

SNARGs for \mathcal{P} from LWE

Arka Rai Choudhuri

Abhishek Jain

Zhengzhong Jin

Johns Hopkins University

Abstract

We provide the first construction of a succinct non-interactive argument (SNARG) for *all* polynomial time deterministic computations based on standard assumptions. For T steps of computation, the size of the proof and the common random string (CRS) as well as the verification time are poly-logarithmic in T . The security of our scheme relies on the hardness of the Learning with Errors (LWE) problem against polynomial-time adversaries. Previously, SNARGs based on standard assumptions could support bounded-depth computations and required sub-exponential hardness assumptions [Jawale-Kalai-Khurana-Zhang, STOC'21].

Along the way, we also provide the first construction of non-interactive batch arguments for NP based solely on the LWE assumption. The size of the proof and CRS for a batch of k statements grows only with the size of a *single* witness.

Contents

1	Introduction	3
1.1	Our Results	3
1.2	Related Work	4
2	Technical Overview	5
2.1	Background	5
2.2	Delegating Polynomial-Time Computations	8
2.3	Non-interactive Batch Arguments for NP	12
3	Preliminaries	17
3.1	Notations	17
3.2	Low-degree Extensions	18
3.3	Learning with Error	18
3.4	Correlation Intractable Hash	18
3.5	Somewhere Extractable Commitment	19
3.6	No-Signaling Somewhere Extractable Commitments	21
4	Non-interactive Batch Arguments	21
4.1	Definition	22
4.2	PCPs with Split Verification	23
4.3	BARGs for Index Languages	27
4.4	BARGs for NP	32
5	RAM Delegation	34
5.1	Turing Machine Delegation	34
5.2	RAM Delegation	34
5.3	Hash Tree	36
5.4	Protocol	37
5.4.1	Efficiency	38
5.4.2	Security Proof	40
6	More Efficient Batch Arguments for NP	46
A	Proof of Theorem 5	52

1 Introduction

Consider the following scenario: a client wishes to evaluate a program P (say, represented as a Turing machine) on an input x but does not have the necessary computational resources. Instead, it delegates the computation to an untrusted server who provides the output $P(x)$ together with a proof Π . The key requirement is that the proof Π should be much faster to verify than the time it takes to compute $P(x)$.

The focus of this work is on constructing such proof systems in the non-interactive setting, where they are referred to as *succinct non-interactive arguments* (SNARGs).¹ The de facto model for such proof systems allows for an initial setup that samples a (reusable) common reference string (CRS) and distributes it to the parties. Furthermore, the soundness guarantee is computational, i.e., it only holds against computationally-bounded provers [BCC88]. The key benefit of such proof systems is that they can be used as short certificates for the correctness of long computations that can be verified by anyone. Applications of SNARGs abound in the literature and include popular real-world systems such as blockchains [BCG⁺14].

In this work, we focus on the task of constructing SNARGs based on *standard assumptions*. Despite extensive research over the recent years (see Section 1.2 for a summary), the following basic question has remained open:

Do there exist SNARGs for all polynomial-time (deterministic) computations based on standard assumptions?

A recent beautiful work of Jawale, Kalai, Khurana and Zhang [JKKZ21] makes progress on this front. They construct SNARGs for *bounded-depth* deterministic computations based on the sub-exponential hardness of the Learning with Errors (LWE) assumption. The goal of our work is to support arbitrary-depth polynomial-time computations, while relying only on standard polynomial-time assumptions.

1.1 Our Results

We construct SNARGs for all polynomial-time deterministic computations based on the hardness of LWE against polynomial time adversaries. Our construction is in the common *random* string model and achieves adaptive soundness.

Theorem 1 (Informal). *Assuming the hardness of LWE, for every polynomial $T = T(\lambda)$, there exists a publicly-verifiable non-interactive delegation scheme with adaptive soundness for any time T Turing machine. The verifier running time, size of the CRS and proof are all $\text{poly}(\log T, \lambda)$ while the prover running time is $\text{poly}(T, \lambda)$.*

Our result also extends, with the same parameters, to delegation of RAM computation.

Non-interactive Batch Arguments. Towards obtaining our main result, we study the related problem of *non-interactive batch arguments* (BARGs) for NP. Informally speaking, such an argument system allows an efficient prover to compute a non-interactive and publicly verifiable “batch proof” of k NP instances, with size smaller than the combined witness length. If any of the k instances is false, then no polynomial-time cheating prover must be able to produce an accepting proof. BARGs allow for delegating non-deterministic computation for a specific sub-class of NP, namely, conjunction of NP statements.

Very recently, Choudhuri, Jain and Jin [CJJ21] provided the first construction of BARGs for NP based on standard assumptions. In their scheme, the size of the proof for proving k statements for the circuit satisfiability problem defined by a circuit C is $\tilde{O}((|C| + \sqrt{k|C|}) \cdot \lambda)$. The security of their scheme is based on the hardness of Quadratic Residuosity (QR) and either LWE or sub-exponential Decisional Diffie-Hellman (DDH).

¹In the literature, the term SNARG is typically associated with NP computations. In this work, similar to [JKKZ21], we focus on deterministic computations.

We improve upon their work along several dimensions that are important towards obtaining our main result in Theorem 1:

- We consider (and achieve) a notion of *semi-adaptive somewhere soundness* which is stronger than the non-adaptive soundness notion considered in [CJJ21]. We discuss this further in Section 2.²
- We reduce the dependence on the number of statements in the proof and CRS size to only *poly-logarithmic*. As a contribution of independent interest (but not crucial towards obtaining Theorem 1), we also remove the dependence on the circuit size.
- We simplify the hardness assumptions and base the security of our scheme *solely* on LWE .³

Theorem 2 (Informal). *Assuming the hardness of LWE , there exists a BARG for NP with the following parameters: in order to prove k instances of a language \mathcal{L} whose NP-relation can be decided by a Turing machine in time T , the size of the CRS and proof are $\text{poly}(\log k, \log T, m, \lambda)$, the prover running time is $\text{poly}(k, T, m, \lambda)$ and the verifier running time is $\text{poly}(\log k, \log T, m, \lambda) + \text{poly}(k, n, \lambda)$, where n is the length of a single instance and m is the length of a single witness.*

Finally, we note that our BARG scheme is in the common random string model. In contrast, [CJJ21] requires a common *reference* string.

1.2 Related Work

We now provide a brief overview of the related work on delegating computation. We borrow from the excellent summaries in [JKKZ21, KPY19] and some of the text below is taken verbatim from these works.

Prior work on delegating computation can be roughly divided in three categories, described below.

- **SNARGs.** There is a large body of work, starting from [Mic94], that constructs SNARGs for non-deterministic computation (see, e.g., [Mic94, Gro10, Lip12, DFH12, GGPR13, BCI⁺13, BCCT13, BCC⁺17]). These schemes are either in the Random Oracle model or require non-falsifiable assumptions [Nao03]. However, some of these schemes form the basis of efficient implementations used in practice. Other constructions of SNARGs for deterministic computations are known based on assumptions related to obfuscation or multilinear maps [CHJV15, KLW15, BGL⁺15, CH16, ACC⁺16, CCC⁺16, PR17].

Recently, [KP19] constructed SNARGs for deterministic computations (as well as batch arguments for NP) from a new falsifiable but non-standard assumption on groups with bilinear maps (which is known to be broken with quantum attacks). Independently, [CCH⁺19] constructed SNARGs for bounded-depth computations based on a very strong assumption, namely, the existence of fully homomorphic encryption with optimal circular security.

Even more recently, [JKKZ21] constructed SNARGs for bounded-depth computations from sub-exponential hardness of LWE . Our work overcomes both limitations of their work, i.e. support for bounded-depth computations and use of sub-exponential hardness assumptions.

- **Designated Verifier Proofs.** An influential line of work starting from [KRR13, KRR14] and continuing with [KP16, BHK17, BKK⁺18, HR18, BK20] construct delegation schemes for deterministic computations and various sub-classes of non-deterministic computations (such as batch arguments) based on standard assumptions. The main drawback of these schemes is that they can only be verified by a designated verifier, who knows the “secret key” corresponding to the CRS. The main benefit of our work compared to this line of work is that we achieve public verifiability.

²As observed in [BHK17], there are significant barriers to constructing BARGs with full adaptive soundness due to implications to adaptively sound SNARGs for NP [GW11].

³An alternative direction that we do not pursue in this work is to base security solely on sub-exponential DDH (as in [JJ21]).

- **Interactive Proofs.** In the setting of interactive protocols, publicly verifiable delegation schemes for NP are known from standard assumptions [Kil92]. The work of [BKP18] constructs three message protocols based on multi-collision resistant hash functions, and [PRV12] constructs two-message schemes (in addition to CRS) for low-depth circuits from attribute-based encryption.

Finally, we note that in the interactive setting, delegation schemes with even unconditional soundness are known. The works of [GKR08, RRR16] construct such schemes for bounded depth and bounded space computations, respectively. Furthermore, [RRR16, RRR18, RR20] construct batch proofs (with unconditional soundness) for UP, a subclass of NP where each statement in the language has a unique witness of membership.

More Related Work. We also mention recent works of [GZ21, GR19] that study publicly verifiable non-interactive argument systems for deterministic computations. In particular, assuming standard assumptions on groups with bilinear maps, [GZ21] constructs a publicly verifiable non-interactive argument system for polynomial-time computations where the proof size does not grow with the size of the computation. Their scheme (as well as [GR19]), however, requires a long CRS (and hence, long verification time) proportional to the circuit size.

The work of [GZ21] introduce a new notion of *no-signaling* commitment schemes. We use this primitive in our construction of SNARGs for polynomial-time computations.

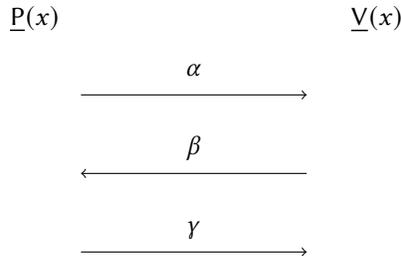
2 Technical Overview

Towards our goal of achieving publicly verifiable delegation schemes for all polynomial time computations, we depart significantly from prior approaches in the designated-verifier setting. We leverage advances in the instantiation of the Fiat-Shamir transformation, recently applied in the context of publicly verifiable delegation schemes for bounded-depth computation [CCH⁺19, JKKZ21]. We start with an overview of the necessary background before describing the main ideas underlying our work.

2.1 Background

Fiat-Shamir (FS) Paradigm. At a very high level, the Fiat-Shamir transform [FS87] is a round collapsing transformation that allows one to start with an *interactive proof* of a specified structure, and transform it into a *non-interactive argument* in the CRS model. Specifically, the starting interactive proof system (P, V) must be public coin i.e. a protocol where the verifier only sends random coins as its messages.

The transformation is defined with respect to some hash function family \mathcal{H} , where the sampled hash function, $h \leftarrow \mathcal{H}$, is set to be the CRS. The prover can then derive the verifier’s messages non-interactively by applying h on the protocol transcript. Consider the following interactive protocol between the prover P and verifier V establishing that $x \in \mathcal{L}$, where the V ’s message β is a uniformly random string.



To generate a non-interactive proof, P computes $\beta = h(x, \alpha)$ with the resultant proof being (α, γ) . V can recompute β (from x and α) and check if the transcript $(x, \alpha, \beta, \gamma)$ is accepting. Note that the total

communication from the prover to the verifier remains unchanged by the transformation. Therefore, when proof size is the main concern in the non-interactive setting, it is important to start with an interactive protocol that already satisfies the communication requirements.

Soundness of Fiat-Shamir transform and CI. Initially, the soundness (i.e. inability of a cheating prover to generate an accepting proof when $x \notin \mathcal{L}$) of the Fiat-Shamir transform was proven, modeling the hash family as a random oracle. But an exciting line of recent results have shown that for several applications [KRR17, CCRR18, HL18, CCH⁺19, PS19, BKM20, CPV20, CKU20, JJ21], the Fiat-Shamir transformation is sound when the hash family is *correlation intractable*.

First, let us see why there is a need to “re-prove” the soundness in the transformed non-interactive protocol. Unlike in the interactive setting, where the prover has no control over the verifier message β , in the transformed protocol, a cheating prover could try various values of α as inputs to h until it arrives on a β it finds favorable. Specifically, for every $x \notin \mathcal{L}$, and every α , we define the set of “bad” β s,

$$\mathcal{B}_{x,\alpha} := \{\beta \mid \exists \gamma \text{ s.t. } V(x, \alpha, \beta, \gamma) = 1\}.$$

Intuitively, these are the set of verifier challenges that could lead the verifier to accept, even if the statement is not in the language. We want it to be computationally intractable to find an α such that $h(x, \alpha) \in \mathcal{B}_{x,\alpha}$, i.e. hard to find α that would result in a *bad* verifier challenge. This is exactly what *correlation intractability* of a hash family captures. Specifically, we say that \mathcal{H} is correlation intractable for a function f if the following holds for all probabilistic polynomial time adversary (PPT) \mathcal{A} ,

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = f(x) \mid \mathcal{A}(h) = x] \leq \text{negl}(\lambda).$$

Assume for the moment that there exists at most one *bad* verifier challenge for every pair (x, α) . Then one can define a function $f(\cdot) := \text{BAD}(\cdot)$ that on input (x, α) outputs the unique $\beta \in \mathcal{B}_{x,\alpha}$ (if it exists). If \mathcal{H} is a CIH for f , then any cheating prover producing an accepting transcript (α, γ) for $x \notin \mathcal{L}$ must break the correlation intractability of \mathcal{H} , since by definition $h(x, \alpha) \in \mathcal{B}_{x,\alpha}$. For any set $\mathcal{B}_{x,\alpha}$ where $|\mathcal{B}_{x,\alpha}|$ is *polynomially bounded*, one can set $f_i(\cdot) := \text{BAD}(\cdot, i)$ to output the i -th element of $\mathcal{B}_{x,\alpha}$. By a simple application of union bound, one can observe that it remains computationally intractable for an adversary to find an α such that $h(x, \alpha)$ is the output of any f_i , and thereby remains intractable to output *any* element in $\mathcal{B}_{x,\alpha}$.

Although our discussion thus far has been for a single verifier message, the above ideas can be extended to multi-round protocols that additionally satisfy certain properties such as *round-by-round soundness* [CCH⁺19].

Note that the BAD functions that can be supported using the above methodology are constrained by the functions for which we can construct CIH. The known CIH from standard assumptions are: bounded-depth polynomial size circuits from LWE [CCH⁺19, PS19], approximable relations for constant-degree polynomials from trapdoor hash functions [BKM20, DGI⁺19], and TC^0 circuits from sub-exponential DDH [JJ21].

Fiat-Shamir Instantiation for Product Relations. While we described above an extension to *any* polynomially bounded $|\mathcal{B}_{x,\alpha}|$, this approach no longer works when $\mathcal{B}_{x,\alpha}$ is super-polynomial. Looking ahead, the set of bad challenges we consider in our work will *not* be of a polynomially bounded size, and therefore the ideas discussed above do not suffice. Instead, we will borrow upon the recent exciting work of [HLR21]. Their work consider sets $\mathcal{B}_{x,\alpha}$ that for every x and α , can be represented as a Cartesian product of t sets,

$$\mathcal{B}_{x,\alpha} = \mathcal{B}_{x,\alpha}^{(1)} \times \cdots \times \mathcal{B}_{x,\alpha}^{(t)}$$

where each $\mathcal{B}_{x,\alpha}^{(i)}$ can be efficiently verified, i.e. there is a circuit C that on input (x, α) , β_i and i outputs 1 if and only $\beta_i \in \mathcal{B}_{x,\alpha}^{(i)}$. For such sets, [HLR21] show that one can construct CI hash families assuming only the hardness of LWE even if $|\mathcal{B}_{x,\alpha}|$ is not polynomially bounded.

Main Barriers. Since the Fiat-Shamir transformation preserves prover communication, it is imperative that our interactive protocol already has low communication. A natural candidate for such an interactive protocol is the interactive delegation protocol for NP by Kilian [Kil92]. The protocol is public coin and the total communication is smaller than the size of the witness. One would then hope that if we are able to instantiate the Fiat-Shamir transform applied to Kilian’s protocol, this would give us, in the CRS model, a non-interactive delegation scheme for polynomial-time computations (and in fact, much more). While the approach appears promising at first, a recent work of [BBH⁺19] established non-trivial barriers to instantiating the hash function in the Fiat-Shamir transformation of Kilian’s protocol.

There is in fact a broader point to consider: Kilian’s protocol is an *argument*, i.e. its soundness holds only against computationally bounded cheating provers. In general, successful applications of the Fiat-Shamir paradigm when used in conjunction with CIH, have been largely limited to starting with interactive *proofs*, i.e. protocols for which even a computationally unbounded adversary cannot convince a verifier of the validity of a false statement. In fact there are examples of certain interactive arguments that are not sound on the application of the Fiat-Shamir transformation (see e.g. [Bar01, GK03]).

For this reason, the state of the art non-interactive delegation schemes that follow this approach [CCH⁺19, JKKZ21] are limited to bounded-depth computations, as they rely upon known interactive delegation schemes with unconditional soundness – in particular, the scheme of [GKR08] for bounded-depth computations. Indeed, constructing publicly verifiable interactive delegation schemes with unconditional soundness for classes beyond those in the works of [GKR08, RRR16] is a major open question.

We also note that [GKR08], and therefore [CCH⁺19, JKKZ21] use the Sumcheck protocol [LFKN92, Sha92] as a crucial component in their scheme. In the context of FS transformation, the resulting “bad” challenge function seems to necessitate *sub-exponential* hardness assumptions (we refer the reader to [JKKZ21] for details).

Our Work. In light of the challenges described above, we take a different approach. We choose to view the problem of delegation of deterministic computations through the lens of *batch arguments for NP*. Here the prover is trying to convince the verifier of the veracity of k different statements for an NP language, with communication smaller than the combined length of the witnesses for all the statements. This is an independently interesting problem, and has seen recent progress in the non-interactive setting based on standard assumptions [CJJ21].

More specifically, we reduce the task of constructing delegation schemes for polynomial-time computations to the task of constructing non-interactive batch arguments (BARGs) for NP. We then use the Fiat-Shamir methodology to construct BARGs for NP. As is to be expected, the same challenges as discussed earlier in the context of using the Fiat-Shamir transformation apply to the problem of constructing BARGs as well. Indeed, presently interactive batch *proofs* are only known for UP (a subset of NP for which each statement has a unique witness) [RRR16, RRR18, RR20], and it is an open problem to construct batch proofs for NP. Nevertheless, as we will discuss later in Section 2.3, we will build upon the “*dual-mode methodology*” from the recent work of [CJJ21] to circumvent these challenges and construct BARGs (with necessary security and efficiency properties that we discuss below) based on LWE. For now, however, we simply assume that such BARGs exist and proceed to describe the main ideas underlying our construction of a delegation scheme for polynomial-time computations.

We remark that some works [BHK17, KPY19] have previously studied both of these problems – delegations schemes for deterministic computations and batch arguments for NP – and used common tools

and techniques to solve both the problems. We make this connection more explicit by reducing the problem of delegation of deterministic computations to batch arguments. A similar approach was taken in the work of [RRR16] who consider the problem of batch verifying *interactive proofs* in the setting of (unconditionally sound) interactive delegation for bounded space computation. At a very high level, in their work, the prover sends several intermediate steps of the computation and then batches proofs that these intermediate steps were computed correctly. We use a similar blueprint; however, our focus is on the non-interactive setting, and all polynomial-time computations. Furthermore, we require stronger efficiency – poly-logarithmic dependence on the number of computation steps, as opposed to sublinear in [RRR16].

2.2 Delegating Polynomial-Time Computations

We start our discussion with the problem of delegating the computation of a Turing machine. Here, for a Turing machine \mathcal{M} and input x , the prover produces a proof Π to convince the verifier that \mathcal{M} accepts x within T steps, with the requirement that both the proof size $|\Pi|$ and the verifier’s running time are $\text{polylog}(T)$. As stated earlier, we want to cast the problem of delegation as a problem of BARGs for NP. Intuitively, a BARG for an NP language \mathcal{L} allows the prover to prove that k statements x_1, \dots, x_k all belong to \mathcal{L} such that communication cost is “small”. For the moment, we leave imprecise the exact communication requirements, but we shall specify it shortly based on our use case.

To cast the delegation problem as a BARG, we look at the intermediate states of the Turing machine computation. Let st_i be the encoding of the state of \mathcal{M} and its tapes after exactly i steps of the computation. We want to prove that for every $i \in [T - 1]$, $st_{i+1} = \text{Step}(st_i)$, where Step is the *deterministic* algorithm computing the state transition of a single step. The states st_i are thus a “witness” to the entire computation. On the surface, this already appears to be a batch problem of T instances, but an observant reader may notice that for each i , the witnesses for i and $i + 1$ “overlap”, specifically the overlapping state st_{i+1} . If this overlap of witnesses are not ensured, then a cheating prover could use witnesses (st_i, st_{i+1}) and (st'_{i+1}, st'_{i+2}) such that $st_{i+1} = \text{Step}(st_i) \wedge st'_{i+2} = \text{Step}(st'_{i+1})$ but $st_{i+1} \neq st'_{i+1}$. This is clearly undesirable since the overlap of witnesses is necessary to establish *continuity* in the computation - otherwise a cheating prover is proving T independent statements, unhelpful to establish correctness of computation. Unfortunately, the notion of batch arguments we have described does not enforce any constraints across statements.

To overcome this problem, we have the prover commit to *all* the internal states $st_1 || \dots || st_T$ - let this committed value be c . The prover now proves that for every $i \in [T]$, $(x, c, i) \in \mathcal{L}$, where \mathcal{L} is defined by the relation circuit C below.

C

Statement: x, c, i
Witness: $st_i, st_{i+1}, \text{open}_{st_i}, \text{open}_{st_{i+1}}$
Output: Output 1 if and only if the following verify

1. check if $\text{Com.Verify}(c, st_i, \text{open}_{st_i}) \stackrel{?}{=} 1$.
2. check if $\text{Com.Verify}(c, st_{i+1}, \text{open}_{st_{i+1}}) \stackrel{?}{=} 1$.
3. if $i = 1$, check if st_1 encodes input x .
4. if $i = T - 1$, check if st_T is the *accept* state.
5. Check if $\text{Step}(st_i) = st_{i+1}$.

Here, $\text{open}_{\text{st}_i}$ corresponds to a proof of opening that st_i was indeed the i -th vector that was committed to in c .

