boldsymbol{\alpha}_i$. Then $\mathbf{a} = \sum_{i\in[m]} \mathbf{a}_i = \mathbf{a}_{i^*} + \sum_{i\neq i^*} \mathbf{M}\boldsymbol{\alpha}_i = \mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta}$. Now, we write

$$\begin{aligned}
\mathbf{u}\tilde{\mathbf{a}}^\mathsf{T} &= \mathbf{u}\hat{\mathbf{a}}_{i^*}^\mathsf{T} = \xi \mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \mathbf{M}\mathbf{t}\hat{\mathbf{a}}_{i^*}^\mathsf{T} \\
\mathbf{a}\tilde{\mathbf{a}}^\mathsf{T} &= \mathbf{a}\hat{\mathbf{a}}_i^\mathsf{T} = (\mathbf{a}_{i^*} + \mathbf{M}\boldsymbol{\beta})\hat{\mathbf{a}}_{i^*}^\mathsf{T} = \mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} + \mathbf{M}\boldsymbol{\beta}\hat{\mathbf{a}}_{i^*}^\mathsf{T}.
\end{aligned}$$

We now consider the verification relations under the projection operator from Claim 4.13 (ii). By Claim 4.13 (iii), (iv), we can write

$$\underbrace{\mathsf{proj}(\mathbf{u}\tilde{\mathbf{a}}^\mathsf{T})}_{\xi\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}} = \underbrace{\mathsf{proj}(v\mathbf{a}\tilde{\mathbf{a}}^\mathsf{T})}_{v\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}} + \underbrace{\mathsf{proj}(\mathbf{M}\hat{\mathbf{W}}^\mathsf{T})}_{0} + \underbrace{\mathsf{proj}(\mathbf{W}\hat{\mathbf{M}}^\mathsf{T})}_{0}.$$

By Claim 4.13 (i), $\mathbf{a}_{i^*}\hat{\mathbf{a}}_{i^*}^\mathsf{T} \neq \mathbf{0}$ with overwhelming probability, so we conclude that $\xi = v$. This means that $\mathbf{u} = v\mathbf{a}_{i^*} + \mathbf{M}\mathbf{t}$. Consider now the value of v' output by $\mathsf{Extract}(\mathsf{td}, [\mathbf{u}]_1, i^*)$ where $\mathsf{td} = (i^*, \mathbf{z})$. By construction, $\mathbf{z} \neq \mathbf{0}$ and $\mathbf{z}^\mathsf{T}\mathbf{M} = \mathbf{0}$. Next, $\mathbf{a}_{i^*}$ is uniform over $\mathbb{Z}_p^{k+1}$ and independent of $\mathbf{z}$, so with overwhelming probability, $\mathbf{z}^\mathsf{T}\mathbf{a}_{i^*} \neq 0$. This means that

$$\mathbf{z}^\mathsf{T}\mathbf{u} = v\mathbf{z}^\mathsf{T}\mathbf{a}_{i^*} + \mathbf{z}^\mathsf{T}\mathbf{M}\mathbf{t} = v\mathbf{z}^\mathsf{T}\mathbf{a}_{i^*}.$$

Thus, $\mathbf{z}^\mathsf{T}\mathbf{u}$ is zero if and only if $v = 0$. By definition of Extract, $v' = v$, as required. □

Somewhere extractability now follows from Lemmas 6.9 and 6.12. □

Theorem 6.14 (Succinctness). *Let $k \in \mathbb{N}$ be a constant. If Π_{SSB} is succinct, then Construction 6.6 is succinct.*

Proof. Take any security parameter λ, message length N, vector $\mathbf{v} \in \{0,1\}^N$, and index $i \in [N]$. Suppose we sample $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $(c, \tau) \leftarrow \mathsf{Commit}(\mathsf{crs}, \mathbf{v})$ and $\pi_i \leftarrow \mathsf{Open}(\mathsf{crs}, \tau, i)$. By construction, we can write $\mathsf{vk} = (\mathcal{G}, \mathsf{hk}, h, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1)$, $c = [\mathbf{u}]_1$ and $\pi_i = ([\hat{\mathbf{a}}_i]_2, \pi_{\mathsf{SSB}}, [\mathbf{W}]_1, [\hat{\mathbf{W}}]_2)$. We consider each of the requirements:

- **Succinct verification key:** The description $\mathcal{G}$ output by $\mathsf{GroupGen}(1^\lambda)$ has length $\mathsf{poly}(\lambda)$. Moreover, the number of bits needed to encode elements of $\mathbb{G}_1, \mathbb{G}_2$ are also $\mathsf{poly}(\lambda)$. For constant k, the encodings $[\mathbf{M}]_1$ and $[\hat{\mathbf{M}}]_2$ and $[\mathbf{a}]_1$ each contain of a constant number of group elements, and can be represented using $\mathsf{poly}(\lambda)$ bits.

  Next, the hash key hk output by SSB.Hash has size $|\mathsf{hk}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$. As noted above, $\ell_{\mathsf{blk}} = \mathsf{poly}(\lambda)$ so $|\mathsf{hk}| = \mathsf{poly}(\lambda, \log N)$. By succinctness of Π_{SSB}, $|h| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}) = \mathsf{poly}(\lambda)$. Putting everything together, $|\mathsf{vk}| = \mathsf{poly}(\lambda, \log N)$, as required.
- **Succinct commitment:** The commitment $c = [\mathbf{u}]_1 \in \mathbb{G}_1^{k+1}$ consists of $k+1$ group elements. For constant k, this means $|c| = \mathsf{poly}(\lambda)$.
- **Succinct opening:** For constant k, the components $[\hat{\mathbf{a}}_i]_2$, $[\mathbf{W}]_1$, and $[\hat{\mathbf{W}}]_2$ in π_i contain a constant number of group elements: $k(k+1)$ elements in $\mathbb{G}_1$ and $(k+1)^2$ elements in $\mathbb{G}_2$. By succinctness of Π_{SSB}, $|\pi_{\mathsf{SSB}}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}) = \mathsf{poly}(\lambda)$. Thus, $|\pi_i| = \mathsf{poly}(\lambda)$.
- **Succinct verification:** Verify is an efficient algorithm (i.e., its running time is polynomial in its input length), so succinct verification follows by the previous properties. □

Combining Theorems 6.7, 6.8 and 6.14, we obtain the following corollary:

Corollary 6.15 (Somewhere Extractable Commitment). *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen *(for any constant $k \geq 1$), and Π_{SSB} is a somewhere statistically binding hash function, then Construction 6.6 is a somewhere extractable commitment scheme with block size 1, locality 1, and CRS size $N^2 \cdot \mathsf{poly}(\lambda, \log N)$, where N is the message length.*

6.3 Somewhere Extractable Commitments with a Short CRS

The size of the CRS in Construction 6.6 has size $N^2 \cdot \mathsf{poly}(\lambda, \log N)$, where N is the bit-length of the input. In this section, we show that a similar type of bootstrapping procedure as that described in Section 5 for the case of BARGs can be used to obtain a somewhere extractable commitment scheme with a CRS whose size is *sublinear* in N. Specifically, for any constant $\varepsilon > 0$, we construct a somewhere extractable commitment with CRS size $N^\varepsilon \cdot \mathsf{poly}(\lambda)$.

Similar to the bootstrapping procedure from Section 5, we start by describing a two-tiered construction. For a batch size B, we break the input vector $\mathbf{v} \in \{0,1\}^N$ into N/B blocks $\mathbf{v}_1, \ldots, \mathbf{v}_{N/B} \in \{0,1\}^B$, each of length B. Let c_i be a commitment to the i^{th} block $\mathbf{v}_i$. Next, we construct a commitment to the vector $(c_1, \ldots, c_{N/B})$ to obtain a commitment c_{top}. To open a commitment at a particular index $i \in [N]$, we first write $i = B(i_{\mathsf{top}} - 1) + i_{\mathsf{base}}$ where $i_{\mathsf{top}} \in [N/B]$ and $i_{\mathsf{base}} \in [B]$. Then, we open c_{top} to $c_{i_{\mathsf{top}}}$ (at index i_{top}) and open $c_{i_{\mathsf{top}}}$ at index i_{base}. It is not difficult to see that if the base commitment scheme satisfies succinctness, then the two-tiered scheme is also succinct. Moreover, since the commitments in the base scheme are succinct ($|c_i| = \mathsf{poly}(\lambda, \log B)$), the two-tiered scheme only needs to commit to vectors of length B and $N/B \cdot \mathsf{poly}(\lambda, \log B)$. By setting the batch size to $B = \sqrt{N}$, we effectively reduce the size of the CRS from $N^2 \cdot \mathsf{poly}(\lambda, \log N)$ to $N \cdot \mathsf{poly}(\lambda, \log N)$. By recursively composing (a constant number of times), we obtain a somewhere extractable commitment with CRS size N^ε for any constant $\varepsilon > 0$. We give the full construction below:

Construction 6.16 (Somewhere Extractable Commitment Bootstrapping). Let $B \in \mathbb{N}$ be a batch size parameter. Let $\Pi^{(0)}_{\mathsf{SECom}} = (\mathsf{SECom}_0.\mathsf{Setup}, \mathsf{SECom}_0.\mathsf{Commit}, \mathsf{SECom}_0.\mathsf{Open}, \mathsf{SECom}_0.\mathsf{Verify})$ be a somewhere extractable commitment scheme with locality $L = 1$. We construct a new somewhere extractable commitment scheme with locality $L = 1$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N)$: On input the security parameter λ, the block size ℓ_{blk}, and the number of blocks N, the setup algorithm does the following:
  - Sample $(\mathsf{crs}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{base}}) \leftarrow \mathsf{SECom}_0.\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^B)$.
  - Let $\ell_c = \ell_c(\lambda, \ell_{\mathsf{blk}}, B)$ be the length of the commitments output by $\mathsf{SECom}_0.\mathsf{Commit}(\mathsf{crs}_{\mathsf{base}}, \cdot)$.
  - Sample $(\mathsf{crs}_{\mathsf{top}}, \mathsf{vk}_{\mathsf{top}}) \leftarrow \mathsf{SECom}_0.\mathsf{Setup}(1^\lambda, 1^{\ell_c}, 1^{N/B})$.
  - Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$ and $\mathsf{vk} = (\mathsf{vk}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{top}})$.

