# SNARGs for P from Sub-exponential DDH and QR

James Hulett* Ruta Jawale* Dakshita Khurana* Akshayaram Srinivasan†

**Abstract**

We obtain publicly verifiable Succinct Non-Interactive Arguments (SNARGs) for arbitrary deterministic computations and bounded space non-deterministic computation from standard group-based assumptions, without relying on pairings. In particular, assuming the sub-exponential hardness of both the Decisional Diffie-Hellman (DDH) and Quadratic Residuosity (QR) assumptions, we obtain the following results, where $n$ denotes the length of the instance:

1. A SNARG for any language that can be decided in non-deterministic time $T$ and space $S$ with communication complexity and verifier runtime $(n+S)\cdot T^{o(1)}$.
2. A SNARG for any language that can be decided in deterministic time $T$ with communication complexity and verifier runtime $n \cdot T^{o(1)}$.

*University of Illinois, Urbana-Champaign. Email: {jhulett2,jawale2,dakshita}@illinois.edu
†Tata Institute of Fundamental Research. Email: akshayaram.srinivasan@tifr.res.in

# Contents

# 1 Introduction

We consider the problem of constructing *succinct, publicly verifiable* arguments to certify the correctness of computation. By succinct, we refer to the setting where the running time of verifier is much smaller than the time required to perform the computation.

The problem of constructing such proof systems has received widespread attention over the last three decades. These are typically called succinct non-interactive arguments (SNARGs), where *argument* refers to any proof system whose soundness holds against polynomial-time provers (under cryptographic assumptions) and the non-interactive setting refers to a single message of communication sent by the prover to the verifier. As in prior work, our work focuses on constructions in the CRS model, where participants have access to a common reference string.

Until recently, a significant amount of prior work on SNARGs focused on constructions proven secure under non-falsifiable assumptions or shown secure only in idealized models (such as the Random Oracle Model). Indeed, Gentry and Wichs [GW11] showed that if such an argument system satisfied a strong form of soundness called as adaptive soundness, then such non-falsifiable assumptions are necessary for SNARGs for NP. There has been recent exciting progress on constructing SNARGs for classes that are subsets of NP under falsifiable standard cryptographic assumptions, and in particular the LWE (Learning with Errors assumption), by instantiating the Fiat-Shamir paradigm, discussed next.

**The Fiat-Shamir Paradigm.** The Fiat-Shamir paradigm is a transformation that converts any public-coin interactive argument $(\mathcal{P}, \mathcal{V})$ for a language $L$ to a non-interactive argument $(\mathcal{P}', \mathcal{V}')$ for $L$. The CRS consists of randomly chosen hash functions $h_1, \ldots, h_\ell$ from a hash family $\mathcal{H}$, where $\ell$ is the number of rounds in $(\mathcal{P}, \mathcal{V})$. To compute a non-interactive argument for $x \in L$, the prover $\mathcal{P}'(x)$ generates a transcript corresponding to $(\mathcal{P}, \mathcal{V})(x)$, by emulating $\mathcal{P}(x)$ and replacing each random verifier message by a hash of the transcript so far. The verifier $\mathcal{V}'(x)$ accepts if and only if $\mathcal{V}(x)$ accepts this transcript and all verifier challenges are computed correctly as the output of the hash function on the transcript so far. This paradigm is sound when applied to constant round protocols in the Random Oracle Model (ROM) [BR93, PS96]. At the same time there are counterexamples that demonstrate its insecurity in the plain model [Bar01, GK03, CGH04a, BBH$^+$19].

The recent work of Canetti *et al.* [CCH$^+$19] and subsequent work of Peikert and Shiehian [PS19] proved the soundness of the Fiat-Shamir paradigm, assuming standard hardness of the Learning With Errors (LWE) problem, when applied to a *specific* zero-knowledge protocol. This gave the first NIZK argument from LWE. This work also obtained a SNARG for all bounded depth computations, assuming the existence of an FHE scheme with optimal circular security – which appears to be an extremely strong assumption. Subsequently, [JKKZ21] gave an instantiation of the Fiat-Shamir paradigm applied to special classes of succinct proofs, which resulted in SNARGs for bounded depth computations from sub-exponential LWE [JKKZ21]. Even more recently, Choudhuri et al. [CJJ21b] gave a construction of SNARGs for the complexity class P from polynomial LWE, using which Kalai et al. [KVZ21] gave a construction of SNARGs for bounded-space nondeterministic computation under sub-exponential LWE. The LWE assumption is a structured cryptographic assumption that is known to imply among several other interesting cryptographic primitives, compact (leveled) homomorphic encryption. In fact, all aforementioned constructions of SNARGs implicitly make use of homomorphic encryption.

On the other hand, foundational group-based assumptions such as Decisional Diffie-Hellman and Quadratic Residuosity are not known to imply homomorphic encryption, and yet their (sub-exponential) variants have surprisingly, via the Fiat-Shamir paradigm, been shown to imply non-interactive zero-knowledge [BKM20, JJ21] as well as non-trivial SNARGs for batched NP statements [CJJ21a]. This motivates the following question:

*Do there exist SNARGs for* P *(and beyond) from standard group-based assumptions like DDH and QR?*

## 1.1 Our Results

We address the above question and obtain the following positive results.

- We build a SNARG for the class of all non-deterministic computations requiring time $T(n)$ and space $S(n)$ (denoted by $\mathsf{NTISP}(T(n); S(n))$) where the prover runs in time $\mathsf{poly}(T(n))$ given a witness for the computation and the verifier runs in time $(n + S(n)) \cdot T(n)^{o(1)}$ where $n$ is the instance length.
- Plugging the SNARG above into a compiler from [KVZ21], we obtain a SNARG for the class P where the prover runs in time $\mathsf{poly}(T(n))$ and the verifier runs in time $n \cdot T(n)^{o(1)}$.

Our construction for NTISP is obtained in three steps.

1. We develop a new *folding technique* for interactive succinct arguments, where we *recursively* break down a time-$T$ computation into smaller subcomputations, each of time $T/k$ (for an appropriate choice of $k$) and have the prover send batch proofs of the validity of each subcomputation. This can be viewed as a computational analogue of the RRR interactive proof [RRR16].
2. We instantiate our protocol using batch interactive arguments for NP[1] that are "FS-compatible", which were in particular developed in [CJJ21a] based on the hardness of QR. Here, FS-compatible refers to the fact that these interactive batch NP arguments can be soundly converted into SNARGs via the Fiat-Shamir paradigm. In addition, we show that our interactive argument for NTISP is FS-compatible as long as the underlying batch NP argument is FS-compatible.
3. We then soundly convert the above succinct interactive argument to a SNARG by making use of correlation-intractable hash functions for low-depth threshold circuits constructed in [JJ21], based on sub-exponential hardness of DDH.

Finally, we note that the works of [BFJ+20, GJJM20, LVW20] observed that in addition to interactive proofs, the Fiat-Shamir paradigm can be soundly instantiated for special types of arguments. They observed that this is possible for arguments that have an *unconditionally sound mode*, and where the prover cannot detect whether the argument is unconditionally or computationally sound. These ideas were then extended to the setting of *succinct* arguments in [CJJ21a, CJJ21b]. As a contribution that may be of independent interest, we abstract out a notion of Fiat-Shamir compatibility of argument systems, which captures these broad requirements (including those used in [BFJ+20, GJJM20, LVW20, CJJ21a, CJJ21b]) that interactive *arguments* satisfy in order to soundly instantiate Fiat-Shamir from standard assumptions using known techniques.

[1] Batch arguments for NP allow a verifier to verify the correctness of $k$ NP instances with circuit complexity smaller than $k$ times the size of the NP verification circuit.

### 1.2 Other Prior Work

The works of [Mic94, Gro10, Lip12, DFH12, GGPR13, BCI+13, BCCT13, BCC+14] obtain SNARGs for non-deterministic computations, with security either in the Random Oracle Model [BR93] or from non-falsifiable "knowledge assumptions." The schemes of [CHJV15, KLW15, BGL+15, CH16, ACC+16, CCC+16, PR17] rely on assumptions related to obfuscation, which are both stronger in flavor and less widely studied than the ones used in this work. More recently, [KPY19] constructed a SNARG (for deterministic computations) based on a (new) efficiently falsifiable decisional assumption on groups with bilinear maps. Later, a line of work [CCH+19, JKKZ21, CJJ21a, CJJ21b, KVZ21] instantiated the Fiat-Shamir paradigm to finally result in SNARGs for P from the learning with errors (LWE) assumption. Very recently, the work of Gonzalez and Zacharias [GZ21] constructed SNARGs from pairing-based assumptions. On the other hand, in this work, we obtain SNARGs from assumptions that hold in pairing-free groups.

Another line of work [KRR13, KRR14, KP16, BHK17, BKK+18, BK20] built *privately verifiable* schemes for deterministic computations and a sub-class of non-deterministic computations, based on standard assumptions (specifically, the hardness of LWE or $\phi$-hiding). These schemes, however, are not publicly verifiable. The CRS is generated together with a secret key which is needed in order to verify the proofs.

In the interactive setting, publicly verifiable schemes exist, even for non-deterministic computations, under standard cryptographic assumptions [Kil92, BKP18, PRV12]. In fact some publicly verifiable interactive proof systems for restricted classes of computations exist even unconditionally, in particular for bounded depth [GKR15] and bounded space computations [RRR16].

## 2 Technical Overview

We start with a high-level overview of our recursively-built interactive argument. To begin with, we will only focus on languages that can be decided in deterministic time $T$ and space $S$. The prover will run in time $\mathsf{poly}(T)$, and the size of our proofs will grow (linearly) in $S$.

### 2.1 Succinct Interactive Arguments for Bounded Space from Succinct Arguments for Batch NP

In what follows, we describe a form of interactive arguments for bounded space computations that can be soundly compressed via the Fiat-Shamir transform. We discuss why these ideas may seem to necessitate the use of LWE, and then describe how our folding technique helps get around the need for the LWE assumption while achieving $T^{o(1)}$ verification time.

Consider a deterministic computation that takes $T$ steps: the prover and verifier agree on a (deterministic) Turing Machine $\mathcal{M}$, an input $y \in \{0,1\}^n$, and two configurations $u, v \in \{0,1\}^S$ (a configuration includes the machine's internal state, the contents of all memory tapes, and the position of the heads). The prover's claim is that after running the machine $\mathcal{M}$ on input $y$, starting at configuration $u$ and proceeding for $T$ steps, the resulting configuration is $v$. This is denoted by

$$(\mathcal{M}, y) : u \xrightarrow{T} v.$$

To prove correctness of this $T$-step computation, the prover will send $(k-1)$ alleged intermediate configurations

$$(s_1, s_2, \ldots, s_{k-1})$$

and will set $s_0 := u, s_k := v$, where for every $i \in [1, k]$, $s_i$ is the alleged configuration of the machine $\mathcal{M}$ after $T/k$ steps when starting at configuration $s_{i-1}$.

Now the prover will attempt to prove correctness of all these intermediate configurations: a naïve way to achieve this is to run $k$ executions of the base protocol, one for every $i \in [k]$. But the trick to achieving succinctness will be to prove correctness of all configurations simultaneously in verification time that is significantly smaller than running the base protocol $k$ times, while also not blowing up the prover's complexity by a factor of $k$. To enable this, the prover and verifier can rely on an appropriate succinct interactive argument for *batch* $\mathsf{NP}$ to establish that all responses would have been accepted by the verifier.

In a succinct argument for batch NP, a prover tries to convince a verifier that $(x_1, \ldots, x_k) \in \mathcal{L}^{\otimes k}$, in such a way that the proof size and communication complexity are smaller than the trivial solution where the prover simply sends all witnesses $(w_1, \ldots, w_k)$ to the verifier, and the verifier computes $\bigwedge_{i \in [k]} \mathcal{R}_{\mathcal{L}}(x_i, w_i)$. In particular, [CJJ21a] recently obtained $\mathsf{SNARG}$s for batching $k$ NP instances (from QR and sub-exponential DDH) where the communication complexity is $\widetilde{O}(|C| + k \log |C|) \cdot \mathsf{poly}(\lambda)$, and verifier runtime is $\widetilde{O}(kn + |C|) \cdot \mathsf{poly}(\lambda)$, where $\lambda$ is the security parameter, $|C|$ denotes the size of the verification circuit and $n$ denotes the size of each instance. In our setting, $|C| \approx (T/k)$, which means that verification time for the $\mathsf{SNARG}$ will be $\widetilde{O}(k + T/k)$. Setting $k = O(\sqrt{T})$, we would obtain communication complexity (and verification runtime) that grows (approx.) with $O(\sqrt{T})$ and this is the best that one can hope for in this case [CJJ21a]. However, in this work, we would like to achieve an overhead of $T^{o(1)}$.

**A Recursive Construction.** The argument described above incurred an overhead of $T/k$ because the verification circuit for each subcomputation had size $T/k$. However, what if we substituted this verification circuit with the (relatively efficient) verifier for a *succinct interactive argument for* $T/k$-time computations?

Specifically, assume there exists a *public-coin interactive* argument for verifying computations of size $T/k$. As before, suppose a prover wants to convince a verifier that

$$(\mathcal{M}, y) : u \xrightarrow{T} v.$$

The prover sends $(k-1)$ intermediate configurations, as before, and then prepares the first messages of all $k$ interactive arguments, where the $i^{th}$ interactive argument attests to the correctness of $(\mathcal{M}, y) : s_{i-1} \xrightarrow{T/k} s_i$. Instead of sending these messages in the clear, the prover sends to the verifier a *succinct commitment* to all $k$ first messages. Here, following [CJJ21a, CJJ21b, KVZ21], one could use a keyed *computationally binding* succinct commitment whose key is placed in the CRS. In fact, looking ahead, we will require a commitment that that is binding to a (hidden) part of the input string [HW15], and in fact the bound parts of the input should be extractable given a trapdoor. We will call such commitments somewhere-extractable (SE) commitments. In more detail, these commitments have a key generation algorithm $\mathsf{Gen}(1^\lambda, i)$ that on input an index $i \in [k]$ outputs a commitment key $\mathsf{ck}$ together with an extraction trapdoor $\mathsf{td}$, and an extraction algorithm that given $\mathsf{td}$ and any commitment string $c$ outputs the unique $i^{th}$ committed block (out of a total of $k$ blocks). Moreover, the commitment key hides the index $i$ in a CPA-sense.

Next, the verifier sends a single (public coin) message that serves as a challenge for all $k$ arguments. Subsequently, the prover prepares a third message for all arguments, and commits to these messages, after which the verifier again generates a single (public coin) message that serves as its fourth message for all $k$ arguments. The prover and verifier proceed until all rounds of all $k$ arguments are committed, and then the prover (as before) must prove to the verifier that all committed transcripts would be accepted.

At this point, one solution is for the prover and verifier to engage in a batch NP argument (as before), where the prover must convince the verifier that for every $i \in [k]$, there is an opening to the commitment that would cause the verifier to accept. In what follows, we will rely on the fact that the batch $\mathsf{NP}$ $\mathsf{SNARG}$ can actually be obtained in two steps: first, build an interactive argument for batch $\mathsf{NP}$, and next compress rounds of interaction via Fiat-Shamir. Indeed, the batch $\mathsf{SNARG}$ from [CJJ21a] that we will use *is obtained* by first building an interactive argument and then compressing it by soundly instantiating the Fiat-Shamir paradigm. From this point on, unless otherwise specified, we will make use of the [CJJ21a] *interactive* batch NP argument, and later separately use the fact that it can be soundly compressed via Fiat-Shamir based on sub-exponential DDH (a property referred to as $\mathsf{FS}$-compatibility). This modified interactive argument $\langle \mathsf{P}, \mathsf{V} \rangle$ for $T$-time computations is described in Figure 1, and it relies on a protocol for $T/k$-time computations.

**Batch NP and the Need for Local Openings.** Unfortunately, the protocol described in Figure 1 is *not succinct*. In particular, each batch NP statement involves verifying an opening of the SE commitment, and therefore the verification complexity of batch NP grows with the complexity of verifying commitment openings. For this to be small, the SE commitment must satisfy an important property: namely, that it is possible to *succinctly decommit* to a part of the committed input in such a way that the size of the opening and complexity of verifying openings depend only on the part being opened, and do not grow with the size of input to the commitment. Unfortunately, such commitments are only known from the learning with errors (LWE) assumption[2]; and therefore we take a different route.

Coincidentally, in the *interactive arguments* for batch NP due to [CJJ21a], the first step requires the prover to *commit* to witnesses $(w_1, \ldots, w_k)$ corresponding to each of the $k$ instances $(x_1, \ldots, x_k)$. This is done via an SE commitment in such a way that when the commitment key is binding at index $i \in [k]$, the extraction algorithm outputs the $i^{th}$ committed witness $w_i$. Moreover, this commitment does not need to have local openings; somewhere extractability suffices[3]. Finally, the [CJJ21a] protocol is actually an *argument of knowledge for one of the instances*: implicit in their proof is the fact that when the SE commitment keys (in the CRS) are binding on index $i$, no efficient prover can commit to $w_i$ that is a non-witness for $x_i$ and produce an accepting transcript (except with negligible probability).

This gives us a way out: in the Batch NP phase of our protocol, instead of proving that *there exists an opening to the commitment*, we omit sending commitments (since we already committed to all $\frac{T}{k}$ transcripts), and simply prove that for each of the transitions $s_{i-1} \to s_i$, there exists a prover strategy corresponding to verifier coins sent in the emulation phase, that would cause the verifier to accept. That is, the prover demonstrates membership of instances $(\widetilde{x}_1, \ldots, \widetilde{x}_k)$ in the language $\widetilde{\mathcal{L}}$, where for any $i \in [k]$,

$$\widetilde{x}_i = (s_{i-1}, s_i, y, \mathcal{M}, \beta)$$

[2]In Appendix A, we show that one can in fact construct a commitment with somewhat succinct local openings from DDH or QR. However, these are significantly less succinct than their LWE-based counterparts, and using these commitments would lead to marginally worse parameters than one can get with the methods described next.

[3]We remark that [CJJ21a] also require some additional linear homomorphism properties from the commitment, but these are not necessary for our discussion.

**Emulation Phase.**

1. P computes and sends $(k-1)$ intermediate configurations $(s_1, \ldots, s_{k-1})$ to V, where $s_i$ is the configuration of machine $\mathcal{M}$ after $T/k$ steps when starting at configuration $s_{i-1}$.

2. P prepares the first messages $\{m_1^{(i)}\}_{i\in[k]}$ for $k$ interactive arguments, where the $i^{th}$ interactive argument attests to the correctness of $(\mathcal{M}, x) : s_{i-1} \xrightarrow{T/k} s_i$. Next, P computes an SE commitment $c^{(1)}$ to these first messages, and sends the commitment string $c^{(1)}$ to V.

3. V generates a single (public coin) message for (a single copy of) the interactive argument for $T/k$-sized computation. All $k$ arguments will share the same verifier message.

4. More generally, for every round $j \in [\rho]$ of the underlying interactive argument,

   - P computes the $j^{th}$ round messages for all $k$ interactive arguments where the $i^{th}$ interactive argument attests to the correctness of $(\mathcal{M}, x) : s_{i-1} \xrightarrow{T/k} s_i$. Next, P computes an SE commitment $c^{(j)}$ to all these first messages, and sends the commitment string $c^{(j)}$ to V.
   - V generates a single (public coin) message for the underlying interactive argument for $T/k$-sized computation. All $k$ arguments will share the same verifier message.

**Batch NP Phase.** P proves to V that there exists an opening of the commitment $c = (c^{(1)}, \ldots, c^{(\rho)})$ where for $i \in [k]$ the $i^{th}$ opened value is an interactive argument such that:

1. The commitment verifier would accept the opening and

2. The verifier for the $T/k$ interactive argument would accept the $i^{th}$ argument.

Figure 1: Recursively Defined Interactive Argument for Bounded Space Deterministic Computation

and $\widetilde{\mathcal{L}}$ is the language of all such $\tilde{x}$ such that there exist prover messages that when combined with the verifier messages $\beta$ create an accepting transcript. We note that an honest prover, by the end of the emulation phase in Figure 1, will already be committed to witnesses for this language.

Thus our final protocol has an emulation phase that is identical to Figure 1, but the batch NP phase is modified as described in Figure 2.

It may appear that the language $\widetilde{\mathcal{L}}$ will contain nearly all strings: since the protocol for $T/k$-sized computations is an *argument*, so there will exist prover messages even for instances not in the language. However, this would only be a problem if we relied on soundness of the batch NP protocol: on the other hand, we are able to use the fact that the [CJJ21a] protocol is an *argument of knowledge* for the $i^{th}$ statement when the SE commitment key is binding at index $i$. In particular, this means that if the SE commitment was binding at index $i$, then it is possible to *efficiently extract* a witness, i.e., an accepting transcript for the $i^{th}$ subcomputation $s_{i-1} \to s_i$.

Now if the prover managed to break soundness of our protocol, this would imply that there

**Updated Batch NP Phase**

- P and V define instances

$$(\widetilde{x}_1, \ldots \widetilde{x}_k) \text{ where } \widetilde{x}_i = (s_{i-1}, s_i, y, \mathcal{M}, \beta)$$

where $\beta$ denote all verifier messages from the emulation phase.

- P additionally defines witnesses

$$(\widetilde{w}_1, \ldots \widetilde{w}_k)$$

where for every $i \in [k]$, $\widetilde{w}_i$ contains the prover messages for the $i^{th}$ subcomputation for size $\frac{T}{k}$.

- Finally, define language

$$\widetilde{\mathcal{L}} = \{(s, s', y, \mathcal{M}, \beta) \ : \ \exists \text{ prover messages } \pi \text{ s.t.}$$
$$(\pi, \beta) \text{ is accepting transcript for } (\mathcal{M}, y) : s \xrightarrow{T/k} s'.\}$$

- P and V execute a batch NP argument to prove that for every $i \in [k]$, $\widetilde{x}_i \in \widetilde{\mathcal{L}}$, where they replace the first round of Batch NP (where prover SE-commits to witnesses) with the transcript of the emulation phase.

Figure 2: Updated Batch NP Phase for Bounded Space Deterministic Computation

exists an index $j \in [k]$ such that the machine $\mathcal{M}$ on input $y$ does not transition from configuration $s_{j-1}$ to $s_j$. But, if $j = i$, where $i$ is the index where the SE commitment is binding, then one can in fact *extract* an accepting transcript for the $j^{th}$ *incorrect* subcomputation $s_{j-1} \to s_j$. This can therefore be used to build a prover that contradicts soundness of the protocol for $T/k$-sized computations. Moreover, hiding of the index $i$ ensures that $j = i$ occurs with non-negligible probability.

Finally, we point out that in the base case, i.e., for unit-time computations, the verifier simply checks the statement on its own (this takes one time-step).

The recursive protocol described so far satisfies succinctness for an appropriate choice of $k$ (that we discuss later) but requires multiple rounds, since each round of recursion adds a few rounds of interaction. The goal of this work is to build a *non-interactive* argument, which we achieve by compressing this interactive argument to a SNARG based on correlation-intractable hash functions for low-depth threshold circuits. We discuss this in detail below.

## 2.2 Obtaining a SNARG

We now discuss why this argument can be compressed by relying on the same CI hash functions as used in [CJJ21a], leading to a sound SNARG.

**Fiat-Shamir Compatible Batch NP.** To soundly compress their *batch NP* interactive argument into a SNARG, the work of [CJJ21a] (building on a line of recent works including [CCH+19, PS19, BKM20, BFJ+20, GJJM20, LVW20, JKKZ21, JJ21]) relies on a special type of hash function, called a correlation intractable hash function. The prover generates verifier messages for the interactive protocol locally by applying this hash function to its partial transcripts, in effect eliminating the need to interact with a verifier. At a high level, a hash family $\mathcal{H}$ is correlation intractable ($\mathsf{CI}$) for a relation $\mathcal{R}(x, y)$ if it is computationally hard, given a random hash key $k$, to find any input $x$ such that $(x, \mathcal{H}(k, x)) \in \mathcal{R}$.

Given a CI hash function, the key observation is that if the $\mathsf{BAD}$ verifier challenge for the interactive argument, which allows a prover to cheat, can be computed by an *efficient* function, then replacing the verifier message by the output of a $\mathsf{CI}$ hash function results in a verifier message that does not allow a prover to cheat, except with negligible probability. But this paradigm is only applicable to protocols where the circuits computing $\mathsf{BAD}$ verifier challenges are supported by constructions of CI hash functions exist based on standard assumptions. In particular, CI hash functions from (sub-exponential) DDH are known for functions that are computable by constant (and in fact, $O(\log\log\lambda)$) depth threshold circuits. Recall that the [CJJ21a] batch NP interactive argument has a first message that contains SE commitments to all witnesses; [CJJ21a] show that the SE commitment they use (which they construct based on the QR assumption) allows for extraction in constant depth. Moreover, given the witness, all other computations can also be performed by constant depth threshold circuits. Therefore, their interactive arguments can be compressed based on the (sub-exponential) DDH assumption. [CJJ21a] call this the *strong* $\mathsf{FS}$-compatible property. We will now prove that our interactive arguments for bounded space, also inherit this property.

**Fiat-Shamir Compatible Bounded Space Arguments.** To begin, we assume that all cryptographic primitives (SE commitments, CI hash functions) satisfy $T$-security, meaning that no $\mathsf{poly}(T)$-size adversary can break the primitive with advantage better than $\mathsf{negl}(T)$.

Our interactive argument begins with $\mathsf{P}$ sending $(k-1)$ intermediate configurations to $\mathsf{V}$. Observe that it is possible to verify (in time $\leq T$) whether or not a given intermediate configuration is correct[4]. Of course, the verifier should not be verifying intermediate configurations directly (as this will make verification inefficient).

As discussed above, a cheating prover must output at least one pair of consecutive intermediate configurations $s_i, s_{i+1}$ such that $\mathcal{M}$ does not transition from $s_i$ to $s_{i+1}$ in $T/k$ steps. Moreover, by $T$-index hiding of the SE commitment, if the SE commitment is set to be binding at a random index $i'$, the probability (over the randomness of $i'$) that the prover cheats on the $i'^{th}$ underlying $T/k$ interactive argument must be (negligibly) close to $1/k$. Finally, because the SE commitment is extractable, in this mode, it becomes possible for a reduction to *extract* an accepting transcript of the underlying $T/k$ argument corresponding to a false statement.