Let us consider some efficiency properties of the commitment scheme,

- *Size of the commitment*: For now we allow the total communication from the prover to the verifier be at most $\text{poly}(\log T, |\text{st}|)$, i.e. the communication grows with the size of the internal state. We shall shortly see how to go beyond the space constraints, but for this discussion we focus on the weaker goal of *bounded-space computation* since it already highlights the main challenges. Enforcing the same communication limit to the size of the commitment - we have that the commitment to a vector of size $T \cdot |\text{st}|$ is at most $\text{poly}(\log T, |\text{st}|)$, i.e. the commitment is succinct.
- *Size of the opening*: Looking ahead, we will require that the size of C , the relation circuit for \mathcal{L} , to also be at most $\text{poly}(\log T, |\text{st}|)$. This means that the commitment must have a *succinct local opening* opening $\text{open}_{\text{st}_i}$ to st_i , since the opening is a part of the witness (and thus contributes to $|C|$). Succinctness here means that the size of the opening is $\text{poly}(\log T, |\text{st}|)$.

Observe that the statements that the prover is batching are *identical* except for the index i . This motivates us to adopt the following useful abstraction we call *batch arguments for index languages*. Formally, an index language is defined as follows,

$$\mathcal{L}' = \{(C, i) \mid \exists w \text{ s.t. } C(i, w) = 1\}$$

where C represents a circuit, and i an index. In a BARG for an index language, the prover tries to convince the verifier that $(C, 1), \dots, (C, T) \in \mathcal{L}'$. For now we assume the existence of such BARGs, and we describe how we construct them in the next section. But before we construct the BARG, it is important to establish both the efficiency and security properties required for the purposes of constructing delegation scheme for polynomial-time computations.

Efficiency. As stated, we require the total communication to be $\text{poly}(\log T, |\text{st}|)$. So, as long as $|C| = \text{poly}(|\text{st}|)$, we allow the size of the batch arguments for T statements to be $\text{poly}(\log T, |C|)$. This already rules out using existing batch arguments based on standard assumptions such as [CJJ21], since they do not meet the efficiency requirements (the BARG proof size in [CJJ21] depends on \sqrt{T}). We will also require the verification time of the BARG to be $\text{poly}(\log T, |C|)$. Usually the verification time for BARG has an overhead to read *all* instances, but due to the above abstraction each instance can be “read” as required by simply specifying the circuit C and an index i .

Security. Given that the commitment is compressing (succinct), for every i there could always *exist* states st'_i and st'_{i+1} with corresponding local commitment openings to c such that $\text{Step}(\text{st}'_i) = \text{st}'_{i+1}$ even if it is computationally hard to find them. Thus for all i it may always be the case that $(C, i) \in \mathcal{L}'$, making soundness of the batch argument a vacuous notion. The fix is to use *somewhere statistical binding commitments* [HW15] such that for a commitment key generated on input i^* there is a *unique* (except with negligible probability) local opening to $\text{st}'_{i^*}, \text{st}'_{i^*+1}$. Thus, if $\text{Step}(\text{st}'_{i^*}) \neq \text{st}'_{i^*+1}$, then $(C, i^*) \notin \mathcal{L}'$.

In more detail, let S_i be the set of indices corresponding to st_i in the vector $\text{st}_1 \parallel \dots \parallel \text{st}_T$. We will require the somewhere statistical binding property to be at indices $S_i \cup S_{i+1}$ when the commitment key is generated in the *trapdoor mode*⁴ on input $S_i \cup S_{i+1}$. We shall shortly see why this is the case. In fact, with an eye towards a future discussion, we will actually require something stronger. Namely, generating a key in trapdoor mode on input $S_i \cup S_{i+1}$ produces a trapdoor that allows for *unique extraction* at positions $S_i \cup S_{i+1}$

⁴Keys generated in this mode (only in the security proof) are computationally indistinguishable from keys in the *normal mode*.

even for a commitment produced by an *unbounded cheating prover*. We refer to this as the *somewhere extractable* property, and use the shorthand SE to refer to it in the sequel.

From the discussion above, by setting the SE commitment to be extractable for st_i, st_{i+1} , the best that one can hope for in terms of BARG soundness is that a cheating prover is not able to produce an accepting proof when the i -th statement is false, i.e. $st_{i+1} \neq \text{Step}(st_i)$. This motivates a notion of *somewhere soundness* where the CRS for the BARG is generated on an index i such that it is hard for a cheating prover to produce an accepting proof when $(C, i) \notin \mathcal{L}'$.

It should be further noted that in our application of BARG to P-delegation, a cheating prover gets to choose the commitment c , which is hardcoded into the circuit C , effectively allowing it to adaptively pick the statements *after* the CRS is generated. This is stronger than the *non-adaptive* security for BARG considered in [CJJ21], where the statements are fixed *before* the CRS is generated. As observed in [BHK17], there are significant barriers to achieving *full adaptivity*, where the cheating prover can choose the statements *after* the CRS is generated.

We overcome this seeming conundrum by considering an intermediate notion of security that we call *semi-adaptive somewhere soundness*. We explain it here for the case of index language. Intuitively, the cheating prover must declare an index i^* of its choice before the CRS is generated. However, it can choose the circuit C after viewing the CRS. The soundness guarantee states that it will not be able to produce an accepting batch proof if $(C, i^*) \in \mathcal{L}$. More specifically, for any computationally bounded cheating prover P^* ,

$$\Pr \left[\begin{array}{l} \Pi \text{ accepting} \\ (C, i^*) \notin \mathcal{L} \end{array} \middle| \begin{array}{l} i^* \leftarrow P^* \\ \text{crs}^* \leftarrow \text{TrapdoorMode}(i^*) \\ (\Pi, C) \leftarrow P^*(\text{crs}^*) \end{array} \right] < \text{negl}(\lambda)$$

Now that we have seemingly fixed the issues raised above, how do we prove that the above scheme is secure? A natural proof strategy is the following: (1) set the index i for the trapdoor generation of the CRS for BARG, and index $S_i \cup S_{i+1}$ for the commitment key for the SE commitment; (2) extract \tilde{st}_i and \tilde{st}_{i+1} from the SE commitment using the trapdoor; (3) if $\text{Step}(\tilde{st}_i) \neq \tilde{st}_{i+1}$, but the proof Π is accepting, output (C, Π) as the cheating proof of the BARG scheme.

Let us see why this is the case. From the somewhere extractability property of the SE commitment, we know that other than with negligible probability, the extracted value is the *only* valid opening. So, if $\text{Step}(\tilde{st}_i) \neq \tilde{st}_{i+1}$, then $(C', i) \notin \mathcal{L}$, and therefore we can break the soundness of the BARG scheme for index languages.

Local vs global soundness. By the above, we are guaranteed that if the proof is accepting, then the i -th instance must be true: $(C, i) \in \mathcal{L}$, i.e. it must be the case that the extracted values do indeed satisfy C . This gives us a *local soundness* guarantee (i -th statement is true), but for the entire computation to be true, we want local soundness to hold simultaneously for all $i \in [T]$, i.e. we want *global soundness*. If one stops to think about this, our argument above for the soundness of the i -th instance crucially relied on extractability at position i and $i + 1$. For simultaneous local soundness to hold, we would require extractability at *all* positions, which is not achievable in a succinct manner (the commitment size would grow with T instead of $\text{poly}(\log T)$ as desired). One might propose an alternate hybrid strategy where one starts by proving the first instance is locally sound, then switch to proving the same for the second instance and so on. But a *local witness*, i.e. the extracted value in each case, could satisfy C even though there exists no *global witness*, i.e. \mathcal{M} does not accept x in T steps.

This problem is not new to our setting, and is in fact well documented in delegation literature starting with [KRR14]. The solution to this problem earlier works have involved using *no-signaling* proofs of some form - either no-signaling PCPs [KRR13, KRR14, KP16, BHK17, BKK⁺18, BK20], or, more recently, quasi-arguments [KPY19, GZ21].

No-signaling commitments. We take a slightly different approach and describe the notion of no-signaling with respect to SE commitments as done very recently in [GZ21]. Specifically, an extractor for an SE scheme is said to be *computationally no-signaling* if for any sets S and S' , both of size at most L , the extracted values in the intersection $S \cap S'$ have computationally indistinguishable marginal distributions whether extracted on set S or S' . Specifically, for any computationally bounded adversary \mathcal{A} , the following distributions are computationally indistinguishable:

$$\left\{ (c, y_{S \cap S'}) \left| \begin{array}{l} (K, \text{td}) \leftarrow \text{TrapdoorMode}(L, S) \\ c \leftarrow \mathcal{A}(K) \\ y = \text{Ext}(\text{td}, c) \end{array} \right. \right\} \approx \left\{ (c, y_{S \cap S'}) \left| \begin{array}{l} (K, \text{td}) \leftarrow \text{TrapdoorMode}(L, S') \\ c \leftarrow \mathcal{A}(K) \\ y = \text{Ext}(\text{td}, c) \end{array} \right. \right\}$$

[GZ21] also describe a generic compiler that transforms *any* SE scheme to a no-signaling one (NS-SE) without additional assumptions, thereby preserving the assumptions from the underlying SE commitment. We observe that the transformation also preserves the desired efficiency requirements. For completeness, we discuss the transformation in the technical sections of our paper.

It is the no signaling property of the SE, in conjunction with the BARG for index languages, that finally gives us a delegation scheme. Consider two experiments:

EXP₁: (a) the BARG CRS is generated on input i , the NS-SE key generated on $S_1 \cup S_2$; (b) extract st_1 and st_2 from the NS-SE and output it along with proof Π if Π is accepting.

EXP₂: (a) the BARG CRS is generated on input i , the NS-SE key generated on $S_2 \cup S_3$; (b) extract st'_2 and st'_3 from the NS-SE and output it along with proof Π if Π is accepting.

By our earlier argument, due of the (local) soundness of BARG, we have already established that st_2 is consistent with st_1 in EXP₁, and st'_2 with st'_3 in EXP₂. By the description of C , we additionally know that the start state st_1 is consistent with the input x , where by consistent we mean that it is the unique correct state at step 1 with respect to x . Now, the no-signaling property of the SE commitment scheme ensures that st_2 and st'_2 have computationally indistinguishable distributions. This suffices to ensure that st_2 and st'_2 must both be consistent with x , since otherwise there is an efficient distinguisher - compute \tilde{st}_2 from x and see which of the two it matches. Therefore, by the fact that st'_2 is consistent with st'_3 , st'_3 is consistent with x . By a hybrid argument we can extend this approach all the way to st_T establishing that st_T is indeed consistent with x .

Remark 1. Note that in each experiment the adversary could choose to output a different x , but the above distinguishing check is done with respect to the x output by the adversary, guaranteeing that the extracted st_i is consistent with the x that the adversary output.

From the proof size of the underlying BARG scheme, the total proof size for the delegation scheme is $\text{poly}(|st|, \log T)$. The same is true of the size of the CRS, which depends on $|C|$, and thus only on st in our setting. This ensures that the above scheme is a *delegation scheme for space bounded computation* with a short CRS. Unlike prior work [KPY19], our CRS is already “small”, and therefore we do not need an additional bootstrapping step to reduce the CRS size.

Beyond bounded space computation. To go beyond delegation for bounded space computation, we use ideas from prior works [KP16, BHK17, KPY19]. The main insight is to simulate a Turing machine \mathcal{M} with large space via a RAM machine \mathcal{R} , where the RAM machine has access to a large untrusted external memory but a small internal memory. A digest of the external memory, in the form of the root of the hash tree, is stored in the internal memory. This has two benefits: (i) the root of the hash tree is small ($\text{poly}(\lambda)$), and thus can be stored in the internal memory; and (ii) the hash tree allows for authenticated access, both read and write, to the external memory where the proof size logarithmic in the size of the external memory. Applying these ideas to our bounded space computation, we achieve a RAM delegation

protocol. Since the size of the CRS is small in the bounded space computation, it continues to be so in the RAM delegation setting.

The notion of RAM delegation we achieve is similar to that considered in [KPY19]. Here a prover is convincing the verifier that a RAM machine \mathcal{R} starting at configuration x (including the large external memory) transitions to configuration y in T steps where the verifier is only given digests h_x and h_y of the two configurations. The notion of security is that a computationally bounded cheating prover, other than with negligible probability, should not be able to produce a configuration x , digest h and proof Π such that: (i) Π is accepting for the digests (h_x, h) where h_x is digest for configuration x ; and (ii) h is *not* the digest of the (unique) configuration of \mathcal{R} T steps after x . We refer the reader to [KPY19] for a detailed discussion of the various notions of RAM delegation considered in prior works.

2.3 Non-interactive Batch Arguments for NP

Now that we have constructed a delegation scheme assuming the existence of BARGs for index languages in the CRS model, we revisit the problem of constructing such a primitive. In fact, we consider the more general case of constructing BARGs for NP.

Recall that in a BARG for NP, a prover wants to convince a verifier of the veracity of k statements (x_1, \dots, x_k) in \mathcal{L} by producing a non-interactive batch proof that is publicly verifiable, such that if *any* of the k instances are false (i.e. $\exists i$ s.t. $x_i \notin \mathcal{L}$), then a computationally bounded cheating prover should not be able to generate an accepting proof. If the witness length is $m = m(|x|)$, we require the communication to be *strictly smaller* than $k \cdot m$.

Prior Work. We know of only two solutions to this problem based on falsifiable assumptions: (i) [KPY19] construct such BARG relying on a new non-standard hardness assumption on groups with bilinear maps; and (ii) more recently [CJJ21] construct the same by assuming the hardness of the quadratic residuosity (QR) assumption in addition to either the hardness of Learning with Errors (LWE) problem, or sub-exponential hardness of the decisional Diffie-Hellman (DDH) problem. In the context of this paper, of particular interest to us is the work of [CJJ21] since they follow the Fiat-Shamir instantiation approach. (As we will soon see, however, the properties achieved by the BARG scheme of [CJJ21] does not meet our requirements from the previous subsection.)

As discussed earlier in the context of non-interactive delegation schemes for polynomial-time computation, there are challenges to starting with an interacting *argument* if we want to go the “Fiat-Shamir instantiation” approach. Instead of tackling the (seemingly harder) problem of constructing interactive batch proofs for NP, [CJJ21] choose an alternate starting point to apply the Fiat-Shamir transform. They introduce the notion of a *dual-mode interactive batch arguments* in the common reference string (CRS) model. The CRS in such protocols can be generated in two *computationally indistinguishable* modes - *normal mode* and *trapdoor mode*. We have already seen a flavor of this notion when describing the delegation scheme. For an honest protocol execution, the CRS is generated in the normal mode, while trapdoor mode is used in the proof of soundness. Specifically, in the trapdoor mode, an index i is specified during CRS generation such that if $x_i \notin \mathcal{L}$, then even a computationally unbounded cheating prover cannot provide an accepting proof.

Such a protocol provides two complementary benefits: (i) building a batch argument is easier than building a batch proof; and (ii) it allows for the possibility of instantiating the Fiat-Shamir transform in the trapdoor mode.

[CJJ21], building on the Spartan protocol [Set20], constructs such a *dual-mode interactive batch argument*. They then show that the specific dual-mode interactive batch argument constructed is Fiat-Shamir *compatible*, i.e. there exists a hash function family such that the Fiat-Shamir transformation when instantiated with this family is a sound non-interactive batch argument. The size of the batch proof in their

protocol is $\tilde{O}((|C| + \sqrt{k|C|}) \cdot \lambda)$ where $|C|$ is the size of the relation circuit for \mathcal{L} , λ is the security parameter and \tilde{O} hides factors that are poly-logarithmic in $|C|$ and k .

Our Work. In our overview of the delegation scheme for polynomial-time computation, we identified both security and efficiency properties for the BARGs we deemed essential for the construction of said delegation scheme. At the time of discussion, we also stated that the BARG constructed in [CJJ21] do not satisfy those requirements.

To achieve the desired properties in our BARG, we adopt the same “dual mode methodology” introduced in [CJJ21], but deviate from their approach in a couple of crucial aspects. First, instead of building upon the Spartan protocol, we work directly with *probabilistic checkable proofs* (PCPs), in a manner conceptually similar to Kilian’s protocol. Next, we leverage this change of approach to recurse over the number of statements (akin to [RRR16]), allowing us to depend only poly-logarithmically on the number of instances. We will elaborate on these points further when we detail the construction shortly.

To summarize our improvements over the BARG in [CJJ21]: (i) we achieve a stronger notion of security - *semi-adaptive somewhere soundness*, (ii) we significantly improve upon the size of the batch proofs to incur only poly-logarithmic dependence on k , (iii) we simplify the underlying assumptions - we no longer additionally require the quadratic residuosity (QR) assumption. We also believe that our protocol is conceptually simpler.

Before we proceed, we note that it may not be evident from the above discussion how we are able to remove the reliance on the QR assumption in [CJJ21]. At a high level, due to the reliance on the Spartan protocol, their constructed protocol necessitated the SE commitments to additionally satisfy some linear homomorphism properties, which in turn they construct assuming the hardness of QR. A consequence of our approach differing from that of [CJJ21] is that it removes the need for these additional requirements, thereby allowing us to build the required SE commitments from LWE.

Since our delegation scheme construction in the previous subsection relied on *batch arguments for index languages*, we first show how to construct them, and then provide a generic transformation from BARG for index languages to BARG for NP.

Batch Arguments for Index Languages. Recall that in a BARG for index languages, the prover is trying to prove that $(C, 1), \dots, (C, k) \in \mathcal{L}'$. To start, the prover generates a *probabilistic checkable proof* (PCP) for the instances. A PCP for a statement is a proof where the verification procedure only needs to query a few locations of the PCP to be reasonably convinced of the validity of the statement. For now, consider a PCP where the length of the PCP is $m = \text{poly}(|C|)$ and the query Q is of size $\text{polylog}(m)$, where C is the relation circuit for \mathcal{L} . The verification procedure only takes in $\text{PCP}|_Q$, the values of the PCP at the locations specified by Q .

The prover generates k PCPs $\text{PCP}_1, \dots, \text{PCP}_k$ using the corresponding witnesses w_1, \dots, w_k . It then arranges the PCPs in rows, and commits to them in a column-wise fashion. On receiving the commitment, the verifier sends the PCP query Q to the prover, who then opens the commitments of the corresponding columns. To be convinced of the proof, the verifier checks if (i) the commitment openings are valid; and (ii) all k PCP proofs verify. This high level overview is represented in Figure 1. Note that the same Q is used for *all* PCPs.

While describing the delegation scheme, we stated the requirement that the proof size be $\text{poly}(\log k, |C|)$. For large values of k , as will be the case in delegation schemes, the proof size is “significantly smaller” than the naïve solution. As in the case of our delegation scheme, we use a *succinct SE commitment with local opening* to commit to each column to bring down the communication cost.

Recall that in our discussion of the security for non-interactive BARG for index languages, we defined a notion of *semi-adaptive somewhere soundness*, which required that a cheating prover after specifying an

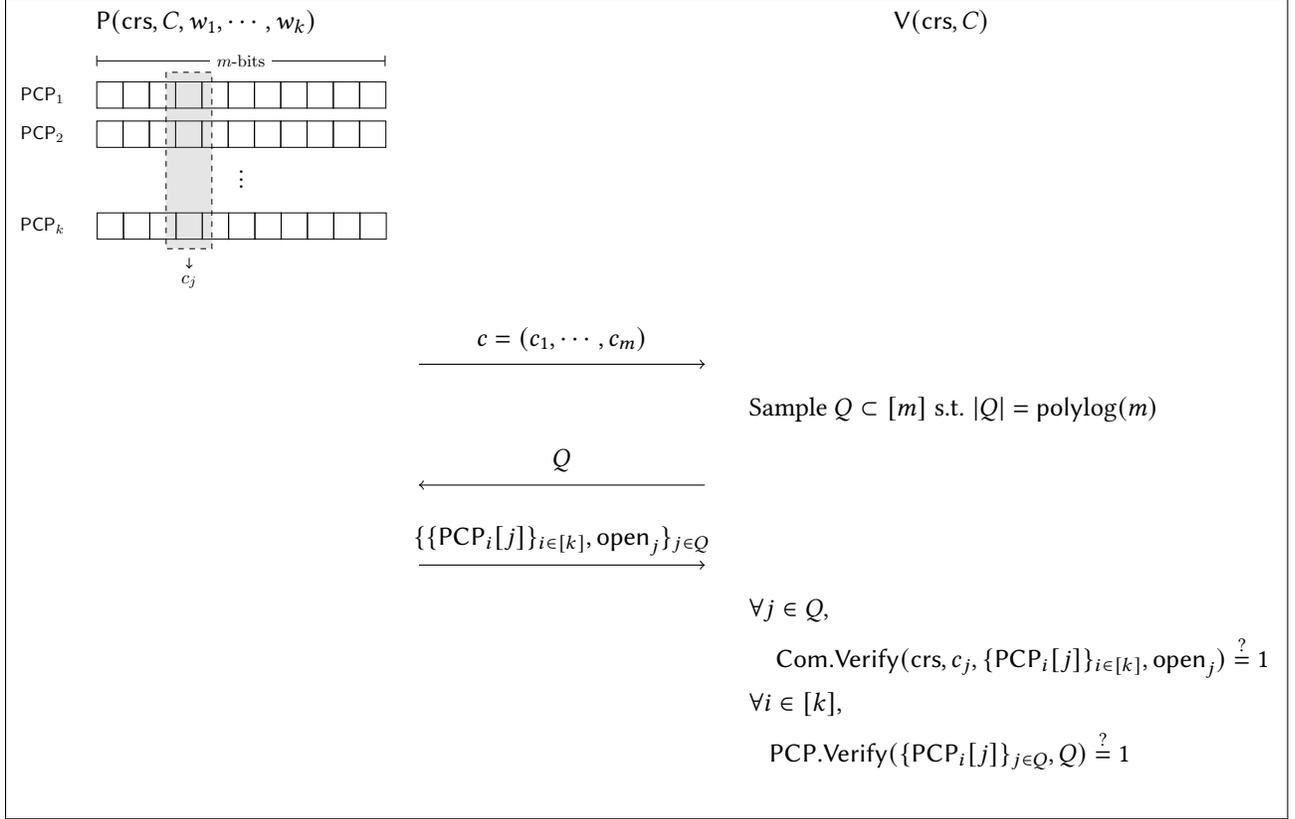


Figure 1: High level overview of initial approach

index i should not be able to produce an accepting proof when $(C, i) \notin \mathcal{L}$, where the CRS is generated in the trapdoor mode on index i . Taking the approach in [CJJ21] of *dual mode* proofs, we extend the same definition to the interactive setting, but here we allow the adversary to be unbounded once the index to the trapdoor is fixed. Specifically, for any (potentially) cheating prover P^* ,

$$\Pr \left[\begin{array}{l} \Pi \text{ accepting} \\ (C, i^*) \notin \mathcal{L} \end{array} \middle| \begin{array}{l} i^* \leftarrow P^* \\ \text{crs}^* \leftarrow \text{TrapdoorMode}(i^*) \\ (\Pi, C) \leftarrow \langle P^*(\text{crs}^*), V \rangle \end{array} \right] < \text{negl}(\lambda)$$

where $\langle P^*(\text{crs}^*), V \rangle$ indicates the interaction between the cheating prover P^* , and verifier V with output the proof Π and the circuit the prover chooses C . Note that the *mode indistinguishability*, i.e. ability to distinguish between the CRS generated for two different indices i and j is still *computational*.