  We will require that $B \le N$.
- $\mathsf{Commit}(\mathsf{crs}, \mathbf{v})$: On input $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$ and a vector $\mathbf{v} = (v_1, \ldots, v_N)$, the commit algorithm proceeds as follows:
  - For each $i \in [N/B]$, compute a commitment $(c_i, \tau_i) \leftarrow \mathsf{SECom}_0.\mathsf{Commit}(\mathsf{crs}_{\mathsf{base}}, (v_{(i-1)B+1}, \ldots, v_{iB}))$.
  - Compute $(c_{\mathsf{top}}, \tau_{\mathsf{top}}) \leftarrow \mathsf{SECom}_0.\mathsf{Commit}(\mathsf{crs}_{\mathsf{top}}, (c_1, \ldots, c_{N/B}))$.
  - Output the commitment $c = c_{\mathsf{top}}$ and the state $\tau = (c_1, \ldots, c_{N/B}, \tau_1, \ldots, \tau_{N/B}, \tau_{\mathsf{top}})$.
- $\mathsf{Open}(\mathsf{crs}, \tau, i)$: On input $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$, a state $\tau = (c_1, \ldots, c_{N/B}, \tau_1, \ldots, \tau_{N/B}, \tau_{\mathsf{top}})$, and an index $i = B(i_{\mathsf{top}} - 1) + i_{\mathsf{base}}$ where $i_{\mathsf{top}} \in [N/B]$ and $i_{\mathsf{base}} \in [B]$, the open algorithm computes openings $\pi_{\mathsf{top}} \leftarrow \mathsf{SECom}_0.\mathsf{Open}(\mathsf{crs}_{\mathsf{top}}, \tau_{\mathsf{top}}, i_{\mathsf{top}})$ and $\pi_{\mathsf{base}} \leftarrow \mathsf{SECom}_0.\mathsf{Open}(\mathsf{crs}_{\mathsf{base}}, \tau_{i_{\mathsf{top}}}, i_{\mathsf{base}})$ and outputs $\pi = (c_{i_{\mathsf{top}}}, \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$.
- $\mathsf{Verify}(\mathsf{vk}, c, i, v, \pi)$: On input the verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{top}})$, a commitment $c = c_{\mathsf{top}}$, an index $i \in [N]$, a value $v \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and a proof $\pi = (c', \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$, the verification algorithm writes $i = B(i_{\mathsf{top}} - 1) + i_{\mathsf{base}}$ where $i_{\mathsf{top}} \in [N/B]$ and $i_{\mathsf{base}} \in [B]$. The algorithm accepts (with output 1) if all of the following properties hold:
  - $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}_{\mathsf{top}}, c_{\mathsf{top}}, i_{\mathsf{top}}, c', \pi_{\mathsf{top}}) = 1$; and
  - $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}_{\mathsf{base}}, c', i_{\mathsf{base}}, v, \pi_{\mathsf{base}}) = 1$.

  Otherwise, the verification algorithm outputs 0.

Theorem 6.17 (Correctness). *If $\Pi^{(0)}_{\mathsf{SECom}}$ is correct, then Construction 6.6 is correct.*

Proof. Correctness follows by construction. Concretely, take any security parameter $\lambda \in \mathbb{N}$ and polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $N = N(\lambda)$. Take any vector $\mathbf{v} = (v_1, \ldots, v_N) \in (\{0,1\}^{\ell_{\mathsf{blk}}})^N$ and index $i \in [N]$. Write $i = B(i_{\mathsf{top}} - 1) + i_{\mathsf{base}}$ where $i_{\mathsf{top}} \in [N/B]$ and $i_{\mathsf{base}} \in [B]$. Let $(\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N)$, $(c, \tau) \leftarrow \mathsf{Commit}(\mathsf{crs}, \mathbf{v})$, $\pi \leftarrow \mathsf{Open}(\mathsf{crs}, \tau, i)$. We can write $\mathsf{crs} = (\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$, $\mathsf{vk} = (\mathsf{vk}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{top}})$, $\tau = (c_1, \ldots, c_{N/B}, \tau_1, \ldots, \tau_{N/B}, \tau_{\mathsf{top}})$, and $\pi = (c_{i_{\mathsf{top}}}, \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$. We show that both verification relations in $\mathsf{Verify}(\mathsf{vk}, c, i, v_i, \pi)$ hold:

- First, c_{top} is a commitment to $(c_1, \ldots, c_{N/B})$ with respect to $(\mathsf{crs}_{\mathsf{top}}, \mathsf{vk}_{\mathsf{top}})$ and π_{top} is an opening of c_{top} to index i_{top}. By correctness of $\Pi^{(0)}_{\mathsf{SECom}}$, $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}_{\mathsf{top}}, c_{\mathsf{top}}, i_{\mathsf{top}}, c_{i_{\mathsf{top}}}, \pi_{\mathsf{top}}) = 1$.
- Next, $c_{i_{\mathsf{top}}}$ is a commitment to $(v_{(i_{\mathsf{top}}-1)B+1}, \ldots, v_{i_{\mathsf{top}}B})$ with respect to $(\mathsf{crs}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{base}})$ and π_{base} is an opening of $c_{i_{\mathsf{top}}}$ to index i_{base}. By definition, $v_{(i_{\mathsf{top}}-1)B+i_{\mathsf{base}}} = v_i$, so $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}_{\mathsf{base}}, c_{\mathsf{base}}, i_{\mathsf{base}}, v_i, \pi_{\mathsf{base}}) = 1$. □

Theorem 6.18 (Somewhere Extractable). *If $\Pi^{(0)}_{\mathsf{SECom}}$ is somewhere extractable, then Construction 6.16 is somewhere extractable.*

Proof. We start by defining the trapdoor setup and extraction algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, i^*)$: On input the security parameter λ, block length ℓ_{blk}, the number of blocks N, and an index $i^* \in [N]$, the trapdoor setup algorithm writes $i^* = B(i^*_{\mathsf{top}} - 1) + i^*_{\mathsf{base}}$ and then samples the following:
  - $(\mathsf{crs}^*_{\mathsf{base}}, \mathsf{vk}^*_{\mathsf{base}}, \mathsf{td}_{\mathsf{base}}) \leftarrow \mathsf{SECom}_0.\mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^B, i^*_{\mathsf{base}})$; and

– $(\mathsf{crs}^*_{\mathsf{top}}, \mathsf{vk}^*_{\mathsf{top}}, \mathsf{td}_{\mathsf{top}}) \leftarrow \mathsf{SECom}_0.\mathsf{TrapSetup}(1^\lambda, 1^{\ell_c}, 1^{N/B}, i^*_{\mathsf{top}})$.

It outputs $\mathsf{crs}^* = (\mathsf{crs}^*_{\mathsf{base}}, \mathsf{crs}^*_{\mathsf{top}})$, $\mathsf{vk}^* = (\mathsf{vk}^*_{\mathsf{base}}, \mathsf{vk}^*_{\mathsf{top}})$, and $\mathsf{td} = (i^*, \mathsf{td}_{\mathsf{base}}, \mathsf{td}_{\mathsf{top}})$.

- $\mathsf{Extract}(\mathsf{td}, c, i)$: On input the trapdoor $\mathsf{td} = (i^*, \mathsf{td}_{\mathsf{base}}, \mathsf{td}_{\mathsf{top}})$, a commitment c and an index $i \in [N]$, if $i \neq i^*$, the extraction algorithm outputs $\perp$. Otherwise, it computes $c_{\mathsf{base}} \leftarrow \mathsf{SECom}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{top}}, c, i^*_{\mathsf{top}})$ and outputs $v \leftarrow \mathsf{SECom}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{base}}, c_{\mathsf{base}}, i^*_{\mathsf{base}})$ where $i^* = B(i^*_{\mathsf{top}} - 1) + i^*_{\mathsf{base}}$.

We now show that the CRS indistinguishability and somewhere extractability properties hold:

Lemma 6.19 (CRS Indistinguishability). *If $\Pi^{(0)}_{\mathsf{SECom}}$ satisfies CRS indistinguishability, then Construction 6.16 also satisfies CRS indistinguishability.*

Proof. This follows by a standard hybrid argument. First, $(\mathsf{crs}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{base}})$ and $(\mathsf{crs}_{\mathsf{top}}, \mathsf{vk}_{\mathsf{top}})$ are sampled independently in Construction 6.16, as is the case in the trapdoor setup algorithm. In the real setup, $(\mathsf{crs}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{base}})$ is sampled by computing $\mathsf{SECom}_0.\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^B)$ and in the trapdoor setup algorithm, they are sampled by computing $\mathsf{SECom}_0.\mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^B, i^*_{\mathsf{base}})$. These two distributions are computationally indistinguishable by CRS indistinguishability of $\Pi^{(0)}_{\mathsf{SECom}}$. A similar argument applies to the distribution of $(\mathsf{crs}_{\mathsf{top}}, \mathsf{vk}_{\mathsf{top}})$. □

Lemma 6.20 (Somewhere Extractable in Trapdoor Mode). *If $\Pi^{(0)}_{\mathsf{SECom}}$ is somewhere extractable in trapdoor mode, then Construction 6.16 is somewhere extractable in trapdoor mode.*

Proof. Fix polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ and $N = N(\lambda)$. Let $i^* \in [N]$ be the index chosen by the adversary. Let $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^N, i^*)$. We write $\mathsf{vk}^* = (\mathsf{vk}^*_{\mathsf{base}}, \mathsf{vk}^*_{\mathsf{top}})$ and $i^* = B(i^*_{\mathsf{top}} - 1) + i^*_{\mathsf{base}}$. Take any commitment c, string $v \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and proof $\pi = (c', \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$. Suppose that $\mathsf{Verify}(\mathsf{vk}^*, c, i^*, v, \pi) = 1$. Let $c_{\mathsf{base}} \leftarrow \mathsf{SECom}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{top}}, c, i^*_{\mathsf{top}})$ and $v' \leftarrow \mathsf{SECom}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{base}}, c_{\mathsf{base}}, i^*_{\mathsf{base}})$. It suffices to show that with overwhelming probability, $v = v'$. Since $\mathsf{Verify}(\mathsf{vk}^*, c, i^*, v, \pi)$ outputs 1, we have that $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{top}}, c, i^*_{\mathsf{top}}, c', \pi_{\mathsf{top}}) = 1$ and $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{base}}, c', i^*_{\mathsf{base}}, v, \pi_{\mathsf{base}}) = 1$.