Peeling off the recursion just a little, we observe that the $(T/k)$ interactive argument itself begins with the prover sending $(k-1)$ intermediate configurations, each corresponding to $(T/k^2)$ steps of the Turing Machine $\mathcal{M}$. Again, one pair of consecutive configurations $s'_j, s'_{j+1}$ must be such that $\mathcal{M}$ does not transition from $s'_j$ to $s'_{j+1}$ in $(T/k^2)$ steps. Moreover, by index hiding of the SE commitment used in the $(T/k)$ argument, if the $(T/k)$ commitment is set to be binding at a uniformly random index $j'$, the probability that the prover cheats on the $j'^{th}$ underlying $(T/k^2)$ argument in addition to cheating on the $i'^{th}$ $(T/k)$ argument must be (negligibly) close to $(1/k^2)$.

[4]This becomes somewhat non-trivial in the non-deterministic setting, which we discuss in an upcoming subsection.

We can recurse $\log_k T$ times all the way to the base case, where the base argument is simply a unit-time computation where the verifier checks the statement on its own. Moreover, letting $\pi$ denote the unit-time protocol obtained by peeling all layers of the recursion, we can establish that with probability (close to) $(1/k^{\log_k T}) = 1/T$, $\pi$ corresponds to a false statement. The rest of our analysis will be conditioned on this event.

Assuming that the base statement $\pi$ (that the prover is statistically bound to) at the end of the first message is false, we must now understand the distribution of BAD verifier challenges in subsequent messages of the argument system. Note that the very next message will consist of the batch NP phase of the interactive argument for $k$-size computations, encrypted under $(\log_k T - 1)$ layers of SE commitments. This phase starts with commitments to $k$ witnesses (in this case, the witnesses are empty transcripts), each one proving the correctness of one of the unit-size subcomputations. The false statement from the emulation phase immediately determines which one of the batch statements is incorrect. As long as the SE commitment is binding at this index, the BAD function at this lowest layer of recursion will correspond to the set of verifier challenges in the corresponding batch NP argument that allow the prover to cheat within that argument. This means that the BAD function can be computed by *peeling off* layers of the commitment (i.e. performing $\log_k T$ sequential extractions), and then computing the BAD function for the batch NP argument (which we know is efficiently computable by a constant-depth circuit).

Next, going back up one step, we have the protocol corresponding to $k^2$-sized computations. It will again be the case that assuming the SE commitment binds at the right index, the BAD function at this layer of recursion will correspond to the set of verifier challenges in the corresponding batch NP argument that allow the prover to cheat within that argument. This means that the BAD function can be computed by peeling off $\log_k T - 1$ layers of the commitment, and then computing the BAD function for the batch NP argument (which we know is efficiently computable by a constant-depth threshold circuit).

More generally, the BAD function of our protocol corresponds to extracting from upto $\log_k T$ layers of commitments, and feeding the result as input to the BAD function circuit of the interactive argument for batch NP.

**Communication Complexity and Verifier Runtime.** Considering now the efficiency of the verifier, we note that in the emulation phase, the verifier simply has to read prover messages and generate random strings. Thus, for our overview, it suffices to focus on the batch NP phase, as the time taken there will dominate that of the emulation phase. If we were to use a trivial batch NP protocol that simply provided all $k$ witnesses and asked the verifier to check them all, this would mean that the run time of the $T$ verifier would increase by a factor of $k$ over the run time of the $T/k$ verifier. Unrolling the recursion, unfortunately, we would obtain a $T$-time verifier. Luckily, we are not constrained to use only a trivial batch NP protocol; by being more efficient, we can improve upon the above analysis. Indeed, applying the batch NP interactive argument from [CJJ21a] described above, we can improve the $k$ multiplicative overhead to a polynomial in $\lambda$ multiplicative overhead, where $\lambda$ is the security parameter of our batch NP scheme.[5]

By choosing $k$ and $\lambda$ such that $\lambda << k$, we can ensure that the difference in verifier efficiency over the $\log_k T$ levels between the unit protocol and the $T$ protocol is $\lambda^{c \cdot \log_k T}$ for some constant $c$, which can be set to $T^{o(1)}$ by a careful choice of parameters. Since the verifier run time is an upper

[5]For simplicity of exposition, we are here ignoring some additional additive overhead as well as polylogarithmic multiplicative factors.

bound on the communication complexity of the protocol (as our verifier needs to at a minimum read all the messages), this gives us the same bound on the size of the proof.

This completes an overview of our SNARGs for deterministic bounded space computation. In what follows, we will discuss how to extend these ideas to the non-deterministic setting.

### 2.3 SNARGs for Bounded Space Non-Deterministic Computation

When the machine $\mathcal{M}$ is non-deterministic it is a-priori no longer clear how to argue or even define "correctness" of intermediate configurations. It may be tempting to consider defining correctness of intermediate configurations with respect to both the instance and the witness. However, the witness used can potentially change every time the prover is queried, and is therefore not well defined. It may also in general be too large to be sent as part of the SNARG.

However, inspired by [BKK+18], we observe that if the non-deterministic Turing Machine reads each bit of the witness only once, then it becomes possible to get around this barrier. Similar to [BKK+18], we consider the class $\mathsf{NTISP}(T(n), S(n))$ of all languages recognizable by nondeterministic Turing Machines in time $O(T(n))$ and space $O(S(n))$. Recall that a non-deterministic Turing Machine allows each step of the computation to non-deterministically transition to a new state. This, in a sense, corresponds to the setting where each bit of the witness is read at most once (and if the machine wishes to remember previous non-deterministic choices it must explicitly write them down on its worktape). Thus an alternative way to describe this class is as the class of languages $\mathcal{L}$ with a corresponding witness relation $R_{\mathcal{L}}$, recognizable by a layered circuit $C_{n,m}$ parameterized by $n = |x|$ and $m = m(n) = |w|$, that on input a pair $(x, w)$ outputs 1 if and only if $R_{\mathcal{L}}(x, w) = 1$. Each layer of gates in this circuit has input wires that directly read the instance, or directly read the witness, or are the output wires of gates in the previous layer. Moreover, each bit of the witness is read by at most one layer. This circuit has depth $D = O(T(n))$ and width $W = O(S(n))$, where $W$ may be smaller than $n$ and $m$.

**The SNARG Construction.** The construction remains largely similar to the one in the deterministic setting. The only (syntactical) difference is that Step 1 in the recursively defined interactive argument from Figure 1 is modified to send wire assignments $(W_1, \dots W_{k-1})$ to $(k-1)$ intermediate layers of the circuit, each at a depth interval of $D/k$ from the base layer. Next, for every $i \in [k]$, the prover runs (parallel) interactive arguments proving that there is an assignment to witness wires such that configuration $W_i$ transitions to $W_{i+1}$ in depth $D/k$.

**Analysis.** As discussed above, unlike the deterministic setting, it appears difficult define a notion of "correctness" of these intermediate wire assignments. Instead, inspired by [BKK+18], we define the notion of an *accepting layer*.

The output layer consists only of the output wire, and thus the only valid assignment for this layer is the symbol 1. For each layer $i$, we partition the wires that are input to gates in layer $i$ into three sets: intermediate wires, instance wires, and witness wires. Intermediate wires for layer $i$ are all wires connecting gates in layer $(i-1)$ to gates in layer $i$; instance wires for layer $i$ are all wires that directly read the instance $x$ and are input to gates in layer $i$; and witness wires for layer $i$ are all wires that directly read the witness and are input to gates in layer $i$. We define $\mathsf{Acc}^D(x) = 1$. The set $\mathsf{Acc}^{D-1}(x)$ contains all possible assignments to intermediate wires connecting a gate in layer $(D-1)$ to a gate in layer $D$, such that when the instance wires for layer $D$ are set consistently with

$x$, there exists some assignment to the witness wires for layer $D$, such that the transition function applied to these wires results in output $1$.

For each layer $i < (D-1)$, the set $\mathsf{Acc}^i(x)$ is defined recursively in a similar manner. That is, for $i < (D-1)$, $\mathsf{Acc}^i(x)$ is the set of all possible assignments to intermediate wires connecting gates in layer $i$ to gates in layer $i+1$, such that when the instance wires for layer $i$ are set consistently with $x$, there exists an assignment to the witness wires for layer $i+1$, such that the transition function applied to these wires outputs intermediate wires connecting layer $i+1$ to layer $(i+2)$ that lie in the set $\mathsf{Acc}^{i+1}(x)$. We note that the lowest $i$ for which this definition is meaningful is $i = 1$, since there are no intermediate wires before the first layer.

By this definition, for $x \notin R_{\mathcal{L}}$, the set $\mathsf{Acc}^1(x)$ is empty. This implies that for any set of claimed intermediate configurations $(W_1, \ldots, W_{k-1})$ sent by P (and for $W_k = 1$), there must exist an $i \in [k-1]$ such that $W_{i+1} \in \mathsf{Acc}^{i+1}(x)$ but $W_i \notin \mathsf{Acc}^i(x)$. This means that there is *no* set of assignments to witness wires that would lead to a correct transition from $W_i$ to $W_{i+1}$. This means that the prover must be cheating in the $i^{th}$ interactive argument for $T/k$-time (non-deterministic) computation.

Moreover, as observed in [BKK+18], for any width $W$ and depth $D$ non-deterministic computation, it is possible to decide whether a set of wire assignments are in $\mathsf{Acc}^i(x)$, for any $i \in [D]$ in time $\mathsf{poly}(D, 2^W)$. This is done via a straightforward dynamic programming approach. We will set parameters so that the SE commitment is index-hiding against $\mathsf{poly}(T, 2^S)$-size adversaries. This, together with the previous claim implies that if the SE commitment is set to be binding at a random index $i'$, the probability that the prover cheats on the $i'^{th}$ underlying $T/k$ interactive argument must be (negligibly) close to $1/k$. Moreover, because the SE commitment is extractable, in this mode, it becomes possible for a reduction to *extract* an accepting transcript of the underlying $T/k$ argument for a false statement.

At this point, it becomes possible to apply the same recursive argument as in the deterministic setting to argue that with probability (negligibly) close to $1/k^{\log_k T} = 1/T$, the base argument corresponds to a false statement. Conditioned on this event, it becomes possible to analyze the batch NP phase in a manner similar to the analysis in the deterministic setting.

**SNARGs for P.** We rely on the recent work of [KVZ21] to compile our SNARGs for non-deterministic bounded-space computations to SNARGs for P.

To this end, we observe that for any language $L \in \mathsf{NTISP}(T, S)$, it holds that $L^{\otimes k} \in \mathsf{NTISP}(kT, S+T)$, where $L^{\otimes k}$ is the language of $k$ instances from $L$. This implies SNARGs for batch NP with improved parameters than the [CJJ21a] SNARGs, from sub-exponential DDH and QR. In particular, this implies SNARGs for batching $k$ instances that have a description of size $n$, and proving that the batched instances are in $L^{\otimes k}$ where $L \in \mathsf{NTISP}(T, S)$, with communication complexity and verifier runtime $k^{o(1)}(n + \mathsf{poly}(T + S))$. By plugging this into a compiler of [KVZ21] from Batch SNARGs to SNARGs for P, we obtain SNARGs for $T$-time deterministic computations with overhead $T^{o(1)}$ from sub-exponential DDH and QR. We point out that the [KVZ21] compiler as stated also requires SE commitments that allows for committing to $T$ values with local openings of size $\mathsf{polylog}(T)$. However, we show that for our setting of parameters, it suffices to have a weaker local opening property, where openings are of size $T^{o(1)}$. We build such commitments from any (sub-exponentially index-hiding) SE commitment without local openings, therefore obtaining our final results also from sub-exponential DDH and QR.

**FS-compatible Arguments.** In the body of our paper, we abstract out some general properties of our interactive arguments, and define a class of FS-compatible interactive arguments that can be soundly compressed using the Fiat-Shamir paradigm based on our technique. We show that any interactive batch NP argument that is an "FS-compatible argument" can also be converted into a proof, (intuitively) as long as its first message essentially contains a succinct commitment to witnesses for all the NP statements. We define FS-compatible interactive arguments to be those that satisfy a variant of round-by-round soundness [CCH+19] *w.r.t. a predicate*. This predicate is computed as a function of the first message of the interactive argument[6] and a trapdoor associated with the CRS. Intuitively, we will say that an interactive argument is FS-compatible w.r.t. a predicate $\phi$ if transcripts that satisfy the predicate, also satisfy round-by-round soundness with sparse and efficiently computable BAD verifier challenges. Moreover, in order to ensure that these arguments can be soundly converted into SNARGs based on CI hash functions, we will require that the predicate be "non-trivial". That is, any adversary that produces accepting transcripts for false statements with non-negligible probability should also produce accepting transcripts *that satisfy the predicate* and correspond to false statements, with non-trivial probability. We show the non-triviality of our predicate using the index hiding property of the underlying SE commitments.

**Roadmap.** A formalization of the FS-compatible property and a proof that such arguments can be converted to SNARGs can be found in Section 4. Next, in Sections 5 and 6 we formalize our constructions of SNARGs for deterministic and non-deterministic bounded-space computations, respectively. We also combine the latter with recent work [KVZ21] to obtain SNARGs for P in Section 6.5.

# 3 Preliminaries

In what follows, when we say we assume $(T_1, T_2)$-hardness of an efficiently falsifiable assumption, we mean that there exists a negligible function $\mu(\cdot)$ such that no $\mathsf{poly}(T_1)$-size adversary can falsify the assumption with probability better than $\mu(T_2)$.

## 3.1 Correlation Intractable Hash Functions

In this section, we recall the notion of a CI hash family. We start by recalling the notion of a hash function family.

**Definition 3.1.** A hash family $\mathcal{H}$ is associated with two algorithms $(\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$, and a parameter $n = n(\lambda)$, such that:

- $\mathcal{H}.\mathsf{Gen}$ is a PPT algorithm that takes as input a security parameter $1^\lambda$ and outputs a key $k$.
- $\mathcal{H}.\mathsf{Hash}$ is a polynomial time computable (deterministic) algorithm that takes as input a key $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$ and an element $x \in \{0,1\}^{n(\lambda)}$ and outputs an element $y$.

We consider hash families $\mathcal{H}$ such that for every $\lambda \in \mathbb{N}$, every key $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$ and every $x \in \{0,1\}^{n(\lambda)}$, the output $y = \mathcal{H}.\mathsf{Hash}(k, x)$ is in $\{0,1\}^\lambda$.

[6] More generally, this can be computed as a function of the entire transcript.

**Definition 3.2** (Correlation Intractable). [CGH04b, CCH+19] Fix any $T_1 = T_1(\lambda) \geq \mathsf{poly}(\lambda)$ and $T_2 = T_2(\lambda) \geq \mathsf{poly}(\lambda)$. A hash family $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ is said to be $(T_1, T_2)$ *correlation intractable* (CI) for a family $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ of efficiently enumerable relations if the following two properties hold:

- For every $\lambda \in \mathbb{N}$, every $R \in \mathcal{R}_\lambda$, and every $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$, the functions $R$ and $\mathcal{H}.\mathsf{Hash}(k, \cdot)$ have the same domain and the same co-domain.

- For every $\mathsf{poly}(T_1)$-size $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ there exists a negligible function $\mu$ such that for every $\lambda \in \mathbb{N}$ and every $R \in \mathcal{R}_\lambda$,

$$\Pr_{\substack{k\leftarrow\mathcal{H}.\mathsf{Gen}(1^\lambda)\\ x\leftarrow\mathcal{A}(k)}}[(x, \mathcal{H}.\mathsf{Hash}(k, x)) \in R] = \mu(T_2(\lambda)).$$

We will use the following theorems from prior work.

**Theorem 3.3.** *[JJ21] Fix any $T = T(\lambda) \geq 2^{\lambda^\epsilon}$ for some $0 < \epsilon < 1$. Assuming the $(T, T)$-hardness of DDH, there exists a constant $c > 0$ such that for any $B = B(\lambda) = \mathsf{poly}(\lambda)$, depth $L \leq O(\log\log\lambda)$ and any family $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ of relations that are enumerable by threshold circuits of size $B(\lambda)$ and depth $L$, there exists a $(T, T)$ correlation intractable (*CI*) hash family $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ computable in time $(B(\lambda) \cdot \lambda \cdot L)^c$, for $\mathcal{R}$ (Definition 3.2).*

## 3.2 Somewhere Extractable (SE) Commitments

**Definition 3.4** (SE Commitments). A somewhere extractable (SE) commitment consists of PPT algorithms $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ along with an alphabet $\Sigma = \{0,1\}^{\ell_{\mathsf{blk}}}$ and a fixed polynomial $p = p(\cdot)$ satisfying the following:

- $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(1^\lambda, L, \ell_{\mathsf{blk}}, i)$: Takes as input an integer $L \leq 2^\lambda$, block length $\ell_{\mathsf{blk}}$ and integer $i \in \{0, \ldots, L-1\}$ and outputs a public commitment key $\mathsf{ck}$ along with an extraction trapdoor $\mathsf{ek}$.

- $h \leftarrow \mathsf{Com}(\mathsf{ck}, x)$: is a deterministic polynomial time algorithm that takes as input $x = (x[0], \ldots, x[L-1]) \in \Sigma^L$ and outputs $h \in \{0,1\}^{\ell_{\mathsf{com}}}$.

- $\pi \leftarrow \mathsf{Open}(\mathsf{ck}, x, i)$: Given the commitment key $\mathsf{ck}$, $x \in \Sigma^L$ and an index $i \in \{0, \ldots, L-1\}$, outputs proof $\pi \in \{0,1\}^{\ell_{\mathsf{open}}}$.

- $b \leftarrow \mathsf{Verify}(\mathsf{ck}, y, i, u, \pi)$: Given a commitment key $\mathsf{ck}$ and $y \in \{0,1\}^{\ell_{\mathsf{com}}}$, an index $i \in \{0, \ldots, L-1\}$, opened value $u \in \Sigma$ and a proof $\pi \in \{0,1\}^{\ell_{\mathsf{open}}}$, outputs a decision $b \in \{0,1\}$.

- $u \leftarrow \mathsf{Extract}(\mathsf{ek}, y)$: Given the extraction trapdoor $\mathsf{ek}$ and a commitment $y \in \{0,1\}^{\ell_{\mathsf{com}}}$, outputs an extracted value $u \in \Sigma$.

We require the following properties:

- **Correctness**: For any integers $L \leq 2^\lambda$ and $i \in \{0, \ldots, L-1\}$, any $\mathsf{ck} \leftarrow \mathsf{Gen}(1^\lambda, L, i)$, $x \in \Sigma^L$, $\pi \leftarrow \mathsf{Open}(\mathsf{ck}, x, j)$: we have $\mathsf{Verify}(\mathsf{ck}, \mathsf{Com}(\mathsf{ck}, x), j, x[j], \pi) = 1$.

- **Index Hiding**: We consider the following game between an attacker $\mathcal{A}$ and a challenger:

- The attacker $\mathcal{A}(1^{T_1})$ outputs an integer $L$ and two indices $i_0, i_1 \in \{0, \ldots, L-1\}$.
- The challenger chooses a bit $b \leftarrow \{0,1\}$ and sets $\mathsf{ck} \leftarrow \mathsf{Gen}(1^\lambda, L, i_b)$.
- The attacker $\mathcal{A}$ gets $\mathsf{ck}$ and outputs a bit $b'$.

We say that an SE commitment satisfies $(T_1, T_2)$ index-hiding if for every $\mathsf{poly}(T_1)$-size attacker $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that:

$$\Big|\Pr[\mathcal{A} = 1 | b = 0] - \Pr[\mathcal{A} = 1 | b = 1]\Big| = \mu(T_2)$$

in the above game.

- **Somewhere Extractable**: We say that a commitment is somewhere extractable if there is a negligible function $\mu$ such that for every $L(\lambda) \le 2^\lambda$ and $i \in \{0, \ldots L-1\}$,

$$\Pr_{(\mathsf{ck},\mathsf{ek}) \leftarrow \mathsf{Gen}(1^\lambda, L, i)}\left[\begin{smallmatrix} \exists y \in \{0,1\}^{\ell_{\mathsf{com}}},\ u \in \Sigma,\ \pi \in \{0,1\}^{\ell_{\mathsf{open}}} \\ \text{s.t. } \mathsf{Verify}(\mathsf{ck}, y, i, u, \pi) = 1\ \wedge\ \mathsf{Extract}(\mathsf{ek}, y) \neq u \end{smallmatrix}\right] = \mu(T_2)$$

**Definition 3.5** (SE Commitments with Local Opening)**.** A somewhere extractable (SE) commitment satisfying Definition 3.4 satisfies the local opening property iff $\ell_{\mathsf{open}} < L\ell_{\mathsf{blk}}$.

**Theorem 3.6** (SE Commitments from QR [CJJ21a])**.** *Fix any $T_1 = T_1(\lambda) \ge \mathsf{poly}(\lambda)$ and $T_2 = T_2(\lambda) \ge \mathsf{poly}(\lambda)$. Assuming $(T_1, T_2)$ hardness of QR, there exists an SE commitment satisfying Definition 3.4 where the extraction algorithm can be implemented by a threshold circuit of constant depth, and which satisfies $(T_1, T_2)$-index hiding. Furthermore, this satisfies the following properties: $\ell_{\mathsf{com}} = \ell_{\mathsf{blk}}\lambda$, $\ell_{\mathsf{open}} = \ell_{\mathsf{blk}}L$, $|\mathsf{ck}| = \ell_{\mathsf{blk}}L\lambda$, $|\mathsf{ek}| = \ell_{\mathsf{blk}}\lambda$, the running time of* Gen *and* Verify *is $\ell_{\mathsf{blk}}L\lambda$ and the running time of* Extract *is $\ell_{\mathsf{blk}}\mathsf{poly}(\lambda)$.*

In Appendix A, we also build SE commitments from DDH or QR, and show how to generically obtain a non-trivial local opening property by stacking such commitments in a Merkle tree of appropriate arity. These are then plugged into the [KVZ21] compiler, together with SNARGs for batch NP from this work, to obtain SNARGs for P.

# 4 Fiat-Shamir for Arguments

In this section, we define a class of (multi-round) interactive arguments to which the Fiat-Shamir paradigm can be soundly applied, based on an (appropriate) correlation-intractable hash function. In particular, we will define a few properties that a multi-mode interactive argument should satisfy, in order to be converted to a non-interactive one by applying our technique. We begin with a natural definition of multi-mode interactive arguments:

**Definition 4.1** ($N$-Mode Protocols)**.** Let $N(\lambda) \ge \lambda$ be a function. We say that $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ is an $N$-mode protocol for a language $\mathcal{L}$ if the following property holds:

- **Syntax:** Setup is a randomized algorithm that obtains input a security parameter $\lambda$ and some $i \in [N(\lambda)]$. Setup outputs common reference string CRS and auxiliary information aux such that aux contains $i$.

Next, we define a notion of a predicate, that applies to the first prover message, the instance and a trapdoor in the CRS.

**Definition 4.2** (Predicate)**.** $\phi$ is a predicate for an $N$-mode (Definition 4.1) protocol $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ if $\phi$ has the following property:

- **Syntax**: For any $i \in [N(\lambda)]$, $\phi$ takes as input instance $x$, the first prover message $\alpha_1$, and some auxiliary information $\mathsf{aux}$ computed by $\mathsf{Setup}(1^\lambda, i)$. $\phi$ outputs a binary value in $\{0, 1\}$.

**Definition 4.3** ($(T', N)$-Non-Trivial Predicate)**.** Let $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be an $N$-mode (Definition 4.1) public-coin interactive proof system for a language $\mathcal{L}$. We say that a predicate $\phi$ for $\Pi$ (Definition 4.2) is time-$T'$ non-trivial for $\Pi$ if the following properties hold:

- **Syntax**: For any $\lambda \in \mathbb{Z}^+$, any instance $x$, any $i \in [N(\lambda)]$, any $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}(1^\lambda, i))$, and any (partial) transcript $\tau = (\alpha_1, \beta_1, \ldots, \alpha_j)$ for some $j \in [\rho(\lambda)]$, we define $\phi(x, \tau, \mathsf{aux}) = \phi(x, \alpha_1, \mathsf{aux})$.

- **Non-Triviality**: There exists a polynomial $p(\cdot)$ such that for $\lambda \in \mathbb{Z}^+$, and any $\mathsf{poly}(T')$-time adversary $\mathcal{A}$, if there exists a polynomial $q(\cdot)$ such that:

$$\Pr_{\substack{i \leftarrow [N], \\ (\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x, \alpha_1) \leftarrow \mathcal{A}(\mathsf{CRS})}} [x \notin \mathcal{L} \wedge x \neq \perp] \geq \frac{1}{q(\lambda)}$$

  then

$$\Pr_{\substack{i \leftarrow [N], \\ (\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x, \alpha_1) \leftarrow \mathcal{A}(\mathsf{CRS})}} [\phi(x, \alpha_1, \mathsf{aux}) = 1 | x \notin \mathcal{L} \wedge x \neq \perp] \geq \frac{1}{p(N(\lambda))}.$$

- **Efficiency**: $\phi$ can be evaluated in $\mathsf{poly}(T')$ time.

### 4.1 Round-by-Round Soundness

We now define a notion of round-by-round soundness for interactive arguments w.r.t. a predicate $\phi$. The definition below is a generalization of the definition in [CCH+19] to the setting of interactive arguments.

Unlike [CCH+19], we don't define State on the empty transcript, instead only starting to define it once the first prover message has been sent. The key difference from [CCH+19] is that we define the State function on the first prover message to reject when the predicate $\phi(x, \alpha_1, \mathsf{aux}) = 1$, instead of defining it to reject when $x \notin \mathcal{L}$. In particular, if we apply the definition below with the predicate $\phi(x, \alpha_1, \mathsf{aux}) = x \notin \mathcal{L}$ (and modify the syntax of Setup appropriately), we will recover the definition in [CCH+19, JKKZ21].

**Definition 4.4** ($b$-Round-by-Round Soundness w.r.t. $\phi$)**.** [CCH+19] Let $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a public-coin $N$-mode (Definition 4.1) interactive proof system for a language $\mathcal{L}$. We say that $\Pi$ is $b$-round-by-round sound with respect to predicate $\phi$ (Definition 4.2), if there exists State such that, denoting the size of every verifier message by $\lambda$, for any $i \in [N(\lambda)]$, any $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}(1^\lambda, i))$, the following properties hold:

1. **Syntax:** State is a deterministic function that takes as input the CRS, an instance $x$, a transcript prefix $\tau$, and auxiliary information aux computed by Setup. State outputs either accept or reject.

For every $x$, every non-empty transcript $\tau = (\alpha_1, \beta_1, \ldots, \alpha_j, \beta_j)$, and any next prover message $\alpha_{j+1}$, we have

$$\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{State}(\mathsf{CRS}, x, \tau \| \alpha_{j+1}, \mathsf{aux}).$$

2. **End Functionality:** For every $x$ and every first prover message $\alpha_1$, $\mathsf{State}(\mathsf{CRS}, x, \alpha_1, \mathsf{aux}) = \mathsf{reject}$ iff $\phi(x, \alpha_1, \mathsf{aux}) = 1$. For every complete transcript $\tau$, if $\mathcal{V}(\mathsf{CRS}, x, \tau) = 1$, $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$.

