Correlation Intractability and SNARGs from Sub-exponential DDH

Arka Rai Choudhuri
NTT Research

Sanjam Garg
UC Berkeley & NTT Research

Abhishek Jain
JHU

Zhengzhong Jin
MIT

Jiaheng Zhang
UC Berkeley

Abstract

We provide the first constructions of SNARGs for Batch-NP and P based solely on the sub-exponential Decisional Diffie Hellman (DDH) assumption. Our schemes achieve poly-logarithmic proof sizes.

Central to our results and of independent interest is a new construction of correlation-intractable hash functions for "small input" product relations verifiable in TC^0, based on sub-exponential DDH.

Contents

1 Introduction **3**
- 1.1 Our Results . . . 4
- 1.2 Related Work . . . 5

2 Technical Overview **6**
- 2.1 CIH for Product Relations . . . 7
- 2.2 SNARGs for P & Batch-NP . . . 10
- 2.3 Somewhere Extractable Hash from DDH . . . 12

3 Preliminaries **17**
- 3.1 Low-degree Extensions . . . 17
- 3.2 Decisional Diffie-Hellman Assumption . . . 17
- 3.3 Correlation Intractable Hash . . . 18
- 3.4 Hash Tree . . . 19
- 3.5 Round-by-Round Soundness . . . 20

4 CIH for Product Relations **20**
- 4.1 Construction . . . 21
- 4.2 Proof of Correlation Intractability . . . 21
- 4.3 Extensions . . . 24

5 Somewhere Extractable Hash from DDH **25**
- 5.1 Definition . . . 26
- 5.2 Construction . . . 28
  - 5.2.1 Construction: Base Hash . . . 28
  - 5.2.2 Proof of Local Opening . . . 29
  - 5.2.3 Non-interactive Argument for $\mathcal{L} = (\mathcal{L}_{\text{YES}}, \mathcal{L}_{\text{NO}})$ . . . 33
  - 5.2.4 Bootstrapping via Tree Hashing . . . 38
- 5.3 No-Signaling Somewhere Extractable Hash . . . 38

6 SNARGs for Small-Circuit Batch-Index **39**
- 6.1 Definition . . . 39
- 6.2 Background: PCP . . . 41
- 6.3 Construction . . . 47

7 SNARGs for P and Batch-NP **47**
- 7.1 SNARGs for P . . . 47
- 7.2 SNARGs for Batch-NP . . . 48

A Interactive Trapdoor Hashing Protocol **55**

B Proof of Theorem 5 **56**

C Protocol: SNARGs for Small-Circuit Batch-Index **59**

D Protocol: SNARGs for P **61**
- D.1 Turing Machine Delegation . . . 61
- D.2 RAM Delegation . . . 62

1 Introduction

Suppose a client wants to learn the outcome of some large computation, but it does not have sufficient resources to perform the computation. Instead, it delegates the computation to an untrusted server. The server returns the output to the client, together with a proof attesting to the correctness of the output. The key requirement is that the client should be able to verify the proof in time that is significantly less than performing the original computation.

Such proofs are referred to as succinct non-interactive arguments (SNARGs). The standard model for SNARGs includes an initial setup, where a common reference string (CRS) is sampled and distributed to the parties. The proof verification procedure is public, and the soundness guarantee is against computationally-bounded provers. Over the years, SNARGs have found numerous applications to the design of advanced cryptographic protocols, and recently seen adoption in the blockchain ecosystem (see, e.g., [BCG+14]).

SNARGs from Standard Assumptions. In this work, we focus on the goal of constructing SNARGs from standard cryptographic assumptions. In this direction, [GW11] demonstrated a barrier to constructing SNARGs for general NP languages based on falsifiable assumptions. To overcome this barrier, a recent line of work has focused on the classes P and Batch-NP, where the former refers to deterministic polynomial-time computations, and the latter refers to the setting where a prover wishes to prove multiple (i.e., a batch of) NP statements via a proof of size smaller than the combined witness length.[1] We provide a (incomplete) summary below and refer the reader to Section 1.2 for details:

- The works of [CJJ21b, KVZ21] demonstrate that SNARGs for Batch–NP imply SNARGs for P.[2]
- SNARGs for Batch-NP (and hence, P) can be obtained from the learning with errors (LWE) assumption [CJJ21b], decisional linear assumption over bilinear maps [WW22], and joint assumptions of quadratic residuosity and (sub-exponential) Decisional Diffie-Hellman (DDH) [CJJ21a, HJKS22].

SNARGs based on the *sole* hardness of DDH, however, remain unknown. We ask:

Do there exist SNARGs for P *and Batch-*NP *based on* DDH*?*

Conceptually, a positive answer to this question would improve our understanding of the power of the discrete-logarithm family of assumptions. In recent years, a sequence of results have provided evidence that the gap between this family and the more powerful tools of lattices and bilinear maps is narrower than what was perceived previously. Some examples include succinct multiparty computation protocols [BGI16], identity based encryption [DG17], and non-interactive zero-knowledge proofs [JJ21] based on (sub-exponential) DDH. A positive resolution of our question would further consolidate this evidence.

Fiat-Shamir and Correlation Intractability. The aforementioned constructions of SNARGs (with the exception of [WW22]) rely on the Fiat-Shamir paradigm [FS87] — a generic method of round-collapsing public-coin interactive protocols. Given a public-coin interactive protocol, the Fiat-Shamir transformation replaces the verifier's messages with the hash of the protocol transcript so far.

The security of Fiat-Shamir was originally proven in the random oracle model. A recent line of work [CCR16, KRR17, CCRR18, HL18, CCH+19, PS19, CKU20, BKM20, JJ21, HLR21] has demonstrated that the Fiat-Shamir paradigm can be securely instantiated in the standard model using correlation intractable hash functions (CIH) [CGH04]. Intuitively, CIH are hash functions whose input-output pairs behave in a

[1] SNARGs for Batch-NP are also referred to as non-interactive batch arguments (BARGs).

[2] Their transformations rely on the existence of somewhere extractable hash functions [HW15], which are known based on many standard assumptions, including the ones considered in this work.

similar way to a random function in that they do not satisfy any "bad" correlations. More specifically, a hash function family $\{\mathcal{H}_k\}_k$ is said to be correlation intractable for a relation class $\mathcal{R}$, if for any relation R in the relation class, given the hash key k to any efficient adversary, it is hard to find an input-output pair of the hash function that satisfies the relation R.

A central goal in this line of research is to expand the class of relations that can be supported by CIH based on standard assumptions. Presently, the best known result is due to [HLR21] who construct CIH for so-called *efficiently verifiable product relations*, based on LWE. This result forms a key ingredient in the work of [CJJ21b] towards building SNARGs from LWE.

We ask whether such CIH can be constructed from other assumptions, and in particular, DDH:

Do there exist CIH *for efficiently verifiable product relations from* DDH*?*

While our primary interest in this work is in using CIH for constructing SNARGs, we mention that recent works have explored many other applications of CIH, including counter-examples to zero-knowledge [KRR17, HLR21], non-interactive zero knowledge proofs [CCRR18, CCH+19, PS19, CKU20, BKM20, JJ21], establishing hardness of complexity classes such as PPAD [CHK+19, JKKZ21], verifiable delay functions [LV20, FPS22], error-correcting codes [GHY20], and more. Thus, the above question is of interest beyond our immediate goal.

1.1 Our Results

SNARGs for P & Batch-NP. We construct SNARGs for P and Batch-NP based on the sub-exponential hardness of DDH. Specifically, we assume that there exists a constant $c \in (0,1)$ such that for any n.u. probabilistic adversary that runs in time $\lambda^{O((\log\log\lambda)^3)}$, its advantage in distinguishing a random tuple and a Diffie-Hellman tuple is at most $2^{-\lambda^c}$.[3]

Theorem 1 (Main Theorem, Informal)**.** *Assuming sub-exponential hardness of* DDH*, there exist:*

- **SNARGs for Batch-NP:** *For every polynomial $T = T(\lambda)$, there exists a* SNARG *for proving validity of T* C-SAT *instances $x_1, \cdots, x_T$ w.r.t. circuit C, where the size of the CRS and proof is* $\mathsf{poly}(\log T, |C|, \lambda)$ *and the verification time is* $\mathsf{poly}(T, |x_i|, \lambda) + \mathsf{poly}(\log T, |C|, \lambda)$.
- **SNARGs for P:** *For every polynomial $T = T(\lambda)$, there exists a* SNARG *for* $\mathsf{DTIME}(T)$ *where the verifier running time, size of CRS and proof are all* $\mathsf{poly}(\log T, \lambda)$ *and the prover running time is* $\mathsf{poly}(T, \lambda)$.

Both our constructions require a common *random* string setup, and their proof-sizes and verifier runtimes match the previous best-known results (ignoring multiplicative overhead in security parameter).

To prove Theorem 1, we follow the framework of [CJJ21b] that relies on two cryptographic tools: CIH for efficiently verifiable product relations, and somewhere-extractable hash (SEH) functions [HW15]. Along the way, we obtain new results on CIH and SEH. We discuss them next.

CIH for Efficiently Verifiable Product Relations. We first recall the definition of efficiently verifiable product relations [HLR21]. Let t be an integer, and $\mathcal{X}$ and $\mathcal{Y}$ be two sets of binary strings. We say that a relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ is a product relation, if for every $x \in \mathcal{X}$, $R_x = \{\mathrm{y} \mid (x, \mathrm{y}) \in R\}$ can be expressed as the Cartesian product of a series of sets $S_1, S_2, \ldots, S_t$. Further, we say that R is efficiently verifiable, if there exists a circuit C such that for every $i \in [t]$, the set S_i contains exactly all $y_i \in \mathcal{Y}$ such that $C(x, y_i, i) = 1$.

We build CIH for product relations whose verification circuit C is in TC^0 (i.e., constant-depth threshold circuits), based on the sub-exponential DDH assumption. In fact, our construction can support slightly super-constant depth threshold circuits.

[3] Our assumption is slightly stronger than the sub-exponential DDH assumption defined in [JJ21] (and used in [CJJ21a, HJKS22]) that only considers polynomial-time adversaries.

Theorem 2 (CIH for Product Relations, Informal)**.** *Assuming sub-exponential hardness of* DDH*, there exists correlation intractable hash functions for product relations verifiable by threshold circuits of depth* $O(\log^* \lambda)$*, with running time* $\mathsf{poly}(\lambda, n^{O(\log\log n)}, \log|\mathcal{Y}|)$*, where* n *is the bit-length of the input to the hash function and* $\mathcal{Y}$ *is the output space.*

Limitation: Small Inputs. The running time of our CIH is slightly super-polynomial in its input length. Hence, to obtain a polynomial-time hashing algorithm, our construction can only support inputs of size $\lambda^{O(1/\log\log\lambda)}$, which is a sub-polynomial in the security parameter λ. As we discuss later, this poses some additional challenges in adopting the framework of [CJJ21b] towards proving Theorem 1, and we develop new tools to overcome these challenges.

Our Approach. The recent work of [HLR21] constructed CIH for (all) efficiently verifiable product relations from LWE. At a high-level, their construction involves concatenating list-recoverable codes with CIH for efficiently searchable relations (that are known from LWE [PS19]). If we restrict ourselves to the sub-exponential DDH assumption, we only know CIH for searchable relations in TC^0 [JJ21]. This means that a direct attempt to port the approach of [HLR21] to the DDH-setting would require list-recoverable codes with *decoding* in TC^0. To the best of our knowledge, it is not known whether such codes exist. We therefore depart from the methodology of [HLR21], and present a new approach to construct CIH for product relations (albeit with the "small input" restriction) without using coding-theory techniques.

Somewhere-Extractable Hash. Another key ingredient in the framework of [CJJ21b] is the notion of somewhere extractable hash (SEH) with local openings [HW15]. Roughly speaking, a somewhere extractable hash is a keyed hash function with two indistinguishable modes: a normal mode and an extraction mode. In the extraction mode, the hash key is associated with an index i, and using a trapdoor, it is possible to extract the value committed at index i from the hash value. The local opening property requires that the committer can open to a particular bit in the committed message, with a succinct opening.

The limitations of our CIH for product relations in Theorem 2 dictate additional requirements on SEH for successfully adapting the framework of [CJJ21b] to the DDH setting:

- TC^0 *extraction*: We require the extraction circuit of SEH to be in TC^0.
- *Large inputs*: We require an SEH that supports input-sizes super-polynomial in the security parameter. Crucially, we still require that the local openings are succinct.

Intuitively, the first property is required since our CIH can only support relations verifiable in TC^0. The second property is dictated by the running time limitation of our CIH; see Section 2 for details.

We prove the following theorem:

Theorem 3 (Informal)**.** *Assuming sub-exponential hardness of* DDH*, there exists a large-input somewhere extractable hash with local openings and extraction algorithm in* TC^0*.*

Constructing SEH with the above two properties based on (sub-exponential) DDH turns out to be quite challenging. A natural adaptation of the tree-based approach to construct SEH with local openings used in prior works does not simultaneously achieve both properties. A key technical ingredient in our solution is a new Bulletproof-style [BBB+18] succinct proof in the pre-processing model that can be round-collapsed in the standard model using our CIH.

1.2 Related Work

SNARGs. We start by providing a brief summary of recent work on SNARGs from falsifiable and standard assumptions. [KPY19] gave the first construction of SNARGs for P and Batch-NP from a falsifiable,

albeit non-standard assumption over bilinear maps. Independently, [CCH+19] constructed SNARGs for bounded-depth deterministic computations assuming the existence of fully homomorphic encryption with optimal circular security. The first construction of SNARGs based on standard assumptions (namely, sub-exponential LWE) was given by [JKKZ21] for the class of bounded-depth deterministic computations.

Recently, [CJJ21b] constructed SNARGs for P and Batch-NP with poly-logarithmic proof size from LWE. Their work, as well as [KVZ21], also provides a transformation from SNARGs for Batch-NP to SNARGs for P.[4] In a separate work, [CJJ21a] constructed SNARGs for Batch-NP with proof size $\sqrt{k}$ for batch size k based on quadratic residuosity and sub-exponential DDH (or LWE). Subsequently, [HJKS22] improved the proof size to sublinear in k. More recently, [WW22] constructed SNARGs for Batch-NP with sublinear proof size based on the decisional linear assumption over bilinear maps. Finally, we mention very recent works that devise efficiency boosting compilers for SNARGs for Batch-NP: [PP22, DGKV22] construct rate-1 proofs with applications to incrementally verifiable computation, and [KLVW22] show how to achieve strong succinctness (i.e., poly-logarithmic proof size) from weak succinctness (i.e., proof size slightly smaller than the total witness length).

We conclude by mentioning three related lines of research: the first line of work, starting from [Mic94], constructs SNARGs for NP (see, e.g., [Gro10, Lip12, GGPR13, BCI+13, BCCT13, Gro16, BCC+17]) in the Random Oracle model or based on non-falsifiable assumptions [Nao03]. This line of work has led to efficient constructions of SNARGs that are currently used in practice. The second line of work, starting from [KRR13, KRR14], constructs designated-verifier SNARGs, where the verifier receives a secret key (sampled together with the common reference string) for verifying proofs. Finally, in the interactive setting (where the prover and the verifier exchange multiple messages), a long sequence of works, starting from [Kil92], have constructed succinct argument and proof systems for various classes. We refer the reader to [CJJ21b] for a more detailed overview.

CIH. A sequence of works [CCR16, KRR17, CCRR18, HL18, CKU20] constructed CIH for various classes of (not necessarily efficiently searchable) relations from strong assumptions. Recently, [CCH+19] constructed CIH for all efficiently searchable relations from circular-secure fully homomorphic encryption, and subsequently, [PS19] improved the assumption to standard LWE. More recently, [BKM20] constructed CIH for relations that can be approximated by constant-degreee polynomials (over $\mathbb{Z}_2$) based on various standard assumptions, and [JJ21] constructed CIH for TC^0 based on sub-exponential DDH. Finally, [HLR21] constructed CIH for efficiently verifiable product relations from LWE.

While the above works focus on single-input relations, a few works also study multi-input CIH for specific relations [Zha16, HL18, LV22]. Finally, we mention [CLMQ21, Mou21] that investigate the minimal assumptions necessary for CIH.

Concurrent Work. In a concurrent work, Kalai, Lombardi and Vaikuntanathan [KLV22] construct SNARGs for bounded-depth deterministic computations based on sub-exponential DDH (of the flavor considered in [JJ21]). Unlike our work, they also prove the hardness of the complexity class PPAD under the same assumption.

2 Technical Overview

We now provide a technical overview of our results. We organize the discussion in two parts: (1) First, in Section 2.1, we describe our construction of CIH for "small input" product relations verifiable in TC^0. (2) Next, in Section 2.2, we describe our constructions of SNARGs for Batch-NP and P. Along the way, we describe our new construction of somewhere extractable hash functions.

[4] The transformation of [KVZ21] also works for NTISP, i.e., bounded-space NP.

2.1 CIH for Product Relations

We start by introducing some terminology. For any binary relation $R \subseteq \mathcal{X} \times \mathcal{Y}$, we refer to an element $x \in \mathcal{X}$ as an *input*, and a $y \in \mathcal{Y}$ as a *challenge*. We say that a challenge y is *bad* with respect to x, if $(x, y) \in R$.[5] We will also interchangeably refer to such a y as a *witness*.

Further, we say that a relation R is *searchable* by f_R, if R is the set of all pairs (x, y) such that $\mathbf{y} = f_R(x)$.

CIH for Unique-Witness Product Relations. As a starting point, we first show how to build CIH for a weaker class of product relations, which we refer to as *unique-witness* product relations. We say that a product relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ is a unique-witness product relation, if for every input x, the witness (i.e., the bad challenge) is unique. That is, for every $x \in \mathcal{X}$, there is at most one $\mathbf{y} \in \mathcal{Y}^t$ such that $(x, \mathbf{y}) \in R$.

Let $|\mathcal{Y}|$ be a polynomial in the security parameter. We observe that for any unique-witness product relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$, there exists a function $f_R : \mathcal{X} \to \mathcal{Y}^t$ such that R is *searchable* by f_R. To see this, note that we can find the unique bad challenge $\mathbf{y} = (y_1, \ldots, y_t) \in \mathcal{Y}^t$ for any input x by enumerating over all values y_i for every i. In more detail, let $C(x, y_i, i)$ be the circuit that verifies the product relation R. Our function f_R takes as input x, and for each i in $[t]$, it enumerates every $y'_i \in \mathcal{Y}$ and if $C(x, y'_i, i) = 1$, then it sets y_i to be this y'_i. Such a function f_R (nearly) preserves the depth of the verification circuit C, since we can run t enumerations of y'_i's in parallel. Moreover, f_R has polynomial-size description since $|\mathcal{Y}|$ is a polynomial in the security parameter.

A recent work of [JJ21] constructed CIH for relations searchable in TC^0. Combining their result with the above observation, we can obtain CIH for unique-witness product relations verifiable in TC^0.

Handling Multiple Witnesses. In general, product relations may not have unique witnesses. Indeed, this is the case for the applications we consider in this work. Hence, CIH for unique witness product relations are seemingly of limited interest. Somewhat surprisingly, we now show that constructing CIH for non-unique-witness product relations can, in fact, be reduced to the task of constructing CIH for unique witness product relations.

Dividing the Challenges. Our high level idea is to divide the challenge space into several smaller parts. Let's first consider a simpler case for $t = 1$. We divide $\mathcal{Y}$ into $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_d$. Looking ahead, we will define the bad challenges in $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_d$, and hope that these bad challenges in $\mathcal{Y}$ will "spread" among $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_d$ such that in each part $\mathcal{Y}_i$, the bad challenge is *unique*. We will then generalize this to $\mathcal{Y}_1^t, \mathcal{Y}_2^t, \ldots, \mathcal{Y}_d^t$ for general t and associate a series of new relations $R_1, R_2, \ldots, R_d$ to these smaller challenge spaces and define bad challenges in $\mathcal{Y}_i^t$ accordingly.

More specifically, we treat each challenge $y \in \mathcal{Y}$ as a binary string of length $\log_2 |\mathcal{Y}|$. We then divide the binary representation of the challenges evenly into d parts. Hence, $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_d$ simply contain binary strings of length $\log_2 |\mathcal{Y}|/d$. A CIH for a product relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$, given any input x, operates as follows: it first computes each part of the hash value $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_d$, where $\mathbf{y}_i \in \mathcal{Y}_i^t$, and then point-wise concatenates $\mathbf{y}_i$'s together to obtain the hash value $\mathbf{y} = \mathbf{y}_1 || \mathbf{y}_2 || \ldots || \mathbf{y}_d$. To determine how we compute $\mathbf{y}_i$, we take inspiration from how CIH is applied in the actual interactive protocols. Usually we use the CIH for the relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ to instantiate the Fiat-Shamir transformation in an interactive protocol, where the prover sends an x in the first round, and the verifier samples a random $y \leftarrow \mathcal{Y}$ in the second round. Executing this protocol in parallel t times results in the challenge space $\mathcal{Y}^t$. In this application, dividing the challenge space $\mathcal{Y}$ into $\mathcal{Y}_1, \ldots \mathcal{Y}_d$ corresponds to the following multi-round protocol: instead of sampling $y \leftarrow \mathcal{Y}$ directly and sending it to the prover, the verifier samples $y_1 \leftarrow \mathcal{Y}_1$ and sends it to the

[5] When x is clear from the context, we simply say that y is bad.

prover, then the prover does nothing, and then the verifier samples $y_2 \leftarrow \mathcal{Y}_2$ and sends it to the prover. They continue this process in d-rounds until all y_i's are sent.

Applying Fiat-Shamir transformation directly to this multi-round protocol, we obtain the following iterative hashing algorithm. For $i = 1, 2, \ldots d$, we compute $\mathrm{y}_i = \mathsf{Hash}_i(x, \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_{i-1})$ where $\mathsf{Hash}_i, i = 1, 2, \ldots, d$ is a series of hash functions. At i-th round, the hash function Hash_i takes the transcript of the protocol so far, which is the tuple $(x, \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_{i-1})$, where $\mathrm{y}_i \in \mathcal{Y}_i^t$ is the vector of challenges in the t parallel repetitions in the i-th round. Inspired by this Fiat-Shamir transformation, we construct our CIH for R by iteratively computing a series of CIH in the same way as above, except that we will instantiate Hash_i as CIH for unique-witness product relations. That is,

$$\text{For } i = 1, 2, \ldots, d, \quad \mathrm{y}_i = \mathsf{Hash}(\mathsf{k}_i, x, \mathrm{y}_1, \ldots, \mathrm{y}_{i-1}),$$

where k_i is the CIH key for the i-th CIH, and we represent the hash algorithm of CIH as $\mathsf{Hash}(\mathsf{k}_i, \cdot)$. To prove the correlation intractability of our CIH construction, we will reduce the security of our CIH construction for R to the correlation intractability of $\mathsf{Hash}(\mathsf{k}_i, \cdot)$ for some relation R_i. To achieve this, we first need to define the relations $\{R_i\}_{i\in[d]}$ carefully so that they are unique-witness product relations.

Defining Unique-Witness Product Relations $\{R_i\}_{i\in[d]}$. For each $i = 1, 2, \ldots d$, we will define the relation R_i over $(\mathcal{X} \times \mathcal{Y}_1^t \times \ldots \times \mathcal{Y}_{i-1}^t) \times \mathcal{Y}_i^t$. Before we define R_i, we introduce the following recursive view of the aforementioned multi-round protocol. Essentially, the aforementioned verifier is dividing the whole challenge space $\mathcal{Y}$ into $|\mathcal{Y}_1|$ parts in the first round, and then it takes the y_1-th part as the remaining challenge space $\{y_1\} \times \mathcal{Y}_2 \ldots \times \mathcal{Y}_d$ in the second round. In the second round, the verifier further divides the remaining challenge space into $|\mathcal{Y}_2|$ parts and choose the y_2-th part and so on. Finally, in the d-round, after chosen y_d, there is only one challenge $y = y_1 || \ldots || y_d$ left in the challenge space and so y is chosen as the challenge for the entire protocol.

To define R_i, we focus on the number of bad challenges in the remaining challenge space after each round in this recursive dividing process. Our key observation is that in the i-th round, there is *at most one* y_i such that the number of bad challenges in the y_i-th part is greater than 50% of the number of the bad challenges in the remaining challenge space before the i-th round, i.e. the challenges with the prefix $y_1 || \ldots || y_{i-1}$. This is because if there are two such y_i's, then the number of bad challenges exceeds 100%, which is a contradiction. Specifically, we use $\mathsf{BadCnt}(x, y_1, y_2, \ldots, y_{i-1})$ to denote the number of bad challenges for x with the prefix $y_1 || y_2 || \ldots || y_{i-1}$. We say y_i is bad with respect to $(x, y_1, \ldots, y_{i-1})$, if $\mathsf{BadCnt}(x, y_1, y_2, \ldots, y_{i-1}, y_i) > \mathsf{BadCnt}(x, y_1, y_2, \ldots, y_{i-1})/2$. Namely, we define $R_{i,x}$ as the set containing the following challenges.

$$R_{i,x} = \left\{ y_i \mid \mathsf{BadCnt}(x, y_1, y_2, \ldots, y_{i-1}, y_i) > \mathsf{BadCnt}(x, y_1, y_2, \ldots, y_{i-1})/2 \right\}.$$

This is our definition of bad challenges for one parallel execution. Next, we define the bad challenge for t parallel executions by taking the product of above bad relations for all parallel executions. Namely, let the relation $R_{i,x}^{(j)}$ be the bad relation defined as above for the j-th parallel execution. Then we define R_i as the product relation specified by $R_{i,x}^{(1)} \times R_{i,x}^{(2)} \times \ldots \times R_{i,x}^{(t)}$. Since each $R_{i,x}^{(j)}$ contains unique bad challenge, we have that R_i is a unique-witness product relation. Now we have defined a series of unique-witness product relations $R_1, R_2, \ldots, R_d$. As we will see, each R_i is associated with the hash function $\mathsf{Hash}(\mathsf{k}_i, \cdot)$.

Proving Correlation Intractability. To prove correlation intractability, we first need to show that the relations $R_1, R_2, \ldots, R_d$ "cover" the relation R. That is, for any x and $\mathrm{y} \in \mathcal{Y}^t$, where y is the point-wise concatenation of $\mathrm{y}_1, \ldots, \mathrm{y}_d$ and y_i is a challenge in $\mathcal{Y}_i^t$, if y is bad in R, then there must be some i^* such that y_{i^*} is bad in R_i. If we have this covering claim, then any adversary for our construction of CIH for R

is essentially also an adversary for the i^*-th CIH for the relation R_{i^*}. Hence the correlation intractability of our construction follows from the correlation intractability of $\mathsf{Hash}(\mathsf{k}_i, \cdot)$. To finish this security proof, it remains to prove the covering claim.

We first prove a "weak covering claim", in which there is only one parallel execution ($t = 1$). Namely, we will show that in the aforementioned definition of $R_{i,x}$, we have that if $y \in \mathcal{Y}$ is bad with respect to x and $y = y_1 || \ldots || y_d$ where $y_i \in \mathcal{Y}_i$, then there must be some i^* such that y_{i^*} is bad (i.e. $y_{i^*} \in S_{i^*,x}$), once we set the parameter d to be greater than $\log_2 \mathsf{BadCnt}(x)$. To see why this is true, if the first $(d-1)$ parts $y_1, y_2, \ldots, y_{d-1}$ contain a bad challenge, then our weak covering claim holds. Otherwise, the first $(d-1)$ parts must half the number of bad challenges in each round, but then the last part y_d must be bad, since the count of the bad challenge in the last round is one and the bad challenge count in the second last round is already $\mathsf{BadCnt}(x)/2^{d-1} \leq 1$.