Claim 6.21. *If $\Pi^{(0)}_{\mathsf{SECom}}$ is somewhere extractable in trapdoor mode, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[c' = c_{\mathsf{base}}] = 1 - \mathsf{negl}(\lambda)$.*

Proof. Suppose there is an adversary $\mathcal{A}$ that outputs $c, v, \pi = (c', \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$ where $c' \neq c_{\mathsf{base}}$ and $\mathsf{Verify}(\mathsf{vk}^*, c, i^*, v, \pi) = 1$. By construction of Verify, this means that $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{top}}, c, i^*_{\mathsf{top}}, c', \pi_{\mathsf{top}}) = 1$. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks the somewhere extractability property of $\Pi^{(0)}_{\mathsf{SECom}}$ with the same advantage:

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ to obtain an index $i^* \in [N]$. It writes $i^* = B(i^*_{\mathsf{top}} - 1) + i^*_{\mathsf{base}}$, gives i^*_{top} to its challenger, and receives $(\mathsf{crs}^*_{\mathsf{top}}, \mathsf{vk}^*_{\mathsf{top}})$ from its challenger.

2. Algorithm $\mathcal{B}$ samples $(\mathsf{crs}^*_{\mathsf{base}}, \mathsf{vk}^*_{\mathsf{base}}, \mathsf{td}_{\mathsf{base}}) \leftarrow \mathsf{SECom}_0.\mathsf{TrapSetup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^B, i^*_{\mathsf{base}})$. It constructs and gives $\mathsf{crs}^* = (\mathsf{crs}^*_{\mathsf{base}}, \mathsf{crs}^*_{\mathsf{top}})$ and $\mathsf{vk}^* = (\mathsf{vk}^*_{\mathsf{base}}, \mathsf{vk}^*_{\mathsf{top}})$ to $\mathcal{A}$.

3. Algorithm $\mathcal{A}$ outputs a commitment c, a string $v \in \{0,1\}^{\ell_{\mathsf{blk}}}$ and a proof $\pi = (c', \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$. Algorithm $\mathcal{B}$ outputs c, c', and π_{top}.

By construction, algorithm $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in the somewhere extractability game. Thus, if $\mathcal{A}$ succeeds with advantage ε, then with the same probability ε, $\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{top}}, c, i^*_{\mathsf{top}}, c', \pi_{\mathsf{top}}) = 1$ and $c' \neq c_{\mathsf{base}}$ where $c_{\mathsf{base}} \leftarrow \mathsf{SECom}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{top}}, c, i^*_{\mathsf{top}})$. Thus, $\mathcal{B}$ breaks somewhere extractability of $\Pi^{(0)}_{\mathsf{SECom}}$ with advantage ε. □

Claim 6.22. *If $\Pi^{(0)}_{\mathsf{SECom}}$ is somewhere extractable in trapdoor mode, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[v = v'] = 1 - \mathsf{negl}(\lambda)$.*

Proof. By assumption, $\mathsf{SECom}_0.\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{base}}, c', i^*_{\mathsf{base}}, v, \pi_{\mathsf{base}}) = 1$ and $v' \leftarrow \mathsf{SECom}_0.\mathsf{Extract}(\mathsf{td}_{\mathsf{base}}, c_{\mathsf{base}}, i^*_{\mathsf{base}})$. By Claim 6.21, $c' = c_{\mathsf{base}}$ with overwhelming probability. Since $(\mathsf{crs}^*_{\mathsf{base}}, \mathsf{vk}^*_{\mathsf{base}}\mathsf{td}_{\mathsf{base}})$ is sampled using $\mathsf{SECom}_0.\mathsf{TrapSetup}$ with index i^*_{base}, we can appeal to a similar argument as used in the proof of Claim 6.21 to conclude that $v = v'$ with probability $1 - \mathsf{negl}(\lambda)$. □

Combining Claims 6.21 and 6.22, we have that the extracted block $v' \in \{0,1\}^{\ell_{\mathsf{blk}}}$ matches the claimed block $v \in \{0,1\}^{\ell_{\mathsf{blk}}}$ with overwhelming probability and the claim follows. □

The claim now follows by combining Lemmas 6.19 and 6.20. □

Theorem 6.23 (Succinctness). *Suppose $\Pi^{(0)}_{\mathsf{SECom}}$ is a succinct somewhere extractable commitment with CRS size $\ell_0(\lambda, \ell_{\mathsf{blk}}, N) = N^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$ for some constant $d \in \mathbb{N}$. Then Construction 6.16 is a succinct somewhere extractable commitment with CRS size*

$$\ell(\lambda, \ell_{\mathsf{blk}}, N, B) = B^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}) + (N/B)^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N).$$

Proof. We show that each of the properties are satisfied:

- **CRS size:** The CRS in Construction 6.16 consists of two common reference strings $(\mathsf{crs}_{\mathsf{base}}, \mathsf{crs}_{\mathsf{top}})$ for $\Pi^{(0)}_{\mathsf{SECom}}$. The size of $\mathsf{crs}_{\mathsf{base}}$ is $\ell_0(\lambda, \ell_{\mathsf{blk}}, B)$ and the size of $\mathsf{crs}_{\mathsf{top}}$ is $\ell_0(\lambda, \ell_c, N/B)$. By succinctness of $\Pi^{(0)}_{\mathsf{SECom}}$, we have that $\ell_c(\lambda, \ell_{\mathsf{blk}}, B) = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log B)$. Thus,
$$\ell(\lambda, \ell_{\mathsf{blk}}, N, B) = B^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}) + (N/B)^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N).$$ □
- **Succinct verification key:** The verification key vk in Construction 6.16 consists of two verification keys $(\mathsf{vk}_{\mathsf{base}}, \mathsf{vk}_{\mathsf{top}})$ for $\Pi^{(0)}_{\mathsf{SECom}}$. By succinctness of $\Pi^{(0)}_{\mathsf{SECom}}$, we have that $|\mathsf{vk}_{\mathsf{base}}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log B)$ and $|\mathsf{vk}_{\mathsf{top}}| = \mathsf{poly}(\lambda, \ell_c, \log N/B) = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$. Thus, $|\mathsf{vk}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$.
- **Succinct commitment:** The commitment consists of a single commitment under $\mathsf{crs}_{\mathsf{top}}$, which has size $\mathsf{poly}(\lambda, \ell_c, \log N/B) = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$.
- **Succinct opening:** An opening $(c_{\mathsf{base}}, \pi_{\mathsf{top}}, \pi_{\mathsf{base}})$ consists of a commitment c under $\mathsf{crs}_{\mathsf{base}}$ and two openings π_{top} and π_{base} under $\mathsf{crs}_{\mathsf{top}}$ and $\mathsf{crs}_{\mathsf{base}}$, respectively. By succinctness of $\Pi^{(0)}_{\mathsf{SECom}}$, $|c_{\mathsf{base}}|, |\pi_{\mathsf{base}}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log B)$ and $|\pi_{\mathsf{top}}| = \mathsf{poly}(\lambda, \ell_c, \log(N/B)) = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$. The overall opening size is then $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$.
- **Succinct verification:** The verification algorithm reduces to two invocations of the verification algorithm for $\Pi^{(0)}_{\mathsf{SECom}}$ which run in time $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log B)$ and $\mathsf{poly}(\lambda, \ell_c, \log(N/B))$. The total running time is thus $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$.

Corollary 6.24 (Somewhere Extractable Commitment with Short CRS). *Suppose there exists a somewhere extractable commitment with locality 1 and commitment size $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, N)$, where ℓ_{blk} is the block size and N is the number of blocks. Then, for every constant $\varepsilon > 0$, there exists a somewhere extractable commitment with locality 1 and a CRS of size $N^\varepsilon \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$.*

Proof. Let $\Pi^{(0)}_{\mathsf{SECom}}$ be a somewhere extractable commitment scheme with locality 1 and a CRS of size bounded by $N^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$ for some constant $d \in \mathbb{N}$. Let $k = \lceil \log(2d/\varepsilon) \rceil \in \mathbb{N}$. For $i \in [k]$, let $\Pi^{(i)}_{\mathsf{SECom}}$ be the somewhere extractable commitment with locality 1 formed by applying Construction 6.16 to $\Pi^{(i-1)}_{\mathsf{SECom}}$ with $B = \sqrt{N}$. Let ℓ_i denote the length of the CRS in $\Pi^{(i)}_{\mathsf{SECom}}$. Since $\ell_0(\lambda, \ell_{\mathsf{blk}}, N) = N^d \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$, we can inductively apply Theorem 6.23 to write

$$\ell_i(\lambda, \ell_{\mathsf{blk}}, N) = N^{d/2^i} \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N).$$

Substituting $k = \lceil \log(2d/\varepsilon) \rceil$ into the above, we have that

$$\ell_k(\lambda, \ell_{\mathsf{blk}}, N) = N^{\varepsilon/2} \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N) < N^\varepsilon \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}),$$

since $2d/\varepsilon$ is a constant. The other succinctness requirements are preserved since we compose a *constant* number of times. □

Corollary 6.25 (Somewhere Extractable Commitment with Short CRS). *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen *(for any constant $k \geq 1$), and if there exists a somewhere statistically binding hash function, then for every constant $\varepsilon > 0$, there exists a somewhere extractable commitment scheme with locality 1 and CRS size $N^\varepsilon \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$ where ℓ_{blk} is the block size and N is the number of blocks in the input.*

Proof. Follows by instantiating Corollary 6.24 with Corollary 6.15 (along with Remark 6.5). □

6.4 Delegation for RAM Programs

In this section, we recall the definition of delegation for RAM machines from the works of [KPY19, CJJ21b]. We refer to Kalai et al. [KPY19, Remark 3.6] for comparisons with earlier definitions of RAM delegation [KP16, BHK17]. Our description here is adapted from that in [KPY19]. A RAM machine $\mathcal{R}$ with word size ℓ is modeled as a deterministic machine with random access to a memory of size 2^ℓ bits and a local state of size $O(\ell)$. On each step of the RAM computation, the machine either reads or writes to a single word in memory and then updates its local state. We refer to the combination of the machine's local state and the memory as its configuration cf. For ease of exposition, we assume that the machine has no input or output other than its initial memory and local state configuration, and moreover, we set the word size $\ell = \lambda$ to the security parameter. For a RAM machine $\mathcal{R}$, we define the language $\mathcal{L}_{\mathcal{R}}$ as

$$\mathcal{L}_{\mathcal{R}} := \{(\ell, \mathsf{cf}, \mathsf{cf}', T) \mid \mathcal{R} \text{ with word size } \ell \text{ transitions from cf to cf}' \text{ in } T \text{ steps}\}.$$

Definition 6.26 (Delegation for RAM Programs [KPY19, CJJ21b, adapted])**.** A publicly-verifiable non-interactive delegation scheme for a RAM program $\mathcal{R}$ with setup time $T_S = T_S(\lambda, T)$ and proof length $\ell_\pi = \ell_\pi(\lambda, T)$ is a tuple of efficient algorithms $\Pi_{\mathsf{RAM}} = (\mathsf{Setup}, \mathsf{Digest}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^T) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{dk})$: On input the security parameter λ, a time bound T, the setup algorithm outputs a prover key pk, a verification key vk, and a digest key dk.
- $\mathsf{Digest}(\mathsf{dk}, \mathsf{cf}) \to \mathsf{h}$: On input the digest key dk and a configuration cf, the digest algorithm outputs a hash h. This algorithm is deterministic.
- $\mathsf{Prove}(\mathsf{pk}, \mathsf{cf}, \mathsf{cf}') \to \pi$: On input the prover key pk, an initial configuration cf and a final configuration cf′, the prove algorithm outputs a proof π. This algorithm is deterministic.
- $\mathsf{Verify}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \pi) \to b$: On input the verification key vk, a pair of digests h, h′, and a proof π, the verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.

