Batch Arguments for NP and More from Standard Bilinear Group Assumptions

Brent Waters[1,2] and David J. Wu[1]

[1] University of Texas at Austin, Austin, TX, USA
[2] NTT Research, Sunnyvale, CA, USA

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.[3] 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 $k \geq 1$)[4], 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 each gate in the circuit, the prover provides a short proof that

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

[4]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]

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 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 the full version of this paper [WW22]. 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 the full version of this paper [WW22] 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 $i \in [m]$).[5] 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.[6] 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

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

[6]Our techniques extend naturally to support binary-valued gates that can compute *arbitrary* quadratic functions of their inputs; see the full version of this paper [WW22].

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.

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 $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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 $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$. 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 $\mathsf{poly}(\lambda)$, 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

$$\mathsf{crs} = \Big(\{A_i := g_p^{\alpha_i}\}_{i \in [m]} \ , \ A := \prod\nolimits_{i \in [m]} A_i = g_p^{\sum_{i \in [m]} \alpha_i} \ , \ \{B_{i,j} := g_p^{\alpha_i \alpha_j}\}_{i \neq j} \Big) . \tag{1.1}$$

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].[7] 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).[8] 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

[7] 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.

[8] 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.

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 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 $(\xi_1, \dots, \xi_t) \in \{0,1\}^t$ constitutes a valid assignment to the wires of $C((\xi_1, \dots, \xi_n), \mathbf{w}^*)$ where $\mathbf{w}^* = (\xi_{n+1}, \dots, \xi_{n+h})$. Again considering the verification relations in the order-q subgroup, the input validity checks ensure that $\mathbf{x}_{i^*} = (\xi_1, \dots, \xi_n)$ 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 $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. 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 the full version of this paper [WW22] 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.[9] 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. 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. We refer to the full version of this paper [WW22] 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 the full version of this paper [WW22].

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 the full version of this paper [WW22], 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,

[9] 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.

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].

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 $\mathsf{DDH} + \mathsf{QR}$ 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 $\mathsf{IP} = \mathsf{PSPACE}$ 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 $x \xleftarrow{\text{R}} S$ 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 the full version of this paper [WW22], 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* $\lambda \in \mathbb{N}$,

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

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* $\mathsf{TrapSetup}$ *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 $\mathsf{TrapSetup}$ *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 $\mathbf{x}_1, \dots, \mathbf{x}_m \in \{0,1\}^n$, and a proof π. The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \dots, \mathbf{x}_m), \pi) = 1$ and $(C, \mathbf{x}_{i^*}) \notin \mathcal{L}_{\mathsf{CSAT}}$. Otherwise, the output is $b = 0$.*

Then, $\mathit{\Pi}_{\mathsf{BARG}}$ 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 $\mathit{\Pi}_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ 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.*

– $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \dots, \mathbf{x}_m), \pi) \to \mathbf{w}^*$ *On input the trapdoor td, statements $\mathbf{x}_1, \dots, \mathbf{x}_m$, 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 $\mathbf{x}_1, \dots, \mathbf{x}_m \in \{0,1\}^n$, and a proof π. Let $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \dots, \mathbf{w}_m), \pi)$.*
  - *The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \dots, \mathbf{x}_m), \pi) = 1$ and $C(\mathbf{x}_{i^*}, \mathbf{w}^*) \neq 1$. Otherwise, the output is $b = 0$.*

  Then $\mathit{\Pi}_{\mathsf{BARG}}$ 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 $\mathit{\Pi}_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ 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 the full version of this paper [WW22] (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,[10] 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* $\mathsf{CompGroupGen}$ *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* $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *is a non-degenerate bilinear map (called the "pairing"). We require that the group operation in* $\mathbb{G}$ *and* $\mathbb{G}_T$ *as well as the pairing operation to be efficiently computable.*

Definition 3.2 (Subgroup Decision [BGN05]). *The subgroup decision assumption holds with respect to a composite-order bilinear group generator* $\mathsf{CompGroupGen}$ *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* $r \xleftarrow{\text{R}} \mathbb{Z}_N$*.*

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 $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_p, e)$.

[10] 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).