Now we attempt to prove the original covering claim, where there are t parallel executions, using the above weak covering claim. We will soon see that this is unachievable, and we need to expand our relations R_i's a little. For every x and $\mathbf{y} \in \mathcal{Y}^t$, let $\mathbf{y}$ be the point-wise concatenation $\mathbf{y}_1 || \ldots || \mathbf{y}_d$. We are going to visualize $\mathbf{y}$ as a matrix of t columns and d rows, where the i-th row, for every $i \in [d]$, is the vector $\mathbf{y}_i \in \mathcal{Y}_i^t$, and every column corresponds to a parallel execution. If $\mathbf{y}$ is bad, then every coordinate of $\mathbf{y}$ is bad. By the weak covering claim, there is at least one bad element in each column of the above matrix. However, we can not conclude that there exists a row $\mathbf{y}_{i^*}$ where all its coordinates are bad, since the bad elements in different columns may be at the different rows. But using the pigeonhole principle, we can still derive something non-trivial. Specifically, there must be some row $\mathbf{y}_{i^*}$ that contains at least t/d bad elements, because the matrix has t columns and thus the entire matrix contains at least t bad elements.

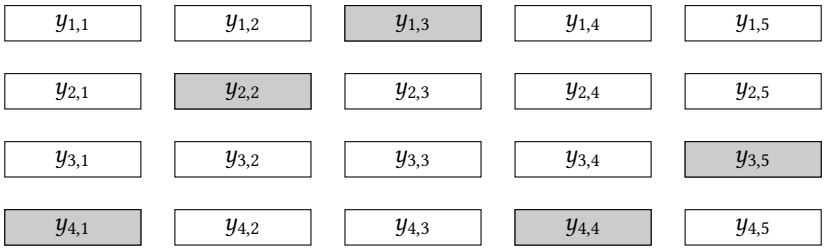

Figure 1: For illustration, consider the setting where $t = 5$, and $d = 4$. Each column represents a parallel execution, where the concatenation of the values within the same column represents a *single element* of $\mathcal{Y}$. The shaded $y_{i,j}$ elements represent the bad elements, where a bad $\mathbf{y}$ guarantees the existence of (at least) one bad element per column. By pigeonhole principle there exists a row (row 4 in the above example) that has at least $\lceil t/d \rceil$ bad elements.

Now we modify the covering claim accordingly. We first use the notion of *α-approximate* product relation [HLR21] to characterize a super-set R_i^α of R_i, which contains all vectors $\mathbf{y}_i \in \mathcal{Y}_i^t$ such that at least $\alpha = (t/d)/t = 1/d$ fraction of coordinates are bad. Then we modify the covering claim to assert that $R_1^\alpha, R_2^\alpha, \ldots, R_d^\alpha$ "cover" R. Then the pigeonhole principle above shows that the there must exist a row $\mathbf{y}_{i^*}$ of the matrix that is bad in $R_{i^*}^\alpha$. Hence, the modified covering lemma is proved.

The reader may wonder how we build CIH for α-approximate product relations R_i^α, since our previous discussion can only handle CIH for unique-witness product relations, and now our covering lemma requires CIH for α-approximate product relations. We observe that CIH for searchable relations in TC^0 by [JJ21] can, in fact, be extended to handle "α-approximate searchable relations" in TC^0, and our previous reduction from unique-witness product relation to searchable relations can be extended to the approximate setting. Namely, by the same enumerating method, we can show that R_i^α's are α approximate searchable relations. Hence, we can instantiate $\mathsf{Hash}(\mathsf{k}_i, \cdot)$ using CIH for α-approximate searchable relations.

Restricting to Small Inputs. Let's examine the running time of our CIH construction. Recall that our CIH construction needs to enumerate the challenges in $\mathcal{Y}$ to count the number of the bad challenges. Hence, the running time of our CIH is at least $|\mathcal{Y}|$. Let λ be the security parameter, then we have $d = \log_2 \mathsf{BadCnt}(x) = \Theta(\log \lambda)$ number of parts, where each part has size $|\mathcal{Y}_i|$. We now see the lower bound of $|\mathcal{Y}_i|$. Indeed, the approximation factor α needs to be at least $1/|\mathcal{Y}_i|$, otherwise the CIH for α-approximate relations R_i^α in each round can't exist. This is because an uniform random challenge $y_i \leftarrow \mathcal{Y}_i$ has probability $1/|\mathcal{Y}_i|$ to be bad (i.e. $y_i \in S_{i,x}$). For t parallel executions, this implies that on average there are $1/|\mathcal{Y}_i|$ fraction of bad coordinates in a random $\mathbf{y}_i \leftarrow \mathcal{Y}_i^t$. However, for CIH to exist, the fraction parameter α at least need to be non-trivial. Hence, α needs to be greater than $1/|\mathcal{Y}_i|$. Recall that $\alpha = 1/d$ in our covering claim. This implies that $|\mathcal{Y}_i|$ needs to be at least d, and hence $|\mathcal{Y}|$ is at least $d^{\log d} = \lambda^{\Theta(\log\log\lambda)}$, which is slightly *super-polynomial*. This means that the running time of our CIH is super-polynomial.

To get around this issue, we observe that once we restrict the number of bad challenges to be $\lambda^{O(1/\log\log\lambda)}$ then $|\mathcal{Y}|$ becomes a polynomial in λ. However, by putting such a restriction, our CIH can only support relations where the fraction of bad challenges in the entire challenge space is $\lambda^{o(1)}/|\mathcal{Y}| = o(1)$. To build CIH for general product relations that supports any fraction of bad challenges, we further apply a subsampling technique from [HLR21]. The idea is to divide the whole t parallel execution into L blocks, where each block consists of $\ell = t/L$ parallel executions. Within each block, we sample a series of challenges $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N \leftarrow \mathcal{Y}^\ell$ randomly, and use a union bound over all possible input to argue that for every input, at most $\approx \log|\mathcal{X}|$ of those N challenges are bad. Now, we can view $[N]$ as a new challenge space, where each challenge is an index $i \in [N]$ corresponding to an original challenge $\mathbf{q}_i$. We can also define a new relation R' over $\mathcal{X} \times [N]^L$, which contains all "bad" tuples of indices. Once we build CIH for R', we obtain CIH for R by selecting the corresponding $\mathbf{q}_i$'s according to the output of CIH for R'. According to aforementioned size lower bound on the challenge space, we derive that N needs to be at least $(\log B)^{\log B}$, where $B \approx \log|\mathcal{X}|$ is the number of bad challenges among $\mathbf{q}_i$'s. Hence, once we restrict $\log|\mathcal{X}| \le \lambda^{O(1/\log\log\lambda)}$, then the N and also the running time of our CIH construction becomes polynomial in λ. For more details, see Section 4.1.

2.2 SNARGs for P & Batch-NP

We now describe applications of our CIH to SNARGs for P and Batch-NP. Recent works of [KVZ21, CJJ21b] demonstrated that SNARGs for Batch-NP are "complete" for constructing SNARGs for P. In fact, [CJJ21b] reduces the task of constructing SNARGs for $\mathsf{DTIME}(T)$ computation to constructing SNARGs for a special Batch-NP language called the *batch-index* language. A batch-index language is associated with a circuit C where the instances are simply indices i that can be represented in $\log k$ bits for a batch of size k. Unlike Batch-NP where the verifier has to read all the instance, one can make a strong verification requirement from SNARGs for batch-index language, namely, that the *total* verification time (including the time to "read" instances) is $\mathrm{poly}(\lambda, \log k, |C|)$.

In the rest of this subsection, we restrict our attention to constructing SNARGs for batch-index.

A Brief Primer on [CJJ21b]. We start by describing the high level paradigm in [CJJ21b] for the construction of SNARGs for batch index. Our construction will be identical except for a few key changes that we shall explain later. A reader who is familiar with the prior work can choose to skip this discussion.

PCP Generation: Using witnesses $\omega_1, \cdots, \omega_T$, the prover generates T PCPs $\pi_1, \cdots, \pi_T$.

Succinct Column-wise Hash: The prover arranges the $\pi_1, \cdots, \pi_T \in \{0,1\}^\ell$ into rows, and hashes them in a column-wise manner (i.e. ℓ hashes each hashing vectors of length T) using a somewhere extractable hash (to be discussed shortly). Let this hash be denoted by $c = (c_1, \cdots, c_\ell)$.

Challenge Generation via CIH: The prover applies the CIH (different from the hash scheme in the previous step) to (C, c) to generate the PCP queries Q.

Two-to-one Compression: The prover then defines a new circuit C' for the index language that hard-codes the $\{c_j\}_{j\in Q}$ (the hash c restricted only on "columns" specified by Q), and takes as input index i and witness ω_i - where ω_i corresponds to the *local* openings of $\{\pi_{i,j}\}_{j\in Q}$. C' checks (i) that the local openings verify; and (ii) the PCP verifier accepts given the PCP responses $\{\pi_{i,j}\}_{j\in Q}$. The prover further reduces the total number of (index) instances from T to $T/2$ applying a 2-to-1 reduction to C' by defining a circuit $C''(i, (\omega_{2i-1}, \omega_{2i})) := C'(2i-1, \omega_{2i-1}) \wedge C'(2i-1, \omega_{2i-1})$.

Recursion: The prover finally recurses by repeating the above steps, sending witnesses in the clear when the witnesses are small enough.

To ensure that the recursion reduces the size of the circuit and the number of instances, one needs to demonstrate that C'' defined above is "small" with respect to C. The size of C' (and thus of C'') is determined by:

1. size of the hash and local opening; and
2. size of the PCP verification circuit.

For (1), [CJJ21b] used a somewhere extractable hash (SEH) [HW15] with local openings. In such a hashing scheme, a trapdoor td is generated along with the key K on input i^* — td can then be used to extract, from c, the uniquely value at index i^* where the resultant hash is c.[6] The SEH used in the aforementioned work has hash size, local opening size and verification time all to be polylogarithmic in T. For (2), [CJJ21b] observed that existing PCP schemes can be viewed in an offline/online verification model where the offline verifier pre-processes the circuit C to produce PCP queries and a small state st such that the online verification depends only polylogarithmically on T and $|C|$ given the PCP responses.

It was shown in [CJJ21b] that CIH for *efficiently verifiable product relations* (as defined in [HLR21]) suffices to prove security of the above described SNARG for batch index. Such a CIH was instantiated from the work of [HLR21] assuming the hardness of LWE.

Key Challenges. The [CJJ21b] template uses two cryptographic primitives based on LWE, namely, SEH and CIH. Thus, a natural idea to port the [CJJ21b] approach to our DDH-setting is as follows: replace the CIH used in [CJJ21b] with our new CIH from Section 2.1, and instantiate the SEH used in their scheme with one based on DDH [OPWW15, DGI+19].

This simple strategy, however, does not work right out of the box due to the limitations of our new CIH. We describe the key challenges that arise in the process:

Input Size: As described in Section 2.1, our new CIH is restricted by its input size being *small*, i.e. for security parameter λ, the input size is bounded by $\mathsf{poly}(\lambda_0)$ where $\lambda_0^{O(\log\log\lambda_0)} = \lambda$. From the above description of the [CJJ21b] protocol, we have that the input to the CIH consists of the circuit C (for the batch-index language) and the output of SEH. This means that *both the circuit size and the hash size are bounded by* $\mathsf{poly}(\lambda_0)$.

1. **Bound on size of C:** The restriction on C is pretty significant since for obtaining a general result, we would like to allow for circuits that are of size $\mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_0^{O(\log\log\lambda_0)})$.
   Despite this seemingly strong restriction, we show that: **(i)** It is already sufficient to achieve SNARGs for P — intuitively, this is because the circuit for batch-index language defined in the transformation from SNARGs for batch-index to SNARGs for P [CJJ21b] are already "small"; **(ii)** With the help of SEH, we can bootstrap SNARGs for *small circuit* batch-index

[6] An SEH implies *somewhere binding* at the same index.

languages to achieve SNARGs for general batch-index languages. At a high level, we split the large circuit C into many smaller components $C_1, \cdots, C_\ell$ such that each such component is of size $\mathsf{poly}(\lambda_0)$, and send ℓ proofs for each corresponding circuit. Care must be taken to ensure that the smaller components are not disparate, but in fact "connect" to verify a batch claim about the original circuit C. We do not discuss the details in this overview and refer the reader to Section 7.2.

2. **Bound on hash size:** The bound on the hash size necessitates that we must set the security parameter of SEH to be λ_0. Note, however, that the number of instances T remains polynomial in λ. This means that the SEH must support **large inputs**, i.e *inputs that are of length super-polynomial in its security parameter.*

   Further, recall that in the [CJJ21b] template discussed earlier, the size of the circuit C' — that is recursed in the next step — is determined by the size of the local opening of SEH. Putting this together with the bound on the circuit size, we have the restriction that the size of the local opening of SEH can depend on $\mathsf{poly}(\lambda_0)$ and only poly-logarithmically on the large input length.

Verification in TC^0: In addition to the restriction on input size, for a secure application of our new CIH, we require that the bad challenge relation circuit be be verifiable in TC^0. The bad challenge relation circuit involves the following key steps:

1. **Extraction of PCP:** The PCP is extracted from the SEH.
2. **PCP Verification:** The online verification of the PCP is executed given its queries.
3. **Checking satisfiability of the circuit:** To ascertain if the i-th instance is true, check circuit satisfiability of C using witness extracted from the PCP.

Thus, we require that each of the above three steps can be performed in TC^0. By choosing an appropriate field, we extend the analysis of [CJJ21b] to show that both the extraction of the witness from the PCP and online PCP verification can be computed in TC^0. Further, given a witness corresponding to a circuit C, checking if the circuit is satisfiable can be done in TC^0 by checking all the gates in parallel. Thus steps 2 and 3 can be performed in TC^0.

This leaves us with step 1, namely, we need SEH that supports **extraction in TC^0**.

In summary, we are left with the task of constructing an SEH with the following key requirements:

- **Large Inputs:** The SEH must support inputs of length super-polynomial in the security parameter, but nevertheless have succinct local openings.
- TC^0 **extraction:** The SEH must support extraction in TC^0.

In the remainder of this overview, we describe the key ideas underlying our construction of such a SEH based on sub-exponential DDH.

2.3 Somewhere Extractable Hash from DDH

We start by recalling prior approaches to constructing SEH. We then discuss the main technical challenge towards achieving the two properties we require. Finally, we present our solution.

Background. We follow the standard *tree-based approach* from prior work for constructing SEH with short local openings. To hash a vector of length T, the idea is to build a hash tree (akin to a Merkle tree) with the leaves corresponding to the vector being hashed, and the root corresponding to the resultant hash. We shall refer to the hash used in the construction of the tree to be the *base hash*, and the overall

construction simply by SEH. The local opening to a bit (a leaf in the tree) consists of all siblings nodes on the path from the leaf to the root, and verification involves recomputing the hash values along the path from the claimed leaf to the root (using the siblings present in the proof) and comparing against the stored root. Thus the size of the opening is determined by the number of siblings and the depth of the tree, both of which are fixed for a given vector length T by the arity (or degree) of the tree. Thus, if we were to set the arity of the tree to be bounded by $\mathsf{poly}(\lambda_0)$, where λ_0 is the security parameter of SEH, the proof of local opening would be bounded by $\mathsf{poly}(\lambda_0, \log_{\lambda_0} T)$.

Main Challenge. Recall that our requirements from SEH are: (1) extraction in TC^0, (2) support for inputs of length super-polynomial in the security parameter λ_0, while still allowing for short local openings.

Let us consider the first requirement. In the context of tree based construction, extraction from SEH translates to the extraction of a leaf value from the hashed root. The extraction procedure (on input the trapdoor) starts from the root and works down to the (desired) leaf by extracting each hash value along the path, one level at a time. The number of sequential steps in the extraction procedure is proportional to the depth of the tree. Thus for extraction to be computable in TC^0, the depth of the tree must be a *constant*.

Now, let us consider the second requirement on SEH. Note that in order to support large inputs, we can no longer set the arity of the tree to be $\mathsf{poly}(\lambda_0)$. This follows from the fact that for a constant depth tree with arity $\mathsf{poly}(\lambda_0)$, the total number of leaves is limited to be polynomial in λ_0. One way to circumvent this, and support inputs of size $\mathsf{poly}(\lambda)$, would be to set the arity in the tree based construction to be $\mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_0^{O(\log\log\lambda_0)})$. While this does support large inputs in constant depth, a local opening, which constitutes sending all the siblings along the path is now of size $\mathsf{poly}(\lambda_0^{O(\log\log\lambda_0)})$, i.e. we lose succinctness. This constitutes the main challenge that we need to overcome in our construction, namely, devising succinct local openings when the arity of the tree is large.

Sibling Compression. Our key idea is to compress the large number of siblings into a single "effective" sibling. The effective sibling must allow for re-computation of the hash, but it must not be possible to manipulate the effective sibling to produce inconsistent local openings.

To explain the idea, we first describe the *base hash* scheme that we use in our tree based construction. The scheme closely follows the construction of trapdoor hash functions based on DDH [DGI+19]. The hash function Hash takes as input $\mathbf{m} \in \{0,1\}^N$, a key $K \in \mathbb{G}^{2\times N}$ and produces a hash output $c \in \mathbb{G}^2$. To generate a hash key that allows for somewhere extraction of the input bit at index i^* [7], the key generation, hashing and extraction algorithms are described below

Key Generation: On input i^*, the key K is generated as

$$K = \begin{bmatrix} g_1 & g_2 & \cdots & g_N \\ h_1 & h_2 & \cdots & h_N \end{bmatrix} = \begin{bmatrix} g_1 & \cdots & g_{i^*-1} & g_{i^*} & g_{i^*+1} & \cdots & g_N \\ g_1^s & \cdots & g_{i^*-1}^s & g_{i^*}^s \cdot g & g_{i^*+1}^s & \cdots & g_N^s \end{bmatrix}$$

where for every $j \in \{1, \cdots, N\}$, g_j is a random group element from $\mathbb{G}$, g is the group generator, and s is randomly sampled. The trapdoor for this generated key is s. At a high level, h_j at every position other than i^* is computed by the exponentiation of g_j to the trapdoor s, whereas h_{i^*} has an additional product term of g. This structure will be exploited for the extraction.

Hashing: On input $\mathbf{m} = (m_1, \cdots, m_N) \in \{0,1\}^N$, and key K, compute the hash $c = (g', h') = \mathsf{Hash}(K, \mathbf{m})$ as $g' := \prod_{j=1}^N g_j^{m_j}$ and $h' := \prod_{j=1}^N h_j^{m_j}$.

[7] We refer the reader to the technical sections for discussion on the extension of the somewhere extraction property to multiple bits

Extraction: On input a hash $c = (g', h')$ and trapdoor s, output m_{i^*} to be 0 if $g'^s = h'$, and 1 otherwise. Intuitively, the correctness of extraction follows from the fact that if m_{i^*} is 0, the additional g term in h_{i^*} will disappear.

Assuming DDH, one can argue (as in [DGI+19]) that the index i^* is hidden in the generated hash key K. Before we discuss the local opening properties of the base scheme, note that the extraction involves computing a group exponentiation g'^s. With an appropriate pre-processing of g', the exponentiation, and thus the extraction can be computed in TC^0, and we refer the reader to the technical section for details.

Now, given a hash value c, instead of computing the naive the local opening to x_{i^*} for some index i^* (of size proportional to the arity of the tree), we compute an "effective" sibling $(g'', h'') = (\prod_{j \neq i^*} g_j^{\widetilde{x}_j}, \prod_{j \neq i^*} h_j^{\widetilde{x}_j})$, and use this as the proof of local opening to x_{i^*}. Given this effective sibling, the verification of local opening now reduces to checking if $c = (g', h')$ is the same as $(g_{i^*}^{x_{i^*}} \cdot g'', h_{i^*}^{x_{i^*}} \cdot h'')$.

While the above effective sibling idea does indeed satisfy correctness and reduce the cost of local opening to be of size $O(1)$, the security of local opening breaks down. Let us elaborate. An adversary trying to cheat by sending inconsistent local opening can efficiently compute two distinct (pairs of) effective siblings (g'', h'') and $(\widetilde{g}'', \widetilde{h}'')$. Specifically, given a hash value (g', h'), if an adversary wants to open index i^* of the input to two *distinct* values, x_{i^*} and $\widetilde{x}_{i^*}$, then it can compute the effective siblings as

$$(g'' = g'/g_{i^*}^{x_{i^*}}, h'' = h'/h_{i^*}^{x_{i^*}}) \text{ and } (\widetilde{g}'' = g'/g_{i^*}^{\widetilde{x}_{i^*}}, \widetilde{h}'' = h'/h_{i^*}^{\widetilde{x}_{i^*}}).$$

The computation of these inconsistent openings are efficient, and it is easy to see from construction that both proofs verify, allowing for openings to different values. Thus the effective sibling, as defined, cannot function as a proof of local opening since there there is no guarantee that the extracted value corresponds to the value that an adversary provides a local opening for.

Proofs of Well-Formedness. To address this problem, we modify the above construction such that a local opening now includes a succinct proof (of size $\mathsf{poly}(\log N)$) that the "effective" sibling was computed correctly, i.e. a proof of well-formedness of the effective sibling. At a high level, the security of the proof system would allow us to reduce the security of the local opening to the *uncompressed* case.

Note that the statement of correct computation includes the key $((g_1, \cdots, g_N), (h_1, \cdots, h_N))$, and thus any proof system we construct will necessarily need to the local opening verifier to read the statement, which would incur a $O(N)$ cost and thus verification would no longer be succinct. To circumvent this issue, we construct the SEH where the verification of the local opening is in the offline/online model - the offline phase is run *once* by the local opening given the SEH key to produce a short state of size $\mathsf{poly}(\lambda_0)$, and the online verification of the local opening can be done in size (and time) *independent* of the arity of the tree. Further, the offline phase runs in time N^k for some fixed constant k.

Putting it all together, when we set $N = \lambda^d$, we have that the verifier computes the pre-processed state in time λ^{kd}, which is independent of the exact value of T since d is a fixed constant. Since the pre-processing is done once (per level) and depends solely on the keys present in the CRS, it is sufficient to consider the efficiency restriction only on the *online* verification of the local opening. Thus, as required, (i) the proof of local opening and its online verification can be done in time $\mathsf{poly}(\lambda_0)$; and (ii) since the depth of the tree remains constant, we can achieve extraction in TC^0. We next discuss the construction of our proof of well-formedness based on the (sub-exponential) hardness of DDH.

More Details. Our construction of the proof of well-formedness for the effective sibling involves multiple steps, which we describe below. For the sake of brevity, in this overview we will only cover a few of them, and refer the reader to the relevant technical sections for complete details.

Reduction to a Promise Language $\mathcal{L}$: As a first step, we show that proving that the effective sibling has been computed correctly can be reduced to proving that some related statement belongs to an appropriately defined promise language $\mathcal{L}$.

Interactive Proof $(\mathsf{P}, \mathsf{V}^{\mathcal{O}})$ for $\mathcal{L}$: We construct an interactive proof for $\mathcal{L}$ in the model where V has oracle access to a large string $\mathcal{O}$, further makes a single query to this oracle at the end of the interactive proof to determine whether to accept or reject[8]. The verifier in this oracle model is *succinct*, and runs in time $\mathsf{poly}(\lambda, \log N)$. Our protocol design is inspired by the Bulletproof protocol [BBB+18].

Apply Fiat-Shamir to $(\mathsf{P}, \mathsf{V}^{\mathcal{O}})$ We show that our constructed CIH based on DDH can be used to round collapse $(\mathsf{P}, \mathsf{V}^{\mathcal{O}})$ and make it non-interactive. This involves primarily showing that the corresponding $\mathcal{B}$ satisfies the required properties to use the CIH. See Section 5.2.3 for details.

Pre-computing the Oracle $\mathcal{O}$: As a last step, we consider an offline/online verifier that pre-compute this oracle $\mathcal{O}$ (in a deterministic fashion) in the offline phase to produce a small digest that is used in the online phase. At a high level the digest will be the Merkle root over the string $\mathcal{O}$, where $\mathcal{O}$ can be computed deterministically simply given the hash key K. To read a specified location, the prover can recompute $\mathcal{O}$ (since it is deterministic), and provide a Merkle proof matching the digest in possession of the verifier. The location to be read is specified by the verifier's random coins sent during the interactive protocol, which the prover knows since it derives it using the CIH. See Section 5.2.3 for details.

In this overview, we will show how the proof of well-formedness reduces to the promise language, and then present our interactive proof for the promise language. For the other steps, we refer the reader to the technical sections.

We now define the promise language $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$

$$\mathcal{L}_{\mathsf{YES}} := \left\{ (K, v = (g', h')) \;\middle|\; \exists \mathbf{m} \in \{0,1\}^n \text{ s.t. } \mathsf{Hash}(K, \mathbf{m}) = v \right\}$$
$$\mathcal{L}_{\mathsf{NO}} := \left\{ (K, v = (g', h')) \;\middle|\; \exists s \text{ s.t. } \forall j \in [n],\ h_j = g_j^s \wedge (h' \neq (g')^s) \right\}$$

First note that the above languages are indeed disjoint. This is fairly easy to see - when K is such that there exists an s such that for every $j \in \{1, \cdots, n\}$, $h_j = g_j^s$, then by construction, for any $\mathbf{m} \in \{0,1\}^n$, $\mathsf{Hash}(K, m)$[9] computed as $g' = \prod_{j \in [n]} g_j^{m_j}$ and $h' = \prod_{j \in [n]} h_j^{m_j} = \prod_{j \in [n]} (g_j^{m_j})^s$ satisfies $h' = (g')^s$. Thus it cannot simultaneously be true that $\mathsf{Hash}(K, m) = v$ and $h' \neq (g')^s$.

Now, to show that the opening to m_{i^*} at position i^* can be reduced to proving a statement in $\mathcal{L}$, the prover first computes $g'' = \prod_{j \neq i^*} g_j^{m_j}$, $h'' = \prod_{j \neq i^*} h_j^{m_j}$ and sets K' as

$$K' := \begin{bmatrix} g_1 & \cdots & g_{i^*-1} & g_{i*+1} & \cdots & g_N \\ h_1 & \cdots & h_{i^*-1} & h_{i^*+1} & \cdots & h_N \end{bmatrix}.$$

Finally, the prover computes the non-interactive proof Π for $\mathcal{L}$ on the statement $(K', v = (g'', h''))$. The opening is then Π, m_{i^*}, i^*. The verifier checks (i) if the hash $c = (g', h')$ satisfies $g' = g'' \cdot g_{i^*}^{m_{i^*}}$ and $h' = h'' \cdot h_{i^*}^{m_{i^*}}$; (ii) the proof Π verifies. The Completeness follows in a straightforward manner from the description of $\mathcal{L}_{\mathsf{YES}}$. For "soundness", we show extraction correctness, i.e. if K was generated with index i^*, then $\mathsf{Ext}(c, i^*, \mathsf{td}) \neq m_{i^*}$ implies that $(K', v = (g'', h'')) \in \mathcal{L}_{\mathsf{NO}}$. First note that if K was generated on

[8] A reader familiar with the sumcheck protocol may notice a similarity in describing the protocol as such

[9] Although we have discussed Hash to take an input a key K generated for a specific index i^*, one can easily generalize this notion to setting where $i^* = \emptyset$. Note that in this case, one cannot hope to extract from any position.

index i^*, K satisfies the property that for all $j \neq i^*$, $h_j = g_j^s$ and because we exclude the i^*-th column from K to construct K', for each j $h_j = g_j^s$ in K'. We argue extraction correctness in two cases (i) If the extracted value $\widetilde{m}_{i^*} = 0$, then $g'^s = h'$. Thus opening $m_{i^*} = 1$ implies $(g'' \cdot g_{i^*}^{m_{i^*}})^s = h_1'' \cdot h_{i^*}^{m_{i^*}}$. By expanding and cancelling terms, we get that $g''^s = h'' \cdot g$, i.e. $g'' \neq h''$; (ii) if the extracted value $\widetilde{m}_{i^*} = 1$, then $g'^{s_1} \neq h_1'$. Thus opening $m_{i^*} = 1$ implies $(g'' \cdot g_{i^*}^{m_{i^*}})^s \neq h_1'' \cdot h_{i^*}^{m_{i^*}}$, thus $g'' \neq h''$. In both cases, we have $g'' \neq h''$, thus if the adversary attempts to break extraction correctness, $(K', v) \in \mathcal{L}_{\mathsf{NO}}$.

Finally, we present our interactive protocol $(\mathsf{P}, \mathsf{V}^O)$, for $\mathcal{L}$, where we specify the string O once we have described the protocol. We describe a recursive protocol, where we fold a statement of length n (i.e. size of the key K is $O(n)$) to a statement of size n/D. In this overview, we describe a simplified recursive protocol for $D = 2$ while the more general case is presented in the technical section.