We require that Π_{RAM} satisfy the following properties:

- **Completeness:** For every $\lambda, T \in \mathbb{N}$ where $T \leq 2^\lambda$ and $\mathsf{cf}, \mathsf{cf}' \in \{0,1\}^*$ where $(\lambda, \mathsf{cf}, \mathsf{cf}', T) \in \mathcal{L}_{\mathcal{R}}$,

  $$\Pr[\mathsf{Verify}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \pi) = 1] = 1,$$

  where $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$, $\mathsf{h} \leftarrow \mathsf{Digest}(\mathsf{dk}, \mathsf{cf})$, $\mathsf{h}' \leftarrow \mathsf{Digest}(\mathsf{dk}, \mathsf{cf}')$, and $\pi \leftarrow \mathsf{Prove}(\mathsf{pk}, \mathsf{cf}, \mathsf{cf}')$.
- **Efficiency:** In the completeness experiment above, we require the following hold:
  - The setup algorithm runs in time $T_S(\lambda, T)$.
  - The digest algorithm on configuration cf runs in time $|\mathsf{cf}| \cdot \mathsf{poly}(\lambda)$ and outputs a digest of size λ.
  - The prover runs in time $\mathsf{poly}(\lambda, T, |\mathsf{cf}|)$ and outputs a proof of length $\ell_\pi(\lambda, T)$.
  - The verifier runs in time $O(\ell_\pi) + \mathsf{poly}(\lambda)$.
- **Collision Resistance:** For every efficient adversary $\mathcal{A}$ and every polynomial $T = T(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

  $$\Pr\left[\mathsf{cf} \neq \mathsf{cf}' \wedge \mathsf{Digest}(\mathsf{dk}, \mathsf{cf}) = \mathsf{Digest}(\mathsf{dk}, \mathsf{cf}') : \begin{array}{c}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T);\\ (\mathsf{cf}, \mathsf{cf}') \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}).\end{array}\right] = \mathsf{negl}(\lambda).$$
- **Soundness:** For every efficient adversary $\mathcal{A}$ and every polynomial $T = T(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

  $$\Pr\left[\begin{array}{c}\mathsf{Verify}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \pi) = 1 \wedge\\ (\lambda, \mathsf{cf}, \mathsf{cf}', T) \in \mathcal{L}_{\mathcal{R}} \wedge\\ \mathsf{h} = \mathsf{Digest}(\mathsf{dk}, \mathsf{cf}) \wedge\\ \mathsf{h}' \neq \mathsf{Digest}(\mathsf{dk}, \mathsf{cf}')\end{array} : \begin{array}{c}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T);\\ (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})\end{array}\right] = \mathsf{negl}(\lambda).$$

Construction and instantiation. Choudhuri et al. [CJJ21b] showed how to construct a delegation scheme for RAM programs from a variant of a somewhere extractable commitment scheme that supports "no-signaling" extraction [GZ21] together with a non-interactive batch argument for an index language (Remark 2.10). As shown by González and Zacharakis (see also [CJJ21b, Theorem 13]), a no-signaling somewhere extractable commitment scheme with locality L can be constructed using L copies of a vanilla somewhat extractable commitment scheme with locality 1 (e.g., from Corollary 6.25). We summarize this instantiation in the following theorem:

Theorem 6.27 (Delegation for RAM Programs [CJJ21b])**.** *Suppose there exists a somewhere extractable commitment scheme with block size $\ell_{\mathsf{blk}} = 1$, locality $L = 1$, and a batch non-interactive argument for index languages. Then, there exists a delegation scheme for RAM programs with setup time $T_S = \mathsf{poly}(\lambda, T)$ and proof length $\ell_\pi = \mathsf{poly}(\lambda, \log T)$.*[9] *Moreover, the size of the digest key is $\mathsf{poly}(\lambda)$ and the size of the proving key is $(|\mathsf{crs}_{\mathsf{indexBARG}}| + |\mathsf{crs}_{\mathsf{SECom}}|) \cdot \mathsf{poly}(\lambda)$, where $\mathsf{crs}_{\mathsf{indexBARG}}$ denotes the length of the CRS for the index BARG (with $m = \mathsf{poly}(T)$ instances and $s = \mathsf{poly}(\lambda)$-size circuits) and $\mathsf{crs}_{\mathsf{SECom}}$ denotes the lengths of the CRS for the somewhere extractable commitment scheme (with message length $N = \mathsf{poly}(\lambda, T)$).*

We can instantiate Theorem 6.27 with our batch non-interactive argument for index languages (Corollary 5.10 and Remark 2.10) together with our somewhere extractable commitment scheme (Corollary 6.25). This yields a delegation scheme for RAM programs from the k-Lin assumption over asymmetric prime-order groups in conjunction with an SSB hash function. We can moreover instantiate the SSB hash function using the DDH-based construction of Okamoto et al. [OPWW15], which yields a delegation scheme for RAM programs from the 1-Lin (i.e., SXDH) assumption on prime-order pairing groups. We state these corollaries formally below:

Corollary 6.28 (Delegation for RAM Programs)**.** *If the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen *(for any constant $k \geq 1$), and there exists a somewhere statistically binding hash function, then for every constant $\varepsilon > 0$, there exists a delegation scheme for RAM programs with setup time $T_S = \mathsf{poly}(\lambda, T)$, proof length $\ell_\pi = \mathsf{poly}(\lambda, \log T)$, digest key size $\mathsf{poly}(\lambda)$, and proving key size $T^\varepsilon \cdot \mathsf{poly}(\lambda)$.*

Theorem 6.29 (SSB Hash Functions from DDH [OPWW15])**.** *Suppose the* DDH *assumption holds with respect to a group generator* GroupGen*. Then, there exists a SSB hash function for any polynomial block length $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$.*

Corollary 6.30 (Delegation for RAM Programs from SXDH)**.** *If the* SXDH *assumption holds with respect to* GroupGen*, then for every constant $\varepsilon > 0$, there exists a delegation scheme for RAM programs with setup time $T_S = \mathsf{poly}(\lambda, T)$, proof length $\ell_\pi = \mathsf{poly}(\lambda, \log T)$, digest key size $\mathsf{poly}(\lambda)$, and proving key size $T^\varepsilon \cdot \mathsf{poly}(\lambda)$.*

7 Aggregate Signatures from BARGs

In this section, we describe the straightforward approach of constructing aggregate signatures from BARGs for NP, and show that we can argue security so long as the BARG is a somewhere argument of knowledge. Importantly, security does *not* require that the BARG be fully extractable. We start by recalling the definition of a standard digital signature scheme and an aggregate signature scheme:

Definition 7.1 (Digital Signature)**.** A digital signature scheme with message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{Sig}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ with the following properties:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$: On input the security parameter λ, the key-generation algorithm outputs a signing key sk and a verification key vk.
- $\mathsf{Sign}(\mathsf{sk}, \mu) \to \sigma$: On input the signing key sk and a message $\mu \in \mathcal{M}$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, \mu, \sigma) \to b$: On input the verification key vk, a message $\mu \in \mathcal{M}$, and a signature σ, the verification algorithm outputs a bit $b \in \{0, 1\}$.

[9] Technically, the construction also requires a collision-resistant hash function, but this is implied by a somewhere extractable commitment.

Moreover, the above algorithms should satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$ and messages $\mu \in \mathcal{M}$,

$$\Pr[\mathsf{Verify}(\mathsf{vk}, \mu, \sigma) = 1 : (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda); \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mu)] = 1.$$

- **Unforgeability:** Define the signature unforgeability game between an adversary $\mathcal{A}$ and a challenger as follows:
  - The challenger samples $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and gives vk to $\mathcal{A}$.
  - The adversary can now make signing queries on messages $\mu \in \mathcal{M}$ of its choosing. On each query μ, the challenger replies with $\mathsf{Sign}(\mathsf{sk}, \mu)$.
  - At the end of the game, the adversary outputs a message-signature pair (μ^*, σ^*). The output of the game is 1 if $\mathsf{Verify}(\mathsf{vk}, \mu^*, \sigma^*) = 1$ and the adversary did not make a signing query on μ^*. Otherwise, the output is 0.

  We say Π_{Sig} is unforgeable if for all efficient adversaries, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the above unforgeability game.

Definition 7.2 (Aggregate Signature [BGLS03, adapted])**.** A bounded aggregate signature scheme with message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{AggSig}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^m) \to \mathsf{pp}$: On input the security parameter λ and an aggregation bound m, the setup algorithm outputs the public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{vk})$: On input the public parameters pp, the key-generation algorithm outputs a signing key sk and a verification key vk.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mu) \to \sigma$: On input the public parameters pp, the signing key sk, and a message $\mu \in \mathcal{M}$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, \mu, \sigma) \to b$: On input the public parameters pp, the verification key vk, a message $\mu \in \mathcal{M}$, and a signature σ, the verification algorithm outputs a bit $b \in \{0, 1\}$.
- $\mathsf{Aggregate}(\mathsf{pp}, \{(\mathsf{vk}_i, \mu_i, \sigma_i)\}_{i \in [T]}) \to \sigma_{\mathsf{agg}}$: On input the public parameters pp, and a collection of up to $T \le m$ verification keys vk_i, messages μ_i, and signatures σ_i, the aggregation algorithm outputs an aggregate signature σ_{agg}.
- $\mathsf{AggVerify}(\mathsf{pp}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_T), (\mu_1, \ldots, \mu_T), \sigma_{\mathsf{agg}}) \to b$: On input the public parameters pp, a collection of $T \le m$ verification keys vk_i and messages μ_i, and an aggregate signature σ_{agg}, the aggregate verification algorithm outputs a bit $b \in \{0, 1\}$.