Thus for security, in the trapdoor mode, the SE key for each column is generated on a set $\{i\}$, ensuring that the prover is uniquely bound to PCP_i *before* it sees the queries Q . This then allows us to rely on the (statistical) soundness of the PCP at index i .

We have thus far ignored the bottleneck in the communication of our high level description, which is the third round message consisting of commitment openings that require at least k bits of communication (length of the message committed).

We resolve this using ideas present in works such as [RRR16], by having the prover no longer explicitly sending the openings, but instead providing a BARG that it has valid openings that result in the PCP verifier accepting, and then recurse on the number of statements k . In more detail, let us consider the relation circuit \tilde{C} for this new language that the prover will have to compute a BARG for.

$$\tilde{C}$$

Hardcoded: c_1, \dots, c_m, Q ,
Witness: $\{\text{PCP}_i[j], \text{open}_{i,j}\}$
Output: Output 1 if and only if the following verify

1. Verify commitment openings:
 $\forall j, \text{Com.Verify}(c_j, i, \text{PCP}_i[j], \text{open}_{i,j}) \stackrel{?}{=} 1$
2. Verify PCP proof
 $\text{PCP.Verify}(\{\text{PCP}_i[j]\}_{j \in Q}, Q) \stackrel{?}{=} 1$

Let the witnesses for this new language be $(\tilde{w}_1, \dots, \tilde{w}_k)$. To recurse on the number of statements, we reduce the number from k to $k/2$ by pairing indices: $(1, 2), \dots, (2i-1, 2i), \dots, (k-1, k)$. The relation circuit defining the new language for the pair of statements is thus $\tilde{C}' = \tilde{C} \wedge \tilde{C}$, i.e. $(2i-1, 2i)$ has witness $(\tilde{w}_{2i-1}, \tilde{w}_{2i})$.

There are two points of note in our description of \tilde{C}' :

1. By the succinctness of the local opening for the SE commitment scheme, the size of the witnesses are $\text{poly}(\log k, \lambda)$.
2. The size of \tilde{C}' is then dominated by the size of the PCP verifier circuit, which can in fact depend (at least) linearly on the circuit size $|C|$. Therefore, from the above description, we have that $|\tilde{C}'| \geq 2 \cdot |C|$. If we were then to recurse $\log k$ times (to go from k to $O(1)$ statements), the circuit size at the last level is $\geq k \cdot |C|$. Recall that the size of the PCP, and thus the commitment sent by the prover, grows polynomially in the circuit size. This leads to total communication $\geq k \cdot |C|$, not satisfying the requirements for a BARG.

PCPs with Split Verification. To avoid the blow-up in the communication cost, we want $|\tilde{C}'|$ to be smaller than $|C|$. But this requirement seems at odds with the size of the PCP verification circuit. To navigate this issue, we introduce the notion of a PCP *with split verification*. Intuitively, for such a PCP, the verification can be split into (i) a pre-computation circuit independent of the statement, whose size can grow with $|C|$, but produces a *short* (i.e. much smaller than $|C|/2$) state st ; and (ii) an online verification circuit, which uses the pre-processed state st , statement and a proof to verify the PCP. The size of the online verification circuit, which is much smaller than $|C|/2$, can now replace PCP.Verify (along with the state st) in $|\tilde{C}'|$, allowing for $|\tilde{C}'|$ to be smaller than $|C|$.

We demonstrate that certain existing PCP schemes [BFLS91, RRR16] can be modified to satisfy the split verification property.

To recurse, we start with k instances of an index language, i.e. $(C^{(0)}, 1), \dots, (C^{(0)}, k)$, where $C^{(0)} = C$. And reduce to $k/2$ instances $(C^{(1)}, 1), \dots, (C^{(1)}, k/2)$ such that (other than with negligible probability) $(C^{(0)}, 1), \dots, (C^{(0)}, k) \in \mathcal{L}'$ if and only if $(C^{(1)}, 1), \dots, (C^{(1)}, k/2) \in \mathcal{L}'$. $C^{(0)}$ corresponds to \tilde{C}' in our description above. As discussed, the circuit size does not grow, i.e. $\forall i \geq 0, |C^{(i)}| \leq |C|$. This gives us the following recursive formula for the proof size $|\Pi|$,

$$|\Pi^{(i)}| \leq |\Pi^{(i+1)}| + |c^{(i)}|$$

where $|c^{(i)}|$ is the length of the commitment sent by the prover at the i -th step. Since $|c^{(i)}|$ depends on the length of the PCP, which in turn depends on $|C^{(i)}|$, by the succinctness property of the commitment scheme, we have $|c^{(i)}| \leq \text{poly}(\log k, |C|)$. Combined with the fact that there are $\log k$ steps of recursion, we have that the total communication is $\text{poly}(\log k, |C|)$, as required.

Applying the Fiat-Shamir Transform. Given our interactive protocol, we want to compress it to a non-interactive protocol via the Fiat-Shamir transform. As discussed earlier, crucial to this transformation is defining the set of bad verifier challenges.

$$\mathcal{B}_{C,c} = \{Q \mid \text{PCP.Verify}(C, \text{PCP}_i|_Q) = 1 \wedge (C, i) \notin \mathcal{L}\}$$

where PCP_i is extracted using the trapdoor for the SE commitment with the commitment key in the trapdoor mode generated on index i .

If we are able to demonstrate that the above set is a Cartesian product of efficiently verifiable sets, then we can apply the [HLR21] result directly, achieving a result based on LWE. At a very high level, this simply follows from the soundness amplification by parallel repetition in the PCP. Specifically, for the desired parameters, the split PCPs we consider have soundness $(1 - \varepsilon)$ for $\varepsilon = 1/\text{poly}(\log |C|)$. Since we want the soundness to be negligible, we amplify soundness by parallel repetition, generating $t = \lambda/\varepsilon$ sets of queries Q_1, \dots, Q_t for the same PCP. This gives the desired negligible soundness as $(1 - \varepsilon)^t = 2^{-\Omega(\lambda)}$. Thus we have the following set of bad challenges for negligible soundness,

$$\mathcal{B}_{C,c} = \mathcal{B}_{C,c}^{(1)} \times \dots \times \mathcal{B}_{C,c}^{(t)}$$

where for each i , $\mathcal{B}_{C,c}^{(i)} = \{Q_i \mid \text{PCP.Verify}(C, \text{PCP}_i|_{Q_i}) = 1 \wedge (C, i) \notin \mathcal{L}\}$. Each $\mathcal{B}_{C,c}^{(i)}$ is also clearly efficiently verifiable since the PCP.Verify can be used to verify if queries $Q_i \in \mathcal{B}_{C,c}^{(i)}$. One also needs to verify that $(C, i) \notin \mathcal{L}$, which can be done after extracting PCP_i if we require further properties from the PCP. We note that while the above description is not fully technically precise, it is helpful in providing the main ideas, and we refer the reader to the technical section for the details.

It should be noted that the unrolled recursion is a multi-round protocol, while the above argument considers the set of bad challenges for a single level of recursion. [CCH⁺19] showed that this approach suffices if the protocol is *round-by-round sound* which in this case intuitively means that if the (batch) claim at one level of the recursion is false, then other than with negligible probability (over the verifier's random coins), the claim remains false in the next level of recursion. Here, when we set the commitment key to be generated in the trapdoor mode for index i , if at the j -th level of recursion $(C^{(j)}, i)$ is false, then other than with negligible probability over the choice of PCP queries Q , $(C^{(j+1)}, \lceil i/2 \rceil)$ is also false.

Parameters. To summarize, the total communication is $\text{poly}(\log k, |C|)$. The size of CRS depends on the $\log k$ CRS for the correlation intractable hash functions in the Fiat-Shamir transform instantiation, one for each level of the recursion. There are also $\log k$ commitment keys that are a part of the CRS. By the specific instantiations of the correlation intractable hash family [HLR21], and SE commitment scheme [HW15], we have that for each level of the recursion, the CRS is of size $\text{poly}(\log k, |C|)$ for a total of $\text{poly}(\log k, |C|)$. This allows the verification time to also be $\text{poly}(\log k, |C|)$.

Generically obtaining BARG for NP from BARG for index languages. While we have already presented a BARG, we demonstrate how we can achieve the same result generically from *any* BARG for index languages. Recall that in a BARG for an NP language \mathcal{L} , the prover is trying to convince the verifier of the validity of k statements x_1, \dots, x_k , i.e. \mathcal{L} has a relation circuit $\mathcal{R}_{\mathcal{L}}$ such that if $x_i \in \mathcal{L}$, there exists a

witness w_i such that $\mathcal{R}_{\mathcal{L}}(x_i, w_i) = 1$. Contrast this with our discussed notion of BARG for index languages, where there is a *single* circuit C that takes in inputs i and witness w_i , and outputs 1 if $C(i, w_i) = 1$.

An immediate idea is to set $w'_i = (x_i, w_i)$ such that C implements the relation circuit $\mathcal{R}_{\mathcal{L}}$. Since our BARG for index languages allows the prover to choose any witness, the above idea allows a cheating prover to choose new statements different from x_1, \dots, x_k , thus the soundness does not translate. The next natural idea is to hardcode the statements x_1, \dots, x_k into the circuit C , which now only takes in input (i, w_i) , but still implements $\mathcal{R}_{\mathcal{L}}$. While we have solved our earlier issue, we have introduced a new one since C now grows linearly in k , and from the efficiency of the BARG scheme, so does the size of the proof. The communication issue we solve as before, by having the prover arrange the *statements* in rows, and commit to them column-wise using an SE commitment scheme. C now hardcodes the commitment c instead, where the witness additionally consists of x_i along with a proof of (local) opening.

But in solving the communication issue we seem to have reintroduced our initial concern. While the cheating prover is bound to the commitment c , there is nothing preventing them from committing to k true statements different from x_1, \dots, x_k . Our last step in the solution is for the prover to use some fixed randomness (e.g. 0) to compute the commitment. Thus, when the verifier constructs the circuit C to use during verification, it can recompute the commitment with the *true* statements x_1, \dots, x_k .

Our construction achieves some nice additional efficiency properties, and we direct the reader to the relevant technical section for the details.

Improving parameters for BARG for NP. Note that our constructed batch arguments for NP has proof size $\text{poly}(\log k, |C|)$. These parameters sufficed for the construction of the delegation scheme since there the circuit size $|C| = \text{poly}(\lambda)$, where λ is the security parameter (which we have not included thus far to avoid clutter). But in the case of BARG, we want to remove the dependence on $|C|$, since the circuit may be large. To do so, we leverage our delegation scheme for deterministic polynomial-time computations. We want to reiterate that the dependence on the circuit size was acceptable in our application of BARGs to the design of delegation scheme since the circuit there did not correspond to the computation that was being delegated but to the PCP verification, so there is no circularity of requirements here.

Consider the NP language $\mathcal{L} = \{x \mid \exists w \text{ s.t. } \mathcal{M}(x, w) \text{ outputs 1 in } T \text{ steps}\}$. The very high level idea is the following: (i) the prover generates k delegation proofs $\{\Pi_i\}_{i \in [k]}$ that \mathcal{M} outputs 1 for each of the inputs $(x_1, w_1), \dots, (x_k, w_k)$. By the efficiency of the delegation scheme, each of these proofs are of size $\text{poly}(\lambda, \log T, |w|)$; (ii) generate a BARG that the delegation verifier will accept for every i , (x_i, w_i, Π_i) .

The $|C|$ in the BARG now corresponds to the size of the delegation verifier circuit, which again by our delegation scheme is only $\text{poly}(\lambda, \log T, |w|)$ giving us the necessary efficiency properties. We have omitted several details in the high level overview, and refer the reader to the technical section for details.

3 Preliminaries

3.1 Notations

For any positive integer n , denote $[n] = \{1, 2, \dots, n\}$. For any positive integer n , any vector $x = (x_1, x_2, \dots, x_n)$, and any subset $S \subseteq [n]$, we denote $x|_S = \{x_i\}_{i \in S}$.

3.2 Low-degree Extensions

For any field \mathbb{H} and any extension field \mathbb{F} of \mathbb{H} , any index $(i_1, i_2, \dots, i_m) \in \mathbb{H}^m$, let $\widetilde{\text{Eq}}_{i_1, i_2, \dots, i_m}$ be the following polynomial over $\mathbb{F}[x_1, x_2, \dots, x_m]$.

$$\widetilde{\text{Eq}}_{i_1, i_2, \dots, i_m}(x_1, x_2, \dots, x_m) = \frac{\prod_{j_1 \in \mathbb{H} \setminus \{i_1\}} (x_1 - j_1) \cdot \prod_{j_2 \in \mathbb{H} \setminus \{i_2\}} (x_2 - j_2) \cdots \prod_{j_m \in \mathbb{H} \setminus \{i_m\}} (x_m - j_m)}{\prod_{j_1 \in \mathbb{H} \setminus \{i_1\}} (i_1 - j_1) \cdot \prod_{j_2 \in \mathbb{H} \setminus \{i_2\}} (i_2 - j_2) \cdots \prod_{j_m \in \mathbb{H} \setminus \{i_m\}} (i_m - j_m)}$$

For any string $x \in \{0, 1\}^n$, where $n = |\mathbb{H}|^m$, we identify the set \mathbb{H}^m with the index set $[n]$. Then we define the low-degree extension of x , $\text{LDE}(x)$, as the following polynomial in $\mathbb{F}[x_1, x_2, \dots, x_m]$,

$$\text{LDE}(x) = \sum_{i_1, i_2, \dots, i_m \in \mathbb{H}} x_{i_1, i_2, \dots, i_m} \cdot \widetilde{\text{Eq}}_{i_1, i_2, \dots, i_m}(x_1, x_2, \dots, x_m).$$

3.3 Learning with Error

The central cryptographic assumption we will require in our work is the Learning with Error (LWE) assumption that we define below.

Definition 1 (Learning with Error Assumption). *For any positive integers n, q , any $\mathbf{s} \in \mathbb{Z}^n$, and any error distribution χ over \mathbb{Z} , the LWE (Learning with Error) distribution $A_{\mathbf{s}, \chi}$ is defined by uniformly sampling a vector \mathbf{a} , and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $e \leftarrow \chi$.*

The $\text{LWE}_{n, q, \chi}$ assumption states that no non uniform PPT adversary can distinguish, with non-negligible probability, between (i) the distribution $A_{\mathbf{s}, \chi}$; and (ii) the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

3.4 Correlation Intractable Hash

We start by describing a hash family $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$, which is defined by the two following algorithms:

Gen: a PPT algorithm that on input the security parameter 1^λ , outputs key k .

Hash: a deterministic polynomial algorithm that on input a key $k \in \text{Gen}(1^\lambda)$, and an element $x \in \{0, 1\}^{n(\lambda)}$ outputs an element $y \in \{0, 1\}^\lambda$.

Given a hash family \mathcal{H} , we are now ready to define what it means for \mathcal{H} to be correlation intractable.

Definition 2 ([CGH04]). *A hash family $\mathcal{H} = (\mathcal{H}.\text{Gen}, \mathcal{H}.\text{Hash})$ is said to be correlation intractable (CI) for a relation family $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ if the following property holds:*

For every PPT adversary \mathcal{A} , there exists a negligible function $\text{negl}(\cdot)$ such that for every $R \in \mathcal{R}_\lambda$,

$$\Pr_{\substack{k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda) \\ x \leftarrow \mathcal{A}(k)}} [(x, \mathcal{H}.\text{Hash}(k, x)) \in R] \leq \text{negl}(\lambda).$$

CIH for Efficiently Verifiable Product Relations. We take the following definitions of product relations, and efficiently verifiable relations, from [HLR21].

Definition 3 (Product Relation, Definition 3.1 [HLR21]). *A relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ is a product relation, if for any x , the set $R_x = \{y \mid (x, y) \in R\}$ is the Cartesian product of several sets $S_{1,x}, S_{2,x}, \dots, S_{t,x}$, i.e.*

$$R_x = S_{1,x} \times S_{2,x} \times \dots \times S_{t,x}.$$

Definition 4 (Efficient Product Verifiability, Definition 3.3 [HLR21]). *A relation R is efficiently product verifiable, if there exists a circuit C such that, for any x , the sets $S_{1,x}, S_{2,x}, \dots, S_{t,x}$ (in Definition 3) satisfy that, for any $i, y_i \in S_{i,x}$ if and only if $C(x, y_i, i) = 1$.*

Definition 5 (Product Sparsity, Definition 3.4 [HLR21]). A relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ has sparsity ρ , if for any x , the sets $S_{1,x}, S_{2,x}, \dots, S_{t,x}$ (in Definition 3) satisfies $|S_{i,x}| \leq \rho|\mathcal{Y}|$.

[HLR21] show that for efficient product verifiable relations, there exists a CIH assuming only the hardness of LWE.

Theorem 3 (CIH for Efficient Product Verifiable Relations, Theorem 5.5 [HLR21]). Let $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ be a T -time product verifiable relation with sparsity at most $1 - \epsilon$, for $\epsilon \geq \lambda^{-O(1)}$. Then, if $t > \lambda/\epsilon$, there exists a hash family $\mathcal{H} = \{\mathcal{H}_\lambda : \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda^{t_\lambda}\}_\lambda$ that is correlation intractable for R under LWE assumption. Furthermore, \mathcal{H} only depends on $(\mathcal{X}_\lambda, \mathcal{Y}_\lambda, T_\lambda, t_\lambda, \epsilon)$, and can be evaluated in time $\text{poly}(\log |X|, t, T)$.

3.5 Somewhere Extractable Commitment

In this subsection, we define *somewhere extractable commitments*. A somewhere extractable commitment has a key with two computationally indistinguishable modes: (i) In the *normal mode*, the key is *uniformly random*; and (ii) in the *trapdoor mode*, the key is generated according to a subset S denoting the coordinates of the committed message.

Furthermore, we require the following properties.

- **Efficiency:** We require that the size of the CRS and commitment roughly grow with $|S|$.
- **Extraction:** The trapdoor mode commitment key is associated with a trapdoor td , such that given the trapdoor, one can extract the message on coordinates in S . Note that the extraction implies the statistical binding property for the coordinates in S .
- **Local Opening:** We allow the prover to generate a *local opening* for any single coordinate of the message. The local opening needs to have a small size, which only grows poly-logarithmically with the total length of the message. Moreover, we require that the value from the local opening should be consistent with the extracted value.

We note that this notion is essentially the same as somewhere statistical binding hash [HW15], except that we explicitly require an extraction property (although as we will see, this property is already satisfied by the construction of [HW15]). This notion is also similar to the notion of somewhere-extractable linearly homomorphic commitment in [CJJ21], except that here we do not require linear homomorphism property, but we further require local opening property.

We now move to the formal definition. A somewhere extractable commitment scheme is a tuple of algorithms (Gen, TGen, Com, Open, Verify, Ext) described below.

Gen($1^\lambda, 1^N, 1^{|S|}$): On input a security parameter, the length of the message N , and the size of a subset $S \subseteq [N]$, the “normal mode” key generation algorithm outputs a *uniformly random* commitment key K .

TGen($1^\lambda, 1^N, S$): On input a security parameter, the length of the message N , an extraction subset $S \subseteq [N]$, the “trapdoor mode” key generation algorithm outputs a commitment key K^* and a trapdoor td .

Com($K, \mathbf{m} \in \{0, 1\}^N; r$): On input the commitment key K , a vector $\mathbf{m} = (m_1, m_2, \dots, m_N) \in \{0, 1\}^N$, and the random coins r , it outputs a commitment c .

Open(K, \mathbf{m}, i, r): On input the commitment key K , a vector $\mathbf{m} = (m_1, m_2, \dots, m_N) \in \{0, 1\}^N$, an index $i \in [N]$, and the random coins r , the opening algorithm outputs a *local opening* π_i to m_i .

Verify(K, c, m_i, i, π_i): On input the commitment key K , a commitment c , a bit $m_i \in \{0, 1\}$, and a local opening π_i , the verification algorithm decides to accept (output 1) or reject (output 0) the local opening.

$\text{Ext}(c, \text{td})$: On input a commitment c , and the trapdoor td generated by the trapdoor key generation algorithm TGen with respect to the subset S , the extraction algorithm outputs an extraction string m_S^* on the subset S .

Furthermore, we require the commitment scheme to satisfy the following properties.

Succinct CRS. The size of the CRS is bounded by $\text{poly}(\lambda, |S|, \log N)$.

Succinct Commitment. The size of the commitment c is bounded by $\text{poly}(\lambda, |S|, \log N)$.

Succinct Local Opening. The size of the local opening $\pi_i \leftarrow \text{Open}(K, m, i, r)$ is bounded by $\text{poly}(\lambda, |S|, \log N)$.

Succinct Verification. The running time of the verification algorithm is bounded by $\text{poly}(\lambda, |S|, \log N)$.

Key Indistinguishability. For any non-uniform PPT adversary \mathcal{A} and any polynomial $N = N(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\left| \Pr \left[S \leftarrow \mathcal{A}(1^\lambda, 1^N), K \leftarrow \text{Gen}(1^\lambda, 1^N, 1^{|S|}) : \mathcal{A}(K) = 1 \right] - \Pr \left[S \leftarrow \mathcal{A}(1^\lambda, 1^N), (K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S) : \mathcal{A}(K^*) = 1 \right] \right| \leq \nu(\lambda).$$

Opening Completeness. For any commitment key K , any message $\mathbf{m} = (m_1, \dots, m_N) \in \{0, 1\}^N$, any randomness r , and any index $i \in [N]$, we have

$$\Pr [c \leftarrow \text{Com}(K, \mathbf{m}; r), \pi_i \leftarrow \text{Open}(K, \mathbf{m}, i, r) : \text{Verify}(K, c, m_i, i, \pi_i) = 1] = 1.$$

Extraction Correctness. For any subset $S \subseteq [N]$, any trapdoor key $(K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S)$, any commitment c , any index $i \in [N]$, any bit $m_{i^*} \in \{0, 1\}$, and any proof π_{i^*} , we have

$$\Pr [\text{Verify}(K, c, m_{i^*}, i^*, \pi_{i^*}) = 1 \Rightarrow \text{Ext}(c, \text{td})|_{i^*} = m_{i^*}] = 1.$$

Since the extracted value $\text{Ext}(c, \text{td})|_{i^*}$ is unique, the extraction correctness implies statistical binding property.