3. **Sparsity:** For every $x$ and every transcript prefix $\tau = (\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}, \alpha_j)$, if $\phi(x, \alpha_1, \mathsf{aux}) = 1$ and $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$, it holds that

$$\Pr_{\beta \leftarrow \{0,1\}^\lambda}[\mathsf{State}(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}] \leq b(\lambda) \cdot 2^{-\lambda}. \quad (1)$$

## 4.2 FS-Compatible Arguments

In the following definition, we formalize the requirements from round-by-round sound arguments w.r.t. $\phi$ that allow them to be compressed by the Fiat-Shamir paradigm via our approach.

**Definition 4.5** (FS-Compatible Multi-mode Argument with Respect to $\phi$)**.** For some $\rho, N : \mathbb{Z}^+ \to \mathbb{Z}^+$, let $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a $\rho$-round $N$-mode (Definition 4.1) public-coin interactive argument system where Setup is a randomized algorithm that obtains input a security parameter $\lambda$ and some $i \in [N(\lambda)]$. For any $B, b, d : \mathbb{Z}^+ \to \mathbb{Z}^+$, we say that $\Pi$ is $(B, b, d)$ FS-compatible with respect to predicate $\phi$ (Definition 4.2) if the following properties hold:

1. **Completeness:** For any $\lambda \in \mathbb{Z}^+$, $i \in \{1, 2, \ldots, N(\lambda)\}$, and $x \in \mathcal{L}$, we have

$$\Pr_{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, i)}[\langle \mathsf{P}, \mathsf{V} \rangle(\mathsf{CRS}, x) = \mathsf{accept}] = 1.$$

2. **$b$-Round-by-round soundness w.r.t. $\phi$:** $\Pi$ is $b$-round-by-round sound with respect to $\phi$ (Definition 4.4); let State be the corresponding state function.

3. **$d$-depth $B$-efficient BAD w.r.t. $\phi$:** For any $\lambda \in \mathbb{Z}^+$, any $i \in [N(\lambda)]$, any $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}(1^\lambda, i))$, there exists a (non-uniform) randomized function $\mathsf{BAD}_{\mathsf{aux}}$ that satisfies the following guarantees:

   - **Syntax:** $\mathsf{BAD}_{\mathsf{aux}}$ is hardwired with aux and takes as input the CRS, instance $x$, a partial transcript $\tau = (\alpha_1, \beta_1, \ldots, \alpha_i)$; and potentially additional uniform randomness $r$.
   - BAD **w.r.t.** $\phi$**:** For every $x$ and every $\tau \triangleq (\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}, \alpha_j)$ s.t. $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ and $\phi(x, \alpha_1, \mathsf{aux}) = 1$, $\mathsf{BAD}_{\mathsf{aux}}(\mathsf{CRS}, x, \tau)$ enumerates the set $\mathcal{B}_{\mathsf{CRS}, \phi, \mathsf{aux}, \tau}$, where

   $$\mathcal{B}_{\mathsf{CRS}, \phi, \mathsf{aux}, \tau} := \{\beta : \mathsf{State}(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}\}.$$

     If $\mathcal{B}_{\mathsf{CRS}, \mathsf{aux}} = \emptyset$, $\mathsf{BAD}_{\mathsf{aux}}(\mathsf{CRS}, x, \tau)$ outputs $\perp$. By Equation (1), $|\mathcal{B}_{\mathsf{CRS}, \mathsf{aux}}| \leq b(\lambda)$.
   - **$d$-Depth, $B$-Efficient computation:** $\mathsf{BAD}_{\mathsf{aux}}$ can be evaluated by a $d(\lambda)$-depth (non-uniform) threshold circuit of size $B = B(\lambda)$.

In what follows, we first recall the Fiat-Shamir paradigm as applied to interactive arguments, and then prove that arguments satisfying Definition 4.5 with respect to a non-trivial predicate (Definition 4.3) can be soundly compressed to obtain a SNARG via this paradigm.

### 4.3 The Fiat-Shamir Paradigm

Let $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be any public-coin interactive argument for a language $\mathcal{L}$. Let $n = n(\lambda)$ denote the communication complexity of $\Pi$. Let $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ be hash family such that, for every security parameter $\lambda \in \mathbb{N}$ and every $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$, $\mathcal{H}.\mathsf{Hash}(k, \cdot)$ is a function with a domain $\{0,1\}^{n(\lambda)}$ and co-domain $\{0,1\}^\lambda$. We will also allow inputs to $\mathcal{H}.\mathsf{Hash}(k, \cdot)$ that are shorter than $n$, by padding all inputs with 0's until the total length is $n$. We define the non-interactive protocol $\Pi_{\mathsf{FS}}^{\mathcal{H}} = (\mathcal{P}', \mathcal{V}')$, obtained by applying the Fiat-Shamir transform to $\Pi$ w.r.t. the hash family $\mathcal{H}$, in Figure 3.

**The Non-Interactive Argument $\Pi_{\mathsf{FS}}^{\mathcal{H}}$.**

Fix an input length $|x|$ and let $\lambda = \lambda(|x|)$.

- The common reference string CRS consists of two parts, $\mathsf{CRS}_1, \mathsf{CRS}_2$ where $\mathsf{CRS}_1 \leftarrow \mathsf{Setup}(1^\lambda, i)$ for $i \xleftarrow{\$} [N]$, and $\mathsf{CRS}_2$ contains one key $k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$.
- The prover $\mathcal{P}'$ takes as input $(\mathsf{CRS}, x)$ and does the following:
  1. Set $i = 1$ and $\tau_0 = \emptyset$.
  2. Compute $\alpha_i \leftarrow \mathcal{P}(x, \tau_{i-1})$ and $\beta_i = \mathcal{H}.\mathsf{Hash}(k, \tau_{i-1}|\alpha_i)$.
  3. Set $\tau_i = (\tau_{i-1}|\alpha_i|\beta_i)$.
  4. If $i = \ell$ then output $\tau_i$. Otherwise, set $i = i + 1$ and go to Item 2.
- The verifier $\mathcal{V}'$ takes as input $(\mathsf{CRS}, x, \tau)$ and does the following:
  1. Parse $\mathsf{CRS} = (\mathsf{CRS}_1, \mathsf{CRS}_2)$ and $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\ell, \beta_\ell)$.
  2. Accept if and only if $\mathcal{V}(\mathsf{CRS}_1, x, \tau) = 1$ and for every $i \in [\ell]$ it holds that $\beta_i = \mathcal{H}.\mathsf{Hash}(\mathsf{CRS}_2, \tau_{i-1}|\alpha_i)$, where $\tau_{i-1} = (\alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1})$.

Figure 3: The Non-Interactive Argument $\Pi_{\mathsf{FS}}^{\mathcal{H}}$

### 4.4 From FS-Compatible Arguments to SNARGs

**Definition 4.6** (($T', N$)-Sound Non-interactive Arguments)**.** For any $T' = T'(\lambda)$ and $N = N(\lambda)$, we say that a $N$-mode protocol (Definition 4.1) $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ is a non-interactive argument for a language $\mathcal{L}$ if the following properties hold:

- **Completeness:** For any $\lambda \in \mathbb{Z}^+$, any $i \in [N(\lambda)]$, and $x \in \mathcal{L}$, we have that

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, i) \\ \tau \leftarrow \mathsf{P}(1^\lambda, \mathsf{CRS})}} [\mathsf{V}(\mathsf{CRS}, x, \tau) = \mathsf{accept}] = 1.$$

- **$N$-Mode Indistinguishability of CRS:** There exists a negligible function $\mu(\cdot)$ such that for any $i_1, i_2 \in [N(\lambda)]$, and any $\mathsf{poly}(T')$-time adversary $\mathcal{A}$ we have that

$$\left| \Pr_{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, i_1)} [\mathcal{A}(\mathsf{CRS}) = 1] - \Pr_{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, i_2)} [\mathcal{A}(\mathsf{CRS}) = 1] \right| = \mu(N(\lambda)).$$

- **Adaptive Soundness:** There exists a negligible function $\mu(\cdot)$ such that for any $\lambda \in \mathbb{Z}^+$, any $i \in [N(\lambda)]$, and any non-uniform $\mathsf{poly}(T')$-time adversary $\mathcal{A}$ we have that

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, i) \\ (x,\tau) \leftarrow \mathcal{A}(1^\lambda, \mathsf{CRS})}} [x \notin \mathcal{L} \ \wedge \ \mathsf{V}(\mathsf{CRS}, x, \tau) = 1] \leq \mu(N(\lambda)).$$

**Theorem 4.7** (FS-Compatible)**.** *Suppose that there exist $N, T', B, b, d$ (all functions of $\lambda$) where $N, T' \geq \lambda$. Let $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a $\rho(\lambda)$-round $N(\lambda)$-mode (Definition 4.1) protocol for a language $\mathcal{L}$ decidable in (deterministic) time $\mathsf{poly}(T')$. Let $\Pi$ have prover runtime $T_\mathsf{P}$ and verifier runtime $T_\mathsf{V}$. Let $\mathcal{H}$ be a hash function. If $\Pi$ and $\mathcal{H}$ are such that:*

- *$\Pi$ is $(B, b, d)$-FS-compatible according to Definition 4.5 with respect to a $(T', N)$ nontrivial predicate $\phi$ (Definition 4.3).*
- *$\mathcal{H}$ is $(T', N)$ CI (Definition 3.2) for all relations sampleable by $d$-depth threshold circuits of size $B$, and is computable in time $p(B)$ for some fixed polynomial $p(\cdot)$.*

*Then $\Pi^{\mathcal{H}}_{\mathsf{FS}}$ (according to Figure 3) is a $(T', N)$-sound non-interactive argument system for $\mathcal{L}$ (Definition 4.6). $\Pi^{\mathcal{H}}_{\mathsf{FS}}$ has $\rho(\lambda) \cdot p(B(\lambda)) + T_\mathsf{P}$ prover runtime and $\rho(\lambda) \cdot p(B(\lambda)) + T_\mathsf{V}$ verifier runtime.*

*Proof.* The completeness of $\Pi^{\mathcal{H}}_{\mathsf{FS}}$ follows directly from that of $\Pi$. Moreover, prover runtime in the non-interactive protocol equals the runtime $T_\mathsf{P}$ of the interactive prover, plus the time needed to hash $\rho$ messages, where each hash computation takes time $p(B(\lambda))$. Therefore total prover runtime equals $\rho(\lambda) \cdot p(B(\lambda)) + T_\mathsf{P}$. Verifier runtime in the non-interactive protocol equals the runtime $T_\mathsf{V}$ of the interactive verifier, plus the time needed to hash $\rho$ messages, where each hash computation takes time $p(B(\lambda))$. Therefore total verifier runtime equals $\rho(\lambda) \cdot p(B(\lambda)) + T_\mathsf{V}$.

Next, we prove that if $\Pi$ is adaptively sound then $\Pi^{\mathcal{H}}_{\mathsf{FS}}$ is adaptively sound.
In what follows, unless specified otherwise, all probabilities are over the randomness of sampling:

$$i \xleftarrow{\$} [N], (\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda), (x, \tau) \leftarrow \mathcal{A}_{\mathsf{FS}}(1^\lambda, \mathsf{CRS} \| k)$$

Suppose for the sake of contradiction that there existed some $\mathsf{poly}(T')$-time adversary $\mathcal{A}_{\mathsf{FS}}$, some polynomial $q(\cdot)$ and infinitely many $\lambda \in \mathbb{Z}^+$,[7]

$$\Pr[x \notin \mathcal{L} \wedge \mathsf{V}^{\mathcal{H}}_{\mathsf{FS}}(\mathsf{CRS} \| k, x, \tau) = 1] \geq \frac{1}{q(N(\lambda))} \tag{2}$$

We claim that there exists a polynomial $p(\cdot)$ such that:

$$\Pr[x \notin \mathcal{L} \wedge \mathsf{V}^{\mathcal{H}}_{\mathsf{FS}}(\mathsf{CRS} \| k, x, \tau) = 1 \wedge \phi(x, \alpha_1, i, \mathsf{aux}) = 1] \geq \frac{1}{p(N(\lambda))} \tag{3}$$

[7] Note that for the purposes of this proof, we will use CRS to denote the CRS of the *interactive* protocol; the CRS for the non-interactive protocol will contain both CRS and the CI hash key $k$.

where $\alpha_1$ denotes the first prover message in $\tau$.

Suppose this claim is not true, then there exists a negligible function $\mu(\cdot)$ such that:

$$\Pr[x \notin \mathcal{L} \wedge \mathsf{V}_{\mathsf{FS}}^{\mathcal{H}}(\mathsf{CRS}\|k, x, \tau) = 1 \wedge \phi(x, \alpha_1, \mathsf{aux}) = 1] = \mu(N(\lambda)) \tag{4}$$

We consider a $\mathsf{poly}(T')$-time adversary $\mathcal{A}_{\mathsf{NT}}$ defined as follows: $\mathcal{A}_{\mathsf{NT}}(1^\lambda, \mathsf{CRS})$ samples $k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$ and then obtains $(x, \tau) \leftarrow \mathcal{A}_{\mathsf{FS}}(1^\lambda, \mathsf{CRS}\|k)$. If $\mathsf{V}_{\mathsf{FS}}^{\mathcal{H}}(\mathsf{CRS}\|k, x, \tau) = 1$ and $x \notin \mathcal{L}$, $\mathcal{A}$ outputs $(x, \alpha_1)$ where $\alpha_1$ is the first message of $\tau$. Otherwise $\mathcal{A}_{\mathsf{NT}}$ outputs $\perp$. Equations (2) and (4) together imply that:

$$\Pr_{\substack{i \xleftarrow{\$} [N] \\ (\mathsf{CRS},\mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i) \\ (x,\alpha_1) \leftarrow \mathcal{A}_{\mathsf{NT}}(1^\lambda, \mathsf{CRS})}} [\phi(x, \alpha_1, \mathsf{aux}) = 1 | x \notin \mathcal{L} \wedge x \neq \perp] \leq \mu(N(\lambda)) \cdot q(\lambda) = \mathsf{negl}(N(\lambda)) \tag{5}$$

which contradicts the non-triviality of $\phi$ according to Definition 4.5. Therefore, equation (3) should be true.

Let $\mathbb{E}_0(k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = \Big(x \notin \mathcal{L} \wedge \mathsf{V}_{\mathsf{FS}}^{\mathcal{H}}(\mathsf{CRS}, x, \tau) = 1 \wedge \phi(x, \alpha_1, \mathsf{aux}) = 1\Big)$. We then obtain that

$$\Pr[\mathbb{E}_0(k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = 1] \geq \frac{1}{p(N(\lambda))}. \tag{6}$$

Let State be the function given by Definition 4.5. For the sake of argument, fix any $k \in \mathsf{Support}(\mathcal{H}.\mathsf{Gen}(1^\lambda))$, any $i \in [N(|x|)]$, any $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}(1^{|x|}, i))$, and any pair of instance and transcript $(x, \tau) \in \mathsf{Support}(\mathcal{A}_{\mathsf{FS}}(1^\lambda, \mathsf{CRS}\|k))$ such that $\mathbb{E}_0(k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = 1$. Since $\phi(x, \alpha_1, \mathsf{aux}) = 1$, $\mathsf{State}(\mathsf{CRS}, x, \alpha_1, \mathsf{aux}) = \mathsf{reject}$ Since $\mathsf{V}_{\mathsf{FS}}^{\mathcal{H}}(\mathsf{CRS}, x, \tau) = 1$, then $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$. Thus, letting $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\rho, \beta_\rho)$ and $\tau_\ell = (\alpha_1, \beta_1, \ldots, \alpha_\ell, \beta_\ell)$ for all $\ell \in [\rho]$, there must exist an index $j = j(\lambda) \in [\rho(\lambda)]$ such that $\mathsf{State}(\mathsf{CRS}, x, \tau_{j-1}\|\alpha_j, \mathsf{aux}) = \mathsf{reject}$ but $\mathsf{State}(\mathsf{CRS}, x, \tau_j, \mathsf{aux})$ $= \mathsf{accept}$. Let $\mathcal{B}_{\mathsf{CRS},\mathsf{aux}}$ be the set defined in Definition 4.5 for instance $x$ and transcript $\tau_{j-1}\|\alpha_j$. By definition, $\beta_j \in \mathcal{B}_{\mathsf{CRS},\mathsf{aux}}$ and since $\mathsf{V}_{\mathsf{FS}}^{\mathcal{H}}(\mathsf{CRS}, x, \tau) = 1$, we must have that $\mathcal{H}.\mathsf{Hash}(k, x\|\tau_{j-1}\|\alpha_j) = \beta_j$. We will now mathematically summarize the result of this argument. Let

$$\mathbb{E}_1(j, k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = \left(\phi(x, \alpha_1, \mathsf{aux}) = 1 \wedge \mathcal{H}.\mathsf{Hash}(k, x\|\tau_{j-1}\|\alpha_j) \in \mathcal{B}_{\mathsf{CRS},\phi,\mathsf{aux},\tau_{j-1}}\right).$$

We have that

$$\begin{aligned}
&\Pr[\exists j \in [\rho] \text{ s.t. } \mathbb{E}_1(j, k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = 1] \\
&\geq \Pr[\mathbb{E}_0(k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = 1] \geq \frac{1}{p(N(\lambda))}.
\end{aligned}$$

When we sample the index $j$ independently and uniformly at random, we have that

$$\Pr_{\substack{i \xleftarrow{\$} [N],\ j \xleftarrow{\$} [\rho] \\ (\mathsf{CRS},\mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i) \\ k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda) \\ (x,\tau) \leftarrow \mathcal{A}_{\mathsf{FS}}(1^\lambda, \mathsf{CRS}\|k)}} [\mathbb{E}_1(j, k, \mathsf{CRS}, \mathsf{aux}, x, \tau) = 1] \geq \frac{1}{\rho(\lambda)} \cdot \frac{1}{p(N(\lambda))}. \tag{7}$$

We will now construct an adversary $\mathcal{A}_{\mathsf{CI}}$ that breaks the correlation-intractability of $\mathcal{H}$ (Definition 3.2). Define relation $\mathcal{R}$ to be the relation sampled by the circuit $\mathsf{BAD}_{\mathsf{aux}}$ for $\Pi$, this circuit exists

Algorithm $\mathcal{A}_{\mathsf{CI}} = \{\mathcal{A}_{\mathsf{CI},\lambda}\}_{\lambda\in\mathbb{N}}$ which does the following:

- Sample $(\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, [N])$.
- Obtain key $k$ (generated as $k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$ where $\mathcal{H}$ is $\phi$-CI w.r.t. relation $\mathcal{R}$).
- Compute $(x, \tau) \leftarrow \mathcal{A}_{\mathsf{FS}}(1^\lambda, \mathsf{CRS}\|k)$.
- Parse $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\rho, \beta_\rho)$ and sample $j \xleftarrow{\$} [\rho(\lambda)]$.
- Output $(x, \tau_{j-1}\|\alpha_j)$ where $\tau_{j-1} = (\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1})$.

Figure 4: Algorithm $\mathcal{A}_{\mathsf{CI}}$ that breaks the correlation intractable property of $\mathcal{H}$.

by Definition 4.5 and on any $x$ outputs one out of a set of $b(\lambda)$ strings. Moreover, for any $x$ such that $\phi(x, \alpha_1, \mathsf{aux}) = 1$, $\mathsf{BAD}_{\mathsf{aux}}$ (w.h.p.) outputs a uniformly random element in the set $\mathcal{B}_{\mathsf{CRS},\phi,\mathsf{aux}}$.

For $\mathcal{A}_{\mathsf{CI}}$ as defined in Figure 4, we can see from Equation 7 and our above argument that there exists a polynomial $p'(\cdot)$ such that

$$\begin{aligned}
&\Pr_{\substack{\mathcal{B}_{\mathsf{CRS},\mathsf{aux}}\leftarrow\mathcal{A}_{\mathsf{CI}}(1^\lambda)\\ k\leftarrow\mathcal{H}.\mathsf{Gen}(1^\lambda)\\ (x,\tau_{j-1}\|\alpha_j)\leftarrow\mathcal{A}_{\mathsf{CI}}(1^\lambda,k)}}[\mathbb{E}_1(j,k,\mathsf{CRS},\mathsf{aux},x,\tau)=1]\\
\geq &\Pr_{\substack{i\xleftarrow{\$}[N],j\xleftarrow{\$}[\rho(\lambda)]\\ (\mathsf{CRS},\mathsf{aux})\leftarrow\mathsf{Setup}(1^\lambda,i)\\ k\leftarrow\mathcal{H}.\mathsf{Gen}(1^\lambda)\\ (x,\tau)\leftarrow\mathcal{A}_{\mathsf{FS}}(1^\lambda,\mathsf{CRS}\|k)}}[\mathbb{E}_1(j,k,\mathsf{CRS},\mathsf{aux},x,\tau)=1] \geq \frac{1}{\rho(\lambda)\cdot p(N(\lambda))} \geq \frac{1}{p'(N(\lambda))}.
\end{aligned}$$

By definition of event $\mathbb{E}_1$, we have that $\mathcal{A}_{\mathsf{CI}}$ will satisfy both $\mathcal{H}.\mathsf{Hash}(k, x\|\tau_{j-1}\|\alpha_j) \in \mathcal{B}_{\mathsf{CRS},\phi,\mathsf{aux},\tau_{j-1}}$ and $\phi(x, \alpha_1, \mathsf{aux}) = 1$ with non-negligible (in $N$) probability. This implies (by definition of the relation $\mathcal{R}$ above) that $\mathcal{H}.\mathsf{Hash}(k, x\|\tau_{j-1}\|\alpha_j) \in \mathcal{R}$ with non-negligible (in $N$) probability. Thus, $\mathcal{A}_{\mathsf{CI}}$ contradicts the $(\lambda, N)$ correlation intractability of $\mathcal{H}$, as desired.

Therefore, $\Pi^{\mathcal{H}}_{\mathsf{FS}}$ is adaptively and computationally sound. □

# 5 FS-compatible Arguments for Bounded Space Computations

In this section, we describe and prove FS-compatibility of our interactive arguments for bounded space computation. Before providing a formal theorem, in the following subsection, we define an FS-Compatible Batch NP argument with respect to a batch predicate and SE commitment. We will bootstrap interactive arguments for batch NP satisfying this definition to obtain interactive arguments for bounded space computation.

## 5.1 FS-Compatible Batch NP Arguments

Let $\Pi_{\mathsf{BNP}} = (\mathsf{Setup}_{\mathsf{BNP}}, \mathsf{P}_{\mathsf{BNP}}, \mathsf{V}_{\mathsf{BNP}})$ be a public-coin argument system for $\mathcal{R}^k$ for some circuit satisfiability relation $\mathcal{R}$ such that $\mathsf{Setup}_{\mathsf{BNP}}(1^\lambda, i)$ runs $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathcal{C}.\mathsf{Gen}(1^\lambda, k, i)$ for some SE commitment scheme $\mathcal{C}$ and puts $\mathsf{ck}$ in CRS and $(i, \mathsf{ek})$ in $\mathsf{aux}$. Then we define a predicate $\phi_{\mathsf{BNP}}$ such that

$$\phi_{\mathsf{BNP}}((x_1, \ldots, x_k), \alpha_1, \mathsf{aux}) = \Big((x_i, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_1)) \notin \mathcal{R}\Big).$$

**Definition 5.1** (FS-compatible Batch NP w.r.t. $\mathcal{C}$ and $\phi_{\mathsf{BNP}}$)**.** Let $\Pi_{\mathsf{BNP}} = (\mathsf{Setup}_{\mathsf{BNP}}, \mathsf{P}_{\mathsf{BNP}}, \mathsf{V}_{\mathsf{BNP}})$ be a public-coin argument system for $\mathcal{R}^k = \mathcal{R}_{n,m,s,\mathbb{F}}$ for C-SAT relation $\mathcal{R} = \mathcal{R}_{n,m,s,\mathbb{F}}$ represented as

$$\mathcal{R}_{n,m,s,\mathbb{F}} = \mathcal{R}_C = \{(x, \omega) \;:\; C(x, \omega) = 1\} \text{ where}$$

$x$ is a vector in $\mathbb{F}^n$, $\omega$ is a vector in $\mathbb{F}^m$, and $|C| = s$. We say that $\Pi_{\mathsf{BNP}}$ is FS-compatible Batch NP with respect to a somewhere extractable commitment scheme $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ (Definition 3.4) if there exist $T', b, d$ (all functions of $\lambda$) such that we have the following properties:

- **Syntax:** $\mathsf{Setup}_{\mathsf{BNP}}(1^\lambda, i)$ runs $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathcal{C}.\mathsf{Gen}(1^\lambda, k, i)$ and puts $\mathsf{ck}$ in CRS and $(i, \mathsf{ek})$ in $\mathsf{aux}$.
- **FS-compatible w.r.t. $\phi_{\mathsf{BNP}}$:** $\Pi_{\mathsf{BNP}}$ is $(T', b, d)$ FS-compatible according to Definition 4.5 with respect to the predicate $\phi_{\mathsf{BNP}}$.
- **Completeness**: For any $\lambda \in \mathbb{Z}^+$, any $i \in [N(\lambda)]$, and any $((x_1, w_1), \ldots, (x_k, w_k)) \in \mathcal{R}^k$,

  $$\Pr_{(\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}_{\mathsf{BNP}}(1^\lambda, i)}[\langle \mathsf{P}_{\mathsf{BNP}}(w_1, \ldots, w_k), \mathsf{V}_{\mathsf{BNP}}\rangle(\mathsf{CRS}, (x_1, \ldots, x_k)) = 1] = 1$$

  where the first message of $\mathsf{P}_{\mathsf{BNP}}(w_1, \ldots, w_k)$ is $\mathcal{C}.\mathsf{Com}(\mathsf{ck}, (w_1, \ldots, w_k))$.
- **Complexity:** For any $\lambda \in \mathbb{Z}^+$, any $i \in [k(\lambda)]$, any $(\mathsf{ck}, \mathsf{ek}) \in \mathsf{Support}(\mathcal{C}.\mathsf{Gen}(1^\lambda, k, i))$, $\Pi_{\mathsf{BNP}}$ has communication complexity $\widetilde{O}\Big(s + k \log s\Big) \cdot \mathsf{poly}\lambda$, verifier runtime of $\widetilde{O}\Big(kn + s\Big) \cdot \mathsf{poly}\lambda$, and prover runtime of $\mathsf{poly}(k \cdot s)$ where $s = |C|$ and $n = |x|$.