Moreover, the above algorithms should satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all values $m \in \mathbb{N}$, all messages $\mu \in \mathcal{M}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, \mu, \sigma) = 1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m); \\ (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mu) \end{array}\right] = 1.$$

  In addition, for all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^m)$ and all collections $\{(\mathsf{vk}_i, \mu_i, \sigma_i)\}_{i \in [T]}$ where $T \le m$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_i, \mu_i, \sigma_i) = 1$ for all $i \in [T]$,

$$\Pr\left[\mathsf{AggVerify}(\mathsf{pp}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_T), (\mu_1, \ldots, \mu_T), \sigma_{\mathsf{agg}}) = 1 : \sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \{(\mathsf{vk}_i, \mu_i, \sigma_i)\}_{i \in [T]})\right] = 1.$$

- **Efficiency:** There exists a fixed polynomial $\mathsf{poly}(\cdot, \cdot)$ such that in the completeness experiment above, the size of the aggregate signature σ_{agg} satisfies $|\sigma_{\mathsf{agg}}| = \mathsf{poly}(\lambda, \log T)$.

- **Unforgeability:** Define the signature unforgeability game between an adversary $\mathcal{A}$ and a challenger as follows:
  - The challenger samples $\mathsf{pp} \leftarrow \mathcal{A}(1^\lambda, 1^m)$ and $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and gives pp and vk^* to $\mathcal{A}$.
  - The adversary can now make signing queries on messages $\mu \in \mathcal{M}$ of its choosing. On each query μ, the challenger replies with $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, \mu)$.
  - At the end of the game the adversary outputs a tuple of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_T)$, a tuple of messages $(\mu_1, \ldots, \mu_T)$ with $T \le m$, and a signature σ^*.
  - The output of the game is 1 if there exists an index $i^* \in [T]$ where $\mathsf{vk}_{i^*} = \mathsf{vk}^*$, algorithm $\mathcal{A}$ did not make a signing query on μ_{i^*}, and $\mathsf{AggVerify}(\mathsf{pp}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_T), (\mu_1, \ldots, \mu_T), \sigma^*) = 1$. Otherwise, the output is 0.

  Then, Π_{AggSig} is unforgeable if for all efficient adversaries $\mathcal{A}$ and all polynomials $m = m(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the above unforgeability game.

Construction 7.3 (Aggregate Signature from BARGs for NP)**.** Let $\Pi_{\mathsf{Sig}} = (\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a digital signature scheme, and let $\Pi_{\mathsf{BARG}} = (\mathsf{BARG.Setup}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$ be a BARG for NP. We require that Π_{BARG} supports proving and verifying a variable number T of instances provided that $T \le m$ where m is the bound on the total number of instances (see Remark 3.11). We construct a bounded aggregate signature scheme as follows:

- $\mathsf{Setup}(1^\lambda, 1^m)$: On input the security parameter λ and the aggregation bound m, let $s = s(\lambda)$ be the size of the circuit that computes $\mathsf{Sig.Verify}$. Sample $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^m, 1^s)$ and output $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$, output $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mu)$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$, the signing key sk, and the message $\mu \in \mathcal{M}$, output $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, \mu)$.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, \mu, \sigma)$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$, the verification key vk, the message $\mu \in \mathcal{M}$, and the signature σ, output $\mathsf{Sig.Verify}(\mathsf{vk}, \mu, \sigma)$.
- $\mathsf{Aggregate}(\mathsf{pp}, \{(\mathsf{vk}_i, \mu_i, \sigma_i)\}_{i \in [T]})$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$ and a collection of tuples $\{(\mathsf{vk}_i, \mu_i, \sigma_i)\}_{i \in [T]}$, the aggregation algorithm computes

  $$\pi \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), (\sigma_1, \ldots, \sigma_T)),$$

  where C_{Ver} is the Boolean circuit that computes $C_{\mathsf{Ver}}((\mathsf{vk}, \mu), \sigma) := \mathsf{Sig.Verify}(\mathsf{vk}, \mu, \sigma)$. The aggregated signature is the proof $\sigma_{\mathsf{agg}} = \pi$.
- $\mathsf{AggVerify}(\mathsf{pp}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_T), (\mu_1, \ldots, \mu_T), \sigma_{\mathsf{agg}})$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$, verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_T$, messages $\mu_1, \ldots, \mu_T \in \mathcal{M}$, and a signature σ_{agg}, output

  $$\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma_{\mathsf{agg}}).$$

Theorem 7.4 (Completeness)**.** *If Π_{Sig} is correct and Π_{BARG} is complete, then Construction 7.3 is correct.*

Proof. Follows by construction. □

Theorem 7.5 (Efficiency)**.** *If Π_{BARG} is succinct, then Construction 7.3 is efficient.*

Proof. The aggregate signature in Construction 7.3 is a BARG proof. Succinctness of the BARG ensures that $|\sigma_{\mathsf{agg}}| \le \mathsf{poly}(\lambda, \log m, s) = \mathsf{poly}(\lambda, \log m)$, since $s = s(\lambda)$ is the size of the verification circuit C_{Ver}. □

Theorem 7.6 (Unforgeability)**.** *If Π_{BARG} is a somewhere argument of knowledge and Π_{Sig} is unforgeable, then Construction 7.3 is unforgeable.*

Proof. We proceed using a hybrid argument:

- Hyb_0: This is the real signature unforgeability game:
  - At the beginning of the game, the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^m, 1^s)$ and sets $\mathsf{pp} = (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$. It also samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$, and gives pp, vk to the adversary $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can then make signing queries on messages $\mu \in \mathcal{M}$ and the challenger replies with $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, \mu)$.
  - At the end of the game the adversary outputs a tuple of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_T)$, a tuple of messages $(\mu_1, \ldots, \mu_T)$ with $T \le m$, and a signature σ^*.
  - The output of the experiment is 1 if there exists an index $i^* \in [T]$ where $\mathsf{vk}_{i^*} = \mathsf{vk}^*$, algorithm $\mathcal{A}$ did not make a signing query on μ_{i^*}, and $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*) = 1$. Otherwise, the output is 0.
- Hyb_1: In this experiment, the challenger starts by guessing an index $j^* \xleftarrow{\mathrm{R}} [m]$. The rest of the experiment then proceeds as in Hyb_0. After the adversary outputs $(\mathsf{vk}_1, \ldots, \mathsf{vk}_T)$, $(\mu_1, \ldots, \mu_T)$ and σ^*, the output of the experiment is 1 if $\mathsf{vk}_{j^*} = \mathsf{vk}^*$, algorithm $\mathcal{A}$ did not make a signing query on μ_{j^*}, and
$$\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*) = 1.$$
  Otherwise, the output is 0.
- Hyb_2: Same as Hyb_1, except the challenger uses the BARG trapdoor sampling algorithm to sample $\mathsf{crs}_{\mathsf{BARG}}$. In particular, after sampling $j^* \xleftarrow{\mathrm{R}} [m]$, the challenger samples $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{BARG.TrapSetup}(1^\lambda, 1^m, 1^s, j^*)$. Everything else is the same as in Hyb_1.
- Hyb_3: Same as Hyb_2 except at the end of the experiment, the challenger additionally computes
$$\hat{\sigma} \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*).$$
  If $C_{\mathsf{Ver}}(\mathsf{vk}_{j^*}, \mu_{j^*}, \hat{\sigma}) \neq 1$, then the output of the game is 0. Otherwise, the output is the same as in Hyb_2.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of experiment Hyb_i with adversary $\mathcal{A}$. Our goal is to show that for all efficient adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Lemma 7.7. *For all adversaries $\mathcal{A}$, we have that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \ge \frac{1}{m} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.

Proof. By construction, the views of the adversary in Hyb_0 and Hyb_1 are identical. The only difference is in how the output of the experiment is computed. Suppose $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \varepsilon$. Then, with probability ε, algorithm $\mathcal{A}$ outputs $(\mathsf{vk}_1, \ldots, \mathsf{vk}_T)$, $(\mu_1, \ldots, \mu_T)$ and σ^* where there exists an index $i^* \in [T]$ satisfying the listed properties with probability at least ε. This is also the case in Hyb_1. Here, if $j^* = i^*$, then the output in $\mathsf{Hyb}_1(\mathcal{A})$ is also 1. Since j^* is uniform, this happens with probability at least ε/m and the lemma holds. □

Lemma 7.8. *If Π_{BARG} is a somewhere argument of knowledge (specifically, the CRS indistinguishability property holds), then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*
$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. This is immediate by CRS indistinguishability. Namely, the only difference between Hyb_1 and Hyb_2 is that the challenger samples $\mathsf{crs}_{\mathsf{BARG}}$ using $\mathsf{BARG.Setup}(1^\lambda, 1^m, 1^s)$ in Hyb_1 and $\mathsf{BARG.TrapSetup}(1^\lambda, 1^m, 1^s, j^*)$ in Hyb_2. By CRS indistinguishability, these two distributions are computationally indistinguishable. Moreover, the output bit in Hyb_1 and Hyb_2 can be efficiently computed from $\mathsf{crs}_{\mathsf{BARG}}$ and the adversary's output. □

Lemma 7.9. *If Π_{BARG} is a somewhere argument of knowledge (specifically, the extractability in trapdoor mode property holds), then for all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*
$$\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. The only difference between Hyb_2 and Hyb_3 is the extra check the challenger performs in Hyb_3. Namely, in order for Hyb_2 to output 1, but Hyb_3 to output 0, it must be the case that

- $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*) = 1$; and
- $C_{\mathsf{Ver}}(\mathsf{vk}_{j^*}, \mu_{j^*}, \hat{\sigma}) \neq 1$ where $\hat{\sigma} \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*)$.