Theorem 4. *There exists a construction of somewhere extractable commitment from LWE.*

Proof Sketch. Theorem 4 is implicit in [HW15]. We briefly recall the construction of the somewhere statistical binding hash in [HW15] here. For the ease of presentation, we only describe the construction for $|S| = 1$ as in [HW15]. The construction for general S can be obtained by using multiple copies of such commitments.

The commitment key consists of a fully homomorphic encryption of the index i^* in the set S . To hash a message (m_1, m_2, \dots, m_N) , they build a Merkle Tree, where each node of the Merkle Tree is associated with a ciphertext. The leaf nodes contains the encryption of m_i 's, and for the path from m_{i^*} to the root, the ciphertext contains an encryption of m_{i^*} . This is achieved by homomorphically evaluating a circuit that selects the left or the right child according to i^* on each node of the Merkle Tree. Since the fully homomorphic encryption ciphertext is computationally indistinguishable with uniformly random string, we can use uniformly random string in the “normal mode”. The local opening follows from the Merkle Tree structure.

The extraction property is implicitly satisfied by the construction. Specifically, the trapdoor corresponds to the secret key of the fully homomorphic encryption. Given the secret key, we can decrypt the root node to extract m_{i^*} . \square

3.6 No-Signaling Somewhere Extractable Commitments

We consider here a slight variant of no-signaling somewhere extractable (NS-SE) commitments introduced in the work of [GZ21]. The no-signaling property, as described in the technical overview is imposed on the extractor of the SE commitment scheme. Intuitively, an extractor for an SE scheme is said to be *computationally no-signaling* if for any sets $S' \subseteq S$, where S is of size at most L , the extracted values corresponding to the indices in S' have computationally indistinguishable marginal distributions whether extracted on set S or S' .

Definition 6. *The extractor of an SECOM commitment scheme $(\text{Gen}, \text{TGen}, \text{Com}, \text{Open}, \text{Verify}, \text{Ext})$ is no-signaling if for any $S' \subseteq S \subseteq [N]$, where $|S| \leq L$, and any PPT adversary $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\left| \Pr \left[\mathcal{D}_2(K^*, c, \vec{y}, z) \left| \begin{array}{l} (K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S') \\ (c, z) \leftarrow \mathcal{D}_1(K^*) \\ \vec{y} := \text{Ext}(c, \text{td}) \end{array} \right. \right] - \Pr \left[\mathcal{D}_2(K^*, c, \vec{y}_{S'}, z) \left| \begin{array}{l} (K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S) \\ (c, z) \leftarrow \mathcal{D}_1(K^*) \\ \vec{y} := \text{Ext}(c, \text{td}) \end{array} \right. \right] \right| \leq \text{negl}(\lambda)$$

We will refer to SECOM schemes satisfying the above definition to be an L -no-signaling NS-SECOM commitment.

Theorem 5 ([GZ21]). *Given L instances of an SECOM commitment scheme $(\text{Gen}, \text{TGen}, \text{Com}, \text{Open}, \text{Verify}, \text{Ext})$ with locality parameter 1, one can construct an L -no-signaling NS-SECOM.*

Construction sketch. We sketch the construction from [GZ21] here, and defer details to Appendix A. For simplicity we consider here the case that $S := \{s_1, \dots, s_L\}$ has size exactly L . The rough idea is to generate L different commitment keys $K' = (K_1, \dots, K_L)$ such that to commit to a vector \vec{m} , one produces L commitments $\text{Com}(K_i, \vec{m})$ (with different randomness for each i). For the trapdoor key generation algorithm, $K'^* = (K_1^*, \dots, K_L^*)$, where each K_i^* is generated for the single element set $\{s_i\}$. Therefore the size of the keys and commitment in the L -no-signaling NS-SECOM are larger by a multiplicative factor of L .

For a full construction and proof of required properties, see Appendix A.

Preservation of succinct local opening. In our work, we will require local opening of the L -no-signaling NS-SECOM to be succinct. From the above construction it is clear that if the underlying SECOM has a succinct local opening, then the size of the succinct opening of the L -no-signaling NS-SECOM is larger by a multiplicative factor of L - one simply provides succinct local openings to each of the L underlying SECOMs.

4 Non-interactive Batch Arguments

This section is organized as follows.

- In section 4.1, we define non-interactive batch arguments (BARGs) for the circuit satisfiability language SAT.
- Next, in section 4.2, we define and construct PCPs with a split verification property which will be necessary for our construction of BARGs.

- In Section 4.3, we define BARGs for index languages. We then construct them generically from PCPs with split verification, and somewhere extractable commitments.
- Finally, in section 4.4, we construct BARGs for circuit satisfiability SAT generically from BARGs for the index languages, and somewhere extractable commitments.

4.1 Definition

Circuit Satisfiability Language. Let SAT be the following language

$$\text{SAT} = \{(C, x) \mid \exists w \text{ s.t. } C(x, w) = 1\},$$

where $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$ is a Boolean function, and $x \in \{0, 1\}^n$ is an instance.

A non-interactive batch argument for SAT is a protocol between a prover and a verifier. The prover and the verifier first agree on a circuit C , and a series of T instances x_1, x_2, \dots, x_T . Then the prover sends a single message to the verifier and tries to convince the verifier that $(C, x_1), (C, x_2), \dots, (C, x_T) \in \text{SAT}$.

More formally, such a protocol is specified by a tuple of algorithms $(\text{Gen}, \text{TGen}, \text{P}, \text{V})$ that work as follows.

- $\text{Gen}(1^\lambda, 1^T, 1^{|C|})$: On input a security parameter λ , the number of instances T , and the size of the circuit C , the CRS generation algorithm outputs crs .
- $\text{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$: On input a security parameter λ , the number of instances T , the size of the circuit C and an index i^* , the trapdoor CRS generation algorithm outputs crs^* .
- $\text{P}(\text{crs}, C, x_1, x_2, \dots, x_T, \omega_1, \omega_2, \dots, \omega_T)$: On input crs , a circuit C , and T instances x_1, x_2, \dots, x_T and their corresponding witnesses $\omega_1, \omega_2, \dots, \omega_T$, the prover algorithm outputs a proof π .
- $\text{V}(\text{crs}, C, x_1, x_2, \dots, x_T, \pi)$: On input crs , a circuit C , a series of instances x_1, x_2, \dots, x_T , and a proof π , the verifier algorithm decides to accept (output 1) or reject (output 0).

Furthermore, we require the aforementioned algorithms to satisfy the following properties.

- **Succinct Communication.** The size of π is bounded by $\text{poly}(\lambda, \log T, |C|)$.
- **Compact CRS.** The size of crs is bounded by $\text{poly}(\lambda, \log T, |C|)$.
- **Succinct Verification.** The verification algorithm runs in time $\text{poly}(\lambda, T, n) + \text{poly}(\lambda, \log T, |C|)$. Moreover, it can be split into the following two parts⁵:
 - **Pre-processing:** There exists a deterministic algorithm $\text{PreVerify}(\text{crs}, x_1, x_2, \dots, x_T)$ that takes as input the CRS, and T instances x_1, x_2, \dots, x_T , and outputs a short sketch c , where $|c| = \text{poly}(\lambda, \log T, |x_1|)$.
 - **Online Verification:** There exists an online verification algorithm $\text{OnlineVerify}(\text{crs}, c, C, \pi)$ that takes as input the sketch c , a circuit C , and a proof π , and outputs 1 (accepts) or 0 (rejects). Furthermore, the running time of the online verification algorithm is $\text{poly}(\lambda, |C|, |c|, |\pi|) = \text{poly}(\lambda, \log T, |C|)$.

⁵We note this is a stronger property than previously considered. However, it is natural, and our construction achieves this property.

- **CRS Indistinguishability.** For any non-uniform PPT adversary \mathcal{A} , and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\left| \Pr \left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \text{crs} \leftarrow \text{Gen}(1^\lambda, 1^T) : \mathcal{A}(\text{crs}) = 1 \right] - \Pr \left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \text{crs}^* \leftarrow \text{TGen}(1^\lambda, 1^T, i^*) : \mathcal{A}(\text{crs}^*) = 1 \right] \right| \leq \nu(\lambda).$$

- **Completeness.** For any circuit C , any T instances x_1, \dots, x_T such that $(C, x_1), (C, x_2), \dots, (C, x_T) \in \text{SAT}$ and witnesses $\omega_1, \omega_2, \dots, \omega_T$ for $(C, x_1), (C, x_2), \dots, (C, x_T)$, we have

$$\Pr \left[\text{crs} \leftarrow \text{Gen}(1^\lambda, 1^T, 1^{|C|}), \pi \leftarrow \text{P}(\text{crs}, C, x_1, x_2, \dots, x_T, \omega_1, \omega_2, \dots, \omega_T) : \text{V}(\text{crs}, C, x_1, x_2, \dots, x_T, \pi) = 1 \right] = 1.$$

- **Semi-Adaptive Somewhere Soundness.** For any non-uniform PPT adversary \mathcal{A} , and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that $\text{Adv}_{\mathcal{A}}^{\text{sound}}(\lambda) \leq \nu(\lambda)$, where $\text{Adv}_{\mathcal{A}}^{\text{sound}}(\lambda)$ is defined as

$$\Pr \left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \text{crs}^* \leftarrow \text{TGen}(1^\lambda, 1^T, i^*), (C, x_1, x_2, \dots, x_T, \Pi) \leftarrow \mathcal{A}(\text{crs}^*) : \right. \\ \left. i^* \in [T] \wedge (C, x_{i^*}) \notin \text{SAT} \wedge \text{V}(\text{crs}, C, x_1, x_2, \dots, x_T, \Pi) = 1 \right].$$

- **Somewhere Argument of Knowledge.** There exists a PPT extractor E such that, for any non-uniform PPT adversary \mathcal{A} , and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Pr \left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \text{crs}^* \leftarrow E(1^\lambda, 1^T, i^*), (C, x_1, x_2, \dots, x_T, \Pi) \leftarrow \mathcal{A}(\text{crs}^*), \right. \\ \left. \omega \leftarrow E(C, x_1, x_2, \dots, x_T, \Pi) : C(x_{i^*}, \omega) = 1 \right] \geq \Pr \left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \text{crs} \leftarrow \text{Gen}(1^\lambda, 1^T), \right. \\ \left. (C, x_1, x_2, \dots, x_T, \Pi) \leftarrow \mathcal{A}(\text{crs}^*) : \text{V}(\text{crs}, C, x_1, x_2, \dots, x_T, \Pi) = 1 \right] - \nu(\lambda).$$

Moreover, the CRS generated by the extractor $\text{crs}^* \leftarrow E(1^\lambda, 1^T, i^*)$ and the CRS in real execution $\text{crs} \leftarrow \text{Gen}(1^\lambda, 1^T)$ are computationally indistinguishable.

4.2 PCPs with Split Verification

In this subsection, we define PCPs with a *split verification property*. At a high level, such a property requires that for any PCP for the circuit satisfiability language

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

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

More formally, for any Boolean circuit $C : \{0, 1\}^{|x|} \times \{0, 1\}^{|w|} \rightarrow \{0, 1\}$, a PCP with split verification for C-SAT is a tuple of polynomial-time algorithms (P, Q, D) , with the following syntax.

- $P(1^\lambda, C, x, \omega)$: The prover algorithm takes as input a security parameter λ , the circuit C , an instance x and its witness ω , and outputs a PCP proof $\pi \in \{0, 1\}^*$.
- $Q(1^\lambda, C, r)$: On input the security parameter λ , the circuit C , and the random coin r , the query algorithm generates a subset $Q \subseteq [|\pi|]$, and a state st .
- $D(x, st, \pi')$: On input an instance x , a state st , and a binary string $\pi' \in \{0, 1\}^{|Q|}$, the online verification algorithm D *deterministically* decides to accept (output 1) or reject (output 0).

Furthermore, we require the following properties of the PCP.

- **Completeness.** For any circuit C , any instance $x \in \text{C-SAT}$, and any witness ω for x , we have

$$\Pr_r \left[\pi \leftarrow P(1^\lambda, C, x, \omega), (Q, st) \leftarrow Q(1^\lambda, C, r) : D(x, st, \pi|_Q) = 1 \right] = 1.$$

- $\rho(\lambda)$ -**Soundness.** For any circuit C , and any $x \notin \text{C-SAT}$, and any string $\pi^* \in \{0, 1\}^*$,

$$\Pr_r \left[(Q, st) \leftarrow Q(1^\lambda, C, r) : D(x, st, \pi^*|_Q) = 1 \right] \leq \rho(\lambda).$$

- **Polynomial Proof Size.** The size of the proof π is bounded by $\text{poly}(\lambda, |C|)$.
- **Small Query Complexity.** The size of the set Q is bounded by $\text{poly}(\lambda, \log |C|)$.
- **Succinct Verification.** The state st can be represented in $\text{poly}(\lambda, |x|, \log |C|)$ bits, and the online verification algorithm runs in time $\text{poly}(\lambda, |x|, \log |C|)$.
- ρ -**Proof of Knowledge.** For any PCP proof π^* , there exists a deterministic polynomial time extractor E such that, if $\Pr_r[(Q, st) \leftarrow Q(1^\lambda, C, r) : D(x, st, \pi^*|_Q) = 1] > \rho(\lambda)$, then $\Pr[\omega \leftarrow E(\pi^*) : C(x, \omega) = 1] = 1$.

Lemma 1. *There exists a PCP with split verification for the C-SAT language with ρ -soundness, and ρ -proof of knowledge property, where $\rho = 1 - 1/\text{poly} \log |C|$.*

Proof Sketch. We show that the PCP in [BFLS91], and the probabilistic checkable interactive proofs in [RRR16], can be modified to obtain a PCP with split verification. For any circuit C , by the Cook-Levin Theorem, there exists a 3-CNF ϕ such that for any x , $\phi(x, \cdot)$ is satisfiable if and only if $x \in \text{C-SAT}$. Furthermore, for any witness ω of $x \in \text{C-SAT}$, we can derive a witness y for $\phi(x, \cdot)$, where $|y| = O(|C|)$.

Parameters and Ingredients. Let \mathbb{H} be a field of size $\text{poly} \log |C|$, and let \mathbb{F} be a large enough extension field of \mathbb{H} with size $\text{poly} \log |C|$. Let $m_x = \lceil \log_{|\mathbb{H}|} |x| \rceil$, and $m_y = \lceil \log_{|\mathbb{H}|} |y| \rceil$. Let $m' = \lceil \log_{|\mathbb{H}|} (|x| + |y|) \rceil$, and $n = |x|$.

Let $I : \mathbb{H}^{m'} \rightarrow \{0, 1\}$ be the following polynomials.

$$I(i) = \begin{cases} 1 & i \leq m_x, \\ 0 & \text{Otherwise.} \end{cases} \quad K(i) = \begin{cases} i & i \leq m_x, \\ i - n & \text{Otherwise.} \end{cases}$$

Let \hat{I}, \hat{K} be the extension of I, K to \mathbb{F} , respectively. Then \hat{I} and \hat{K} has degree at most $\text{poly} \log |C|$. Let $\hat{x} = \text{LDE}(x), \hat{y} = \text{LDE}(y)$ be the low-degree extension of x, y over \mathbb{F} , respectively.

Let $\hat{P}(i_1, i_2, i_3, b_1, b_2, b_3)$ be the following polynomial.

$$\hat{P}(i_1, i_2, i_3, b_1, b_2, b_3) = \prod_{j \in \{1, 2, 3\}} \left(\hat{I}(i_j) \cdot \hat{x}(\hat{K}(i_j)) + (1 - \hat{I}(i_j)) \cdot \hat{y}(\hat{K}(i_j)) - b_j \right)$$

Let $C'(i_1, i_2, i_3, b_1, b_2, b_3)$ be a circuit such that $C'(i_1, i_2, i_3, b_1, b_2, b_3) = 1$ if and only if $(x_{i_1} = b_1) \vee (x_{i_2} = b_2) \vee (x_{i_3} = b_3)$ is a clause in the 3-CNF ϕ , and let \hat{C} be the extension of C' to \mathbb{F} . Then we have that $\phi(x, \cdot)$ is satisfiable, if and only if there exists a \hat{y} such that the following polynomial $F(z)$ of $3m' + 3$ variables is a zero polynomial:

$$F(z) = \sum_{i_1, i_2, i_3 \in \mathbb{H}, b_1, b_2, b_3 \in \{0,1\}} \hat{C}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \hat{P}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \widetilde{\text{Eq}}_{i_1, i_2, i_3, b_1, b_2, b_3}(z)$$

Construction Sketch. The PCP construction is the unrolling of the following interactive protocol consisting of two parts.

- **Low-Degree Testing:** The prover sends $\hat{y} = \text{LDE}(y)$. The verifier performs a low-degree test on \hat{y} . Then the verifier sends a random $\hat{z} \in \mathbb{F}^{3m'+3}$.
- **Sumcheck:** The prover and the verifier then execute a sumcheck protocol to prove $F(\hat{z}) = 0$ at the point \hat{z} specified above. At the end of the sumcheck protocol, the verifier obtains a random point $(\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3) \in \mathbb{F}^{3m'+3}$ (corresponding to its messages in the protocol) and a value $v \in \mathbb{F}$. The verifier then checks whether

$$\hat{C}(\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3) \cdot \hat{P}(\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3) \cdot \widetilde{\text{Eq}}_{\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3}(\hat{z}) = v.$$

We now describe how to fit this PCP construction into our definition of PCP with split verification.

- **PCP.Q($1^\lambda, C, r$):** We now show that the PCP queries can be generated independently of x . The PCP query consists of the queries in (i) the low-degree testing of \hat{y} ; and (ii) the sumcheck. The low-degree testing queries only query some values of \hat{y} . Hence, these queries are generated independently of x . The sumcheck protocol is public-coin. Therefore, the queries in sumcheck can also be generated independent of x .

In addition, for the sumcheck, we do the following “preprocessing” to save time in online verification. We evaluate $\hat{C}(\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3)$ in time $\text{poly}(C)$, and store the resultant value in the state st . This can be done since $\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3$ are a part of the verifier’s random coins during the sumcheck protocol.

- **PCP.D(x, st, π'):** We will show that given the state st , the verification runs in $\text{poly}(|x|, \log |C|)$ time. For the low-degree testing, the verifier performs the same verification procedure as the underlying low-degree testing. This takes time $\text{poly}(\log |C|)$.

For the sumcheck, the verifier performs the same checks as in the underlying sumcheck protocol. At the end, the verifier uses the “preprocessed” values of $\hat{C}(\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3)$ present in the state st , and computes $\hat{P}(\hat{i}_1, \hat{i}_2, \hat{i}_3, \hat{b}_1, \hat{b}_2, \hat{b}_3)$ in $\tilde{O}(|x|)$ time by computing $\{\hat{x}(\hat{K}(i_j))\}_{j=1,2,3}$ which takes time $\text{poly}(|x|, \log |C|)$.

Hence, the online verification takes time $\text{poly}(|x|, \log |C|)$ in total.

By the above running time analysis, the succinct verification property is satisfied. The small query complexity follows from the small query complexity of the low-degree testing and the sumcheck protocol. Since the sumcheck has $O(m')$ -rounds, and the prover sends $O(1)$ elements in \mathbb{F} in each round, the unrolled proof has size $|\mathbb{F}|^{O(m')} = \text{poly}(|C|)$. Hence, polynomial proof size property follows.

The completeness and soundness follows from the completeness and soundness of the zero-testing and the sumcheck protocol. The proof of knowledge property follows from the decoding of \hat{y} . \square

Next, we define the *bad* relation for any PCP with split verification with an eye towards our BARG construction we describe next. As described in the overview in section 2, we commit several PCP proofs “columnwise” using a somewhere extractable commitment and apply a CIH to these commitments to obtain the query PCP Q .

In the soundness proof, we first switch the commitment key to the trapdoor mode. The *bad* relation is defined with respect to the trapdoor td of the commitment. Specifically, we can use td to extract a PCP proof π from the commitment. Now given the extracted proof π , we define a query Q to be bad, when $\pi|_Q$ is accepting but we cannot extract a witness from π . However, the verification algorithm not only needs Q , but also the state st . To resolve this issue, in the following definition, we have the CIH output the randomness r . We then use this randomness to generate Q and st via $PCP.Q$.

Definition 7 (Bad relation for PCP). *Let $SECOM = (SECOM.Gen, SECOM.TGen, SECOM.Com, SECOM.Open, SECOM.Verify, SECOM.Ext)$ be a somewhere extractable commitment scheme, and $PCP = (P, Q, D)$ be any PCP with split verification, we define the bad relation $\mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ for PCP as follows.*

For any instance length $n = n(\lambda)$, witness length $m = m(\lambda)$, proof length $\ell = \ell(\lambda)$, and a parameter $T = T(\lambda)$, we define the bad relation for PCP as $\mathcal{R}_\lambda = \{R_{\lambda,x,td}\}$, where td is obtained from $(K^, td) \leftarrow SECOM.TGen(1^\lambda, 1^T, i^*)$ for a index $i^* \in [T]$, and*

$$R_{\lambda,x,td} = \{((C, c), r) \mid C(x, E(\pi)) \neq 1 \wedge D(x, st, \pi|_Q) = 1\},$$

where $(Q, st) = Q(1^\lambda, C, r)$, $c = \{c_q\}_{q \in [\ell]}$, $\pi = \{SECOM.Ext(c_q, td)\}_{q \in [\ell]}$, $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$ is a Boolean circuit, and x is a string of length n , and E is the proof of knowledge extractor.

Theorem 6 (CIH for PCP). *There exists a PCP with split verification (P, Q, D) and a hash family \mathcal{H} such that, \mathcal{H} is correlation intractable for its bad relation family $\mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ (in Definition 7). Furthermore, \mathcal{H} can be evaluated in time $\text{poly}(\lambda, \log T, |C|)$*

Proof. Intuitively, we will take the PCP in Lemma 1, and repeat its verification several times in parallel (with independent randomness), and apply Theorem 3 to the resulting PCP.

Let $PCP' = (PCP'.P, PCP'.Q, PCP'.D)$ be the $(1 - \epsilon)$ -sound PCP with split verification from Lemma 1, where $\epsilon = 1/\text{poly} \log |C|$. We build a new PCP $= (P, Q, D)$ as follows.