Let us set some notation. For $\mathbf{m} \in \mathbb{Z}_p^n$, we represent $\mathbf{m}$ as $(\mathbf{m}_L, \mathbf{m}_R)$, where $\mathbf{m}_L, \mathbf{m}_R \in \mathbb{Z}_p^{n/2}$. Analogously, we partition the key $K = \begin{bmatrix} \mathbf{g}_L & \mathbf{g}_R \\ \mathbf{h}_L & \mathbf{h}_R \end{bmatrix}$. Further, $\mathbf{g}_j^{\mathbf{x}_i} = \prod_{k=1}^{n/2} g_{k+jn/2}^{x_{k+in/2}} \in \mathbb{G}$, and for every scalar $\gamma \in \mathbb{Z}_p$, $\mathbf{g}_{j+1}^{\gamma} = (g_{1+jn/2}^{\gamma}, g_{2+jn/2}^{\gamma}, \cdots, g_{(j+1)n/2}^{\gamma}) \in \mathbb{G}^{n/2}$. Lastly, $\mathbf{g}_{j+1}^{\gamma} \cdot \mathbf{g}_{i+1} = (g_{1+jn/2}^{\gamma} \cdot g_{1+in/2}, g_{2+jn/2}^{\gamma} \cdot g_{2+in/2}, \cdots, g_{(j+1)n/2}^{\gamma} \cdot g_{1+(i+1)n/2}) \in \mathbb{G}^{n/2}$.

As stated earlier, at each step the prover is going to fold K into a key using the random challenge γ sent by the verifier. Let the common input by (K, v), and the prover's auxiliary input be $\mathbf{m}$. The following is the i-th iteration.

1. Prover P computes
   (a) $v_L = (\mathbf{g}_L^{\mathbf{m}_L}, \mathbf{h}_L^{\mathbf{m}_L})$ and $v_R = (\mathbf{g}_R^{\mathbf{m}_R}, \mathbf{h}_R^{\mathbf{m}_R})$.
   (b) $\widetilde{v}_1 = (\mathbf{g}_L^{\mathbf{m}_R}, \mathbf{h}_L^{\mathbf{m}_R})$, $\widetilde{v}_2 = v$, $\widetilde{v}_3 = (\mathbf{g}_R^{\mathbf{m}_L}, \mathbf{h}_R^{\mathbf{m}_L})$.

   send (v_L, v_R) and $(\widetilde{v}_1, \widetilde{v}_2, \widetilde{v}_3)$ to verifier V^O.
2. Verifier V^O
   (a) Checks if $v = v_L \cdot V_R$, and $v = \widetilde{v}_2$.
   (b) Sample $\gamma_i \in \mathscr{C}$

   send γ_i to P.
3. P and V set $v = \widetilde{v}_1 \cdot \widetilde{v}_2^{\gamma_i}, \widetilde{v}_3^{\gamma_i^2}$.
4. P sets $\mathbf{m} = \mathbf{m}_R + \gamma_i \mathbf{m}_L$ and $K = (\mathbf{g}_L \cdot \mathbf{g}_R^{\gamma_i}, \mathbf{h}_L \cdot \mathbf{h}_R^{\gamma_i})$.
5. Repeat until $n = 1$.

Finally, when $n = 1$, the prover sends its reduced witness x in the clear, at which point V^O queries O on its challenges $\gamma_1, \cdots \gamma_r$ and receives a key $K = (g, h) \in \mathbb{G}^2$ and check if $v = (g^x, h^x)$ for the reduced statement v. Thus, we want O to function as essentially a look up table such that for every tuple $\gamma_1, \cdots \gamma_r$, it simply reads the corresponding reduced key in $\mathbb{G}^2$.

Completeness: For completeness, at each step i, note that $v = (g', h')$ where $g' = (\mathbf{g}_L^{\mathbf{m}_R + \gamma_i \mathbf{m}_L} \cdot \mathbf{g}_R^{\gamma_i(\mathbf{m}_R + \gamma_i \mathbf{m}_L)})$, and $h' = (\mathbf{h}_L^{\mathbf{m}_R + \gamma_i \mathbf{m}_L} \cdot \mathbf{h}_R^{\gamma_i(\mathbf{m}_R + \gamma_i \mathbf{m}_L)})$. Thus the new key K and $\mathbf{m}$ are set correctly in the protocol to guarantee completeness.

Soundness: To argue soundness, we denote each $\widetilde{v}_i = (g^{\alpha_i}, g^{\beta_i})$ for some α_i, β_i. This is done since the cheating prover may send arbitrary group elements. But we know that $\widetilde{v}_2 = v$, and thus if $(K, v) \in \mathcal{L}_{\mathsf{NO}}$, then $g^{\beta_2} \neq g^{s\alpha_2}$. The newly computed v,

$$\widetilde{v}_1 \cdot \widetilde{v}_2^{\gamma_i}, \widetilde{v}_3^{\gamma_i^2} = (g'', h'') = (g^{\alpha_1 + \gamma_i \alpha_2 + \gamma_i^2 \alpha_3}, g^{\beta_1 + \gamma_i \beta_2 + \gamma_i^2 \beta_3}).$$

Then $(g'')^s/h'' = g^{(s\alpha_1-\beta_1)+\gamma_i(s\alpha_2-\beta_2)+\gamma_i^2(s\alpha_3-\beta_3)} = g^{P(\gamma_i)}$ where $P(X)$ is a polynomial of degree at most 2. Since $s\alpha_2 \neq \beta_2$, P is not a zero polynomial, and thus $\Pr[P(\gamma_i) = 0] \leq 2/|\mathscr{C}|$. Therefore overly a randomly sampled γ_i, $(g'')^s = h''$ only with probability $2/|\mathscr{C}|$.

Looking ahead, the verifier will need to compute the oracle O as a part of its pre-processing, so we need it to be of size $\mathsf{poly}(n)$. By our description above, the size of O is $|\mathscr{C}|^r$, where r is the total number of rounds, i.e. for every tuple $\in \mathscr{C}^r$, the verifier needs to store a key. In the above description, $D = 2$, $r = \log(n)$, and thus to ensure $|\mathscr{C}|^r$ is of size $\mathsf{poly}(n)$, we can only allow $|\mathscr{C}|$ to be a constant, which results in a high soundness error. Instead, we pick $D = \log(n)$, and $|\mathscr{C}| = \log^2(n)$, this gives us $r = \frac{\log(n)}{\log\log(n)}$ and finally $|\mathscr{C}|^r = n^2$. Finally note that we extend the above description to arbitrary D by partitioning the statement into D parts, and folding them using a higher degree polynomial in γ_i, where one can view the above as folding using a degree 2 polynomial as is evident from the soundness analysis.

Remark 1. *An observant reader may note that since the proof of local opening only satisfies computational soundness, it is possible that the value extracted differs from the local opening sent by a cheating prover. This event happens only with negligible probability, and the extracted value is indeed the unique value (at the specified location) hashed by the cheating prover. Thus in our applications, we condition on the non-occurrence of the event and argue soundness as if it the extracted value and the opened value always match.*

3 Preliminaries

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

We will say that a cryptographic primitive is (T, ρ)-secure if any T time adversary breaks the primitive with probability $\geq \rho$. Thus, for schemes that satisfy "standard" security for a security parameter λ, we say that it is $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$.

3.1 Low-degree Extensions

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

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

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

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

3.2 Decisional Diffie-Hellman Assumption

In the following, we state the decisional Diffie-Hellman (DDH) assumption.

Definition 1 (Decisional Diffie-Hellman)**.** *A prime-order group generator is an algorithm $\mathcal{G}$ that takes the security parameter λ as input, and outputs a tuple $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of prime order $p(\lambda)$, and g is a generator of $\mathbb{G}$.*

Let $\mathcal{G}$ be a prime-order group generator. We say that $\mathcal{G}$ satisfies the DDH assumption if for any n.u. PPT distinguisher $\mathcal{D}$, there exists a negligible function $v(\lambda)$ such that

$$\left|\Pr\left[(\mathbb{G},p,g)\leftarrow\mathcal{G}(1^\lambda),a,b\leftarrow\mathbb{Z}_p:\mathcal{D}(1^\lambda,\mathbb{G},p,g,g^a,g^b,g^{ab})=1\right]-\right.$$
$$\left.\Pr\left[(\mathbb{G},p,g)\leftarrow\mathcal{G}(1^\lambda),a,b,u\leftarrow\mathbb{Z}_p:\mathcal{D}(1^\lambda,\mathbb{G},p,g,g^a,g^b,g^u)=1\right]\right|\leq v(\lambda)$$

Sub-exponential DDH Assumption. In this work, we use the following version of sub-exponential DDH assumption.

We say that $\mathcal{G}$ satisfies the sub-exponential DDH assumption, if there exists a constant $0<c<1$ such that for any non-uniform distinguisher D that runs in time $\lambda^{O((\log\log\lambda)^3)}$, the advantage $v(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

3.3 Correlation Intractable Hash

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

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

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

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

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

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

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

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

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

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

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

Furthermore, if C is a threshold circuit of depth d, then we say R is a d-depth verifiable product relation.

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

Definition 6 (Approximate Product Relation). *We say a relation $R\subseteq\mathcal{X}\times\mathcal{Y}^t$ is an α-approximate product relation, if for any x, there exists subsets $S_{1,x},S_{2,x}\ldots,S_{t,x}\subseteq\mathcal{Y}^t$ such that $R_x=\{y\mid(x,y)\in R\}$ can be written as*

$$|\{i\in[t]\mid y_i\in S_{i,x}\}|\geq\alpha t.$$

Definition 7 (Searchable Relation)**.** *We say a relation $R \subseteq \mathcal{X} \times \mathcal{Y}$ is searchable, if there exists a function $f : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$ such that for every $x \in \mathcal{X}$, we have that $(x, y) \in R$ if and only if $y = f(x)$.*

Definition 8 (Approximate Searchable Relation)**.** *Let $\alpha \in (0, 1)$ be a real and $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ be a relation. We say R is α-approximable by threshold circuits $f_1, f_2, \ldots, f_t : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$, if $R_x = \{y \mid (x, y) \in R\}$ is the following set*

$$|\{i \in [t] \mid y_i = f_i(x)\}| \geq \alpha t.$$

Furthermore, If the depth of the threshold circuits $f_1, f_2, \ldots, f_t$ is bounded by d, then we say R is an α-approximate searchable relation of depth d.

3.4 Hash Tree

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

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

Definition 9 (Hash Tree)**.** *A hash tree scheme* ($\mathsf{HT.Gen}$, $\mathsf{HT.Hash}$, $\mathsf{HT.Read}$, $\mathsf{HT.Write}$, $\mathsf{HT.VerRead}$, $\mathsf{HT.VerWrite}$) *satisfies the following properties:*

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

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

Completeness of Write. *For every $\lambda \in \mathbb{N}, D \in \{0,1\}^L$ for $L \leq 2^\lambda$, $\ell \in [L]$ and $b \in \{0,1\}$ let D' be the string with its ℓ-th location set to b. We have that:*

$$\Pr\left[\begin{array}{l|l} \mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}, \ell, b, \mathsf{rt}', \Pi) = 1 & \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ (\mathsf{tree}', \mathsf{rt}') = \mathsf{HT.Hash}(\mathsf{dk}, D') & (\mathsf{tree}, \mathsf{rt}) \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, D) \\ & (\mathsf{tree}', \mathsf{rt}', \Pi) \coloneqq \mathsf{HT.Write}(\mathsf{tree}, \ell, b) \end{array}\right] = 1$$

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

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

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

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

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

Theorem 4 ([Mer88])**.** *From any family of collision resistant hash functions, one can construct a hash tree scheme.*

3.5 Round-by-Round Soundness

Definition 10 (Round-by-Round Soundness [CCH+19])**.** *Let* $\Pi = (\mathsf{P}, \mathsf{V})$ *be a public-coin interactive protocol for a promise language* $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$*, and* $d(\lambda)$ *be a function of* λ*. Let* $\mathscr{C}$ *be the space of randomness for the verifier's message on each round.*

We say Π *satisfies* $d(\lambda)$*-round-by-round soundness, if there exists a deterministic (possibly inefficient) function* State *that satisfies the following properties.* State *takes as input* (x, trans) *where* $x \in \{0,1\}^*$ *is an instance, and* trans *is a prefix of the transcript of the protocol, and* State *outputs either* Accept *or* Reject*.*

1. *Let* ϕ *be the empty transcript prefix. For any instance* $x \in \mathcal{L}_{\mathsf{YES}}$, $\mathsf{State}(x, \phi) = \mathsf{Accept}$*. For any* $x \in \mathcal{L}_{\mathsf{NO}}$, $\mathsf{State}(x, \phi) = \mathsf{Reject}$*.*
2. *For any instance* x *and transcript prefix* $\mathsf{trans} = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_{i-1}, \beta_{i-1})$*, if* $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$*, then for any (possibly unbounded) adversary* $\mathcal{A}$*,*
$$\Pr\left[\alpha \leftarrow \mathcal{A}(1^\lambda, x, \mathsf{trans}), \beta \leftarrow \mathscr{C} : \mathsf{State}(x, \mathsf{trans}|\alpha|\beta) = \mathsf{Accept}\right] \leq d(\lambda)/|\mathscr{C}|.$$
3. *For any complete protocol transcript* trans*, If* $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$*, then* $\mathsf{V}(x, \mathsf{trans}) = 0$*.*

As observed in [CCH+19], an ℓ round protocol satisfying d round-by-round soundness, also satisfies standard soundness with bound $d\ell/|\mathscr{C}|$. This follows from a simple application of the union bound over each round.

4 CIH for Product Relations

We observe that the parameters of the CIH in [JJ21] can be modified to build the following CIH for α-approximate searchable relations in TC^0.

Theorem 5 (CIH for α-approximate searchable relations in TC^0)**.** *Let* $\{\alpha_\lambda\}_{\lambda\in\mathbb{N}}$ *be a sequence of reals in* $(0,1)$*, and* $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ *be a family of relations, where each relation* $R \in \mathcal{R}_\lambda$ *is an* α_λ*-approximate searchable relations of depth* $O(\log^* \lambda)$ *(Definition 8) defined over* $\mathcal{X}_\lambda \times \mathcal{Y}_\lambda^{t_\lambda}$*.*

Assuming sub-exponential DDH assumption, if the parameters satisfies $t_\lambda \geq \lambda^2/\alpha_\lambda, |\mathcal{Y}_\lambda| \geq 2e/\alpha_\lambda$*, then there exists a CIH for* $\mathcal{R}$ *with running time* $\mathsf{poly}(\lambda, |\mathcal{Y}_\lambda|, n, t_\lambda, |f_\lambda|)$*, where* $n = \log_2 |\mathcal{X}_\lambda|$ *is the input length, and* $|f_\lambda|$ *is an upper bound on the size of the threshold circuits defined in Definition 8.*

Moreover, for any adversary that runs in time $\lambda^{O((\log\log\lambda)^2)}$*, its advantage for the aforementioned CIH is at most* $2^{-\lambda'}$*, where* $\lambda' = \lambda^{O(1/\log\log\lambda)}$*.*

We defer the proof of this lemma to Appendix B.

4.1 Construction

Lemma 1. *Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of relations, where each $R \in \mathcal{R}_\lambda$ is $O(\log^* \lambda)$-depth verifiable product relation (Definition 4) of sparsity ρ_λ defined over $\mathcal{X}_\lambda \times \mathcal{Y}_\lambda^{t_\lambda}$. Let $n = \log_2 |\mathcal{X}_\lambda|$ be the length of bit strings representing $\mathcal{X}_\lambda$.*

Assuming sub-exponential DDH assumption, if $t_\lambda = \Omega(\lambda^3/\log(1/\rho_\lambda))$, there exists a CIH for $\mathcal{R}$ with running time $\mathsf{poly}(\lambda, t_\lambda, n^{O(\log\log n)}, \log|\mathcal{Y}|, |C|, \log(1/\rho_\lambda))$, where C is the circuit that verifies the product relation (See Definition 4).

Furthermore, for any adversary that runs in time $\lambda^{O((\log\log\lambda)^2)}$, its advantage for the aforementioned CIH is at most $2^{-\lambda'}$, where $\lambda' = \lambda^{O(1/\log\log\lambda)}$.

Remark 2. *If the sparsity $\rho = 1 - \epsilon$ for some $\epsilon = o(1)$, then the above parameters requirement on t becomes $t = \Omega(\lambda^3/\epsilon)$. The running time of CIH is $\mathsf{poly}(\lambda, t_\lambda, n^{O(\log\log n)}, \log|\mathcal{Y}|, |C|, 1/\epsilon)$.*

Ingredients. Our construction will use the CIH for α-approximate searchable relations of depth $O(\log^* \lambda)$ $\mathsf{CIH} = (\mathsf{CIH.Gen}, \mathsf{CIH.Hash})$ given by Theorem 5.

Construction. We describe our construction of CIH for small input product relations in Figure 2. As we described in Section 2.1, we divide $[N]$ into $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_{\mathsf{depth}}$, where each $|\mathcal{Y}_i| = 2e\log_2(n+\lambda)$, $\mathsf{depth} = \log_2(n+\lambda)$, and we set $N = \prod_{i\in[\mathsf{depth}]} |\mathcal{Y}_i| = n^{O(\log\log n)}$.

4.2 Proof of Correlation Intractability

Before we start to prove Lemma 1, we first prove the following lemma, which bounds the number of bad challenges in $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_\ell$ by roughly the input length.

Lemma 2. *Let $R \in \mathcal{X} \times \mathcal{Y}^\ell$ be a product relation with sparsity ρ, input length $n = \log_2|\mathcal{X}|$, and let $B = \log_2|\mathcal{X}| + \lambda$. If $\ell \geq \log_{1/\rho}(2eN/B)$ and we sample $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N$ randomly from its challenge space $\mathcal{Y}^\ell$, then the number of the bad challenges in $\mathbf{q}_1, \mathbf{q}_2, \ldots \mathbf{q}_N$ is at most B for any input with overwhelming probability. Namely,*

$$\Pr_{q_1,q_2,\ldots,q_N \leftarrow \mathcal{Y}^\ell}[\forall x \in \mathcal{X} : |\{i \mid (x, q_i) \in R\}| \leq B] \geq 1 - 2^{-\Omega(\lambda)}.$$

Proof. Since we sample $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N$ independently from $\mathcal{Y}^\ell$, we can bound the probability that there exist B elements of them are bad as

$$2^n \cdot \binom{N}{B} \cdot \left(\rho^\ell\right)^B \leq 2^n \cdot \left(\frac{eN}{B}\rho^\ell\right)^B,$$

where 2^n follows from an union bound over all possible inputs, $\binom{N}{B}$ follows from an union bound over all possible choice of the B elements. Since we set $\ell = \log_{1/\rho}(2eN/B)$, and $B = n + \lambda$, we can bound the right hand side of the above inequality as $2^n \cdot (1/2)^B = 2^n \cdot (1/2)^{n+\lambda} = (1/2)^\lambda$. We finish the proof. □

In the rest of the proof, we only need to focus on whether the indices $\{i_j\}_{j\in[L]}$ can lead to a bad $\{\mathbf{q}_{i_j}\}_{j\in[L]}$. Namely, we only need to focus on the following relation

$$R' = \{(x, (i_1, i_2, \ldots, i_L)) \mid (x, \{\mathbf{q}_{i_j}\}_{j\in[L]}) \in R\}.$$

R' is an product relation if $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ is a product relation. Let $R'_x = \{\mathbf{i} \mid (x, \mathbf{i}) \in R'\}$, then we there exists $S'_x{}^{(1)}, S'_x{}^{(2)}, \ldots S'_x{}^{(L)}$ such that $R'_x = S'_x{}^{(1)} \times \ldots \times S'_x{}^{(L)}$.

CIH for Small Input Product Relations

Parameters: $[N] = \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_{\mathsf{depth}}$, $t = \ell \cdot L$.

- $\mathsf{Gen}(1^\lambda)$:
  - Sample N queries from $\mathcal{Y}^\ell$: $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N \leftarrow \mathcal{Y}^\ell$.
  - Sample a series of CIH keys for α-approximate searchable relations in $\mathcal{X}_i \times \mathcal{Y}_i^L$, where $\alpha = 1/\mathsf{depth}, L \geq \lambda^2/\alpha$, and $\mathcal{X}_i = \mathcal{X} \times \mathcal{Y}_1^L \times \ldots \times \mathcal{Y}_{i-1}^L$.

    $$\forall i \in [L], \quad \mathsf{k}_i \leftarrow \mathsf{CIH.Gen}(1^\lambda),$$

  - Output $\mathsf{k} = (\{\mathbf{q}_i\}_{i\in[N]}, \{\mathsf{k}_i\}_{i\in[L]})$.
- $\mathsf{Hash}(\mathsf{k}, x)$:
  - Parse $\mathsf{k} = (\{\mathbf{q}_i\}_{i\in[N]}, \{\mathsf{k}_i\}_{i\in[L]})$.
  - For every $i = 1, 2, \ldots, \mathsf{depth}$, we hash the input x and the hash values we obtained so far.

    $$y_i \leftarrow \mathsf{CIH.Hash}(\mathsf{k}_i, (x, \mathbf{y}_1, \mathbf{y}_2 \ldots, \mathbf{y}_{i-1})).$$

  - Then for every $i \in [\mathsf{depth}]$, we first decompose $\mathbf{y}_i$ as a "row" vector $(y_{i,1}, y_{i,2}, \ldots, y_{i,L})$, where each $y_{i,j} \in \mathcal{Y}_i$ for every $j \in [L]$.

    Then we concatenate those rows to obtain a matrix and then take each column of it. Namely, for each $j \in [L]$, we compose the j-th column of the matrix as $i_j = y_{1,j}||y_{2,j}|| \ldots ||y_{\mathsf{depth},j} \in \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_{\mathsf{depth}}$.
  - We use i_j's as indices to select the queries in $\{\mathbf{q}_i\}_{i\in[N]}$, and output

    $$(\mathbf{q}_{i_1}, \mathbf{q}_{i_2}, \ldots, \mathbf{q}_{i_L}).$$

Figure 2: Description of CIH for small input product relations.

Product Relations $\{R_i\}_i$. For each $i \in [\mathsf{depth}]$, we define the following bad relation $R_i \subseteq \mathcal{X}_i \times \mathcal{Y}_i^L$ for the i-th CIH. Recall that, we define $\mathcal{X}_i = \mathcal{X} \times \mathcal{Y}_1^L \times \ldots \times \mathcal{Y}_{i-1}^L$. For any $(x, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{i-1}) \in \mathcal{X}_i$, let $y_{k,j} \in \mathcal{Y}_k$, $(j \in [L])$ be the j-th coordinate of the vector $\mathbf{y}_k$ for every $k \in [i-1]$. We say $y_{i,j}$ is bad, if it does not half the number of bad challenges. Specifically, we define $S_i^{(j)}$ as the sets of bad challenges as follows.

$$S_i^{(j)} = \left\{ y \in \mathcal{Y}_k \mid \mathsf{BadCnt}(j, x, y_{1,j}, y_{2,j}, \ldots, y_{i-1,j}, y) > \mathsf{BadCnt}(j, x, y_{1,j}, y_{2,j}, \ldots, y_{i-1,j})/2 \right\},$$

where $\mathsf{BadCnt}(j, x, \mathbf{r})$ counts how many bad challenges has the prefix $\mathbf{r} \in \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_i$. Namely, it is defined as the cardinality of the following set.

$$\left\{ \mathbf{r}' \in \mathcal{Y}_{i+1} \times \ldots \times \mathcal{Y}_{\mathsf{depth}} \mid \mathbf{r}||\mathbf{r}' \in {S'_x}^{(j)} \right\}.$$

We define R_i as the product relation characterized by

$$S_i^{(1)} \times S_i^{(2)} \times \ldots \times S_i^{(L)}.$$

Approximate Product Relations $\{R_i^\alpha\}_i$. Let $\alpha = 1/\mathsf{depth}$. We define the α-approximate product relation R_i^α as the α-approximate relation of R_i. That is, R_i^α contains exactly all $(x, \mathbf{y})$ where there exists some $\mathbf{y}'$ such that $(x, \mathbf{y}') \in R_i$ and at least α-fraction of coordinates of $\mathbf{y}$ and $\mathbf{y}'$ agrees.

We first show that R_i are searchable by the almost same depth of threshold circuits as R up to a constant factor.

Lemma 3. *If the product relation R is verifiable in threshold circuit of depth d, then for any $i \in [\mathsf{depth}]$, R_i is searchable by threshold circuits of depth $O(d)$ and size $\mathsf{poly}(L, N, |C|)$, where C is the circuit that verifies the product relation R.*

Proof. Note that this lemma starts from an efficiently *verifiable* relation R and claims that the product relation described above is *searchable* in the same threshold circuit class. To prove this lemma, for each $i \in [\mathsf{depth}]$, we build the following circuit f^i. f^i takes $x \in \mathcal{X}$, and $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{i-1}$ as input, where $\mathbf{y}_i \in \mathcal{Y}_i^L$, and it outputs a vector $\mathbf{y}^* = (y_1^*, y_2^*, \ldots, y_L^*)$. For each $j \in [L]$, we enumerate all possible $y_{i,j}$ and if

$$\mathsf{BadCnt}(j, x, y_{1,j}, \ldots, y_{i-1,j}, y_{i,j}) > \mathsf{BadCnt}(j, x, y_{1,j}, \ldots, y_{i-1,j})/2,$$

then we set $y_j^* = y_{i,j}$. Otherwise, we let $y_j^* = \perp$. Finally, we have f^i output $\mathbf{y}^*$. Note that for each $j \in [L]$, f^i only sets y_j^* once, since $S^i_{x,j}$ can only contain at most one element.

Next, we show that f^i can be computed by threshold circuits of depth $O(d)$. Since the comparison function and the division is known to be in TC^0, we only need to show that $\mathsf{BadCnt}(j, x, \mathbf{r})$ can be computed by threshold circuit of depth $O(d)$. To achieve this, we compute $\mathsf{BadCnt}(j, x, \mathbf{r})$ by enumerating all $\mathbf{r}' \in \mathcal{Y}_{i+1} \times \ldots \times \mathcal{Y}_{\mathsf{depth}}$ in parallel and checking whether $(\mathbf{r}, \mathbf{r})$ is a bad challenge (i.e. $\mathbf{r}||\mathbf{r}' \in S'^{(j)}_x$). This checking involves in selecting a query $\mathbf{q}_{\mathbf{r}||\mathbf{r}'}$ among $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N$ according to the index $\mathbf{r}||\mathbf{r}'$ and check whether $\mathbf{q}_{\mathbf{r}||\mathbf{r}'}$ is bad. The selection can be computed in TC^0 and the checking can be done by threshold circuit of depth d. Hence, we finish the proof. □

Lemma 4 (Covering Lemma)**.** *Suppose we choose the parameters $\mathsf{depth} > \log_2 B + 1$, where $B \geq \max_{j,x}\{\mathsf{BadCnt}(j, x)\}$ is an upper bound on the number of bad challenges.*

For any $x \in \mathcal{X}$, and $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{\mathsf{depth}}$ with $\mathbf{y}_i \in \mathcal{Y}_i^L$ for every $i \in [\mathsf{depth}]$, let $i_1, i_2, \ldots, i_L \in [N]$ be the "columns" of the matrix whose rows are $\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_{\mathsf{depth}}$. If $(x, (i_1, i_2, \ldots, i_L)) \in R'$, then there exists $i^* \in [\mathsf{depth}]$ such that $\mathbf{y}_{i^*}$ is bad in $R^\alpha_{i^*}$. That is, $((x, \mathbf{y}_1, \ldots, \mathbf{y}_{i^*-1}), \mathbf{y}_{i^*}) \in R_{i^*}$.

Proof. We first show that, for any $j \in [L]$, if the challenge $(y_{1,j}, y_{2,j}, \ldots, y_{\mathsf{depth},j})$ is bad, i.e. $(y_{1,j}, y_{2,j}, \ldots, y_{\mathsf{depth},j}) \in S'^{(j)}_x$, then there must be an index $i^* \in [\mathsf{depth}]$ such that $y_{i^*,j}$ is bad, (i.e. $y_{i^*,j} \in S^{(j)}_{i^*}$).