In Hyb_2 and Hyb_3, $\mathsf{crs}_{\mathsf{BARG}}$ is sampled using $\mathsf{BARG.TrapSetup}(1^\lambda, 1^m, 1^s, j^*)$, so any adversary $\mathcal{A}$ that produces an output that successfully triggers both of the above conditions with advantage ε also breaks somewhere extractability in trapdoor mode property with identical advantage. □

Lemma 7.10. *If Π_{Sig} is unforgeable, then for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists an efficient algorithm $\mathcal{A}$ where $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ that breaks unforgeability of Π_{Sig}:

1. Algorithm $\mathcal{B}$ receives the verification key vk^* from its challenger.
2. Algorithm $\mathcal{B}$ starts by sampling $j^* \xleftarrow{\mathrm{R}} [m]$ and $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{BARG.TrapSetup}(1^\lambda, 1^m, 1^s, j^*)$. It sets $\mathsf{pp} \leftarrow (1^\lambda, \mathsf{crs}_{\mathsf{BARG}})$ and gives pp to $\mathcal{A}$.
3. Whenever algorithm $\mathcal{A}$ makes a signing query on a message $\mu \in \mathcal{M}$, algorithm $\mathcal{B}$ makes a signing query on μ and obtains a signature σ. It replies to $\mathcal{A}$ with the signature σ.
4. At the end of the game, algorithm $\mathcal{A}$ outputs $(\mathsf{vk}_1, \ldots, \mathsf{vk}_T)$, $(\mu_1, \ldots, \mu_T)$ and σ^*. Algorithm $\mathcal{B}$ checks that $\mathsf{vk}_{j^*} = \mathsf{vk}^*$, algorithm $\mathcal{A}$ did *not* issue a signing query on μ_{j^*}, and that

$$\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*) = 1.$$

   If any checks do not pass, algorithm $\mathcal{B}$ aborts with output $\perp$. Otherwise, it computes

$$\hat{\sigma} \leftarrow \mathsf{BARG.Extract}(\mathsf{td}_{\mathsf{BARG}}, C_{\mathsf{Ver}}, ((\mathsf{vk}_1, \mu_1), \ldots, (\mathsf{vk}_T, \mu_T)), \sigma^*)$$

   and outputs $\mu_{j^*}, \hat{\sigma}$ as its forgery.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_3 for $\mathcal{A}$. Thus, with probability at least ε, algorithm $\mathcal{A}$ outputs $(\mathsf{vk}_1, \ldots, \mathsf{vk}_T)$, $(\mu_1, \ldots, \mu_T)$ and σ^* where $\mathsf{vk}_{j^*} = \mathsf{vk}^*$, the adversary never queried the signing oracle on μ_{j^*}, and $C_{\mathsf{Ver}}(\mathsf{vk}_{j^*}, \mu_{j^*}, \hat{\sigma}) = 1$. Since C_{Ver} is the verification circuit, this means that $\hat{\sigma}$ is a valid signature on μ_{j^*}, and so algorithm $\mathcal{B}$ succeeds with the same advantage ε. □

By Lemmas 7.8 to 7.10, we have that for all efficient adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. By Lemma 7.7, this means that $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq m \cdot \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$ since $m = \mathsf{poly}(\lambda)$. □

Corollary 7.11 (Bounded Aggregate Signature from Pairings)**.** *For any constant $k \geq 1$, if the k-Lin assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to a prime-order group generator* GroupGen *(or alternatively, if the subgroup decision assumption holds with respect to a composite-order group generator* CompGroupGen*), then for all constants $\varepsilon > 0$, there exists an bounded aggregate signature scheme with public parameter size $m^\varepsilon \cdot \mathsf{poly}(\lambda)$, where m is the aggregation bound.*

Acknowledgments

B. Waters is supported by NSF CNS-1908611, a Simons Investigator award, and the Packard Foundation Fellowship. D. J. Wu is supported by NSF CNS-1917414, CNS-2045180, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM CCS*, 2010.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018, 2018.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, 2004.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4), 2017.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*, 2013.

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

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC*, 2014.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, 2003.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, 2005.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, 2017.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In *EUROCRYPT*, 2017.

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

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *PKC*, 2013.

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

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT*, 2020.

[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.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT*, 2020.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, 2012.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge L. Villar. An algebraic framework for Diffie-Hellman assumptions. In *CRYPTO*, 2013.

[FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In *CRYPTO*, 2013.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *EUROCRYPT*, 2010.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, 2008.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.

[GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *PKC*, 2006.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.

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

[GZ21] Alonso González and Alexandros Zacharakis. Succinct publicly verifiable computation. In *TCC*, 2021.

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

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*, 2007.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *EUROCRYPT*, 2015.

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

[HW18] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In *EUROCRYPT*, 2018.

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In *EUROCRYPT*, 2021.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In *STOC*, 2021.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *TCC*, 2016.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, 2019.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *STOC*, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *STOC*, 2014.

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

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *FOCS*, 1990.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *ASIACRYPT*, 2013.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT*, 2004.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, 2006.

[LP21] Helger Lipmaa and Kateryna Pavlyk. Gentry-Wichs is tight: a falsifiable non-adaptively sound SNARG. In *ASIACRYPT*, 2021.

[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.

[Mic95] Silvio Micali. Computationally-sound proofs. In *Proceedings of the Annual European Summer Meeting of the Association of Symbolic Logic*, 1995.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, 2003.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT*, 2015.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, 2013.

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

[RR20] Guy N. Rothblum and Ron D. Rothblum. Batch verification and proofs of proximity with polylog overhead. In *TCC*, 2020.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, 2016.

[RRR18] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Efficient batch verification for UP. In *CCC*, 2018.

[RS09] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *ISA*, 2009.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO*, 2020.

[Sha90] Adi Shamir. IP=PSPACE. In *FOCS*, 1990.

[Sha07] Hovav Shacham. A Cramer-Shoup encryption scheme from the linear assumption and from progressively weaker linear variants. *IACR Cryptol. ePrint Arch.*, 2007.

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

[Wic22] Daniel Wichs, 2022. Personal communication.

A A More General View: BARGs with Fixed Wires

As discussed in Section 1.2.2, it is straightforward to generalize our BARG constructions (Constructions 3.3 and 4.5) to achieve better efficiency when the statements $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ admit a more compact representation. For instance, when using BARGs to construct delegation schemes [CJJ21b, KVZ21], the underlying statements indeed have a succinct description. In our setting, we show how to achieve better efficiency when some of the bits of the statements $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are *a priori* fixed.

Notation. For a bit string $\mathbf{x} \in \{0,1\}^n$ and a set $S \subseteq [n]$, we write $\mathbf{x}|_S \in \{0,1\}^{|S|}$ to denote the subset of bits indexed by S: $\mathbf{x}|_S := (x_i \mid i \in S) \in \{0,1\}^{|S|}$

Definition A.1 (Batch Circuit Satisfiability with Constraints)**.** Let $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be a Boolean circuit and $m \in \mathbb{N}$ be the number of instances. A *fixed-wire constraint* φ for C is a pair $(j, \boldsymbol{\sigma})$ where $j \in [n]$ is an index and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m) \in \{0,1\}^m$ is an assignment. We say that a tuple of statements $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ satisfies φ if $\mathbf{x}_{i,j} = \sigma_i$ for all $i \in [m]$; we denote this by writing $\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_m) = 1$. We will say that a set of constraints Φ is *admissible* if it contains at *most* one constraint for each index j. Unless otherwise noted, we will only consider admissible sets of constraints. For an admissible set of constraints Φ, we define

$$\mathcal{L}_{\mathsf{BatchCSAT},m,\Phi} = \{(C, \mathbf{x}_1, \ldots, \mathbf{x}_m) \mid (C, \mathbf{x}_1, \ldots, \mathbf{x}_m) \in \mathcal{L}_{\mathsf{BatchCSAT},m} \text{ and } \forall \varphi \in \Phi : \varphi(\mathbf{x}_1, \ldots, \mathbf{x}_m) = 1\}$$

to be the batch circuit satisfiability language with fixed-wire constraints. For a collection of fixed-wire constraints $\Phi = \{(j, \boldsymbol{\sigma}) \mid j \in [n], \boldsymbol{\sigma} \in \{0,1\}^m\}$, we define $A_\Phi := \{\boldsymbol{\sigma} \in \{0,1\}^m \mid \exists j : (j, \boldsymbol{\sigma}) \in \Phi\}$ to be the set of assignments associated with Φ and we define $S_\Phi := \{j \in [n] \mid \exists \boldsymbol{\sigma} : (j, \boldsymbol{\sigma}) \in \Phi\}$ to be the set of indices fixed by Φ.

Definition A.2 (Batch Argument with Fixed Wires)**.** A non-interactive batch argument for circuit satisfiability with fixed-wire constraints is a tuple of three efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^m, 1^s, A) \to (\mathsf{crs}, \mathsf{vk}, \mathsf{D})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $m \in \mathbb{N}$, a bound on the circuit size $s \in \mathbb{N}$, and a collection of fixed-wire assignments $A \subseteq \{0,1\}^m$, the setup algorithm outputs a common reference string crs, a verification key vk, and a dictionary $\mathsf{D}\colon A \to \mathcal{E}$ that associates each $\boldsymbol{\sigma} \in A$ with an encoding from some set $\mathcal{E}$ of encodings.
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{D}, C, \Phi, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m)) \to \pi$: On input the common reference string crs, a dictionary D, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a set of fixed-wire constraints Φ, statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{vk}, C, (\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S), \{(i, \mathsf{enc}_i)\}_{i \in [n] \setminus S}, \pi) \to b$: On input the verification key vk, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a collection of statements $\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S \in \{0,1\}^{|S|}$ restricted to some set $S \subseteq [n]$, and a collection of encodings (i, enc_i)[10] for the indices $[n] \setminus S$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

We say Π_{BARG} is a non-interactive batch argument with *fully fixed wires* if Verify only takes vk, C, $\{(i, \mathsf{enc}_i)\}_{i \in [n]}$, and π as input (i.e., the set S of non-fixed wires is $S = \emptyset$).