**Theorem 5.2** (FS-compatible Batch NP w.r.t. $\mathcal{C}$ [CJJ21a])**.** *Assuming the hardness of* QR*, for any $n = n(\lambda), m = m(\lambda), s = s(\lambda), k = k(\lambda)$, and field $\mathbb{F}$ where $|\mathbb{F}| \leq 2^\lambda$ there exists an FS-compatible Batch NP w.r.t. $\mathcal{C}$ and $\phi_{\mathsf{BNP}}$ (Definition 5.1), where $\mathcal{C}$ satisfies Definition 3.4, for $\mathcal{R}^k_{n,m,s,\mathbb{F}}$ where $\mathcal{R}_{n,m,s,\mathbb{F}}$ is any* C-SAT *relation.*

We sketch the proof of this theorem in Appendix B. In our proofs, we will additionally require the following property from the state function of the batch NP.

**Definition 5.3** (Accepting State)**.** Let State be a state function as in Definition 4.4. We say that State has the accepting state property if for all CRS, all $x$, all partial transcripts $\tau = (\alpha_1, \beta_1, \ldots, \alpha_j)$, and all aux such that $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$, we have that $\mathsf{State}(\mathsf{CRS}, x, \tau \| \beta_j, \mathsf{aux}) = \mathsf{accept}$ for all $\beta_j$.

Note that not all valid state functions will satisfy this additional property. However, if a protocol has a valid state function, it will also have a (possibly different) valid state function that satisfies the accepting state property. We formalize this in the following lemma, which we prove in Appendix C.

**Lemma 5.4.** *Let $\Pi$ be any protocol, and suppose that $\Pi$ is $(B, b, d)$ FS-Compatible with respect to some predicate $\phi$ (Definition 4.5) using the state function* State*. Then there exists a state function* State′ *satisfying the accepting state property (Definition 5.3) such that $\Pi$ is $(B, b, d)$ FS-Compatible with respect to $\phi$ using* State′ *as the state function.*

### 5.2 Bounded-Space Protocol Construction

For any $T \in \mathbb{N}$, consider a language $\mathcal{L}_T$ that contains the set of all strings $(\mathcal{M}, s_0, s_T, y)$ where $\mathcal{M}$ is the description of a Turing machine, $s_0$ is the initial state, $s_T$ is the final state and $y$ is an input such that running $\mathcal{M}$ on $y$ with the start state to be $s_0$ for $T$ time steps results in the final state $s_T$. We construct an interactive $\mathsf{FS}$-compatible argument for the language $\mathcal{L}_T$.

For any $k, \gamma \geq 1$ where $k^\gamma = T$, for every $\ell \in [\gamma]$, we construct an argument for $k^\ell$-time, $S$-space computations in Figure 6 in terms of an interactive argument for $k^{\ell-1}$-time, $S$-space computations.

- Let $\Pi_0 = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ denote a trivial protocol (Figure 5) for unit-time computations where the verifier given a machine $\mathcal{M}$, instance $x$ and states $s_0, s_1$, outputs 1 if $\mathcal{M}(x, s_0)$ transitions to state $s_1$ in one time step. $\mathsf{Setup}(1^\lambda)$ outputs $(\perp, \perp)$.
- Let $\Pi_{k^{\ell-1}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a $\rho$-round public-coin protocol for $(k^{\ell-1})$-time computations with $\nu$-length prover messages whose verifier $\mathsf{V} = (\mathsf{V}_1, \ldots, \mathsf{V}_\rho)$ where $r^{(i)} \leftarrow \mathsf{V}_i(1^\lambda, |x|)$ for $i \in [\rho - 1]$ and $\{0, 1\} \leftarrow \mathsf{V}_\rho(x, \tau)$ for transcript $\tau$.
- Let $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ be an SE commitment satisfying Definition 3.4.
- Let $\Pi_{\mathsf{BNP}}$ be a batch NP protocol for circuit satisfiability satisfying Definition 5.1.

Unit Time Interactive Protocol

$\mathsf{P}$ and $\mathsf{V}$ obtain an instance $x^{(0)}$, which $\mathsf{P}$ wishes to prove is in the language

$$\mathcal{L}^{(0)} \triangleq \Big\{(\mathcal{M}, y, s_0, s_1) \ : \ s_1 \leftarrow \mathcal{M}(y, s_0, 1)\Big\}.$$

That is, $\mathcal{M}$ with initial state $s_0$ reaches state $s_1$ in one time step on input $y$.

1. $\mathsf{P}$ sends dummy message $\alpha$ to $\mathsf{V}$.
2. $\mathsf{V}$ sends dummy message $\beta$ to $\mathsf{P}$.
3. $\mathsf{V}$ computes $s_1' \leftarrow \mathcal{M}(y, s_0)$. $\mathsf{V}$ accepts iff $s_1' = s_1$.

Figure 5: Unit Time Interactive Protocol $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$

### 5.3 Non-trivial predicate for Bounded-Space Protocol

We start with the description of the predicate $\phi$ for the protocol $\Pi_{k^\gamma}$. Let $\Pi_T = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be the protocol defined by Figure 6, where $T = k^\gamma$. The predicate $\phi$ equals $\phi_\gamma$, where $\phi_\ell$ is defined recursively for every $x, \alpha, \mathsf{aux}$ and $\ell \in [\gamma]$.

- $\phi_0(x, \alpha, \mathsf{aux}) = 1 \iff x \notin \mathcal{L}^{(0)}$.
- $\phi_\ell(x, \alpha, \mathsf{aux})$ for $\ell \in [1, \gamma]$: Parse $\mathsf{aux} = (\mathsf{aux}', ek, (i_1, \ldots, i_\ell))$ and $\alpha = ((s_0, \ldots, s_k), C^{(1)})$. Define instances $(x_1', \ldots, x_k')$ as in Figure 6, where $x_j' = (\mathcal{M}, s_{j-1}, s_j, y)$ for $j \in [k]$.

  Set $\phi_\ell(x, \alpha, \mathsf{aux}) = (x_{i_\ell}' \notin \mathcal{L}_{k^{\ell-1}}) \wedge \phi_{\ell-1}(x_{i_\ell}', \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}')$.

Interactive Argument for $k^\ell$-Time $S$-Space Computation

**Common Input:** An instance $x = (\mathcal{M}, s_0, s_T, y)$ of the language $\mathcal{L}_{k^\ell}$.

$\mathsf{Setup}(1^\lambda, i, k)$ **does the following.**

- Parse $i$ as a tuple $(i_1, i_2, \ldots, i_\ell) \in [k]^\ell$.
- Obtain $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathcal{C}.\mathsf{Gen}(1^\lambda, i_\ell, k)$ and $(\mathsf{CRS}', \mathsf{aux}') = \Pi_{k^{\ell-1}}.\mathsf{Setup}(1^\lambda, (i_1, \ldots, i_{\ell-1}), k)$.
- Output $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck}), \mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, \ldots, i_\ell))$.

**Initial Processing.**

- $\mathsf{P}$ send $s = (s_0, \ldots, s_k)$ for initial state $s_0$ and $\{s_j \triangleq \mathcal{M}\left(y, s_0, 1^{T \cdot j/k}\right)\}_{j \in [k]}$, to $\mathsf{V}$.
- $\mathsf{P}, \mathsf{V}$ define $k$ instances $(x'_1, \ldots, x'_k)$ for language $\mathcal{L}_{k^{\ell-1}}$ where $\{x'_j = (\mathcal{M}, s_{j-1}, s_j, y)\}_{j \in [k]}$.

**Emulation Phase.**

- For every $r \in [1, \rho]$, let $\nu$ denote the maximum message size of $\Pi_{k^{\ell-1}}.\mathsf{P}$. $\mathsf{P}$ computes $k$ parallel executions of $\Pi_{k^{\ell-1}}.\mathsf{P}$'s $r^{th}$ round message, $\Pi_{k^{\ell-1}}.\mathsf{P}_r$:

$$\pi^{(r)} = \begin{bmatrix} | & & | \\ \pi^{(r)}[1] & \cdots & \pi^{(r)}[\nu] \\ | & & | \end{bmatrix}$$
$$\triangleq \begin{bmatrix} — & \pi_1^{(r)} \triangleq \Pi_{k^{\ell-1}}.\mathsf{P}_r(\mathsf{CRS}', x'_1, \mathcal{L}_{k^{\ell-1}}, \{\beta^{(1)}, \ldots, \beta^{(r-1)}\}) & — \\ & \cdots & \\ — & \pi_k^{(r)} \triangleq \Pi_{k^{\ell-1}}.\mathsf{P}_r(\mathsf{CRS}', x'_k, \mathcal{L}_{k^{\ell-1}}, \{\beta^{(1)}, \ldots, \beta^{(r-1)}\}) & — \end{bmatrix}.$$

  $\mathsf{P}$ sends $C^{(r)} = (C^{(r)}[1], \ldots, C^{(r)}[\nu])$ where $C^{(r)}[j] \triangleq \mathcal{C}.\mathsf{Com}(\mathsf{ck}, \pi^{(r)}[j])$ for $j \in [\nu]$.
- $\mathsf{V}$ sends $\Pi_{k^{\ell-1}}.\mathsf{V}$'s $r^{th}$ round message computed as $\beta^{(r)} \leftarrow \Pi_{k^{\ell-1}}.\mathsf{V}_r(1^\lambda)$.

**Batch NP Phase.**

- $\mathsf{P}$ and $\mathsf{V}$ define the instances $(x''_1, \ldots, x''_k)$ and $\mathsf{P}$ defines the witnesses $(\omega''_1, \ldots, \omega''_k)$ as: For $j \in [k]$, $x''_j = (x'_j, \{\beta^{(r)}\}_{r \in [\rho]})$, $\omega''_j = \{\pi_j^{(r)}\}_{r \in [\rho]}$.
- Define language $\mathcal{L}'' \triangleq \Big\{(x, \{\beta_r\}_{r \in [\rho]}) : \exists \{\pi_r\}_{r \in [\rho]} \text{ s.t. } \Pi_{k^{\ell-1}}.\mathsf{V}(\mathsf{CRS}', x, \{\pi_r, \beta_r\}_{r \in [\rho]}) = 1\Big\}$.
- $\mathsf{P}$ and $\mathsf{V}$ execute $\Pi_{\mathsf{BNP}}$ on input $\mathsf{ck}$, instances $(x''_1, \ldots, x''_k)$ and witnesses $(\omega''_1, \ldots, \omega''_k)$ where the first round message of $\mathsf{P}$ in $\Pi_{\mathsf{BNP}}$ is ignored and replaced by $\{C^{(r)}\}_{r \in [\rho]}$ as sent in the emulation phase.
- If $\Pi_{\mathsf{BNP}}.\mathsf{V}$ accepts, then $\mathsf{V}$ accepts.

Figure 6: Bounded Space Computation Protocol $\Pi_{k^\ell}$ w.r.t. $\Pi_{\mathsf{BNP}}$ and $\mathcal{C}$

**Theorem 5.5** (Non-trivial predicate)**.** *For every $T = T(\lambda), T' = T'(\lambda)$, assuming the $(T', T)$-index hiding property of SE commitments, $\phi$ is a $(T', T)$-non-trivial predicate for the protocol $\Pi_T$.*

*Proof.* We set $k, \gamma$ such that $k^\gamma = T$ (as above) and prove the non-triviality of predicate $\phi$ by induction on $\ell \in [\gamma]$. We define $\mathcal{A}$ to be an admissible adversary if there exists a polynomial $q'(\cdot)$

such that:

$$\Pr_{\substack{i \leftarrow [k]^\gamma, \\ (\mathsf{CRS},\mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x,\alpha_1) \leftarrow \mathcal{A}(\mathsf{CRS})}} [x \notin \mathcal{L} \wedge x \neq \perp] \geq \frac{1}{q'(T)}.$$

The base case where $\ell = 0$ follows directly from the definition of $\phi_0$. For any $\ell \in [\gamma]$, our induction hypothesis assumes that for every non-uniform $\mathsf{poly}(T')$-time admissible adversary $\mathcal{A}$,

$$\Pr_{\substack{i=(i_1,\ldots,i_{\ell-1}) \leftarrow [k]^\gamma, \\ (\mathsf{CRS},\mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x,\alpha) \leftarrow \mathcal{A}(\mathsf{CRS})}} [\phi_{\ell-1}(x, \alpha, \mathsf{aux}) = 1 | x \notin \mathcal{L}_{k^{\ell-1}} \wedge x \neq \perp] \geq \frac{1}{k^{\ell-1}} - \mathsf{negl}(T) \tag{8}$$

Our inductive step will show that for every non-uniform $\mathsf{poly}(T')$-time admissible adversary $\mathcal{A}$,

$$\Pr_{\substack{i=(i_1,\ldots,i_{\ell}) \leftarrow [k]^\gamma, \\ (\mathsf{CRS},\mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x,\alpha) \leftarrow \mathcal{A}(\mathsf{CRS})}} [\phi_{\ell}(x, \alpha, \mathsf{aux}) = 1 | x \notin \mathcal{L}_{k^{\ell}} \wedge x \neq \perp] \geq \frac{1}{k^{\ell}} - \mathsf{negl}(T) \tag{9}$$

Recall that by definition $\phi_\ell(x, \alpha, \mathsf{aux}) = (x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}) \wedge \phi_{\ell-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}')$. The LHS of Equation (9) can be written as (without explicitly writing the random variables over which the probability is defined):

$$\Pr[x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}} | x \notin \mathcal{L}_{k^\ell} \wedge x \neq \perp] \cdot \Pr[\phi_{\ell-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}') = 1 | x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}]$$

Since $\phi_{\ell-1}$ is non-trivial (by induction hypothesis), we have that

$$\Pr[\phi_{\gamma-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}') = 1 | x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}] \geq \frac{1}{k^{\ell-1}} - \mathsf{negl}(T)$$

To complete the proof, we show that:

$$\Pr[x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}} | x \notin \mathcal{L}_{k^\ell} \wedge x \neq \perp] \geq \frac{1}{k} - \mathsf{negl}(T)$$

Suppose the above probability is at most $1/k - 1/q(T)$ for some polynomial $q(T)$, we give a reduction that breaks the $(\lambda, T)$ index hiding property of the SE commitment.

The reduction interacts with the external challenger and provides a uniform index $i_\ell \leftarrow [k]$ to the challenger. It obtains $\mathsf{ck}$ from the challenger, that is binding at either index $1$ or $i_\ell$. The reduction samples $i_1, \ldots, i_{\ell-1}, i_{\ell+1}, \ldots, i_\gamma$ uniformly from $[k]^{\gamma-1}$, samples the related keys $\mathsf{ck}_j \leftarrow \mathcal{C}.\mathsf{Gen}(1^\lambda, j, k)$ for $j \in \{1, \ldots, \ell-1, \ell+1, \ldots, \gamma\}$, sets $\mathsf{CRS} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_{\ell-1}, \mathsf{ck}, \mathsf{ck}_\ell, \ldots, \mathsf{ck}_\gamma)$ and runs $\mathcal{A}(\mathsf{CRS})$ to obtain $(x, \alpha)$. The reduction checks if $x \in \mathcal{L}_{k^\ell}$ and if it is the case, then it outputs a random bit to the challenger. If $x \notin \mathcal{L}_{k^\ell}$, the reduction outputs $1$ if $x'_{i_\ell} \in \mathcal{L}_{k^{\ell-1}}$ and $0$ otherwise.

Note that when the commitment key $\mathsf{ck}$ is generated as binding at index $1$, then conditioned on $x \notin \mathcal{L}_{k^\ell}$, the probability that $x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}$ is $1/k$ since $i_\ell$ is uniformly distributed with respect to the adversary's view. On the other hand, if $\mathsf{ck}$ is generated as binding at index $i_\ell$ then conditioned on $x \notin \mathcal{L}_{k^\ell}$, the probability that $x'_{j_\ell} \notin \mathcal{L}_{k^{\ell-1}}$ is at most $1/k - 1/q(T)$ (by assumption).

Let $\epsilon$ be the probability that $\mathcal{A}$ outputs $x \in \mathcal{L}_{k^\ell}$. Since $\mathcal{A}$ is admissible, we have that $\epsilon < 1 - 1/q'(T)$ for some polynomial $q'(\cdot)$. Thus, the probability that the reduction outputs $1$ when $\mathsf{ck}$ is generated as binding at index $1$ at least $\epsilon \cdot 1/2 + (1-\epsilon)(1/k)$. On the other hand, if $\mathsf{ck}$ is generated as binding at index $i_\ell$, then the probability that the reduction outputs $1$ is at most $\epsilon \cdot 1/2 + (1-\epsilon)(1/k - 1/q(T))$. Thus, the reduction breaks $(\lambda, T)$-index hiding of SE commitments with advantage $(1-\epsilon)1/q(T) \geq 1/q(T)q'(T)$ (which is a contradiction). $\square$

### 5.4 $\mathsf{FS}$-Compatibility for Bounded-Space Protocol

**Theorem 5.6** (FS-Compatibility w.r.t. Predicate $\phi$)**.** *Let $\mathcal{C}$ be a somewhere extractable commitment (Definition 3.4) with security parameter $\lambda$ whose extraction algorithm* $\mathsf{Extract}$ *has depth $d_{\mathsf{Extract}}$ and size $B_{\mathsf{Extract}}$. Suppose there exist $B_{\mathsf{BNP}}, b_{\mathsf{BNP}}, d_{\mathsf{BNP}}, k$ (all functions of $\lambda$) such that $\Pi_{\mathsf{BNP}}$ is a $k$-mode $(B_{\mathsf{BNP}}, b_{\mathsf{BNP}}, d_{\mathsf{BNP}})$-FS-compatible batch NP argument with respect to $\mathcal{C}$ and $\phi_{\mathsf{BNP}}$.*

*Then for any $T = T(\lambda) \geq \lambda$ and $k = k(\lambda)$, $\Pi$ (Fig. 6) is a $T$-mode $(B, b, d)$-FS-compatible argument (Definition 4.5) with respect to the predicate $\phi$, where*

$$B = \log_k T \cdot B_{\mathsf{Extract}} + B_{\mathsf{BNP}}, \quad b = b_{\mathsf{BNP}}, \quad d = \log_k T \cdot d_{\mathsf{Extract}} + d_{\mathsf{BNP}}.$$

*Furthermore, $\Pi$ has communication complexity and verifier complexity $|\Pi_{k^\gamma}.\mathsf{V}| = (kS + |y|) \cdot (\lambda \cdot \log(kS + |y|))^{O(\gamma)}$ and prover complexity $\mathsf{poly}(k^\gamma)$ for a fixed polynomial $\mathsf{poly}(\cdot)$.*

### 5.5 Proof of $\mathsf{FS}$-Compatibility.

We prove Theorem 5.6 by demonstrating the completeness, round-by-round soundness, and FS-compatibility of $\Pi_{k^\gamma}$ below, after which we compute efficiency. For the rest of this proof, we let $\mathcal{C}$ be a somewhere extractable commitment (Definition 3.4) with security parameter $\lambda$ whose extraction algorithm $\mathsf{Extract}$ has depth $d_{\mathsf{Extract}}$ and size $B_{\mathsf{Extract}}$. We also assume there exist $B_{\mathsf{BNP}}, b_{\mathsf{BNP}}, d_{\mathsf{BNP}}, k$ (all functions of $\lambda$) such that $\Pi_{\mathsf{BNP}}$ is a $k$-mode $(B_{\mathsf{BNP}}, b_{\mathsf{BNP}}, d_{\mathsf{BNP}})$-$\mathsf{FS}$-compatible batch NP argument with respect to $\mathcal{C}$ and $\phi_{\mathsf{BNP}}$.

**Completeness:** It is easy to see that the unit time protocol satisfies perfect completeness, since the verifier accepts iff $s_1 = \mathcal{M}(y, s_0)$. Assume that for any $\ell > 1$ the protocol $\Pi_{k^{\ell-1}}$ satisfies perfect completeness. We will prove that the protocol $\Pi_{k^\ell}$ satisfies perfect completeness. By assumption on the completeness of $\Pi_{k^{\ell-1}}$, at the end of the emulation phase, $\mathsf{P}$ and $\mathsf{V}$ obtain a commitment to $k$ accepting transcripts of $\Pi_{k^{\ell-1}}$. In other words, they obtain instances $x'' = (x''_1, \ldots, x''_k)$ and the prover obtains witnesses $\omega'' = (\omega''_1, \ldots, \omega''_k)$ of the language $\mathcal{L}''$. Then, by completeness of $\Pi_{\mathsf{BNP}}$, we have that $\mathsf{V}_{\mathsf{BNP}}$ accepts with probability 1 at the end of the batch NP phase.

**Round-by-Round Soundness: State Function.** We define the state function $\mathsf{State}_\ell$ in Figures 7 and 8. These are defined in terms of $\mathsf{State}'_{\mathsf{BNP}}$, which is the state function of $\Pi_{\mathsf{BNP}}$ guaranteed by Lemma 5.4, and hence satisfies the accepting state property (Definition 5.3) as well as all the properties of round-by-round soundness (Definition 4.4).

In order to aid us in proving that $\mathsf{State}_\ell$ satisfies our requirements, we will first prove the following two propositions.

Intuitively, the first proposition demonstrates that the batch NP predicate is true (that is, the extracting from the commitment would yield a non-witness for the corresponding batch NP statement) whenever the predicate $\phi_\ell$ is true and the State function of $\Pi_\ell$ rejects. In particular, this will mean that we can use that the batch NP predicate is true when we need to prove that the bad challenge function of $\mathsf{State}_\ell$ is sparse.

**Proposition 5.7.** *Fix any security parameter $\lambda \in \mathbb{Z}^+$, any $\ell \in [\gamma]$ , any indices $i_1, \ldots, i_\ell \in [k]$, any common reference string and auxiliary information $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}_\ell(1^\lambda, (i_1, \ldots, i_\ell)))$, any instance $x$, and any $\tau = (\alpha_1, \beta_1, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1}, \alpha_\zeta)$ for $\zeta > \rho'$, where $\rho'$ denotes the number of rounds for protocol $\Pi_{k^{\ell-1}}$. Parse $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, i_2, \ldots, i_\ell))$ and $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$. Parse*

$\alpha_1 = (s_0, s_1, \ldots, s_k, C^{(1)})$. *Redefine* $\alpha_{\rho'+1} = C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}$ *and define* $(x''_1, \ldots, x''_k)$ *according to Figure 6. If* $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ *for* $\mathsf{State}_\ell$ *defined in Figure 8 and* $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$*, then* $\phi_{\mathsf{BNP}}((x''_1, \ldots, x''_k), \alpha_{\rho'+1}, (i_\ell, \mathsf{ek})) = 1$.

*Proof.* Let $\widetilde{\tau} = (\alpha_1, \beta_1, \ldots, \alpha_{\rho'}, \beta_{\rho'})$, $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$, $\pi^{(\iota)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_\iota)$ for $\iota \in [2, \rho']$, $\widetilde{\tau}' = (\pi^{(1)}, \beta_1, \ldots, \pi^{(\rho')}, \beta_{\rho'})$, and $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$. Let $\omega''_{i_\ell} = (\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(\rho')})$ and $\mathcal{R}''$ be the relation corresponding to $\mathcal{L}''$ defined in Figure 6.

Since $\mathsf{State}_\ell$ runs $\mathsf{State}'_{\mathsf{BNP}}$ on any partial transcript ending in the Batch NP phase (that is for all partial transcripts $\tau^*$ of $\tau$ such that $\tau^* = (\alpha_1, \beta_1, \ldots, \alpha_{\zeta^*})$ for $\zeta^* > \rho'$), we have that $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), (\alpha_{\rho'+1}, \beta_{\rho'+1}, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1}, \alpha_\zeta), (i_\ell, \mathsf{ek})) = \mathsf{reject}$. By the accepting state property of $\mathsf{State}'_{\mathsf{BNP}}$, this implies that $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \alpha_{\rho'+1}, (i_\ell, \mathsf{ek})) = \mathsf{reject}$. Once again, since $\mathsf{State}_\ell$ runs $\mathsf{State}'_{\mathsf{BNP}}$ on any partial transcript ending in the Batch NP phase, we have that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tilde{\tau} \| \alpha_{\rho'+1}, \mathsf{aux}) = \mathsf{reject}$.

By the syntax of state functions, $\mathsf{State}_\ell(\mathsf{CRS}, x, \tilde{\tau} \| \alpha_{\rho'+1}, \mathsf{aux}) = \mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux})$. Hence, $\mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux}) = \mathsf{reject}$. By definition of $\mathsf{State}_\ell$ (Fig. 8) observe that $\mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux}) = \mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \widetilde{\tau}', \mathsf{aux}')$. Hence we must have $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \widetilde{\tau}', \mathsf{aux}') = \mathsf{reject}$. By end functionality, we must have that $\Pi_{k^{\ell-1}}.\mathsf{V}(\mathsf{CRS}', x_{i_\ell}, \widetilde{\tau}') = 0$. By our protocol's invocation of the Batch NP protocol $\Pi_{\mathsf{BNP}}$, we will have that $(x''_{i_\ell}, \omega''_{i_\ell}) \notin \mathcal{R}''$. By definition of our Batch NP predicate $\phi_{\mathsf{BNP}}$, $\phi_{\mathsf{BNP}}((x''_1, \ldots, x''_k), \alpha'_{\rho+1}, (i_\ell, \mathsf{ek})) = 1$. ☐

The next proposition intuitively demonstrates that when the state function for $\Pi_{k^\ell}$ rejects and the predicate for $\Pi_{k^\ell}$ is true, then depending on where the partial transcript ends (i.e. in the emulation phase or batch NP phase), the state function of the underlying protocol $\Pi_{k^{\ell-1}}$ or the batch NP protocol $\Pi_{\mathsf{BNP}}$ must also reject. Note that this holds by definition if our transcript $\tau$ ends with a verifier message, but needs a few steps to prove if $\tau$ ends with a prover message.