To see this, if there exists an $i^* < \mathsf{depth}$ such that $y_{i^*,j}$ is bad, then we finish the proof. Otherwise, suppose $y_{k,j}$ is not bad for every $k < \mathsf{depth}$, we will show that $y_{\mathsf{depth},j}$ is bad. This is because $y_{k,j} \notin S^{(j)}_k$ implies that the challenge $y_{k,j}$ half the number of the bad challenges, and hence we have

$$\mathsf{BadCnt}(j, x, y_{1,j}, \ldots y_{k,j}) \leq \mathsf{BadCnt}(j, x, y_{1,j}, \ldots, y_{k-1,j})/2,$$

for every $k < \mathsf{depth}$. Note that we have an upper bound of bad challenges B. Hence, iteratively applying the above inequality, we derive that $\mathsf{BadCnt}(j, x, y_{1,j}, \ldots, y_{\mathsf{depth}-1,j}) \leq 1$, and according to our definition of BadCnt, $\mathsf{BadCnt}(j, x, y_{1,j}, \ldots, y_{\mathsf{depth},j}) = 1$ because $(y_{1,j}, \ldots, y_{\mathsf{depth},j})$ is bad. Hence, $y_{\mathsf{depth},j}$ is bad since it does not half the number of bad challenges.

Now, for L indices $i_1, i_2, \ldots, i_L$, each index $i_j = (y_{1,j}, y_{2,j}, \ldots, y_{\mathsf{depth},j})$ contains at least one bad challenge. Hence, in total there are L bad challenges among all $\{y_{k,j}\}_{k \in [\mathsf{depth}], j \in [L]}$. By pigeonhole principle, there exists an $i^* \in [\mathsf{depth}]$ such that $\{y_{i^*,j}\}_{j \in [L]}$ contains at least L/depth elements, which is at least $\alpha = 1/\mathsf{depth}$-fraction of the coordinates. Hence, we finish the proof. □

Put Things Together. Next, we prove Lemma 1.

Proof of Lemma 1. For any n.u. PPT adversary $\mathcal{A}$ for the CIH construction in Figure 2, we build the following adversary $\mathcal{A}_i$ to break the correlation intractability for the i-th CIH.

$\mathcal{A}_i(1^\lambda, \mathsf{k}_i)$ generates other CIH keys $\{\mathsf{k}_j\}_{j\in[\mathsf{depth}]\setminus\{i\}}$ for $\mathcal{A}$, and invokes $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$. Then it computes $\mathbf{y}_1, \mathbf{y}_2, \ldots$ to the i-th level, as follows. For every $j = 1, 2, \ldots, i-1$, compute

$$\mathbf{y}_j := \mathsf{CIH.Hash}(\mathsf{k}_i, (x, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{i-1})).$$

Output $(x, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_i)$.

Parameters. We first examine the parameters requirements of the underlying CIH is satisfied. For the underlying CIH for α-approximate searchable relation, we require $L \geq \lambda^2/\alpha = \lambda^2 \cdot \mathsf{depth} = \widetilde{O}(\lambda^2)$. For $|\mathcal{Y}_i|$, we require $|\mathcal{Y}_i| \geq 2e/\alpha = 2e \cdot \mathsf{depth}$. Hence, $N = |\mathcal{Y}_i|^{\mathsf{depth}} = \lambda^{O(\log\log\lambda)}$, since we take $\mathsf{depth} = \log_2(n+\lambda)$. For the parameter t, our construction requires it to be at least $\ell \cdot L = \log_{1/\rho}(2eN/B) \cdot \widetilde{O}(\lambda^2) = \widetilde{O}(\lambda^2)/\log(1/\rho)$. Since we require $t \geq \Omega(\lambda^3/\log(1/\rho))$, this requirement is satisfied.

By Lemma 2, the condition of Lemma 4 holds. By Lemma 4, if adversary $\mathcal{A}$ attacks successfully, i.e. $(x, \mathsf{Hash}(\mathsf{k}, x)) \in R$, then there exists an index $i^* \in [\mathsf{depth}]$ such that α-fraction of the entries in $\mathbf{y}_{i^*}$ is bad. By Lemma 3, this implies that the adversary $\mathcal{A}_k$ attacks successfully. Hence, we have

$$\Pr[\mathcal{A} \text{ succeed}] \leq \sum_{i\in[\mathsf{depth}]} \Pr[\mathcal{A}_i \text{ succeed}].$$

Since $\mathsf{CIH.Hash}(\mathsf{k}_i, \cdot)$ are correlation intractable, $\Pr[\mathcal{A}_i \text{ succeed}]$ are negligible for all $i \in [\mathsf{depth}]$. Hence, $\Pr[\mathcal{A} \text{ succeed}] \leq \mathsf{negl}(\lambda)$. We finish the proof. □

4.3 Extensions

Lemma 5 (CIH for Approximate Product Relations)**.** *Let $\{\beta_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of reals in $(0,1)$. Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of relations, where each relation $R \in \mathcal{R}_\lambda$ is a β_λ-approximate product relation of depth $O(\log^*\lambda)$ defined over $\mathcal{X}_\lambda \times \mathcal{Y}_\lambda^{t_\lambda}$ with sparsity $\rho_\lambda < \beta_\lambda$.*

Assuming sub-exponential DDH assumption holds and $t_\lambda = \Omega(\lambda^3) \cdot (1-\rho_\lambda)/(\beta_\lambda - \rho_\lambda)^3$, then there exists a CIH for $\mathcal{R}$ with running time $\mathsf{poly}\left(\lambda, \lambda^{O(\log\log\lambda)+O(\log\frac{1-\rho_\lambda}{\beta_\lambda-\rho_\lambda})}, t_\lambda, |C|\right)$, where C is the circuit that verifies the product relation R.

Moreover, for any adversary that runs in time $\lambda^{O((\log\log\lambda)^2)}$, its advantage is bounded by $2^{-\lambda'}$, where $\lambda' = \lambda^{O(1/\log\log\lambda)}$.

Proof Sketch. The proof is almost the same as Lemma 1, and the construction of the CIH is almos the same as Figure 2, except we modify the parameters and the bad relations R_i, R_i^α as follows.

We say $\mathbf{q}_i$ as bad, if it contains at least $(\beta_\lambda+\rho_\lambda)/2$-fraction of bad coordinates. By Hoeffding's inequality, if we choose $\ell \geq 4\ln(2eN/B)/(\beta_\lambda - \rho_\lambda)^2$, then the number of bad challenges among $\mathbf{q}_1, \ldots, \mathbf{q}_N$ can be bounded by $B = n + \lambda$ with probability $2^{-\lambda}$. That is,

$$\begin{aligned}\Pr\left[\mathbf{q}_1, \mathbf{q}_2, \ldots \mathbf{q}_N \leftarrow \mathcal{Y}^\ell : \exists x, |\{i \in [N] : (x, \mathbf{q}_i) \in R'\}| > B\right] &\leq 2^n \cdot \binom{N}{B}\left(\exp((-\frac{\beta_\lambda - \rho_\lambda}{2})^2\ell)\right)^B \\ &\leq \left(\frac{eN}{B}e^{-(\frac{\beta_\lambda-\rho_\lambda}{2})^2\ell}\right)^B 2^n\end{aligned}$$

Since we choose $\ell \geq 4\ln(2eN/B)/(\beta_\lambda - \rho_\lambda)^2$, the right hand side can be upper bounded by $2^{-B}2^n = 2^{-\lambda}$.

We claim that for any $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_L) \in (\mathcal{Y}^\ell)^L$, where each $\mathbf{y}_i$ is in $\{\mathbf{q}_1, \ldots, \mathbf{q}_N\}$, if $\mathbf{y}$ is bad for $R \in \mathcal{R}_\lambda$, i.e. (if we treat $\mathbf{y}$ as a vector in $\mathcal{Y}^{\ell \times L}$, then β_λ-fraction of its entries are bad), then at least $(\beta_\lambda - \rho_\lambda)/2(1-\rho_\lambda)$-fraction of $\mathbf{y}_1, \ldots, \mathbf{y}_L$ is bad. The claim can be proved as follows. Let x be the fraction of bad vectors among $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N\}$, then the total fraction of bad entries in $\mathbf{y}$ is at most

$$(1-x) \cdot \frac{\beta_\lambda + \rho_\lambda}{2} + x.$$

Since there are β_λ-fraction of bad entries among all entries of $\mathbf{y}$, we can lower bound the above formula by β_λ. Solving this inequality for x, we obtain $x \geq (\beta_\lambda - \rho_\lambda)/2(1 - (\beta_\lambda + \rho_\lambda)/2) \geq (\beta_\lambda - \rho_\lambda)/2(1 - \rho_\lambda)$.

Hence, we further modify the α in our CIH construction (Figure 2) as $\alpha = (\beta_\lambda - \rho_\lambda)/(2(1-\rho_\lambda) \cdot \mathsf{depth})$, and $N = |\mathcal{Y}_i|^{\mathsf{depth}} = (2e/\alpha)^{\mathsf{depth}} = \lambda^{O(\log\log\lambda) + O(\log \frac{1-\rho_\lambda}{\beta_\lambda - \rho_\lambda})}$. Hence, once we choose $t \geq \ell \cdot L$ where $L \geq \lambda^2/\alpha$ due to Theorem 5, then it's large enough for our modified CIH construction. That is, $t \geq \widetilde{\Omega}(\lambda^2)(1-\rho_\lambda)/(\beta_\lambda - \rho_\lambda)^3$. □

We adapt the following lemma implicit in [HLR21].

Lemma 6 (CIH for Round-by-Round Protocols)**.** *For any $2r+1$-rounds public-coin interactive protocol Π with round-by-round soundness δ (Definition 10). Let t be the number of parallel repetitions, and $\alpha = \frac{1}{r}(1 - 1/t)$ be a real, then we can instantiate the Fiat-Shamir transformation for the t-fold parallel repetition Π^t with CIH for α-approximate product relations $R_i \subseteq \mathcal{X}_i \times \mathcal{Y}^t$, where R_i contains all bad challenges in the i-th round. Namely,*

$$((x, \tau_i), \{\beta_j\}_{j \in [t]}) \in R_i, \text{ if and only if there exists at least } \alpha \text{ fraction of } j \in [t] \text{ such that } \mathsf{State}(x, \tau_i|_j) = \mathsf{Reject} \wedge \mathsf{State}(x, \tau_i|_j \mid \beta_j) = \mathsf{Accept}.$$

where $\tau_i = (\alpha_1, \beta_1, \ldots, \beta_{i-1}, \alpha_i)$ contains the transcript of the t-fold repetition protocol Π^t in the first $(2i-1)$-rounds, and $\tau_i|_j$ is the partial transcript restricted to the j-th parallel execution.

Combining Lemma 6 and Lemma 5, we obtain the following corollary.

Corollary 1 (Fiat-Shamir for Round-by-Round Protocols via CIH)**.** *Let Π be a $2r+1$-rounds public-coin interactive protocol Π with round by round soundness $\delta < 1/r$, where the* State *function can be computed by threshold circuits of depth $O(\log^*)$, and let t be the number of parallel repetitions with $t = \Omega(\lambda^3) \cdot (1-\delta)/(\frac{1}{r} - \delta)^3$.*

Assuming sub-exponential DDH assumption, there exists a CIH family that instantiate the Fiat-Shamir transformation for t-fold parallel repetition protocol Π^t. The CIH runs in time $\mathsf{poly}\left(\lambda, t, \lambda^{O(\log\log\lambda) + O(\log \frac{1-\delta}{\frac{1}{r} - \delta})}\right)$ *and can take input of length* $\mathsf{poly}(\lambda)$.

Moreover, for any adversary that runs in time $\lambda^{O((\log\log\lambda)^2)}$, its advantage for the soundness of the resulting non-interactive protocol is at most $2^{-\lambda'}$, where $\lambda' = \lambda^{O(1/\log\log\lambda)}$.

5 Somewhere Extractable Hash from DDH

In this section, we define and construct a somewhere extractable hash (with additional properties) assuming the sub-exponential hardness of DDH. We first define the standard notion of a somewhere extractable hash, then specify the additional properties required for our applications, and finally describe our construction.

5.1 Definition

In this subsection, we define a somewhere extractable hash [HW15], along with the required additional. A somewhere extractable hash has a key with two computationally indistinguishable modes: (i) In the *normal mode*, the key is *uniformly random*; and (ii) in the *trapdoor mode*, the key is generated according to a subset S denoting the coordinates of the message to be hashed.
Furthermore, a standard somewhere extractable hash require the following properties.

Efficiency: We require that the size of the CRS and hash roughly grows with $|S|$.

Extraction: The trapdoor mode hash key is associated with a trapdoor td, such that given the trapdoor, one can extract the message on coordinates in S. Note that the extraction implies the statistical binding property for the coordinates in S.

Local Opening: We allow the prover to generate a *local opening* for any single coordinate of the message. The local opening needs to have a small size, which only grows poly-logarithmically with the total length of the message. Moreover, we require that the value from the local opening should be consistent with the extracted value.

For our application, we require some additional properties specified below.

Low-depth Extraction: We require that the extraction circuit for the hash can be implemented in TC^0. Specifically, we split the extraction into an *pre-processing* and *online* extraction phase, where the trapdoor td is used *only* in the online phase. We require the online phase of the extraction to be implemented in TC^0.

Large inputs: We require that the somewhere extractable hash scheme to support input sizes that are super-polynomial in the security parameter, i.e. $N = \lambda^{\omega(1)}$. Further, we will continue to require that the local opening is small for such inputs. In a manner similar to the extraction, we will split the local opening verification opening into a pre-processing and online phase, and require that the online verification is *small*.

We now move to the formal definition.

Definition 11 (Somewhere Extractable Hash)**.** *A somewhere extractable hash scheme for large inputs with extraction in* TC^0 *is a tuple of algorithms* $\mathsf{SEHash} = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{PreVer}, \mathsf{OnlineVer}, \mathsf{PreExt}, \mathsf{OnlineExt})$ *described below.*

- $\mathsf{Gen}(1^\lambda, 1^N, 1^{|S|})$**:** *On input a security parameter λ, the length of the message N, and the size of a subset $S \subseteq [N]$, the "normal mode" key generation algorithm outputs a* uniformly random *hash key K.*
- $\mathsf{TGen}(1^\lambda, 1^N, S)$**:** *On input a security parameter λ, the length of the message N, an extraction subset $S \subseteq [N]$, the "trapdoor mode" key generation algorithm outputs a hash key K^* and a trapdoor* td.
- $\mathsf{Hash}(K, \mathbf{m} \in \{0,1\}^N)$**:** *On input the hash key K, a vector $\mathbf{m} = (m_1, m_2, \ldots, m_N) \in \{0,1\}^N$ it outputs a hash c.*
- $\mathsf{Open}(K, \mathbf{m}, i)$**:** *On input the hash key K, a vector $\mathbf{m} = (m_1, m_2, \ldots, m_N) \in \{0,1\}^N$, an index $i \in [N]$, the opening algorithm outputs a* local opening *π_i to m_i.*
- $\mathsf{PreVer}(K)$**:** *On input the hash key K, the pre-processing algorithm for the local verification outputs a short state st_π of size $\mathsf{poly}(\lambda, \log N)$*
- $\mathsf{OnlineVer}(c, m_i, \pi_i, \mathsf{st}_\pi)$**:** *On input a hash c, a bit $m_i \in \{0,1\}$, a local opening $\pi_i \in \{0,1\}^{\mathsf{poly}(\lambda, \log N)}$ and the pre-processed state st_π, the verification algorithm decides to accept (output 1) or reject (output 0) the local opening.*

$\mathsf{PreExt}(1^\lambda, c)$: *On input the security parameter λ and a hash c, output a* pre-processed *value c' that is to be used for* online extraction.

$\mathsf{OnlineExt}(c', S, \mathsf{td})$: *On input the pre-processed hash c' and a trapdoor* td *and the trapdoor* td *generated by the trapdoor key generation algorithm* TGen *with respect to the subset S, the online extraction algorithm outputs an S bit string $m_S \in \{0,1\}^{|S|}$.*

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

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

Succinct Hash. *The size of the hash c is bounded by* $\mathsf{poly}(\lambda, |S|, \log N)$.

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

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

Key Indistinguishability. *For any non-uniform T-time adversary $\mathcal{A}$ where $T = \mathsf{poly}(\lambda, N)$, there exists a negligible function $\nu(T)$ such that*

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

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

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

Computational Extraction Correctness. *For any subset $S \subseteq [N]$, any trapdoor key $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, for any adversary* PPT *adversary $\mathcal{A}$:*

$$\Pr\left[\begin{array}{l} i^* \in S \\ \wedge\ \mathsf{Verify}(K, c, m_{i^*}, i^*, \pi_{i^*}) = 1 \\ \wedge\ \mathsf{Ext}(c, \mathsf{td})|_{i^*} \neq m_{i^*} \end{array} \middle| \; (c, m_{i^*}, i^*, \pi_{i^*}) \leftarrow \mathcal{A}(\lambda, K) \right] \leq \mathsf{negl}(\lambda)$$

Since the extracted value $\mathsf{Ext}(c, \mathsf{td})|_{i^*}$ is unique, except with negligible probability, if the proof is accepting, the opening and extracted values are consistent.

We note that although this doesn't achieve somewhere statistical *extraction, we will see that it suffices for our use case. Intuitively, this follows from the fact that the extracted value is* unique*, and the computational aspect only comes from the proof of local opening. Thus, we will extract from the hash, and argue that for* any *computationally bounded adversary, except with negligible probability, the extracted value is indeed the value that the adversary opens to. And thus we shall simply use the extracted value in the analysis of our application.*

TC^0 **Extraction.** *We require the online extraction algorithm* $\mathsf{OnlineExt}$ *to be computable in* TC^0*, whereas we allow* PreExt *(without knowledge of the trapdoor* td*), to be computable in* $\mathsf{poly}(\lambda)$ *depth.*

As described in the technical overview (Section 2), it is *both* additional requirements that necessitates a new construction. Even removing one of the requirements would allow us to directly invoke constructions in prior works [OPWW15, DGI+19] from *polynomial* hardness of DDH.

Remark 3. *We note that we do not impose any efficiency requirements on the pre-computation algorithm* PreVer. *In fact, in our construction it will be the case that* PreVer *runs in time* $\mathsf{p}(m)$ *for some fixed polynomial* $\mathsf{p}(\cdot)$*, where* m *is fixed independent of* N *such that* $\log_m(N)$ *is a constant. This* weak *efficiency guarantee from* PreVer *will be sufficient for our applications.*

We shall prove the following theorem in the subsequent sections.

Theorem 6. *There exists a construction of somewhere extractable hash for large inputs with extraction in* TC^0 *assuming the sub-exponential hardness of* DDH.

5.2 Construction

We construct the somewhere extractable hash in two steps: (i) *base case:* construct a somewhere extractable hash for N bit messages in the *pre-processing model* where the online verification is $\mathsf{poly}(\lambda, \log N)$ while the pre-processing takes time $O(N^z)$ for some fixed constant z; and then (ii) *extend to a tree* bootstrap the construction by constructing an m-ary tree of depth $\log_m(N)$ to get the pre-processing down to $O(m^z)$, and online verification to be $\mathsf{poly}(\lambda, \log_m(N), \log m)$. Looking ahead, both N and m will both be super-polynomial in λ such that $\log_m(N) = O(1)$. This will ensure that the online verification will continue to be $\mathsf{poly}(\lambda, \log N)$ while the other components will be polynomial in m. We will fix m independent of N.

5.2.1 Construction: Base Hash

We construct below a somewhere extractable hash scheme in the pre-processing model for the base construction. Recall, that here for hashing strings of length N, we allow the pre-processing to depend on N^z for some fixed constant z.

Our construction closely follows the construction of the trapdoor hash based on DDH [DGI+19], but to achieve the properties discussed above, namely the succinct local opening in the pre-processing model, we augment the construction with a new *proof* of opening. We start with a discussion of all the algorithms except those pertaining to the proving of local opening, which we describe and prove separately. For the subsequent discussion, let $\ell := |S|$. Let the group be generated on λ_0 where $\lambda = \lambda_0^{O(\log\log\lambda_0)}$ is the security parameter for the scheme.

$\mathsf{Gen}(1^\lambda, 1^N, 1^{|S|})$**:** Sample $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^{\lambda_0})$, $s_1, \cdots, s_\ell \leftarrow\!\$\ \mathbb{Z}_p$, and set the hash key $K \in \mathbb{G}^{(\ell+1)\times N}$ to be,

$$K = \begin{bmatrix} g_1 & g_2 & \cdots & g_N \\ h_{1,1} & h_{1,2} & \cdots & h_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ h_{\ell,1} & h_{\ell,2} & \cdots & h_{\ell,N} \end{bmatrix}$$

where $\forall j \in [N]$, $g_j \leftarrow\!\$\ \mathbb{G}$, and $\forall i \in [\ell]$, $j \in [N]$, $h_{i,j} = g_j^{s_i}$. Output K.

$\mathsf{TGen}(1^\lambda, 1^N, S)$**:** Sample $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^{\lambda_0})$, $s_1, \cdots, s_\ell \leftarrow\!\$\ \mathbb{Z}_p$, where $\ell := |S|$. Let $S = \{j_1^*, \cdots, j_\ell^*\}$. Set the hash key $K \in \mathbb{G}^{(\ell+1)\times N}$ to be,

$$K^* = \begin{bmatrix} g_1 & g_2 & \cdots & g_N \\ h_{1,1} & h_{1,2} & \cdots & h_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ h_{\ell,1} & h_{\ell,2} & \cdots & h_{\ell,N} \end{bmatrix}$$

where $\forall j \in [N]$, $g_j \leftarrow\!\$\ \mathbb{G}$, and $\forall i \in [\ell]$, $j \in [N]$,

$$h_{i,j} = \begin{cases} g_j^{s_i} \cdot g & \text{if} j = j_i^* \\ g_j^{s_i} & \text{otherwise.} \end{cases}$$

let $\mathsf{td} := (s_1, \cdots, s_\ell)$. Output (K^*, td).

$\mathsf{Hash}(K, \mathbf{m} = (m_1, \cdots, m_N) \in \{0,1\}^N)$**:** Compute the hash $c \in \mathbb{G}^{(\ell+1)}$ as $c := (g', h'_1, \cdots, h'_\ell)$, where $g' := \prod_{j\in[N]} g_j^{m_j}$, and $\forall i \in [\ell]$, $h'_i := \prod_{j\in[N]} h_{1,j}^{m_j}$. Note that since we don't care about our hash being hiding, we ignore the random coins, i.e. Hash is a deterministic algorithm in our construction.

$\mathsf{PreExt}(1^\lambda, c)$**:** Parse c as $(g', h'_1, \cdots, h'_\ell)$ and output $c' = (\{g'^{2^k}\}_{k=0}^{[\lambda_0]-1}, h'_1, \cdots, h'_\ell)$

$\mathsf{OnlineExt}(c', S, \mathsf{td})$**:** Parse td as $(s_1, \cdots, s_\ell)$ and let $S = \{j_1^*, \cdots, j_\ell^*\}$. We compute m_S as follows: for each $i \in [\ell]$,

$$m_{j_i^*} = \begin{cases} 0 & \text{if } g'^{s_i} = h'_i \\ 1 & \text{otherwise} \end{cases}$$

Output m_S.

Efficiency. Note that the size of the keys are $O(\lambda_0 \cdot N \cdot \ell)$, the size of the commitment $O(\lambda_0 \cdot \ell)$ and the size of the pre-processed value c' is $O(\ell \cdot \mathsf{poly}(\lambda_0))$. The time taken to compute the commitment is $\mathsf{poly}(\lambda_0, N, \ell)$ while the extraction can be done $\mathsf{poly}(\lambda_0, \ell)$. Further note that for each i, g'^{s_i} can be computed in TC^0 given s_i and $\{g'^{2^k}\}_{k=0}^{[\lambda_0]-1}$, contained in the *pre-computed* hash c'. This can be done simply by looking at the binary representation of s, and multiplying the relevant g'^{2^k} - which can be done in TC^0, thus ensuring that OnlineExt can be computed in TC^0.

Lemma 7. *The above scheme satisfies key indistinguishability.*

The key indistinguishability of the scheme follows from the security of DDH, argued in a manner identical to [DGI+19]. Here we provide a sketch for the case that $\ell = 1$, and can be extended easily to the more general case. We in fact prove that the key output in the trapdoor mode on input i is indistinguishable from from $U \leftarrow\$ \mathbb{G}^{2\times N}$. Given a tuple (g^a, g^b, g^c), the key $K = \begin{bmatrix} (g^a)^{r_1} & \cdots & (g^a)^{r_i} & \cdots & (g^a)^{r_N} \\ (g^c)^{r_1} & \cdots & (g^c)^{r_i} \cdot g & \cdots & (g^c)^{r_N} \end{bmatrix}$. If $g^c = g^{ab}$, then K has the same distribution as the key generated by TGen on input i, else it is random.

Finally note that since the security parameter for the group in λ_0, we require $(\mathsf{poly}\left(\lambda_0^{\log\log\lambda_0}\right), \mathsf{negl}\left(\lambda_0^{\log\log\lambda_0}\right))$-security from DDH in the chosen group.

5.2.2 Proof of Local Opening

In this section, we shall describe the proof of local opening in the common references string (CRS) model. As described earlier, we require strong efficiency properties from the proof verification algorithm. We will first define below a promise language $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$ for which we shall construct a non-interactive argument system in the CRS model. But before we construct this non-interactive argument, we describe how such an argument system, in conjunction with a hash tree scheme, can be used to achieve computational extraction correctness. We then build a non-interactive argument for $\mathcal{L}$.

Our non-interactive argument for $\mathcal{L}$ is going to involve multiple steps that we outline here, and refer the reader to the relevant technical sections for the details. As a first step, we build an interactive *proof*[10] $(\mathsf{P}, \mathsf{V}^L)$ for $\mathcal{L}$ where the verifier V^L will have access to an oracle L, and will otherwise be extremely efficient - the efficiency will in fact be very close to the requirement from OnlineVer. Our interactive protocol is inspired by the Bulletproof protocol [BBB+18], but our parameters are such that soundness of the protocol is quite large. We thus parallel repeat and show that it satisfies the necessary requirements to apply the Fiat-Shamir transformation to make it non-interactive. We then apply Lemma 10 to obtain a non-interactive

[10] statistically sound interactive protocol

argument system $(\mathsf{P}_{\mathsf{FS}}, \mathsf{V}^L_{\mathsf{FS}})$ in the CRS (and oracle) model. Finally, we show how to remove the need for an oracle access in the non-interactive argument by the way of verifier pre-processing, and the use of hash trees. The final non-interactive argument we denote by $(\mathsf{P}', (\mathsf{V}'_1, \mathsf{V}'_2))$ for $\mathcal{L}$ with CRS $\mathsf{crs}_{\mathcal{L}}$. V'_1 will run in time $N^z \mathsf{poly}(\lambda_0, \lambda)$ for a fixed constant z. The output produced is of size $\mathsf{poly}(\lambda)$. The running time of V'_2 is $\mathsf{poly}(\lambda)$. Further, V'_1 is a *deterministic* procedure.

Remark 4. *Note that to handle arbitrary polynomial many instances T, we cannot choose a fixed constant depth. Since the number of keys correspond the depth of the hash tree, we set the number of keys to a super-constant depth, $\log^* \lambda$. When running the protocol for any polynomial T, the prover and verifier will select the appropriate constant c such that $T \leq \lambda^c$. But since the* CIH *needs to be able to handle arbitrary polynomial computation, the* CIH *is constructed to allow $\mathcal{B}_{C,c}$ that can be verified by threshold circuits of depth $O(\log^* \lambda)$. It is a technicality of our construction, and explains the reasoning behind our* CIH *theorem allowing for $O(\log^* \lambda)$ verification, but we will not discuss it further in this overview, and assume for further discussion $\mathcal{B}$ verifiable via constant depth threshold circuits, i.e. TC^0.*

Promise Language. We start by defining the following promise language $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$ where $K = \begin{bmatrix} g_1 \cdots g_n \\ h_1 \cdots h_n \end{bmatrix}$, and $v = (g', h')$

$$\mathcal{L}_{\mathsf{YES}} := \left\{ (K, v) \in \mathbb{G}^{2\times n} \times \mathbb{G}^2 \;\middle|\; \exists \mathbf{m} \in \{0,1\}^n \text{ s.t. } \mathsf{Hash}(K, \mathbf{m}) = v \right\}$$

$$\mathcal{L}_{\mathsf{NO}} := \left\{ (K, v) \in \mathbb{G}^{2\times n} \times \mathbb{G}^2 \;\middle|\; \exists s \in \mathbb{Z}_p \text{ s.t. } \forall j \in [n],\ h_j = g_j^s \wedge (h' \neq (g')^s) \right\}$$

First note that the above languages are indeed disjoint. This is fairly easy to see - when K is such that $\exists s \in \mathbb{Z}_p$ such that $\forall j \in [n]$, $h_j = g_j^s$, then by construction, for any $\mathbf{m} \in \{0,1\}^n$, $g' = \prod_{j\in[n]} g_j^{m_j}$ and $h' = \prod_{j\in[n]} h_j^{m_j} = \prod_{j\in[n]} (g_j^{m_j})^s$ satisfy $h' \neq (g')^s$.