  - For each $i \in [m]$, sample $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. 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 $\mathsf{crs} = \big(\mathcal{G}, A, \{A_i\}_{i \in [m]}, \{B_{i,j}\}_{i \neq j}\big)$.

– $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_m), (\mathbf{w}_1, \ldots, \mathbf{w}_m))$: On input the common reference string $\mathsf{crs} = (\mathcal{G}, A, \{A_i\}_{i \in [m]}, \{B_{i,j}\}_{i \neq j})$, the 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 $\mathbf{w}_1, \ldots, \mathbf{w}_m \in \{0,1\}^h$, 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 $W_\ell = \prod_{i \neq j} B_{i,j}^{1 - w_{i,k_1} w_{j,k_2} - w_{j,k_3}}$

  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 $e(A, A)$:

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

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 $\alpha_i\xleftarrow{\text{R}}\mathbb{Z}_N$. 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 $\mathsf{crs}^*=\big(\mathcal{G},A,\{A_i\}_{i\in[m]},\{B_{i,j}\}_{i\neq j}\big)$ 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 $\pi=\big(\{U_k,V_k\}_{k\in[t]},\{W_\ell\}_{\ell\in[s]}\big)$, 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 $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$. 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 $\mathsf{crs} = \big(\mathcal{G}, A, \{A_i\}_{i \in [m]}, \{B_{i,j}\}_{i \neq j}\big)$.

  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 $\mathsf{TrapSetup}$:
  - For each $i \in [m]$, sample $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$. 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 $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$. 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 $\mathsf{GroupGen}$. Then, for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that distinguishes Hyb_1 and Hyb_2 with non-negligible advantage 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 $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_N$ 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 $\mathsf{crs} = \big(\mathcal{G}, A, \{A_i\}_{i \in [m]}, \{B_{i,j}\}_{i \neq j}\big)$ 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 ε.

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 $\pi = \big(\{U_k, V_k\}_{k \in [t]}, \{W_\ell\}_{\ell \in [s]}\big)$ 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 $\alpha_{i^*} \xleftarrow{\text{R}} \mathbb{Z}_N$. 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 $\mathsf{OnlineVerify}$ 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 $T \leq m$ 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

Due to space limitations, we defer our BARG construction from asymmetric prime-order pairing groups (where the k-Lin assumption holds) to the full version of this paper [WW22]. In this setting, the pairing $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently-computable bilinear map from the base groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to the target group $\mathbb{G}_T$. The construction relies on a similar underlying principle as the construction from symmetric composite-order groups (Construction 3.3). Here, we summarize the key differences and refer readers to the full version for the complete description and analysis:

– **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 the full version of this paper [WW22]).
– **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 the full version of this paper [WW22], this enables a similar strategy to prove soundness.

5 Extensions and Applications

Bootstrapping to reduce CRS size. As mentioned inSection 1.2.2, we can leverage a similar type of bootstrapping from the work of Kalai et al. [KPY19] to reduce the size of the CRS in our BARG constructions to grow with m^ε for any $\varepsilon > 0$ and where m is the number of instances. We refer to Section 1.2.2 for the overview of this approach and to the full version of this paper [WW22] for the full details.

Application to delegation. In the full version of this paper [WW22], we show how to use our BARG for NP to obtain a delegation scheme for RAM programs (equivalently, a SNARG for P). Our construction follows the approach from Choudhuri et al. [CJJ21b] of combining a BARG for "index languages" (or more generally, any BARG with the split verification property (Definition 2.9)) with a somewhere extractable commitment scheme. The BARGs we construct in this work

both satisfy the required split verification property. In the full version of this paper [WW22], we also show how to use our techniques in conjunction with somewhere statistically binding hash functions [HW15] to obtain a somewhere extractable commitment scheme. This suffices to obtain a RAM delegation scheme from the SXDH assumption in asymmetric pairing groups.

Application to aggregate signatures. In the full version of this paper [WW22], we describe a simple approach of constructing aggregate signatures that supports bounded aggregation from a BARG for NP. Together with our BARG for NP (from either subgroup decision or k-Lin), we obtain an aggregate signature scheme from the same assumption.

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.

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.

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.

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.

WW22. Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. *IACR Cryptol. ePrint Arch.*, 2022.