**Proposition 5.8.** *Fix any security parameter* $\lambda \in \mathbb{Z}^+$*, any* $\ell \in [\gamma]$*, any indices* $i_1, \ldots, i_\ell \in [k]$*, any common reference string and auxiliary information* $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}_\ell(1^\lambda, (i_1, \ldots, i_\ell)))$*, any instance* $x$*, and any partial transcript* $\tau$ *ending on a prover message for* $\Pi_{k^\ell}$ *such that* $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ *for* $\mathsf{State}_\ell$ *defined in Figure 8 and* $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$*. Let* $\rho'$ *denote the number of rounds for protocol* $\Pi_{k^{\ell-1}}$*. Parse* $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, i_2, \ldots, i_\ell))$ *and* $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$*. Parse* $\alpha_1 = (s_0, s_1, \ldots, s_k, C^{(1)})$*. Then the following is true:*

- *If* $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ *for some* $\zeta \leq \rho'$*, let* $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$ *and* $\pi^{(\iota)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_\iota)$ *for* $\iota \in [2, \zeta]$*. Let* $\tau' = (\pi^{(1)}, \beta_1, \ldots, \pi^{(\zeta)})$ *and* $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$*. Then we have that* $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau', \mathsf{aux}') = \mathsf{reject}$.
- *If* $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ *for some* $\zeta > \rho'$*, redefine* $\alpha_{\rho'+1} = C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}$*. Futhermore, let* $\tau' = (\alpha_{\rho'+1}, \beta_{\rho'+1}, \ldots, \alpha_\zeta)$ *and define* $(x''_1, \ldots, x''_k)$ *according to Figure 6. Then we have that* $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau', (i_\ell, \mathsf{ek})) = \mathsf{reject}$.

*Proof.* Fix any epoch $\ell \in [\gamma]$, any security parameter $\lambda \in \mathbb{Z}^+$, any indices $i_1, \ldots, i_\ell \in [k]$, any common reference string and auxiliary information $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}_\ell(1^\lambda, (i_1, \ldots, i_\ell)))$, any instance $x$, and any partial non-empty transcript $\tau$ ending on a prover message for the $\Pi_{k^\ell}$ protocol such that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ for $\mathsf{State}_\ell$ defined in Figure 8 and $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$. Let $\rho'$ denote the number of rounds for protocol $\Pi_{k^{\ell-1}}$. Parse $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, i_2, \ldots, i_\ell))$ and $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$. Parse $\alpha_1 = (s_0, s_1, \ldots, s_k, C^{(1)})$. We have four cases: either the transcript $\tau$ ends on the first emulation message, or at a different point in the Emulation phase, on the first batch NP message, or at a different message in the Batch NP phase.

- If $\tau = (\alpha_1)$, let $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$. Let $\tau' = (\pi^{(1)})$ and $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$.

  By the definition of $\phi_\ell$, since $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$, we have that $\phi_{\ell-1}(x_{i_\ell}, \pi^{(1)}, \mathsf{aux}') = 1$. By the end functionality property, we have $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau', \mathsf{aux}') = \mathsf{reject}$.

- If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta \leq \rho'$, let $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$ and $\pi^{(\iota)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_\iota)$ for $\iota \in [2, \zeta]$. Let $\tau' = (\pi^{(1)}, \beta_1, \ldots, \pi^{(\zeta)})$ and $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$. Furthermore, let $\widetilde{\tau} = (\alpha_1, \beta_1, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1})$ and $\widetilde{\tau}' = (\pi^{(1)}, \beta_1, \ldots, \beta_{\zeta-1})$.

  By the syntax property of state functions, $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux})$ and $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \widetilde{\tau}' \| \pi^{(\zeta)}, \mathsf{aux}') = \mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \widetilde{\tau}', \mathsf{aux}')$. We refer to the definition of $\mathsf{State}_\ell$ in Figure 8 to see that $\mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux}) = \mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \widetilde{\tau}', \mathsf{aux}')$. Since $\widetilde{\tau}' \| \pi^{(\zeta)} = \tau'$ and $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$, we have that $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau', \mathsf{aux}') = \mathsf{reject}$.

- If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for $\zeta = \rho' + 1$, then redefine $\alpha_{\rho'+1} = C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}$ and let $\tau' = (\alpha_{\rho'+1})$. Define $(x''_1, \ldots, x''_k)$ according to Figure 6. By Proposition 5.7, we have that the predicate $\phi_{\mathsf{BNP}}((x''_1, \ldots, x''_k), \alpha_{\rho'+1}, (i_\ell, \mathsf{ek})) = 1$. By the end functionality property of state functions, we have that $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau', (i_\ell, \mathsf{ek})) = \mathsf{reject}$.

- If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta > \rho' + 1$, let $\alpha_{\rho'+1} = C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}$ and $\tau' = (\alpha_{\rho'+1}, \beta_{\rho'+1}, \ldots, \alpha_\zeta)$. Define $(x''_1, \ldots, x''_k)$ according to Figure 6. Furthermore, let $\widetilde{\tau} = (\alpha_1, \beta_1, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1})$ and $\widetilde{\tau}' = (\alpha_{\rho'+1}, \beta_{\rho'+1}, \ldots, \beta_{\zeta-1})$.

  By the syntax property of state functions, $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux})$ and $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{CRS}', (x''_1, \ldots, x''_k), \widetilde{\tau}' \| \alpha_\zeta, \mathsf{aux}') = \mathsf{State}'_{\mathsf{BNP}}(\mathsf{CRS}', (x''_1, \ldots, x''_k), \widetilde{\tau}', \mathsf{aux}')$. By the definition of $\mathsf{State}_\ell$ (Figure 8), $\mathsf{State}_\ell(\mathsf{CRS}, x, \widetilde{\tau}, \mathsf{aux}) = \mathsf{State}'_{\mathsf{BNP}}(\mathsf{CRS}', (x''_1, \ldots, x''_k), \widetilde{\tau}', \mathsf{aux}')$. Since $\widetilde{\tau}' \| \alpha_\zeta = \tau'$ and $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$, $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{CRS}', (x''_1, \ldots, x''_k), \tau', \mathsf{aux}') = \mathsf{reject}$.

This completes the proof. □

**$b_{\mathsf{BNP}}$-Round-by-round soundness w.r.t. $\phi_\ell$:** We will show that for all $\ell$, $\Pi_{k^\ell}$ is $b_{\mathsf{BNP}}$-round-by-round sound with respect to $\phi_\ell$ (Definition 4.4). First, we show that this is true for $\ell = 0$:

> $\mathsf{State}_0(\mathsf{CRS}, x, \tau, \mathsf{aux})$ for $\Pi_1$:
>
> - Check if $x \in \mathcal{L}^{(0)}$. Output accept if so and reject otherwise.

Figure 7: Unit-Time State function $\mathsf{State}_0$.

**Lemma 5.9.** *$\Pi_1$ is 0-round-by-round sound with respect to $\phi_0$, with $\mathsf{State}_0$ (Figure 7) as the corresponding state function.*

*Proof.* Note that for $\Pi_1$, $\mathsf{State}_0$, $\phi_0$, and $\mathsf{V}_0$ all simply check if $x \in \mathcal{L}^{(0)}$, ignoring the dummy prover and verifier messages. This immediately gives us that $\mathsf{State}_0$ satisfies the syntax and end functionality properties of Definition 4.4. As for sparsity, note that $\mathsf{State}_0$ ignores $\tau$, so there can be no $\mathsf{CRS}$, $x$, $\tau$, $\beta$, and $\mathsf{aux}$ such that $\mathsf{State}_0(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ but $\mathsf{State}_0(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}$. In particular, this means that the probability of sampling a $\beta$ that makes this happen is zero as desired. □

$\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux})$ for $\Pi_{k^\ell}$ (for $\ell > 0$):

- **Notation.** Let $\phi_\ell$ denote the predicate of protocol $\Pi_{k^\ell}$. Let $\mathsf{State}_{\ell-1}$ denote the state function of the $\rho'$-round protocol $\Pi_{k^{\ell-1}}$. $\mathsf{State}'_{\mathsf{BNP}}$ denotes the state function of $\Pi_{\mathsf{BNP}}$ as guaranteed by Lemma 5.4. Parse $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, i_2, \ldots, i_\ell))$ and $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$. Letting $\alpha_1$ be the first prover message in $\tau$, parse $\alpha_1 = (s_0, s_1, \ldots, s_k, C^{(1)})$.
- **(Almost empty transcript).** If $\tau = \alpha_1$, output reject if $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$ and accept otherwise.
- **(Partial Transcript Ending in Prover Message).** If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta > 1$, output $\mathsf{State}_\ell(\mathsf{CRS}, x, (\alpha_1, \beta_1, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1}), \mathsf{aux})$.
- **(Emulation Phase).** If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta, \beta_\zeta)$ for some $\zeta \le \rho'$,
  - Let $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$ and $\pi^{(\iota)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_\iota)$ for $\iota \in [2, \zeta]$.
  - Output $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau', \mathsf{aux}')$ where $\tau' = (\pi^{(1)}, \beta_1, \ldots, \pi^{(\zeta)}, \beta_\zeta)$ and $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$.
- **(Batch NP Phase).** If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta, \beta_\zeta)$ for some $\zeta > \rho'$,
  - Let $\tau' = (C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}, \beta_{\rho'+1}, \ldots, \alpha_\zeta, \beta_\zeta)$ and define $(x''_1, \ldots, x''_k)$ according to Figure 6.
  - Output $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau', (i_\ell, \mathsf{ek}))$.

Figure 8: State function State.

Note that because $0 \le b_{\mathsf{BNP}}(\lambda)$, this lemma in particular tells us that $\Pi_1$ is also $b_{\mathsf{BNP}}$-round-by-round sound. Combining this repeatedly with the following lemma will give us our desired result.

**Lemma 5.10.** *Let $\ell > 0$, and suppose that $\Pi_{k^{\ell-1}}$ is $b_{\mathsf{BNP}}$-round-by-round sound with respect to $\phi_{\ell-1}$ with $\mathsf{State}_{\ell-1}$ (Figure 7 if $\ell = 1$ or 8 otherwise) as the state function. Then $\Pi_{k^\ell}$ is $b_{\mathsf{BNP}}$-round-by-round sound with respect to $\phi_\ell$ with $\mathsf{State}_\ell$ (Figure 8) as the state function, according to Definition 4.4.*

*Proof.* We will prove that $\mathsf{State}_\ell$ as defined satisfies each of the properties from Defintion 4.4 below:

- **Syntax**: This follows directly from the definition of $\mathsf{State}_\ell$. Note that $\mathsf{State}_{\ell-1}$ and $\mathsf{State}'_{\mathsf{BNP}}$ are both deterministic (as they also satisfy the syntax property), which means $\mathsf{State}_\ell$ as defined is also deterministic.
- **End Functionality**: Note that we have defined $\mathsf{State}_\ell(\mathsf{CRS}, x, \alpha_1, \mathsf{aux})$ to be reject if and only if $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$ as required. Additionally, for a complete transcript $\tau$, $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) =$ accept if and only if $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau'', (i_\ell, \mathsf{ek}))$ does. But $\tau''$ will be a complete transcript for $\Pi_{\mathsf{BNP}}$ (as $\tau$ is a complete transcript for $\Pi_{k^\ell}$), so by the round-by-round soundness of $\Pi_{\mathsf{BNP}}$, we have that this happens if and only if $\mathsf{V}_{\mathsf{BNP}}$ accepts. Finally, we note that $\mathsf{V}_\ell$ accepts if and only if $\mathsf{V}_{\mathsf{BNP}}$ does, and hence we have the proper end functionality.

- **Sparsity**: We consider two cases for what $\tau$ could be. First, suppose $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta \le \rho'$, where $\rho'$ is the number of rounds in $\Pi_{k^{\ell-1}}$. Suppose additionally that we are given CRS, $x$, and aux such that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ and $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$. As in Proposition 5.8, we parse $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$, $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, \ldots, i_\ell))$, and $\alpha_1 = (s_0, \ldots, s_k, C^{(1)})$. Furthermore letting $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$, $\pi^{(\iota)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_\iota)$ for $\iota > 1$, $\tau' = (\pi^{(1)}, \beta_1, \ldots, \pi^{(\zeta)})$, and $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$, Proposition 5.8 gives us that $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau', \mathsf{aux}') = \mathsf{reject}$. Combining this with the fact that $\phi_{\ell-1}(x_{i_\ell}, \pi^{(1)}, \mathsf{aux}') = 1$ whenever $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$,[8] we have by the sparsity of $\mathsf{State}_{\ell-1}$ with respect to $\phi_{\ell-1}$ that

  $$\Pr_\beta[\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau' \| \beta, \mathsf{aux}') = 1] \le b_{\mathsf{BNP}}(\lambda) \cdot 2^{-\lambda}$$

  Noting that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux})$ is defined to be $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau' \| \beta, \mathsf{aux})$ immediately gives us that

  $$\Pr_\beta[\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = 1] \le b_{\mathsf{BNP}}(\lambda) \cdot 2^{-\lambda}$$

  as desired.

  The other case to consider is if $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta > \rho'$. As before, suppose that we are given CRS, $x$, and aux such that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ and $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$, and parse $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$, $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, \ldots, i_\ell))$, and $\alpha_1 = (s_0, \ldots, s_k, C^{(1)})$. If we then redefine $\alpha_{\rho'+1} = C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}$, define $\tau' = (\alpha_{\rho'+1}, \beta_{\rho'+1}, \ldots, \alpha_\zeta)$, and define $(x''_1, \ldots, x''_k)$ as in Figure 6, Proposition 5.8 tells us that $\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau', (i_\ell, \mathsf{ek})) = \mathsf{reject}$. Additionally, Proposition 5.7 gives us that $\phi_{\mathsf{BNP}}((x''_1, \ldots, x''_k), \alpha_{\rho'+1}, (i_\ell, \mathsf{ek})) = 1$. Thus, by the sparsity of $\mathsf{State}'_{\mathsf{BNP}}$, we have

  $$\Pr_\beta[\mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau' \| \beta, (i_\ell, \mathsf{ek})) = \mathsf{accept}] \le b_{\mathsf{BNP}}(\lambda) \cdot 2^{-\lambda}$$

  Noting that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau' \| \beta, (i_\ell, \mathsf{ek}))$ by definition, we get

  $$\Pr_\beta[\mathsf{State}_\ell(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}] \le b_{\mathsf{BNP}}(\lambda) \cdot 2^{-\lambda}$$

  as desired.

This completes the proof of the lemma. □

**Bad challenge function** BAD **with respect to** $\phi$**.** We define the bad challenge function $\mathsf{BAD}_{\ell,\cdot}$ in Figures 9 and 10, in terms of $\mathsf{BAD}_{\mathsf{BNP}}$ which is the bad challenge function of $\Pi_{\mathsf{BNP}}$ as guaranteed by Lemma 5.4. In what follows, We will show that for all $\ell$, $\Pi_{k^\ell}$ is $(\gamma d_{\mathsf{Extract}} + d_{\mathsf{BNP}})$-depth $(\gamma B_{\mathsf{Extract}} + B_{\mathsf{BNP}})$-efficient BAD with respect to $\phi_\ell$ (Definition 4.4).
First, we show that $\Pi_{k^0}$ has a unit depth BAD function with a unit-sized circuit, with respect to $\phi_0$.

**Lemma 5.11** (Base case)**.** *$\Pi_1$ has a unit-depth* BAD *function with unit-sized circuits with respect to $\phi_0$. Furthermore, $\Pi_1$ has bad challenge function* $\mathsf{BAD}_{0,\perp}$ *as defined in Figure 9.*

*Proof.* As defined in Figure 5, the verifier accepts iff $x \in \mathcal{L}^{(0)}$. As such, there will be no bad challenges. Since the function $\mathsf{BAD}_{0,\perp}$ simply outputs $\perp$, the circuit representing this will have depth 1 and size 1. □

[8] This holds because $\phi_{\ell-1}(x_{i_\ell}, \pi^{(1)}, \mathsf{aux}') = 1$ is one of the conditions needed in the definition of $\phi_\ell$ in order for $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$.

$\mathsf{BAD}_{0,\mathsf{aux}}(\mathsf{CRS}, x, \tau)$ for $\Pi_1$ (Figure 5):

- **Notation.** Parse $\mathsf{aux} = \bot$, $\tau = (\alpha_1)$, and $\mathsf{CRS} = \bot$.
- Output $\bot$.

Figure 9: Bad challenge function BAD.

$\mathsf{BAD}_{\ell,\mathsf{aux}}(\mathsf{CRS}, x, \tau)$ for $\Pi_{k^\ell}$ (where $\ell > 0$):

- **Notation.** Let $\rho'$ denote the number of rounds for protocol $\Pi_{k^{\ell-1}}$. Parse $\mathsf{aux} = (\mathsf{aux}', \mathsf{ek}, (i_1, i_2, \ldots, i_\ell))$ and $\mathsf{CRS} = (\mathsf{CRS}', \mathsf{ck})$. Letting $\alpha_1$ be the first prover message in $\tau$, parse $\alpha_1 = (s_0, s_1, \ldots, s_k, C^{(1)})$.
- **(Emulation Phase).** If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta \leq \rho'$,
  - Let $\pi^{(1)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)})$ and $\pi^{(\iota)} \leftarrow \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, \alpha_\iota)$ for $\iota \in [2, \zeta]$.
  - Output $\mathsf{BAD}_{\ell-1,\mathsf{aux}'}(\mathsf{CRS}', x_{i_\ell}, \tau')$ where $\tau' = (\pi^{(1)}, \beta_1, \ldots, \pi^{(\zeta)})$ and $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell - 1}, s_{i_\ell})$.
- **(Batch NP Phase).** If $\tau = (\alpha_1, \beta_1, \ldots, \alpha_\zeta)$ for some $\zeta > \rho'$,
  - Let $\alpha'_{\rho'+1} = C^{(1)} \| \alpha_2 \| \ldots \| \alpha_{\rho'}$ and $\tau' = (\alpha'_{\rho'+1}, \beta_{\rho'+1}, \ldots, \alpha_\zeta)$. Define $(x''_1, \ldots, x''_k)$ according to Figure 6.
  - Output $\mathsf{BAD}_{\mathsf{BNP},\mathsf{ek}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau')$ where $\mathsf{BAD}_{\mathsf{BNP},\mathsf{ek}}$ denotes the bad challenge function of $\Pi_{\mathsf{BNP}}$ hardwired with $\mathsf{ek}$.

Figure 10: Bad challenge function BAD.

We will now show that for all $\ell \in [\gamma]$, $\Pi_{k^\ell}$ is $(\gamma d_{\mathsf{Extract}} + d_{\mathsf{BNP}})$-depth $(\gamma B_{\mathsf{Extract}} + B_{\mathsf{BNP}})$-efficient BAD with respect to $\phi_\ell$ (Definition 4.4) through use of Propositions 5.7 and 5.8.

**Lemma 5.12** (Induction step). *For any $\ell \in [\gamma]$, suppose that $\Pi_{k^{\ell-1}}$ is $d_{\ell-1}$-depth $B_{\ell-1}$-efficient* BAD *with respect to $\phi_{\ell-1}$ where the bad challenge function $\mathsf{BAD}_{\ell-1,\cdot}$ is defined in Figure 10. Then $\mathsf{BAD}_{\ell,\cdot}$ as defined in Figure 10 is a bad challenge function for $\Pi_{k^\ell}$ w.r.t. $\phi_\ell$. Moreover, on input a partial transcript $\tau$ that ends in the emulation phase $\mathsf{BAD}_{\ell,\cdot}$ can be computed by a circuit of size $(B_{\mathsf{Extract}} + B_{\ell-1})$ and $(d_{\mathsf{Extract}} + d_{\ell-1})$ depth; and on input a partial transcript $\tau$ that ends in the emulation phase $\mathsf{BAD}_{\ell,\cdot}$ can be computed by a circuit of size $B_{\mathsf{BNP}}$ and depth $d_{\mathsf{BNP}}$.*

*Proof.* For any $\ell \in [\gamma]$, we will prove that $\mathsf{BAD}_{\ell,\cdot}$ is the bad challenge for $\Pi_{k^\ell}$ satisfying the properties in the lemma. Note that we follow notation from $\mathsf{BAD}_{\ell,\cdot}$ and define $\mathsf{Setup}_\ell$ to be $\Pi_{k^\ell}.\mathsf{Setup}$ (see Figure 6).

- **Syntax**: This follows from the definition of $\mathsf{BAD}_{\ell,\cdot}$ for $\ell \in [0, \gamma]$ (Figures 9 and 10). Fix any $\ell \in [0, \gamma]$. We hardwire $\mathsf{BAD}_{\ell,\cdot}$ with aux from $\mathsf{Setup}_\ell$, the setup algorithm for the $\Pi_{k^\ell}$ protocol. We provide inputs CRS from $\mathsf{Setup}_\ell$, the instance $x$ for $\Pi_{k^\ell}$, a partial transcript $\tau$ for $\Pi_{k^\ell}$.

- $\mathsf{BAD}_{\ell,\cdot}$ **with respect to** $\phi_\ell$: We will first prove that Figure 10 is the correct bad challenge function with respect to $\phi_\ell$. Fix any epoch $\ell \in [\gamma]$, security parameter $\lambda \in \mathbb{Z}^+$, any indices $i_1, \ldots, i_\ell \in [k]$, any common reference string and auxiliary information $(\mathsf{CRS}, \mathsf{aux}) \in \mathsf{Support}(\mathsf{Setup}_\ell(1^\lambda, (i_1, \ldots, i_\ell)))$, any instance $x$, and any partial non-empty transcript $\tau$ for the $\Pi_{k^\ell}$ protocol such that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ and $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$,

  Let $\rho'$ denote the number of rounds in protocol $\Pi_{k^{\ell-1}}$. The set of bad challenges for $\Pi_{k^\ell}$ is the set $\mathcal{B}_{\ell,\mathsf{CRS},\mathsf{aux}}$ defined as

  $$\mathcal{B}_{\ell,\mathsf{CRS},\mathsf{aux}} := \{\beta \ : \ \mathsf{State}_\ell(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}\}.$$

  We have two cases depending on whether the transcript $\tau$ ends in the Emulation phase or in the Batch NP phase.

  - Suppose $\tau = (\alpha_1, \beta_1, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1}, \alpha_\zeta)$ where $\zeta \leq \rho'$.
    Through fixing the instance $x$ and transcript $\tau$, we have fixed an instance $x_{i_\ell} = (\mathcal{M}, y, s_{i_\ell-1}, s_{i_\ell})$. By the correctness of extraction and the structure of our protocol $\Pi_{k^\ell}$ (Figure 6), $\mathsf{BAD}_{\ell,\mathsf{aux}}$ correctly extracts $\tau'$ which is the partial transcript for the $i_\ell^{th}$ parallel run of $\Pi_{k^{\ell-1}}$[9]. By Proposition 5.8, we have that $\mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau', \mathsf{aux}') = \mathsf{reject}$. By definition of our predicate $\phi_\ell$, $\phi_\ell(x, \alpha_1, \mathsf{aux}) = 1$ implies $\phi_{\ell-1}(x_{i_\ell}, \pi^{(1)}, \mathsf{aux}') = 1$.
    By definition of $\mathsf{BAD}_{\ell,\mathsf{aux}}$ (Figure 10), $\mathsf{BAD}_{\ell,\mathsf{aux}}(\mathsf{CRS}, x, \tau) = \mathsf{BAD}_{\ell-1,\mathsf{aux}'}(\mathsf{CRS}', x_{i_\ell}, \tau')$. The assumption in our lemma (Lemma 5.12) asserts that $\mathsf{BAD}_{\ell-1,\mathsf{aux}'}(\mathsf{CRS}', x_{i_\ell}, \tau')$ enumerates the set $\mathcal{B}_{\ell-1,\mathsf{CRS}',\mathsf{aux}'}$ defined as

    $$\mathcal{B}_{\ell-1,\mathsf{CRS}',\mathsf{aux}'} := \{\beta \ : \ \mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau' \| \beta, \mathsf{aux}') = \mathsf{accept}\}.$$

    By definition of $\mathsf{State}_\ell$ (Fig. 8), $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{State}_{\ell-1}(\mathsf{CRS}', x_{i_\ell}, \tau' \| \beta, \mathsf{aux}')$. Hence, we have that $\mathcal{B}_{\ell,\mathsf{CRS},\mathsf{aux}} = \mathcal{B}_{\ell-1,\mathsf{CRS}',\mathsf{aux}'}$. Thus, $\mathsf{BAD}_{\ell,\mathsf{aux}}(\mathsf{CRS}, x, \tau)$ enumerates the set $\mathcal{B}_{\ell,\mathsf{CRS},\mathsf{aux}}$.
  - Suppose $\tau = (\alpha_1, \beta_1, \ldots, \alpha_{\zeta-1}, \beta_{\zeta-1}, \alpha_\zeta)$ where $\zeta > \rho'$.
    By construction, $(\mathsf{ck}, \mathsf{ek}) \in \mathsf{Support}(\mathsf{Setup}_{\mathsf{BNP}}(1^\lambda, i_\ell))$. Fixing an instance $x$ and transcript $\tau$ fixes the Batch NP instance $(x''_1, \ldots, x''_k)$ (Figure 6). Therefore, $\mathsf{BAD}_{\ell,\mathsf{aux}}$ correctly computes $\tau'$ as the partial transcript for the Batch NP protocol $\Pi_{\mathsf{BNP}}$. By Proposition 5.8, $\mathsf{State}_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau', (i_\ell, \mathsf{ek})) = \mathsf{reject}$. Thus, applying Proposition 5.7, $\phi_{\mathsf{BNP}}((x''_1, \ldots, x''_k), \alpha'_{\rho'+1}, (i_\ell, \mathsf{ek})) = 1$.
    By definition of $\mathsf{BAD}_{\ell,\mathsf{aux}}$ (Figure 10), we have that $\mathsf{BAD}_{\ell,\mathsf{aux}}(\mathsf{CRS}, x, \tau) = \mathsf{BAD}_{\mathsf{BNP},\mathsf{ek}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau')$. We rely on the properties of $\mathsf{BAD}_{\mathsf{BNP},\cdot}$ (see Lemma 5.4) asserting that $\mathsf{BAD}_{\mathsf{BNP},\mathsf{ek}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau')$ enumerates the set $\mathcal{B}_{\mathsf{BNP},\mathsf{ck},\mathsf{ek}}$ defined as

    $$\mathcal{B}_{\mathsf{BNP},\mathsf{ck},\mathsf{ek}} := \{\beta \ : \ \mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau' \| \beta, (i_\ell, \mathsf{ek})) = \mathsf{accept}\}.$$

    By definition of $\mathsf{State}_\ell$ (Fig. 8), we have that $\mathsf{State}_\ell(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{State}'_{\mathsf{BNP}}(\mathsf{ck}, (x''_1, \ldots, x''_k), \tau' \| \beta, (i_\ell, \mathsf{ek}))$. Hence, we have that $\mathcal{B}_{\ell,\mathsf{CRS},\mathsf{aux}} = \mathcal{B}_{\mathsf{BNP},\mathsf{ck},\mathsf{ek}}$. As such, we have that $\mathsf{BAD}_{\ell,\mathsf{aux}}(\mathsf{CRS}, x, \tau)$ enumerates the set $\mathcal{B}_{\ell,\mathsf{CRS},\mathsf{aux}}$.