Before we proceed with our construction of a SNARG in the pre-processing model for the above promise languages. We show how such a SNARG would be sufficient to provide a proof of local opening, and the *somewhere soundness* property it satisfies. Note that we will extend the function definitions to additionally take in as input the CRS $\mathsf{crs}_{\mathcal{L}}$ and digest key dk. Here the digest key is output by the hash tree scheme (Section) on input security parameter λ. Recall that the security parameter for $\mathbb{G}$ is λ_0.

We describe the overview for $\ell = 1$, and then describe how it extends to the more general case. At a high level, to open the hash c to a value m_i at position i, the prover considers the key K' excluding the i-th column of the key K (i.e (g_i, h_i)). It then hashes the input bits except the bit at position i using this new key K' to obtain a new hash $c' = (g', h')$. Given (g_i, h_i), the claimed opening m_i, the verifier can efficiently check that $c = (g_i^{m_i} \cdot g',)$. Then the prover can then use the proof system of $\mathcal{L}$ to additional send a proof that the pair $(K', c') \in \mathcal{L}_{\mathsf{YES}}$. Unfortunately, this requires the online verification procedure to access the key K, which is of length N. Instead, the verifier computes a hash tree digest over K, and asks the prover to provide (g_i, h_i) along with the proof of opening. Note that the verifier V'_1 pre-processes on input K', which depends on the index i, but we want the pre-processing to be independent of the position the prover is trying to prove, and thus repeat the pre-processing for each position, and compute a hash tree digest over it. Since the pre-processing (and hash tree computation) is deterministic, the prover computes it as well, and then provides the appropriate proof with respect to the same digest that the prover and verifier hold. For instance, when the prover is proving a statement about an index i, it also provides the corresponding proof of local opening to the state computed by V'_1 on input K' defined earlier. Finally note that to extend to $\ell > 1$, we consider ℓ keys $K'_1, \cdots, K'_\ell$ where each $K'_i = ((g_1, \cdots, g_N), (h_{i,1}, \cdots, h_{i,N}))$, and repeat in parallel.

$\underline{\mathsf{Open}(K, \mathbf{m}, j^*, \mathsf{dk}, \mathsf{crs}_{\mathcal{L}})}$:

1. $(\mathsf{tree}, \mathsf{rt}, \mathsf{tree}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}) \coloneqq \mathsf{PreVer}(K, \mathsf{dk})$ //we abuse notation here since PreVer does not output the corresponding hash trees, but provided here for notational brevity.
2. $((g_{j^*}, \{h_{i,j^*}\}_{i\in[\ell]}), \Pi) \coloneqq \mathsf{HT.Read}(\mathsf{dk}, \mathsf{rt}, \mathsf{tree}, j^*)$.
3. $g'' = \prod_{j\neq j^*} g_j^{m_j}$, for all $i \in [\ell]$, $h_i'' = \prod_{j\neq j^*} h_{i,j}^{m_j}$.
4. For each $i \in [\ell]$,
   (a) Set K_i as $\begin{bmatrix} g_1 & \cdots & g_{j^*-1} & g_{j^*+1} & \cdots & g_N \\ h_{i,1} & \cdots & h_{i,j^*-1} & h_{i,j^*+1} & \cdots & h_N \end{bmatrix}$
   (b) $\Pi_i \leftarrow \mathsf{P}'(K_i, v = (g'', h_i''), m_{j\neq j^*}, \mathsf{crs}_{\mathcal{L}}, \mathsf{dk})$.
5. $(\mathsf{st}_{j^*}, \Pi_{\mathsf{st}}) \coloneqq \mathsf{HT.Read}(\mathsf{dk}, \mathsf{rt}_{\mathsf{st}}, \mathsf{tree}_{\mathsf{st}}, j^*)$.
6. Output $m_{j^*}, \pi_{j^*} = (g'', \{h_i''\}_i, (g_{j^*}, \{h_{i,j^*}\}_i), \Pi, \{\Pi_i\}_i, \mathsf{st}_{j^*}, \Pi_{\mathsf{st}})$.

$\underline{\mathsf{PreVer}(K, \mathsf{dk})}$:

1. $\mathsf{tree}, \mathsf{rt} \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, K)$.
2. For each $j \in [N]$
   (a) For each $i \in [\ell]$
      i. Set $K_{i,j}$ as $\begin{bmatrix} g_1 & \cdots & g_{j-1} & g_{j+1} & \cdots & g_N \\ h_{i,1} & \cdots & h_{i,j-1} & h_{i,j+1} & \cdots & h_N \end{bmatrix}$
      ii. $\mathsf{st}'_{i,j} \coloneqq \mathsf{V}'_1(\mathsf{dk}, K_{i,j})$
   (b) $\mathsf{st}'_j \coloneqq \{\mathsf{st}'_{i,j}\}$
3. $\mathsf{tree}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}} \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, \{\mathsf{st}'_j\}_{j\in[n]})$.
4. Output $\mathsf{st} \coloneqq (\mathsf{rt}, \mathsf{rt}_{\mathsf{st}})$.

$\underline{\mathsf{OnlineVer}(c, m_{j^*}, j^*, \pi_{j^*}, \mathsf{st}, \mathsf{dk}, \mathsf{crs}_{\mathcal{L}})}$:

1. Parse st as $(\mathsf{rt}, \mathsf{rt}_{\mathsf{st}})$
2. Parse π_{j^*} as $(g'', \{h_i''\}_i, (g_{j^*}, \{h_{i,j^*}\}_i), \Pi, \{\Pi_i\}_i, \mathsf{st}_{j^*}, \Pi_{\mathsf{st}})$.
3. Parse st_{j^*} as $\{\mathsf{st}'_{i,j^*}\}_{i\in[\ell]}$
4. Check if the following hold
   (a) $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, j^*, (g_{j^*}, \{h_{i,j^*}\}_{i\in[\ell]}), \Pi) = 1$
   (b) $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{\mathsf{st}}, j^*, \mathsf{st}_{j^*}, \Pi_{\mathsf{st}}) = 1$
   (c) $c = \begin{bmatrix} g'' \cdot g_{j^*}^{m_{j^*}} \\ h_1'' \cdot h_{1,j^*}^{m_{j^*}} \\ \vdots \\ h_\ell'' \cdot h_{\ell,j^*}^{m_{j^*}} \end{bmatrix}$
   (d) For each $i \in [\ell]$, $\mathsf{V}'_2(v = (g'', h_i''), \mathsf{crs}_{\mathcal{L}}, \mathsf{dk}, \mathsf{st}'_{i,j^*}, \Pi_i) = 1$
5. accept if none of the checks fail.

Efficiency. The total time to compute the pre-processing is $N^{z+1}\mathsf{poly}(\lambda, \ell+1)$, and the online verification takes time $\mathsf{poly}(\ell, \lambda_0, \lambda, \log N)$.

Lemma 8. *The above scheme satisfies computational extraction correctness.*

Proof. Let $S \subset [N]$ be a subset, and let $j^* \in S$ be the index output by the adversary $\mathcal{A}$. We show that computational extraction correctness holds for every such S and j^*. Without loss of generality, assume that j^* is the *first* element of S.

Let $\mathcal{A}$ output $\pi_{j^*} = (g'', \{h''_i\}_i, (\widetilde{g}_{j^*}, \{\widetilde{h}_{i,j^*}\}_i), \Pi, \{\Pi_i\}_i, \widetilde{\mathsf{st}}_{j^*}, \Pi_{\mathsf{st}})$. We define the event E to occur when both instances of HT.VerRead in OnlineVer accept (on proofs Π and Π_{st}), but the values are "incorrect", i.e. (i) $(\widetilde{g}_{j^*}, \{\widetilde{h}_{i,j^*}\}_i) \neq (g_{j^*}, \{h_{i,j^*}\}_i)$; and (ii) $\widetilde{\mathsf{st}}_{j^*} \neq \widetilde{\mathsf{st}}_{j^*}$. Note that these events are well defined given the key K, since $(g_{j^*}, \{h_{i,j^*}\}_i)$ and $\widetilde{\mathsf{st}}_{j^*}$ are defined deterministically given K, and can be computed in time $T_{\mathsf{HT}} := O(\lambda n^3) \cdot \mathsf{poly}(\kappa)$. Thus $\Pr[\mathsf{E}] < \mathsf{negl}(\lambda)$.

Now, let BREAK denote the event that $\mathcal{A}$ breaks the computational extraction correctness. Then,

$$\begin{aligned}\Pr[\mathsf{BREAK}] &= \Pr[\mathsf{BREAK} \mid \mathsf{E}]\Pr[\mathsf{E}] + \Pr[\mathsf{BREAK} \mid \overline{\mathsf{E}}]\Pr[\overline{\mathsf{E}}] \\ &< \Pr[\mathsf{E}] + \Pr[\mathsf{BREAK} \mid \overline{\mathsf{E}}] < \rho + \Pr[\mathsf{BREAK} \mid \overline{\mathsf{E}}]\end{aligned}$$

We will argue that $\Pr[\mathsf{BREAK} \mid \overline{\mathsf{E}}]$ will correspond to the probability of breaking the soundness of the non-interactive argument for the promise problem, i.e. we argue that conditioned on event $\overline{\mathsf{E}}$, $\mathcal{A}$ is attempting to generate a proof for $(K_i, v) \in \mathcal{L}_{\mathsf{NO}}$ for some i ($i = 1$ in our case).

As we stated before, we assumed without loss of generality that $j^* = j^*_1$. Thus the extracted value m_{j^*} is computed as 0 if $g'^{s_1} = h'_1$, and 1 otherwise. To break extraction correctness, $\mathcal{A}$ outputs $\widetilde{m}_{j^*} \neq m_{j^*}$. Note that for any key K sampled in the trapdoor mode on input S where $j^*_1 = j^*$, $h_{1,j^*} = g^s_{j^*} \cdot g$ and for each $j \neq j^*$, $h_{1,j} = g^s_j$. This implies that K_{1,j^*} as defined in PreVer is of the form $\begin{bmatrix} g_1 \cdots g_{N-1} \\ h_1 \cdots h_{N-1} \end{bmatrix}$ where for all j, $h_j = g^s_j$. Thus, for it to be the case that $(K_1, v = (g'', h''_1)) \in \mathcal{L}_{\mathsf{NO}}$, we need to prove that $h''_1 \neq g''^s$. Here we implicitly conditioning on event $\overline{\mathsf{E}}$ to ensure that K_{1,j^*} that is used in the verification, is indeed exactly as computed in PreVer.

We split the analysis into two cases:

Case (i): extracted value $m_{j^*} = 0$. Since $g'^{s_1} = h'_1$, $\widetilde{m}_{j^*} = 1$ implies

$$\begin{aligned}&(g'' \cdot g^{\widetilde{m}_{j^*}}_{j^*})^{s_1} = h''_1 \cdot h^{\widetilde{m}_{j^*}}_{1,j^*} \\ &\Rightarrow (g'' \cdot g_{j^*})^{s_1} = h''_1 \cdot h_{1,j^*} \\ &\Rightarrow (g'' \cdot g_{j^*})^{s_1} = h''_1 \cdot g^{s_1}_{1,j^*} \cdot g \\ &\Rightarrow (g'')^{s_1} = h''_1 \cdot g \\ &\Rightarrow (g'')^{s_1} \neq h''_1\end{aligned}$$

Case (ii): extracted value $m_{j^*} = 1$. Since $g'^{s_1} \neq h'_1$, $\widetilde{m}_{j^*} = 0$ implies

$$\begin{aligned}&(g'' \cdot g^{\widetilde{m}_{j^*}}_{j^*})^{s_1} \neq h''_1 \cdot h^{\widetilde{m}_{j^*}}_{1,j^*} \\ &\Rightarrow (g'')^{s_1} \neq h''_1\end{aligned}$$

Again, in the analysis of both of the two cases we implicitly use the fact that the "correct" values of g_{j^*} and h_{1,j^*} are used, which is guaranteed conditioned on the event $\overline{\mathsf{E}}$ occurring.

Thus, conditioned on the event $\overline{\mathsf{E}}$, if $\mathcal{A}$ outputs $\widetilde{m}_{j^*} \neq m_{j^*}$, where m_{j^*} is the extracted value, then $(K_1, v = (g'', h_1'')) \in \mathcal{L}_{\mathsf{NO}}$, which gives us,

$$\Pr[\mathsf{BREAK} \mid \overline{\mathsf{E}}] = \Pr\left[\mathcal{A} \text{ breaks soundness for } (\mathsf{P}', (\mathsf{V}_1', \mathsf{V}_2'))\right] < \mathsf{negl}(\lambda).$$

where the SNARG for $\mathcal{L}$ is ρ' secure against $T_\Pi = \mathsf{poly}(\lambda)$ adversaries. Putting it altogether,

$$\Pr[\mathsf{BREAK}] < \mathsf{negl}(\lambda)$$

□

5.2.3 Non-interactive Argument for $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$

In this section, we will construct non-interactive arguments for $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$. As described earlier, our construction will proceed in multiple steps. We first define notation that will be instructive for this section.

Notations. In the following discussion, we will require that keys and inputs are partitioned into blocks, with operations performed on the entire block. To improve readability, we will define some shorthand notations below. We will consider the setting where the number of blocks are fixed to be some parameter D. Further, for simplicity, we assume that the parameter n is some power of D. Consider a vector $\mathbf{x} = (x_1, \cdots, x_n) \in \mathbb{Z}_p^n$, we represent $\mathbf{x}$ by partitioning into D vectors $(\mathbf{x}_1, \cdots, \mathbf{x}_D)$ such that $\mathbf{x}_{j+1} = (x_{1+jn/D}, x_{2+jn/D}, \cdots, x_{(j+1)n/D}) \in \mathbb{Z}_p^{n/D}$. We extend this analogously to partitioning the key $K \in \mathbb{G}^{2\times n}$ as $\begin{bmatrix} \mathbf{g}_1 & \cdots & \mathbf{g}_D \\ \mathbf{h}_1 & \cdots & \mathbf{h}_D \end{bmatrix}$ where each $\mathbf{g}_{j+1} = (g_{1+jn/D}, g_{2+jn/D}, \cdots, g_{(j+1)n/D}) \in \mathbb{G}^{n/D}$ and $\mathbf{h}_{j+1}$ is defined similarly. Further, $\mathbf{g}_{j+1}^{\mathbf{x}_{i+1}} = \prod_{k=1}^{n/D} g_{k+jn/D}^{x_{k+in/D}} \in \mathbb{G}$, and for every scalar $\gamma \in \mathbb{Z}_p$, $\mathbf{g}_{j+1}^{\gamma} = (g_{1+jn/D}^{\gamma}, g_{2+jn/D}^{\gamma}, \cdots, g_{(j+1)n/D}^{\gamma}) \in \mathbb{G}^{n/D}$. Lastly, $\mathbf{g}_{j+1}^{\gamma} \cdot \mathbf{g}_{i+1} = \left(g_{1+jn/D}^{\gamma} \cdot g_{1+in/D}, g_{2+jn/D}^{\gamma} \cdot g_{2+in/D}, \cdots, g_{(j+1)n/D}^{\gamma} \cdot g_{1+(i+1)n/D}\right) \in \mathbb{G}^{n/D}$. Note that we will overload notation and use $\prod$ over both for elements in $\mathbb{G}$ and $\mathbb{G}^{n/D}$, where the resultant element will be in $\mathbb{G}$ and $\mathbb{G}^{n/D}$ respectively. The usage will be clear from context.

Interactive Protocol. We start with the interactive protocol where the prover P and V will start with a key $K \in \mathbb{G}^{2\times n}$, and at each step *fold* it into a new key $K' \in \mathbb{G}^{2\times n/2}$ of half the size based on the verifier challenge γ. After $\log_D(n)$ steps, only the final folded key $K \in \mathbb{G}^{2\times D}$ is used by the verifier to perform checks, and thus V *does not* need access to the intermediate folded keys during the protocol. From the description of our protocol, it will become evident that the folded key depends solely on the starting key K, and the verifier randomness $(\gamma_1, \cdots, \gamma_n)$. To simplify the protocol and analysis, we describe the interactive protocol in the pre-processing model where the verifier V has oracle access to a *pre-processed* string denoted by V^L. The string L depends on K, and our protocol will require the verifier to make a *single query* on the randomness used during the protocol, i.e. $(\gamma_1, \cdots, \gamma_n)$, which returns the appropriate folded key. We will subsequently discuss (i) how to compute the string L; and (ii) how to enable the prover to P to provide read access to the appropriate location of L.

The interactive protocol is presented in Figure 3.

Lemma 9. *The above protocol* $(\mathsf{P}, \mathsf{V}^L)$ *is complete.*

Proof. We argue completeness for the first level, and it then follows that the protocol is complete. Specifically, we show that at the first fold to obtain the *new* $K = \begin{bmatrix} \mathbf{g} \\ \mathbf{h} \end{bmatrix} \in \mathbb{G}^{2\times n/D}$, $\mathbf{x} \in \mathbb{G}^{n/D}$ and $v \in \mathbb{G}^2$ we have

Protocol: Interactive Protocol $(\mathsf{P}, \mathsf{V}^L)$ **for** $\mathcal{L}_{\mathsf{YES}}$

Common input: input $(K, v) \in \mathbb{G}^{2\times n} \times \mathbb{G}^2$, security parameter 1^λ.
P**'s auxiliary input:** input $\mathbf{x} = (x_1, \cdots, x_n)$

1. Set $i = 1$.
2. if $n > D$ repeat:
   (a) Prover does the following:
      i. Parse K as $\begin{bmatrix} \mathbf{g}_1 & \cdots & \mathbf{g}_D \\ \mathbf{h}_1 & \cdots & \mathbf{h}_D \end{bmatrix}$ and $\mathbf{x}$ as $(\mathbf{x}_1, \cdots, \mathbf{x}_D)$
      ii. For each $j \in [D]$, set $v_j \coloneqq \begin{bmatrix} \mathbf{g}_j^{\mathbf{x}_j} \\ \mathbf{h}_j^{\mathbf{x}_j} \end{bmatrix}$.
      iii. For each $j \in [D-1]$, set
      $$\widetilde{v}_j = \begin{bmatrix} \prod_{i=1}^{j} \mathbf{g}_i^{\mathbf{x}_{D-j+i}} \\ \mathbf{h}_i^{\mathbf{x}_{D-j+i}} \end{bmatrix} \text{ and } \widetilde{v}_{D+j} = \begin{bmatrix} \prod_{i=1}^{D-j} \mathbf{g}_{j+i}^{\mathbf{x}_i} \\ \mathbf{h}_{j+i}^{\mathbf{x}_i} \end{bmatrix}$$
      iv. Set $\widetilde{v}_D \coloneqq v$.
      v. Send $(v_1, \cdots, v_D)$ and $((\widetilde{v}_1, \cdots, \widetilde{v}_{2D-1}))$ to V.
   (b) Verifier V^L does the following:
      i. Checks if $v = \prod_{j=1}^{D} v_j$ and $\widetilde{v}_D = v$.
      ii. Sample $\gamma_i \leftarrow\!\!\$\ \mathscr{C}$.
      iii. Send γ_i to P.
   (c) P and V set $v = \prod_{j=1}^{2D-1} \widetilde{v}_j^{\gamma^{j-1}}$.
   (d) P sets $\mathbf{x} = \sum_{j=1}^{D} \mathbf{x}_{D-j+1} \cdot \gamma^{j-1}$, and $K = \begin{bmatrix} \prod_{j=1}^{D} \mathbf{g}_j^{\gamma^{j-1}} \\ \mathbf{h}_j^{\gamma^{j-1}} \end{bmatrix}$
   (e) $n \coloneqq n/D$, $i \coloneqq i + 1$
3. Let $h \coloneqq i$
4. Prover sends $\mathbf{x} = (x_1, \cdots, x_D)$ to V^L.
5. Verifier V^L does the following:
   (a) Query L at $(\gamma_1, \cdots, \gamma_h)$ and receive $K = \begin{bmatrix} \mathbf{g} \\ \mathbf{h} \end{bmatrix} \in \mathbb{G}^{2\times D}$ as response.
   (b) Check $v = \begin{bmatrix} \mathbf{g}^{\mathbf{x}} \\ \mathbf{h}^{\mathbf{x}} \end{bmatrix}$

   Accept if none of the checks have failed.

Figure 3: Interactive Protocol $(\mathsf{P}, \mathsf{V}^L)$ for $\mathcal{L}_Y$

that $v = \begin{bmatrix} \mathbf{g}^{\mathbf{x}} \\ \mathbf{h}^{\mathbf{x}} \end{bmatrix}$. Thus in the last step when the prover sends $\mathbf{x}$ in the clear, V will accept by performing

the computation.

By expanding each of the terms in the exponent for v we have

$$\prod_{j=1}^{2D-1} \widetilde{v}_j^{\gamma^{j-1}} = \begin{bmatrix} \prod_{j=1}^{D} \mathbf{g}_j^{\gamma^{j-1}(\mathbf{x}_D + \gamma \mathbf{x}_{D-1} \cdots + \gamma^{D-1}\mathbf{x}_1)} \\ \prod_{j=1}^{D} \mathbf{h}_j^{\gamma^{j-1}(\mathbf{x}_D + \gamma \mathbf{x}_{D-1} \cdots + \gamma^{D-1}\mathbf{x}_1)} \end{bmatrix}.$$

From the construction of the "folded" $K = \begin{bmatrix} \mathbf{g} \\ \mathbf{h} \end{bmatrix}$ in protocol, we see that the above equation is indeed

$$\prod_{j=1}^{2D-1} \widetilde{v}_j^{\gamma^{j-1}} = \begin{bmatrix} \mathbf{g}^{(\mathbf{x}_D + \gamma \mathbf{x}_{D-1} \cdots + \gamma^{D-1}\mathbf{x}_1)} \\ \mathbf{h}^{(\mathbf{x}_D + \gamma \mathbf{x}_{D-1} \cdots + \gamma^{D-1}\mathbf{x}_1)} \end{bmatrix}.$$

and the protocol sets $\mathbf{x}$ to be $\sum_{j=1}^{D} \mathbf{x}_{D-j+1} \cdot \gamma^{j-1}$. □

Instead of proving simply the soundness, we shall make a stronger claim about the protocol that will allow us to apply the Fiat-Shamir transform on the (parallel repeated version of the) above protocol to achieve a non-interactive argument in the CRS model. Specifically, we prove that the protocol is round-by-round sound (Definition 10).

Lemma 10. *The above protocol* $(\mathsf{P}, \mathsf{V}^L)$ *is* $(2D-2)$ *round-by-round sound.*

Proof. To argue that the protocol is round-by-round sound, we define the function State. For any $(K, v) \in \mathcal{L}_{\mathsf{YES}}$, we define $\mathsf{State}(x, \phi) = \mathsf{Accept}$, and for any $(K, v) \in \mathcal{L}_{\mathsf{YES}}$, we define $\mathsf{State}(x, \phi) = \mathsf{Reject}$. This can done by (i) checking if $\forall j, \exists s$ such that $h_j = g_j^s$; (ii) if so, check if $h' \neq (g')^s$ and output Reject ; else check if $\exists \mathbf{m} \in \{0,1\}^n$ s.t. $\mathsf{Hash}(K, \mathbf{m}) = v$. This is extended to $\mathsf{State}(x, \mathsf{trans})$ where one simply checks if the *current* claim at the end of the transcript is in $\mathcal{L}_{\mathsf{YES}}$ or $\mathcal{L}_{\mathsf{NO}}$, with one small adjustment that now it could be the case that $\mathbf{m}$ such that $\mathsf{Hash}(K, \mathbf{m}) = v$ now comes from $\mathbb{Z}_p$ instead of binary.

We now argue the probability of the State switching from Reject to Accept. This can happen in two ways: (i) the key K is such that there exists j such that $h_j \neq g_j^s$; or (ii) v is such that $h' = g'^s$. Case (i) never occurs by construction because if one were to start with a key K where $h_j = g_j^s$ for all j, the folding for *any* γ will ensure that for the new key K $h_j = g_j^s$ for all j. Thus the focus is on case (ii). Note that if the key satisfies the condition, K $h_j = g_j^s$ for all j, then $h' = g'^s$ implies the *existence* of an opening to v, which would result in the new claim (K, v) to be in $\mathcal{L}_{\mathsf{YES}}$. Let us compute the probability over the verifier randomness that case (ii) occurs.

Since g is the generator of $\mathbb{G}$, we represent each $\widetilde{v}_i$ as $\begin{bmatrix} g^{\alpha_i} \\ g^{\beta_i} \end{bmatrix}$. Since V checks that $\widetilde{v}_D = v$, we know that since in a Reject state, it must be the case that $g^{\beta_D} \neq g^{s\alpha_D}$. Thus the newly computed v,

$$\widetilde{v}_1 \cdot \widetilde{v}_2^{\gamma} \cdot \ldots \cdot \widetilde{v}_{2D-1}^{\gamma^{2D-2}} = \begin{bmatrix} g'' \\ h'' \end{bmatrix} = \begin{bmatrix} g^{\alpha_1 + \gamma\alpha_2 + \cdots + \gamma^{2D-2}\alpha_{2D-1}} \\ g^{\beta_1 + \gamma\beta_2 + \cdots + \gamma^{2D-2}\beta_{2D-1}} \end{bmatrix}.$$

Then,

$$\frac{(g'')^s}{h''} = g^{\sum_{i=1}^{2D-1}(s\alpha_i - \beta_i)\gamma^{i-1}} = g^{P(\gamma)}$$

where s is defined by the key, and $P(X)$ is a polynomial of degree at most $2D-2$. Note that since $(s\alpha_D - \beta_D) \neq 0$, $P(X)$ is not a zero-polynomial. Thus $(g'')^s = h''$ if and only if γ is root of $P(X)$. Since the number of such roots are bounded by $2D-2$, we have achieved our claimed round-by-round soundness. □

Lemma 11. *The BAD relation defined above can be verified in* TC^0 *given the trapdoor.*

Proof. From the above discussion, we need to check if the challenge is the root of the polynomial $P(X)$. To do so, BAD simply computes v based on the prover messages to obtain (g', h'). It then checks if $g'^s = h'$. To be able to compute g'^s in TC^0, we must pre-process the input as $g', g'^2, g'^{2^2}, g'^{2^3}, \cdots$ such that given s, g'^s can be computed in TC^0 by selecting and multiplying the pre-processed elements. □

Parameters and Efficiency. Looking ahead, we will require the pre-processing time to be "small", and thus set the challenge space to $|\mathscr{C}| = \log^2(n)$, setting $D = \log(n)$ we get the total number rounds $h = \frac{\log(n)}{\log\log(n)}$. This in turn means that the total possible verifier challenges is $|\mathscr{C}^h| = n^2$. Setting these parameters in Lemma 10 gives us a round-by-round soundness of $1/\log(n)$, which is quite large (not negligible in λ). To achieve negligible soundness, we need parallel repeat the protocol and then apply the Fiat-Shamir transform to round collapse the protocol. Further, note that the total running time of the verifier V^L is $(\log n/\log\log n)D \cdot \mathsf{poly}(\lambda_0)$.