Definition A.3 (Completeness)**.** A BARG with fixed-wire constraints $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is complete if for all $\lambda, m, s \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, all sets of fixed-wire assignments $A \subseteq \{0,1\}^m$, all admissible sets of fixed-wire constraints Φ whose assignments $A_\Phi \subseteq A$ are contained

[10] Note that we allow the same encoding to be used across multiple indices. For instance, it may be the case that $\mathsf{enc}_i = \mathsf{enc}_j$ with $i \neq j$.

in A, all statements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, all witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$ where $C(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all $i \in [m]$ and $\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_m) = 1$ for all $\varphi \in \Phi$,

$$\Pr[\mathsf{Verify}(\mathsf{vk}, C, (\mathbf{x}_1|_{\bar{S}_\Phi}, \ldots, \mathbf{x}_m|_{\bar{S}_\Phi}), \{(j, \mathsf{D}[\boldsymbol{\sigma}_j])\}_{(j,\boldsymbol{\sigma}_j)\in\Phi}, \pi) = 1,$$

where $S_\Phi \subseteq [n]$ is the set of indices fixed by Φ, $\bar{S}_\Phi = [n] \setminus S_\Phi$ is the set of unfixed indices, and we sample $(\mathsf{crs}, \mathsf{vk}, \mathsf{D}) \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s, A)$, and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{D}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$.

Definition A.4 (Somewhere Argument of Knowledge)**.** A BARG with fixed-wire constraints $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a somewhere argument of knowledge if there exists a pair of efficient algorithms $(\mathsf{TrapSetup}, \mathsf{Extract})$ with the following properties:

- $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*, A) \to (\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $m \in \mathbb{N}$, the size of the circuit $s \in \mathbb{N}$, an index $i^* \in [m]$, and a collection of fixed-wire assignments $A \subseteq \{0,1\}^m$, the trapdoor setup algorithm outputs a common reference string crs^*, a verification key vk^*, a dictionary D^*, and an extraction trapdoor td.
- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S), \{(i, \mathsf{enc}_i^*)\}_{i\in[n]\setminus S}, \pi) \to \mathbf{w}^*$: On input the trapdoor td, a collection of statements $\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S \in \{0,1\}^{|S|}$ restricted to some set $S \subseteq [n]$, a collection of encodings (i, enc_i^*) for the indices $[n] \setminus S$, and a proof π, the extraction algorithm outputs a witness $\mathbf{w}^* \in \{0,1\}^h$. The extraction algorithm is deterministic.

We require $(\mathsf{TrapSetup}, \mathsf{Extract})$ to satisfy the following two properties:

- **CRS indistinguishability:** For integers $m \in \mathbb{N}$, $s \in \mathbb{N}$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, define the CRS indistinguishability experiment $\mathsf{ExptCRS}_{\mathcal{A}}(\lambda, m, s, b)$ as follows:
  1. Algorithm $\mathcal{A}(1^\lambda, 1^m, 1^s)$ outputs a collection of fixed-wire assignments $A \subseteq \{0,1\}^m$ and an index $i^* \in [m]$.
  2. If $b = 0$, the challenger computes and gives $(\mathsf{crs}, \mathsf{vk}, \mathsf{D}) \leftarrow \mathsf{Setup}(1^\lambda, 1^m, 1^s, A)$ to $\mathcal{A}$. If $b = 1$, the challenger computes $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*, A)$ and gives $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*)$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  Then, Π_{BARG} satisfies CRS indistinguishability if for every efficient adversary $\mathcal{A}$ and every polynomial $m = m(\lambda)$, $s = s(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

- **Somewhere extractable in trapdoor mode:** Define the somewhere extractable security game between an adversary $\mathcal{A}$ and a challenger as follows:
  - Algorithm $\mathcal{A}(1^\lambda, 1^m, 1^s)$ outputs an index $i^* \in [m]$ and a set of fixed-wire assignments $A \subseteq \{0,1\}^m$.
  - The challenger samples $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*, A)$ and gives $\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*$ 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, an admissible set of fixed-wire constraints Φ whose assignments $A_\Phi \subseteq A$ are contained in A, a set of statements $\hat{\mathbf{x}}_1|_{\bar{S}_\Phi}, \ldots, \hat{\mathbf{x}}_m|_{\bar{S}_\Phi} \in \{0,1\}^{|\bar{S}_\Phi|}$ restricted to the set of indices $\bar{S}_\Phi = [n] \setminus S_\Phi$ not fixed by Φ, and a proof π.
  - The challenger computes $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, \mathsf{D}^*, \{(j, \mathsf{D}^*[\boldsymbol{\sigma}_j])\}_{(j,\boldsymbol{\sigma}_j)\in\Phi}, \pi)$
  - For $i \in [m]$, define $\mathbf{x}_i|_{\bar{S}_\Phi} = \hat{\mathbf{x}}_i|_{\bar{S}_\Phi}$. For indices $j \in S_\Phi$ fixed by Φ, let $x_{i,j} = a_{j,i}$ where $(j, (a_{j,1}, \ldots, a_{j,m})) \in \Phi$. The output of the game is $b = 1$ if the following conditions hold:
    - $\mathsf{Verify}(\mathsf{vk}^*, C, (\hat{\mathbf{x}}_1|_{\bar{S}_\Phi}, \ldots, \hat{\mathbf{x}}_m|_{\bar{S}_\Phi}), \{(j, \mathsf{D}^*[\boldsymbol{\sigma}_j])\}_{(j,\boldsymbol{\sigma}_j)\in\Phi}, \pi) = 1$
    - $C(\mathbf{x}_{i^*}, \mathbf{w}^*) \neq 1$.

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

In the case of a BARG with fully fixed wires, we additionally restrict the adversary $\mathcal{A}$ to choosing admissible sets of fixed-wire constraint Φ where $S_\Phi = [n]$ (i.e., Φ fixes *all* input wires to C).

Definition A.5 (Succinctness)**.** A BARG with fixed-wire constraints $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is succinct if there exists a fixed polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $\lambda, m, s \in \mathbb{N}$, all sets of fixed-wire assignments $A \subseteq \{0,1\}^m$, all $(\mathsf{crs}, \mathsf{vk}, \mathsf{D})$ in the support of $\mathsf{Setup}(1^\lambda, 1^m, 1^s, A)$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, and all sets of fixed-wire constraints Φ, the following properties hold:

- **Succinct proofs:** The proof π output by $\mathsf{Prove}(\mathsf{crs}, \mathsf{D}, C, \cdot, \cdot, \cdot)$ satisfies $|\pi| \le \mathsf{poly}(\lambda, \log m, s)$.
- **Succinct verification key:** We require

$$|\mathsf{vk}| + |\{\mathsf{D}[\sigma]\}_{\sigma \in A}| \le \mathsf{poly}(\lambda, m, n) + \mathsf{poly}(\lambda, \log m, s) + \mathsf{poly}(\lambda, \log m, |A|).$$

  In the setting of fully fixed wires, we require

$$|\mathsf{vk}| + |\{\mathsf{D}[\sigma]\}_{\sigma \in A}| \le \mathsf{poly}(\lambda, \log m, s) + \mathsf{poly}(\lambda, \log m, |A|).$$

- **Succinct verification:** The running time of $\mathsf{Verify}(\mathsf{vk}, C, (\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S), \{(i, \mathsf{enc}_i)\}_{i \in [n] \setminus S})$ is bounded by $\mathsf{poly}(\lambda, m, |S|) + \mathsf{poly}(\lambda, \log m, s)$. In the setting of fully fixed wires, $S = \varnothing$ and this requirement collapses to Verify needing to run in time $\mathsf{poly}(\lambda, \log m, s)$.

Remark A.6 (Special Cases of BARGs with Fixed-Wire Constraints)**.** We describe two important special cases of BARGs with fixed-wire constraints:

- **BARG for** NP: When there are no fixed-wire assignments $A = \varnothing$ or constraints $\Phi = \varnothing$, Definition A.2 is equivalent to a standard BARG for NP (Definition 2.2).
- **BARG for index languages**: The special case of an index BARG (Remark 2.10) on m instances corresponds to a BARG with fully fixed wires where the set of fixed-wire assignments A input to Setup has size $|A| = n = O(\log m)$. The verification key $\mathsf{vk}_{\mathsf{indexBARG}}$ for the index BARG would be the verification key for the BARG with fixed wires together with the encodings of the assignments in A. In this case, succinctness (Definition A.5) requires that $|\mathsf{vk}_{\mathsf{indexBARG}}| = \mathsf{poly}(\lambda, \log m, s)$ and similarly, that the verification time is $\mathsf{poly}(\lambda, \log m, s)$. This matches the succinctness requirement for index BARGs.

Construction A.7 (BARG for NP with Fixed Wires from k-Lin)**.** Let $k \in \mathbb{N}$ be an integer. We show how to adapt Construction 4.5 to construct a BARG for the language of circuit satisfiability that supports fixed wires as follows. For ease of exposition, we do not describe Verify with split verification (Definition 2.9), but it is straightforward to modify the scheme to support it.

- $\mathsf{Setup}(1^\lambda, 1^m, 1^s, A)$: On input the security parameter λ, the number of instances m, the bound on the circuit size s, and the set of fixed-wire assignments $A \subseteq \{0,1\}^m$, the setup algorithm constructs the verification key $\mathsf{vk} = (\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i \in [m]})$ and the common reference string $\mathsf{crs} = (\mathsf{vk}, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i \ne j})$ exactly as in Construction 4.5. Then, for each $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m) \in A$, compute encodings

$$[\mathbf{u}_{\boldsymbol{\sigma}}]_1 \leftarrow \sum_{i \in [m]} \sigma_i [\mathbf{a}_i]_1 \quad \text{and} \quad [\hat{\mathbf{u}}_{\boldsymbol{\sigma}}]_2 \leftarrow \sum_{i \in [m]} \sigma_i [\hat{\mathbf{a}}_i]_2.$$

  The setup algorithm outputs crs, vk, and the dictionary D where $\mathsf{D}[\boldsymbol{\sigma}] \mapsto ([\mathbf{u}_{\boldsymbol{\sigma}}]_1, [\hat{\mathbf{u}}_{\boldsymbol{\sigma}}]_2)$

  To obtain a BARG with fully fixed wires, Setup removes the encodings of $[\mathbf{a}_i]_1$ and $[\hat{\mathbf{a}}_i]_2$ from the verification key. Namely, it sets

$$\mathsf{vk}' = (\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2).$$

  It outputs crs, vk′, and D. Note that Setup outputs the same crs as Construction 4.5.

- $\mathsf{Prove}(\mathsf{crs}, \mathsf{D}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$: On input the common reference string crs, a dictionary D of encodings, the circuit $C\colon \{0,1\}^n \to \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$, the prover proceeds constructs π using the same procedure as in Construction 4.5.