[9] Here we assume that the commitment scheme is perfectly extractable. Note that this can be relaxed to statistical extractability, where only $1 - \mathsf{negl}(\lambda)$ fraction of extraction keys satisfy perfect correctness, since it suffices to ignore any CRS that contains an extraction key that does not satisfy perfect correctness.

- **Low depth, Efficient** BAD: We will now prove that Figure 10 defines a low depth and efficient function. Note that by the structure of $\mathsf{BAD}_{\ell,\cdot}$ either we are in the emulation phase or in the batch NP phase.
  - In the emulation phase, $\mathsf{BAD}_{\ell,\cdot}$ sequentially runs one run of the $\mathcal{C}.\mathsf{Extract}$ extraction algorithm (we parallelize the separate extractions for the whole transcript) and one run of the $\mathsf{BAD}_{\ell-1,\cdot}$ algorithm. By our assumption in the theorem, we have that $\mathcal{C}$ has a threshold circuit which runs Extract in depth $d_{\mathsf{Extract}}$ and size $B_{\mathsf{Extract}}$. By our assumption in the lemma, we have that $\mathsf{BAD}_{\ell-1,\cdot}$ has a threshold circuit which runs in depth $d_{\ell-1}$ and size $B_{\ell-1}$. As such, in this branch, we have depth $d_{\mathsf{Extract}} + d_{\ell-1}$ and size $B_{\mathsf{Extract}} + B_{\ell-1}$.
  - In the batch NP phase, $\mathsf{BAD}_{\ell,\cdot}$ sequentially runs one run of the $\mathsf{BAD}'_{\mathsf{BNP},\cdot}$ algorithm. By our assumption in the theorem, we have that $\mathsf{BAD}'_{\mathsf{BNP},\cdot}$ has a threshold circuit which runs in depth $d_{\mathsf{BNP}}$ and size $B_{\mathsf{BNP}}$. As such, in this branch, we have depth $d_{\mathsf{BNP}}$ and size $B_{\mathsf{BNP}}$.

This completes the proof of the lemma. □

By Lemma 5.11 and Lemma 5.12, we have that $\Pi_{k^\gamma}$ is $(\gamma d_{\mathsf{Extract}} + d_{\mathsf{BNP}})$-depth $(\gamma B_{\mathsf{Extract}} + B_{\mathsf{BNP}})$-efficient BAD with respect to $\phi = \phi_\gamma$.

### 5.6 Complexity of $\Pi_{k^\gamma}$

**Prover Runtime.** Fix any $\ell \in [2, \gamma]$. The prover's running time in the emulation phase of $\Pi_{k^\ell}$ equals $k$ times the prover's running time in the protocol $\Pi_{k^{\ell-1}}$, and the time required to commit to each round of prover messages in $\Pi_{k^{\ell-1}}$. The latter of these terms is included in the prover's running time in the batch NP phase of $\Pi_{k^\ell}$ (Definition 5.1) since our protocol ignores and replaces the Batch NP first message, $\alpha_{\rho'+1}$, with the prover's emulation phase messages (See Figure 6).

As such, we now account for the prover's running time in the Batch NP phase of $\Pi_{k^\ell}$ which comes to $(k \cdot |\Pi_{k^{\ell-1}}.\mathsf{V}|)^c$ (Definition 5.1) for a fixed constant $c > 0$ where $|\Pi_{k^{\ell-1}}.\mathsf{V}|$ denotes the size of the verification circuit in $\Pi_{k^{\ell-1}}$. Therefore, the running time of the prover for $\Pi_{k^\ell}$ is $k \cdot T_{\mathsf{P},\ell-1} + (k \cdot |\Pi_{k^{\ell-1}}.\mathsf{V}|)^c$ where $T_{\mathsf{P},\ell-1}$ denotes the running time of the prover for $\Pi_{k^{\ell-1}}$. Unrolling the recursion, we get

$$|\Pi_{k^\ell}| \leq k^{c'\gamma} = T^{c'}$$

for some constant $c' > 0$.

**Verifier Runtime.** Note that the verifier sends uniform challenges during the emulation phase, and only performs verification at the end of the batch NP phase. Therefore, the verifier's computation costs are subsumed by those at the end of the batch NP phase. From Definition 5.1 , there exists a constant $c > 0$ such that the circuit size of the verifier in the Batch-NP phase of $\Pi_{k^\ell}$ for any $\ell \in [2, \gamma]$ is at most $\widetilde{O}(\sum_{i\in[k]} |x''_i| + |\Pi_{k^{\ell-1}}.\mathsf{V}|) \cdot \lambda^c$ (where $\widetilde{O}$ hides multiplicative $\log$ factors, $|\Pi_{k^{\ell-1}}.\mathsf{V}|$ denotes the size of verification circuit in $\Pi_{k^{\ell-1}}$, and $x''_1, \ldots x''_k$ are defined in Figure 6) . We note that $\sum_{i\in[k]} |x''_i|$ is the sum of sizes of the $k$ instances where we don't repeat common factors, this is upper bounded by $kS + |y| + |\Pi_{k^{\ell-1}}.\mathsf{V}|$ where we additionally upperbound the size of the verifier's messages by $|\Pi_{k^{\ell-1}}.\mathsf{V}|$. Thus, the cost of the Batch-NP phase is $\widetilde{O}(kS + |y| + |\Pi_{k^{\ell-1}}.\mathsf{V}|) \cdot \lambda^c$. Moreover, the verification circuit for $\Pi_1$ has size $O(S)$.

Unrolling the recursion, we get

$$|\Pi_{k^\gamma}.\mathsf{V}| = (kS + |y|) \cdot (\lambda \cdot \log(kS + |y|))^{O(\gamma)}$$

where the constant in the exponent is a function of $\gamma$.

**Communication complexity.** The communication complexity is upper bounded by verifier runtime, and is therefore at most

$$(kS + |y|) \cdot (\lambda \cdot \log(kS + |y|))^{O(\gamma)}.$$

**Corollary 5.13.** *Assuming the subexponential hardness of QR, for any time-$T$ space-$S$ deterministic computation, there is a $T$-mode $(B, b, d)$-**FS**-compatible argument (Definition 4.5) w.r.t. a $(\lambda, T)$ non-trivial predicate $\phi$, where each verifier message is of size $\lambda$, and where verifier runtime and communication complexity are bounded by $\left(T^{\frac{c}{\sqrt{\log\log\log T}}} \cdot (S + n)\right)$, $c$ is a constant $> 0$, $n$ denotes the size of the input and $\lambda = T^{\frac{1}{\log\log\log T}}$, where $B = T^{\frac{c}{\sqrt{\log\log\log T}}}$, $b = \mathsf{poly}(\lambda)$, $d = O(\sqrt{\log\log\log T})$.*

*Proof.* We set $\lambda$ such that $\lambda^\gamma = k$, this implies that $T = k^\gamma = \lambda^{\gamma^2}$, and thus $\log T = \gamma^2 \log \lambda$. Then we set $\gamma = \sqrt{\log\log\log T}$. This implies that $\lambda = T^{\frac{1}{\gamma^2}} = T^{\frac{1}{\log\log\log T}}$.

Note that $\log(kS + |y|) \leq 2 \log T$ (because $T \geq k, T \geq S, T \geq |y|$). Because $\lambda = T^{\frac{1}{\log\log\log T}}$, we have $\log T < \log^2 \lambda$. Substituting, this implies that there is a constant $c > 0$ such that

$$|\Pi_T.\mathsf{V}| \leq (kS + |y|) \cdot (k^c)$$

which implies that there is a constant $c > 0$ such that

$$|\Pi_T.\mathsf{V}| \leq T^{\frac{c}{\sqrt{\log\log\log T}}} \cdot (S + |y|)$$

Finally, we note that $\lambda = T^{\frac{1}{\log\log\log T}}$, which completes our proof. $\square$

The following Corollary follows from Corollary 5.13, Theorem 4.7 and Theorem 3.3.

**Corollary 5.14.** *Assuming the subexponential hardness of QR and subexponential hardness of DDH, there exists a* SNARG *for any time-$T$ space-$S$ deterministic computation with verifier runtime and communication complexity $\left(T^{\frac{c}{\sqrt{\log\log\log T}}} \cdot (S + n)\right)$ and prover runtime $\mathsf{poly}(T, S)$, where $n$ denotes the size of the input, and $c$ is a constant $> 0$.*

# 6 FS-compatible Arguments for Non-Deterministic Bounded Space

We now describe our interactive arguments for $\mathsf{NTISP}(T(n), S(n))$, which is the class of all languages recognizable by non-deterministic Turing Machines in time $T(n)$ and space $S(n)$. Such a Turing Machine allows each step of the computation to non-deterministically transition to a new state. Thus, in a sense, this corresponds to the setting where each bit of the witness is read at most once[10]. An alternative way to describe this class is as the class of languages $L$ with a corresponding witness relation $R_L$, recognizable by a deterministic Turing Machines with access to an input

[10] If a non-deterministic Turing Machine wishes to remember what non-deterministic choices it made, it has to write them down to its work tape.

tape and a read-only, read-once witness tape, in addition to a work tape where only $O(S(n))$ space is used, and that runs in $O(T(n))$ time.

First, we introduce some notation and provide some background on NTISP computations. The following subsection closely mirrors [BKK+18].

### 6.1 Background.

Fix any $L \in \mathsf{NTISP}(T(n), S(n))$. Denote by $\mathcal{R}_L$ its corresponding $\mathsf{NP}$ relation, and denote by $M = M_L$ a $T(n)$-time $S(n)$-space (non-deterministic) Turing machine for deciding $L$. $M$ can alternately be defined as a two-input Turing machine, that takes as input a pair $(x, w)$ and outputs $1$ if and only if $(x, w) \in \mathcal{R}_L$.

**Corresponding Layered Circuit $C^M_{n,m}$.** Any such Turing machine $M$ can be converted into a layered circuit, denoted by $C^M_{n,m}$, which takes as input a pair $(x, w)$, where $n = |x|$ and $|w| = m = m(n)$ (where $m(n)$ is an upper bound on the length of a witness corresponding to a length $n$ instance), and outputs $1$ if and only if $M(x, w) = 1$.

Moreover, $C^M_{n,m}$ is a layered circuit, with $W = O(S(n))$ denoting the maximum of the number of gates and number of wires in each layer, and depth $D = O(T(n))$, such that an incoming wire to a gate in layer $i + 1$ is either an input wire (i.e. a wire that reads the input), a witness wire (i.e. a wire that is attached to a trivial witness gate with fan-in and fan-out $1$ whose output equals its input, and whose input wire reads the witness), or the output wire of a gate in layer $i$. Moreover, any witness gate has fan-out $1$ (this corresponds to read-once access to the witness tape), and any layer of the circuit reads at most one (unique) bit from the witness tape. In addition, there is a deterministic Turing machine $M'$ of space $O(\log T)$ that on input $n$ outputs the (description of the) circuit $C^M_{n,m}$.

**Notation for $C^M_{n,m}$.** We introduce some detailed notation for the wires of $C^M_{n,m}$.

We call all wires that are inputs to gates in layer $i$, the wires for layer $i$. The set of wires for layer $i$ is denoted by $q^i$, and a set of assignments to these wires is denoted by $s^i$. The $j^{th}$ wire (from the left) in layer $i$ is denoted by $q^i_j$ and an assignment to this wire is denoted by $s^i_j$.

We partition wires for layer $i$ into 3 sets, denoted by $\mathsf{Instance}^i, \mathsf{Witness}^i, \mathsf{Intermediate}^i$, where $\mathsf{Instance}^i$ is the set of all wires for layer $i$ that read the instance $x$, $\mathsf{Witness}^i$ is the set of all wires for layer $i$ that read the witness $w$, and $\mathsf{Intermediate}^i$ is the set of remaining wires for layer $i$ which are output wires of gates in layer $(i - 1)$. We will denote by $s^D = C^M_{n,m}(x, \omega, s^1, D)$ the set of assignments to intermediate and input wires at depth $D$, when $s^1$ denotes the assignments to intermediate and input wires in the first layer of $C$.

Because any witness gate has fan-out $1$, and any layer of the circuit reads at most one bit from the witness tape, we can further partition the witness into substrings, each of which are read by witness wires in different layers of the circuit.

To help us in our analysis, we will now define the sets $\mathsf{Acc}^i_x$ for all layers $i$ of $C^M_{n,m}$.

**Defining the set $\mathsf{Acc}^i_x$.** We will now recursively define the sets of *allowed* wire values for which $C^M_{n,m}$ outputs $1$, corresponding to $M$ accepting $x$. Gate $D$ has a single output wire, so $\mathsf{Acc}^{D+1}_x = 1$. For any layer $i \in \{D, D-1, \ldots, 1\}$, we define the set $\mathsf{Acc}^i_x$ recursively, as follows:

- $\mathsf{Acc}_x^D$ is the set of all possible assignments $\{s_j^D\}_{j:(q_j^D \in \mathsf{Intermediate}^D)}$ to intermediate wires, such that when the assignment $\{s_j^D\}_{j:(q_j^D \in \mathsf{Instance}^D)}$ to the wires in $\mathsf{Instance}^D$ are set consistently with $x$, there exists an assignment to the witness wires $\{s_j^D\}_{j:(q_j^D \in \mathsf{Witness}^D)}$ such that for this assignment,
$$\delta_D(s^D = \{a_1^D, a_2^D, \dots a_{2W}^D\}) = 1,$$
where $\delta_D$ denotes the transition function corresponding to the set of gates at the $D^{th}$ layer. In other words, we require that the output of the $D^{th}$ layer (and hence the output of the circuit) is 1.

- For $i \in [D-1]$, $\mathsf{Acc}_x^i$ is defined as the set of all possible assignments $\{s_j^i\}_{j:(q_j^i \in \mathsf{Intermediate}^i)}$ to intermediate wires such that when the assignment $\{s_j^i\}_{j:(q_j^i \in \mathsf{Instance}^i)}$ to wires in $\mathsf{Instance}^i$ are set consistently with $x$, there exists an assignment to the witness wires $\{s_j^i\}_{j:(q_j^i \in \mathsf{Witness}^i)}$ such that for this assignment,
$$\delta_i(s^i = \{a_1^i, a_2^i, \dots a_{2W}^i\}) \in \mathsf{Acc}_x^{i+1}$$
where $\delta_i$ denotes the transition function corresponding to the set of gates at the $i^{th}$ layer.

Next, we prove the following claim.

**Claim 6.1.** *There exists a deterministic (inefficient) Turing Machine $\mathcal{Y}$ that takes as input a triplet $(M, x, s, k)$, runs in time $T(|x|) \cdot 2^{O(S(|x|))}$ and decides whether $s \in \mathsf{Acc}_x^k$ for $k \in [D(|x|)]$.*

*Proof.* The Turing Machine $\mathcal{Y}$ on input $(x, s, k)$ does the following: By backward induction, starting from $i = D(|x|)$ until $i = k$, it computes a table consisting of all intermediate wires in $\mathsf{Acc}_x^i$. Note that given the table of all intermediate wires in $\mathsf{Acc}_x^{i+1}$ it takes roughly $2^{O(W)}$ time to compute the table of all intermediate wires in $\mathsf{Acc}_x^i$. Thus, it takes roughly $D(|x|) \cdot 2^{O(W(|x|))} = T(|x|) \cdot 2^{O(S(|x|))}$ trials to compute all such tables. Once $\mathcal{Y}$ computes the table of all intermediate wires in $\mathsf{Acc}_x^k$, all that remains is to check whether $s$ belongs to this table, which can be done in time bounded by the size of this table. The Turing Machine $\mathcal{Y}$ is formally described in Figure 11. $\square$

### 6.2 Interactive Arguments for Bounded Space Non-Deterministic Computation.

For any $\ell \geq 1$, we construct an interactive argument that proves correctness of wire assignments to layered circuits $C_{n,m}$ where $n = |x|$ and $m = |w|$ that are of the form described above (i.e. corresponding to computations of a Turing Machine $M$). We will assume that $C_{n,m}$ has depth $D = k^\ell$ and width $W$, and describe an interactive argument in terms of an interactive argument for $k^{\ell-1}$-depth, $W$-width circuits. We will prove in subsequent sections that this protocol is FS-compatible.

- Let $\Pi_0 = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ denote a trivial protocol for unit-depth circuits where the prover sends a dummy message followed by a dummy verifier message, and the verifier given a circuit $C_{n,1}$ with a single layer, instance $x$ s.t. $|x| = n$ and states $s_0, s_1$, outputs 1 iff $s_1 = C_{n,1}(x, 0, s_0, 1)$ or $s_1 = C_{n,1}(x, 1, s_0, 1)$. $\mathsf{Setup}(1^\lambda)$ outputs $(\bot, \bot)$.

- Let $\Pi_{k^{\ell-1}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a $\rho$-round public-coin protocol for $(k^{\ell-1})$-time computations with $\ell$-length prover messages whose verifier $\mathsf{V} = (\mathsf{V}_1, \dots, \mathsf{V}_\rho)$ where $r^{(i)} \leftarrow \mathsf{V}_i(1^\lambda, |x|)$ for $i \in [\rho - 1]$ and $\{0,1\} \leftarrow \mathsf{V}_\rho(x, \tau)$.

**Description of Turing Machine $\mathcal{Y}$**

1. Obtain inputs $(M, x, s, k)$.
2. Set $\mathsf{Acc}_x^{D+1} = \{1\}$.
3. Set $i = D$.
4. While $i \geq k$, compute $\mathsf{Acc}_x^i$ as follows:
   - List all possible assignments to intermediate wires $\{a_j^i\}_{j:q_j^i \in \mathsf{Intermediate}^i}$ for gates in layer $i$, such that:
     When instance wires $\{a_j^i\}_{j:(q_j^i \in \mathsf{Instance}^i)}$ are set consistently with $x$, there exists an assignment to the witness wires $\{a_j^i\}_{j:(q_j^i \in \mathsf{Witness}^i)}$ such that for this assignment, $\{a_j^{i+1}\}_{j:(q_j^{i+1} \in \mathsf{Intermediate}^{i+1})} \in \mathsf{Acc}_x^{i+1}$, where $\delta_i(s^i = \{a_1^i, a_2^i, \ldots a_{2W}^i\}) = \{a_j^{i+1}\}_{j:(q_j^{i+1} \in \mathsf{Intermediate}^{i+1})}$.
   - Set $i = i - 1$ and repeat Step 3.
5. Output $1$ if $s \in \mathsf{Acc}_x^k$, else output $0$.

Figure 11: Description of Turing Machine $\mathcal{Y}$.

- Let $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ be an SE commitment satisfying Definition 3.4.
- Let $\Pi_{\mathsf{BNP}}$ be a batch NP protocol for circuit satisfiability.

For any $D \in \mathbb{N}$, consider language $\mathcal{L}_D$ defined by the NP relation $\mathcal{R}_{\mathcal{L}_D}$ where $\mathcal{R}_{\mathcal{L}_D}(x, w) = 1$ iff $x = (M', n, s^n, s^D, y)$ where $M'$ outputs a description of a circuit $C$ such that $s^n$ is a set of wire assignments to the intermediate and input wires in the $n^{th}$ layer of the circuit, $y$ and $w$ respectively define assignments to all input and witness wires in the circuit, and $s^D$ is a set of consistent wire assignments to intermediate and input wires of the circuit at layer $D + n$.

Our argument is identical to the one in Figure 6, except for the following (syntactic) changes to the inputs of both players, and to the initial processing.

**Inputs.** The common input for $\mathsf{P}$ and $\mathsf{V}$ is an instance $x = (M', s_0, s_T, y)$ of the language $\mathcal{L}_{k^\ell}$. $\mathsf{P}$ also obtains a witness $\omega$, such that $(x, \omega) \in \mathcal{R}_{\mathcal{L}_D}$.

**Initial Processing.**

- $\mathsf{P}$ sends $s = (s_0, \ldots, s_k)$ for $\{s_i \triangleq C\,(y, \omega, s_0, iD/k)\}_{i \in [k]}$ to $\mathsf{V}$.
- $\mathsf{P}$ and $\mathsf{V}$ define instances $(x_1, \ldots, x_k)$ where $\{x_i \triangleq (M', \frac{(i-1)D}{k}, s_{i-1}, s_i, y)\}_{i \in [k]}$ of the language $\mathcal{L}_{k^{\ell-1}}$.

- P partitions $\omega$ into witnesses $\omega_1, \ldots, \omega_k$ where for all $i \in [k]$, $\omega_i$ is used to generate assignments to witness wires in layers $\frac{(i-1)D}{k}$ through $\frac{iD}{k}$.

### 6.3 Non-Trivial Predicate

Let $\Pi_T = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be the protocol defined by Figure 6, where $D = k^\gamma$, and with modifications from the previous section.

The predicate $\phi$ equals $\phi_\gamma$, where $\phi_\ell$ is defined recursively for every $x, \alpha, \mathsf{aux}$ and $\ell \in [\gamma]$.

- $\phi_0(x, \alpha, \mathsf{aux}) = 1 \iff x \notin \mathcal{L}^{(0)}$.
- $\phi_\ell(x, \alpha, \mathsf{aux})$ for $\ell \in [1, \gamma]$: Parse $\mathsf{aux} = (\mathsf{aux}', ek, (i_1, \ldots, i_\ell))$ and $\alpha = ((s_0, \ldots, s_k), C^{(1)})$. Define instances $(x'_1, \ldots, x'_k)$ as in Figure 6, where $x'_j = (\mathcal{M}, s_{j-1}, s_j, y)$ for $j \in [k]$. Set $\phi_\ell(x, \alpha, \mathsf{aux}) = (x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}) \wedge \phi_{\ell-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}')$.

**Theorem 6.2** (Non-trivial predicate)**.** *Assuming the $(T \cdot 2^S, T)$-index hiding property of SE commitments, $\phi$ is a $(T \cdot 2^S, T)$-non-trivial predicate for the protocol $\Pi_T$ where $T = k^\gamma$.*

*Proof.* We prove the non-triviality of predicate $\phi_\gamma$ using induction on $\ell \in [\gamma]$. We define $\mathcal{A}$ to be an admissible adversary if there exists a polynomial $q(\cdot)$ such that:

$$\Pr_{\substack{i \leftarrow [k]^\gamma, \\ (\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x, \alpha_1) \leftarrow \mathcal{A}(\mathsf{CRS})}} [x \notin \mathcal{L} \wedge x \neq \bot] \geq \frac{1}{q(\lambda)}.$$

The base case where $\ell = 0$ follows directly from the definition of $\phi_0$. For any $\ell \in [\gamma]$, our induction hypothesis assumes that for every non-uniform $\mathsf{poly}(T \cdot 2^S)$-time admissible adversary $\mathcal{A}$,

$$\Pr_{\substack{i = (i_1, \ldots, i_{\ell-1}) \leftarrow [k]^{\ell-1}, \\ (\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x, \alpha) \leftarrow \mathcal{A}(\mathsf{CRS})}} [\phi_{\ell-1}(x, \alpha, \mathsf{aux}) = 1 | x \notin \mathcal{L}_{k^{\ell-1}} \wedge x \neq \bot] \geq \frac{1}{k^{\ell-1}} - \mathsf{negl}(T) \tag{10}$$

Our inductive step will show that for every non-uniform $\mathsf{poly}(T \cdot 2^S)$-time admissible adversary $\mathcal{A}$,

$$\Pr_{\substack{i = (i_1, \ldots, i_\ell) \leftarrow [k]^\ell, \\ (\mathsf{CRS}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, i), \\ (x, \alpha) \leftarrow \mathcal{A}(\mathsf{CRS})}} [\phi_\ell(x, \alpha, \mathsf{aux}) = 1 | x \notin \mathcal{L}_{k^\ell} \wedge x \neq \bot] \geq \frac{1}{k^\ell} - \mathsf{negl}(T) \tag{11}$$

Recall that by definition $\phi_\ell(x, \alpha, \mathsf{aux}) = (x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}) \wedge \phi_{\ell-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}')$. The LHS of Equation (11) can be written as (without explicitly writing the random variables over which the probability is defined):

$$\Pr[x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}} | x \notin \mathcal{L}_{k^\ell} \wedge x \neq \bot] \cdot \Pr[\phi_{\ell-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}') = 1 | x'_{i_\ell} \notin \mathcal{L}_{k^{\gamma-1}}]$$

Since $\phi_{\ell-1}$ is non-trivial (by induction hypothesis), we have that

$$\Pr[\phi_{\gamma-1}(x'_{i_\ell}, \mathcal{C}.\mathsf{Extract}(\mathsf{ek}, C^{(1)}), \mathsf{aux}') = 1 | x_{i_\ell} \notin \mathcal{L}_{k^{\gamma-1}}] \geq \frac{1}{k^{\ell-1}} - \mathsf{negl}(T)$$

To complete the proof, we show that:

$$\Pr[x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}} | x \notin \mathcal{L}_{k^\ell} \wedge x \neq \bot] \geq \frac{1}{k} - \mathsf{negl}(T)$$

Suppose this the above probability is at most $1/k - 1/q(T)$ for some polynomial $q(\lambda)$, we give a reduction that breaks the $(T \cdot 2^S, T)$ index hiding property of the SE commitment.

The reduction interacts with the external challenger and provides a uniform index $i_\ell \leftarrow [k]$ to the challenger. It obtains $\mathsf{ck}$ from the challenger, that is binding at either index $1$ or $i_\ell$. The reduction samples $i_1, ..., i_{\ell-1}$ uniformly in $[k]^{\ell-1}$, samples $(\mathsf{CRS}', \mathsf{aux}') \leftarrow \Pi_{k^{\ell-1}}.\mathsf{Setup}(1^\lambda, (i_1, \ldots, i_{\ell-1}), k)$, sets $\mathsf{CRS} = (\mathsf{CRS}', ck)$ and runs $\mathcal{A}(\mathsf{CRS})$ to obtain $(x, \alpha)$. The reduction checks if $x \in \mathcal{L}_{k^\ell}$ and if it is the case, then it outputs a random bit to the challenger. If $x \notin \mathcal{L}_{k^\ell}$, the reduction outputs $1$ if $x'_{i_\ell} \in \mathcal{L}_{k^{\ell-1}}$ and $0$ otherwise.

Note that when the commitment key $\mathsf{ck}$ is generated as binding at index $1$, then conditioned on $x \notin \mathcal{L}_{k^\ell}$, the probability that $x'_{i_\ell} \notin \mathcal{L}_{k^{\ell-1}}$ is $1/k$ since $i_\ell$ is uniformly distributed with respect to the adversary's view. On the other hand, if $\mathsf{ck}$ is generated as binding at index $i_\ell$ then conditioned on $x \notin \mathcal{L}_{k^\ell}$, the probability that $x'_{j_\ell} \notin \mathcal{L}_{k^{\ell-1}}$ is at most $1/k - 1/q(\lambda)$ (by assumption).