Round Collapse via Fiat-Shamir. Given that we meet all the pre-conditions for applying the Fiat-Shamir transform, the resultant protocol is a non-interactive argument in the CRS model. Note that the size of the input to the CIH is $D \cdot O(\lambda_0) = \log n \cdot \lambda_0 = \mathsf{poly}(\log\log\lambda\log\lambda, \lambda_0)$. Here we use the fact that we will consider applications where $n = \lambda^{O(\log\log\lambda)}$. Thus by Corollary 1 we have non-interactive protocol $(\mathsf{P}_{\mathsf{FS}}, \mathsf{V}^L_{\mathsf{FS}})$ with CRS $\mathsf{crs}_{\mathsf{FS}}$ where the running time of $\mathsf{V}^L_{\mathsf{FS}}$ is $\mathsf{poly}(\lambda)$. Further, as described in Corollary 1, the resultant protocol is also $(\lambda^{O((\log\log\lambda)^2)}, 1/2^{\lambda^{1/\log\log\lambda}})$ secure, i.e. secure against super-polynomial adversaries.

Pre-Processing. Thus far we haven't addressed how the verifier in the interactive protocol has access to an oracle L. At a high level, given the key $K \in \mathbb{G}^{2\times n}$, L is going to represent *all possible* folded keys, each of the form $\mathbb{G}^{2\times D}$. From our discussion of the parameters above, the total number of challenge tuples is n^2, and thus the total length of L will be $O(\lambda_0 n^2)$. We describe how to generate these keys in time $O(\lambda_0 n^3)$. Importantly, note that once the string L has been generated, the same string can be used across all repetitions - or any instance that the key K is used.

Since there are ℓ such pairs - $(\mathbf{g}, \mathbf{h}_1), \cdots, (\mathbf{g}, \mathbf{h}_\ell)$, we generate a separate string L for each such pair. Our construction will use a hash tree scheme (HT.Gen,HT.Hash,HT.Read,HT.VerRead,) (Definition 9) with security parameter λ, where the write functions are omitted since we will not be requiring them. We split the somewhere extractable key $K \in \mathbb{G}^{(\ell+1)\times n}$ (as in the hash scheme), into keys $K_1, \cdots, K_\ell \in \mathbb{G}^{2\times n}$, which are in line with the keys we have used for our interactive proof. Where for each $i \in [\ell]$

$$K_i = \begin{bmatrix} g_1 & \cdots & g_n \\ h_{i,1} & \cdots & h_{i,n} \end{bmatrix}.$$

The pre-processing gets as input the digest key for the hash tree scheme $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(\kappa)$.

$\underline{\mathsf{PreVer}(\mathsf{dk}, K_1, \cdots, K_\ell):}$

1. For each $i \in [\ell]$ do the following:
   (a) Set L_i to be the empty string
   (b) For each tuple $(\gamma_1, \cdots, \gamma_h) \in \mathscr{C}^h$, set $L_i[(\gamma_1, \cdots, \gamma_h)] = \mathsf{FoldKey}_D(K_i, (\gamma_1, \cdots, \gamma_h), 1, n)$
   (c) $(\mathsf{tree}_{K,i}, \mathsf{rt}_{K,i}) = \mathsf{HT.Hash}(\mathsf{dk}, L_i)$.
2. Output $\left\{\mathsf{tree}_{K,i}\right\}_{i\in[\ell]}, \left\{\mathsf{rt}_{K,i}\right\}_{i\in[\ell]}$.

Note that for each i, the size of each $L_i[(\gamma_1, \cdots, \gamma_h)] =$ is $O(\lambda_0)$, thus the total size of L_i is $O(\lambda_0 n^2)$ and the size of dk is $\mathsf{poly}(\lambda)$. Now FoldKey is defined below,

$\underline{\mathsf{FoldKey}_D(K, (\gamma_1, \cdots, \gamma_h), r, n)}$:

1. Parse K as $\begin{bmatrix} \mathbf{g}_1 & \cdots & \mathbf{g}_D \\ \mathbf{h}_1 & \cdots & \mathbf{h}_D \end{bmatrix}$
2. Set $\alpha := \gamma_r$
3. Set $K' = \begin{bmatrix} \prod_{j=1}^{D} \mathbf{g}_j^{\alpha^{j-1}} \\ \prod_{j=1}^{D} \mathbf{h}_j^{\alpha^{j-1}} \end{bmatrix}$
4. If $n/D > 1$, return $\mathsf{FoldKey}(K', (\gamma_1, \cdots, \gamma_h), r+1, n/D)$; else return K'

From the description of FoldKey above, it is easy to observe that the folded key indeed corresponds exactly to the folded key in the execution of $(\mathsf{P}, \mathsf{V}^L)$.

Getting rid of the oracle in $(\mathsf{P}, \mathsf{V}^L)$. Given the description of the pre-processing step for the verifier, the approach to getting rid of the oracle is natural. The new verifier $\mathsf{V}' = (\mathsf{V}'_1, \mathsf{V}'_2)$ and prover P' is described below.

$\underline{\mathsf{P}'(K, v, \mathbf{x}, \mathsf{crs}_{FS}, \mathsf{dk})}$:

1. $\pi_{\mathsf{FS}} := (\alpha_1, \gamma_1, \cdots, \alpha_h, \gamma_h) \leftarrow \mathsf{P}_{\mathsf{FS}}(\mathsf{crs}_{FS}, K, v, \mathbf{x})$.
2. $\mathsf{tree}_K, \mathsf{rt}_K := \mathsf{PreVer}(K)$.
3. $(K', \Pi_{\mathsf{HT}}) := \mathsf{HT.Read}(\mathsf{dk}, \mathsf{rt}_K, \mathsf{tree}_K, (\gamma_1, \cdots, \gamma_h))$.
4. Output $\pi_{\mathsf{FS}}, K', \Pi_{\mathsf{HT}}$

$\underline{\mathsf{V}'_1(K, \mathsf{dk})}$:

1. $\mathsf{tree}_K, \mathsf{rt}_K := \mathsf{PreVer}(\mathsf{dk}, K)$.
2. Output $\mathsf{st}' = \mathsf{rt}_K$

$\underline{\mathsf{V}'_2(v, \mathsf{crs}_{FS}, \mathsf{dk}, \mathsf{st}', (\pi_{\mathsf{FS}}, K', \Pi_{\mathsf{HT}}))}$:

1. Parse π_{FS} as $(\alpha_1, \gamma_1, \cdots, \alpha_h, \gamma_h)$.
2. Check $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{st}', (\gamma_1, \cdots, \gamma_h), K', \Pi_{\mathsf{HT}}) = 1$.
3. Run $\mathsf{V}^L_{\mathsf{FS}}(\mathsf{crs}_{FS}, v, \pi_{\mathsf{FS}})$ where the query to L is answered by sending K'.
4. Accept if all checks pass.

Note that we have slightly abused HT.Read and HT.VerRead to read more than a single bit, but these functions extend naturally to multiple bits by simply concatenating the proof for each contained bit.

Corollary 2. *Assuming the sub-exponential hardness of* DDH*, and the security of the hash tree scheme,* $(\mathsf{P}', (\mathsf{V}'_1, \mathsf{V}'_2))$ *is a non-interactive argument for* $\mathcal{L} = (\mathcal{L}_{\mathsf{YES}}, \mathcal{L}_{\mathsf{NO}})$ *in the CRS model. Further* V'_1 *runs in time* $O(\lambda_0 n^3)$ *and outputs a state of size* $\mathsf{poly}(\lambda_0)$*. The total running time of* V'_2 *is* $\mathsf{poly}(\lambda)$*.*

5.2.4 Bootstrapping via Tree Hashing

Note that in the previous section, we constructed a somewhere extractable hash scheme for vectors of length N. The online verification was only $\mathsf{poly}(\ell, \lambda_0, \lambda, \log N)$ but the offline pre-processing was $N^4\mathsf{poly}(\lambda, \ell + 1)$. In our use cases, if we commit to a string of length T, we will want the verifier's pre-computation time to run in time significantly less than T. It might help the reader to think of T as the number of steps in a $\mathsf{DTIME}(T)$ computation, and it defeats the purpose of a SNARG if the offline phase depends on T. We consider the natural approach of building a hash tree using the *base hash* described in the previous section. As stated previously, we will require the extraction of the value from the leaves to be done in TC^0, and thus the depth of the tree must be a constant. Therefore we pick N (to be used in the base hash) to be such that $\log_N(T) = d$ for some constant d. Since this approach is standard, we only sketch the construction here.

1. Select d keys $K_1, \cdots, K_\ell$, one for each level of the tree.
2. For each i starting from the lowest level, partition the T bits into T/N blocks of size N, and compute the hash for each such block.
3. Parse the output as *bits*, and repeat again. Where $T' = T/N \times |\mathsf{Hash}|$, where $|\mathsf{Hash}|$ indicates the number of bits in the output of the hash.

First note each element in the group can be represented by $O(\lambda_0)$ bits. Since we are converting the hash output, which are group elements, into bits, there will be an increase in the output size at each level. Specifically, if at the leaf level, the key K_0 is extractable at ℓ positions, then the output is $O(\ell\lambda_0)$ bits. At the next level, we need *all* of these bits to be extractable in order to extract from the leaf level, therefore K_1 must be statistically binding at $O(\ell\lambda_0)$ positions, and the output in bits is $O(\ell\lambda_0^2)$ at the second level. Therefore, $K_i \in \mathbb{G}^{((l+1)\lambda_0^i)\times N}$. This means that by a simple calculation, the total number of bits at level i is $O(T/N^i \times (\ell+1)^i \times \lambda_0^{O(i^2)})$. Thus for the above construction to work, we need to argue that there is indeed compression at each step. This follows from the fact that N is super-polynomial in λ, and thus the function is indeed compressing.

Further, since the *same* key K_i is used in *all* the blocks of level i, the verifier only needs to do the offline pre-computation *once* per level, and does d such pre-computations in *total*. This gives us the following corollary.

Corollary 3. *Assuming sub-exponential hardness of* DDH*, the above scheme is a somewhere extractable hash scheme for large inputs with extraction in* TC^0 *(Definition 11). Further, to commit to strings of length T, let N be fixed such that* $\log_N(T) = O(1)$*. Then, the offline pre-processing runs in time* $N^4\mathsf{poly}(\lambda, \ell)$*, and the proof size as well as the online verification is* $\mathsf{poly}(\ell, \lambda_0, \lambda, \log N)$*.*

5.3 No-Signaling Somewhere Extractable Hash

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

Definition 12. *The extractor of an* SEHash *hash scheme* (Gen, TGen, Hash, Open, Verify, Ext) *is no-signaling if for any $S' \subseteq S \subseteq [N]$, where $|S| \leq L$, and any* PPT *adversary $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ there exists a negligible*

function $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$,

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

We will refer to SEHash schemes satisfying the above definition to be an L-no-signaling NS-SEHash hash scheme.

Theorem 7 ([GZ21])**.** *Given* L *instances of an* SEHash *hash scheme* (Gen, TGen, Hash, Open, Verify, Ext) *with locality parameter 1, one can construct an* L*-no-signaling* NS-SEHash*. Furthermore, if* SEHash *has succinct local openings, then so does* NS-SEHash*.*

At a very high level, the construction of an L-no-signaling NS-SEHash makes use of L repetitions of the underlying SEHash. Thus if the underlying SEHash satisfies computational extraction correctness, so does the resultant NS-SEHash.

6 SNARGs for Small-Circuit Batch-Index

In this section, we shall construct SNARGs for *Small-Circuit Batch-Index*. We start by defining SNARGs for Batch NP, and then defining the special case relevant for this work, SNARGs for Batch-Index. Before describing the construction, we cover the necessary background on PCPs and the properties relevant for this work.

6.1 Definition

Let SAT be the following language

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

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

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

Definition 13 (SNARGs for Batch-NP)**.** *A SNARG for Batch-*NP *a tuple of algorithms* (Gen, TGen, P, V) *that work as follows.*

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

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

- **Succinct Communication.** *The size of* π *is bounded by* $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Compact CRS.** *The size of* crs *is bounded by* $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Succinct Verification.** *The verification algorithm runs in time* $\mathsf{poly}(\lambda, T, n)+\mathsf{poly}(\lambda, \log T, |C|)$. *Moreover, it can be split into the following two parts*[11]*:*
  - **Pre-processing:** *There exists a deterministic algorithm* $\mathsf{PreVerify}(\mathsf{crs}, x_1, x_2, \ldots, x_T)$ *that takes as input the CRS, and* T *instances* $x_1, x_2, \ldots, x_T$*, and outputs a short sketch* c*, where* $|c| = \mathsf{poly}(\lambda, \log T, |x_1|)$.
  - **Online Verification:** *There exists an online verification algorithm* $\mathsf{OnlineVerify}(\mathsf{crs}, c, C, \pi)$ *that takes as input the sketch* c*, a circuit* C*, and a proof* π*, and outputs* 1 *(accepts) or* 0 *(rejects). Furthermore, the running time of the online verification algorithm is* $\mathsf{poly}(\lambda, |C|, |c|, |\pi|) = \mathsf{poly}(\lambda, \log T, |C|)$.
- **CRS Indistinguishability.** *For any non-uniform PPT adversary* $\mathcal{A}$*, and any polynomial* $T = T(\lambda)$*, there exists a negligible function* $\nu(\lambda)$ *such that*

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

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

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

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

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

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

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

[11] We note this is a stronger property than previously considered. However, its is natural, and our construction achieves this property.

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

SNARGs for Batch-Index. We now define SNARGs for Batch-Index. We start by defining the index language.

Definition 14 (Index Language)**.** *Let index language be the following language*

$$\mathcal{L}^{\mathsf{idx}} = \{(C, i) \mid \exists\ w \text{ s.t. } C(i, w) = 1\},$$

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

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

6.2 Background: PCP

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

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

the verification algorithm can be split into two parts: (i) a query algorithm Q which generates the PCP queries that depend on C but are independent of x; and (ii) an online verification algorithm D, which depends on x but its running time grows only *polylogarithmically* in $|C|$ and polynomially in $|x|$, further that D can be implemented in TC^0. It was shown in [CJJ21b] that the PCP in [BFLS91], and the probabilistic checkable interactive proofs in [RRR16], can be modified to obtain a PCP with fast online verification, i.e. *without* the low depth property. We sketch below how one can slightly modify Q to ensure that D can be implemented in TC^0.

The following text is taken largely verbatim from [CJJ21b], with the appropriate changes discussed.

Definition 15. *For any Boolean circuit* $C : \{0,1\}^{|x|} \times \{0,1\}^{|w|} \to \{0,1\}$*, a PCPs with a* fast and low depth online verification property *for* C-SAT *is a tuple of polynomial-time algorithms* $(\mathsf{P}, \mathsf{Q}, \mathsf{D})$*, with the following syntax.*

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

Furthermore, we require the following properties of the PCP.

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

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

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

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

– **Polynomial Proof Size.** *The size of the proof π is bounded by* $\mathsf{poly}(\lambda, |C|)$.

– **Small Query Complexity.** *The size of the set Q is bounded by* $\mathsf{poly}(\lambda, \log|C|)$.

– **Succinct Verification.** *The state* st *can be represented in* $\mathsf{poly}(\lambda, |x|, \log|C|)$ *bits, and the online verification algorithm runs in time* $\mathsf{poly}(\lambda, |x|, \log|C|)$. *The query algorithm* Q *runs in time* $\mathsf{poly}(\lambda, |C|)$.

– **Low Depth Verification.** *The online verification algorithm can be computed in* TC^0.

– ρ-**Proof of Knowledge in** TC^0**.** *For any PCP proof π^*, there exists a deterministic polynomial time extractor E, that can be implemented in* TC^0*, such that, if* $\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^\lambda, C, r) : \mathsf{D}(x, \mathsf{st}, \pi^*|_Q) = 1] > \rho(\lambda)$*, then* $\Pr[\omega \leftarrow E(\pi^*) : C(x, \omega) = 1] = 1$.

We first state a following claim. But before we do so, we specify the field that will be working with. Specifically, $\mathbb{F}$ will be a field of characteristic 2, i.e. $\mathbb{F} = \mathbb{F}_{2^n}$. This is necessary condition for us to claim that certain functions can be computed in TC^0. For such fields, consider $\{\alpha_i \in \mathbb{F}\}_i$, iterated addition ($\sum_i \alpha_i$) and iterated multiplication ($\prod_i \alpha_i$) can be computed in TC^0 [HV06, HAM02]. Henceforth, for notational simplicity, we will refer to these fields simply by $\mathbb{F}$ and drop the subscript.

Claim 1. *For any set of $d+2$ pairs in $\mathbb{F}^2$, $\{(x_1, y_1), \cdots, (x_{d+2}, y_{d+2})\}$, checking if (x_{d+2}, y_{d+2}) lie on the degree d univariate polynomial specified by $\{(x_1, y_1), \cdots, (x_{d+2}, y_{d+1})\}$ can be implemented in* TC^0 *with pre-processed input that depend only on $x_1, \cdots, x_{d+2}$.*

Proof Sketch. At a high level, this is done by computing a degree d univariate polynomial $L(x)$ that interpolates the the points $\{(x_1, y_1), \cdots, (x_{d+2}, y_{d+1})\}$, and then checking if $L(x_{d+2}) = y_{d+2}$. Note that $L(x_{d+2}) = \sum_{j=1}^{d+1} y_{d_j} \ell_j(x_{d+2})$, where $\ell_1, \cdots, \ell_{d+1}$ is the Lagrange basis for polynomials of degree $\leq d$. The reader can observe that given $\{\ell_j(x_{d+2})\}_{j\in[d+1]}$, computing $L(x_{d+2})$ and comparing to y_{d+2} can be done in TC^0, since iterated addition and field multiplication are in TC^0. All we need to show is that $\{\ell_j(x_{d+2})\}_{j\in[d+1]}$ can be computed given $x_1, \cdots, x_{d+2}$. This follows directly from the definition of Lagrange basis, where $\ell_j(x) := \prod_{i\neq j} \frac{x - x_i}{x_j - x_i}$, thus proving our claim. □

A *low degree test* for a polynomial $f : \mathbb{F}^m \to \mathbb{F}$, given only oracle access to f checks whether f is a polynomial of total degree[12] d, rejecting if f is "far" from any degree d polynomial. We do not formalize the notion of "far" since it is not relevant to our discussion, and we refer the reader to [RS96, AS03, Sud95] for more details. We make the following observation about the low degree tests in [RS96, AS03, Sud95], which will be useful for our setting.

Observation 1. *There exists low degree tests for polynomial $f : \mathbb{F}^m \to \mathbb{F}$, that can be implemented in* TC^0 *given pre-processing information that depends only on the queries made to f, and* not *on the evaluated responses on f.* [13]

[12] Total degree refers to the sum of all individual degrees in an m-variate polynomial.

[13] We note that this claim also extends to the case where one needs to check the individual degree of the polynomial (as opposed to total degree in the observation). This follows from the fact that the individual degree test consists of (i) a total degree test (as in the claim); followed by (ii) a univariate test by interpolating points as in Claim 1. See [GR15] for more details.

Proof Sketch. At a high level, the tests in [RS96, AS03, Sud95] involves randomly sampling $\overline{r}, \overline{s} \leftarrow\!\!\$\ \mathbb{F}^m$, and $i \leftarrow\!\!\$\ \mathbb{F}$. Next, the test queries f at $\overline{r}, \overline{r}+\overline{s}, \overline{r}+2\overline{s}, \cdots, \overline{r}+d\overline{s}$ and $\overline{r}+i\overline{s}$ and accepts only if $(i, \overline{r}+i\overline{s})$ lies on the univariate polynomial p, where $p(j) = f(\overline{r}+j\overline{s})$ for all $j \in \{0, \cdots, d\}$. Thus the verification reduces to Claim 1, which means that the verification can be done in TC^0 with pre-processing computed given only $\overline{r}$, $\overline{s}$ and i. Note that there exists tests with fixed co-efficients $\alpha_j = (-1)^{j+1}\binom{d+1}{j}$ such that one can simply check whether $\sum_j \alpha_j f(\overline{r}+j\overline{s}) = 0$. Looking ahead, we mention this test as this will be relevant in us showing that extraction of the witness from the PCP can be done in TC^0. □

We are now ready to extend the lemma in [CJJ21b] to state the following.

Lemma 12. *There exists a PCP with fast and low depth online verification property for the* C-SAT *language with ρ-soundness, and ρ-proof of knowledge property, where $\rho = 1 - 1/\mathrm{poly}\log|C|$.*

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

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

Let $I : \mathbb{H}^{m'} \to \{0,1\}, K : \mathbb{H}^{m'} \to \mathbb{H}^{\max(m_x, m_y)}$ be the following polynomials.

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

where we identify the index set $[|x|+|y|]$ with $\mathbb{H}^m$. Let $\widetilde{I}, \widetilde{K}$ be the extension of I, K to $\mathbb{F}$, respectively. Then $\widetilde{I}$ and $\widetilde{K}$ has degree at most $\mathrm{poly}(m')$. Let $\widetilde{x} = \mathsf{LDE}(x), \widetilde{y} = \mathsf{LDE}(y)$ be the low-degree extension of x, y over $\mathbb{F}$, respectively.

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

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

Let $C' : \mathbb{H}^{3m'+3} \to \{0,1\}$ be a circuit such that $C'(i_1, i_2, i_3, b_1, b_2, b_3) = 1$ if and only if $b_1, b_2, b_3 \in \{0,1\}$ and $(x_{i_1} = b_1) \vee (x_{i_2} = b_2) \vee (x_{i_3} = b_3)$ is a clause in the 3-CNF ϕ, and let $\widetilde{C} : \mathbb{F}^{3m'+3} \to \mathbb{F}$ be the extension of C' to $\mathbb{F}$. Then we have that $\phi(x,\cdot)$ is satisfiable, if and only if there exists a $\widetilde{y}$ such that the following polynomial $F(z)$ of $3m'+3$ variables is a zero polynomial:

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

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

- **Low-Degree Testing:** The prover sends $\widetilde{y} = \mathsf{LDE}(y)$. The verifier performs a low-degree test on $\widetilde{y}$.
- **Sumcheck:** Then the verifier sends a random $z^* \in \mathbb{F}^{3m'+3}$. The prover and the verifier then execute a sumcheck protocol to prove $F(z^*) = 0$. Let

$$\widetilde{\phi}_{z^*}(i_1, i_2, i_3, b_1, b_2, b_3) = \mathsf{LDE}(\{\widetilde{\mathsf{Eq}}_{i_1, i_2, i_3, b_1, b_2, b_3}(z^*)\}_{i_1, i_2, i_3 \in \mathbb{H}^{m'}, b_1, b_2, b_3 \in \{0,1\}})$$

be the low-degree extension of the linear coefficients $\widetilde{\mathsf{Eq}}_{i_1,i_2,i_3,b_1,b_2,b_3}(z^*)$. Then the prover and the verifier run the sumcheck protocol for the sum

$$\sum_{i_1,i_2,i_3\in\mathbb{H}^{m'},b_1,b_2,b_3\in\{0,1\}} \widetilde{C}(i_1,i_2,i_3,b_1,b_2,b_3)\cdot\widetilde{P}(i_1,i_2,i_3,b_1,b_2,b_3)\cdot\widetilde{\phi}_{z^*}(i_1,i_2,i_3,b_1,b_2,b_3)=0.$$

At the end of the sumcheck protocol, the verifier obtains a random point $(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)\in\mathbb{F}^{3m'+3}$ (corresponding to its messages in the protocol) and a value $v\in\mathbb{F}$. The verifier then checks whether

$$\widetilde{C}(i_1^*,i_2^*,i_3^*,i_1^*,b_2^*,b_3^*)\cdot\widetilde{P}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)\cdot\widetilde{\phi}_{z^*}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)=v. \tag{2}$$

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

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

  In addition, for the sumcheck, we do the following "preprocessing" to ensure low depth and small time in online verification. For $\widetilde{C}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$, we evaluate it directly, and store the resultant value in the state st. Furthermore, to help the online verification algorithm (described below) compute $\widetilde{P}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$ in time only polylogarithmic in $|C|$, we compute $\widetilde{K}(i_j^*),\widetilde{I}(i_j^*)$ and $\{\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_{m_x}}(\widetilde{K}(i_j^*))\}_{i_1,i_2,\ldots,i_{m_x}\in\mathbb{H}}$ for $j\in\{1,2,3\}$ in time $\mathsf{poly}(C)$, and also store the resultant values in the state st. Finally, we compute and store $\widetilde{\phi}_{z^*}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$ in st. Finally, for the low degree test, compute the necessary pre-computed state as defined in 1.

  Since there are $O(|\mathbb{H}|n)$ number of field elements in the state st, the size of st is bounded by $\mathsf{poly}(\lambda,|x|,\log|C|)$.

- $\mathsf{PCP.D}(x,\mathsf{st},\pi')$: We will show that given the state st, the verification runs in $\mathsf{poly}(|x|,\log|C|)$ time, and computable in TC^0.

  For the low-degree testing, the verifier performs the same verification procedure as the underlying low-degree testing. This takes time $\mathsf{poly}(\log|C|)$. Note that this uses the pre-computed values in st, and thus can be computed in TC^0.

  For the sumcheck, the verifier performs the same checks as in the underlying sumcheck protocol. At the end, the verifier uses the "preprocessed" values of $\widetilde{C}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$ present in the state st. To compute $\widetilde{P}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*.b_3^*)$, the verifier will obtain each term in Equation 1. For $\widetilde{K}(i_j^*),\widetilde{I}(i_j^*)$, the verifier can obtain them from the state st. For $\widetilde{y}(\widetilde{K}(i_j^*))$, the verifier obtains it from the PCP response π'. For $\widetilde{x}(\widetilde{K}(i_j^*))$, the verifier computes it by the definition of low-degree extension (see Section 3.1) using $\{\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_{m_x}}(\widetilde{K}(i_j^*))\}_{i_1,i_2,\ldots,i_{m_x}}$ in the state st. Since this is an iterated addition using the values $\{\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_{m_x}}(\widetilde{K}(i_j^*))\}_{i_1,i_2,\ldots,i_{m_x}}$ contained in st, $\widetilde{x}(\widetilde{K}(i_j^*))$ can be computed in TC^0. Now the verifier obtains all terms in Equation 1, and hence can compute $\widetilde{P}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$. Further, by Equation 1, $\widetilde{P}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$ is a product of sum of constant many terms, and thus can be computed in TC^0. Finally, the verifier obtains $\widetilde{\phi}_{z^*}(i_1^*,i_2^*,i_3^*,b_1^*,b_2^*,b_3^*)$ from the state st, and verifies Equation 2, which can be computed in TC^0.

  For the running time, the computation of the low-degree extension $\widetilde{x}(\widetilde{K}(i_j^*))$ takes time $O(|x|\cdot\mathsf{poly}(\log|C|))$. Hence, the online verification takes time $\mathsf{poly}(|x|,\log|C|)$ in total, and as discussed for each component, computable in TC^0.

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

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

Low Depth Extraction. The low degree test only guarantees that the $\widetilde{y}$ is "close" to a degree d polynomial. To extract, one needs to apply a *self-correction procedure*[Sud95, GLR+91] to decode $\widetilde{y}$. Specifically, if the low degree test rejects with probability ρ, then one can construct a polynomial g of degree d, that is 2ρ close to f. For $\overline{x} \in \mathbb{F}^m$,

$$g(\overline{x}) = \mathsf{maj}_{\overline{s}\in\mathbb{F}^m}\left\{\sum_{i=1}^{d+1} \alpha_i \cdot f(\overline{x} + i\overline{s})\right\}$$

where $\alpha_i = (-1)^{i+1}\binom{d+1}{i}$.