- P is the same as $PCP'.P$.
- $Q(1^\lambda, C, r)$: Parse $r = (r_1, r_2, \dots, r_t)$, where $t = \lambda/\epsilon$.
 - For each $i \in [t]$, let $(Q_i, st_i) = PCP'.Q(1^\lambda, C, r_i)$.
 - Output $Q = (Q_1, Q_2, \dots, Q_t)$, $st = (st_1, st_2, \dots, st_t)$.
- $D(x, st, \pi')$ Parse $\pi' = (\pi'_1, \pi'_2, \dots, \pi'_t)$, and $st = (st_1, st_2, \dots, st_t)$.
 - For each $i \in [t]$, verify if $PCP'.D(x, st_i, \pi'_i) = 1$.
 - If all verification passes, then output 1 (accept). Otherwise output 0 (reject).
- Proof of knowledge Extractor E : We use the proof of knowledge extractor of PCP' as the extractor for PCP.

The resultant PCP satisfies $\rho = (1 - \epsilon)^t = 2^{-\Omega(\lambda)}$ -soundness and ρ -proof of knowledge property. By construction, for each security parameter λ , instance x , and trapdoor td , $R_{\lambda,x,td}$ is a product relation, since the bad relation for PCP is the product of the bad relations for PCP' . The bad relation for PCP' is efficiently verifiable in time $\text{poly}(\lambda, \log T, |C|)$.

To demonstrate sparsity, for any instance x and extracted PCP proof π , if $C(x, E(\pi)) \neq 1$, then $\Pr_r[(Q, st) \leftarrow Q(1^\lambda, C, r) : D(x, st, \pi|_Q) = 1] \leq \rho$, otherwise this contradicts the ρ -soundness of PCP. Since our construction is a parallel repetition,

$$\Pr_r[(Q, st) \leftarrow Q(1^\lambda, C, r) : D(x, st, \pi|_Q) = 1] = \Pr_r[(Q, st) \leftarrow \text{PCP}'.Q(1^\lambda, C, r) : \text{PCP}'.D(x, st, \pi|_Q) = 1]^t.$$

Hence, if the left hand is bounded by ρ , then we have

$$\Pr_r[(Q, st) \leftarrow \text{PCP}'.Q(1^\lambda, C, r) : \text{PCP}'.D(x, st, \pi|_Q) = 1] \leq 1 - \epsilon.$$

Hence, the relation $R_{\lambda, \text{td}}$ has sparsity $(1 - \epsilon)$. Therefore, by Theorem 3, there exists a correlation intractable hash family \mathcal{H} for \mathcal{R} . \square

4.3 BARGs for Index Languages

Index Language. Let the index language be the following language

$$\text{Idx} = \{(C, i) \mid \exists w \text{ s.t. } C(i, w) = 1\},$$

where C is a Boolean function, and i is an index.

Non-interactive batch arguments for the index language is a special case of BARGs for general circuit satisfiability when the instances x_1, x_2, \dots, x_T are simply the indices $1, 2, \dots, T$. We therefore omit x_1, x_2, \dots, x_T as inputs to the prover and the verifier algorithms BARG.P, BARG.V (and also as an output of the adversary \mathcal{A} describing the semi-adaptive somewhere soundness property). Furthermore, since the verifier does not need to read the instances, there is no pre-processing in this case, and the succinct verification property requires the verifier to run in time $\text{poly}(\lambda, \log T, |C|)$.

In this subsection, we mainly prove the following theorem.

Theorem 7 (BARGs for Index Language). *Assuming LWE, there exist batch arguments for the index language with succinct verification property.*

We proceed to describe the construction of the BARGs for the index language.

Ingredients. Our construction is recursive. We use an index L to index the level of recursion. Note that we can assume without loss of generality that T is a power of two by padding the last instance $(2^{\lceil \log_2 T \rceil} - T)$ times. The BARGs at the L -level can handle $T = 2^L$ instances. To construct $\text{BARG}_L = (\text{Gen}, \text{TGen}, \text{P}, \text{V})$ at the L -level, we need the following ingredients.

- A somewhere extractable commitment (section 3.5) $\text{SECOM} = (\text{SECOM.Gen}, \text{SECOM.TGen}, \text{SECOM.Com}, \text{SECOM.Open}, \text{SECOM.Verify}, \text{SECOM.Ext})$.
- A PCP with split verification scheme $\text{PCP} = (\text{PCP.P}, \text{PCP.Q}, \text{PCP.D})$ with proof length $\ell = \ell(\lambda, |C|)$ from Theorem 6.
- A CIH (definition 2) $\mathcal{H} = (\mathcal{H.Gen}, \mathcal{H.Hash})$ for the bad relation of PCP with split verification from Theorem 6.
- A non-interactive batch arguments at the $(L - 1)$ -level $\text{BARG}_{L-1} = (\text{BARG}'.\text{Gen}, \text{BARG}'.\text{TGen}, \text{BARG}'.\text{P}, \text{BARG}'.\text{V})$.

$$\text{Circuit } \widehat{C}_{[K,Q,\{c_q\}_{q \in Q},st]}(i, (\vec{\rho}, \pi', \vec{\rho}', \pi''))$$

$$\text{Output } \widetilde{C}_{[K,Q,\{c_q\}_{q \in Q},st]}(2i-1, \vec{\rho}, \pi') \wedge \widetilde{C}_{[K,Q,\{c_q\}_{q \in Q},st]}(2i, \vec{\rho}', \pi'').$$

Figure 2: The grouped new circuit, where the ungrouped new circuit \widetilde{C} is depicted in Figure 3.

$$\text{Circuit } \widetilde{C}_{[K,Q,\{c_q\}_{q \in Q},st]}(i, \vec{\rho}, \pi')$$

Hardwired: The commitment key K , the set Q , the commitments $\{c_q\}_{q \in Q}$, and the state st for PCP verification.

Parse the input $\vec{\rho} = \{\rho_q\}_{q \in Q}$, and $\pi' = \{\pi'_q\}_{q \in Q}$.

- For each $q \in Q$, verify the opening π'_q to the commitment c_q . Specifically, verify

$$\forall q \in Q, \text{SECOM.Verify}(K, c_q, \pi'_q, i, \rho_q) = 1.$$
- Verify π' is accepted by the PCP online verification, i.e. verify $\text{PCP.D}(st, i, \pi') = 1$.
- If all verification passes, then output 1 (accept), otherwise output 0 (reject).

Figure 3: The ungrouped new circuit.

Construction. We proceed to describe the construction. In the base case $L = 0$ and $T = 1$, we have the prover send the witness directly to the verifier, and have the verifier verify the witness. When $L \geq 1$, we reduce the batch argument to verify a batch of $T/2$ instances, and apply the $(L-1)$ -level BARG recursively. In more detail, we construct BARG for $L \geq 1$ as follows.

- $\text{Gen}(1^\lambda, 1^{T=2^L}, 1^{|\mathcal{C}|})$: The CRS generation algorithm generates a CRS, which contains (i) a somewhere extractable commitment key; (ii) a CRS for the smaller non-interactive batch arguments BARG_{L-1} ; and (iii) a key for the CIH \mathcal{H} .
 - Let $K \leftarrow \text{SECOM.Gen}(1^\lambda, 1^T, 1^1)$, $\text{crs}' \leftarrow \text{BARG}'.\text{Gen}(1^\lambda, 1^{T'}, 1^{|\widehat{\mathcal{C}}|})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda)$, where $T' = T/2$.
 - Let $\text{crs} = (K, \text{crs}', \mathcal{H}.k)$ and output crs .
- $\text{TGen}(1^\lambda, 1^T, 1^{|\mathcal{C}|}, i^*)$: The trapdoor CRS generation algorithm generates the trapdoor CRS as follows.
 - Generate $(K^*, \text{td}) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$,
 - Let $\text{crs}^{*'} \leftarrow \text{BARG}.\text{TGen}(1^\lambda, 1^{T'}, 1^{|\widehat{\mathcal{C}}|}, \lfloor (i^* + 1)/2 \rfloor)$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda)$.
 - Let $\text{crs}^* = (K^*, \text{crs}^{*'}, \mathcal{H}.k)$, and output crs^* .
- $\text{P}(\text{crs}, C, \omega_1, \omega_2, \dots, \omega_T)$: The prover algorithm first commits to all PCP strings in a ‘‘columnwise’’ manner, and then applies the CIH to the commitment.

- For each $i \in [T]$, compute the PCP proof $\pi_i \leftarrow \text{PCP.P}(1^\lambda, C, i, \omega_i)$ for i -th instance (C, i) .
- Committing the $\pi = \{\pi_i\}_{i \in [T]}$ “columnwise”,

$$\forall q \in [\ell], c_q \leftarrow \text{SECOM.Com}(K, \{\pi_i|_q\}_{i \in [T]}; r_q),$$

with uniformly random r_q .

- Applying the CIH to $c = \{c_q\}_{q \in [\ell]}$, $r \leftarrow \mathcal{H}.\text{Hash}(\mathcal{H}.k, (C, c))$, and let $(Q, \text{st}) \leftarrow \text{PCP.Q}(1^\lambda, C, r)$.
- For each $i \in [T]$, let $\vec{\rho}_i$ be the opening of $\pi_i|_Q$. Specifically,

$$\vec{\rho}_i = \{\text{SECOM.Open}(K, \{\pi_i|_q\}_{i \in [T]}, i, r_q)\}_{q \in Q}.$$

- Compute a smaller BARG proof, let

$$\Pi' \leftarrow \text{BARG}'.\text{P}(\text{crs}', \widehat{C}_{[K, Q, \{c_q\}_{q \in Q}, \text{st}]}, \{\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q, \vec{\rho}_{2i}, \pi_{2i}|_Q\}_{i \in [T']}),$$

where \widehat{C} is depicted in Figure 2.

- Output the proof $\Pi = (c, \Pi')$.
- $\text{V}(\text{crs}, C, \Pi)$: The verification algorithm parses the proof Π as the commitment and the proof for the smaller BARG, then it utilizes the split verification property of the PCP to delegate the online verification to the smaller BARG.
 - Parse $\Pi = (c, \Pi')$. Applying CIH to c , let $r \leftarrow \mathcal{H}.\text{Hash}(\mathcal{H}.k, (C, c))$.
 - Generate the PCP query, $(Q, \text{st}) \leftarrow \text{PCP.Q}(1^\lambda, C, r)$.
 - Verify the smaller BARG, output $\text{BARG}'.\text{V}(\text{crs}', \widehat{C}_{[K, Q, \{c_q\}_{q \in Q}, \text{st}]}, \Pi')$.

Before analysing the efficiency, we first bound the size of the circuit $\widehat{C}_{[K, Q, \{c_q\}_{q \in Q}, \text{st}]}$.

Lemma 2. *In the construction of \widehat{C} in Figure 2, we have*

$$|\widehat{C}_{[K, Q, \{c_q\}_{q \in Q}, \text{st}]}| = \text{poly}(\lambda, \log T, \log C).$$

Proof. By the construction of \widehat{C} , since $2i - 1$ and $2i$ can be computed by a circuit of size $O(\log T)$, we have $|\widehat{C}| = O(|\widetilde{C}|)$. For \widetilde{C} , we analyse the size of the circuit computing each step.

- First, \widetilde{C} verifies if the opening π'_q is accepted with respect to c_q . By the succinct verification property of the somewhere extractable commitment, this step can be computed by a circuit of size $\text{poly}(\lambda, \log T) \cdot |Q|$. By the small query complexity of the PCP, this term is bounded by $\text{poly}(\lambda, \log T, \log C)$.
- Second, \widetilde{C} verifies the PCP proof. By the succinct verification property of PCP. This step can be computed by a circuit of size $\text{poly}(\lambda, \log T, \log C)$.

Since each step of \widetilde{C} can be computed by a circuit of size $\text{poly}(\lambda, \log T, \log C)$, which completes the proof. \square

Lemma 3 (Succinct Proof). *The aforementioned construction satisfies the succinct proof property.*

Proof. We analyse the length of the proof recursively. By construction, we have

$$|\Pi_L| = |c| + |\Pi_{L-1}|,$$

where Π_{L-1} is the proof length of the $(L - 1)$ -level BARG. By the succinct commitment property, the size of c is bounded by $\text{poly}(\lambda, \log T) \cdot \ell$. Since the length of the PCP proof is bounded by $\text{poly}(|C|)$, we have that $\ell = \text{poly}(|C|)$. Hence, $|\Pi_L| = \text{poly}(\lambda, \log T, |C|) + |\Pi_{L-1}|$. Hence, recursively applying Lemma 2, we have $\Pi_L = \text{poly}(\lambda, \log T, |C|)$. \square

Lemma 4 (Succinct Verification). *The aforementioned construction satisfies succinct verification property.*

Proof. Let Time_L and Time_{L-1} be the verification time for BARG_L and BARG_{L-1} respectively. Then we have

$$\text{Time}_L = \text{Time}_{\mathcal{H}.\text{Hash}} + \text{Time}_{\text{PCP.Q}} + \text{Time}_{L-1},$$

where $\text{Time}_{\mathcal{H}.\text{Hash}}$ is the running time of $\mathcal{H}.\text{Hash}$, and $\text{Time}_{\text{PCP.Q}}$ is the running time of PCP.Q . From Theorem 6, $\text{Time}_{\mathcal{H}.\text{Hash}} = \text{poly}(\lambda, \log T, |C|)$. From the PCP construction, PCP.Q needs to be a polynomial time algorithm. Hence, we have $\text{Time}_{\text{PCP.Q}} = \text{poly}(\lambda, |C|)$. Therefore,

$$\text{Time}_L = \text{poly}(\lambda, \log T, |C|) + \text{Time}_{L-1}.$$

Recursively applying this equation, we obtain $\text{Time} = \text{poly}(\lambda, \log T, |C|)$. \square

Lemma 5 (Compact CRS). *The aforementioned construction satisfies compact CRS property.*

Proof. By construction,

$$|\text{crs}_L| = |K| + |\text{crs}_{L-1}| + |\mathcal{H}.k|,$$

where crs_L and crs_{L-1} are the CRS of the BARG at the L -th level and $(L - 1)$ -th level respectively. From succinct CRS property of the commitment scheme, we have $|K| = \text{poly}(\lambda, \log T)$. The size of the hash key is bounded by the running time of $\mathcal{H}.\text{Hash}$, which is $\text{poly}(\lambda, \log T, |C|)$. Hence, we have

$$|\text{crs}_L| = \text{poly}(\lambda, \log T, |C|) + |\text{crs}_{L-1}|.$$

Recursively applying this equation, by Lemma 2, we obtain $|\text{crs}| = \text{poly}(\lambda, \log T, |C|)$. \square

Lemma 6 (CRS indistinguishability). *The aforementioned construction satisfies CRS indistinguishability.*

Proof. Since the construction is recursive, we prove the CRS indistinguishability by induction. In the base case, when $T = 1$, the CRS generated by Gen and TGen is clearly indistinguishable. Now, assuming BARG_{L-1} satisfies CRS indistinguishability, we prove the CRS indistinguishability of BARG by the following hybrid arguments.

- Hyb₀: Let $\text{crs} \leftarrow \text{Gen}(1^\lambda, 1^T, 1^{|C|})$. Output crs .
- Hyb₁: Let $(K^*, \text{td}) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$, $\text{crs}' \leftarrow \text{BARG.Gen}(1^\lambda, 1^T, 1^{|\widehat{C}|})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.Gen(1^\lambda)$. Output $\text{crs} = (K^*, \text{crs}', \mathcal{H}.k)$.

This hybrid is computationally indistinguishable with Hyb_0 , from the key indistinguishability of the commitment scheme SECOM.

- Hyb₂: Let $(K^*, \text{td}) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$, $\text{crs}^{*'} \leftarrow \text{BARG.Gen}(1^\lambda, 1^T, 1^{|\widehat{C}|}, \lfloor (i^* + 1)/2 \rfloor)$, and $\mathcal{H}.k \leftarrow \mathcal{H}.Gen(1^\lambda)$. Output $\text{crs} = (K^*, \text{crs}^{*'}, \mathcal{H}.k)$.

This hybrid is computationally indistinguishable with Hyb_1 , from the CRS indistinguishability of the smaller BARG. This hybrid is identical to $\text{crs}^* \leftarrow \text{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$.

By the hybrid argument, we finish the proof. \square

Lemma 7 (Completeness). *The aforementioned construction satisfies the completeness property.*

Proof. We prove the completeness by induction. For the base case $L = 0, T = 1$, since the prover sends the witness directly, the completeness is satisfied. Now, assuming the BARG_{L-1} satisfies the completeness, we need to show the completeness of BARG_L . Since the verification algorithm of BARG_L invokes BARG_{L-1} and verifies Π' , it suffices to show that $\{\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q, \vec{\rho}_{2i}, \pi_{2i}|_Q\}_{i \in [T']}$ are the witnesses for $\widehat{C}_{[K, Q, \{c_q\}_{q \in Q}, \text{st}]}$. According to the construction, to prove this we only need to show $(\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q)$ is a witness for $\widehat{C}_{[K, c, C]}$. This follows from the completeness of the local opening and completeness of the PCP. \square

Before proving the semi-adaptive somewhere soundness, we first show in Lemma 8 that any adversary for the semi-adaptive somewhere soundness of BARG_L can be used to build a new adversary for the semi-adaptive somewhere soundness of BARG_{L-1} . Then in Lemma 9, we will apply this Lemma recursively to prove the semi-adaptive somewhere soundness of BARG_L .

Lemma 8. *For any level L and non-uniform PPT adversary \mathcal{A} for the semi-adaptive somewhere soundness of BARG_L with advantage $\text{Adv}_{\mathcal{A}}^{\text{sound}}(\lambda)$, there exists a non-uniform PPT adversary \mathcal{A}' for BARG_{L-1} such that $\text{Adv}_{\mathcal{A}'}^{\text{sound}}(\lambda) \geq \text{Adv}_{\mathcal{A}}^{\text{sound}}(\lambda) - \text{negl}(\lambda)$. Furthermore, let $\text{Time}_{\mathcal{A}}$ and $\text{Time}_{\mathcal{A}'}$ be the running time of \mathcal{A} and \mathcal{A}' respectively. Then $\text{Time}_{\mathcal{A}'} = \text{Time}_{\mathcal{A}} + \text{poly}(\lambda)$.*

Proof. We build the adversary \mathcal{A}' trying to break the semi-adaptive somewhere soundness for the underlying $(L-1)$ -level BARG scheme BARG_{L-1} as follows.

- $\mathcal{A}'(1^\lambda, 1^T)$: Invoke the adversary $i^* \leftarrow \mathcal{A}(1^\lambda, 1^T)$. Output $i^{*'} = \lfloor (i^* + 1)/2 \rfloor$.
- $\mathcal{A}'(\text{crs}^{*'})$: The adversary generates crs for \mathcal{A} , and obtains the proof from \mathcal{A} .
 - Generate $(K^*, \text{td}) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.Gen(1^\lambda)$.
 - Compose the CRS for BARG, let $\text{crs}^* = (K^*, \text{crs}^{*'}, \mathcal{H}.k)$. Feed it to \mathcal{A} , $(C, \Pi) \leftarrow \mathcal{A}(\text{crs}^*)$.
 - Parse $\Pi = (c, \Pi')$. Applying CIH to c , $r \leftarrow \mathcal{H}.Hash(\mathcal{H}.k, c)$, and $(Q, \text{st}) \leftarrow \text{PCP.Q}(1^\lambda, C, r)$.
 - Output $(\widehat{C}_{[K, Q, \{c_q\}_{q \in Q}, \text{st}]}, \Pi')$.

To argue $\text{Adv}_{\mathcal{A}'}^{\text{sound}}(\lambda) \geq \text{Adv}_{\mathcal{A}}^{\text{sound}}(\lambda) - \text{negl}(\lambda)$, we build the following adversary \mathcal{B} for the correlation intractable hash family \mathcal{H} .

First, the adversary \mathcal{B} invokes the adversary \mathcal{A} by executing $i^* \leftarrow \mathcal{A}(1^\lambda, 1^{T=2^L})$. Then it generates a SECOM key $(K^*, \text{td}) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$. The adversary then chooses the bad relation $R_{\lambda, i^*, \text{td}} \in \mathcal{R}_\lambda$ to break the correlation intractability. Next, \mathcal{B} is given a CIH key $\mathcal{H}.k$. \mathcal{B} generates the CRS for the smaller BARG $\text{crs}^{*'} \leftarrow \text{BARG.TGen}(1^\lambda, 1^{T'=T/2}, 1^{|\widehat{C}|}, \lfloor (i^* + 1)/2 \rfloor)$, and composes the CRS $\text{crs} = (K^*, \text{crs}^{*'}, \mathcal{H}.k)$, and feeds it to \mathcal{A} . Let $(C, \Pi) \leftarrow \mathcal{A}(\text{crs})$. Parse $\Pi = (c, \Pi')$. Output (C, c) .

By the correlation intractability, on the one hand we have that

$$\Pr \left[\mathcal{H}.k \leftarrow \mathcal{H}.Gen(1^\lambda), (C, c) \leftarrow \mathcal{B}(\mathcal{H}.k) : ((C, c), \mathcal{H}.Hash(\mathcal{H}.k, (C, c))) \in R_{\lambda, i^*, \text{td}} \right] \leq \text{negl}(\lambda). \quad (1)$$

But on the other hand,

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{sound}}(\lambda) &= \Pr [(C, i^*) \notin L \wedge \forall (\text{crs}, C, \Pi) = 1] \\ &= \Pr \left[(C, i^*) \notin L \wedge \forall (\text{crs}, C, \Pi) = 1 \wedge (\widehat{C}, i^{*'}) \in L \right] + \Pr \left[(C, i^*) \notin L \wedge \forall (\text{crs}, C, \Pi) = 1 \wedge (\widehat{C}, i^{*'}) \notin L \right]. \end{aligned}$$

By the definition of $\text{Adv}_{\mathcal{A}'}^{\text{sound}}(\lambda)$, we have that $\text{Adv}_{\mathcal{A}'}^{\text{sound}}(\lambda) \geq \Pr \left[(C, i^*) \notin L \wedge \forall (\text{crs}, C, \Pi) = 1 \wedge (\widehat{C}, i^{*'}) \notin L \right]$.