Let $\epsilon$ be the probability that $\mathcal{A}$ outputs $x \in \mathcal{L}_{k^\ell}$. Since $\mathcal{A}$ is admissible, we have that $\epsilon < 1 - 1/q'(\lambda)$ for some polynomial $q'(\lambda)$. Thus, the probability that the reduction outputs $1$ when $\mathsf{ck}$ is generated as binding at index $1$ at least $\epsilon \cdot 1/2 + (1-\epsilon)(1/k)$. On the other hand, if $\mathsf{ck}$ is generated as binding at index $i_\ell$, then the probability that the reduction outputs $1$ is at most $\epsilon \cdot 1/2 + (1-\epsilon)(1/k - 1/q(T))$.

Finally, by Claim 6.1, the reduction runs in time $\mathsf{poly}(T \cdot 2^S)$. Thus the reduction breaks $(T \cdot 2^S, T)$-index hiding of SE commitments with advantage $(1-\epsilon)1/q(T) \geq 1/q(T)q'(\lambda)$ (which is a contradiction). □

### 6.4 FS-Compatibility w.r.t. Predicate $\Phi$

**Theorem 6.3** (FS-Compatibility w.r.t. Predicate $\phi$)**.** *Let $\mathcal{C}$ be a somewhere extractable commitment (Definition 3.4) whose extraction circuit has depth $d_{\mathsf{Extract}}$ and size $B_{\mathsf{Extract}}$. Let $\Pi_{\mathsf{BNP}}$ be a $k$-mode $\rho$-round $(B_{\mathsf{BNP}}, b_{\mathsf{BNP}}, d_{\mathsf{BNP}})$-$\mathsf{FS}$-compatible batch NP with respect to $\mathcal{C}$ (Definition 5.1), for some $B_{\mathsf{BNP}}, b_{\mathsf{BNP}}, d_{\mathsf{BNP}}, k$ (all functions of $\lambda$).*

*Then for any $T = T(\lambda) \geq \lambda$ and $k = k(\lambda)$, $\Pi$ defined above is a $T$-mode $(B, b, d)$-$\mathsf{FS}$-compatible argument (Definition 4.5) with respect to the predicate $\phi$ defined above, where*

$$B = \log_k T \cdot B_{\mathsf{Extract}} + B_{\mathsf{BNP}}, \quad b = b_{\mathsf{BNP}}, \quad d = \log_k T \cdot d_{\mathsf{Extract}} + d_{\mathsf{BNP}}.$$

*Furthermore, $\Pi$ has communication complexity and verifier runtime $|\Pi_{k^\gamma}.\mathsf{V}| = (kS+|y|)\cdot(\lambda\cdot\log(kS+|y|))^{O(\gamma)}$ and prover runtime $\mathsf{poly}(T)$ given the witness, for a fixed polynomial $\mathsf{poly}(\cdot)$.*

The proof of this theorem follows identically to that of Theorem 5.6. In particular, the proofs of completeness, round-by-round soundness, FS-compatibility and efficiency follow identically to that of Theorem 5.6. We obtain the following corollaries of this theorem.

**Corollary 6.4.** *Assuming the $(T\cdot 2^S, T)$-hardness of QR, for any time-$T$ space-$S$ non-deterministic computation, there is a $T$-mode $(B, b, d)$-$\mathsf{FS}$-compatible argument (Definition 4.5) w.r.t. a $(T \cdot 2^S, T)$ non-trivial predicate $\phi$, where each verifier message is of size $\lambda$ (which also denotes the security parameter), and where verifier runtime and communication complexity are bounded by $T^{\frac{c}{\sqrt{\log\log\log T}}} \cdot (S+|y|)$, $c$ is a constant $> 0$,*

$|y|$ *denotes the size of the input and where* $\lambda = T^{\frac{1}{\log\log\log T}}$*, where* $B = T^{\frac{c}{\sqrt{\log\log\log T}}}, b = \mathsf{poly}(\lambda), d = O(\sqrt{\log\log\log T})$.

*Proof.* We set $\lambda$ such that $\lambda^\gamma = k$, this implies that $T = \lambda^{\gamma^2}$, and $\log T = \gamma^2 \log \lambda$. We also set $\gamma^2 = \log\log\log T$, This implies that $\lambda = T^{\frac{1}{\gamma^2}} = T^{\frac{1}{\log\log\log T}}$, and $\log T < \log^2 \lambda$. Substituting, this implies that there is a constant $c > 0$ such that

$$|\Pi_T.\mathsf{V}| \le (kS + |y|) \cdot (k^c)$$

which implies that there is a constant $c > 0$ such that

$$|\Pi_T.\mathsf{V}| \le T^{\frac{c}{\sqrt{\log\log\log T}}} \cdot (S + |y|)$$

Finally, we note that $\lambda = T^{\frac{1}{\log\log\log T}}$, which completes our proof. □

The following Corollary follows from Corollary 6.4, Theorem 4.7 and Theorem 3.3, where we set the security parameter $\lambda = \max(S^{1/\epsilon}, T^{\frac{1}{\log\log\log T}})$, where $\epsilon$ is the smaller of the subexponential parameters for QR/DDH hardness. Then, $(2^{\lambda^\epsilon}, 2^{\lambda^\epsilon})$-hardness of QR implies the conditions of the corollary above. In addition, $(2^{\lambda^\epsilon}, 2^{\lambda^\epsilon})$-hardness of DDH implies the conditions of Theorem 3.3.

**Corollary 6.5.** *Assuming the subexponential hardness of QR and subexponential hardness of DDH, there exists a* $\mathsf{SNARG}$ *for any time-*$T$ *space-*$S$ *non-deterministic computation with verifier runtime and communication complexity* $T^{\frac{c}{\sqrt{\log\log\log T}}} \cdot (S + n)$ *and prover runtime* $\mathsf{poly}(T, S)$ *given the witness, where* $n$ *denotes the size of the input, and c is a constant* $> 0$.

We also obtain the following corollary about improved SNARGs for Batch $\mathsf{NTISP}$, which follows from the observation that if $L \in \mathsf{NTISP}(T, S)$, then $L^{\otimes k} \in \mathsf{NTISP}(kT, S)$, where $L^{\otimes k}$ is the language containing $k$ instances of $L$. This is because we can verify the $k$ different instances by verifying each one individually in time $T$, and reusing the same workspace for every instance.

**Corollary 6.6.** *For every* $L \in \mathsf{NTISP}(T, S)$ *and every* $k \ge S$*, assuming the sub-exponential hardness of QR and DDH, there exists a SNARG for* $L^{\otimes k}$ *where verifier runtime and communication complexity are bounded by* $(kT)^{\frac{c}{\sqrt{\log\log\log kT}}} \cdot (S + n)$ *and prover runtime is* $\mathsf{poly}(k, T, S)$ *given the NP witnesses where* $n$ *denotes the size of the claimed (potentially succinctly described) instance of* $L^{\otimes k}$*, and* $c > 0$ *is a constant.*

## 6.5 SNARGs for $\mathsf{P}$ and beyond

Given the SNARG for batch-$\mathsf{NTISP}$ above, we can use methods from [KVZ21] to build a SNARG for any language decidable in deterministic time $T$ (and in fact, any language that has a no-signaling PCP, just as in [KVZ21]). We instantiate what is essentially their approach with different parameters, specifically, while they obtain $\mathsf{polylog}(T)$ overhead from sub-exponential LWE, we obtain $T^{o(1)}$ overhead from sub-exponential DDH and QR. Thus, we obtain the following Corollary.

**Corollary 6.7** (cf Corollary 6.6 in [KVZ21])**.** *Let* $\mathcal{L}$ *be a language and* $T = T(n)$ *be a function such that* $\mathsf{poly}(n) \le T(n) \le \exp(n)$ *and* $\mathcal{L} \in \mathsf{DTIME}(T)$*. Then assuming the subexponential hardness of QR and DDH, there exists a SNARG for* $\mathcal{L}$ *with prover time* $\mathsf{poly}(T)$*, verifier time* $n \cdot \mathsf{poly}\left(T^{\frac{1}{\sqrt{\log\log\log T}}}\right)$*, and communication complexity* $n \cdot \mathsf{poly}\left(T^{\frac{1}{\sqrt{\log\log\log T}}}\right)$.

A proof of this corollary, including a discussion of the [KVZ21] approach, is presented in Appendix D for completeness.

## 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 *TCC*, pages 3–30, 2016.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001.

[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*, volume 11892 of *Lecture Notes in Computer Science*, pages 522–551. Springer, 2019.

[BCC+14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *IACR Cryptology ePrint Archive*, 2014:580, 2014.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BFJ+20] Saikrishna Badrinarayanan, Rex Fernando, Aayush Jain, Dakshita Khurana, and Amit Sahai. Statistical ZAP arguments. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT*, volume 12107 of *Lecture Notes in Computer Science*, pages 642–667. Springer, 2020.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. *IACR Cryptology ePrint Archive*, 2015:356, 2015.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, pages 474–482, 2017.

[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*, volume 12111, pages 97–123. Springer, 2020.

[BKK+18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In *STOC*, pages 709–721, 2018.

[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*, volume 12172 of *Lecture Notes in Computer Science*, pages 738–767. Springer, 2020.

[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, *STOC*, pages 671–684. ACM, 2018.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[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 *ITCS*, pages 179–190. ACM, 2016.

[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, *STOC*, pages 1082–1090. ACM, 2019.

[CGH04a] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CGH04b] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In *ITCS*, pages 169–178. ACM, 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In *STOC*, pages 429–437. ACM, 2015.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. *IACR Cryptol. ePrint Arch.*, 2021:807, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for $P$ from LWE. *IACR Cryptol. ePrint Arch.*, page 808, 2021.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 54–74, 2012.

[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, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2019.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[GJJM20] Vipul Goyal, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Statistical zaps and new oblivious transfer protocols. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, May 10-14, 2020, Proceedings, Part III*, volume 12107, pages 668–699. Springer, 2020.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–, 2003.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27, 2015.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[GZ21] Alonso González and Alexandros Zacharakis. Fully-succinct publicly verifiable delegation from constant-size assumptions. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 529–557. Springer, 2021.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 163–172. ACM, 2015.

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from subexponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2021.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 708–721. ACM, 2021.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 723–732. ACM, 1992.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, pages 419–428. ACM, 2015.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 91–118, 2016.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1115–1124. ACM, 2019.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 565–574, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *STOC*, pages 485–494. ACM, 2014.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and snargs. Cryptology ePrint Archive, Report 2021/788, 2021. https://ia.cr/2021/788.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, pages 169–189, 2012.

[LVW20] Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Statistical ZAPR arguments from bilinear maps. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 620–641. Springer, 2020.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 436–453, 1994. Full version in [Mic00].

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *TCC*, pages 283–315, 2017.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 422–439, 2012.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 1996.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 89–114. Springer, 2019.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.

# A Somewhere Extractable Commitments with Non-trivial Local Openings

First, we provide a construction of somewhere-extractable commitments from the DDH assumption.

**Theorem A.1** (SE Commitments from DDH)**.** *Fix any $T_1 = T_1(\lambda) \geq \mathsf{poly}(\lambda)$ and $T_2 = T_2(\lambda) \geq \mathsf{poly}(\lambda)$. Assuming $(T_1, T_2)$ hardness of DDH, there exists an SE commitment satisfying Definition 3.4 which satisfies $(T_1, T_2)$ index hiding. Furthermore, this satisfies the following properties: $\ell_{\mathsf{com}} = \ell_{\mathsf{blk}}\lambda$, $\ell_{\mathsf{open}} = \ell_{\mathsf{blk}}L$, $|\mathsf{ck}| = \ell_{\mathsf{blk}}L\lambda$, $|\mathsf{ek}| = \ell_{\mathsf{blk}}\lambda$, the running time of* Gen *and* Verify *is $\ell_{\mathsf{blk}}L\lambda$, and the running time of* Extract *is $\ell_{\mathsf{blk}}\mathsf{poly}(\lambda)$.*

*Proof.* We prove this theorem by providing an instantiation of SE commitments over the binary alphabet from the DDH assumption (which closely follows the construction of trapdoor hash in [DGI+19]). It is easy to observe that this extends to an arbitrary alphabet of size $2^{\ell_{\mathsf{blk}}}$ by separately committing to each bit of the input.

Let $\mathbb{G}$ be a cyclic group of order $p$ with generator $g$. Let us assume that the DDH assumption holds in $\mathbb{G}$. Let us consider the alphabet $\Sigma = \{0, 1\}$.

- $\mathsf{Gen}(1^\lambda, 1^L, 1, k)$: It does the following:
  1. For each $i \in L$ and $b \in \{0, 1\}$, sample a random element $g_{i,0}$ from $\mathbb{G}$.
  2. Set $H_1 = \begin{pmatrix} g_{1,0} & \cdots & g_{L,0} \\ g_{1,1} & \cdots & g_{L,1} \end{pmatrix}$.
  3. Sample a random $s \leftarrow \mathbb{Z}_p$.
  4. Set $H_2 = \begin{pmatrix} g_{1,0}^s & \cdots & g_{i,0}^s & \cdots & g_{L,0}^s \\ g_{1,1}^s & \cdots & g \cdot g_{i,1}^s & \cdots & g_{L,1}^s \end{pmatrix}$.
  5. Output $\mathsf{ck} = (H_1, H_2)$ and $\mathsf{ek} = s$.
- $\mathsf{Com}(\mathsf{ck}, x)$: This algorithm does the following:
  1. Parses $x$ as $(x_1, \ldots, x_L)$.
  2. Parses $H_1$ as $\begin{pmatrix} h_{1,0}^1 & \cdots & h_{L,0}^1 \\ h_{1,1}^1 & \cdots & h_{L,1}^1 \end{pmatrix}$ and $H_2$ as $\begin{pmatrix} h_{1,0}^2 & \cdots & h_{L,0}^2 \\ h_{1,1}^2 & \cdots & h_{L,1}^2 \end{pmatrix}$.
  3. It computes $h_1 = \prod_{j \in [L]} h_{j,x_j}^1$ and $h_2 = \prod_{j \in [L]} h_{j,x_j}^2$.

4. It outputs $(h_1, h_2)$.

- $\mathsf{Open}(\mathsf{ck}, x)$: It outputs $x \in \{0,1\}^L$.
- $\mathsf{Verify}(\mathsf{ck}, y, x)$: It checks if $\mathsf{Com}(\mathsf{ck}, x) = y$ and outputs 1 only in this case.
- $\mathsf{Extract}(\mathsf{ek}, y)$: It does the following:
  1. It parses $y$ as $(h_1, h_2)$ and $\mathsf{ek}$ as $s$.
  2. If $h_1^s = h_2$ then it outputs 0. If $h_1^s \cdot g = h_2$ then it outputs 1. Otherwise, it outputs $\perp$.

The correctness is easy to verify.

**Index Hiding.** Fix any $i \in [L]$. The distribution of $hk$ which is output of $\mathsf{Gen}(1^\lambda, 1^L, i)$ is given by:

$$H_1 = \begin{pmatrix} g_{1,0} & \cdots & g_{L,0} \\ g_{1,1} & \cdots & g_{L,1} \end{pmatrix}$$

and

$$H_2 = \begin{pmatrix} g_{1,0}^s & \cdots & g_{i,0}^s & \cdots & g_{L,0}^s \\ g_{1,1}^s & \cdots & g \cdot g_{i,1}^s & \cdots & g_{L,1}^s \end{pmatrix}$$

From the DDH assumption, the joint distribution of $(H_1, H_2)$ is computationally indistinguishable to $(H_1, H_2')$ where

$$H_2 = \begin{pmatrix} g_{1,0}^s & \cdots & g_{i,0}^s & \cdots & g_{L,0}^s \\ g_{1,1}^s & \cdots & g_{i,1}^s & \cdots & g_{L,1}^s \end{pmatrix}$$

and thus, index hiding follows.

**Somewhere Extractability.** If Verify outputs 1, then $h_1 = \prod_{j \in [L]} h^1_{j,x_j}$ and $h_2 = \prod_{j \in [L]} h^2_{j,x_j}$. It now follows from the construction that if $x_i = 0$, then $h_1^s = h_2$ and if $x_1 = 1$, then $g \cdot h_1^s = h_2$ and thus, somewhere extractability holds. $\square$

**Somewhere Extractable Commitments with Non-trivial Local Openings.** We now describe a compiler that given any somewhere-extractable (SE) commitment (including ones from DDH and QR described in this paper) outputs an SE commitment satisfying a non-trivial local opening property. This compiler simply generates a Merkle tree of arity $k$ and depth $d$ (i.e. $k^d = L$), and in this tree a parent node is an SE commitment to its $k$ children nodes.

**Theorem A.2.** *Assume the existence of an SE commitment satisfying Definition 3.4, with $(T_1, T_2)$ index hiding, and where: $\ell_{\mathsf{com}} = \lambda \ell_{\mathsf{blk}}, \ell_{\mathsf{open}} = L\ell_{\mathsf{blk}}, |\mathsf{ck}| = L\lambda\ell_{\mathsf{blk}}, |\mathsf{ek}| = \lambda\ell_{\mathsf{blk}}$, the running time of* $\mathsf{Gen}$ *and* $\mathsf{Verify}$ *is $L\lambda\ell_{\mathsf{blk}}$, and the running time of* $\mathsf{Extract}$ *is* $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$.

*Then there exists a constant $c > 0$ such that for any $L$ and any $d \le \log L$ there exists an SE commitment satisfying Definition 3.5 which satisfies $(T_1, T_2)$ index hiding as long as $T_2 \ge \lambda^d$, and where: $\ell_{\mathsf{blk}} = 1, \ell_{\mathsf{com}} = \lambda^d, \ell_{\mathsf{open}} = d \cdot L^{\frac{1}{d}} \lambda^{d+1}, |\mathsf{ck}| = d \cdot L^{\frac{1}{d}} \cdot (\lambda)^{d+1}$, $|\mathsf{ek}| \le (\lambda)^{d+1}$, the running time of* $\mathsf{SELO.KeyGen}, \mathsf{SELO.Extract}, \mathsf{SELO.Open}, \mathsf{SELO.Verify}$ *is $d \cdot L^{\frac{1}{d}} \lambda^{d+1+c}$ and the running time of* $\mathsf{SELO.Com}$ *is* $\mathsf{poly}(L)$.

*Proof.* Let $\mathcal{C} = (\mathcal{C}.\mathsf{KeyGen}, \mathcal{C}.\mathsf{Com}, \mathcal{C}.\mathsf{Open}, \mathcal{C}.\mathsf{Verify}, \mathcal{C}.\mathsf{Extract})$ denote an SE-commitment scheme without local openings. Let $d = \log_k L$. An SE commitment with non-trivial local openings SELO can be obtained from this commitment as follows:

- $\mathsf{SELO.KeyGen}(1^\lambda, L, 1, i)$ : Let $i_1, \ldots, i_d$ denote the $k$-ary representation of the index $i$, i.e. $i_1 = 1 + (i \mod k), i_2 = 1 + ((i/k) \mod k), \ldots, i_j = 1 + ((i/k^{j-1}) \mod k), \ldots, i_d = 1 + ((i/k^{d-1}) \mod k)$.

  For every $\iota \in [d]$, obtain $\{(\mathsf{ck}^{(\iota)}, \mathsf{ek}^{(\iota)}) \leftarrow \mathcal{C}.\mathsf{KeyGen}(1^\lambda, k, \lambda^{\iota-1}, i_\iota)\}$. Output $\mathsf{ck} = \{\mathsf{ck}^{(\iota)}\}_{\iota \in [d]}$ as the commitment key and $\mathsf{ek} = \{\mathsf{ek}^{(\iota)}\}_{\iota \in [d]}$ as the extraction key.

- $\mathsf{SELO.Com}(\mathsf{ck}, x)$ : Let $x_1^{(1)}, \ldots, x_L^{(1)}$ denote each bit of $x$. For every $\iota \in [2, d]$, (recursively) compute the nodes of a $k$-ary tree with leaves $x_1^{(1)}, \ldots, x_L^{(1)}$, as

  $$x_j^{(\iota)} = \mathcal{C}.\mathsf{Com}\left(\mathsf{ck}^{(\iota)}, (x^{(\iota-1)}_{(j-1)k+1}, x^{(\iota-1)}_{(j-1)k+2}, \ldots, x^{(\iota-1)}_{(j-1)k+k})\right), \text{ for } j \in [L/k^{\iota-1}].$$

  The output of the commitment is the root $x_1^{(d)}$ of the $k$-ary tree. Note that block length increases by a factor of $\lambda$ at every level of the tree.

- $\mathsf{SELO.Open}(\mathsf{ck}, x, i)$ : Output all nodes along the path from the root to the $i^{th}$ leaf. In addition, output the $(k-1)$ siblings of all nodes along this path.

- $\mathsf{SELO.Verify}(\mathsf{ck}, y, i, u, \pi)$ : Output $1$ if and only if every parent node in the opening corresponds to a commitment to its child nodes using the appropriate commitment key, and $y$ (i.e. the commitment) matches the root of the tree.

- $\mathsf{SELO.Extract}(ek, y)$ : Set $y^{(d)} = y$. Then for each $\iota \in \{d, d-1, \ldots, 1\}$, compute $y^{(\iota-1)}$ as $\mathcal{C}.\mathsf{Extract}(\{\mathsf{ek}_j^{(\iota)}\}_{j \in [\lambda], \iota \in [d]}, y^\iota)$. That is, recursively extract $d$ times to obtain the committed bit.

Then, for $\ell_{\mathsf{blk}} = 1$, we obtain $\ell_{\mathsf{com}} = \lambda^d$ (since the tree contains $d$ layers of commitments, with every commitment in each layer a $\lambda$ multiplicative factor larger than every commitment in the previous one – the root is a single commitment of size $\lambda^d$). The length of the commitment keys is bounded by $dk\lambda^{d+1}$, and the size of the extraction key is bounded by $d\lambda^{d+1}$. The size of local openings is bounded by $dk\lambda^{d+1}$. In addition, SELO.KeyGen, SELO.Extract, SELO.Open and SELO.Verify run in time bounded by $(d \cdot L^{\frac{1}{d}} \lambda^{d+1+c})$, and that of SELO.Com is $\mathsf{poly}(L)$. □

# B Proof Sketch of (Imported) Theorem 5.2

In this section we prove that the Batch NP protocol of [CJJ21a] is a FS-compatible Batch NP for R1CS with respect to a somewhere extractable commitment scheme $\mathcal{C}$ and batch NP predicate $\phi_{\mathsf{BNP}}$ defined above Definition 5.1. We accomplish this by noting that [CJJ21a] prove that their Batch NP protocol has *strong Fiat-Shamir compatibility* which they define as round-by-round soundness, efficient BAD challenge function, and low-depth BAD challenge function.

Our definition of $b$-round-by-round soundness w.r.t. $\phi_{\mathsf{BNP}}$ requires a slight modification of their proof. We modify the behavior of their $\mathsf{State}_{\mathsf{BNP}}$ function to not operate on empty transcripts and to operate on almost empty transcripts, transcripts with a single message from the prover, as follows: If $\tau = \alpha_1$, output reject if $\phi_{\mathsf{BNP}}((x_1, \ldots, x_k), \alpha_1, \mathsf{aux}) = 1$ and accept otherwise. This does not modify proofs of syntax. With this modification, the first condition of end functionality will be achieved. Since we did not modify the behavior of $\mathsf{State}_{\mathsf{BNP}}$ on the complete transcript, the second condition of end functionality will be achieved from [CJJ21a]'s proof.

The proof of sparsity requires more explanation. Since our modification changes the $\mathsf{State}_{\mathsf{BNP}}$ function's behavior on the first message, we need to explain the sparsity on the transition from the prover's first message to the verifier's first message. Everything after this point will follow from [CJJ21a]'s proof. What we need to prove, is that if $\phi_{\mathsf{BNP}}((x_1,\ldots,x_k),\alpha_1,\mathsf{aux}) = 1$ and $\mathsf{State}_{\mathsf{BNP}}(\mathsf{CRS},(x_1,\ldots,x_k),\alpha_1,\mathsf{aux}) = \mathsf{reject}$, the likelihood that $\mathsf{State}(\mathsf{CRS},(x_1,\ldots,x_k),\alpha_1|\beta,\mathsf{aux}) = \mathsf{accept}$ of $b(\lambda)/2^\lambda$ over choices of next verifier message $\beta$.

**Lemma B.1** ([CJJ21a, Set20])**.** $\forall z \in \{0,1\}^s$, $\widetilde{F}_{io}(z) = 0$ *if and only if* $\mathcal{R}_{\mathsf{R1CS}}(x,w) = 1$.

By definition of $\phi_{\mathsf{BNP}}$ for R1CS, we have that $\phi_{\mathsf{BNP}}((x_1,\ldots,x_k),\alpha_1,\mathsf{aux}) = 1$ implies that for $\mathsf{aux} = (i,\mathsf{ek})$, that $\mathcal{R}_{\mathsf{R1CS}}(x_i,\mathcal{C}.\mathsf{Extract}(\mathsf{ek},\alpha_1)) \neq 1$. We make use of Lemma B.1 to have that $\exists z \in \{0,1\}^s$ s.t. $\widetilde{F}_{io}(z) \neq 0$.

**Lemma B.2** ([CJJ21a, Set20])**.**

$$\Pr_{\tau\leftarrow\mathbb{F}^2}\left[Q_{io}(\tau) = 0 \middle| \exists x \in \{0,1\}^s s.t. \widetilde{F}_{io}(x) \neq 0\right] \leq \frac{s}{|\mathbb{F}|}$$

Given that $\exists z \in \{0,1\}^s$ s.t. $\widetilde{F}_{io}(z) \neq 0$, we want to know if the verifier's next message $\beta$ will cause the $\mathsf{State}_{\mathsf{BNP}}$ function to accept. By the state function's definition, this happens if $Q(\beta) = 0$. Hence we make use of Lemma B.2 to say that the likelihood this event happens will be $s/|\mathbb{F}|$.

It follows from their proofs that their protocol satisfies the property of $d$-depth $B$-efficient BAD w.r.t. $\phi_{\mathsf{BNP}}$. The main remaining differences is that [CJJ21a] makes use of promise languages $\mathcal{L} = (\mathcal{L}_{YES},\mathcal{L}_{NO})$ where as we make use of a predicate $\phi_{\mathsf{BNP}}$. Hence whenever they use $(x_1,\ldots,x_k) \in \mathcal{L}_{NO}$, we translate to $\phi_{\mathsf{BNP}}(x,\alpha_1,\mathsf{aux}) = 1$.