We show that the above term can be computed in TC^0. First note that we will chose $\mathbb{F}^m$ such that $|\mathbb{F}|^m$ will be $\mathsf{poly}(\lambda)$. Thus, for each $\overline{s} \in \mathbb{F}^m$, we compute $\sum_{i=1}^{d+1} \alpha_i \cdot f(\overline{x} + i\overline{s})$, which is an iterated sum and field multiplication, and can thus be done in TC^0. Let the outputs be $v_1, \cdots, v_{|\mathbb{F}|^m}$. In order to compute the majority, we first compute the frequency of each output, and then find the output that has the highest frequency. Finding highest frequencies can be done in TC^0 - let the elements of $\mathbb{F}$ be $\alpha_1, \cdots, \alpha_{|\mathbb{F}|}$, the frequency for each element α_i can be computed as $\sum_{j\in|\mathbb{F}|^m}(\alpha_i = v_j)$. The comparison of two elements in $\mathbb{F}$ can be done in TC^0, since this can be reduced to n parallel copies of *boolean* equality comparison for fields of characteristic 2. Let $\mathsf{c}(\alpha_i)$ be the count corresponding to α_i. To check if α_i has the highest count, one computes $\bigwedge_j(\mathsf{c}(\alpha_i) \geq \mathsf{c}(\alpha_j))$, where each comparison is done in TC^0, and the AND over polynomially inputs can be implemented by a threshold gate, thus ensuring the entire computation is in TC^0. This is done for each α_i Finally, output α_{i^*} if the corresponding AND gate results in 1 (breaking ties arbitrarily). □

Next, we define the *bad* relation for any PCP with fast and low depth online verification property with an eye towards our SNARGs for small-circuit batch-index construction we describe next. As described in the overview in section 2, we commit several PCP proofs "columnwise" using a somewhere extractable hashand apply a CIH to these commitments to obtain the query PCP Q.

In the soundness proof, we first switch the commitment key to the trapdoor mode. The *bad* relation is defined with respect to the trapdoor td of the commitment. Specifically, we can use td to extract a PCP proof π from the commitment. Now given the extracted proof π, we define a query Q to be bad, when $\pi|_Q$ is accepting but we cannot extract a witness from π. However, the verification algorithm not only needs Q, but also the state st. To resolve this issue, in the following definition, we have the CIH output the randomness r. We then use this randomness to generate Q and st via PCP.Q. The SEHash scheme used below will have security parameter λ_0 where $\lambda = \lambda_0^{O(\log\log\lambda_0)}$.

Definition 16 (Bad relation for PCP)**.** *Let* $\mathsf{SEHash} = (\mathsf{SEHash.Gen}, \mathsf{SEHash.TGen}, \mathsf{SEHash.Hash}, \mathsf{SEHash.Open}, \mathsf{SEHash.Verify}, \mathsf{SEHash.Ext})$ *be a somewhere extractable hash scheme, and* $\mathsf{PCP} = (\mathsf{P}, \mathsf{Q}, \mathsf{D})$ *be any PCP with fast online verification, we define the bad relation* $\mathcal{R} = \{\mathcal{R}_{\lambda_0}\}_{\lambda_0}$ *for* PCP *as follows.*

For any instance length $n = n(\lambda_0)$*, witness length* $m = m(\lambda_0)$*, proof length* $\ell = \ell(\lambda_0)$*, and a parameter* $T = T(\lambda)$*, we define the bad relation for* PCP *as* $\mathcal{R}_\lambda = \{R_{\lambda_0,x,\mathsf{td}}\}$*, where* td *is obtained from* $(K^*, \mathsf{td}) \leftarrow \mathsf{SEHash.TGen}(1^{\lambda_0}, 1^T, i^*)$ *for a index* $i^* \in [T]$*, and*

$$R_{\lambda_0,x,\mathsf{td}} = \{((C,c),r) \mid C(x, E(\pi)) \neq 1 \wedge \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1\},$$

where $(Q, \mathsf{st}) = \mathsf{Q}(1^{\lambda_0}, C, r)$, $c = \{c_q\}_{q\in[\ell]}$, $\pi = \{\mathsf{SEHash.Ext}(c_q, \mathsf{td})\}_{q\in[\ell]}$, $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *is a Boolean circuit, and* x *is a string of length* n*, and* E *is the proof of knowledge extractor.*

Note that we do not require T to be $\mathsf{poly}(\lambda_0)$, it can be $\mathsf{poly}(\lambda)$.

Theorem 8 (CIH for PCP)**.** *There exists a PCP with fast and low depth online verification* $(\mathsf{P}, \mathsf{Q}, \mathsf{D})$ *and a hash family* $\mathcal{H}$ *such that,* $\mathcal{H}$ *is correlation intractable for its bad relation family* $\mathcal{R} = \{\mathcal{R}_{\lambda_0}\}_{\lambda_0}$ *(in Definition 16). Furthermore,* $\mathcal{H}$ *can be evaluated in time* $\mathsf{poly}(\lambda, |C|)$.

Proof. Intuitively, we will take the PCP in Lemma 12, and repeat its verification several times in parallel (with independent randomness), and apply Lemma 1 to the resulting PCP.

Let $\mathsf{PCP}' = (\mathsf{PCP}'.\mathsf{P}, \mathsf{PCP}'.\mathsf{Q}, \mathsf{PCP}'.\mathsf{D})$ be the $(1-\epsilon)$-sound PCP with fast online verification from Lemma 12, where $\epsilon = 1/\mathsf{poly}\log|C|$. We build a new $\mathsf{PCP} = (\mathsf{P}, \mathsf{Q}, \mathsf{D})$ as follows.

- P is the same as $\mathsf{PCP}'.\mathsf{P}$.
- $\mathsf{Q}(1^{\lambda_0}, C, r)$: Parse $r = (r_1, r_2, \ldots, r_t)$, where $t = \lambda_0^3/\epsilon$.
  - For each $i \in [t]$, let $(Q_i, \mathsf{st}_i) = \mathsf{PCP}'.\mathsf{Q}(1^{\lambda_0}, C, r_i)$.
  - Output $Q = (Q_1, Q_2, \ldots, Q_t)$, $\mathsf{st} = (\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_t)$.
- $\mathsf{D}(x, \mathsf{st}, \pi')$ Parse $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_t)$, and $\mathsf{st} = (\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_t)$.
  - For each $i \in [t]$, verify if $\mathsf{PCP}'.\mathsf{D}(x, \mathsf{st}_i, \pi'_i) = 1$.
  - If all verification passes, then output 1 (accept). Otherwise output 0 (reject).
- Proof of knowledge Extractor E: We use the proof of knowledge extractor of PCP' as the extractor for PCP.

The resultant PCP satisfies $\rho = (1-\epsilon)^t = 2^{-\Omega(\lambda_0)}$-soundness and ρ-proof of knowledge property. By construction, for each security parameter λ, instance x, and trapdoor td, $R_{\lambda,x,\mathsf{td}}$ is a product relation, since the bad relation for PCP is the product of the bad relations for PCP'. The bad relation for PCP' is efficiently verifiable in time $\mathsf{poly}(\lambda_0, \log T, |C|)$, where $|C| = \mathsf{poly}(\lambda_0)$. Further, given our discussion of the PCP construction, the bad relation is also verifiable in TC^0. Note that the online verification will take time of the PCP will take time $\mathsf{poly}(\lambda_0, |C|)$

To demonstrate sparsity, for any instance x and extracted PCP proof π, if $C(x, E(\pi)) \neq 1$, then $\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^{\lambda_0}, C, r) : \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1] \leq \rho$, otherwise this contradicts the ρ-soundness of PCP. Since our construction is a parallel repetition,

$$\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^{\lambda_0}, C, r) : \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1] = \Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{PCP}'.\mathsf{Q}(1^{\lambda_0}, C, r) : \mathsf{PCP}'.\mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1]^t.$$

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

$$\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{PCP}'.\mathsf{Q}(1^{\lambda_0}, C, r) : \mathsf{PCP}'.\mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1] \leq 1 - \epsilon.$$

Hence, the relation $R_{\lambda_0,\mathsf{td}}$ has sparsity $(1-\epsilon)$. Therefore, by Lemma 1, there exists a correlation intractable hash family $\mathcal{H}$ for $\mathcal{R}$. □

6.3 Construction

In this section, we will use the constructed SEHash, and CIH for PCP to construct SNARGs for batch-index, where the circuit is "small". Specifically, we consider the setting where $|C| = \mathsf{poly}(\lambda_0)$ where $\lambda = \lambda_0^{\log\log\lambda_0}$ is the security parameter of the scheme.

Our construction is identical to [CJJ21b], except that we replace (i) the SEHash with our constructed SEHash assuming (sub-exponential hardness of) DDH; and (ii) the CIH with the CIH for small inputs. The construction is presented in Section C for completeness, with the changes highlighted below. We argue that these changes are sufficient to achieve the same result as [CJJ21b] for circuits with size $\mathsf{poly}(\lambda_0)$, but under the sub-exponential DDH assumption.

Differences from [CJJ21b]. We elaborate on the differences below.

1. To apply the input restricted CIH, we need to show that at each step, the input to the CIH is bounded by $\mathsf{poly}(\lambda_0)$. Since the CIH is applied on the SEHash which has security parameter λ_0, we only need to argue that the circuit NewRel, and thus VerifyC, is bounded in size by $\mathsf{poly}(\lambda_0)$. This simply follows from the description of VerifyC, which performs an online verification of (i) the local opening of SEHash; and (ii) of the PCP responses. Since we have chosen the parameters appropriate (for PCP and SEHash) that VerifyC is bounded in size by $\mathsf{poly}(\lambda_0)$ as required at each step.

2. The proof follows in an identical manner, except that when we rely on primitives with security parameter λ_0, we require the primitive to be $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ where $\lambda = \lambda_0^{\log\log\lambda_0}$. This follows from the fact that in the security reductions to these primitives, since the parameter of the scheme is λ, the reduction will run a $\mathsf{poly}(\lambda)$ adversary that succeeds with probability $1/\mathsf{poly}(\lambda)$. This can be done by appropriately setting the security parameter of the underlying primitives.

3. Lastly, unlike in [CJJ21b], due to the efficiency constraints, we use the SEHash constructed in Section 5.1, which only satisfies computational extraction correctness. This means that except with negligible probability, if the SEHash opening proof is accepting, the uniquely extracted value from the SEHash is the same as the value in the opening. This does not affect the security of the transformation - as described in Section 6.2 the PCP extraction is done via the SEHash extractor, which outputs a unique value given the commitment. This extracted value is used in the analysis, and one can then simply argue that except with negligible probability, this corresponds exactly to the value the adversary opens the SEHash commitment to. Here we continue to crucially rely on the fact that given the SEHash key, the extracted value is statistically determined for *any* commitment.

This gives us the following theorem,

Theorem 9. *Assuming the sub-exponential hardness of* DDH*, there exists a SNARG for small-circuit batch-index with circuit size bounded by* $\mathsf{poly}(\lambda_0)$*, where* $\lambda = \lambda_0^{\log\log\lambda_0}$ *is the security parameter of the SNARG.*

7 SNARGs for P and Batch-NP

In this section, we describe how SNARGs for small-circuit batch-index can be used to obtain SNARGs for P and Batch-NP. The transformations are fairly straightforward, and we only give a high-level sketch.

7.1 SNARGs for P

Similar to Section 6.3, our SNARG for P (more generally, SNARGs for RAM computation) is exactly the construction in [CJJ21b] with minor changes: (i) the security parameter used for the underlying no-signalling

somewhere extractable hash scheme and the tree hash scheme are λ_0, where $\lambda = \lambda_0^{\log\log\lambda_0}$; and (ii) use the SNARGs for batch index for circuits of size $\mathsf{poly}(\lambda_0)$. We present the definition for the more general setting of RAM delegation, and a complete description of the construction in [CJJ21b] in Section D.

As in Section 6.3, the primitives with security parameter λ_0 will require $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ security. All that's left to do is show that the index circuit that is batched is of size $\mathsf{poly}(\lambda_0)$. From Figure 7, observe that the index circuit C_{index} performs the online verification of the local openings of the NS-SEHash hash, and then checks the circuit φ. Since the security parameter for both the NS-SEHash and hash tree scheme are λ_0, it follows that $|C_{\mathsf{index}}| = \mathsf{poly}(\lambda_0)$. Thus, following [CJJ21b], we have the theorem.

Theorem 10. *Assuming the sub-exponential hardness of* DDH*, for every polynomial $T = T(\lambda)$, there exists a publicly verifiably non-interactive RAM delegation scheme with CRS size, proof size and verifier time all* $\mathsf{poly}(\lambda, \log T)$ *while the prover running time is* $\mathsf{poly}(\lambda, T)$.

7.2 SNARGs for Batch-NP

Here we will sketch the transformation that extends SNARGs for small circuits batch-index, to SNARGs for batch-index, i.e. removing the small circuit requirements. As a corollary to this extension, from [CJJ21b] we also achieve SNARGs for Batch NP. At a high level, we want to reduce the problem of SNARGs for batch-index for a (potentially large) circuit C to *many* invocations of the SNARGs for small circuits batch-index.

The idea is simple: each of these circuits will only verify a *single gate*[14] in the original circuit C, and thus the total number of these new circuits will be $D = |C|$, which we denote by $\widetilde{C}_1, \cdots, \widetilde{C}_D$. As long as we can show that $\widetilde{C}_i = \mathsf{poly}(\lambda_0)$, the prover can send a batch index proof for each of these circuits, for a multiplicative overhead in communication of $|C|$, which satisfies the succinctness requirement.

Let the number of gates be D, and the number of wire values be $W = O(|C|)$. We have made the simplifying assumption that both the fan-in and fan-out of the circuit is 2. Thus, we can define a function $G : [D] \mapsto [W]^4$, that maps the gate to its corresponding 4 wire indices. Thus, given the 4 wire values, of gate j, specified by $G(j)$, the correctness of the gate can be checked in $O(1)$ time. We denote the four indices by $G(j)_1, G(j)_2, G(j)_3, G(j)_4$. Note that we need to ensure consistency across the circuits, else a cheating prover will try to convince the verifier of D independent circuits. We solve this by the following method to batch T instances:

1. Prover P computes the wire values for the index circuit C, i.e. for instance i, the wire values are $\omega_{i,1}, \cdots, \omega_{i,W}$
2. The prover uses a somewhere extractable hash SEHash to hash these in a column-wise fashion. Specifically, $c = \{c_j\}_{j\in[W]}$ where for each $j \in [W]$, $c_j \coloneqq \mathsf{SEHash.Hash}(K, \omega_{1,j}, \cdots, \omega_{T,j})$.
3. The prover next defines D index circuits $\widetilde{C}_1, \cdots, \widetilde{C}_D$ as follow:
   (a) $\widetilde{C}_j$ has hardcoded $c_{G(j)_1}, c_{G(j)_2}, c_{G(j)_3}, c_{G(j)_4}$, and the pre-processed state $\mathsf{st}_{\mathsf{SEHash}}$ of SEHash.
   (b) On input i and witness $w_{i,j} \coloneqq (\omega_{G(j)_1}, \omega_{G(j)_2}, \omega_{G(j)_3}, \omega_{G(j)_4}, \pi_{G(j)_1}, \pi_{G(j)_2}, \pi_{G(j)_3}, \pi_{G(j)_4})$, runs the online verification for SEHash to check if (i) the π_k correspond to the local opening of c_k to ω_k; and (ii) $\omega_{G(j)_1}, \omega_{G(j)_2}, \omega_{G(j)_3}, \omega_{G(j)_4}$ satisfy the gate constraint for gate j.
4. For each $j \in [D]$, prover computes $\Pi \leftarrow \mathsf{BARG.P}(\widetilde{C}_j, T, \{w_{i,j}\}_{i\in[T]})$.
5. Sends proof $\{\Pi_j\}_{j\in[D]}$ to the verifier.

[14] For simplicity we assume that the gate has fan-in 2, and fan-out 2.

If we set the security parameter of SEHash to be λ_0, then by succinct local opening property of SEHash above description we have that for each j, $|\widetilde{C}_j| = \mathsf{poly}(\lambda_0)$. The somewhere soundness of the scheme follows from (i) the somewhere extractability property of SEHash since once can generate the key for each circuit $\widetilde{C}_j$ on the same input i^*; and (ii) somewhere soundness of the SNARGs for batch index of small circuits. We thus have the following theorem.

Theorem 11 (SNARGs for Batch-Index)**.** *Assuming the existence of SNARGs for small-circuit batch-index and somewhere extractable hash (SEH), there exists SNARGs for Batch-index where the size of the CRS and proof is* $\mathsf{poly}(\lambda, \log T, |C|)$*, and the verification time is* $\mathsf{poly}(\lambda, \log T, |C|)$*.*

Instantiating the primitives with the constructions in this paper, we have the following corollary.

Corollary 4. *Assuming sub-exponential* DDH*, there exists SNARGs for Batch-index where the size of the CRS and proof is* $\mathsf{poly}(\lambda, \log T, |C|)$*, and the verification time is* $\mathsf{poly}(\lambda, \log T, |C|)$*.*

Finally, from [CJJ21b], we have the following corollary for SNARGs for Batch NP.

Corollary 5 (SNARGs for Batch NP)**.** *Assuming sub-exponential* DDH*, there exists SNARGs for Batch-*NP *where the size of the CRS and proof is* $\mathsf{poly}(\lambda, \log T, |C|)$*, and the verification time is* $\mathsf{poly}(\lambda, T, n) + \mathsf{poly}(\lambda, \log T, |C|)$*.*

Acknowledgments

Abhishek Jain and Zhengzhong Jin were supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. Abhishek Jain was also supported in part by AFOSR Award FA9550-19-1-0200 and the Office of Naval Research Grant N00014-19-1-2294. Zhengzhong Jin was additionally supported in part by NSF CAREER 1845349.

Arka Rai Choudhuri and Sanjam Garg were supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

This research was conducted in part when Arka Rai Choudhuri was at UC Berkeley, and Jiaheng Zhang was an intern at NTT Research.

References

[AS03] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Comb.*, 23(3):365–426, 2003. 42, 43

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. 5, 15, 29

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017. 6

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013. 6

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. 3

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013. 6

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *23rd ACM STOC*, pages 21–31. ACM Press, May 1991. 41, 43

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016. 3

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 474–482. ACM Press, June 2017. 63

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 738–767. Springer, Heidelberg, August 2020. 3, 4, 6

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. 3, 4, 6, 20

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 389–415. Springer, Heidelberg, January 2016. 3, 6

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018. 3, 4, 6

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004. 3, 18

[CHK+19] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1103–1114. ACM Press, June 2019. 4

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg. 3, 4, 6

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for P from LWE. *FOCS*, 2021. 3, 4, 5, 6, 10, 11, 12, 41, 43, 47, 48, 49, 59, 61, 64

[CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. Non-interactive zero-knowledge in pairing-free groups from weaker assumptions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 442–471. Springer, Heidelberg, May 2020. 3, 4, 6

[CLMQ21] Yilei Chen, Alex Lombardi, Fermi Ma, and Willy Quach. Does fiat-shamir require a cryptographic hash function? In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 334–363, Virtual Event, August 2021. Springer, Heidelberg. 6

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569. Springer, Heidelberg, August 2017. 3

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019. 11, 13, 14, 27, 28, 29

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Tauman Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. *FOCS*, 2022. 6

[FPS22] Cody Freitag, Rafael Pass, and Naomi Sirkin. Parallelizable delegation from lwe. In *TCC*, 2022. 4

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. 3

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. 6

[GHY20] Ofer Grossman, Justin Holmgren, and Eylon Yogev. Transparent error correcting in a computationally bounded world. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 530–549. Springer, Heidelberg, November 2020. 4

[GLR+91] Peter Gemmell, Richard J. Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *23rd ACM STOC*, pages 32–42. ACM Press, May 1991. 45

[GR15] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In Tim Roughgarden, editor, *ITCS 2015*, pages 133–142. ACM, January 2015. 42

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010. 6

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. 6

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. 3

[GZ21] Alonso González and Alexandros Zacharakis. Fully-succinct publicly verifiable delegation from constant-size assumptions. Cryptology ePrint Archive, Report 2021/353, 2021. https://eprint.iacr.org/2021/353. 38, 39

[HAM02] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. Special Issue on Complexity 2001. 42

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 520–549. Springer, Heidelberg, May / June 2022. 3, 4, 6

[HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th FOCS*, pages 850–858. IEEE Computer Society Press, October 2018. 3, 6

[HLR21] Justin Holmgren, A. Lombardi, and R. Rothblum. Fiat-shamir via list-recoverable codes (or: Parallel repetition of gmw is not zero-knowledge). *STOC*, 2021. 3, 4, 5, 6, 9, 10, 11, 18, 25

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006*, pages 672–683, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 42

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. 3, 4, 5, 11, 26

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 3–32. Springer, Heidelberg, October 2021. 3, 4, 5, 6, 7, 9, 20, 55, 56, 57, 58

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. SNARGs for Bounded Depth Computations and PPAD Hardness from Sub-Exponential LWE. In *STOC*. ACM, 2021. 4, 6

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. 6

[KLV22] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs and PPAD hardness from the decisional diffie-hellman assumption. 2022. https://eprint.iacr.org/2022/1409. 6

[KLVW22] Yael Tauman Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. *IACR Cryptol. ePrint Arch.*, page 1320, 2022. 6

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 91–118. Springer, Heidelberg, October / November 2016. 63

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. 5, 19, 61, 63

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 565–574. ACM Press, June 2013. 6

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th ACM STOC*, pages 485–494. ACM Press, May / June 2014. 6

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017. 3, 4, 6

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Heidelberg, November 2021. 3, 6, 10

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012. 6

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 632–651. Springer, Heidelberg, August 2020. 4

[LV22] Alex Lombardi and Vinod Vaikuntanathan. Correlation-intractable hash functions via shift-hiding. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 102:1–102:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 6

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988. 20

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 6

[Mou21] Tamer Mour. Correlation intractability vs. one-wayness. Cryptology ePrint Archive, Report 2021/057, 2021. https://eprint.iacr.org/2021/057. 6

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003. 6

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Heidelberg, November / December 2015. 11, 27

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. *FOCS*, 2022. 6

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019. 3, 4, 5, 6

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016. 41, 43

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. 42, 43

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs anf the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995. 42, 43, 45

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, 2022. 3, 6

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016. 6

A Interactive Trapdoor Hashing Protocol

The following definition of interactive trapdoor hashing protocol is taken verbatim from [JJ21].

Let $C = \{C_{n,u}\}_{n,u}$ be a family of circuits, where each circuit $f \in C_{n,u}$ is a circuit of input length n and output length u. An L-level interactive trapdoor hashing protocol for the circuit family C is a tuple of algorithms ITDH = (KGen, Hash&Enc, Dec) that are described below.

We use $\lambda_1, \ldots, \lambda_L$ to denote the security parameters for different levels. Throughout this work, these parameters are set so that they are polynomially related. That is, there exists a λ such that $\lambda_1, \ldots, \lambda_L$ are polynomials in λ.

- $\mathsf{KGen}(1^{\lambda_\ell}, \ell, f, \mathsf{h}_{\ell-1}, \mathsf{td}_{\ell-1})$: The key generation algorithm takes as input a security parameter λ_ℓ (that varies with the level number), a level number ℓ, a circuit $f \in C_{n,u}$, a level $(\ell-1)$ hash value $\mathsf{h}_{\ell-1}$ and trapdoor $\mathsf{td}_{\ell-1}$ (for $\ell = 1$, $\mathsf{h}_{\ell-1} = \mathsf{td}_{\ell-1} = \bot$). It outputs an ℓ^{th} level key k_ℓ and a trapdoor td_ℓ.
- $\mathsf{Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1})$: The hash-and-encode algorithm takes as input a level ℓ hash key k_ℓ, an input x, and a level $(\ell-1)$ encoding $\mathsf{e}_{\ell-1}$. It outputs an ℓ^{th} level hash value h_ℓ and an encoding $\mathsf{e}_\ell \in \{0,1\}^u$. When $\ell = 1$, we let $\mathsf{e}_{\ell-1} = \bot$.
- $\mathsf{Dec}(\mathsf{td}_L, \mathsf{h}_L)$: The decoding algorithm takes as input a level L trapdoor td_L and hash value h_L, and outputs a value $\mathsf{d} \in \{0,1\}^u$.

We require ITDH to satisfy the following properties:

- **Compactness:** For each level $\ell \in [L]$, the bit length of h_ℓ is at most λ_ℓ.
- (Δ, ϵ)-**Approximate Correctness:** For any $n, u \in \mathbb{N}$, any circuit $f \in C_{n,u}$ and any sequence of security parameters $(\lambda_1, \ldots, \lambda_L)$, we have

$$\Pr_{r_1, r_2, \ldots, r_L}\left[\forall x \in \{0,1\}^n, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f(x)) < \Delta(u)\right] > 1 - \epsilon(u, \lambda_1, \ldots, \lambda_L),$$

  where e, d are obtained by the following procedure: Let $\mathsf{h}_0 = \mathsf{td}_0 = \mathsf{e}_0 = \bot$. For $\ell = 1, 2, \ldots, L$,

  - Compute $(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{KGen}(1^{\lambda_\ell}, \ell, f, \mathsf{h}_{\ell-1}, \mathsf{td}_{\ell-1}; r_\ell)$ using random coins r_ℓ.
  - Hash and encode the input x: $(\mathsf{h}_\ell, \mathsf{e}_\ell) \leftarrow \mathsf{Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1})$.

  Finally, let $\mathsf{e} = \mathsf{e}_L$ be the encoding at the final level, and $\mathsf{d} = \mathsf{Dec}(\mathsf{td}_L, \mathsf{h}_L)$.
- **Leveled Function Privacy:** There exist a simulator Sim and a negligible function $\nu(\cdot)$ such that for any level $\ell \in [L]$, any polynomials $n(\cdot)$ and $u(\cdot)$ in the security parameter, any circuit $f \in C_{n,u}$, any trapdoor $\mathsf{td}' \in \{0,1\}^{|\mathsf{td}_{\ell-1}|}$, any hash value $\mathsf{h}' \in \{0,1\}^{|\mathsf{h}_{\ell-1}|}$, and any n.u. PPT distinguisher $\mathcal{D}$,

$$\left|\Pr\left[(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{KGen}(1^{\lambda_\ell}, \ell, f, \mathsf{h}', \mathsf{td}') : \mathcal{D}(1^{\lambda_\ell}, \mathsf{k}_\ell) = 1\right] - \Pr\left[\widetilde{\mathsf{k}}_\ell \leftarrow \mathsf{Sim}(1^{\lambda_\ell}, 1^n, 1^u, \ell) : \mathcal{D}(1^{\lambda_\ell}, \widetilde{\mathsf{k}}_\ell) = 1\right]\right| \leq \nu(\lambda_\ell).$$

  We say that the ITDH satisfies sub-exponential leveled function privacy, if there exists a constant $0 < c < 1$ such that for any non-uniform distinguisher that runs in time $\lambda^{O(\log\log\lambda)^2}$, $\nu(\lambda_\ell)$ is bounded by $2^{-\lambda_\ell^c}$ for any sufficiently large λ_ℓ.

Assuming sub-exponential DDH holds, [JJ21] provides the construction of ITDH for P/poly with sub-exponential leveled function privacy. Here we only care about ITDH for $O(\log^* \lambda)$ depth threshold circuits. We state the theorem from [JJ21] as follows.

Theorem 12 (Implicit in [JJ21])**.** *Let $\{L_\lambda\}_{\lambda\in\mathbb{N}}, \{w_\lambda\}_{\lambda\in\mathbb{N}}$ be two sequences of integers with $L_\lambda = O(\log^* \lambda)$ and $w \geq \lambda$, and $\{C_{\lambda,w}\}_{\lambda,\in\mathbb{N}}$ be a class of threshold circuits of depth L and output length w.*

There exists a $2L$-level interactive trapdoor hashing protocol for $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ that achieves (Δ, ϵ)-approximate correctness and sub-exponential function privacy, where $\Delta = \lambda$ and $\epsilon = 2^{-\Omega(\lambda)}$, and we require that $\lambda_1 < \lambda_2 < \ldots < \lambda_{2L} < \lambda/2L$.

Moreover, if we assume the sub-exponential DDH assumption (as defined in Definition 1), then the construction satisfies sub-exponential leveled function privacy.

B Proof of Theorem 5

Construction. The construction of our CIH is almost the same as the CIH in [JJ21]. We take it verbatim here. The only difference is in the proof of correlation intractability.