Hence, it suffices to bound the first term $\Pr[(C, i^*) \notin L \wedge \forall (\text{crs}, C, \Pi) = 1 \wedge (\widehat{C}, i^{*'}) \in L]$. We have

$$\begin{aligned} \Pr[(C, i^*) \notin L \wedge (\widehat{C}, i^{*'}) \in L] &= \Pr[(C, i^*) \notin L \wedge \widehat{C}(i^{*'}, \cdot) \text{ is satisfiable}] \\ &\leq \Pr[(C, i^*) \notin L \wedge \widetilde{C}(i^*, \cdot) \text{ is satisfiable}]. \end{aligned}$$

Note that in the last term, if $\widetilde{C}(i^*, \cdot)$ is satisfiable, then there exists $\vec{\rho}, \pi'$ such that $\vec{\rho}$ is the opening of π' and the PCP verification of π' accepts. By the extraction correctness of SECOM, π' here should be equal to the $\{\text{SECOM.Ext}(c_q, \text{td})\}_{q \in [\ell]}$ in the bad relation $R_{\lambda, i^*, \text{td}}$ definition. Hence, $\text{PCP.D}(i^*, \text{st}, \pi|_Q) = 1$, which implies that $((C, c), r) \in R_{\lambda, i^*, \text{td}}$. By correlation intractability (Equation 1), this event is bounded by a negligible probability, which completes the proof. \square

Lemma 9 (Semi-adaptive Somewhere Soundness). *The aforementioned construction satisfies the semi-adaptive somewhere soundness.*

Proof. We recursively apply the Lemma 8 $L = \log_2 T$ times. At the end, we obtain a polynomial time adversary for the 0-level base case construction. Since the base case protocol has the prover send the witness directly, it is statistically sound. Hence, we reach a contradiction, which completes the proof. \square

Remark 2. *The proof of Lemma 9 can be extended to show somewhere argument of knowledge property. We provide a proof sketch here. For the base case $L = 0, T = 1$, since the prover sends the witness, we can extract the witness from the proof directly. For any $L \geq 1$, we firstly use the extractor of SECOM to extract the PCP proof π as in the soundness proof, and then apply the proof of knowledge property of the PCP to obtain a witness.*

4.4 BARGs for NP

In this subsection, we present a generic approach to generalize the BARG for index language and obtain BARGs for SAT.

Theorem 8. *If there exists a batch argument $\text{BARG}' = (\text{BARG}'.\text{Gen}, \text{BARG}'.\text{TGen}, \text{BARG}'.\text{P}, \text{BARG}'.\text{V})$ for the index language Idx , then there exists a batch argument $\text{BARG} = (\text{Gen}, \text{TGen}, \text{P}, \text{V})$ for SAT with succinct verification property.*

Proof Sketch. We construct the BARG as follows.

Circuit $C'_{[K, c, C]}(i, \{\rho_j\}_{j \in [n]}, x, \omega)$

Hardwired: The commitment key K , the commitments $c = \{c_j\}_{j \in [n]}$, and a circuit C .

- Verify the opening $\{\rho_j\}_{j \in [n]}$ and the instance x are the i -th coordinate of c .

$\forall j \in [n]$, verify $\text{SECOM.Verify}(K, c_j, x[j], i, \rho_j) = 1$.

- Verify the C-SAT: verify if $C(x, \omega) = 1$.

If all verification passes, then output 1. Otherwise, output 0.

Figure 4: The new circuit C' for batch argument.

- $\text{Gen}(1^\lambda, 1^T, 1^{|C|})$: When generating the CRS, in addition to generating the CRS for BARG' , we also generate a somewhere extractable commitment key K .
 - Let $\text{crs}' \leftarrow \text{BARG}'.\text{Gen}(1^\lambda, 1^T, 1^{|C|})$, where the circuit C' is depicted in Figure 4.
 - Let $K \leftarrow \text{SECOM.Gen}(1^\lambda, 1^T, 1^1)$ be a somewhere extractable commitment key with set size 1.
 - Output $\text{crs} = (K, \text{crs}')$.

- $\text{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$: In the trapdoor CRS generation algorithm, we generate the trapdoor CRS of BARG', and also generate the trapdoor somewhere extractable commitment key which is extractable at i^* -th coordinate.

- Let $\text{crs}^{*'} \leftarrow \text{BARG}'.\text{Gen}(1^\lambda, 1^T, 1^{|C|}, i^*)$ be the trapdoor CRS of BARG' for i^* .
- Let $K^* \leftarrow \text{SECOM}.\text{Gen}(1^\lambda, 1^T, 1^1, \{i^*\})$ be the trapdoor somewhere extractable commitment key.
- Output $\text{crs}^* = (K^*, \text{crs}^{*'})$.

- $\text{P}(\text{crs}, C, x_1, \dots, x_T, \omega_1, \dots, \omega_T)$: The prover first uses the somewhere extractable commitment to commit all the instances, and obtain a short commitment c . Then we have the prover use BARG' to prove the statement: “for each $i \in [T]$, there exists an accepting local opening x_i at the i -th coordinate for the commitment c , and there exists a witness ω_i such that $C(x_i, \omega_i) = 1$ ”.

- Parse $\text{crs} = (K, \text{crs}')$, where K is a somewhere extractable commitment key, and crs' is a CRS for BARG'.
- Recall that $n = |x_1|$ is the length of the instances. Commit the instances by

$$\forall j \in [n], c_j := \text{SECOM}.\text{Com}(K, x_1[j], x_2[j], \dots, x_T[j]; 0)^6,$$

where $x_i[j]$ is the j -th bit of the string x_i . Let $c = \{c_j\}_{j \in [n]}$.

- For each $i \in [T], j \in [n]$, generate the opening $\rho_{i,j} \leftarrow \text{SECOM}.\text{Open}(K, (x_1[j], \dots, x_T[j]), i)$.
- Generate batch argument with witness $(\{\rho_{i,j}\}_{j \in [n]}, x_i, \omega_i)$,

$$\Pi' \leftarrow \text{BARG}'.\text{P}(\text{crs}', C'_{[K,c,C]}, \{\{\rho_{i,j}\}_{j \in [n]}, x_i, \omega_i\}_{i \in [T]}).$$

- Output $\Pi = \Pi'$.

- $\text{V}(\text{crs}, C, x_1, x_2, \dots, x_T, \Pi)$: The verifier parses $\Pi = \Pi'$ and $\text{crs} = (K, \text{crs}')$. We construct the following pre-processing and online verification algorithms.

- **Pre-processing** $\text{PreVerify}(\text{crs}, x_1, x_2, \dots, x_T)$: Commit the instances by

$$\forall j \in [n], c_j := \text{SECOM}.\text{Com}(K, x_1[j], x_2[j], \dots, x_T[j]; 0).$$

Output the short sketch $c = \{c_j\}_{j \in [n]}$.

- **Online Verification** $\text{OnlineVerify}(\text{crs}, c, C, \Pi')$: Output $\text{BARG}'.\text{V}(\text{crs}', C'_{[K,c,C]}, \Pi')$.

The succinct communication property follows directly from the succinct communication property of the BARGs for the index language. The succinct verification property follows from the construction. The compact CRS property is also satisfied, since the underlying BARGs for index language satisfy compact CRS property, and $|K|$ is also bounded by $\text{poly}(\lambda, \log T)$ from the succinct CRS property of somewhere extractable commitment. The CRS indistinguishability property follows from the key indistinguishability of somewhere extractable commitment and the CRS indistinguishability of BARG'.

To prove semi-adaptive somewhere soundness, for any adversary \mathcal{A} for the semi-adaptive somewhere soundness of BARG, we construct a new adversary \mathcal{A}' for BARG', as follows: \mathcal{A}' first invokes \mathcal{A} , and obtain an index i^* which it outputs directly. Then, \mathcal{A}' receives a CRS crs' , generates a trapdoor commitment key K^* which is extractable at the i^* -th coordinate, and feeds $\text{crs} = (K = K^*, \text{crs}')$ to \mathcal{A} . Next, \mathcal{A}

⁶The commitment is computed with fixed randomness, without loss of generality we fix this to be 0.

outputs $(C, x_1, x_2, \dots, x_T, \Pi)$. \mathcal{A}' computes $c = \text{PreVerify}(\text{crs}, x_1, x_2, \dots, x_T)$, and outputs $(C'_{[K,c,C]}, \Pi)$. The adversary \mathcal{A}' simulates the environment for the adversary \mathcal{A} . Furthermore, if $(C, x_{i^*}) \notin \text{SAT}$, then by the extraction correctness of the commitment, we have $(C', i^*) \notin \text{Idx}$. Hence, if the attack of \mathcal{A} succeeds, then \mathcal{A}' also succeeds. Since the underlying BARG' is somewhere sound, we prove the semi-adaptive somewhere soundness of BARG . \square

5 RAM Delegation

We follow the notions of delegation, for both Turing Machines and RAM, as defined in [KPY19] who further show that their notion of RAM delegation implies Turing Machine delegation. This allows us to focus on constructing RAM delegation schemes for the rest of the paper.

5.1 Turing Machine Delegation

Consider a Turing machine \mathcal{M} . A publicly verifiable non-interactive delegation scheme for \mathcal{M} consists of the following polynomial time algorithms:

Del.S - randomized setup algorithm that on input security parameter 1^λ , time bound T and input length n outputs a pair of public keys - prover key pk and verifier key vk .

Del.P - deterministic prover algorithm that on input prover key pk and an input $x \in \{0, 1\}^n$ outputs a proof Π .

Del.V - deterministic verifier algorithm that on input verifier key pk , input $x \in \{0, 1\}^n$ and proof Π outputs either 0 or 1.

For any Turing machine \mathcal{M} , we define the corresponding language $\mathcal{U}_{\mathcal{M}}$ below,

$$\mathcal{U}_{\mathcal{M}} := \{(x, T) \mid \mathcal{M} \text{ accepts } x \text{ within } T \text{ steps}\}$$

Definition 8. A publicly verifiable non-interactive delegation scheme (Del.S, Del.P, Del.V) for \mathcal{M} with setup time $T_S = T_S(\lambda, T)$ and proof length $L_\Pi = L_\Pi(\lambda, T)$.

Completeness. For every $\lambda, T, n \in \mathbb{N}$ such that $n \leq T \leq 2^\lambda$, and $x \in \{0, 1\}^n$ such that $(x, T) \in \mathcal{U}_{\mathcal{M}}$,

$$\Pr \left[\text{Del.V}(\text{vk}, x, \Pi) = 1 \mid \begin{array}{l} (\text{pk}, \text{vk}) \leftarrow \text{Del.S}(1^\lambda, T, n) \\ \Pi := \text{Del.P}(\text{pk}, x) \end{array} \right] = 1$$

Efficiency. In the completeness experiment above,

- Del.S runs in time T_S .
- Del.P runs in time $\text{poly}(\lambda, T)$ and outputs a proof of length L_Π .
- Del.V runs in time $O(L_\Pi) + n \cdot \text{poly}(\lambda)$.

Soundness. For every PPT adversary \mathcal{A} and pair of polynomials $T = T(\lambda)$ and $n = n(\lambda)$ there exists a negligible function $\text{negl}(\text{cot})$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{Del.V}(\text{vk}, x, \Pi) = 1 \\ (x, T) \notin \mathcal{U}_{\mathcal{M}} \end{array} \mid \begin{array}{l} (\text{pk}, \text{vk}) \leftarrow \text{Del.S}(1^\lambda, T, n) \\ (x, \Pi) \leftarrow \mathcal{A}(\text{pk}, \text{vk}) \end{array} \right] \leq \text{negl}(\lambda)$$

5.2 RAM Delegation

A RAM machine of word size ℓ is modeled as a deterministic machine with random access to memory of size 2^ℓ where the local state of the machine has size only $O(\ell)$. At each time step, the machine updates its local state by either reading or writing a single memory. At any given time, the memory and the local state

together represent the configuration cf of the machine. For simplicity, we assume that the machine has no input outside of its local state and memory, and the word size ℓ will correspond to the security parameter λ .

A publicly verifiable non-interactive delegation scheme for \mathcal{R} consists of the following polynomial time algorithms:

RDel.S - randomized setup algorithm that on input security parameter 1^λ , time bound T outputs a triple of public keys - prover key pk , verifier key vk and a digest key dk .

RDel.D - deterministic digest algorithm that on input digest key dk and configuration cf outputs a digest h .

RDel.P - deterministic prover algorithm that on input prover key pk and a pair of source and destination configurations cf, cf' outputs a proof Π .

RDel.V - deterministic verifier algorithm that on verifier key pk , pair of digests h, h' and proof Π outputs either 0 or 1.

For any machine \mathcal{R} , we define the corresponding language $\mathcal{U}_{\mathcal{R}}$ below,

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

Definition 9. A publicly verifiable non-interactive delegation scheme (RDel.S, RDel.D, RDel.P, RDel.V) for \mathcal{R} with setup time $T_S = T_S(\lambda, T)$ and proof length $L_\Pi = L_\Pi(\lambda, T)$.

Completeness. For every $\lambda, T \in \mathbb{N}$ such that $n \leq T \leq 2^\lambda$, and $cf, cf' \in \{0, 1\}^*$ such that $(\lambda, cf, cf', T) \in \mathcal{U}_{\mathcal{R}}$,

$$\Pr \left[\text{RDel.V}(vk, h, h', \Pi) = 1 \mid \begin{array}{l} (pk, vk, dk) \leftarrow \text{RDel.S}(1^\lambda, T) \\ h := \text{RDel.D}(dk, cf) \\ h' := \text{RDel.D}(dk, cf') \\ \Pi := \text{RDel.P}(pk, cf, cf') \end{array} \right] = 1$$

Efficiency. In the completeness experiment above,

- RDel.S runs in time T_S .
- RDel.D on input cf runs in time $|cf| \cdot \text{poly}(\lambda)$ and outputs a digest of length λ .
- RDel.P runs in time $\text{poly}(\lambda, T, |cf|)$ and output a proof of length L_Π .
- RDel.V runs in time $O(L_\Pi) + \text{poly}(\lambda)$.

Collision resistance. For every PPT adversary \mathcal{A} and pair of polynomials $T = T(\lambda)$ there exists a negligible function $\text{negl}(\text{cot})$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} cf \neq cf' \\ \text{RDel.D}(dk, cf) = \text{RDel.D}(dk, cf') \end{array} \mid \begin{array}{l} (pk, vk, dk) \leftarrow \text{RDel.S}(1^\lambda, T, n) \\ (cf, cf') \leftarrow \mathcal{A}(pk, vk, dk) \end{array} \right] \leq \text{negl}(\lambda)$$

Soundness. For every PPT adversary \mathcal{A} and pair of polynomials $T = T(\lambda)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{RDel.V}(vk, h, h', \Pi) = 1 \\ (\lambda, cf, cf', T) \in \mathcal{U}_{\mathcal{R}} \\ h = \text{RDel.D}(dk, cf) \\ h' \neq \text{RDel.D}(dk, cf') \end{array} \mid \begin{array}{l} (pk, vk, dk) \leftarrow \text{RDel.S}(1^\lambda, T, n) \\ (cf, cf', h, h', \Pi) \leftarrow \mathcal{A}(pk, vk, dk) \end{array} \right] \leq \text{negl}(\lambda)$$

As discussed in [KPY19], the notion of RAM delegation considered in their work is different from those in prior works [KP16, BHK17] - namely that in prior works the adversary was not required to output the full configuration explicitly, only that it was difficult to produce accepting proofs for two different statements

(h, h') and (h, h'') that share the same initial digest. We refer the reader to [KPY19] for a more detailed comparison of the notions.

The following theorem establishes that RAM delegation implies Turing machine delegation for the definitions described above.

Theorem 9 ([KPY19]). *Suppose that for any RAM machine there exists a publicly verifiable non-interactive delegation scheme with setup time T'_S and proof length L'_Π . Then for any Turing machine there exists a publicly verifiable non-interactive delegation scheme with setup time T_S and proof length L_Π where $T_S(\lambda, T) = T'_S(\lambda, T')$, $L_\Pi(\lambda, T) = L'_\Pi(\lambda, T')$ for $T' = O(T)$.*

5.3 Hash Tree

For going beyond space bounded computation, we recall the definition of hash trees as defined in [KPY19]. A *hash tree* consists of the following algorithms:

- HT.Gen - randomized algorithm that on input the security parameter 1^λ outputs a hash key dk
- HT.Hash - deterministic algorithm that on input the hash key dk and string $D \in \{0, 1\}^L$ outputs a hash tree tree and a root rt.
- HT.Read - deterministic algorithm that on input hash tree tree and memory location ℓ outputs a bit b along with a proof Π .
- HT.Write - deterministic algorithm that on input hash tree tree, memory location ℓ and bit b outputs a new tree tree', a new root rt' along with a proof Π .
- HT.VerRead - deterministic algorithm on input hash key dk, root rt, memory location ℓ , bit b and proof Π outputs either 0 or 1.
- HT.VerWrite - deterministic algorithm on input hash key dk, root rt, memory location ℓ , bit b , new root rt' and proof Π outputs either 0 or 1.

Definition 10 (Hash Tree). *A hash tree scheme (HT.Gen, HT.Hash, HT.Read, HT.Write, HT.VerRead, HT.VerWrite) satisfies the following properties:*

Completeness. *For every $\lambda \in \mathbb{N}$, $D \in \{0, 1\}^L$ for $L \leq 2^\lambda$ and $\ell \in [L]$:*

$$\Pr \left[\begin{array}{l} \text{HT.VerRead}(\text{dk}, \text{rt}, \ell, b, \Pi) = 1 \\ D[\ell] = b \end{array} \middle| \begin{array}{l} \text{dk} \leftarrow \text{HT.Gen}(1^\lambda) \\ (\text{tree}, \text{rt}) := \text{HT.Hash}(\text{dk}, D) \\ (b, \Pi) := \text{HT.Read}(\text{tree}, \ell) \end{array} \right] = 1$$

Efficiency. *In the completeness experiment, the running time of HT.Hash is $|D| \cdot \text{poly}(\lambda)$. The length of the root rt, and proofs produced by HT.Read and HT.Write are $\text{poly}(\lambda)$.*

Soundness of Read. *For every polynomial size adversary \mathcal{A} there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\Pr \left[\begin{array}{l} b_1 \neq b_2 \\ \text{HT.VerRead}(\text{dk}, \text{rt}, \ell, b_1, \Pi_1) = 1 \\ \text{HT.VerRead}(\text{dk}, \text{rt}, \ell, b_2, \Pi_2) = 1 \end{array} \middle| \begin{array}{l} \text{dk} \leftarrow \text{HT.Gen}(1^\lambda) \\ (\text{rt}, \ell, b_1, \Pi_1, b_2, \Pi_2) \leftarrow \mathcal{A}(\text{dk}) \end{array} \right] \leq \text{negl}(\lambda)$$

Soundness of Write. *For every polynomial size adversary \mathcal{A} there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\Pr \left[\begin{array}{l} \text{rt}_1 \neq \text{rt}_2 \\ \text{HT.VerWrite}(\text{dk}, \text{rt}, \ell, b, \text{rt}_1, \Pi_1) = 1 \\ \text{HT.VerWrite}(\text{dk}, \text{rt}, \ell, b, \text{rt}_2, \Pi_2) = 1 \end{array} \middle| \begin{array}{l} \text{dk} \leftarrow \text{HT.Gen}(1^\lambda) \\ (\text{rt}, \ell, b, \text{rt}_1, \Pi_1, \text{rt}_2, \Pi_2) \leftarrow \mathcal{A}(\text{dk}) \end{array} \right] \leq \text{negl}(\lambda)$$

Theorem 10 ([Mer88]). *From any family of collision resistant hash functions, one can construct a hash tree scheme.*

5.4 Protocol

The protocol follows the construction of the base case for RAM delegation in [KPY19]. The crucial difference in our setting is that we are able to reduce the computation to an instance of a non-interactive BARG for index languages with a short CRS. The benefit of this is that we no longer have to do the bootstrapping since the CRS is already small. Our proof also closely follows the proof structure of their construction, although the no-signaling properties used in our proof are derived from the commitment rather than the underlying argument scheme.

RAM machine steps to circuit satisfiability. We use the translation from a *single* step of the machine \mathcal{R} as described in [KPY19]. Without loss of generality, assume that every step of \mathcal{R} consists of a *single read operation, followed by a single write operation*. Therefore, a single step can be decomposed into the following *deterministic* polynomial time algorithms:

StepR: On input the local state st of \mathcal{R} , outputs the memory location ℓ that \mathcal{R} while in state st would read from.

StepW: On input the local state st and bit b , outputs a bit b' , memory location ℓ' and state st' such that \mathcal{R} while in state st on reading bit b would write b' to location ℓ' and then transition to new local state st' .

We denote by φ the circuit representing a single step of \mathcal{R} , i.e. given a pair of digests $h = (st, rt)$, $h' = (st', rt')$, bit b and proof Π, Π' there exists an efficiently computable w (given (h, h', b, Π, Π')) such that $\varphi(h, h', b, \Pi, \Pi', w) = 1$ if and only if

$$\begin{aligned} \ell &= \text{StepR}(st) \\ (b', \ell', st'') &= \text{StepW}(st, b) \\ st' &= st'' \\ \text{HT.VerRead}(dk, rt, \ell, b, \Pi) &= 1 \\ \text{HT.VerWrite}(dk, rt, \ell', b', rt', \Pi') &= 1 \end{aligned}$$

From the efficiency of the hash tree scheme, there exists a φ such that the above can be represented as a formula of $L = \text{poly}(\lambda)$ variables.

We will use φ_i to denote the i -th step in the above formula ϕ . Note that the subscript will be helpful in our discussion of security, but the circuits themselves are identical for all i .

For T steps of \mathcal{R} , we then have the following formula ϕ over $M := O(L \cdot T)$ variables:

$$\phi \left(h_0, \{h_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i \in [T]} \right) := \bigwedge_{i \in [T]} \varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i)$$

Note that the above formula is *not an index language*. This is because for all i , φ_i and φ_{i+1} share a part of the witness, something not handled by the index language since we would have to ensure that the (partial) witness is the *same*. As described in the technical overview, we handle this by using a NS-SECOM to commit to the witnesses, and then prove for each i that values in the commitment satisfy the clause ϕ_i . The no-signaling property will help ensure consistency of the shared witness across different clauses φ_i and φ_{i+1} .

The components we require for our delegation scheme are listed below:

- An L -no-signaling-SECOM commitment scheme (Definition 6) $\text{NS-SECOM} = (\text{Gen}, \text{TGen}, \text{Com}, \text{Open}, \text{Verify}, \text{Ext})$.
- A non interactive batch argument (Section 4.1) for an index language $(\text{BARG.Gen}, \text{BARG.TGen}, \text{BARG.P}, \text{BARG.V})$ (Section 4.3).

We present our RAM delegation scheme in Figure 5.

Theorem 11. *Assuming the hardness of the Learning with Errors (LWE) (Definition 1), for every polynomial $T = T(\lambda)$, the protocol in Figure 5 is a publicly-verifiable non-interactive RAM delegation scheme (Definition 9) with CRS size, proof size and verifier time all $\text{poly}(\lambda, \log T)$ while the prover running time is $\text{poly}(\lambda, T)$.*

The assumptions required for our construction follow from the assumptions of the underlying primitives. We focus on proving the efficiency and security of our protocol below.

5.4.1 Efficiency

Before proving security, we prove that the language above is indeed an index language, and that the efficiency criteria for a RAM delegation scheme are satisfied.