We now show that a minor modification of the protocol will yield a $\mathsf{FS}$-compatible Batch NP for C-SAT. The modification is that the prover first sends a commitment to the C-SAT witness. The verifier sends a dummy challenge. Then the prover sends a commitment to the remainder of the R1CS witness. The verifier will send the first verifier challenge and the protocol continues from here. The properties for $\mathsf{FS}$-compatible Batch NP for C-SAT reduce cleanly to the corresponding properties for $\mathsf{FS}$-compatible Batch NP for R1CS.

# C Proof of Lemma 5.4

*Proof.* We construct $\mathsf{State}'$ from $\mathsf{State}$ by letting $\mathsf{State}'(\mathsf{CRS},x,\tau,\mathsf{aux}) = \mathsf{accept}$ if $\mathsf{State}(\mathsf{CRS},x,\tau',\mathsf{aux}) = \mathsf{accept}$ for any non-empty prefix $\tau'$ of $\tau$, and reject otherwise. Note that $\mathsf{State}'$ does indeed have the accepting state property. If $\mathsf{State}'(\mathsf{CRS},x,\tau',\mathsf{aux}) = \mathsf{accept}$, we know that there exists some non-empty prefix $\tau'$ of $\tau$ such that $\mathsf{State}(\mathsf{CRS},x,\tau',\mathsf{aux}) = \mathsf{accept}$. But $\tau'$ is also a prefix of $\tau\|\beta$ for any choice of $\beta$, and so $\mathsf{State}'(\mathsf{CRS},x,\tau\|\beta,\mathsf{aux}) = \mathsf{accept}$ as well. It now remains to prove that $\Pi$ still satisfies all the properties of Definition 4.5 when we replace $\mathsf{State}$ by $\mathsf{State}'$.

- **Completeness**: $\mathsf{State}$ plays no role in completeness, so this will trivially continue to hold when we use $\mathsf{State}'$.

- **$b$-Round-by-round soundness with respect to $\phi$**: We consider each of the properties required in Definition 4.4 below.

  - **Syntax**: By definition, we have that $\mathsf{State}'$ takes the proper inputs and outputs, and is deterministic since $\mathsf{State}$ is. Additionally, for any $\tau = (\alpha_1, \beta_1, \ldots, \alpha_j, \beta_j)$ and any $\alpha_{j+1}$, note that the only prefix of $\tau \| \alpha_{j+1}$ that is not also a prefix of $\tau$ is $\tau \| \alpha_{j+1}$ itself. Thus, we have

    $$\mathsf{State}'(\mathsf{CRS}, x, \tau \| \alpha_{j+1}, \mathsf{aux}) = \mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) \vee \mathsf{State}(\mathsf{CRS}, x, \tau \| \alpha_{j+1}, \mathsf{aux})$$

    But since $\mathsf{State}$ satisfies the syntax property, note that

    $$\mathsf{State}(\mathsf{CRS}, x, \tau \| \alpha_{j+1}, \mathsf{aux}) = \mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux})$$

    This tells us that

    $$\begin{aligned} \mathsf{State}'(\mathsf{CRS}, x, \tau \| \alpha_{j+1}, \mathsf{aux}) &= \mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) \vee \mathsf{State}(\mathsf{CRS}, x, \tau \| \alpha_{j+1}, \mathsf{aux}) \\ &= \mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) \vee \mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) \\ &= \mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) \end{aligned}$$

    where the last equality holds because $\tau$ is a prefix of itself, and hence $\mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$ whenever $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$.

  - **End Functionality**: Note that the only non-empty prefix of any transcript of the form $\alpha_1$ is itself, and hence we have that $\mathsf{State}'(\mathsf{CRS}, x, \alpha_1, \mathsf{aux}) = \mathsf{State}(\mathsf{CRS}, x, \alpha_1, \mathsf{aux})$ by definition. Thus, since $\mathsf{State}$ satisfies the end functionality requirement, we have that $\mathsf{State}'(\mathsf{CRS}, x, \alpha_1, \mathsf{aux}) = \mathsf{reject}$ iff $\phi(x, \alpha_1, \mathsf{aux}) = 1$. Additionally, for any complete transcript $\tau$, if $\mathsf{V}(\mathsf{CRS}, x, \tau) = 1$, we have that $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$ since $\mathsf{State}$ satisfies the end functionality property. But $\tau$ is a prefix of itself, and hence $\mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{accept}$ as well.

  - **Sparsity**: Fix any partial transcript $\tau = (\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}, \alpha_j)$ such that $\phi(x, \alpha_1, \mathsf{aux}) = 1$ and $\mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$. This means that $\mathsf{State}(\mathsf{CRS}, x, \tau', \mathsf{aux}) = \mathsf{reject}$ for all non-empty prefixes $\tau'$ of $\tau$. In particular, this means that $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$, and so by the sparsity of $\mathsf{State}$ with respect to $\phi$ we have

    $$\Pr_{\beta \leftarrow \{0,1\}^\lambda}[\mathsf{State}(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}] \leq b(\lambda) \cdot 2^{-\lambda}$$

    Note that since $\mathsf{State}(\mathsf{CRS}, x, \tau', \mathsf{aux}) = \mathsf{reject}$ for all non-empty prefixes $\tau'$ of $\tau$, we have that $\mathsf{State}'(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}$ if and only if $\mathsf{State}(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}$. This immediately gives us that

    $$\Pr_{\beta \leftarrow \{0,1\}^\lambda}[\mathsf{State}'(\mathsf{CRS}, x, \tau \| \beta, \mathsf{aux}) = \mathsf{accept}] \leq b(\lambda) \cdot 2^{-\lambda}$$

- **$d$-depth $B$-efficient $\mathsf{BAD}$ w.r.t. $\phi$**: Here we will use exactly the same $\mathsf{BAD}$ function that is guaranteed by Definition 4.5 when using $\mathsf{State}$, and prove that it still satisfies all the required properties when using $\mathsf{State}'$ instead.

  - **Syntax**: This property only depends on $\mathsf{BAD}$ and not on $\mathsf{State}$, so will trivially continue to hold when we replace $\mathsf{State}$ with $\mathsf{State}'$.

– BAD **w.r.t.** $\phi$: Let $\tau = (\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}, \alpha_j)$ denote any transcript such that we have $\mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ and $\phi(x, \alpha_1, \mathsf{aux}) = 1$. Since $\mathsf{State}'$ is reject, we know that in particular $\mathsf{State}(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$ (as $\tau$ is a prefix of itself), and hence $\mathsf{BAD}(x, \tau)$ enumerates the set $\mathcal{B}_{\mathsf{CRS},\mathsf{aux}}$, or outputs $\perp$ if $\mathcal{B}_{\mathsf{CRS},\mathsf{aux}}$ is empty. But now note that because $\mathsf{State}'(\mathsf{CRS}, x, \tau, \mathsf{aux}) = \mathsf{reject}$, we have that $\mathsf{State}(\mathsf{CRS}, x, \tau', \mathsf{aux}) = \mathsf{reject}$ for every non-empty prefix $\tau'$ of $\tau$. Since the only prefix of $\tau \| \beta$ that isn't also a prefix of $\tau$ is $\tau\|\beta$ itself, this tells us that $\mathsf{State}'(\mathsf{CRS}, x, \tau\|\beta, \mathsf{aux}) = \mathsf{accept}$ iff $\mathsf{State}(\mathsf{CRS}, x, \tau\|\beta, \mathsf{aux}) = \mathsf{accept}$. Thus, defining the set

$$\mathcal{B}'_{\mathsf{CRS},\mathsf{aux}} := \{\beta \;:\; \mathsf{State}'(\mathsf{CRS}, x, \tau\|\beta, \mathsf{aux}) = \mathsf{accept}\}$$

we have that $\mathcal{B}'_{\mathsf{CRS},\mathsf{aux}} = \mathcal{B}_{\mathsf{CRS},\mathsf{aux}}$, and hence we have that $\mathsf{BAD}(x, \tau)$ enumerates the set $\mathcal{B}'_{\mathsf{CRS},\mathsf{aux}}$, or outputs $\perp$ if $\mathcal{B}'_{\mathsf{CRS},\mathsf{aux}}$ is empty.

– **Low depth, $B$-efficient computation**: This property only depends on BAD and not on State, so will trivially continue to hold when we replace State with State$'$.

□

# D SNARGs for P

We begin by sketching the [KVZ21] approach, adapted to our setting. Their approach requires three main ingredients: a computational non-signaling PCP verifiable by tests (Definitions 2.5 and A.2 in [KVZ21]), a multi-extractable commitment scheme (Definition 3.4 in [KVZ21][11]), and a SNARG for BatchNP (Definition 6.2 in [KVZ21][12]). We give theorems to construct the first two of these ingredients below; for the SNARG, we will use the one obtained in Corollary 6.6.

**Theorem D.1** (Imported from [KRR14], [BHK17]). *Let $T = T(n)$ be a function such that $\mathsf{poly}(n) \leq T(n) \leq \exp(n)$ and let $\mathcal{L}$ be a language in $\mathsf{DTIME}(T)$. Then there exists an adaptive $T$-computational non-signaling PCP for $\mathcal{L}$ that can be verified via tests. The resulting PCP has length $L(n) = \mathsf{poly}(T(n))$ and can be generated in time $\mathsf{poly}(T(n))$. Letting $\lambda_{\mathsf{PCP}}$ be the security parameter, the PCP can be verified using $\ell(T) = \lambda_{\mathsf{PCP}} \cdot \mathsf{polylog}(T)$ queries, where the queries can be generated in $\mathsf{poly}(\ell(T))$ time and verified in $n \cdot \mathsf{poly}(\ell(T))$ time via tests; there are $\theta(T) = \mathsf{poly}(T)$ many possible tests.*

**Corollary D.2.** *Assume the $(T, T)$-hardness of QR. Then there exists a multi-extractable commitment scheme $\mathcal{C}$ satisfying $(T, T)$-index hiding with $\ell_{\mathsf{com}} = \ell \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}}$, $\ell_{\mathsf{open}} = \ell \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}}$, $|\mathsf{ck}| = \ell \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}+1}$, run time $\ell \cdot \mathsf{poly}(L)$ for $\mathcal{C}.\mathsf{Com}$, run time $\ell \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}} \cdot \mathsf{poly}(\lambda_{\mathcal{C}})$ for $\mathcal{C}.\mathsf{Open}$ and $\mathcal{C}.\mathsf{Verify}$. where $\lambda_{\mathcal{C}}$ is the security parameter of the commitment, $L$ is the length of the committed string, and $\ell$ is the number of locations where extraction is possible.*

*Proof.* Our starting point is the SE commitment from QR guaranteed by Theorem 3.6 for a block size $\ell_{\mathsf{blk}} = 1$. This satisfies $(T, T)$-index hiding and has $\ell_{\mathsf{com}} = \lambda_{\mathcal{C}}$, $\ell_{\mathsf{open}} = L$, $|\mathsf{ck}| = L \cdot \lambda_{\mathcal{C}}$.

[11] In [KVZ21], this was called a multi-extractable somewhere statistically binding scheme. Here we will assume (and achieve) perfect extraction rather than the statistical extraction in [KVZ21]; this simplifies the proof.

[12] The definition in [KVZ21] requires that the proof length for $k$ NP instances of size $n$ each is $\mathsf{poly}(\lambda, n, \log k)$. We will work instead with proof length $(k \cdot \mathsf{poly}(n, \lambda))^{O\left(\frac{1}{\sqrt{\log\log\log(k \cdot \mathsf{poly}(n,\lambda))}}\right)} \cdot \mathsf{poly}(n, \lambda)$; the rest of the definition is unchanged.

We then plug this scheme into our somewhere extractable commitments with non-trivial local openings (Theorem A.2) with $d = \sqrt{\log\log L}$ to get an SE commitment with $(T, T)$-index hiding and $\ell_{\mathsf{com}} = \lambda_{\mathcal{C}}^{\sqrt{\log\log L}}$, $\ell_{\mathsf{open}} = \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}}$, $|\mathsf{ck}| = \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}+1}$, run time $\mathsf{poly}(L)$ for $\mathsf{Com}$, run time $\sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}} \cdot \mathsf{poly}(\lambda_{\mathcal{C}})$ for $\mathsf{Open}$ and $\mathsf{Verify}$.

Finally, we apply Lemma 3.5 from [KVZ21] to get a multi-extractable commitment scheme satisfying $(T, T)$-index hiding with $\ell_{\mathsf{com}} = \ell \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}}$, $\ell_{\mathsf{open}} = \ell \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}}$, $|\mathsf{ck}| = \ell \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}+1}$, run time $\ell \cdot \mathsf{poly}(L)$ for $\mathsf{Com}$, run time $\ell \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}} \cdot \mathsf{poly}(\lambda_{\mathcal{C}})$ for $\mathsf{Open}$ and $\mathsf{Verify}$. □

The following theorem (cf Theorem 6.5 in [KVZ21]) says that if we instantiate Figure 12 with appropriate primitives, we get a SNARG for $\mathcal{L}$.

**SNARG for $\mathcal{L} \in \mathsf{P}$**

Fix any language $\mathcal{L}$, and let $T_{\mathcal{L}} = T_{\mathcal{L}}(n)$ be such that $\mathcal{L} \in \mathsf{DTIME}(T_{\mathcal{L}})$. Let $\mathsf{PCP} = (\mathsf{P}, \mathsf{Q}, \mathsf{V})$ be a PCP system for $\mathcal{L}$, $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ be a multi-extractable commitment scheme, and $\Pi_{\mathcal{M}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a SNARG for $\mathcal{M}^{\otimes \theta(T_{\mathcal{L}})}$ where $\mathcal{M}$ is the language described in Section 6.2 of [KVZ21].

- $\mathsf{Setup}_{\mathcal{L}}$ takes as input unary representations of security parameters $\lambda_{\mathsf{PCP}}$, $\lambda_{\mathcal{C}}$, and $\lambda_{\Pi}$ for PCP, $\mathcal{C}$, and $\Pi_{\mathcal{M}}$ respectively. It samples
  - A query set $Q \leftarrow \mathsf{PCP.Q}(1^{\lambda_{\mathsf{PCP}}})$
  - A hash key and trapdoor $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathcal{C}.\mathsf{Gen}(1^{\lambda_{\mathcal{C}}}, L, Q)$ where $L$ is the length of the PCP string
  - $\mathsf{CRS}_{\Pi} \leftarrow \Pi_{\mathcal{M}}.\mathsf{Setup}(1^{\lambda_{\Pi}})$

  $\mathsf{Setup}_{\mathcal{L}}$ then outputs $\mathsf{CRS} = (\mathsf{ck}, \mathsf{CRS}_{\Pi})$ and $\mathsf{aux} = (Q, \mathsf{td})$
- $\mathsf{P}_{\mathcal{L}}$ takes as input $\mathsf{CRS} = (\mathsf{ck}, \mathsf{CRS}_{\Pi})$ and an instance $x$. It computes
  - The PCP $\pi \leftarrow \mathsf{PCP}_{\mathcal{L}}.\mathsf{P}(x)$
  - A commitment $c \leftarrow \mathcal{C}.\mathsf{Com}(\mathsf{ck}, \pi)$
  - For each $j \in [\theta(T_{\mathcal{L}})]$, the $j$th possible test $\zeta_j$
  - For each $j \in [\theta(T_{\mathcal{L}})]$, a witness $w_j = (\pi|_{\zeta_j}, (\mathcal{C}.\mathsf{Open}(\mathsf{ck}, \pi, i))_{i \in \zeta_j})$
  - The proof $\sigma_{\Pi} = \Pi_{\mathcal{M}}.\mathsf{P}(\mathsf{CRS}_{\Pi}, (\zeta_j, x, \mathsf{ck}, c)_{j \in [\theta(T_{\mathcal{L}})]}, (w_j)_{j \in [\theta(T_{\mathcal{L}})]})$

  $\mathsf{P}_{\mathcal{L}}$ then outputs a proof $\sigma = (c, \sigma_{\Pi})$
- $\mathsf{V}_{\mathcal{L}}$ takes as input $\mathsf{CRS} = (\mathsf{ck}, \mathsf{CRS}_{\Pi})$, an instance $x$, and a proof $\sigma = (c, \sigma_{\Pi})$. It runs $\Pi_{\mathcal{L}}.\mathsf{V}(\mathsf{CRS}_{\Pi}, \langle (x, \mathsf{ck}, c) \rangle, \sigma_{\Pi})$ and outputs the result.

Figure 12: SNARG for P, cf Figure 6 in [KVZ21]

**Theorem D.3** ([KVZ21]). *Fix any $\epsilon > 0$. Suppose that* PCP *is an adaptive $T_{\mathcal{L}}$-computational non-signaling PCP for $\mathcal{L}$ with $\lambda_{\mathsf{PCP}} = \log(T_{\mathcal{L}})^{1/\epsilon}$, that $\mathcal{C}$ has $(T_{\mathcal{L}}, T_{\mathcal{L}})$-index hiding, and that $\Pi_{\mathcal{M}}$ is adaptively $T_{\mathcal{L}}$-sound*[13]*. Then Figure 12 describes a correct and sound SNARG for $\mathcal{L}$.*

[13]In [KVZ21], the requirement here was $2^{\ell_{\mathsf{com}}}$-soundness, where $\ell_{\mathsf{com}}$ is the length of the output of $\mathcal{C}$. However, this

In order to get SNARGs from LWE, [KVZ21] applied Theorem D.3 with a multi-extractable commitment built from [HW15] and the BatchNP SNARG from [CJJ21b]. In order to get SNARGs from DDH and QR, we will instead apply the theorem with the commitment from Corollary D.2 and the SNARG from Corollary 6.6.

**Theorem D.4** (cf Corollary 6.6 in [KVZ21])**.** *Let $\mathcal{L}$ be a language and $T_{\mathcal{L}} = T_{\mathcal{L}}(n)$ be a function such that $\mathsf{poly}(n) \leq T_{\mathcal{L}}(n) \leq \exp(n)$ and $\mathcal{L} \in \mathsf{DTIME}(T_{\mathcal{L}})$. Then assuming the subexponential hardness of QR and DDH, there exists a SNARG for $\mathcal{L}$ with prover time $\mathsf{poly}(T_{\mathcal{L}})$, verifier time $n \cdot \mathsf{poly}\left(T_{\mathcal{L}}^{\frac{1}{\sqrt{\log\log\log T_{\mathcal{L}}}}}\right)$, and communication complexity $n \cdot \mathsf{poly}\left(T_{\mathcal{L}}^{\frac{1}{\sqrt{\log\log\log T_{\mathcal{L}}}}}\right)$.*

*Proof.* We will use the SNARG from Figure 12 instantiated with

- PCP as the adaptive $T_{\mathcal{L}}$-computational non-signaling PCP guaranteed by Theorem D.1 with $\lambda_{\mathsf{PCP}} = \log(T_{\mathcal{L}})^{1/\epsilon}$ for some constant $\epsilon > 0$
- $\mathcal{C}$ as the commitment scheme guaranteed by Corollary D.2 with $\lambda_{\mathcal{C}} = T_{\mathcal{L}}^{\frac{1}{\log\log T_{\mathcal{L}}}}$
- $\Pi_{\mathcal{M}}$ as the SNARG for $\mathcal{M}^{\otimes\theta(T_{\mathcal{L}})}$ guaranteed by Corollary 6.6

For notational simplicity, in what follows we will define $\ell_2(T_{\mathcal{L}}) = T_{\mathcal{L}}^{\frac{1}{\sqrt{\log\log T_{\mathcal{L}}}}}$ and $\ell_3(T_{\mathcal{L}}) = T_{\mathcal{L}}^{\frac{1}{\sqrt{\log\log\log T_{\mathcal{L}}}}}$.

As a first step, we need to determine what values of $T$ and $S$ suffice to put $\mathcal{M} \in \mathsf{NTISP}(T, S)$ to determine what parameters we get out of Corollary 6.6. Note that verifying a single instance of $\mathcal{M}$ simply requires checking that openings in the witness verify and that the resulting values satisfy the test. Since there are at most $\ell(T_{\mathcal{L}})$ queries in any given test, checking them all takes $\ell(T_{\mathcal{L}}) \cdot \left(\ell(T_{\mathcal{L}}) \cdot \sqrt{\log\log L} \cdot L^{\frac{1}{\sqrt{\log\log L}}} \cdot \lambda_{\mathcal{C}}^{\sqrt{\log\log L}} \cdot \mathsf{poly}(\lambda_{\mathcal{C}})\right) = \mathsf{poly}\left(\ell(T_{\mathcal{L}}), L^{\frac{1}{\sqrt{\log\log L}}}, \lambda_{\mathcal{C}}^{\sqrt{\log\log L}}\right)$ time. Since $\lambda_{\mathcal{C}} = T_{\mathcal{L}}^{\frac{1}{\log\log T_{\mathcal{L}}}}$, $\ell(T_{\mathcal{L}}) = \lambda_{\mathsf{PCP}} \cdot \mathsf{polylog}(T_{\mathcal{L}}) = \mathsf{polylog}(T_{\mathcal{L}})$ and $L = \mathsf{poly}(T_{\mathcal{L}})$, this simplifies to $\mathsf{poly}(\ell_2(T_{\mathcal{L}}))$ time. The time to check that the test passes is at most the time it takes to verify the PCP, which is $n \cdot \mathsf{poly}(\ell(T_{\mathcal{L}})) = n \cdot \mathsf{polylog}(T_{\mathcal{L}})$. Putting these two together and noting that $\ell_2(T_{\mathcal{L}}) >> \log T_{\mathcal{L}}$, we have that $\mathcal{M} \in \mathsf{NTISP}\left(n \cdot \mathsf{poly}(\ell_2(T_{\mathcal{L}})), n \cdot \mathsf{poly}(\ell_2(T_{\mathcal{L}}))\right)$.

Note that $\mathsf{P}_{\mathcal{L}}$ needs to do three things: generate the PCP $\pi$, commit to $\pi$ using $\mathcal{C}$, and generate the proof for $\Pi_{\mathcal{L}}$. By Theorem D.1, the first part of this takes time $\mathsf{poly}(T_{\mathcal{L}})$. By Corollary D.2, the second part takes $\ell(T_{\mathcal{L}}) \cdot \mathsf{poly}(L)$ time. Since $\ell(T_{\mathcal{L}}) = \mathsf{polylog}(T_{\mathcal{L}})$ and $L = \mathsf{poly}(T_{\mathcal{L}})$, this becomes $\mathsf{poly}(T_{\mathcal{L}})$ time. Finally, by Corollary 6.6 with the parameters computed above, the third part takes $\mathsf{poly}\left(\theta(T_{\mathcal{L}}), n \cdot \mathsf{poly}(\ell_2(T_{\mathcal{L}}))\right)$; since $\theta(T_{\mathcal{L}}) = \mathsf{poly}(T_{\mathcal{L}})$, this simplifies to just $\mathsf{poly}(T_{\mathcal{L}})$ time. Putting these all together, we have that $\mathsf{P}_{\mathcal{L}}$ runs in time $\mathsf{poly}(T_{\mathcal{L}})$.

For the verifier time, note that $\mathsf{V}_{\mathcal{L}}$ needs only to run $\Pi_{\mathcal{L}}.\mathsf{V}$. Plugging in $T = S = n \cdot \mathsf{poly}(\ell_2(T_{\mathcal{L}}))$ and $k = \theta(T_{\mathcal{L}}) = \mathsf{poly}(T_{\mathcal{L}})$ into Corollary 6.6, we get that the time to run the verifier in $\Pi_{\mathcal{M}}$ is $(n \cdot \mathsf{poly}(T_{\mathcal{L}}))^{\frac{c}{\sqrt{\log\log\log(n \cdot \mathsf{poly}(T_{\mathcal{L}}))}}} \cdot (n \cdot \mathsf{poly}\left(\ell_2(T_{\mathcal{L}})\right) + |y|)$. Note that $y = \langle x, \mathsf{ck}, c\rangle$, and so has size

---

was only needed because the SNARG they were working with had non-adaptive soundness; since our SNARG achieves adaptive soundness, the requirement drops to just $T_{\mathcal{L}}$ as stated here.

$n + |\mathsf{ck}| + \ell_{\mathsf{com}}$. Plugging in $\ell(T_{\mathcal{L}}) = \mathsf{polylog}(T_{\mathcal{L}})$, $L = \mathsf{poly}(T_{\mathcal{L}})$, and $\lambda_{\mathcal{C}} = T_{\mathcal{L}}^{\frac{1}{\log\log T_{\mathcal{L}}}}$ to Corollary D.2, we get $|\mathsf{ck}|$ and $\ell_{\mathsf{com}}$ are both $\mathsf{poly}\left(\ell_2(T_{\mathcal{L}})\right)$. Thus, we have that $(n \cdot \mathsf{poly}\left(\ell_2(T_{\mathcal{L}})\right) + |y|)$ simplifies to $n \cdot (\mathsf{poly}\left(\ell_2(T_{\mathcal{L}})\right))$. Noting additionally that $n \cdot \mathsf{poly}(T_{\mathcal{L}}) = \mathsf{poly}(T_{\mathcal{L}})$, we have that $(n \cdot \mathsf{poly}(T_{\mathcal{L}}))^{\frac{c}{\sqrt{\log\log\log(n \cdot \mathsf{poly}(T_{\mathcal{L}})}}}$ simplifies to $T_{\mathcal{L}}^{O\left(\frac{1}{\sqrt{\log\log\log T_{\mathcal{L}}}}\right)} = \mathsf{poly}\left(\ell_3(T_{\mathcal{L}})\right)$. Thus the total verifier run time simplifies to $n \cdot \mathsf{poly}\left(\ell_3(T_{\mathcal{L}}), \ell_2(T_{\mathcal{L}})\right)$. Finally, since $\ell_3(T_{\mathcal{L}}) > \ell_2(T_{\mathcal{L}})$, we have that $\mathsf{V}_{\mathcal{L}}$ runs in time $n \cdot \mathsf{poly}\left(\ell_3(T_{\mathcal{L}})\right)$.

Finally, for communication complexity, we have to account for the commitment $c$ and the $\mathcal{L}_{\mathsf{PCP}}^{\otimes T_{\mathcal{L}}}$ proof $\sigma_{\Pi}$. As in the previous paragraph, we have that $\ell_{\mathsf{com}} = \mathsf{poly}\left(\ell_2(T_{\mathcal{L}})\right)$. Corollary 6.6 gives us the same bound of $n \cdot \mathsf{poly}\left(\ell_3(T_{\mathcal{L}})\right)$ on the communication complexity of $\Pi_{\mathcal{M}}$ as it gave for the run time of $\Pi_{\mathcal{M}}.\mathsf{V}$, so using the fact that $\ell_3(T_{\mathcal{L}}) > \ell_2(T_{\mathcal{L}})$ we get an overall bound of $n \cdot \mathsf{poly}\left(\ell_3(T_{\mathcal{L}})\right)$. $\square$