CIH for α-Approximate Product Relations

- $\mathsf{Gen}(1^\lambda)$:
  - For each $\ell \in [L]$, set $\lambda_\ell = \lambda^{\frac{1}{2}(\frac{c}{2})^{L-\ell}}$.
  - Compute simulated receiver's messages for ITDH:
$$\forall \ell \in [L], \mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell)$$
  - Sample a mask $\mathsf{u} \leftarrow \{0,1\}^w$ uniformly at random.
  - Output $\mathsf{k} = \left(\{\mathsf{k}_\ell\}_{\ell\in[L]}, \mathsf{u}\right)$.
- $\mathsf{Hash}(\mathsf{k}, x)$:
  - Parse $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell\in[L]}, \mathsf{u})$.
  - Let $\mathsf{e}_0 = \perp$. Compute hash values and encodings for ITDH:
$$\forall \ell \in [L], (\mathsf{h}_\ell, \mathsf{e}_\ell) \leftarrow \mathsf{ITDH.Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1}).$$
  - Output $\mathsf{e} \oplus \mathsf{u}$, where $\mathsf{e} = \mathsf{e}_L$.

Figure 4: Description of CIH.

Proof Sketch. We follow the proof in [JJ21] with some modifications. We prove Theorem 5 by contradiction. Let $\{f_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of threshold circuits for $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ with depth $L = O(\log^* \lambda)$ and let $\mathcal{A}$ be a n.u. PPT adversary breaking the correlation intractability with probability $\epsilon(\lambda)$. We present a series of hybrids in the proof of [JJ21] as follows.

For any relation $R \in \mathcal{R}_\lambda$, we use R_x to denote the set $\{\mathsf{y} \in \mathcal{Y}_\lambda^{t_\lambda} \mid (x, \mathsf{y}) \in R\}$.

- Hyb_0: In this hybrid, if the adversary's attack successes, then output 1, otherwise output 0.

- Sample the CIH key $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$, and run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
- If $\mathsf{Hash}(\mathsf{k}, x) \in R_x$, then output 1, otherwise output 0.

– $\mathsf{Hyb}_1^{\ell^*}$: Here the index $\ell^* \in \{1, \ldots, L+1\}$. This hybrid is almost the same as $\mathsf{Hyb}_1^{\ell^*-1}$ (or Hyb_0 when $\ell^* = 1$), except that we additionally guess the hash value h_{ℓ^*-1} by sampling h'_{ℓ^*-1} from uniform distribution. Hence $\mathsf{Hyb}_1^1 = \mathsf{Hyb}_0$.

- Let $\mathsf{h}_0 = \mathsf{td}_0 = \perp$. For $\ell = 1, 2, \ldots, \ell^* - 1$, if $\ell > 1$, let $\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}$. Let

$$(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1})$$

- If $\ell^* > 1$, sample $\mathsf{h}'_{\ell^*-1} \leftarrow \{0,1\}^{\lambda_{\ell^*-1}}$ uniformly at random. Let $\mathsf{k}_{\ell^*} \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_{\ell^*}}, 1^n, 1^w, \ell^*)$.
- For $\ell = \ell^* + 1, \ldots, L$, let $\mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell)$.
- Sample $\mathsf{u} \leftarrow \{0,1\}^w$ uniformly at random. Let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{u})$.
- Run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
- If $\mathsf{Hash}(\mathsf{k}, x) \in R_x$ and $\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

– $\mathsf{Hyb}_{1.5}^{\ell^*}$: Here the index $\ell^* \in \{1, \ldots, L+1\}$. This hybrid is the same as $\mathsf{Hyb}_1^{\ell^*}$, except that we replace the ℓ^{th} level key with a "real key" generated by ITDH.KGen.

- Let $\mathsf{h}_0 = \mathsf{td}_0 = \perp$. For $\ell = 1, 2, \ldots, \ell^*$, if $\ell > 1$, let $\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}$. Let

$$(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1})$$

- For $\ell = \ell^* + 1, \ldots, L$, let $\mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell)$.
- Sample $\mathsf{u} \leftarrow \{0,1\}^w$ uniformly at random. Let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{u})$.
- Run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
- If $\mathsf{Hash}(\mathsf{k}, x) \in R_x$ and $\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

– Hyb_2: This hybrid is the same as Hyb_1^{L+1}.

- Let $\mathsf{h}_0 = \mathsf{td}_0 = \perp$. For $\ell = 1, 2, \ldots, L$, if $\ell > 1$, let $\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}$. Let

$$(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1})$$

- Sample $\mathsf{h}'_L \leftarrow \{0,1\}^{\lambda_L}$ uniformly at random.
- Sample $\mathsf{u} \leftarrow \{0,1\}^w$ uniformly at random. Let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{u})$.
- Run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
- If $\mathsf{Hash}(\mathsf{k}, x) \in R_x$ and $\forall i \in [L], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

With the same proof in [JJ21], we can get $\Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1] \geq \Pr[\mathsf{Hyb}_1^{\ell^*} = 1] - 2^{-\lambda_{\ell^*}^c}$ and $\Pr[\mathsf{Hyb}_1^{\ell^*+1} = 1] \geq \Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1]/2^{\lambda_{\ell^*}}$,where c is a constant. Next we modify the proof in [JJ21] to prove $\Pr[\mathsf{Hyb}_2 = 1] < 2^{-\Omega(\lambda)}$.

Bound the Probability $\Pr[\mathbf{Hyb}_2 = 1]$**.** In Hyb_2, we check if $\forall i \in [L], \mathsf{h}'_\ell = \mathsf{h}_\ell$. Note that if such check passes, then $\mathsf{Hash}(\mathsf{k}, x) = \mathsf{e} \oplus \mathsf{u}$ in R_x, where e is the encoding in the final level in an honest execution. Hence,

$$\Pr[\mathsf{Hyb}_2 = 1] \leq \Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x : \mathsf{e} \oplus \mathsf{u} \in R_x],$$

where $r_1, r_2, \ldots, r_L$ are the random coins for the ITDH, and the encoding e is obtained from the following procedure.

Let $\mathsf{H}_0 = \mathsf{td}_0 = \mathsf{e}_0 = \perp$. For $\ell = 1, 2, \ldots, L$,

- Compute $(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}_{\ell-1}, \mathsf{td}_{\ell-1}; r_\ell)$ with random coins r_ℓ.
- Hash the input x using the hash key $(\mathsf{h}_\ell, \mathsf{e}_\ell) \leftarrow \mathsf{ITDH.Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1})$

Finally, let $\mathsf{e} = \mathsf{e}_L$ be the encoding at the final level, and also let $\mathsf{d} = \mathsf{ITDH.Dec}(\mathsf{td}_L, \mathsf{H}_L)$. Since the ITDH used in [JJ21] for $O(\log^* \lambda)$-depth threshold circuits satisfies $(\Delta = \lambda, \epsilon = 2^{-\Omega(\lambda)})$-approximate correctness by Theorem 12, we have

$$\Pr_{r_1, r_2, \ldots, r_L} [\exists x, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) > \lambda] < 2^{-\Omega(\lambda)}.$$

Hence, except with probability $2^{-\Omega(\lambda)}$, we have that $\forall x, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) \leq \lambda$. Denote the Hamming error vector between $\mathsf{e} \oplus \mathsf{d}$ and $f_\lambda(x)$ as ε. Then we have $f_\lambda(x) = \mathsf{e} \oplus \mathsf{d} \oplus \varepsilon$, and the weight of ε is at most λ. For any vector $\mathbf{y} \in \mathcal{Y}^t$, given a set S containing αt indexes $i_1, \ldots, i_{\alpha t} \in [t]$, we use $\mathbf{y}|_S$ to represent the vector of $(y_{i_1}, \ldots, y_{i_{\alpha t}})^T$ in $\mathcal{Y}^{\alpha t}$. Now, we have,

$$\begin{aligned}
&\Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x : (x, \mathsf{e} \oplus \mathsf{u}) \in R] \leq \Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x, S, |S| \geq \alpha t : \mathsf{e} \oplus \mathsf{u}|_S = f_\lambda(x)|_S] \\
&\leq \Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x, S, |S| \geq \alpha t : \mathsf{e} \oplus \mathsf{u}|_S = f_\lambda(x)|_S \wedge \forall x : \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) \leq \Delta] \\
&\quad + \Pr_{r_1, \ldots, r_L} [\exists x, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) > \Delta] \\
&\leq \Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x, S, |S| \geq \alpha t : \mathsf{e} \oplus \mathsf{u}|_S = f_\lambda(x)|_S \wedge \exists \varepsilon : \mathsf{e} \oplus \mathsf{d} = f_\lambda(x) + \varepsilon] + 2^{-\Omega(\lambda)} \\
&\leq \binom{t}{\alpha t} \Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x, \varepsilon : \mathsf{e} \oplus \mathsf{u}|_S = \mathsf{e} \oplus \mathsf{d} \oplus \varepsilon|_S] + 2^{-\Omega(\lambda)} \\
&\leq \binom{t}{\alpha t} \cdot \Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x, \varepsilon : \mathsf{u}|_S = \mathsf{d} \oplus \varepsilon|_S] + 2^{-\Omega(\lambda)}.
\end{aligned}$$

The first inequality holds as R is α-approximable by f_λ. The second inequality holds since we enumerate all possibilities of whether $\mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) \leq \Delta$ holds or not. The third inequality follows from the $(\Delta = \lambda, \epsilon = 2^{-\Omega(\lambda)})$-approximate correctness of ITDH. The fourth inequality holds by the union bound on the set S. The last one holds as we cancel e on both sides of the equation. Note that, for any fixed random coins $r_1, r_2, \ldots, r_L$, the decoding value d only depends on $\mathsf{h}_1, \mathsf{h}_2, \ldots, \mathsf{h}_L$. The number of possible choice of $\mathsf{d}|_S$ is $2^{(\lambda_1 + \lambda_2 + \ldots + \lambda_L)} \leq 2^{2\lambda_L} \leq 2^{2\lambda^{1/2}}$. The number of possible values of $\varepsilon|_S$ is at most $\binom{|S|}{\Delta} \leq (e|S|/\Delta)^\Delta = 2^{\widetilde{O}(\lambda)}$. Hence, we have

$$\Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x, \varepsilon : \mathsf{u}|_S = \mathsf{d} \oplus \varepsilon|_S] < 2^{2\lambda^{1/2}} \cdot 2^{\widetilde{O}(\lambda)} \cdot \left(\frac{1}{|\mathcal{Y}|}\right)^{|S|} = 2^{\widetilde{O}(\lambda)} \cdot \frac{1}{|\mathcal{Y}|^{\alpha t}}$$

In summary, we have

$$\Pr_{\mathsf{u} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x : (x, \mathsf{e} \oplus \mathsf{u}) \in R] \leq \binom{t}{\alpha t} \cdot 2^{\widetilde{O}(\lambda)} / |\mathcal{Y}|^{\alpha t} \leq \left(\frac{e}{\alpha |\mathcal{Y}|}\right)^{\alpha t} \cdot 2^{\widetilde{O}(\lambda)}.$$

If $|\mathcal{Y}| \geq 2e/\alpha$, then the above inequality can be bounded by $2^{-\alpha t + \widetilde{O}(\lambda)}$. Since we set $t \geq \lambda^2/\alpha$, this probability is bounded by $2^{-\Omega(\lambda^2)}$.

Completing the Proof. Let $\epsilon(\lambda) = \Pr[\mathsf{Hyb}_0 = 1]$. We first claim that, for each $\ell^* \in [L+1]$, we have

$$\Pr\left[\mathsf{Hyb}_1^{\ell^*} = 1\right] \geq (\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*-1}},$$

where $\lambda_0 = 0$.

We now prove this claim by induction on ℓ^*. For $\ell^* = 1$, since Hyb_1^1 and Hyb_0 are identical, we have $\Pr\left[\mathsf{Hyb}_1^1 = 1\right] = \Pr\left[\mathsf{Hyb}_0 = 1\right] \geq \epsilon(\lambda)$. Hence, then the claim holds for $\ell^* = 1$. Now, we assume the claim holds for ℓ^*, we prove that the claim holds for $\ell^* + 1$ as follows.

By $\Pr\left[\mathsf{Hyb}_{1.5}^{\ell^*} = 1\right] \geq \Pr\left[\mathsf{Hyb}_1^{\ell^*} = 1\right] - 2^{\lambda_{\ell^*}^c}$, we have

$$\Pr\left[\mathsf{Hyb}_{1.5}^{\ell^*} = 1\right] \geq \Pr\left[\mathsf{Hyb}_1^{\ell^*} = 1\right] - 2^{\lambda_{\ell^*}^c} \geq (\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*-1}} - 2^{-\lambda_{\ell^*}^c}.$$

By the choice of the parameters, we have $\lambda_{\ell^*} = (\lambda_{\ell^*-1})^{\frac{2}{c}}$ Hence, the right hand side is bounded by

$$\begin{aligned}\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1}/2^{2\lambda_{\ell^*-1}} - 2^{-\lambda_{\ell^*-1}^2} &= (\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1} - 2^{-\lambda_{\ell^*-1}^2+\lambda_{\ell^*-1}})/2^{2\lambda_{\ell^*-1}} \\ &\geq (\epsilon(\lambda) - (\ell^*+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*-1}}\end{aligned}$$

By $\Pr\left[\mathsf{Hyb}_1^{\ell^*+1} = 1\right] \geq \Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1]/2^{\lambda_{\ell^*}}$, then we have

$$\begin{aligned}\Pr\left[\mathsf{Hyb}_1^{\ell^*+1} = 1\right] \geq \Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1]/2^{\lambda_{\ell^*}} &\geq (\epsilon(\lambda) - (\ell^*+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*-1}+\lambda_{\ell^*}} \\ &> (\epsilon(\lambda) - (\ell^*+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*}}.\end{aligned}$$

Hence, we finish prove the claim.

By this claim, and the fact that Hyb_2 is identical to Hyb_1^{L+1} we know that

$$\Pr[\mathsf{Hyb}_2 = 1] = \Pr[\mathsf{Hyb}_1^{L+1} = 1] \geq (\epsilon(\lambda) - (L+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda^{1/2}}.$$

As $\Pr[\mathsf{Hyb}_2 = 1] < 2^{-\Omega(\lambda)}$, we have

$$\epsilon(\lambda) < 2^{-\Omega(\lambda)+2\lambda^{1/2}} + (L+1)2^{-\lambda_1^2+2\lambda_1}.$$

Since $L = O(\log^* \lambda)$ and $\lambda_1 = \lambda^{O(\frac{1}{\log\log\log\lambda})}$, we finish the proof. □

C Protocol: SNARGs for Small-Circuit Batch-Index

The following section is taken largely verbatim from [CJJ21b], with the differences highlighted below in the description.

The description of the SNARGs for Batch-Index is recursive - to construct $\mathsf{BARG}_L = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{P}, \mathsf{V})$ at the L-level of recursion, the following components are used:

- A somewhere extractable hash (section 5.1) $\mathsf{SEHash} = (\mathsf{SEHash.Gen}, \mathsf{SEHash.TGen}, \mathsf{SEHash.Hash}, \mathsf{SEHash.Open}, \mathsf{SEHash.Verify}, \mathsf{SEHash.Ext}, \mathsf{SEHash.PreVer}, \mathsf{SEHash.OnlineVer})$ with security parameter λ_0.
- A PCP with fast and low depth verification $\mathsf{PCP} = (\mathsf{PCP.P}, \mathsf{PCP.Q}, \mathsf{PCP.D})$ with proof length $\ell = \ell(\lambda_0, |C|)$ with security parameter λ_0.
- A CIH (definition 2) $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ for the bad relation of PCP from Theorem 8 with security parameter λ_0.
- A non-interactive batch arguments at the $(L-1)$-level $\mathsf{BARG}_{L-1} = (\mathsf{BARG}_{L-1}.\mathsf{Gen}, \mathsf{BARG}_{L-1}.\mathsf{TGen}, \mathsf{BARG}_{L-1}.\mathsf{P}, \mathsf{BARG}_{L-1}.\mathsf{V})$ with security parameter λ.

Circuit $\mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st},\ \mathsf{st}_{\mathsf{SEHash}}]}(i,(\vec{\rho},\pi',\vec{\rho}',\pi''))$

Output

$\mathsf{VerifyC}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st},\ \mathsf{st}_{\mathsf{SEHash}}]}(2i-1,\vec{\rho},\pi') \wedge \mathsf{VerifyC}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st},\ \mathsf{st}_{\mathsf{SEHash}}]}(2i,\vec{\rho}',\pi'')$.

Figure 5: The grouped new circuit, where the ungrouped new circuit VerifyC is depicted in Figure 6.

Circuit $\mathsf{VerifyC}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st},\ \mathsf{st}_{\mathsf{SEHash}}]}(i,\vec{\rho},\pi')$

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

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

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

$$\forall q \in Q,\ \mathsf{SEHash.OnlineVer}(c_q, \pi'_q, i, \rho_q, \mathsf{st}_{\mathsf{SEHash}}) = 1.$$

- Verify π' is accepted by the PCP online verification, i.e. verify $\mathsf{PCP.D}(\mathsf{st}, i, \pi') = 1$.
- If all verification passes, then output 1 (accept), otherwise output 0 (reject).

Figure 6: The ungrouped new circuit.

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

- $\mathsf{Gen}(1^\lambda, 1^{T=2^L}, 1^{|C|})$: The CRS generation algorithm generates a CRS, which contains (i) a somewhere extractable hash key; (ii) a CRS for the smaller non-interactive batch arguments BARG_{L-1}; and (ii) a key for the CIH $\mathcal{H}$.
  - Let $K \leftarrow \mathsf{SEHash.Gen}(1^{\lambda_0}, 1^T, 1^1)$, $\mathsf{crs}' \leftarrow \mathsf{BARG}_{L-1}.\mathsf{Gen}(1^\lambda, 1^{T'}, 1^{|\mathsf{NewRel}|})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^{\lambda_0})$, where $T' = T/2$.
  - Let $\mathsf{crs} = (K, \mathsf{crs}', \mathcal{H}.k)$ and output crs.
- $\mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$: The trapdoor CRS generation algorithm generates the trapdoor CRS as follows.
  - Generate $(K^*, \mathsf{td}) \leftarrow \mathsf{SEHash.TGen}(1^{\lambda_0}, 1^T, \{i^*\})$,
  - Let $\mathsf{crs}^{*\prime} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T'}, 1^{|\mathsf{NewRel}|}, \lfloor (i^*+1)/2 \rfloor)$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^{\lambda_0})$.
  - Let $\mathsf{crs}^* = (K^*, \mathsf{crs}^{*\prime}, \mathcal{H}.k)$, and output crs^*.

- $\underline{\mathsf{P}(\mathsf{crs}, C, \omega_1, \omega_2, \ldots, \omega_T)}$: The prover algorithm first commits to all PCP strings in a "columnwise" manner, and then applies the CIH to the commitment.
  - For each $i \in [T]$, compute the PCP proof $\pi_i \leftarrow \mathsf{PCP.P}(1^{\lambda_0}, C, i, \omega_i)$ for i-th instance (C, i).
  - Committing the $\pi = \{\pi_i\}_{i\in[T]}$ "columnwise",
    $$\forall q \in [\ell], c_q \leftarrow \mathsf{SEHash.Hash}(K, \{\pi_i|_q\}_{i\in[T]}; r_q),$$
    with uniformly random r_q.
  - Applying the CIH to $c = \{c_q\}_{q\in[\ell]}$, $r \leftarrow \mathcal{H}.\mathsf{Hash}(\mathcal{H}.k, (C, c))$, and let $(Q, \mathsf{st}) \leftarrow \mathsf{PCP.Q}(1^{\lambda_0}, C, r)$.
  - For each $i \in [T]$, let $\vec{\rho}_i$ be the opening of $\pi_i|_Q$. Specifically,
    $$\vec{\rho}_i = \{\mathsf{SEHash.Open}(K, \{\pi_i|_q\}_{i\in[T]}, i, r_q)\}_{q\in Q}.$$
  - Compute a smaller BARG proof, let
    $$\Pi' \leftarrow \mathsf{BARG}'.\mathsf{P}(\mathsf{crs}', \mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st},\mathsf{st}_{\mathsf{SEHash}}]}, \{\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q, \vec{\rho}_{2i}, \pi_{2i}|_Q\}_{i\in[T']}),$$
    where NewRel is depicted in Figure 5.
  - Output the proof $\Pi = (c, \Pi')$.
- $\underline{\mathsf{V}(\mathsf{crs}, C, \Pi)}$: The verification algorithm parses the proof Π as the commitment and the proof for the smaller BARG, then it utilizes the fast online verification property of the PCP to delegate the online verification to the smaller BARG.
  - Parse $\Pi = (c, \Pi')$. Applying CIH to c, let $r \leftarrow \mathcal{H}.\mathsf{Hash}(\mathcal{H}.k, (C, c))$.
  - $\mathsf{st}_{\mathsf{SEHash}} := \mathsf{PreVer}(K)$.
  - Generate the PCP query, $(Q, \mathsf{st}) \leftarrow \mathsf{PCP.Q}(1^{\lambda_0}, C, r)$.
  - Verify the smaller BARG, output $\mathsf{BARG}_{L-1}.\mathsf{V}(\mathsf{crs}', \mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st},\mathsf{st}_{\mathsf{SEHash}}]}, \Pi')$.

D Protocol: SNARGs for P

The following text is taken largely verbatim from [CJJ21b], where the definitions are from [KPY19].

D.1 Turing Machine Delegation

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

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

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

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

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

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

Definition 17. *A publicly verifiable non-interactive delegation scheme* (Del.S, Del.P, Del.V) *for* $\mathcal{M}$ *with setup time* $T_{\mathsf{S}} = T_{\mathsf{S}}(\lambda, T)$ *and proof length* $L_\Pi = L_\Pi(\lambda, T)$.

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

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

Efficiency. *In the completeness experiment above,*

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

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

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

D.2 RAM Delegation

A RAM machine of word size ℓ is modeled as a deterministic machine with random access to memory of size 2^ℓ where the local state of the machine has size only $O(\ell)$. At each time step, the machine updates its local state by either reading or writing a single memory. At any given time, the memory and the local state together represent the configuration cf of the machine. For simplicity, we assume that the machine has no input outside of its local state and memory, and the word size ℓ will correspond to the security parameter λ.

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

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

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

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

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

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

$$\mathcal{U}_\mathcal{R} \coloneqq \left\{ (\ell, \mathsf{cf}, \mathsf{cf}', T) \;\middle|\; \mathcal{R} \text{ with word size } \ell \text{ transitions from cf to cf}' \text{ in } T \text{ steps} \right\}$$

Definition 18. *A publicly verifiable non-interactive delegation scheme* (RDel.S, RDel.D, RDel.P, RDel.V) *for* $\mathcal{R}$ *with setup time* $T_{\mathsf{S}} = T_{\mathsf{S}}(\lambda, T)$ *and proof length* $L_\Pi = L_\Pi(\lambda, T)$.

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

$$\Pr\left[\mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^\lambda,) \\ \mathsf{h} \coloneqq \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}) \\ \mathsf{h}' \coloneqq \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}') \\ \Pi \coloneqq \mathsf{Del.P}(\mathsf{pk}, \mathsf{cf}, \mathsf{cf}') \end{array} \right] = 1$$

Efficiency. *In the completeness experiment above,*

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

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

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

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

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

As discussed in [KPY19], the notion of RAM delegation considered in their work is different from those in prior works [KP16, BHK17] - namely that in prior works the adversary was not required to output the full configuration explicitly, only that it was difficult to produce accepting proofs for two different statements $(\mathsf{h}, \mathsf{h}')$ and $(\mathsf{h}, \mathsf{h}'')$ that share the same initial digest. We refer the reader to [KPY19] for a more detailed comparison of the notions.

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

Theorem 13 ([KPY19])**.** *Suppose that for any RAM machine there exists a publicly verifiable non-interactive delegation scheme with setup time* T'_{S} *and proof length* L'_{Π}. *Then for any Turing machine there exists a publicly verifiable non-interactive delegation scheme with setup time* T_{S} *and proof length* L_{Π} *where* $T_{\mathsf{S}}(\lambda, T) = T'_{\mathsf{S}}(\lambda, T')$, $L_{\Pi}(\lambda, T) = L'_{\Pi}(\lambda, T')$ *for* $T' = O(T)$.

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

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

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

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

that $\varphi(\mathsf{h}, \mathsf{h}', b, \Pi, \Pi', w) = 1$ *if and only if*

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

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

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

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

$$\phi\left(\mathsf{h}_0, \left\{\mathsf{h}_i, b_i, \Pi_i, \Pi_i', w_i\right\}_{i\in[T]}\right) := \bigwedge_{i\in[T]} \varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi_i', w_i)$$

Note that the above formula is *not an index language*. This is because for all i, φ_i and φ_{i+1} share a part of the witness, something not handled by the index language since we would have to ensure that the (partial) witness is the *same*. As described in the technical overview, we handle this by using a NS-SEHash to commit to the witnesses, and then prove for each i that values in the commitment satisfy the clause ϕ_i. The no-signaling property will help ensure consistency of the shared witness across different clauses φ_i and φ_{i+1}. The changes from [CJJ21b] are highlighted.

- An L-no-signaling-SEHash hash scheme NS-SEHash = (NS-SEHash.Gen, NS-SEHash.TGen, NS-SEHash.Hash, NS-SEHash.Open, NS-SEHash.Verify, NS-SEHash.Ext, NS-SEHash.PreVer,NS-SEHash.OnlineVer) with security parameter λ_0.
- A non interactive batch argument for an index language (BARG.Gen, BARG.TGen, BARG.P, BARG.V) with security parameter λ for circuits of size $\mathsf{poly}(\lambda_0)$.
- A hash tree scheme (HT.Gen, HT.Hash, HT.Read, HT.Write, HT.VerRead, HT.VerWrite) with security parameter λ_0.

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

$$K \leftarrow \mathsf{Gen}(1^{\lambda_0}, 1^M, 1^L),\ \mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}),\ \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^{\lambda_0}).$$

Ouput $(\mathsf{pk} := (K, \mathsf{crs}, \mathsf{dk}), \mathsf{vk} := (K, \mathsf{crs}), \mathsf{dk})$.

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

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

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

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

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

Then for every $i \in [T]$

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

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

Compute the no-signaling commitment to $(\mathsf{h}_0, \{\mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i\in[T]})$

$$c \coloneqq \mathsf{NS\text{-}SEHash.Hash}\left(K, \left(\mathsf{h}_0, \left\{\mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\right\}_{i\in[T]}\right)\right)$$

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

For $A \in \left\{\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\right\}$,

$$\rho_A \coloneqq \mathsf{NS\text{-}SEHash.Open}(K, A, R)$$

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

$$\Pi \coloneqq \mathsf{BARG.P}\left(\mathsf{crs}, C_{\mathsf{index}}, \left\{\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i, \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \rho_{b_i}, \rho_{\Pi_i}, \rho_{\Pi'_i}, \rho_{w_i}\right\}_{i\in[T]}\right)$$

Output (c, Π).

$\underline{\mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi)}$**:** Given c and K, compute

$$\mathsf{st}_{\mathsf{NS\text{-}SEHash}} \coloneqq \mathsf{NS\text{-}SEHash.PreVer}(K),$$

compute C_{index} (described below) and output 1 if and only if

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

Circuit C_{index}

Hardwired: $K, c, \varphi, , \mathsf{st}_{\text{NS-SEHash}}$
Input: $i, \mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i, \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \rho_{b_i}, \rho_{\Pi_i}, \rho_{\Pi'_i}, \rho_{w_i}$
Output: Output 1 if and only if

1. Verify commitment openings:
   (a) $\text{NS-SEHash.OnlineVer}(c, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}, \mathsf{st}_{\text{NS-SEHash}}) = 1$
   (b) $\text{NS-SEHash.OnlineVer}(c, \mathsf{h}_i, \rho_{\mathsf{h}_i}, \mathsf{st}_{\text{NS-SEHash}}) = 1$
   (c) $\text{NS-SEHash.OnlineVer}(c, b_i, \rho_{b_i}, \mathsf{st}_{\text{NS-SEHash}}) = 1$
   (d) $\text{NS-SEHash.OnlineVer}(c, \Pi_i, \rho_{\Pi_i}, \mathsf{st}_{\text{NS-SEHash}}) = 1$
   (e) $\text{NS-SEHash.OnlineVer}(c, \Pi'_i, \rho_{\Pi'_i}, \mathsf{st}_{\text{NS-SEHash}}) = 1$
   (f) $\text{NS-SEHash.OnlineVer}(c, w_i, \rho_{w_i}, \mathsf{st}_{\text{NS-SEHash}}) = 1$
2. $\varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$

Figure 7: Circuit C_{index}.