Claim 1. *For all $i \in [T]$, $(C_{\text{index}}, i) \in \mathcal{L}$.*

Proof. This just follows from the construction in Figure 6. The witness corresponds to the inputs as in the delegation protocol. \square

Claim 2. $|C_{\text{index}}| = \text{poly}(\lambda, \log T)$

Proof. $|C_{\text{index}}|$ consists of the following:

- The hardcoded commitment key K : $|K| = L \cdot \text{poly}(\lambda, \log M) = \text{poly}(\lambda, \log T)$.
- The hardcoded commitment c : $|c| = L \cdot \text{poly}(\lambda, \log M) = \text{poly}(\lambda, \log T)$ (same as key size).
- The hardcoded circuit φ : $|\varphi_i| = O(L) = \text{poly}(\lambda)$.
- Size of openings as a part of the witness of size $L \cdot \text{poly}(\lambda, \log M) = \text{poly}(\lambda, \log T)$.
- Verifier circuit for commitment of size $\text{poly}(\lambda, \log M) = \text{poly}(\lambda, \log T)$.

\square

CRS size. The CRS consists of the commitment key K , the BARG CRS crs and the digest key dk . By the corresponding properties of the underlying scheme we have: $|K| + |\text{crs}| + |\text{dk}| = \text{poly}(\lambda, \log T) + \text{poly}(\lambda, \log T, |C_{\text{index}}|) = \text{poly}(\lambda, \log T)$.

Proof length. The proof consists of the commitment c and the BARG proof: $|c| + |\Pi| = \text{poly}(\lambda, \log T) + \text{poly}(\lambda, \log T, |C_{\text{index}}|) = \text{poly}(\lambda, \log T)$

Verifier time: The verification time is the time taken to compute the circuit $|C_{\text{index}}|$ and verify the BARG proof: $\text{poly}(\lambda, \log T, |C_{\text{index}}|) = \text{poly}(\lambda, \log T)$.

⁷We overload notation here to specify opening to many bits.

⁸Since these are not a single bit, we overload notation and skip the index i that is passed to the Verify algorithm.

RAM Delegation

RDel.S($1^\lambda, T$): Generate the public parameters for the underlying primitives

$$K \leftarrow \text{Gen}(1^\lambda, 1^M), \text{crs} \leftarrow \text{BARG.Gen}(1^\lambda, 1^T, 1^{|\mathcal{C}_{\text{index}}|}), \text{dk} \leftarrow \text{HT.Gen}(1^\lambda).$$

Output $(\text{pk} := (K, \text{crs}, \text{dk}), \text{vk} := (K, \text{crs}), \text{dk})$.

RDel.D($\text{dk}, \text{cf} = (\text{st}, D)$): Compute the hash tree,

$$(\text{tree}, \text{rt}) := \text{HT.Hash}(\text{dk}, D)$$

Output $h := (\text{st}, \text{rt})$.

RDel.P($(\text{pk}, \text{dk}), \text{cf}, \text{cf}'$): Prover emulates \mathcal{R} for T steps from cf to cf' to obtain the satisfying assignment for ϕ as follows: define

$$(\text{st}_0, D_0) := \text{cf}, \quad (\text{tree}_0, \text{rt}_0) := \text{HT.Hash}(\text{dk}, D_0), \quad h_0 := (\text{st}_0, \text{rt}_0)$$

Then for every $i \in [T]$

$$\begin{aligned} \ell_i &= \text{StepR}(\text{st}_{i-1}), \quad (b_i, \Pi_i) := \text{HT.Read}(\text{tree}_{i-1}, \ell_i) \\ (b'_i, \ell'_i, \text{st}_i) &:= \text{StepW}(\text{st}_{i-1}, b_i), \quad (\text{tree}_i, \text{rt}_i, \Pi'_i) := \text{HT.Read}(\text{tree}_{i-1}, \ell'_i, b'_i) \\ h_i &:= (\text{st}_i, \text{rt}_i) \end{aligned}$$

and then compute (efficiently) w_i be such that $\varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$.

Compute the no-signaling commitment to $(h_0, \{h_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i \in [T]})$

$$c \leftarrow \text{Com}\left(K, \left(h_0, \{h_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i \in [T]}\right); R\right)$$

For every $i \in [T]$, compute the local opening to the commitment:

$$\text{For } A \in \{h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i\},$$

$$\rho_A := \text{Open}(K, A, R)^7$$

Compute the circuit C_{index} as described in Figure 6, and then compute the proof of the underlying BARG

$$\Pi := \text{BARG.P}\left(\text{crs}, C_{\text{index}}, \{h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i, \rho_{h_{i-1}}, \rho_{h_i}, \rho_{b_i}, \rho_{\Pi_i}, \rho_{\Pi'_i}, \rho_{w_i}\}_{i \in [T]}\right)$$

Output (c, Π) .

RDel.V(vk, h, h', Π): Given c and K , compute C_{index} (as in Figure 6) and output 1 if and only if

$$\text{BARG.V}(\text{crs}, C_{\text{index}}, \Pi) = 1$$

Figure 5: RAM delegation scheme

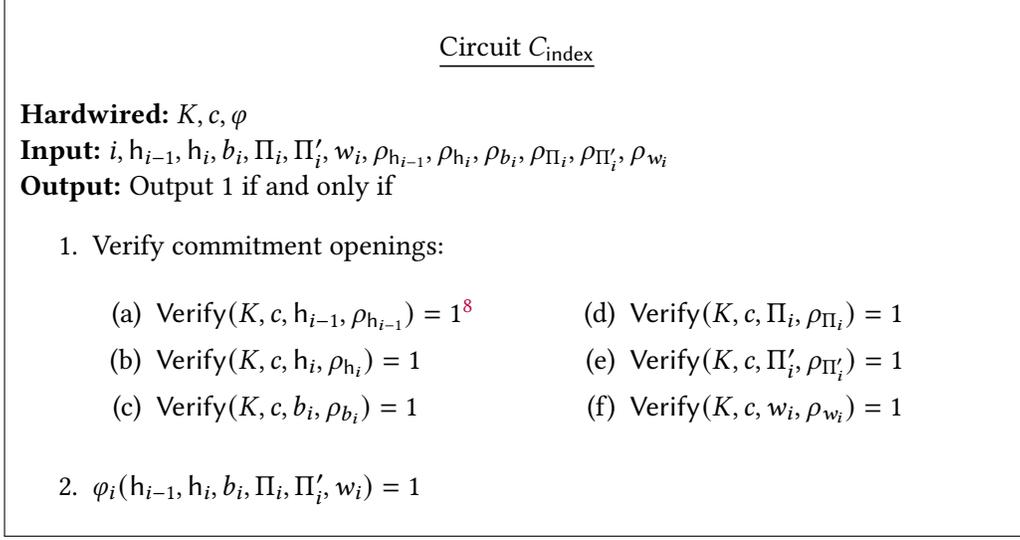


Figure 6: Circuit

5.4.2 Security Proof

Let us assume for the sake of contradiction that the soundness of the above scheme does not hold. Then fix \mathcal{A} and T such that there exists a polynomial $p(\cdot)$ where, for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{RDel.V}(\text{vk}, h, h', \Pi) = 1 \\ (\lambda, \text{cf}, \text{cf}', T) \in \mathcal{U}_{\mathcal{R}} \\ h = \text{RDel.D}(\text{dk}, \text{cf}) \\ h' \neq \text{RDel.D}(\text{dk}, \text{cf}') \end{array} \middle| \begin{array}{l} (\text{pk}, \text{vk}, \text{dk}) \leftarrow \text{RDel.S}(1^\lambda, T, n) \\ (\text{cf}, \text{cf}', h, h', \Pi) \leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}) \end{array} \right] \geq \frac{1}{p(\lambda)}. \quad (2)$$

We now use an averaging argument to fix a *bad* digest key dk . Specifically, a digest key dk is *bad* if Equation (2) holds with probability at least $1/2p(\lambda)$ when dk is sampled by RDel.S . By an averaging argument the fraction of such *bad* digest keys must be at least $1/2p(\lambda)$. Therefore, conditioned on a fixed *bad* digest key dk^* we have,

$$\Pr \left[\begin{array}{l} \text{RDel.V}(\text{vk}, h, h', \Pi) = 1 \\ (\lambda, \text{cf}, \text{cf}', T) \in \mathcal{U}_{\mathcal{R}} \\ h = \text{RDel.D}(\text{dk}^*, \text{cf}) \\ h' \neq \text{RDel.D}(\text{dk}^*, \text{cf}') \end{array} \middle| \begin{array}{l} (\text{pk}, \text{vk}, \text{dk}) \leftarrow \text{RDel.S}(1^\lambda, T, n)|_{\text{dk}=\text{dk}^*} \\ (\text{cf}, \text{cf}', h, h', \Pi) \leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}^*) \end{array} \right] \geq \frac{1}{p(\lambda)}. \quad (3)$$

where $(\text{pk}, \text{vk}, \text{dk}) \leftarrow \text{RDel.S}(1^\lambda, T, n)|_{\text{dk}=\text{dk}^*}$ is the setup algorithm RDel.S conditioned on the digest key output being dk^* . Moving forward, for ease of notation, the fixed bad dk^* will be denoted simply by dk .

Next, we change the setup algorithm in both the underlying commitment scheme (NS-SECOM) and the batch argument (BARG) to be the trapdoor setup algorithm with the \emptyset as the argument. We want to argue that the distribution of \mathcal{A} 's output $(\text{cf}, \text{cf}', h, h')$ does not change by more than a negligible probability. Specifically,

Claim 3. For all PPT distinguisher \mathcal{D} ,

$$\left| \Pr \left[\mathcal{D}(\text{cf}, \text{cf}', h, h') = 1 \mid \begin{array}{l} (\text{pk}, \text{vk}, \text{dk}) \leftarrow \text{RDel.S}(1^\lambda, T, n)|_{\text{dk}} \\ (\text{cf}, \text{cf}', h, h', \Pi) \leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}) \end{array} \right] \right. \\ \left. - \Pr \left[\mathcal{D}(\text{cf}, \text{cf}', h, h') = 1 \mid \begin{array}{l} K \leftarrow \text{TGen}(1^\lambda, 1^M, \emptyset) \\ \text{crs} \leftarrow \text{BARG.TGen}(1^\lambda, 1^T, 1^{|\text{C}_{\text{index}}|}, \emptyset) \\ \text{pk} := (K, \text{crs}, \text{dk}), \text{vk} := (K, \text{crs}) \\ (\text{cf}, \text{cf}', h, h', \Pi) \leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}) \end{array} \right] \right| \leq \text{negl}(\lambda). \quad (4)$$

where in each experiment above $(\text{cf}, \text{cf}', h, h') = \perp$ if $\text{RDel.V}(\text{vk}, h, h', \Pi) = 0$.

Proof. Consider Hyb_0 to be the experiment

$$\begin{aligned} (\text{pk}, \text{vk}, \text{dk}) &\leftarrow \text{RDel.S}(1^\lambda, T, n)|_{\text{dk}} \\ (\text{cf}, \text{cf}', h, h', \Pi) &\leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}) \end{aligned}$$

with output $(\text{cf}, \text{cf}', h, h')$. Similarly, Hyb_1 is the experiment

$$\begin{aligned} K &\leftarrow \text{TGen}(1^\lambda, 1^M, \emptyset) \\ \text{crs} &\leftarrow \text{BARG.TGen}(1^\lambda, 1^T, 1^{|\text{C}_{\text{index}}|}, \emptyset) \\ \text{pk} &:= (K, \text{crs}, \text{dk}), \text{vk} := (K, \text{crs}) \\ (\text{cf}, \text{cf}', h, h', \Pi) &\leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}) \end{aligned}$$

with output $(\text{cf}, \text{cf}', h, h')$. It then suffices to show that $\text{Hyb}_0 \approx \text{Hyb}_1$. For this we introduce an intermediate hybrid Hyb' that has the same output, but the experiment is:

$$\begin{aligned} K &\leftarrow \text{Gen}(1^\lambda, 1^M) \\ \text{crs} &\leftarrow \text{BARG.TGen}(1^\lambda, 1^T, 1^{|\text{C}_{\text{index}}|}, \emptyset) \\ \text{pk} &:= (K, \text{crs}, \text{dk}), \text{vk} := (K, \text{crs}) \\ (\text{cf}, \text{cf}', h, h', \Pi) &\leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}). \end{aligned}$$

Note that for all of the above experiments, set $(\text{cf}, \text{cf}', h, h', \Pi)$ to \perp if Π is not accepting.

We now prove the following:

$\text{Hyb}_0 \approx \text{Hyb}'$: We rely on the *key-indistinguishability* property of the BARG scheme. Specifically, if there exists a PPT distinguisher \mathcal{D} that distinguishes Hyb_0 and Hyb' , we can construct an adversary \mathcal{B} (with a bad dk as an additional argument) as below,

$\mathcal{B}(1^\lambda, \text{dk})$:

1. Send \emptyset to the BARG key indistinguishability.
2. On obtaining crs from the challenger, run the rest of the experiment.
3. Check if Π is accepting. If not, set $(\text{cf}, \text{cf}', h, h', \Pi)$ to \perp .
4. Output $\mathcal{D}(\text{cf}, \text{cf}', h, h')$.

Depending on the response received from the challenger, the experiment corresponds either to Hyb_0 or Hyb' , and \mathcal{B} succeeds if \mathcal{D} succeeds.

$\text{Hyb}' \approx \text{Hyb}_1$: We rely on the *key-indistinguishability* property of the NS-SECOM scheme. The proof follows identically as in the above case.

This completes the proof. \square

If $(\text{cf}, \text{cf}', \overline{h}, \overline{h}') \neq \perp$, i.e. the proof is accepting, then starting with configuration cf we define the “true” configuration $\overline{\text{cf}}_i$ after i -steps of computation. Formally, for each $i \in [0, T]$, $\overline{\text{cf}}_i$ is the *unique configuration* such that $(\lambda, \text{cf}, \overline{\text{cf}}_i, i) \in \mathcal{U}_{\mathcal{R}}$. The corresponding digest \overline{h}_i is defined to be $\text{RDel.D}(\text{dk}, \overline{\text{cf}}_i)$. We will refer to such configurations (resp. digests) to be the “true” configuration (resp. digest). We define below the event and experiment that will be relevant to the analysis.

CHEAT: The event that $\text{BARG.Vf}(\text{crs}, C_{\text{index}}, \pi) = 1$, and $\overline{h} = \overline{h}_0$ but $\overline{h}' \neq \overline{h}_T$.

EXP_i: Let S_i denote the set of ϕ 's variables that represent the variables for the i -th clause - $h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i$. **EXP_i** then corresponds to the experiment where the keys for the commitment scheme and the batch argument are generated with trapdoor for S_i and i respectively. More formally

$$K \leftarrow \text{TGen}(1^\lambda, 1^M, S_i), \text{crs} \leftarrow \text{BARG.TGen}(1^\lambda, 1^T, 1^{|\text{C}_{\text{index}}|}, i)$$

If the proof π is non-accepting, the output of the experiment is \perp . Otherwise the output is $(\text{cf}, \text{cf}', \overline{h}, \overline{h}')$ along with the extracted value

We claim that in the above experiment, the adversary still cheats with an inverse polynomial probability.

Claim 4. For all $i \in [T]$,

$$\Pr_{\text{EXP}_i}[\text{CHEAT}] \geq \frac{1}{\text{poly}(\lambda)} \quad (5)$$

Proof. This follows in an identical manner to Claim 3 from the key indistinguishability of the underlying schemes, and the fact that the event CHEAT is efficiently checkable. \square

Claim 5. For all $i \in [T]$,

$$\Pr_{\text{EXP}_i}[\text{CHEAT} \implies \varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i) = 1] \geq 1 - \text{negl}(\lambda) \quad (6)$$

where $h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i$ is extracted from the commitment using the trapdoor.

Proof. Given that CHEAT holds, we know that the proof Π output by \mathcal{A} is accepting. We first use the semi-adaptive somewhere soundness of the BARG to show that the $(C_{\text{index}}, i) \in \mathcal{L}$. If not, we construct an adversary \mathcal{B} as below

$\mathcal{B}(1^\lambda, \text{dk})$:

1. Send i to the BARG challenger and receive crs .
2. Compute the rest of the experiment EXP_i using crs .
3. Return (C_{index}, Π) .

Thus, we have $(C_{\text{index}}, i) \in \mathcal{L}$ which means that there is an accepting witness to C_{index} in Figure 6. Now, from the somewhere statistical binding property of the NS-SECOM for S_i , we have that other than with negligible probability the extracted values $(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i)$ (along with their opening proofs) are the only valid local openings to c on S_i . Therefore, since $(C_{\text{index}}, i) \in \mathcal{L}$ and $(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i)$ is the only valid opening to c we have that other than with negligible probability, C_{index} for index i has a *unique* (partial) witness $(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i)$.

This in turn implies (by the construction of C_{index}), we have $\varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$. \square

Therefore we have

$$\Pr_{\text{EXP}_i} \left[\begin{array}{l} \text{CHEAT} \wedge \\ \varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i) = 0 \end{array} \right] < \text{negl}(\lambda) \quad (7)$$

In EXP_1 , φ_1 is consistent with h , therefore $h = h_0$ which gives us,

$$\Pr_{\text{EXP}_1} \left[\begin{array}{l} \text{CHEAT} \wedge \\ h_0 \neq \bar{h}_0 \end{array} \right] < \text{negl}(\lambda) \quad (8)$$

In EXP_T , φ_T is consistent with h' , therefore $h' = h_T$ which, because the adversary is cheating, gives us

$$\Pr_{\text{EXP}_T} \left[\begin{array}{l} \text{CHEAT} \wedge \\ h_T = \bar{h}_T \end{array} \right] < \text{negl}(\lambda) \quad (9)$$

We now want to use the no-signaling property to claim that in both EXP_i and EXP_{i+1} it must be the case that the extracted h_i is the corresponding “true” digest at the i -step of the execution.

Claim 6. For all $i \in [T - 1]$

$$\left| \Pr_{\text{EXP}_i} \left[\begin{array}{l} \text{CHEAT} \wedge \\ h_i = \bar{h}_i \end{array} \right] - \Pr_{\text{EXP}_{i+1}} \left[\begin{array}{l} \text{CHEAT} \wedge \\ h_i = \bar{h}_i \end{array} \right] \right| < \text{negl}(\lambda) \quad (10)$$

Proof. We will prove this using a sequence of hybrids. Let Hyb_0 be the distribution in EXP_i with output (cf, cf', h, h', h_i) where h_i is extracted from the NS-SECOM commitment scheme using the trapdoor for S_i . Similarly, let Hyb_1 be the distribution in EXP_i with output (cf, cf', h, h', h_i) where h_i is extracted from the NS-SECOM commitment scheme using the trapdoor for S_{i+1} .

It suffices to show these two distributions are indistinguishable since given (cf, cf', h, h', h_i) one can compute \bar{h}_i and check if $\bar{h}_i = h_i$. If the probability of the check succeeded differed in the Hyb_0 and Hyb_1 by a non-negligible amount, we would have an efficient distinguisher.

We introduce an intermediate distribution Hyb' where the experiment is:

$$\begin{aligned} K &\leftarrow \text{TGen}(1^\lambda, 1^M, S_{i+1}) \\ \text{crs} &\leftarrow \text{BARG.TGen}(1^\lambda, 1^T, 1^{|\text{C}_{\text{index}}|}, i) \\ \text{pk} &:= (K, \text{crs}, \text{dk}), \text{vk} := (K, \text{crs}) \\ (cf, cf', h, h', \Pi) &\leftarrow \mathcal{A}(\text{pk}, \text{vk}, \text{dk}). \end{aligned}$$

with the output being (cf, cf', h, h', h_i) .

$\text{Hyb}_0 \approx \text{Hyb}'$: We rely on the *no-signaling* property of the commitment scheme NS-SECOM. Specifically, if there exists a PPT distinguisher \mathcal{D} that distinguishes Hyb_0 and Hyb' , we can construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ (with a bad dk as non-uniform advice) as below,

$$\underline{\mathcal{B}_1(1^\lambda, \text{dk})}$$

1. On input K , run the rest of the experiment with K as the NS-SECOM key.
2. Separate c from \mathcal{A} 's proof.
3. Set $z := (cf, cf', h, h')$ as the auxiliary input, setting z to \perp if Π is not accepting.
4. Output (c, z) .

$\mathcal{B}_2(1^\lambda, dk)$:

1. On input K, c, z, h_i , output $\mathcal{D}(cf, cf', h, h', h_i)$.

Depending on the response received from the challenger, the experiment corresponds either to Hyb_0 or Hyb' , and \mathcal{B} succeeds if \mathcal{D} succeeds.

$\text{Hyb}' \approx \text{Hyb}_1$: We rely on the *key-indistinguishability* property of the BARG scheme. Specifically, if there exists a PPT distinguisher \mathcal{D} that distinguishes Hyb' and Hyb_0 we construct an adversary \mathcal{B} (with a bad dk as non-uniform advice) as below,

$\mathcal{B}(1^\lambda, dk)$:

1. Send $i, i + 1$ to the BARG key indistinguishability challenger.
2. On receiving crs run the rest of the experiment using received crs .
3. Use the NS-SECOM to extract h_i .
4. Output $\mathcal{D}(cf, cf', h, h', h_i)$.

Note that we have directly reduced to the key indistinguishability of the BARG between keys generated for i and $i + 1$, while our security definition only states that it is computationally intractable to differentiate between the key generated in the normal mode, and the key generated for an index i . It is easy to see that our definition also implies key indistinguishability on any two inputs i and j (with the normal mode as the intermediate hybrid experiment).

This completes the proof. □

From the above equations, given that $h_T \neq \bar{h}_T$ (other than with negligible probability), it must be the case that there exists an i^* such that the input digest is “true”, while the output digest is not “true”. Formally, from equations (5) to (10), we have that there exists $i^* \in [T]$ such that

$$\Pr_{\text{EXP}_{i^*}} \left[\begin{array}{l} \text{CHEAT} \wedge \\ h_{i^*-1} = \bar{h}_{i^*-1} \wedge \\ h_{i^*} \neq \bar{h}_{i^*} \wedge \\ \varphi_i(h_{i^*-1}, h_{i^*}, b_{i^*}, \Pi_{i^*}, \Pi'_{i^*}, w_{i^*}) = 1 \end{array} \right] \geq \frac{1}{T \cdot \text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)} \quad (11)$$

Let $i := i^*$ as described above. Then for $(st_{i-1}, rt_{i-1}) := h_{i-1}$,

$$\begin{aligned} \ell_i &:= \text{StepR}(st_{i-1}) \\ (b'_i, \ell'_i, st_i) &:= \text{StepW}(st_{i-1}, b_i) \end{aligned}$$

Next, compute the corresponding “true” variables starting from $(\overline{st}_{i-1}, \overline{D}_{i-1}) := \overline{cf}_{i-1}$:

$$\begin{aligned} (\overline{tree}_i, \overline{rt}_i) &:= \text{HT.Hash}(\text{dk}, \overline{D}_{i-1}) \\ \overline{\ell}_i &:= \text{StepR}(\overline{st}_{i-1}) \\ (\overline{b}_i, \overline{\Pi}_i) &:= \text{HT.Read}(\overline{tree}_{i-1}, \overline{\ell}_i) \\ (\overline{b}'_i, \overline{\ell}'_i, \overline{st}_i) &:= \text{StepW}(\overline{st}_{i-1}, \overline{b}_i) \\ (\overline{tree}_i, \overline{rt}_i, \overline{\Pi}'_i) &:= \text{HT.Read}(\overline{tree}_{i-1}, \overline{\ell}'_i, \overline{b}'_i) \end{aligned}$$

We describe two events below.