- $\mathsf{Verify}(\mathsf{vk}, C, (\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S), \{(i, \mathsf{enc}_i)\}_{i\in[n]\setminus S}, \pi) \to b$: On input the verification key

$$\mathsf{vk} = \big(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}\big),$$

  the circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a set of instances $(\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S) \in \{0,1\}^{|S|}$ restricted to the set S, a collection of encodings $\{(i, \mathsf{enc}_i)\}_{i\in[n]\setminus S}$, and the proof

$$\pi = \big(\{[\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2\}_{d\in[t]}, \{[\mathbf{V}_{n+d,i}]_1, [\hat{\mathbf{V}}_{n+d,i}]_2\}_{d\in[h],i\in\{1,2\}}, \{[\mathbf{W}_{\ell,i}]_1, [\hat{\mathbf{W}}_{\ell,i}]_2\}_{\ell\in[s],i\in\{1,2\}}\big).$$

  the verification algorithm starts by checking the following:

  - **Validity of statement:** For each statement wire $d \in [n]$, if $d \in S$, then the verifier checks that

$$[\mathbf{u}_d]_1 = \sum_{i\in[m]} x_{i,d}[\mathbf{a}_i]_1 \quad \text{and} \quad [\hat{\mathbf{u}}_d]_2 = \sum_{i\in[m]} x_{i,d}[\hat{\mathbf{a}}_i]_2,$$

    exactly as in Construction 4.5. For statement wires $d \in [n] \setminus S$, the verifier looks up the encoding (i, enc_i) and checks that $([\mathbf{u}_d]_1, [\hat{\mathbf{u}}_d]_2) = \mathsf{enc}_d$.

  The verifier performs the remaining checks exactly as described in the OnlineVerify algorithm of Construction 4.5.

Theorem A.8 (Completeness)**.** *Construction A.7 is complete.*

Proof (Sketch). The difference between Construction 4.5 and Construction A.7 is that instead of having the prover and verifier compute encodings of the fixed wires, those encodings are precomputed and provided as input to Prove and Verify. Completeness follows by an analogous argument as in the proof of Theorem 4.6. □

Theorem A.9 (Somewhere Argument of Knowledge)**.** *Take any positive integer $k \in \mathbb{N}$. If the k-*Lin *assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ with respect to* GroupGen*, then Construction A.7 is a somewhere argument of knowledge*

Proof (Sketch). The argument follows by a similar argument as in the proof of Theorem 4.7. For completeness, we describe the TrapSetup and Extract algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^m, 1^s, i^*)$: The TrapSetup algorithm samples $\mathsf{vk}^* = \big(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2, \{[\mathbf{a}_i]_1, [\hat{\mathbf{a}}_i]_2\}_{i\in[m]}\big)$, $\mathsf{crs}^* = \big(\mathsf{vk}^*, \{[\mathbf{B}_{i,j}]_1, [\hat{\mathbf{B}}_{i,j}]_2\}_{i\neq j}\big)$, and $\mathsf{td} = \boldsymbol{\tau} \in \mathbb{Z}_p^{k+1}$ using exactly the same procedure as TrapSetup in the proof of Theorem 4.7. Then, for each $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m) \in A$, it computes encodings

$$[\mathbf{u}_{\boldsymbol{\sigma}}]_1 \leftarrow \sum_{i\in[m]} \sigma_i[\mathbf{a}_i]_1 \quad \text{and} \quad [\hat{\mathbf{u}}_{\boldsymbol{\sigma}}]_2 \leftarrow \sum_{i\in[m]} \sigma_i[\hat{\mathbf{a}}_i]_2.$$

  Let D^* be the dictionary that maps $\mathsf{D}^*[\boldsymbol{\sigma}] \mapsto ([\mathbf{u}_{\boldsymbol{\sigma}}]_1, [\hat{\mathbf{u}}_{\boldsymbol{\sigma}}]_2)$ for all $\boldsymbol{\sigma} \in A$. The trapdoor setup algorithm outputs crs^*, vk^*, D^*, and td.

  In the case of a BARG with fully fixed wires, TrapSetup removes the encodings of $[\mathbf{a}_i]_1$ and $[\hat{\mathbf{a}}_i]_2$ from vk^*. Namely, it now sets $\mathsf{vk}^* = \big(\mathcal{G}, [\mathbf{M}]_1, [\hat{\mathbf{M}}]_2, [\mathbf{a}]_1, [\hat{\mathbf{a}}]_2\big)$. The other components crs^*, D^*, td are unchanged.

- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1|_S, \ldots, \mathbf{x}_m|_S), \{(i, \mathsf{enc}_i^*)\}_{i\in[n]\setminus S}, \pi)$: The extraction algorithm is the same as Extract in the proof of Theorem 4.7 (which only depends on td and π).

We now sketch the arguments for the CRS indistinguishability and somewhere extractability in trapdoor mode properties. Both follow by the corresponding argument from the proof of Theorem 4.7.

- **CRS indistinguishability:** This follows by the same argument as in the proof of Lemma 4.8. Namely, Lemma 4.8 shows that crs^* output by TrapSetup is computationally indistinguishable from crs output by Setup in Construction 4.5. These are the *exact* same components in the common reference string and verification key in Construction A.7. Next, the encodings in the dictionary D^* and D are public (and efficiently-computable) functions of the elements in crs^* and crs, respectively. Thus, the tuple $(\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*)$ output by TrapSetup (for any index $i^* \in [m]$) and $(\mathsf{crs}, \mathsf{vk}, \mathsf{D})$ output by Setup in Construction A.7 are computationally indistinguishable.
- **Somewhere extractable in trapdoor mode:** Since the structure of π in Construction A.7 and Construction 4.5 is *identical*, this property follows by the same argument as in the proof of Lemma 4.12. More precisely, we can show that an adversary that breaks the somewhere extractability property for Construction A.7 implies a corresponding adversary that breaks the same property for Construction 4.5. We provide a brief sketch here:

  Suppose there exists an adversary $\mathcal{A}$ that wins the somewhere extractable game with non-negligible probability ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that wins the somewhere extractable game for Construction 4.5 with the same probability:

  - Algorithm $\mathcal{B}$ runs $\mathcal{A}$ to obtain an index $i^* \in [m]$ and a set of fixed-wire assignments $A \subseteq \{0,1\}^m$.
  - Algorithm $\mathcal{B}$ submits i^* to its challenger and receives crs^* from the challenger. It forms vk^* from crs^* (which consists of a subset of the components of crs^*). Algorithm $\mathcal{B}$ computes D^* as described in TrapSetup (which only depends on components in crs^*). Algorithm $\mathcal{B}$ gives $\mathsf{crs}^*, \mathsf{vk}^*, \mathsf{D}^*$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a set of fixed-wire constraints Φ, a set of statements $\hat{\mathbf{x}}_1|_{\bar{S}_\Phi}, \ldots, \hat{\mathbf{x}}_m|_{\bar{S}_\Phi} \in \{0,1\}^{\bar{S}_\Phi}$ restricted to the set $\bar{S}_\Phi = [n] \setminus S_\Phi$, and a proof π.
  - For $i \in [m]$, define $\mathbf{x}_i|_{\bar{S}_\Phi} = \hat{\mathbf{x}}_i|_{\bar{S}_\Phi}$. For indices $j \in S_\Phi$ fixed by Φ, let $x_{i,j} = a_{j,i}$ where $(j, (a_{j,1}, \ldots, a_{j,m})) \in \Phi$.
  - Algorithm $\mathcal{B}$ outputs the circuit C, statements $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ and the proof π.

  By construction, if $\mathsf{Verify}(\mathsf{vk}^*, C, (\hat{\mathbf{x}}_1|_{\bar{S}_\Phi}, \ldots, \hat{\mathbf{x}}_m|_{\bar{S}_\Phi}), \{(j, \mathsf{D}^*[\boldsymbol{\sigma}_j])\}_{(j,\boldsymbol{\sigma}_j)\in\Phi}, \pi) = 1$, then $\big(C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), \pi\big)$ verifies under the same procedure in Construction 4.5. Moreover, the extraction algorithm Extract is identical to the corresponding algorithm in the proof of Lemma 4.12. Thus, algorithm $\mathcal{B}$ succeeds with the same advantage as $\mathcal{A}$ and the claim holds. □

Theorem A.10 (Succinctness)**.** *For all constants $k \in \mathbb{N}$, Construction A.7 is succinct.*

Proof. Take any $\lambda, m, s \in \mathbb{N}$ and any set of fixed-wire assignments $A \subseteq \{0,1\}^m$. Take any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s and any set of fixed-wire constraints. We check each property individually:

- **Proof size:** By construction, the size of the proof is the same as in Construction 4.5. By Theorem 4.15, $|\pi| = \mathsf{poly}(\lambda, s)$.
- **Verification key size:** We can appeal to the analysis in Theorem 3.10 to show that $|\mathsf{crs}|, |\mathsf{vk}| = \mathsf{poly}(\lambda, m)$. Next, for each $\boldsymbol{\sigma} \in A$, the encoding $\mathsf{D}[\boldsymbol{\sigma}]$ consists of $k+1$ elements in each of $\mathbb{G}_1$ and $\mathbb{G}_2$. Thus, $|\mathsf{D}[\boldsymbol{\sigma}]| = \mathsf{poly}(\lambda)$. Thus,
$$|\mathsf{vk}| + |\{\mathsf{D}[\boldsymbol{\sigma}]\}_{\boldsymbol{\sigma}\in A}| = \mathsf{poly}(\lambda, m) + \mathsf{poly}(\lambda, |A|).$$
  In the fully fixed wire setting, the verification key consists of the group description $\mathcal{G}$ along with $(k+1)^2$ elements in each of $\mathbb{G}_1$ and $\mathbb{G}_2$. In this case, $|\mathsf{vk}| = \mathsf{poly}(\lambda)$, and so
$$|\mathsf{vk}| + |\{\mathsf{D}[\boldsymbol{\sigma}]\}_{\boldsymbol{\sigma}\in A}| = \mathsf{poly}(\lambda, |A|).$$
- **Verification time:** The verification procedure in Construction A.7 is a slimmed-down version of the procedure from Construction 4.5 where some of the statement validity checks are replaced with direct equality checks against the provided encodings. The claim follows by a similar analysis. In particular, in the fully-fixed setting, the verifier does not need to perform *any* statement-validity check, which yields an overall verification time of $\mathsf{poly}(\lambda, s)$ in this setting. □