CHEAT₁: the event that the following is true

$$\begin{aligned} b_i &\neq \overline{b}_i \\ \text{HT.VerRead}(\text{dk}, \text{rt}_{i-1}, \ell_i, b_i, \Pi_i) &= 1 \\ \text{HT.VerRead}(\text{dk}, \text{rt}_{i-1}, \ell_i, \overline{b}_i, \overline{\Pi}_i) &= 1 \end{aligned}$$

CHEAT₂ the event that the following is true

$$\begin{aligned} \text{rt}_i &\neq \overline{\text{rt}}_i \\ \text{HT.VerWrite}(\text{dk}, \text{rt}_{i-1}, \ell'_i, b'_i, \text{rt}'_i, \Pi'_i) &= 1 \\ \text{HT.VerWrite}(\text{dk}, \text{rt}_{i-1}, \ell'_i, b'_i, \overline{\text{rt}}'_i, \overline{\Pi}'_i) &= 1 \end{aligned}$$

As observed in [KPY19], the following claim establishes that by the completeness of the hash tree scheme, either the read or the write operation. We have reproduced the proof here for completeness.

Claim 7.

$$\Pr_{\text{EXP}_i} \left[\begin{array}{l} \text{CHEAT} \wedge \\ h_{i-1} = \overline{h}_{i-1} \wedge \\ h_i \neq \overline{h}_i \wedge \\ \varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i) = 1 \end{array} \right] \implies \text{CHEAT}_1 \vee \text{CHEAT}_2 \quad (12)$$

Proof. This follows from the completeness of the hash tree scheme. Specifically, by definition, we have that $h_{i-1} = (\overline{st}_{i-1}, \overline{rt}_{i-1})$ and $h_i = (\overline{st}_i, \overline{rt}_i)$. By the completeness of the hash tree scheme we have,

$$\begin{aligned} \text{HT.VerRead}(\text{dk}, \overline{\text{rt}}_{i-1}, \overline{\ell}_i, \overline{b}_i, \overline{\Pi}_i) &= 1 \\ \text{HT.VerWrite}(\text{dk}, \overline{\text{rt}}_{i-1}, \overline{\ell}'_i, \overline{b}'_i, \overline{\text{rt}}'_i, \overline{\Pi}'_i) &= 1 \end{aligned}$$

Now from the claim, we have that $h_{i-1} = \overline{h}_{i-1}$, which means that $(\text{st}_{i-1}, \text{rt}_{i-1}) = (\overline{\text{st}}_{i-1}, \overline{\text{rt}}_{i-1})$. Since StepR is a deterministic algorithm, we also have $\ell_i = \overline{\ell}_i$. Finally, since $\varphi_i(h_{i-1}, h_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$ we have,

$$\begin{aligned} h_i &= (\text{st}_i, \text{rt}_i) \\ \text{HT.VerRead}(\text{dk}, \text{rt}_{i-1}, \ell_i, b_i, \Pi_i) &= 1 \\ \text{HT.VerWrite}(\text{dk}, \text{rt}_{i-1}, \ell'_i, b'_i, \text{rt}'_i, \Pi'_i) &= 1 \end{aligned}$$

We consider two cases below to complete the proof. Note that in both the cases, the required proofs from the events verify, so we omit them in the discussion below.

Case $b_i \neq \bar{b}_i$: CHEAT₁ holds.

Case $b_i = \bar{b}_i$: Then, because StepW is deterministic, we have that $(\ell'_i, b'_i, st_i) = (\bar{\ell}'_i, \bar{b}'_i, \bar{st}_i)$. Combining this with the fact that $h_i \neq \bar{h}_i$, it must be the case that $rt_i \neq \bar{rt}_i$. This ensures that CHEAT₂ holds in this case.

□

From the above claims, since CHEAT₁ \vee CHEAT₂ happens with probability $1/\text{poly}(\lambda)$. We can now construct an adversary \mathcal{B} for the hash tree scheme that breaks either the soundness of read (when CHEAT₁ holds) or the soundness of write (when CHEAT₂ holds). Specifically,

$\mathcal{B}(1^\lambda)$:

1. Receive dk from the HT challenger.
2. Runs EXP _{i} (from the above claim either CHEAT₁ or CHEAT₂ hold).
3. If CHEAT₁ holds, output $(rt_{i-1}, \ell_i, b_i, \Pi_i, \bar{b}_i, \bar{\Pi}_i)$.
4. Else, if CHEAT₂ holds, output $(dk, rt_{i-1}, \ell'_i, b'_i, rt'_i, \bar{rt}'_i, \bar{\Pi}'_i)$

Correctness follows from the description of the events. Since we have that the received dk is *bad* with probability $1/2p(\lambda)$, \mathcal{B} successfully breaks the soundness of the hash tree scheme. This completes the security proof.

6 More Efficient Batch Arguments for NP

In this section, we show how to use batch arguments for NP and the Turing machine delegation scheme to build more efficient batch arguments for NP by removing the dependence on the circuit size. The idea is to delegate the verification of an NP instance to the Turing machine, and then use the batch argument to prove that the verifier in the delegation scheme will accept the delegation proof for all instances.

In more detail, we use batch arguments for SAT and a Turing machine delegation scheme to construct BARGs for the following NP language

$$\mathcal{L} = \{x \mid \exists \omega : \mathcal{M}(x, \omega) \text{ outputs } 1 \text{ (accepts) in } T \text{ steps}\},$$

where \mathcal{M} is a Turing machine. The proof size and CRS size are polynomial in λ , $|w|$, and $\log T$.

Theorem 12. *Let $\text{Del} = (\text{Del.S}, \text{Del.P}, \text{Del.V})$ be a Turing machine delegation scheme, and $\text{BARG}' = (\text{BARG}'.\text{Gen}, \text{BARG}'.\text{TGen}, \text{BARG}'.\text{P}, \text{BARG}'.\text{V})$ be a batch argument for SAT with efficient online verification property. Then we can construct a batch argument $\text{BARG} = (\text{Gen}, \text{TGen}, \text{P}, \text{V})$ with efficient online verification property for \mathcal{L} with the following efficiency. Let $n = |x|$ be the length of the instances, $m = |\omega|$ be the length of the witnesses, and k be the number of instances.*

- **CRS size:** *The size of the CRS is $\text{poly}(\lambda, m, \log k, \log T)$.*
- **Proof size:** *The size of the argument is $\text{poly}(\lambda, m, \log k, \log T)$.*
- **Efficient Online Verification:** *The running time of the offline preprocessing is $\text{poly}(\lambda, k, n)$, and the online verification algorithm runs in $\text{poly}(\lambda, m, \log k, \log T)$.*

Proof. We build the BARG as follows. We will use a circuit DelVerify in Figure 7.

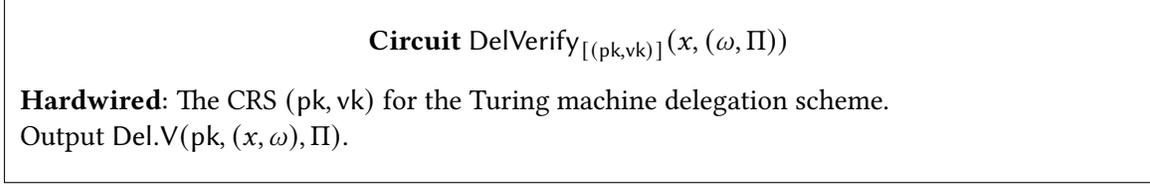


Figure 7: The new circuit C' for batch argument.

- $\text{Gen}(1^\lambda, 1^k, 1^T)$: Generate a CRS for the Turing machine delegation scheme, and a CRS for the batch argument for C-SAT.
 - Let $(pk, vk) \leftarrow \text{Del.S}(1^\lambda, T)$, and $\text{crs}' \leftarrow \text{BARG}'.\text{Gen}(1^\lambda, 1^k, 1^{|\text{DelVerify}|})$.
 - Output $\text{crs} = ((pk, vk), \text{crs}')$.
- $\text{TGen}(1^\lambda, 1^k, 1^T, i^*)$: The trapdoor CRS generation algorithm generates (pk, vk) in the same way as Gen , and generates a trapdoor CRS for BARG' .
 - Let $(pk, vk) \leftarrow \text{Del.S}(1^\lambda, T)$, and $\text{crs}^{*'} \leftarrow \text{BARG}'.\text{Gen}(1^\lambda, 1^k, 1^{|\text{DelVerify}|}, i^*)$.
 - Output $\text{crs}^* = ((pk, vk), \text{crs}^{*'})$.
- $\text{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k)$: The prover first delegates each instance using Turing machine delegation, and then uses batch arguments to prove all the delegated instances verify.

Parse $\text{crs} = ((pk, vk), \text{crs}')$, where (pk, vk) is the CRS for Turing machine delegation, and crs' is the CRS for the batch argument.

 - Delegate the computation of \mathcal{M} using Turing machine delegation.
$$\forall i \in [k], \Pi_i \leftarrow \text{Del.P}(pk, x_i, \omega_i).$$
 - Use batch arguments to generate the poof.
$$\Pi' \leftarrow \text{BARG}'.\text{P}(\text{crs}', \text{DelVerify}_{[(pk, vk)]}, x_1, \dots, x_k, \{(\omega_i, \Pi_i)\}_{i \in [k]}).$$
 - Output $\Pi = \Pi'$.
- $\text{V}(\text{crs}, x_1, \dots, x_k, \Pi)$: The verifier parses $\Pi = \Pi'$, and verifies
$$\text{BARG}'.\text{V}(\text{crs}', \text{DelVerify}, x_1, \dots, x_k, \Pi') = 1.$$

The CRS size is $|pk, vk| + |\text{crs}'|$. Since the delegation scheme has CRS size $\text{poly}(\lambda, \log T)$, and crs' is also bounded by $\text{poly}(\lambda, \log k, |\text{DelVerify}|) = \text{poly}(\lambda, \log k, \text{poly}(\lambda, \log k, \log T, m))$, the CRS size is bounded by $\text{poly}(\lambda, \log T, m)$. The proof size is the same as the proof size of BARG' . Hence, the proof size is also bounded by $\text{poly}(\lambda, m, \log k, \log T)$. Finally, since the verification algorithm is the same as the verification of BARG' . The efficient online verification property follows from the same property of the underlying BARG' .

The completeness follows from the completeness of the delegation scheme Del and the batch argument BARG' . The CRS indistinguishability follows the same property of BARG' .

To prove the semi-adaptive somewhere soundness, we require the somewhere argument of knowledge property of the underlying batch argument BARG' , which allows us extraction a witness for the i^* -th instance from an accepting proof (See Remark 2).

Given somewhere argument of knowledge property, we can extract the witness $(\omega_{i^*}, \Pi'_{i^*})$ for the i^* -th instance from any cheating prover for BARG, and output Π'_{i^*} as the attacking proof for the underlying delegation scheme Del, with input (x_{i^*}, ω_{i^*}) . Since the delegation scheme is sound, this proves the soundness of our batch argument construction. \square

Acknowledgments

Arka Rai Choudhuri, Abhishek Jain and Zhengzhong Jin are supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. Arka Rai Choudhuri and Abhishek Jain are also supported in part by the Office of Naval Research Grant N00014-19-1-2294. Arka Rai Choudhuri is also supported in part by NSF Grant CNS-1414023. Zhengzhong Jin is also supported in part by NSF CAREER 1845349.

References

- [ACC⁺16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating RAM computations with adaptive soundness and privacy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 3–30. Springer, Heidelberg, October / November 2016. 4
- [Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001. 7
- [BBH⁺19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of kilian-based SNARGs. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 522–551. Springer, Heidelberg, December 2019. 7
- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988. 3
- [BCC⁺17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinfeld, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017. 4
- [BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013. 4
- [BCG⁺14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014. 3
- [BCI⁺13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013. 4
- [BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *23rd ACM STOC*, pages 21–31. ACM Press, May 1991. 15, 24

- [BGL⁺15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015. [4](#)
- [BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 474–482. ACM Press, June 2017. [4](#), [7](#), [10](#), [11](#), [35](#)
- [BK20] Zvika Brakerski and Yael Kalai. Witness indistinguishability for any single-round argument with applications to access control. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 97–123. Springer, Heidelberg, May 2020. [4](#), [10](#)
- [BKK⁺18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 709–721. ACM Press, June 2018. [4](#), [10](#)
- [BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 738–767. Springer, Heidelberg, August 2020. [6](#)
- [BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 671–684. ACM Press, June 2018. [5](#)
- [CCC⁺16] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel RAM from indistinguishability obfuscation. In Madhu Sudan, editor, *ITCS 2016*, pages 179–190. ACM, January 2016. [4](#)
- [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 Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. [4](#), [5](#), [6](#), [7](#), [16](#)
- [CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018. [6](#)
- [CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004. [18](#)
- [CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016*, pages 169–178. ACM, January 2016. [4](#)
- [CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 429–437. ACM Press, June 2015. [4](#)

- [CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for np from standard assumptions. In *CRYPTO*, Lecture Notes in Computer Science. Springer, 2021. [3](#), [4](#), [7](#), [9](#), [10](#), [12](#), [13](#), [14](#), [19](#)
- [CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. Non-interactive zero-knowledge in pairing-free groups from weaker assumptions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 442–471. Springer, Heidelberg, May 2020. [6](#)
- [CPV20] Michele Ciampi, Roberto Parisella, and Daniele Venturi. On adaptive security of delayed-input sigma protocols and fiat-shamir NIZKs. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 670–690. Springer, Heidelberg, September 2020. [6](#)
- [DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012. [4](#)
- [DGI⁺19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019. [6](#)
- [FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO’86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. [5](#)
- [GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. [4](#)
- [GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003. [7](#)
- [GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008. [5](#), [7](#)
- [GR19] Alonso González and Carla Ràfols. Shorter pairing-based arguments under standard assumptions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 728–757. Springer, Heidelberg, December 2019. [5](#)
- [Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010. [4](#)
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. [4](#)
- [GZ21] Alonso González and Alexandros Zacharakis. Succinct publicly verifiable computation. Cryptology ePrint Archive, Report 2021/353, 2021. <https://eprint.iacr.org/2021/353>. [5](#), [10](#), [11](#), [21](#)

- [HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th FOCS*, pages 850–858. IEEE Computer Society Press, October 2018. [6](#)
- [HLR21] Justin Holmgren, A. Lombardi, and R. Rothblum. Fiat-shamir via list-recoverable codes (or: Parallel repetition of gmw is not zero-knowledge). *STOC*, 2021. [6](#), [7](#), [16](#), [18](#), [19](#)
- [HR18] Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In Mikkel Thorup, editor, *59th FOCS*, pages 124–135. IEEE Computer Society Press, October 2018. [4](#)
- [HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. [9](#), [16](#), [19](#), [20](#)
- [JJ21] Abhishek Jain and Zhengzhong Jin. Non-Interactive Zero Knowledge from Sub-exponential DDH. In *EUROCRYPT*, Lecture Notes in Computer Science. Springer, 2021. [4](#), [6](#)
- [JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. SNARGs for Bounded Depth Computations and PPAD Hardness from Sub-Exponential LWE. In *STOC*. ACM, 2021. [3](#), [4](#), [5](#), [7](#)
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. [5](#), [7](#)
- [KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015. [4](#)
- [KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 91–118. Springer, Heidelberg, October / November 2016. [4](#), [10](#), [11](#), [35](#)
- [KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. [4](#), [7](#), [10](#), [11](#), [12](#), [34](#), [35](#), [36](#), [37](#), [45](#)
- [KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 565–574. ACM Press, June 2013. [4](#), [10](#)
- [KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th ACM STOC*, pages 485–494. ACM Press, May / June 2014. [4](#), [10](#)
- [KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017. [6](#)
- [LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. [7](#)

- [Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012. 4
- [Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO’87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988. 37
- [Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 4
- [Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003. 4
- [PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 283–315. Springer, Heidelberg, November 2017. 4
- [PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Heidelberg, March 2012. 5
- [PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019. 6
- [RR20] Guy N. Rothblum and Ron D. Rothblum. Batch verification and proofs of proximity with polylog overhead. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 108–138. Springer, Heidelberg, November 2020. 5, 7
- [RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016. 5, 7, 8, 13, 14, 15, 24
- [RRR18] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Efficient batch verification for UP. In *Computational Complexity Conference*, volume 102 of *LIPICs*, pages 22:1–22:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 5, 7
- [Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020. 12
- [Sha92] Adi Shamir. $IP = PSPACE$. *J. ACM*, 39(4):869–877, 1992. 7

A Proof of Theorem 5

Theorem 13. *Given ℓ instances of an SECOM commitment scheme $SECOM = (SECOM.Gen, SECOM.TGen, SECOM.Com, SECOM.Open, SECOM.Verify, SECOM.Ext)$ with locality parameter 1, one can construct an L -no-signaling NS-SECOM.*

Proof. We construct the L -no-signaling NS-SECOM as follows.

- TGen($1^\lambda, 1^N, S$): The key generation algorithm generates a 1-SECOM key for each element in S , and also generates the remaining $(L - |S|)$ -SECOM keys. Then it outputs a random shuffle of all the generated keys.
 - Let $S = \{s_1, s_2, \dots, s_{|S|}\}$. For each $i \in [|S|]$, let $(K_i^*, \text{td}_i) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^N, \{s_i\})$.
 - For $[L] \setminus [|S|]$, $K_i^* \leftarrow \text{SECOM.TGen}(1^\lambda, 1^N, \phi)$, where ϕ is the empty set.
 - Let $\pi : [L] \rightarrow [L]$ be a random shuffle. Output $K^* = \{K_{\pi(i)}^*\}_{i \in [L]}$, and $\text{td} = (\{\text{td}_i\}_{i \in [|S|]}, \pi)$.
- Com($K, \mathbf{m} \in \{0, 1\}^N; r$): The commitment algorithm commits the message \mathbf{m} for each key specified in K .
 - Parse $r = r_1, r_2, \dots, r_L$, and $K = \{K'_i\}_{i \in [L]}$.
 - For each $i \in [L]$, compute $c_i \leftarrow \text{SECOM.Com}(K'_i, \mathbf{m}; r_i)$. Output $c = \{c_i\}_{i \in [L]}$.
- Ext(c, td): The extraction algorithm extracts for each element in S . It uses π to recover the order.
 - Parse $c = \{c_i\}_{i \in [L]}$, and $\text{td} = (\{\text{td}_i\}_{i \in [|S|]}, \pi)$.
 - For each $i \in [|S|]$, let $m_i^* \leftarrow \text{SECOM.Ext}(c_{\pi(i)}, \text{td}_i)$. Output $m_S^* = \{m_i^*\}_{i \in [|S|]}$.

The local opening algorithm `Open` and the local verification algorithm `Verify` repeat the same algorithms in SECOM for L times. We omit the details here.

To prove the aforementioned construction is L -no-signaling, we build a series of hybrids. For any two sets $S' \subseteq S$,

- Hyb_0 : This hybrid uses S to generate the commitment key.
 - $(K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S)$, $(c, z) \leftarrow \mathcal{D}_1(K^*)$, $\vec{y} := \text{Ext}(c, \text{td})$, Output $\mathcal{D}_2(K^*, c, \vec{y}|_{S'}, z)$.
- Hyb_1 : This hybrid is almost the same as Hyb_0 , except that, we rearrange the elements in S as $\{s_1, s_2, \dots, s_{|S|}\}$ such that the first $|S'|$ elements are exactly S' , i.e. $S' = \{s_1, s_2, \dots, s_{|S|}\}$. Furthermore, we replace the computation of $\vec{y}|_{S'}$ as direct computation without using $\{\text{td}_i\}_{i=|S'|+1, \dots, |S|}$.
 - $(K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S)$, $(c, z) \leftarrow \mathcal{D}_1(K^*)$.
 - Parse $c = \{c_i\}_{i \in [L]}$. For each $i \in [|S'|]$, let $y'_i := \text{SECOM.Ext}(c_{\pi(i)}, \text{td}_i)$. Output $\mathcal{D}_2(K^*, c, \{y'_i\}_i, z)$.

This hybrid is identical to the hybrid Hyb_0 , because the random shuffle π completely hides the order we represent S .

- $\text{Hyb}_2^{j^*}$: This hybrid is almost the same as Hyb_1 , except that we replace the TGen as follows.
 - For each $i \in |S|$, if $i \leq |S'| + j^*$, then let $K_i^* \leftarrow \text{SECOM.TGen}(1^\lambda, 1^N, \phi)$ and $\text{td}_i = \phi$. For each $|S'| + j^* < i \leq |S|$, let $(K_i^*, \text{td}_i) \leftarrow \text{SECOM.TGen}(1^\lambda, 1^N, \{s_i\})$.
 - For $[L] \setminus [|S|]$, $K_i^* \leftarrow \text{SECOM.TGen}(1^\lambda, 1^N, \phi)$, where ϕ is the empty set.
 - Let $\pi : [L] \rightarrow [L]$ be a random shuffle. Output $K^* = \{K_{\pi(i)}^*\}_{i \in [L]}$, and $\text{td} = (\{\text{td}_i\}_{i \in [|S|]}, \pi)$.

Hyb_2^1 is computationally indistinguishable with Hyb_1 . Furthermore, $\text{Hyb}_2^{j^*}$ is computationally indistinguishable with $\text{Hyb}_2^{j^*+1}$. This follows from the key indistinguishability of SECOM.

- $\text{Hyb}_3 = \text{Hyb}_2^{|S|-|S'|}$. This hybrid is identical to the following hybrid.

- $(K^*, \text{td}) \leftarrow \text{TGen}(1^\lambda, 1^N, S'), (c, z) \leftarrow \mathcal{D}_1(K^*), \vec{y} := \text{Ext}(c, \text{td}), \text{Output } \mathcal{D}_2(K^*, c, \vec{y}, z).$

By the hybrid argument, Hyb_0 and Hyb_3 are computationally indistinguishable. We finish the proof. \square